Contents

1 Instrumentation Guide

Detailed logging has been added to help debug the RNN training and evaluation pipeline.

1.1 Files Instrumented

1.1.1 1. skol_classifier/rnn_model.py - RNNSkolModel.predict()

Added logging at these key points:

Verbosity Level 1 (Major steps): - `[RNN Predict] Prediction completed successfully` - Final step

Verbosity Level 2 (Detailed steps): - `[RNN Predict] Starting prediction` - Entry point - `[RNN Predict] Input data count` - Dataset size - `[RNN Predict] Transforming data into sequences` - Sequence preprocessing - `[RNN Predict] Applying prediction UDF to sequences` - Actual prediction - `[RNN Predict] Exploding predictions and labels for evaluation` - Result formatting -

`[RNN Predict] Joining predictions with labels` - Alignment - `[RNN Predict] Joined result count` - Final result size

Verbosity Level 3 (Debug details): - Input/output data schemas - Column names at each stage - Sample data (via `.show()`) - Intermediate counts

1.1.2 2. skol_classifier/classifier_v2.py - SkolClassifierV2.fit()

Added logging after model training:

Verbosity Level 1 (Major steps): - `[Classifier Fit] Model training completed, starting evaluation` - `[Classifier Fit] Splitting data for evaluation (80/20)` - `[Classifier Fit] Making predictions on test set` - `[Classifier Fit] Predictions completed, validating output` - `[Classifier Fit] Calculating statistics` - `[Classifier Fit] Statistics calculated, adding metadata` - `[Classifier Fit] Evaluation complete, returning stats`

Verbosity Level 2 (Detailed steps): - Label mapping count - Train/test split counts - Prediction count - Final statistics dictionary

Verbosity Level 3 (Debug details): - Test data schema - Test data samples - Predictions schema - Predictions samples

1.2 Usage

The RNN model's verbosity level is passed through the `model_params`:

```python
from skol_classifier import SkolClassifierV2

classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    model_params={
        'input_size': 1000,
        'num_classes': 3,
        'verbosity': 3,  # Set to 1, 2, or 3
        # ... other params
    }
)

results = classifier.fit()
```

1.3 Verbosity Levels

1.3.1 Level 0 (Default)

- No instrumentation logging
- Only user-facing messages

1.3.2 Level 1 (Major Steps)

- High-level progress indicators
- Shows major phase transitions
- Recommended for normal training

1.3.3 Level 2 (Detailed)

- Detailed step-by-step progress
- Counts and data sizes
- Recommended for debugging

1.3.4 Level 3 (Debug)

- Everything from Level 2
- Data schemas
- Sample data (`.show(5)`)
- Use when troubleshooting specific issues

1.4 Debugging the Failure

Based on your report that training completes but fails before returning stats, run with **verbosity=2** or higher:

```
model_params = {
    'verbosity': 2,  # or 3 for maximum detail
    # ... other params
}
```

1.4.1 Expected Output Flow

You should see this sequence:

```
[RNN Fit] Training completed successfully
[Classifier Fit] Model training completed, starting evaluation
[Classifier Fit] Splitting data for evaluation (80/20)
[Classifier Fit] Making predictions on test set
[RNN Predict] Starting prediction
[RNN Predict] Input data count: <number>
```

```
[RNN Predict] Transforming data into sequences
[RNN Predict] Applying prediction UDF to sequences
[RNN Predict] Exploding predictions and labels for evaluation
[RNN Predict] Joining predictions with labels
[RNN Predict] Joined result count: <number>
[RNN Predict] Prediction completed successfully
[Classifier Fit] Predictions completed, validating output
[Classifier Fit] Calculating statistics
[RNN Stats] Calculating statistics for RNN predictions
[Classifier Fit] Statistics calculated, adding metadata
[Classifier Fit] Evaluation complete, returning stats
```

1.4.2 Finding the Failure Point

The **last message printed** before the failure/hang tells you where the issue is:

1. **Stops at "Applying prediction UDF to sequences"**
   - Issue in the Pandas UDF execution
   - Check Spark executor logs for Python/TensorFlow errors
   - May be running out of memory in executors
2. **Stops at "Exploding predictions"**
   - Issue with posexplode operation
   - May be OOM during explosion
   - Check predictions structure
3. **Stops at "Joining predictions with labels"**
   - Issue with join operation
   - May be data skew or OOM
   - Check join key cardinality
4. **Stops at "Calculating statistics"**
   - Issue with Spark ML evaluators
   - Check prediction/label column types
   - May be NaN or invalid values
5. **Stops at "Statistics calculated, adding metadata"**
   - Issue with train_data.count() or test_data.count()
   - May have unpersisted DataFrames

1.5 Additional Debugging

1.5.1 Check Spark Logs

If using verbosity doesn't reveal the issue, check Spark logs:

```
# Check Spark driver logs
tail -f $SPARK_HOME/logs/spark-*-driver*.out

# Check Spark executor logs (if running in distributed mode)
```

1.5.2 Enable Spark SQL Logging

Add before creating classifier:

```
spark.sparkContext.setLogLevel("INFO")
```

1.5.3 Memory Issues

If the failure is silent (no exception), it may be OOM. Check:

```
# Before fit()
print(f"Spark executor memory: {spark.conf.get('spark.executor.memory')}")
print(f"Spark driver memory: {spark.conf.get('spark.driver.memory')}")
```

1.6 Performance Note

Verbosity Level 3 includes `.count()` and `.show()` operations that trigger Spark actions. This can: - Significantly slow down execution - Increase memory usage - Trigger computation multiple times

Use Level 3 only when actively debugging a specific issue.

1.7 Example Debug Session

```
# Enable maximum verbosity
model_params = {
    'input_size': 1000,
    'hidden_size': 128,
    'num_classes': 3,
    'verbosity': 3,  # Maximum detail
    'epochs': 1,     # Reduce for faster debugging
    'batch_size': 16 # Smaller batch for memory
}

classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    model_params=model_params
)

# Run and watch output
try:
    results = classifier.fit()
    print(f"Success! Results: {results}")
except Exception as e:
    print(f"Failed with: {e}")
    import traceback
    traceback.print_exc()
```

Look for the last `[RNN Predict]` or `[Classifier Fit]` message to identify where it fails.