Contents

1 Custom Loss Function Serialization Fix

1.1 Problem

When using class weights with RNN models, a custom `weighted_categorical_crossentropy` loss function is created. This caused serialization errors when:

1. **Distributed Prediction**: Model is serialized to JSON for Spark UDFs
2. **Model Loading**: Model is loaded from disk (.h5 files) or Redis

1.1.1 Error Message

```
TypeError: Could not locate function 'weighted_categorical_crossentropy'.
Make sure custom classes and functions are decorated with `@keras.saving.registe
```

1.1.2 Root Cause

The custom loss function is created as a closure inside `build_bilstm_model()`:

```
def build_bilstm_model(..., class_weights=None, labels=None):
    if class_weights is not None:
        # Create weight tensor
        weight_tensor = tf.constant(weight_list, dtype=tf.float32)

        # Create custom loss function (CLOSURE - captures weight_tensor)
        def weighted_categorical_crossentropy(y_true, y_pred):
            loss = -tf.reduce_sum(y_true * tf.math.log(y_pred), axis=-1)
            class_indices = tf.argmax(y_true, axis=-1)
            weights = tf.gather(weight_tensor, class_indices)
            weighted_loss = loss * weights
            return tf.reduce_mean(weighted_loss)

        loss_fn = weighted_categorical_crossentropy
```

When Keras serializes the model (via `to_json()` or `save()`), it saves: - The model architecture - The **name** of the loss function: `"weighted_categorical_crossentropy"` - The compiled configuration

When deserializing, Keras tries to find this function but can't because: 1. It's not a registered Keras function 2. It's a dynamically created closure 3. Each model has a different closure (different `weight_tensor`)

1.2 Solution

For **prediction only**, we don't need the loss function. The solution is to provide a dummy version of the custom loss function that Keras can deserialize, then load without compilation:

1.2.1 1. Distributed Prediction (Spark UDF)

File: `skol_classifier/rnn_model.py` **Lines**: 971-978

```
# Before
model = keras.models.model_from_json(model_config)

# After - provide custom_objects with dummy loss function
def weighted_categorical_crossentropy(y_true, y_pred):
    """Dummy loss function for model deserialization. Not used for prediction.""
    return tf.keras.losses.categorical_crossentropy(y_true, y_pred)

custom_objects = {'weighted_categorical_crossentropy': weighted_categorical_cros
model = keras.models.model_from_json(model_config, custom_objects=custom_objects
```

Note: `model_from_json()` accepts `custom_objects` parameter to handle custom loss functions.

1.2.2 2. Loading from Disk (RNNSkolModel)

File: `skol_classifier/rnn_model.py` **Lines**: 1609-1619

```
# Before
self.keras_model = keras.models.load_model(path)

# After - provide custom_objects with dummy loss and compile=False
def weighted_categorical_crossentropy(y_true, y_pred):
    """Dummy loss function for model deserialization. Not used for prediction.""
    import tensorflow as tf
    return tf.keras.losses.categorical_crossentropy(y_true, y_pred)

custom_objects = {'weighted_categorical_crossentropy': weighted_categorical_cros
self.keras_model = keras.models.load_model(path, custom_objects=custom_objects,
```

Note: `load_model()` **does** support `compile=False`, but we also provide `custom_objects` for safety.

1.2.3 3. Loading from Disk/Redis (via SkolClassifierV2)

File: `skol_classifier/classifier_v2.py` **Lines**: 983-998 (disk), 1151-1161 (redis)

```
# Before
keras_model = keras.models.load_model(str(classifier_path))

# After - provide custom_objects with dummy loss and compile=False
def weighted_categorical_crossentropy(y_true, y_pred):
    """Dummy loss function for model deserialization. Not used for prediction.""
    return tf.keras.losses.categorical_crossentropy(y_true, y_pred)

custom_objects = {'weighted_categorical_crossentropy': weighted_categorical_cros
keras_model = keras.models.load_model(
    str(classifier_path),
    custom_objects=custom_objects,
    compile=False
)
```

1.3 Why This Works

1.3.1 Custom Objects + `compile=False`

The solution uses two complementary approaches:

1. **`custom_objects` dictionary**: Provides a dummy `weighted_categorical_crossentropy` function
   - Keras can now deserialize the model config (which references this function name)
   - The dummy function just returns standard categorical crossentropy
   - Since we're only doing prediction, the actual loss function is never called
   - Works with both `model_from_json()` (for UDFs) and `load_model()` (for disk/Redis)
2. **`compile=False` parameter** (for `load_model()` only):
   - Skips loading the optimizer state and compiling the model
   - Loads only the architecture and weights
   - Model can still be used for prediction via `model.predict()`
   - Much faster loading since no compilation step
   - Note: `model_from_json()` doesn't have a `compile` parameter, so we only use this with `load_model()`

1.3.2 For Prediction

```
# This works fine without compilation
predictions = model.predict(X)
probabilities = model.predict(X)  # softmax outputs
```

1.3.3 For Training

If you need to continue training, call `fit()` which rebuilds the model:

```
# Load model (not compiled)
classifier.load_model()

# Call fit() - this rebuilds and recompiles with current class_weights
classifier.fit()  # Works! Rebuilds model with loss function
```

1.4 Impact

1.4.1 ✅ What Still Works

1. **Prediction**: All prediction methods work normally
   - `classifier.predict()`
   - `classifier.predict_proba()`
   - Distributed prediction via Spark
2. **Training**: Can still train/retrain models
   - `classifier.fit()` rebuilds the model with proper loss
   - Class weights are reapplied
3. **Model Saving/Loading**: No changes to workflow

- `classifier.save_model()`
- `classifier.load_model()`

1.4.2 ⚠️ Limitations

1. **Cannot inspect loss function**: After loading, `model.loss` is not available
   - This is fine - we don't use it for prediction
   - Training rebuilds it anyway
2. **Cannot continue training without fit()**: Can't call `model.fit()` directly on loaded Keras model
   - Solution: Use `classifier.fit()` which rebuilds the model
3. **Metrics not available**: Loaded model has no compiled metrics
   - Solution: Use `classifier.model.calculate_stats()` for evaluation

1.5 Alternative Solutions Considered

1.5.1 ❌ Registering Custom Loss Function

```
@keras.saving.register_keras_serializable()
def weighted_categorical_crossentropy(y_true, y_pred):
    # Problem: Can't capture weight_tensor in decorated function
    # Each model has different weights
    pass
```

Why not: The decorator requires a static function, but our loss is a closure with model-specific weights.

1.5.2 ❌ Custom Loss Class

```
@keras.saving.register_keras_serializable()
class WeightedCategoricalCrossentropy(keras.losses.Loss):
    def __init__(self, weights, **kwargs):
        super().__init__(**kwargs)
        self.weights = weights

    def call(self, y_true, y_pred):
        # Implementation
        pass
```

Why not: More complex, requires changes to `build_bilstm_model()`, and still needs special handling for serialization of the weights array.

1.5.3 ✅ `compile=False` (Chosen Solution)

Why yes: - Simple one-line change - No architectural changes needed - Works for all use cases - Standard Keras pattern for prediction-only loading - Faster loading

1.6 Testing

1.6.1 Test That It Works

```python
from skol_classifier.classifier_v2 import SkolClassifierV2
import redis

# 1. Train with class weights
classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    weight_strategy='inverse',  # Uses custom loss
    model_storage='redis',
    redis_client=redis.Redis(),
    redis_key='test_model'
)
classifier.fit()
classifier.save_model()

# 2. Load and predict (should work now!)
new_classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    model_storage='redis',
    redis_client=redis.Redis(),
    redis_key='test_model',
    auto_load_model=True  # Loads with compile=False
)

# Prediction works
predictions = new_classifier.predict(test_data)  # ✅ Works!

# Can also retrain
new_classifier.fit()  # ✅ Works! Rebuilds with loss
```

1.6.2 Verify Distributed Prediction

```python
# Large dataset - uses Spark UDFs
large_data = spark.read.parquet("large_dataset.parquet")
```

```
# This serializes model to JSON for UDFs
predictions = classifier.predict(large_data)  # ✅ Works!
```

1.7 Files Modified

1. **`skol_classifier/rnn_model.py`**:
   - Lines 971-978: Added `custom_objects` parameter to `model_from_json()` with dummy loss function
   - Lines 1609-1619: Added `custom_objects` and `compile=False` to `load_model()`
   - Updated docstring for `load()` method
2. **`skol_classifier/classifier_v2.py`**:
   - Lines 983-998: Added `custom_objects` and `compile=False` to `load_model()` in `_load_model_from_disk()`
   - Lines 1151-1161: Added `custom_objects` and `compile=False` to `load_model()` in `_load_model_from_redis()`
   - Fixed `_load_model_from_disk()` to properly handle RNN models (was trying to load .h5 as PipelineModel)

1.8 References

- Keras documentation: Saving & Loading Models
- TensorFlow guide: Custom Losses
- Related: `docs/class_weights_implementation_summary.md`