Contents

1 Focal F1 Loss for RNN Models

1.1 Overview

The **focal F1 loss** is a custom loss function designed for highly imbalanced classification problems where you only care about a subset of classes. Instead of optimizing for overall accuracy or weighted cross-entropy, it directly optimizes the mean F1 score for the labels you specify.

1.2 When to Use Focal F1 Loss

Use focal F1 loss when: 1. **You have severe class imbalance** (e.g., 1:10:100+ ratios) 2. **You only care about specific minority classes** (e.g., Nomenclature and Description, but not Misc) 3. **F1 score is your evaluation metric** - optimizing F1 directly often works better than using weighted loss 4. **Other approaches haven't worked** - try this if weighted cross-entropy isn't giving good results

1.2.1 Focal F1 Loss vs. Class Weights

Feature	Focal F1 Loss	Class Weights
Optimization target	Mean F1 score for focal labels	Weighted cross-entropy
Labels considered	Only specified focal labels	All labels (weighted differently)
Best for	When you only care about specific classes	When all classes matter but some more than others
F1 optimization	Direct	Indirect
Interpretability	Training loss = 1 - mean F1	Training loss = weighted cross-entropy

1.3 Usage

1.3.1 Basic Example

```
from skol_classifier.classifier_v2 import SkolClassifierV2

classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=['data/annotated/*.ann'],
    model_type='rnn',
    window_size=15,
    hidden_size=128,
    num_layers=2,
    epochs=6,
    focal_labels=['Nomenclature', 'Description'],  # Only optimize for these lab
    verbosity=1
)

results = classifier.fit()
```

```
# Output shows:
# [BiLSTM] Using mean F1 loss for focal labels:
#   Nomenclature
#   Description
```

1.3.2 Comparing Approaches

```
import redis
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("F1 Loss Comparison").getOrCreate()
redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)

annotated_files = ['data/annotated/*.ann']

# Common config
base_config = {
    'spark': spark,
    'input_source': 'files',
    'file_paths': annotated_files,
    'model_type': 'rnn',
    'window_size': 15,
    'hidden_size': 128,
    'num_layers': 2,
    'epochs': 6,
    'batch_size': 32,
    'verbosity': 1
}

# 1. Standard loss (baseline)
print("\n=== Standard Cross-Entropy ===")
classifier1 = SkolClassifierV2(
    model_storage='redis',
    redis_client=redis_client,
    redis_key='rnn_standard',
    **base_config
)
results1 = classifier1.fit()
print(f"Nomenclature F1: {results1['test_stats']['Nomenclature_f1']:.4f}")
print(f"Description F1: {results1['test_stats']['Description_f1']:.4f}")

# 2. Weighted cross-entropy
print("\n=== Weighted Cross-Entropy ===")
classifier2 = SkolClassifierV2(
    model_storage='redis',
    redis_client=redis_client,
    redis_key='rnn_weighted',
    weight_strategy='inverse',  # Automatic weights
    **base_config
)
results2 = classifier2.fit()
print(f"Nomenclature F1: {results2['test_stats']['Nomenclature_f1']:.4f}")
print(f"Description F1: {results2['test_stats']['Description_f1']:.4f}")

# 3. Focal F1 loss
print("\n=== Focal F1 Loss ===")
classifier3 = SkolClassifierV2(
    model_storage='redis',
    redis_client=redis_client,
    redis_key='rnn_focal_f1',
    focal_labels=['Nomenclature', 'Description'],  # Optimize F1 for these only
    **base_config
)
results3 = classifier3.fit()
print(f"Nomenclature F1: {results3['test_stats']['Nomenclature_f1']:.4f}")
print(f"Description F1: {results3['test_stats']['Description_f1']:.4f}")

# Compare results
print("\n=== Comparison ===")
print(f"Nomenclature F1 improvement:")
print(f"  Weighted: +{(results2['test_stats']['Nomenclature_f1'] - results1['tes
print(f"  Focal F1: +{(results3['test_stats']['Nomenclature_f1'] - results1['tes
```

1.4 How It Works

1.4.1 Soft F1 Score

The focal F1 loss uses a **differentiable approximation** of F1 score:

```
# For each focal label:
tp = sum(y_true * y_pred)              # True positives (soft)
fp = sum((1 - y_true) * y_pred)        # False positives (soft)
fn = sum(y_true * (1 - y_pred))        # False negatives (soft)

precision = tp / (tp + fp + epsilon)
recall = tp / (tp + fn + epsilon)

f1 = 2 * precision * recall / (precision + recall + epsilon)
```

The loss returned is **1 - mean(F1)** across focal labels, so minimizing loss maximizes F1.

1.4.2 Why Soft F1?

Traditional F1 requires hard predictions (0 or 1), which aren't differentiable. Soft F1: - Uses probabilities directly (no thresholding) - Is differentiable everywhere - Allows gradient-based optimization - Approximates true F1 well in practice

1.4.3 Ignoring Non-Focal Labels

If you specify `focal_labels=['Nomenclature', 'Description']`, the model: - ✅ **Maximizes F1** for Nomenclature and Description - ❌ **Ignores** Misc class entirely in the loss - □ Still **predicts all classes** at inference time

This is useful when: - Some classes are unimportant (Misc lines are okay to misclassify) - You want to focus training on what matters

1.5 Advanced Usage

1.5.1 Focal Labels with Custom Model Parameters

```
classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=annotated_files,
    model_type='rnn',

    # Architecture
    window_size=20,         # Larger context window
    hidden_size=256,        # More capacity
    num_layers=3,           # Deeper network
    dropout=0.4,            # Higher dropout for regularization

    # Training
    epochs=10,              # More training
    batch_size=16,          # Smaller batches

    # Focal F1 loss
    focal_labels=['Nomenclature', 'Description'],

    verbosity=2
)

results = classifier.fit()
```

1.5.2 Manual Specification via model_params

```
# Alternative: pass focal_labels via model_params
classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=annotated_files,
    model_type='rnn',
    focal_labels=['Nomenclature', 'Description'],  # Direct parameter (recommend
    verbosity=1
)

# OR (equivalent):
classifier = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=annotated_files,
    model_type='rnn',
    **{'focal_labels': ['Nomenclature', 'Description']}  # Via model_params
)
```

1.6 Restrictions

1.6.1 Mutually Exclusive with Class Weights

You **cannot** use both `focal_labels` and `class_weights` together:

```
# ❌ ERROR: Both specified
classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    class_weights={'Nomenclature': 100.0, 'Misc': 0.1},  # ❌
    focal_labels=['Nomenclature', 'Description']         # ❌
)
# Raises: ValueError: class_weights and focal_labels are mutually exclusive

# ✅ OK: Choose one
classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',
    focal_labels=['Nomenclature', 'Description']  # ✅
)
```

1.6.2 RNN Models Only

Focal F1 loss is only available for RNN models:

```python
# ❌ ERROR: focal_labels with non-RNN model
classifier = SkolClassifierV2(
    spark=spark,
    model_type='logistic',                            # ❌ Not RNN
    focal_labels=['Nomenclature', 'Description']    # Only works with RNN
)

# ✅ OK: RNN model
classifier = SkolClassifierV2(
    spark=spark,
    model_type='rnn',                                 # ✅ RNN
    focal_labels=['Nomenclature', 'Description']
)
```

1.7 Expected Results

1.7.1 Typical Improvements

With a 1:10:100 imbalance (Nomenclature:Description:Misc):

Approach	Nomenclature F1	Description F1	Notes
Standard loss	0.15 - 0.30	0.50 - 0.65	Baseline
Weighted loss	0.30 - 0.50	0.60 - 0.75	Good improvement
Focal F1 loss	**0.40 - 0.60**	**0.65 - 0.80**	**Best for focal labels**

Note: Focal F1 loss may give worse results on non-focal labels (Misc) since it doesn't optimize for them at all.

1.7.2 What to Expect During Training

```
[BiLSTM] Using mean F1 loss for focal labels:
  Nomenclature
  Description

[RNN Fit] Rebuilding model to apply focal F1 loss...

Epoch 1/6
432/432 - 45s - loss: 0.6234 - accuracy: 0.7521
```

```
Epoch 2/6
432/432 - 43s - loss: 0.4156 - accuracy: 0.8012
Epoch 3/6
432/432 - 43s - loss: 0.3421 - accuracy: 0.8234
...

Test Results:
  Nomenclature_precision: 0.5234
  Nomenclature_recall: 0.4821
  Nomenclature_f1: 0.5019        ← Directly optimized!
  Description_precision: 0.7012
  Description_recall: 0.6834
  Description_f1: 0.6922          ← Directly optimized!
  Misc_precision: 0.8934
  Misc_recall: 0.9123
  Misc_f1: 0.9027                 ← Not optimized (may be worse)
```

1.8 Troubleshooting

1.8.1 Loss Not Decreasing

Symptom: Training loss stays high (> 0.8) and doesn't decrease

Possible causes: 1. Learning rate too high or too low 2. Not enough training data for focal classes 3. Classes are truly hard to separate

Solutions:

```python
# Try adjusting architecture
classifier = SkolClassifierV2(
    ...,
    hidden_size=256,      # Increase capacity
    num_layers=3,         # Add depth
    dropout=0.3,          # Reduce dropout if underfitting
    epochs=12             # Train longer
)

# Or reduce learning rate (requires manual model building)
# By default uses Adam with lr=0.001
```

1.8.2 F1 Score Lower Than Expected

Symptom: Test F1 scores are worse than with weighted loss

Possible causes: 1. Validation set distribution differs from training 2. Need more epochs for F1 optimization to converge 3. Soft F1 approximation not perfect for your data

Solutions:

```
# Train longer
classifier = SkolClassifierV2(..., epochs=10)

# Try weighted loss instead
classifier = SkolClassifierV2(..., weight_strategy='aggressive')

# Combine: use focal F1 for initial training, then fine-tune with weighted loss
```

1.8.3 Non-Focal Labels Have Bad Performance

Symptom: Misc class has very poor F1

This is expected! Focal F1 loss ignores non-focal labels during training.

Solutions: 1. If you care about Misc, add it to `focal_labels`: `['Nomenclature', 'Description', 'Misc']` 2. Use `class_weights` instead if you care about all classes 3. Accept the tradeoff - focal F1 is for when you DON'T care about certain labels

1.9 Implementation Details

1.9.1 Code Location

The focal F1 loss is implemented in: - **skol_classifier/rnn_model.py**: - `build_bilstm_model()`: Lines 132-200 (loss function definition) - `RNNSkolModel.__init__()`: Lines 401, 467 (parameter handling) - `RNNSkolModel.fit()`: Lines 758-776 (model rebuilding)

1.9.2 Loss Function Signature

```
def mean_f1_loss(y_true, y_pred):
    """
    Mean F1 loss for specified focal labels.

    Args:
        y_true: One-hot encoded true labels, shape (batch, timesteps, num_classe
        y_pred: Predicted probabilities, shape (batch, timesteps, num_classes)

    Returns:
        Loss scalar (1 - mean F1 across focal labels)
    """
```

1.9.3 Custom Loss Registration

The loss function is automatically handled during serialization: - Dummy loss provided for `model_from_json()` and `load_model()` - See docs/custom_loss_serialization_fix.md for details

1.10 References

- Related: docs/class_weights_usage.md - Weighted cross-entropy approach
- Related: docs/weight_strategy_usage.md - Automatic weight calculation
- Related: docs/class_imbalance_strategies.md - All strategies
- Paper: Focal Loss for Dense Object Detection - Inspiration (though different approach)
- Resource: Differentiable F1 Score - Similar concept