**ML Classification Pipeline –**

**Functional Specification**

**1. Overview**

**1.1 Purpose**

This system implements a machine learning classification pipeline to forecast business experience, education level, and training experience based on textual input data. The pipeline processes data through multiple stages from raw input to final predictions using a modular approach with sophisticated text preprocessing and Random Forest classification.

**1.2 System Architecture**

The pipeline consists of 5 main components executed sequentially:

* **Pipeline Controller** (0 - Pipeline.py) - Orchestrates the entire workflow
* **Data Normalization** (1 - Normalization.py / 1-1 - Normalization Predict.py) - Cleans and standardizes input text
* **Data Vectorization** (2 - Vectorization.py) - Converts normalized text into semantic embeddings
* **Model Building** (3 - Build Model.py) - Trains Random Forest classifier with hyperparameter tuning
* **Prediction** (4 - Prediction.py) - Generates predictions using trained model and optimized thresholds

**1.3 Model Type**

Multi-output classification targeting three binary categories:

* **func\_bizstr**: Business experience classification
* **edu\_stem**: Education level classification
* **entre\_training\_experience**: Training experience classification

**1.4 Key Technologies**

* **NLP**: spaCy for text preprocessing and normalization
* **Embeddings**: SentenceTransformer (all-MiniLM-L6-v2) for semantic vectorization
* **ML Model**: Random Forest with multi-output classification
* **Optimization**: 5-fold cross-validation with grid search and threshold tuning

**2. Pipeline Controller (0 - Pipeline.py)**

**2.1 Functional Description**

The main orchestrator that executes all pipeline stages in the correct sequence. It manages script execution, error handling, and provides visual feedback on pipeline progress.

**2.2 Key Features**

* **Sequential Script Execution**: Runs each pipeline stage in predetermined order
* **Error Handling**: Stops pipeline execution if any stage fails
* **Progress Tracking**: Provides visual indicators (🚀, ✅, ❌) for each stage
* **File Path Management**: Handles input/output file routing between stages
* **Command Line Safety**: Uses proper shell escaping for security

**2.3 Data Flow**

Training Path:

RA\_analysis.xlsx → [Normalization] → RA\_analysis - Normalized.xlsx →

[Vectorization] → RA\_analysis - Normalized - Vectorized.csv → [Model Building]

Prediction Path:

feedback-merge-categorization.xlsx → [Normalization Predict] →

feedback-merge-categorization - Normalized.xlsx → [Vectorization] →

feedback-merge-categorization - Normalized - Vectorized.csv → [Prediction]

**3. Data Normalization (1 - Normalization.py & 1-1 - Normalization Predict.py)**

**3.1 Purpose**

Preprocesses and normalizes textual data to create consistent, clean input for vectorization. Two versions handle training data (with labels) and prediction data (without labels).

**3.2 Text Processing Pipeline**

1. **Keyword Removal**: Eliminates specified keywords ("chatgpt", "gpt", "peer", "gemini")
2. **Bullet Point Cleaning**: Removes numbered/bulleted list formatting
3. **Data Grouping**: Aggregates text by analysis\_ID
4. **NLP Normalization**: Lemmatization, stopword removal, punctuation cleaning
5. **Output Generation**: Creates normalized Excel files with required columns

**3.3 Training Data Processing (1 - Normalization.py)**

**Input Requirements**:

* Excel file with "original" sheet
* Required columns: analysis\_ID, original, func\_bizstr, edu\_stem, entre\_training\_experience

**Processing Steps**:

* Groups multiple text entries per analysis\_ID
* Concatenates original text while preserving raw version
* Aggregates labels using max() function
* Applies text preprocessing and normalization

**Output Columns**:

* analysis\_ID: Unique identifier
* original: Concatenated raw text
* normalized: Processed and normalized text
* func\_bizstr, edu\_stem, entre\_training\_experience: Target labels

**3.4 Prediction Data Processing (1-1 - Normalization Predict.py)**

**Input Requirements**:

* Excel file with "prediction" sheet
* Required columns: analysis\_ID, original

**Processing Steps**:

* Similar text processing to training version
* Groups by analysis\_ID and concatenates text
* Preserves metadata columns: comparison, group, Categorization-final

**Output Columns**:

* analysis\_ID: Unique identifier
* comparison, group, Categorization-final: Metadata columns
* original: Concatenated raw text
* normalized: Processed and normalized text

**3.5 Text Preprocessing Functions**

* **remove\_keywords()**: Case-insensitive removal of specified terms
* **remove\_bullet\_symbols()**: Regex-based removal of numbered/bulleted formatting
* **preprocess\_text()**: Combined preprocessing pipeline
* **normalize\_text()**: spaCy-based normalization (lemmatization, stopword removal)

**4. Data Vectorization (2 - Vectorization.py)**

**4.1 Purpose**

Converts normalized text into high-dimensional semantic embeddings using pre-trained transformer models for machine learning input.

**4.2 Technical Specifications**

* **Model**: SentenceTransformer "all-MiniLM-L6-v2" (384 dimensions)
* **Input**: Excel files with normalized column
* **Output**: CSV files with original columns + embedding features
* **Batch Processing**: Configurable batch size (default: 64)

**4.3 Processing Pipeline**

1. **Input Validation**: Ensures XLSX format and normalized column presence
2. **Text Extraction**: Extracts and cleans normalized text
3. **Model Loading**: Loads specified SentenceTransformer model
4. **Batch Encoding**: Processes texts in batches with progress tracking
5. **Feature Generation**: Creates emb\_0 through emb\_383 columns
6. **Output Generation**: Combines original data with embeddings in CSV format

**4.4 Output Features**

* **Original Columns**: All columns from input file preserved
* **Embedding Columns**: 384 numerical features (emb\_0 to emb\_383)
* **File Format**: CSV for efficient ML processing

**5. Model Building (3 - Build Model.py)**

**5.1 Purpose**

Trains a Random Forest multi-output classifier with comprehensive hyperparameter optimization and threshold tuning for optimal performance across all target variables.

**5.2 Model Architecture**

* **Base Model**: Random Forest with balanced class weights
* **Multi-Output**: Separate classifier for each target variable
* **Optimization**: Grid search + 5-fold cross-validation + threshold tuning

**5.3 Hyperparameter Search Space**

MAX\_DEPTH: [3, 6, 9, 12, 15]

MIN\_SAMPLES\_LEAF: [1, 3, 5, 7]

N\_ESTIMATORS: [200]

MAX\_FEATURES: ["sqrt", "log2"]

MIN\_SAMPLES\_SPLIT: [2, 4]

**5.4 Optimization Process**

1. **Grid Search Phase**:
   * 5-fold cross-validation with macro-F1 scoring
   * Fixed threshold at 0.5 for initial comparison
   * Selects best hyperparameter combination
2. **Threshold Tuning Phase**:
   * Tests 19 threshold values (0.05 to 0.95)
   * Optimizes per-target thresholds independently
   * Maximizes macro-F1 score for each classification task
3. **Final Model Training**:
   * Retrains on full dataset with optimal parameters
   * Saves model, thresholds, and metadata with timestamps

**5.5 Output Artifacts**

* **Model File**: rf\_multioutput\_[timestamp].joblib - Trained classifier
* **Thresholds File**: rf\_thresholds\_[timestamp].json - Optimal decision thresholds
* **Metadata File**: rf\_metadata\_[timestamp].json - Training parameters and performance metrics

**5.6 Performance Metrics**

* **Primary Metric**: Macro-F1 score (handles class imbalance)
* **Per-Target Metrics**: Individual F1 scores for each classification task
* **Cross-Validation**: 5-fold CV with stratified sampling

**6. Prediction (4 - Prediction.py)**

**6.1 Purpose**

Applies trained model to new data using optimized thresholds to generate classification predictions with probability scores.

**6.2 Model Loading Strategy**

* **Automatic Detection**: Finds latest model files by timestamp
* **Threshold Matching**: Automatically pairs model with corresponding thresholds
* **Fallback Logic**: Uses latest available files if exact matches not found

**6.3 Prediction Process**

1. **Artifact Loading**: Loads trained model and threshold configuration
2. **Data Validation**: Ensures presence of embedding columns
3. **Probability Generation**: Calculates class probabilities for each target
4. **Threshold Application**: Applies optimized thresholds per target
5. **Output Generation**: Creates predictions with probabilities and binary classifications

**6.4 Output Format**

For each target variable (func\_bizstr, edu\_stem, entre\_training\_experience):

* **[target]\_p1**: Probability of positive class (0.0-1.0)
* **[target]\_pred**: Binary prediction (0 or 1) based on optimized threshold

**6.5 Output Columns**

* analysis\_ID: Unique identifier (if present in input)
* func\_bizstr\_p1, func\_bizstr\_pred: Business experience probability and prediction
* edu\_stem\_p1, edu\_stem\_pred: Education level probability and prediction
* entre\_training\_experience\_p1, entre\_training\_experience\_pred: Training experience probability and prediction

**7. Data Flow and File Specifications**

**7.1 Input File Requirements**

**Training Data** (RA\_analysis.xlsx):

* Sheet: "original"
* Required columns: analysis\_ID, original, func\_bizstr, edu\_stem, entre\_training\_experience
* Data types: Mixed (ID: string/int, original: text, targets: binary 0/1)

**Prediction Data** (feedback-merge-categorization.xlsx):

* Sheet: "prediction"
* Required columns: analysis\_ID, original
* Optional columns: comparison, group, Categorization-final

**7.2 Intermediate Files**

* **Normalized Files**: [filename] - Normalized.xlsx
* **Vectorized Files**: [filename] - Normalized - Vectorized.csv
* **Prediction Output**: [filename] - Predictions.csv

**7.3 Model Artifacts Directory Structure**

models/

├── rf\_multioutput\_[timestamp].joblib

├── rf\_thresholds\_[timestamp].json

└── rf\_metadata\_[timestamp].json

**8. Technical Requirements**

**8.1 Dependencies**

* **Core**: pandas, numpy, scikit-learn, joblib
* **NLP**: spacy (en\_core\_web\_sm model), sentence-transformers
* **File Processing**: openpyxl
* **System**: subprocess, pathlib, argparse

**8.2 Hardware Considerations**

* **Memory**: Scales with dataset size and embedding dimensions (384 \* samples)
* **Processing**: CPU-intensive for Random Forest training and text processing
* **Storage**: Models and embeddings require significant disk space

**8.3 Performance Characteristics**

* **Training Time**: Varies with grid search parameters and data size
* **Prediction Time**: Fast inference once model is loaded
* **Scalability**: Linear scaling with input text volume

**9. Error Handling and Validation**

**9.1 Input Validation**

* File existence and format validation
* Required column presence checking
* Data type validation and conversion
* Empty/null value handling

**9.2 Processing Error Handling**

* Graceful degradation for text processing failures
* Model loading and compatibility validation
* Memory management for large datasets
* Progress tracking and error reporting

**9.3 Output Validation**

* File format compliance
* Data integrity checks
* Performance metric validation
* Artifact consistency verification

**10. Usage Examples**

**10.1 Complete Pipeline Execution**

python "0 - Pipeline.py"

**10.2 Individual Component Execution**

# Normalization

python "1 - Normalization.py" --input RA\_analysis.xlsx

# Vectorization

python "2 - Vectorization.py" --input "RA\_analysis - Normalized.xlsx"

# Model building

python "3 - Build Model.py" --input "RA\_analysis - Normalized - Vectorized.csv"

# Prediction

python "4 - Prediction.py" --input "feedback-merge-categorization - Normalized - Vectorized.csv"

**11. Configuration and Customization**

**11.1 Configurable Parameters**

* **Keyword Library**: Customizable terms for removal
* **Model Selection**: Different SentenceTransformer models
* **Hyperparameter Ranges**: Grid search boundaries
* **Cross-Validation**: Number of folds and random seeds
* **Threshold Grid**: Range and granularity for optimization

**11.2 Future Enhancements**

* **Configuration Files**: External YAML/JSON configuration
* **Logging System**: Comprehensive execution logging
* **Model Versioning**: Advanced model lifecycle management
* **Performance Monitoring**: Real-time metrics tracking
* **Distributed Processing**: Multi-GPU/cluster support