SimpleML Documentation

Release 0.7

Elisha Yadgaran

Oct 08, 2019

Contents

Machine learning that just works, for effortless production applications.

It was inspired by common patterns I found myself developing over and over again for new modeling projects. At the core, it is designed to be minimally intrusive and provide a clean abstraction for the most common workflows. Each supported framework details methods to save, version, and subsequently load objects to easily move from training to production.

SimpleML is designed for data scientists comfortable writing code. Please refer to the enterprise version (*Enterprise*) for details on the extended offering. The enterprise version contains a non-technical interface to SimpleML as well as additional components to streamline the rest of the machine learning product workflow.

CHAPTER 1

What It Is

SimpleML is a framework that manages the persistence and tracking of machine learning objects.

CHAPTER 2

What It Is NOT

As an abstracted persistence layer, SimpleML does not define any native predictive algorithms. It wraps existing ones with convenience methods to save, load, and otherwise manage modeling work.

Prototypical Use Cases:

- deploy locally trained models to remote servers
- define model configs to be trained on a remote server
- experiment with hundreds of different config combinations and track performance

Why use SimpleML over a SAS cloud solution?

- Avoid vendor lockin - fully open source codebase, compatible with any cloud infrastructure and algorithm backend.
- Drop in replacement for most workflows
- Can still deploy your models on-prem without changing your application

CHAPTER 3

Supported Frameworks

SimpleML can easily be extended to support almost any modeling framework. These are the ones that have been developed already:

	Supervised	Unsupervised	Reinforcement Learning
Scikit-Learn	X		
Keras	X		
PyTorch			
Tensorflow			
Theano			
AI-Gym			
Caffe			
CNTK			
MXNet			

CHAPTER 4

Source

You can access the source code at: https://github.com/eyadgaran/SimpleML

CHAPTER 5

Contributing

See guidelines here: Contributing

CHAPTER 6

Support

SimpleML core is open source and is powered by generous donations. Please donate if you find it contributing to your projects. Technical support and contract opportunities are also available - contact the author, Elisha Yadgaran, for details.

Ready to get started? Check out the *Quickstart* guide.

CHAPTER 7

Index

7.1 SimpleML Enterprise Edition

7.1.1 Why Enterprise?

SimpleML is geared towards data scientists, automating the repetitive workflow and extending their capabilities. The enterprise version aims to do the same for the rest of your organization by enabling analysts, developers, operations, and potentially even end-users to utilize machine learning. The platform is designed to be non-technical, but builds upon SimpleML to yield the same functionality for training models and extends beyond that to cover the entire workflow.

7.1.2 Enterprise Features

- Drag & Drop GUI for anyone in your organization to train models
- Automated Dev-Ops for single click model deployment

7.1.3 Moving to Enterprise, and Back Again

SimpleML enterprise is intentionally built around the open-source core, so developer functionality and interaction will not change. Non technical developers (analysts and/or devops) will lose access to the visual GUI and deployment shortcuts. *Again, all the same code functionality will remain, but it will require developers to code it up.*

7.1.4 Pricing

Enterprise use is customizable and can be tailored to your organization's needs. Contact the author, Elisha Yadgaran for details.

7.2 Installation

SimpleML currently runs best on Python 2.7.x and 3.5+; other versions may work but are not explicitly supported.

You can install SimpleML via several different methods. The simplest is via pip:

```
pip install simpleml
```

While the above is the simplest method, the recommended approach is to create a virtual environment via conda, before installation. Assuming you have conda installed, you can then open a new terminal session and create a new virtual environment:

```
conda create -n simpleml python
source activate simpleml
```

Once the virtual environment has been created and activated, SimpleML can be installed via `pip install simpleml` as noted above. Alternatively, if you have the project source, you can install SimpleML using the distutils method:

```
cd path-to-source
python setup.py install
```

If you have Git installed and prefer to install the latest bleeding-edge version rather than a stable release, use the following command:

```
pip install -e "git+https://github.com/eyadgaran/simpleml.git@master#egg=simpleml"
```

7.2.1 Optional packages

SimpleML comes configured with a few optional dependencies. The base installation will work without any issues if they are not installed, but extended functionality will not be available.

The general command is `pip install simpleml[extras]` (substituting "extras" with the group of dependencies)

These are the current supported extras:

```
'postgres': ["psycopg2"]
'deep-learning': ["keras", "tensorflow"]
'hdf5': ["hickle"]
'cloud': ["onedrivesdk", "apache-libcloud", "pycrypto"]
```

Additionally, a convenience extra titled `all` is defined to install the full list of optional dependencies.

7.2.2 Dependencies

When SimpleML is installed, the following dependent Python packages should be automatically installed. These are necessary dependencies and SimpleML will not function properly without them.

- sqlalchemy, to manage database communication
- sqlalchemy_mixins, for database active record functionality
- alembic, for database schema management
- pandas, for data processing

- numpy, for numerical foundations
- cloudpickle, for code pickling
- future, for Python 2 and 3 compatibility
- configparser, for config management
- simplejson, extended support for json handling
- scikit-learn, base machine learning support and metric computation

7.2.3 Upgrading

If you installed SimpleML via `pip` and wish to upgrade to the latest stable release, you can do so by adding `--upgrade`:

```
pip install --upgrade simpleml
```

If you installed via distutils or the bleeding-edge method, simply perform the same step to install the most recent version.

7.3 Configuration

SimpleML contains a series of fallback locations for defining configuration parameters.

By default, it will start with the `simpleml.conf` file in the library folder (with the exception of the folder path, since it can't read that before knowing where to look...).

The next location, if the config file does not contain a value, is through an environment variable.

Finally, it will default to hard-coded values configured as default class parameters.

To recap:

```
1) Configuration File (`simpleml.conf`)
2) Environment Variable (`SIMPLEML_{parameter}`)
3) Code Defaults (Check class definitions)
```

7.3.1 Configuration File

The configuration file is designed as follows:

```
[path]
home_directory = ~/.simpleml

[cloud]
section = gcp-read-only

[onedrive]
client_secret = aaaaaabbbbbbbbbcccccc
root_id = xxxxxyyyyyyzzzzzzzz
client_id = abcdefg-hijk-lmno-pqrs-tuvwxyz
scopes = onedrive.readwrite
redirect_uri = http://localhost:8000/example/callback

```

(continues on next page)

(continued from previous page)

```
[gcp-read-write]
driver = GOOGLE_STORAGE
connection_params = key,secret
key = read-write@iam.gserviceaccount.com
secret = ./gcp-read-write.json
container = simpleml

[gcp-read-only]
driver = GOOGLE_STORAGE
connection_params = key,secret
key = read-only@iam.gserviceaccount.com
secret = ./gcp-read-only.json
container = simpleml

[simpleml-database]
database=SimpleML
username=simpleml
password=simpleml
drivername=postgresql
host=localhost
port=5432

[app-database]
database=APPLICATION_DB
username=simpleml
password=simpleml
drivername=postgresql
host=localhost
port=5432
```

Note: python configparser interprets '%' as a special character for interpolation. Add a second, like '%%' to escape the literal value

Only the sections that are used are necessary. Don't include unnecessary sections in your config to minimize security exposure if they leak! This entire file would be unnecessary if cloud storage is not being used, default filepaths are used, and the database connections are initialized by a different means than *configuration_section*. Breaking down this particular example:

```
[cloud]  <-- This section is used for any persistable that specifies a cloud␣
↪persistence location
section = gcp-read-only  <-- The name of the heading for the cloud credentials

[onedrive]  <--- This section outlines an example authorization scheme with onedrive␣
↪personal
client_secret = aaaaaabbbbbbbbbcccccc  <--- Put your client secret here
root_id = xxxxxyyyyyyzzzzzzzz  <--- Put the item id of the root filestore bucket here
client_id = abcdefg-hijk-lmno-pqrs-tuvwxyz  <--- Put your client_id here
scopes = onedrive.readwrite  <--- Mark the scopes here (reference the onedrive api␣
↪for examples)
redirect_uri = http://localhost:8000/example/callback  <--- Put the callback url here␣
↪to return the auth token

[gcp-read-write]  <--- This section outlines an example for a read/write iam in GCP
driver = GOOGLE_STORAGE  <--- Apache-libcloud driver used (can be any of the␣
↪supported ones)
connection_params = key,secret  <--- Which parameters in this section to pass to␣
↪apache-libcloud
```

(continues on next page)

(continued from previous page)

```
key = read-write@iam.gserviceaccount.com  <--- The gcp iam account
secret = ./gcp-read-write.json  <--- The token for that gcp account
container = simpleml  <--- The gcp container (or "bucket") that houses the files

[gcp-read-only]  <--- Duplicate example with a read only IAM -- recommended practice␣
↪to train with the cloud section = gcp-read-write and deploy in production with read␣
↪only access
driver = GOOGLE_STORAGE
connection_params = key,secret
key = read-only@iam.gserviceaccount.com
secret = ./gcp-read-only.json
container = simpleml

[simpleml-database]  <--- Database credentials for the simpleml models (used by␣
↪specifying Database(configuration_section='simpleml-database'))
database=SimpleML
username=simpleml
password=simpleml
drivername=postgresql
host=localhost
port=5432

[app-database]  <--- Database credentials for application logs (used by specifying␣
↪Database(configuration_section='app-database'))
database=APPLICATION_DB
username=simpleml
password=simpleml
drivername=postgresql
host=localhost
port=5432
```

7.3.2 Environment Variables

The full list of variables that can be referenced are:

```
- SIMPLEML_CONFIGURATION_FILE
- SIMPLEML_DATABASE_NAME
- SIMPLEML_DATABASE_USERNAME
- SIMPLEML_DATABASE_PASSWORD
- SIMPLEML_DATABASE_HOST
- SIMPLEML_DATABASE_PORT
- SIMPLEML_DATABASE_DRIVERNAME
- SIMPLEML_DATABASE_CONF
- SIMPLEML_DATABASE_URI
```

7.3.3 Code Defaults

Defaults are specified for expected database parameters:

```
- SIMPLEML_CONFIGURATION_FILE = ~/.simpleml/simpleml.conf
- SIMPLEML_DATABASE_NAME = SimpleML
- SIMPLEML_DATABASE_USERNAME = simpleml
- SIMPLEML_DATABASE_PASSWORD = simpleml
```

(continues on next page)

(continued from previous page)

```
- SIMPLEML_DATABASE_HOST = localhost
- SIMPLEML_DATABASE_PORT = 5432
- SIMPLEML_DATABASE_DRIVERNAME = postgresql
- SIMPLEML_DATABASE_CONF = None
- SIMPLEML_DATABASE_URI = None
```

The first is the location of the configuration file. The remainder are database initialization defaults – used only if a database class is initialized without the particular parameters.

7.4 Quickstart

Reading through all the documentation is highly recommended, but for the truly impatient, following are some quick steps to get started.

7.4.1 Installation

Assuming conda is installed, you can create a new environment as follows (this step is optional, but recommended to provide a clean working environment):

```
conda create -n simpleml python
source activate simpleml
```

Install SimpleML on Python 2.7.x or Python 3.5+ by running the following command:

```
pip install simpleml
```

Refer to the installation guide for other methods of installation (*Installation*)

7.4.2 Set Up a Database

Postgres is the preferred (and tested) database flavor, but since sqlalchemy is used to manage all communication, any supported database should work. While it is possible to use SimpleML on an existing database (provided there are no overlapping tables), it is recommended to create a new database with the appropriate role-based access.

For example (using postgres), we can create a user *simpleml* and a database *SIMPLEML*:

```
CREATE USER simpleml with password 'simpleml';
CREATE DATABASE SIMPLEML OWNER simpleml;
```

7.4.3 Create a Project

Once the database is up and running, you can get started on the modeling task. There are no restrictions on the project setup, though a juptyer notebook is advised for prototyping and a python package for deployed services.

Follow on below for an example project in a notebook. Check out the examples repository for larger implementations.

You can now install jupyter, and any other dependencies via pip:

```
pip install jupyter
```

Starting up the jupyter notebook is as simple as calling:

```
jupyter notebook
```

7.4.4 Train a Model

From inside the notebook we will conduct a very minimal modeling exercise using the titanic dataset from kaggle:

```
from simpleml.utils import Database
from simpleml.datasets import PandasDataset
from simpleml.pipelines import RandomSplitPipeline
from simpleml.transformers import SklearnDictVectorizer, DataframeToRecords,␣
↪FillWithValue
from simpleml.models import SklearnLogisticRegression
from simpleml.metrics AccuracyMetric
from simpleml import TEST_SPLIT


# Initialize Database Connection
db = Database(
  username='simpleml', password='simpleml', database='SIMPLEML', host='localhost'
).initialize(upgrade=True)

# Define Dataset and point to loading file
class TitanicDataset(PandasDataset):
    def build_dataframe(self):
        self._external_file = self.load_csv('filepath/to/train.csv')

# Create Dataset and save it
dataset = TitanicDataset(name='titanic', label_columns=['Survived'])
dataset.build_dataframe()
dataset.save()

# Define the minimal transformers to fill nulls and one-hot encode text columns
transformers = [
    ('fill_zeros', FillWithValue(values=0.)),
    ('record_coverter', DataframeToRecords()),
    ('vectorizer', SklearnDictVectorizer())
]

# Create Pipeline and save it - Use basic 80-20 test split
pipeline = RandomSplitPipeline(name='titanic', transformers=transformers,
                               train_size=0.8, validation_size=0.0, test_size=0.2)
pipeline.add_dataset(dataset)
pipeline.fit()
pipeline.save()

# Create Model and save it
model = SklearnLogisticRegression(name='titanic')
model.add_pipeline(pipeline)
model.fit()
model.save()

# Create Metric and save it
metric = AccuracyMetric(dataset_split=TEST_SPLIT)
metric.add_model(model)
metric.score()
metric.save()
```

7.4.5 Deploy to Production

Production models can be hosted pretty much anywhere. We'll just define a basic API layer using flask and serve predictions from our trained model:

```
from flask import Flask, jsonify, request
import pandas as pd
from simpleml.utils import PersistableLoader

# Initialize Database Connection
db = Database(
  username='simpleml', password='simpleml', database='SIMPLEML', host='localhost'
).initialize()

app = Flask(__name__)
MODEL = PersistableLoader.load_model(name='titanic', version=1)

@app.route(/predict, methods=['POST'])
def predict()
    X = pd.DataFrame(request.json)
    prediction_probability = float(MODEL.predict_proba(X)[:, 1])
    prediction = int(round(prediction_probability, 0))
    return jsonify({'probability': prediction_probability, 'prediction': prediction}),
↪ 200


if __name__ == '__main__':
    app.run()
```

7.5 Persistables

Persistables are the fundamental blocks of SimpleML objects. They are defined as the abstract base that details the expected methods and attributes of all inheriting subclasses.

7.5.1 SQLAlchemy

7.5.2 Registry

7.5.3 Hashing

7.5.4 Saving

7.5.5 Serializing

7.6 Datasets

7.6.1 Uniqueness

When creating a dataset persistable, there are a few limitations to ensure uniqueness. The main one is that the data (or reference to it, depending on hashing rules) is unambiguously captured in the hash. This allows different datasets to be compared and even the same dataset to be referenced in different runtimes. A cascading effect of this is that the data

has to be accessible and unchanging over time - otherwise the hash would be considered unstable and yield different experiment results during different executions.

More concretely this means that a dataset cannot be defined uniquely by a sql query or file based data generator. It has to also include the data, in a stable form to ensure the same outcomes downstream. This is the main reason that the default hashing routine includes the resulting data (at the cost of delayed hash evaluation).

It is, however, possible to introduce a different form of dataset iteration after the dataset has been "built" and saved. This has the advantage of maintaining a reproducible data source while allowing custom iterations over that data for later use. The API lists the commands and parameters to tune usage.

7.7 Pipelines

7.7.1 Dataset Reference

When creating a pipeline persistable, the referenced dataset is assumed to exist already. This subtle but important detail means that generators or other lazy consumption mechanisms cannot be used as part of the initial creation. When transforming after for model use, it is possible to introduce a different form of dataset iteration. The API lists the commands and parameters to tune usage.

7.8 Models

7.9 Metrics

7.10 Utilities

7.11 API Reference