Contents

1 Evaluation Refactoring: Moving Stats to Model Methods

1.1 Summary

Refactored evaluation statistics calculation to be methods on the `SkolModel` class instead of standalone utility functions. This improves encapsulation and allows model-specific evaluation logic.

1.2 Changes

1.2.1 1. Updated `base_model.py`

Added two new methods to `SkolModel`:

1.2.1.1 `_create_evaluators()` - Protected Method

Creates model-specific evaluators for evaluation metrics:

```python
def _create_evaluators(self) -> Dict[str, MulticlassClassificationEvaluator]:
    """
    Create evaluation metrics for this model type.

    Returns:
        Dictionary containing evaluators for various metrics
    """
    # Default implementation for standard multiclass classification
    evaluators = {
        'accuracy': MulticlassClassificationEvaluator(
            labelCol=self.label_col,
            predictionCol="prediction",
            metricName="accuracy"
        ),
        # ... other metrics
    }
    return evaluators
```

Benefits: - Uses model's `label_col` instead of hardcoded `"label_indexed"` - Can be overridden by subclasses for custom evaluation logic - Encapsulates evaluator creation within the model

1.2.1.2 `calculate_stats()` - Public Method

Calculates evaluation statistics using model-specific evaluators:

```python
def calculate_stats(
    self,
    predictions: DataFrame,
    verbose: bool = True
) -> Dict[str, float]:
    """
    Calculate evaluation statistics for predictions.

    Args:
        predictions: DataFrame with predictions and labels
        verbose: Whether to print statistics

    Returns:
        Dictionary containing accuracy, precision, recall, f1_score
    """
    evaluators = self._create_evaluators()

    stats = {
        'accuracy': evaluators['accuracy'].evaluate(predictions),
        'precision': evaluators['precision'].evaluate(predictions),
        'recall': evaluators['recall'].evaluate(predictions),
        'f1_score': evaluators['f1'].evaluate(predictions)
    }

    if verbose:
        print(f"Test Accuracy: {stats['accuracy']:.4f}")
        # ... other metrics

    return stats
```

Benefits: - Instance method that uses model's internal state - Consistent interface across all model types - Can be customized by subclasses if needed

1.2.2 2. Updated `classifier_v2.py`

Changed from using standalone function to model method:

```
# Before:
from .utils import calculate_stats
stats = calculate_stats(test_predictions, verbose=False)

# After:
stats = self._model.calculate_stats(test_predictions, verbose=False)
```

1.2.3 3. Deprecated `utils.py` Functions

Marked `create_evaluators()` and `calculate_stats()` as deprecated:

```
def create_evaluators():
    """
    .. deprecated::
        Use ``model._create_evaluators()`` instead.
        This function is deprecated and will be removed in a future version.
    """
    warnings.warn(
        "create_evaluators() is deprecated and will be removed in a future versi
        "Use model._create_evaluators() instead.",
        DeprecationWarning,
        stacklevel=2
    )
    # ... original implementation
```

Note: Functions remain for backward compatibility but emit deprecation warnings.

1.3 Benefits

1.3.1 1. Better Encapsulation

- Evaluation logic belongs with the model, not in a separate utility module
- Model-specific configuration (like `label_col`) is used automatically

1.3.2 2. Extensibility

- Subclasses can override `_create_evaluators()` for custom metrics
- RNN models could add sequence-specific evaluation metrics
- Different model types can have different evaluation strategies

1.3.3 3. Consistency

- All evaluation goes through the model interface
- No need to remember to pass the correct `label_col` to evaluators
- Follows object-oriented design principles

1.3.4 4. Type Safety

- Methods are part of the `SkolModel` interface
- Better IDE support and type checking
- Clear relationship between models and their evaluation

1.4 Example Usage

1.4.1 Before:

```
from skol_classifier.utils import calculate_stats

# Train model
classifier.fit(train_data)

# Evaluate - need to import separate function
predictions = classifier.predict(test_data)
stats = calculate_stats(predictions)
```

1.4.2 After:

```
# Train model
classifier.fit(train_data)
```

```
# Evaluate - use model's method
predictions = classifier.predict(test_data)
stats = classifier._model.calculate_stats(predictions)
```

1.4.3 Internal Usage (classifier_v2.py):

```
# Split data
train_data, test_data = featured_df.randomSplit([0.8, 0.2], seed=42)

# Predict on test set
test_predictions = self._model.predict(test_data)

# Calculate stats using model's method
stats = self._model.calculate_stats(test_predictions, verbose=False)
```

1.5 Future Enhancements

1.5.1 Custom Evaluators for Different Model Types

Subclasses can now override `_create_evaluators()`:

```
class RNNSkolModel(SkolModel):
    def _create_evaluators(self):
        # Call parent for standard metrics
        evaluators = super()._create_evaluators()

        # Add RNN-specific metrics
        evaluators['sequence_accuracy'] = SequenceEvaluator(...)

        return evaluators
```

1.5.2 Model-Specific Statistics

Models can override `calculate_stats()` for custom reporting:

```
class RNNSkolModel(SkolModel):
    def calculate_stats(self, predictions, verbose=True):
        # Get standard stats
        stats = super().calculate_stats(predictions, verbose=False)

        # Add RNN-specific stats
        stats['avg_sequence_length'] = self._calculate_avg_seq_length(prediction

        if verbose:
            # Custom verbose output for RNN
            self._print_rnn_stats(stats)
        return stats
```

1.6 Migration Guide

1.6.1 For Users

No immediate action required. Deprecated functions still work but emit warnings.

To update code: 1. Replace `calculate_stats(predictions)` with `model.calculate_stats(predictions)` 2. Remove imports of `calculate_stats` from `skol_classifier.utils`

1.6.2 For Developers

1. When adding new model types, consider if custom evaluation is needed
2. Override `_create_evaluators()` for custom metrics
3. Override `calculate_stats()` for custom statistics reporting
4. Always use `self.label_col` instead of hardcoding `"label_indexed"`

1.7 Files Modified

- `skol_classifier/base_model.py`: Added `_create_evaluators()` and `calculate_stats()`
- `skol_classifier/classifier_v2.py`: Changed to use `model.calculate_stats()`
- `skol_classifier/utils.py`: Added deprecation warnings to old functions
- `skol_classifier/__init__.py`: No changes (exports remain for compatibility)

1.8 Backward Compatibility

✅ **Fully backward compatible** - Old `calculate_stats()` function still works - Emits deprecation warning to guide migration - Will be removed in future major version