Contents

1 Command-Line Arguments for Environment Configuration

1.1 Overview

The `bin/env_config.py` module now supports command-line arguments in addition to environment variables. This allows you to override configuration settings on a per-invocation basis without modifying environment variables.

1.2 Configuration Priority

Settings are resolved in the following order (highest priority first):

1. **Command-line arguments** (e.g., `--couchdb-database mydb`)

2. **Environment variables** (e.g., `COUCHDB_DATABASE=mydb`)
3. **Default values** (e.g., `'skol_dev'`)

1.3 Argument Naming Convention

Configuration dictionary keys use underscores, while command-line arguments use dashes:

- Config key: `couchdb_database`
- Argument: `--couchdb-database`
- Config key: `prediction_batch_size`
- Argument: `--prediction-batch-size`

1.4 Usage Examples

1.4.1 Basic Override

Override the CouchDB database name:

```
./bin/with_skol bin/predict_classifier.py --couchdb-database test_db
```

1.4.2 Multiple Overrides

Override multiple configuration values:

```
./bin/with_skol bin/train_classifier.py \
    --couchdb-database training_db \
    --cores 8 \
    --prediction-batch-size 48 \
    --redis-port 6380
```

1.4.3 Mixed with Script-Specific Arguments

The argument parser uses `parse_known_args()`, so it won't interfere with script-specific arguments:

```
./bin/with_skol bin/predict_classifier.py \
    --model logistic_sections \
    --verbosity 1 \
    --couchdb-database custom_db \
    --cores 8
```

In this example: - `--model` and `--verbosity` are parsed by `predict_classifier.py` - `--couchdb-database` and `--cores` are parsed by `env_config.py`

1.5 Available Arguments

1.5.1 CouchDB Settings

```
--couchdb-url <url>              # CouchDB server URL
--couchdb-host <host>            # CouchDB host (e.g., 127.0.0.1:5984)
--couchdb-username <username>    # CouchDB username
--couchdb-password <password>    # CouchDB password
--couchdb-database <database>    # Main CouchDB database name
--couchdb-pattern <pattern>      # Attachment pattern (e.g., *.txt)
```

1.5.2 Ingest Database Settings

```
--ingest-url <url>               # Ingest database URL
--ingest-database <database>     # Ingest database name
--ingest-username <username>     # Ingest database username
--ingest-password <password>     # Ingest database password
--ingest-db-name <name>          # Ingest database name
```

1.5.3 Taxon Database Settings

```
--taxon-url <url>                # Taxon database URL
--taxon-database <database>      # Taxon database name
--taxon-username <username>      # Taxon database username
--taxon-password <password>      # Taxon database password
--taxon-db-name <name>           # Taxon database name
```

1.5.4 Training Database Settings

```
--training-database <database>   # Training database name
```

1.5.5 Redis Settings

```
--redis-host <host>              # Redis host
--redis-port <port>              # Redis port (integer)
--redis-url <url>                # Redis URL
```

1.5.6 Model Settings

```
--model-version <version>        # Model version
--classifier-model-expire <seconds>  # Model expiration time
```

1.5.7 Embedding Settings

```
--embedding-name <name>          # Embedding name
--embedding-expire <seconds>     # Embedding expiration time (integer)
```

1.5.8 Prediction Settings

```
--pattern <pattern>              # Pattern for annotated files
--prediction-batch-size <size>   # Prediction batch size (integer)
--num-workers <num>              # Number of workers (integer)
--union-batch-size <size>        # DataFrame union batch size (integer, default:
                                 # Controls memory usage during large-scale oper
```

1.5.9 Data Paths

```
--annotated-path <path>          # Path to annotated data directory
```

1.5.10 Spark Settings

```
--cores <num>                    # Number of Spark cores (integer)
--bahir-package <package>        # Bahir package specification
--spark-driver-memory <size>     # Spark driver memory (e.g., 4g)
--spark-executor-memory <size>   # Spark executor memory (e.g., 4g)
```

1.6 Programmatic Access

In your Python scripts, simply call `get_env_config()` as before:

```python
from env_config import get_env_config

config = get_env_config()

# Command-line args are automatically parsed and override env vars
database = config['couchdb_database']
cores = config['cores']
```

1.7 Implementation Details

1.7.1 How It Works

1. `get_env_config()` first builds a base configuration from environment variables and defaults
2. It then creates an `ArgumentParser` with `add_help=False` to avoid conflicts
3. All known configuration keys are registered as optional arguments
4. `parse_known_args()` is called to parse recognized arguments while ignoring unknown ones
5. Parsed arguments override the base configuration values

1.7.2 Type Handling

- **String arguments**: Most configuration values (URLs, names, patterns)
- **Integer arguments**: Port numbers, batch sizes, worker counts, expiration times
- **Path arguments**: File system paths (converted to Path objects)

1.7.3 Compatibility

The implementation is fully backward compatible:

- Existing scripts work without modification
- Scripts with their own argument parsers are not affected
- Unknown arguments are silently ignored
- The function signature hasn't changed

1.8 Testing

To test the command-line argument parsing:

```
# Test without arguments (uses env vars and defaults)
python3 bin/test_env_config_args.py

# Test with arguments
python3 bin/test_env_config_args.py \
    --couchdb-database test_db \
    --cores 8 \
    --redis-port 6380

# Test with unknown arguments (should not break)
python3 bin/test_env_config_args.py \
    --some-unknown-arg value \
    --couchdb-database override_db
```

1.9 Related Documentation

- Progress Tracking
- PDF Page Marker Preservation