Contents

1 Auto-Load Feature for Redis Models

1.1 Overview

The SKOL Classifier now automatically loads models from Redis when initializing, if the specified Redis key exists. This eliminates the need to manually call `load_from_redis()` and makes the API more intuitive.

1.2 How It Works

When you create a `SkolClassifier` with a Redis client and key:

1. The constructor checks if the Redis key exists
2. If it exists, the model is automatically loaded
3. If it doesn't exist or loading fails, initialization continues normally
4. You can check `classifier.labels` to see if a model was loaded

1.3 Basic Usage

1.3.1 Before (Manual Loading)

```python
import redis
from skol_classifier import SkolClassifier

redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)

# Initialize
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

# Manually load
if classifier.load_from_redis():
    print("Model loaded!")
else:
    print("No model found")
```

1.3.2 After (Auto-Loading)

```python
import redis
from skol_classifier import SkolClassifier

redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)

# Initialize - model auto-loads!
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

# Check if loaded
if classifier.labels is not None:
    print("Model loaded!")
else:
    print("No model found")
```

1.4 Common Patterns

1.4.1 Pattern 1: Train Once, Use Many Times

```python
import redis
from skol_classifier import SkolClassifier, get_file_list

redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)

# First run: trains and saves
classifier = SkolClassifier(redis_client=redis_client, redis_key="production_mod
if classifier.labels is None:
    # No model exists, train it
    files = get_file_list("/data/annotated")
    classifier.fit(files)
    classifier.save_to_redis()
    print("Model trained and saved")

# Subsequent runs: just loads
# classifier = SkolClassifier(redis_client=redis_client, redis_key="production_m
# Model is already loaded! Ready to use.
```

1.4.2 Pattern 2: Microservice with Lazy Loading

```python
import redis
from skol_classifier import SkolClassifier

class TextClassifierService:
    def __init__(self):
        self.redis_client = redis.Redis(
            host='redis',
            port=6379,
            decode_responses=False
        )
        # Model auto-loads when service starts
        self.classifier = SkolClassifier(
            redis_client=self.redis_client,
            redis_key="production_model_v1"
        )

        if self.classifier.labels is None:
            raise RuntimeError("Model not found in Redis!")

    def classify(self, text_files):
        return self.classifier.predict_raw_text(text_files)

# Service starts with model already loaded
service = TextClassifierService()
```

1.4.3 Pattern 3: Fallback Strategy

```python
import redis
from skol_classifier import SkolClassifier

redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)
```

```
# Try to load from Redis (auto)
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

if classifier.labels is None:
    # Fallback to disk
    print("Redis model not found, loading from disk...")
    classifier.load_from_disk("/backup/models/classifier")

if classifier.labels is None:
    # Last resort: train
    print("No cached model, training from scratch...")
    files = get_file_list("/data/annotated")
    classifier.fit(files)
```

1.5 Disabling Auto-Load

Sometimes you want to create a fresh classifier without loading:

```
# Disable auto-load
classifier = SkolClassifier(
    redis_client=redis_client,
    redis_key="my_model",
    auto_load=False
)

# Now you have a fresh classifier, even if "my_model" exists in Redis
# You can train a new model and overwrite the old one
```

This is useful when: - Retraining an existing model - Training multiple model variants - Benchmarking different configurations

1.6 Error Handling

Auto-loading fails silently - if there's an error loading from Redis, the classifier simply starts without a model:

```
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

# Even if loading failed, you can still use the classifier
if classifier.labels is None:
    print("No model loaded (key doesn't exist or loading failed)")
    # Train a new model or load from elsewhere
else:
    print(f"Model loaded successfully with labels: {classifier.labels}")
```

1.7 Checking Model Status

Several ways to check if a model is loaded:

```
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

# Method 1: Check labels
if classifier.labels is not None:
    print("Model loaded")

# Method 2: Check pipeline_model
if classifier.pipeline_model is not None:
    print("Pipeline loaded")

# Method 3: Check both models
if classifier.pipeline_model and classifier.classifier_model:
    print("Both models loaded")

# Get label information
if classifier.labels:
    print(f"Model classifies into {len(classifier.labels)} categories:")
    print(f"  {classifier.labels}")
```

1.8 Performance Considerations

- **Initialization Time**: Auto-loading adds ~3-10 seconds to initialization
- **Network Overhead**: Loading from remote Redis adds latency
- **Memory**: Loaded models consume memory immediately
- **Retry Logic**: No automatic retry - fails silently on error

For latency-sensitive applications, consider: 1. Pre-warming the classifier before accepting requests 2. Using connection pooling for Redis 3. Monitoring Redis availability

1.9 Best Practices

1.9.1 1. Always Check Model Status

```
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")
assert classifier.labels is not None, "Model not found in Redis!"
```

1.9.2 2. Use Descriptive Keys

```
# Good
key = f"skol_model_{env}_{version}_{date}"
```

```python
# Not as good
key = "model"
```

1.9.3 3. Handle Missing Models Gracefully

```python
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")

if classifier.labels is None:
    # Provide helpful error message
    raise ValueError(
        "Model 'my_model' not found in Redis. "
        "Train a model first using fit() and save_to_redis()"
    )
```

1.9.4 4. Document Expected Behavior

```python
def get_classifier() -> SkolClassifier:
    """
    Get the production classifier.

    Returns:
        SkolClassifier with pre-loaded production model

    Raises:
        RuntimeError: If production model not found in Redis
    """
    redis_client = get_redis_client()
    classifier = SkolClassifier(
        redis_client=redis_client,
        redis_key="production_model_v1"
    )

    if classifier.labels is None:
        raise RuntimeError("Production model not available")

    return classifier
```

1.10 Migration from Manual Loading

If you have existing code using manual `load_from_redis()` calls:

1.10.1 Old Code

```
classifier = SkolClassifier(redis_client=redis_client)
success = classifier.load_from_redis(redis_key="my_model")
if not success:
    raise ValueError("Failed to load model")
```

1.10.2 New Code (Option 1: Use auto-load)

```
classifier = SkolClassifier(redis_client=redis_client, redis_key="my_model")
if classifier.labels is None:
    raise ValueError("Failed to load model")
```

1.10.3 New Code (Option 2: Keep explicit loading)

```
classifier = SkolClassifier(
    redis_client=redis_client,
    redis_key="my_model",
    auto_load=False  # Disable auto-load
)
success = classifier.load_from_redis()
if not success:
    raise ValueError("Failed to load model")
```

Both approaches work! The auto-load approach is more concise, while explicit loading gives you more control over error handling.

1.11 See Also

- REDIS_INTEGRATION.md - Complete Redis integration guide
- skol_classifier/README.md - Full API documentation
- examples/redis_usage.py - Working examples