Ensemble Pipeline V2

A comprehensive machine learning pipeline for ensemble modeling with ONNX export capabilities and optimization features.

Overview

This pipeline provides a robust framework for training, evaluating, and deploying ensemble machine learning models. It includes features for:

- Model training and evaluation with cost-sensitive optimization
- Hyperparameter tuning using Bayesian optimization
- ONNX model export for deployment
- Interactive configuration through Streamlit UI
- Comprehensive model evaluation and visualization

Installation

1. Clone the repository:

```
git clone <repository-url>
cd ensemble_pipelineV2
```

2. Set up Python environment:

   - **Python Version**: Python 3.12 is recommended for best compatibility
   - **Windows Users**: If using Python < 3.12, you must enable long path support:

   ```
   # Run PowerShell as Administrator
   Set-ItemProperty -Path
   "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem" -Name
   "LongPathsEnabled" -Value 1
   ```

   - Create and activate virtual environment:

   ```
   # Windows
   python -m venv venv
   .\venv\Scripts\activate

   # Linux/Mac
   python -m venv venv
   source venv/bin/activate
   ```

3. Install dependencies:

```
pip install -r requirements.txt
```

4. Optional ONNX dependencies (for model export):
   - For most systems (Python 3.8-3.12):

     ```
     pip install onnx skl2onnx
     ```

   - For Python 3.13 or if you encounter issues:

     ```
     # Try installing specific versions
     pip install onnx==1.17.0 skl2onnx

     # Or use conda if available
     conda install -c conda-forge onnx skl2onnx
     ```

   - If installation fails:
     - The application will automatically fall back to pickle export
     - You'll see a warning message: "ONNX dependencies not available. ONNX export will be disabled."
     - All functionality will work normally, just without ONNX export capability

Project Structure

```
ensemble_pipelineV2/
├── app/                    # Streamlit application
│   ├── sidebar.py         # Configuration sidebar
│   └── tabs.py            # Interactive model configuration
├── helpers/               # Core functionality
│   ├── modeling.py        # Model training and optimization
│   ├── metrics.py         # Evaluation metrics
│   ├── model_export.py    # ONNX export functionality
│   └── plotting.py        # Visualization utilities
├── config.py              # Configuration settings
├── run.py                 # Main pipeline execution
└── requirements.txt       # Project dependencies
```

Configuration

The pipeline is highly configurable through `config.py` and the Streamlit UI. Key settings include:

Model Settings

- **Cost Weights**: Configure false positive (C_FP) and false negative (C_FN) costs

- **Cross-Validation**: Enable/disable k-fold cross-validation (N_SPLITS=5 by default)
- **Feature Filtering**: Optional variance and correlation-based feature selection
  - Variance Threshold: 0.01 (default)
  - Correlation Threshold: 0.95 (default)

Optimization Settings

- **Hyperparameter Optimization**: Bayesian optimization using Optuna
  - Number of iterations: 50 (default)
  - Supported models: XGBoost, RandomForest
- **Final Model Optimization**: Optional optimization of production models
- **SMOTE**: Class imbalance handling (ratio=0.5)

Model Export

- **ONNX Export**: Convert models to ONNX format
  - Opset Version: 12 (default)
  - Automatic fallback to pickle if ONNX unavailable
  - Supported Models:
    - LogisticRegression
    - RandomForestClassifier
    - MLPClassifier
    - KNeighborsClassifier
    - XGBoost models (with additional setup)
  - File Extensions:
    - ONNX enabled: Models saved as `.onnx` files
    - ONNX disabled: Models saved as `.pkl` files
    - Downloads tab shows appropriate file types based on configuration

Usage

Running the Pipeline

1. Start the Streamlit interface:

```
streamlit run app/main.py
```

2. Configure settings through the UI:

   - Adjust cost weights
   - Enable/disable k-fold cross-validation
   - Configure feature filtering
   - Set optimization parameters
   - Choose export format

3. Run the pipeline:

```
python run.py
```

Model Training and Evaluation

The pipeline supports two training modes:

1. **Single Split** (default):
    - 80/20 train-test split
    - Optional hyperparameter optimization
    - Cost-sensitive threshold optimization
2. **K-Fold Cross-Validation**:
    - 5-fold stratified cross-validation
    - Nested cross-validation for hyperparameter tuning
    - Averaged performance metrics

Model Export and Deployment

ONNX Export

The pipeline automatically handles model conversion to ONNX format:

1. **Export Process**:
    - Models are converted using `skl2onnx`
    - Feature names are preserved for input mapping
    - Opset version 12 is used by default
    - Automatic fallback to pickle if ONNX unavailable
2. **ONNX to Halcon**:
    - ONNX models can be imported into Halcon
    - Input/output tensor names are preserved
    - Model metadata includes feature names and thresholds

Export Settings

- Enable ONNX export in the UI or `config.py`
- Adjust opset version if needed (9-15 supported)
- Models are saved in `output/models/`

Optimization Details

Hyperparameter Optimization

The pipeline uses Optuna for Bayesian optimization:

1. **XGBoost Parameters**:

  - max_depth: [3,4,5,6]
  - learning_rate: [0.01,0.05,0.1]
  - subsample: [0.6,0.8,1.0]
  - colsample_bytree: [0.6,0.8,1.0]
  - n_estimators: [100,200,400]
  - gamma: [0,0.1,0.2]

2. **RandomForest Parameters**:

  - n_estimators: [100,200,300]
  - max_depth: [None,5,10]
  - min_samples_split: [2,5,10]
  - min_samples_leaf: [1,2,5]

Cost-Sensitive Optimization

- False Positive Cost (C_FP): 1
- False Negative Cost (C_FN): 30
- Thresholds optimized for both cost and accuracy
- Results visualized in threshold sweep plots

Model Evaluation

The pipeline provides comprehensive evaluation metrics:

1. **Performance Metrics**:

  - Precision, Recall, Accuracy
  - Cost-weighted performance
  - Confusion matrix
  - ROC and PR curves

2. **Visualization**:

  - Threshold sweep plots
  - Model comparison plots
  - Class balance visualization
  - Feature importance plots

Troubleshooting

Common Issues

1. **ONNX Export**:

  - Ensure correct Python version (3.8-3.12 recommended)
  - Windows Users:
    - Either use Python 3.12 (recommended)
    - Or enable long path support in Windows Registry
    - Or use conda instead of pip

- Check ONNX and skl2onnx installation
- Verify feature names are provided
- Windows Path Length Error:
  - Enable long path support in Windows Registry (see Installation section)
  - Or use Python 3.12
  - Or use conda instead of pip
- Python 3.13 Compatibility:
  - ONNX may not have pre-built wheels for the latest Python versions
  - Consider using Python 3.11 or 3.12 for better compatibility
- Import Errors:
  - The application will automatically detect missing dependencies
  - Check the console output for warning messages
  - Ensure you're using the virtual environment

2. **Model Training**:

   - Check class imbalance (SMOTE enabled by default)
   - Verify feature filtering thresholds
   - Monitor optimization progress

3. **Performance**:

   - Adjust cost weights if needed
   - Try different hyperparameter ranges
   - Consider feature selection

Contributing

1. Fork the repository
2. Create a feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

License

[Your License Here]