Contents

0.1 CouchDB Integration for SKOL Classifier

0.2 Overview

The SKOL Classifier now supports reading raw text from CouchDB attachments and writing annotated results back to CouchDB. This is perfect for managing large document collections where:

- Raw documents are stored as `.txt` attachments in CouchDB
- Annotated results should be saved as `.txt.ann` attachments in the same documents
- Document IDs and attachment names need to be preserved

0.3 Installation

CouchDB support requires the `couchdb` package (version 1.2 or later):

```
pip install couchdb>=1.2
```

Or install with updated requirements:

```
pip install -e .
```

0.4 Features

1. **Distributed Processing**: Uses PySpark UDFs to process documents in parallel across cluster
2. **Read from CouchDB**: Load `.txt` attachments using distributed workers
3. **Automatic Processing**: Maintains document ID and attachment name through the pipeline
4. **Write to CouchDB**: Save annotated results in parallel using distributed UDFs
5. **Integration with Redis**: Combine CouchDB I/O with Redis model storage
6. **Scalability**: Handles large datasets by distributing I/O across Spark workers

   Note: This implementation uses **distributed UDFs** where each Spark worker connects to CouchDB independently. This avoids loading all data on the driver and enables horizontal scaling. See DISTRIBUTED_COUCHDB.md for details.

0.5 Quick Start

0.5.1 Basic Workflow

```
import redis
from skol_classifier import SkolClassifier

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

# Initialize classifier with Redis and CouchDB configuration
classifier = SkolClassifier(
    redis_client=redis_client,
    redis_key="production_model",
    couchdb_url="http://localhost:5984",
    database="my_documents",
    username="admin",
    password="password"
)

# Process CouchDB documents
predictions = classifier.predict_from_couchdb()

# Save back to CouchDB as .ann attachments
results = classifier.save_to_couchdb(predictions=predictions, suffix=".ann")

print(f"Processed {len(results)} documents")
```

0.6 API Reference

0.6.1 SkolClassifier Methods

0.6.1.1 `load_from_couchdb(pattern="*.txt")` Load raw text from CouchDB attachments.

Uses the CouchDB configuration set in the constructor.

Args: - `pattern`: Pattern for attachment names (default: "*.txt")

Returns: DataFrame with columns: `doc_id`, `attachment_name`, `value`

Example:

```
# Initialize with CouchDB configuration
classifier = SkolClassifier(
    couchdb_url="http://localhost:5984",
    database="documents",
    username="admin",
```

```
    password="password"
)

df = classifier.load_from_couchdb(pattern="*.txt")
print(f"Loaded {df.count()} documents")
```

0.6.1.2 predict_from_couchdb(pattern="*.txt", output_format="annotated")

Load text from CouchDB, predict labels, and return predictions.

Uses the CouchDB configuration set in the constructor.

Args: - `pattern`: Pattern for attachment names - `output_format`: Output format ('annotated' or 'simple')

Returns: DataFrame with predictions, including `doc_id` and `attachment_name`

Example:

```
# Initialize with CouchDB configuration
classifier = SkolClassifier(
    redis_client=redis_client,
    couchdb_url="http://localhost:5984",
    database="documents",
    username="admin",
    password="password"
)

predictions = classifier.predict_from_couchdb()

# View predictions
predictions.select("doc_id", "attachment_name", "predicted_label").show()
```

0.6.1.3 save_to_couchdb(predictions, suffix=".ann")

Save annotated predictions back to CouchDB as attachments.

Uses the CouchDB configuration set in the constructor.

Args: - `predictions`: DataFrame with predictions (must include `annotated_value` column) - `suffix`: Suffix to append to attachment names (default: ".ann")

Returns: List of results from CouchDB operations

Example:

```
# Initialize with CouchDB configuration
classifier = SkolClassifier(
    redis_client=redis_client,
    couchdb_url="http://localhost:5984",
```

```
    database="documents",
    username="admin",
    password="password"
)

results = classifier.save_to_couchdb(predictions=predictions, suffix=".ann")

# Check results
for r in results:
    if r['success']:
        print(f"Saved: {r['doc_id']}/{r['attachment_name']}")
    else:
        print(f"Failed: {r['doc_id']}")
```

0.6.2 CouchDBReader Class

For more control over reading from CouchDB:

```
from skol_classifier import CouchDBReader

reader = CouchDBReader(
    url="http://localhost:5984",
    database="documents",
    username="admin",
    password="password"
)

# Get all .txt attachments
attachments = reader.get_text_attachments(pattern="*.txt")

for att in attachments:
    print(f"Doc: {att['doc_id']}, File: {att['attachment_name']}")
    print(f"Content: {att['content'][:100]}...")

# Convert to Spark DataFrame
df = reader.to_spark_dataframe(spark_session, pattern="*.txt")
```

0.6.3 CouchDBWriter Class

For more control over writing to CouchDB:

```
from skol_classifier import CouchDBWriter

writer = CouchDBWriter(
    url="http://localhost:5984",
    database="documents",
```

```
    username="admin",
    password="password"
)

# Save single annotated file
results = writer.save_annotated_predictions([
    ("doc_id_123", "article.txt", "[@ Annotated content #Label]")
])

# Save from DataFrame
results = writer.save_from_dataframe(predictions_df, suffix=".ann")
```

0.7 Data Format

0.7.1 Input: CouchDB Documents

Documents in CouchDB should have text attachments:

```
{
  "_id": "article_001",
  "_rev": "1-abc123",
  "title": "My Article",
  "_attachments": {
    "article.txt": {
      "content_type": "text/plain",
      "data": "...base64 encoded content..."
    }
  }
}
```

0.7.2 Output: Annotated Attachments

The classifier adds new attachments with `.ann` suffix:

```
{
  "_id": "article_001",
  "_rev": "2-def456",
  "title": "My Article",
  "_attachments": {
    "article.txt": {
      "content_type": "text/plain",
      "data": "...original content..."
    },
    "article.txt.ann": {
      "content_type": "text/plain",
      "data": "[@ Paragraph 1 #Nomenclature]\n[@ Paragraph 2 #Description]..."
```

```
    }
  }
}
```

0.7.3 Multiple Attachments Per Document

Important: Documents can have multiple `.txt` attachments, and each will be processed independently:

```
{
  "_id": "article_001",
  "_attachments": {
    "abstract.txt": {...},
    "methods.txt": {...},
    "results.txt": {...}
  }
}
```

After processing, each `.txt` file gets a corresponding `.ann` file:

```
{
  "_id": "article_001",
  "_attachments": {
    "abstract.txt": {...},
    "abstract.txt.ann": {...},
    "methods.txt": {...},
    "methods.txt.ann": {...},
    "results.txt": {...},
    "results.txt.ann": {...}
  }
}
```

The system automatically: - Finds ALL `.txt` attachments in each document - Processes each one independently (in parallel) - Saves each result as a separate `.ann` attachment

0.8 Usage Patterns

0.8.1 Pattern 1: Complete Pipeline

Train model, save to Redis, process CouchDB documents:

```
import redis
from skol_classifier import SkolClassifier, get_file_list

# Settings
redis_client = redis.Redis(host='localhost', port=6379, decode_responses=False)
couchdb_url = "http://localhost:5984"
```

```
database = "documents"

# Initialize classifier with both Redis and CouchDB configuration
classifier = SkolClassifier(
    redis_client=redis_client,
    redis_key="production_model",
    couchdb_url=couchdb_url,
    database=database,
    username="admin",
    password="password"
)

# Train if needed
if classifier.labels is None:
    print("Training model...")
    files = get_file_list("/data/annotated")
    classifier.fit(files)
    classifier.save_to_redis()
else:
    print(f"Model loaded: {classifier.labels}")

# Process CouchDB
predictions = classifier.predict_from_couchdb()

# Save results
results = classifier.save_to_couchdb(predictions=predictions)

print(f"Processed {len(results)} documents")
```

0.8.2 Pattern 2: Batch Processing

Process documents in batches:

```
from skol_classifier import CouchDBReader

reader = CouchDBReader(couchdb_url, database, username, password)
attachments = reader.get_text_attachments()

batch_size = 100
for i in range(0, len(attachments), batch_size):
    batch = attachments[i:i+batch_size]
    print(f"Processing batch {i//batch_size + 1}...")

    # Create DataFrame for batch
    batch_df = classifier.spark.createDataFrame(
        [(att['doc_id'], att['attachment_name'], att['content']) for att in batc
```

```
        ['doc_id', 'attachment_name', 'value']
    )

    # Process and save batch
    # ... processing code ...
```

0.8.3 Pattern 3: Error Handling

Handle failures gracefully:

```
predictions = classifier.predict_from_couchdb(
    couchdb_url=couchdb_url,
    database=database,
    username="admin",
    password="password"
)

results = classifier.save_to_couchdb(
    predictions=predictions,
    couchdb_url=couchdb_url,
    database=database,
    username="admin",
    password="password"
)

# Check for failures
failed = [r for r in results if not r['success']]
if failed:
    print(f"Failed to save {len(failed)} documents:")
    for r in failed:
        print(f"  {r['doc_id']}/{r['attachment_name']}: {r['error']}")

    # Retry failed documents
    # ... retry logic ...
```

0.8.4 Pattern 4: Custom Attachment Suffix

Use different suffixes for different model versions:

```
# Save with versioned suffix
results = classifier.save_to_couchdb(
    predictions=predictions,
    couchdb_url=couchdb_url,
    database=database,
    username="admin",
    password="password",
```

```
    suffix=".v2.ann"  # Custom suffix
)

# Document will have: article.txt.v2.ann
```

0.9 CouchDB Setup

0.9.1 Create Database

```
curl -X PUT http://admin:password@localhost:5984/documents
```

0.9.2 Upload Document with Attachment

```
# Create document
curl -X PUT http://admin:password@localhost:5984/documents/doc001 \
  -H "Content-Type: application/json" \
  -d '{"title": "Sample Document"}'

# Add attachment
curl -X PUT http://admin:password@localhost:5984/documents/doc001/article.txt?re
  -H "Content-Type: text/plain" \
  --data-binary @article.txt
```

0.9.3 View Attachments

```
# List documents
curl http://admin:password@localhost:5984/documents/_all_docs

# Get document with attachments
curl http://admin:password@localhost:5984/documents/doc001

# Download attachment
curl http://admin:password@localhost:5984/documents/doc001/article.txt
```

0.10 Docker CouchDB

Run CouchDB in Docker for development:

```
docker run -d \
  --name couchdb \
  -p 5984:5984 \
  -e COUCHDB_USER=admin \
  -e COUCHDB_PASSWORD=password \
  couchdb:latest
```

Connect from Python:

```
classifier = SkolClassifier()

predictions = classifier.predict_from_couchdb(
    couchdb_url="http://localhost:5984",  # Docker host
    database="documents",
    username="admin",
    password="password"
)
```

0.11 Authentication

0.11.1 Basic Authentication

```
predictions = classifier.predict_from_couchdb(
    couchdb_url="http://localhost:5984",
    database="documents",
    username="admin",
    password="password"
)
```

0.11.2 No Authentication

```
predictions = classifier.predict_from_couchdb(
    couchdb_url="http://localhost:5984",
    database="documents"
    # No username/password
)
```

0.11.3 Environment Variables

```
import os

predictions = classifier.predict_from_couchdb(
    couchdb_url=os.getenv("COUCHDB_URL"),
    database=os.getenv("COUCHDB_DATABASE"),
    username=os.getenv("COUCHDB_USER"),
    password=os.getenv("COUCHDB_PASSWORD")
)
```

0.12 Performance Considerations

0.12.1 Large Documents

- **Memory**: Large attachments are loaded into memory

- **Batching**: Process documents in batches for better memory management
- **Spark**: Leverage Spark's distributed processing for large datasets

0.12.2 Network Latency

- **Local CouchDB**: Best performance with local CouchDB instance
- **Caching**: Consider caching models in Redis for faster access
- **Connection Pooling**: Reuse connections when processing multiple documents

0.12.3 CouchDB Indexing

Create views for faster document retrieval:

```
// Design document for finding text attachments
{
  "_id": "_design/attachments",
  "views": {
    "text_files": {
      "map": "function(doc) { if (doc._attachments) { for (var att in doc._attac
    }
  }
}
```

0.13 Troubleshooting

0.13.1 Connection Errors

Issue: Cannot connect to CouchDB

```
Error: Connection refused
```

Solution: Check CouchDB is running and URL is correct

```
curl http://localhost:5984/
```

0.13.2 Authentication Errors

Issue: Unauthorized

```
Error: 401 Unauthorized
```

Solution: Verify credentials

```
# Test connection
import requests
r = requests.get(
```

```
    "http://localhost:5984/_all_dbs",
    auth=("admin", "password")
)
print(r.json())
```

0.13.3 Document Update Conflicts

Issue: Document update conflict

```
Error: 409 Conflict
```

Solution: The document was modified between read and write. The code automatically handles this by fetching the latest `_rev` before updating.

0.13.4 Attachment Too Large

Issue: Attachment exceeds size limit

```
Error: 413 Payload Too Large
```

Solution: Increase CouchDB's max attachment size in configuration

0.14 Best Practices

1. **Use Redis for Models**: Store trained models in Redis for fast loading
2. **Batch Processing**: Process documents in batches to manage memory
3. **Error Handling**: Always check results for failed uploads
4. **Version Suffixes**: Use versioned suffixes (`.v1.ann`, `.v2.ann`) for different model versions
5. **Monitoring**: Monitor CouchDB disk usage as annotated files accumulate
6. **Backup**: Regularly backup CouchDB database
7. **Indexing**: Create CouchDB views for faster document queries

0.15 Complete Example

See examples/couchdb_usage.py for complete working examples.

0.16 See Also

- REDIS_INTEGRATION.md - Redis model storage
- AUTO_LOAD_FEATURE.md - Auto-loading models
- skol_classifier/README.md - Full API documentation
- CouchDB Documentation