Multimodal Data Preprocessing System - Team Report

Project: Multimodal Data Preprocessing
Course: Machine Learning Pipeline
Team: Group 11
GitHub Repository: https://github.com/excelasaph/Data-Preprocessing-Group-11
Demo Video: https://youtube.com/watch?v=YOUR_VIDEO_ID

Executive Summary

This report shows the implementation of a multimodal biometric security system that integrates facial recognition, voiceprint verification, and personalized product recommendations. The project fulfills all requirements across four main tasks, with each team member contributing significantly to different aspects of the system.

Project Overview

- **Data Collection**: 4 team members × 3 expressions + 2 audio phrases with augmentation
- **Model Development**: 3 machine learning models (facial recognition, voice verification, product recommendation)
- **System Integration**: Demo with real-time authentication and unauthorized access simulation
- **Feature Engineering**: Pipeline for image and audio feature extraction

Model Metrics

Product Recommendation Model:

- **Accuracy**: 65%
- **F1-Score** - 61%
- **Loss** - 0.8349

Facial Recognition Model:

- **Accuracy**: 90%
- **F1-Score** - 90%
- **Loss** - 0.6440

Voiceprint Verification Model:

- **Accuracy**: 85.71%
- **F1-Score** - 67%
- **Loss** - 0.5188

Overall Evaluation Metrics

- **Accuracy**: Face (90%), Voice (85.71%), Product (65%)

Team Member Contributions

1. Anne Marie Twagirayezu - Image Processing & Facial Recognition Model

My Approach to Image Data Collection:

I started by gathering photos from all four team members, making sure to capture different facial expressions (neutral, smiling, surprised) to create a diverse dataset. This was crucial because facial recognition models need to handle variations in expressions. I organized the data systematically, storing each person's images in separate folders by expression type.

Building the Image Augmentation Pipeline:

Thinking of how to go about the augmentation, I designed a strategy to increase our dataset size while maintaining realistic variations. Instead of just applying random transformations, I carefully chose specific augmentations that would help the model generalize better:

```
# Image augmentation pipeline
def augment_image(image, size=(64, 64)):
    image = clean_image(image, size)
    flipped = clean_image(cv2.flip(image, 1), size)
    rotated = clean_image(cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE), size)
    gray = clean_image(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY), size)
    return [image, flipped, rotated, gray]
```

Feature Extraction Development:

- Implemented color histogram feature extraction
- Created 512-dimensional feature vectors (8×8×8 RGB bins)
- Developed automated feature extraction pipeline
- Generated `image_features.csv` with 48 samples (12 originals × 4 augmentations)

Developing the Feature Extraction System:

While experimenting with different feature extraction methods, I found that color histograms worked best for our use case. I chose an **8×8×8 RGB histogram** because it captures color distribution patterns that are unique to each person while being computationally efficient. This gave us **512-dimensional** feature vectors that provided enough detail without overwhelming the model.

Training the Facial Recognition Model:

I tested multiple algorithms (Random Forest, Logistic Regression, and XGBoost) and found that **XGBoost** performed best with **90% accuracy**. The key was tuning the hyperparameters and ensuring the model could handle the variations in our augmented dataset. I also implemented confidence thresholds to make the authentication system more clean.

Files Created/Modified:

- `Data/pictures/` - Original image collection
- `augmented/` - Augmented image storage
- `Datasets/image_features.csv` - Extracted image features
- `models/facial_recognition_xgboost_model.joblib` - Trained model
- `Notebooks/Image_processing&_Facial_recognition_model.ipynb` - Processing notebook

---

2. Excel Asaph - Data Merging & Product Recommendation Model

Process for Merging Datasets:

To merge both customer datasets, I began by carefully analyzing both to understand their structure and identify potential challenges. The social profiles had customer IDs like `"A100"` while transactions used numeric IDs like `"100"`, so I had to create a mapping strategy. I also noticed duplicate entries and missing values that needed careful handling.

Building the Data Preprocessing Pipeline:

Here is my approach to cleaning and merging the data - First, I handled duplicates by aggregating social profiles by `customer ID`, taking the *mean* for numerical values and the *mode* for categorical ones. For transactions, I aggregated by customer ID to get total purchase amounts and transaction counts:

```
# Data merging and feature engineering
def merge_customer_data(social_profiles, transactions):
    # Handle duplicates and null values
    social_clean = social_profiles.groupby('customer_id_new').agg({
        'engagement_score': 'mean',
        'purchase_interest_score': 'mean',
        'review_sentiment': lambda x: x.mode()[0] if not x.mode().empty else
x.iloc[0],
        'social_media_platform': lambda x: x.mode()[0] if not x.mode().empty else
x.iloc[0]
    }).reset_index()

    # Clean transactions data
    transactions_clean = transactions.groupby('customer_id_legacy').agg({
        'transaction_id': 'count',
        'purchase_amount': 'sum',
        'purchase_date': 'first',
        'product_category': lambda x: x.mode()[0] if not x.mode().empty else
x.iloc[0],
        'customer_rating': 'mean'
    }).reset_index()

    # Merge datasets
    merged_data = pd.merge(social_clean, transactions_clean,
                          left_on='customer_id',
                          right_on='customer_id_legacy',
                          how='left')

    return merged_data
```

Feature Engineering:

- Created 15 engineered features for product recommendation
- Implemented categorical encoding (one-hot encoding)
- Developed temporal feature extraction (`purchase_date` encoding)
- Created engagement and purchase pattern metrics

Training the Product Recommendation Model:

For my model training, I chose **Random Forest** because it handles both numerical and categorical features well, and it's robust to overfitting. I used **GridSearchCV** to systematically test different hyperparameter combinations, which helped me find the optimal settings: `max_depth=None`, `min_samples_leaf=2`, `min_samples_split=2`, `n_estimators=300`. The **65% accuracy**, while not perfect, is reasonable given the complexity of predicting product preferences from limited customer data.

Files Created/Modified:

- `Data/datasets/` - Original customer datasets
- `Datasets/merged_customer_data.csv` - Merged dataset
- `models/product_recommendation_model.pkl` - Trained model
- `encoders/product_recommendation_scaler.pkl` - Feature scaler
- `Notebooks/Data_Merging_Product_Recommendation_Model_Training.ipynb` - Processing notebook

---

3. Christophe Gakwaya - Audio Processing & Voiceprint Verification Model

How I Approached the Audio Data Collection:

First, I recorded voice samples from all team members saying two different phrases: **"Yes, approve"** and **"Confirm transaction."** Then, I chose these phrases because they represent realistic authentication scenarios. I made sure to record in a quiet environment and asked everyone to speak naturally, as this would make the voice recognition more reliable in real-world conditions.

Designing the Audio Augmentation Pipeline:

I designed the augmentation strategy to simulate real-world variations in voice recordings. I chose specific parameters that would help the model handle different speaking conditions:

```
# Audio augmentation pipeline
def augment_audio(y, sr):
    # Pitch shift (±2 semitones)
    y_pitch_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=2)

    # Time stretch (1.2x speed)
    y_time_stretched = librosa.effects.time_stretch(y, rate=1.2)

    # Noise addition (0.5% amplitude)
    noise = np.random.randn(len(y))
    noise_amplitude = 0.005 * np.max(np.abs(y))
    y_noisy = y + noise_amplitude * noise

    return [y, y_pitch_shifted, y_time_stretched, y_noisy]
```

Audio Processing Details:

- **Sample Rate**: 22050 Hz
- **Original Duration**: 2.76s - 5.74s (Christophe: 2.76s, Anne: 5.74s)
- **Trimmed Duration**: 1.49s - 3.04s (after silence removal)
- **Processing**: Top_db=30 for silence trimming

Processing and Feature Extraction:

I standardized all audio to **22050 Hz** sample rate and trimmed silence using `top_db=30`, which removed unnecessary silence while preserving the important speech content. For feature extraction, I chose **MFCCs** because they capture the spectral characteristics of speech that are unique to each person. I also included spectral **roll-off** and **RMS energy** features to capture additional voice characteristics.

Training the Voiceprint Verification Model:

For my model selection, I tested several algorithms and found that **Random Forest** worked best for our voice verification task, achieving **85.71% accuracy**. The model learned to distinguish between the four team members based on their unique voice characteristics captured in the MFCC and spectral features.

Files Created/Modified:

- `Data/audios/` - Original audio collection
- `Datasets/audio_features.csv` - Extracted audio features
- `models/voiceprint_verification_model.joblib` - Trained model
- `encoders/voice_feature_scaler.joblib` - Feature scaler
- `Notebooks/Audio_Processing_Features.ipynb` - Processing notebook

---

4. Kanisa Rebecca Majok Thiak - System Demo Implementation and Simulation

Building the Demo System:

The demo system has to showcase the complete multimodal authentication workflow in a user-friendly way. I wanted to create an interactive experience that would demonstrate both the technical capabilities and the security features of our system. The key challenge was integrating three different models (face, voice, and product recommendation) into a one workflow.

Creating the BiometricSecuritySystem Class:

I created a class in the `system_demo.py` file that loads all the trained models and handles the complete authentication pipeline.

The system follows a logical sequence: first *face authentication*, then *voice verification*, and finally *product recommendation*. I included proper error handling to ensure the system fails if any step fails:

```
class BiometricSecuritySystem:
    def __init__(self):
        # Load all models and scalers
        self.load_models()
        self.load_customer_data()

    def run_full_transaction(self, user_name, image_path, audio_path):
        # Step 1: Face Authentication
        face_auth_success, predicted_user = self.authenticate_face(image_path)
        if not face_auth_success:
            return False

        # Step 2: Voice Verification
        voice_auth_success, final_user = self.verify_voice(audio_path,
predicted_user)
        if not voice_auth_success:
            return False

        # Step 3: Product Recommendation
        recommended_category, user_profile =
self.get_product_recommendations(final_user)

        # Step 4: Transaction Approval
        print("All authentication steps passed successfully!")
        print("User authorized to proceed with transaction")
        print(f"Recommended products: {recommended_category}")
        print("TRANSACTION COMPLETED SUCCESSFULLY!")

        return True
```

Interactive Demo Features:

- **Authorized User Simulation**: Complete transaction flow for all team members
- **Unauthorized Access Simulation**: Security denial demonstration
- **Custom Transaction**: User-defined image and audio paths
- **Real-time Authentication**: Live face and voice verification

Demo Menu System:

```
def main_menu(self):
    print("1.  Simulate Authorized Transaction (Anne)")
    print("2.  Simulate Authorized Transaction (Christophe)")
    print("3.  Simulate Authorized Transaction (Excel)")
    print("4.  Simulate Authorized Transaction (Kanisa)")
    print("5.  Simulate Unauthorized Attempt")
    print("6.  Custom Transaction (Specify paths)")
    print("7.  Exit")
```

Files Created/Modified:

- `system_demo.py` - Main demo application
- `setup_demo.py` - Environment setup script
- `requirements.txt` - Python dependencies
- `README.md` - Project documentation

GitHub Repository Information

Repository: https://github.com/excelasaph/Data-Preprocessing-Group-11
Main Branch: main
Contributors: 4 team members
Total Commits: 40+ commits
Languages: Python, Jupyter Notebook, Markdown
Technologies: OpenCV, Librosa, Scikit-learn, XGBoost, Joblib