Online Prediction Framework (OPF)

See the OPF Guide for an overview of this API.

Here is the complete program we are going to use as an example. In sections below, we'll break it down into parts and explain what is happening (without some of the plumbing details).

```
import csv
import datetime
import os
import yaml
from itertools import islice

from nupic.frameworks.opf.model_factory import ModelFactory

_NUM_RECORDS = 3000
_EXAMPLE_DIR = os.path.dirname(os.path.abspath(__file__))
_INPUT_FILE_PATH = os.path.join(_EXAMPLE_DIR, os.pardir, "data", "gymdata.csv")
_PARAMS_PATH = os.path.join(_EXAMPLE_DIR, os.pardir, "params", "model.yaml")



def createModel():
  with open(_PARAMS_PATH, "r") as f:
    modelParams = yaml.safe_load(f)
  return ModelFactory.create(modelParams)



def runHotgym(numRecords):
  model = createModel()
```

```
    model.enableInference({"predictedField": "consumption"})
    with open(_INPUT_FILE_PATH) as fin:
      reader = csv.reader(fin)
      headers = reader.next()
      reader.next()
      reader.next()

      results = []
      for record in islice(reader, numRecords):
        modelInput = dict(zip(headers, record))
        modelInput["consumption"] = float(modelInput["consumption"])
        modelInput["timestamp"] = datetime.datetime.strptime(
          modelInput["timestamp"], "%m/%d/%y %H:%M")
        result = model.run(modelInput)
        bestPredictions = result.inferences["multiStepBestPredictions"]
        allPredictions = result.inferences["multiStepPredictions"]
        oneStep = bestPredictions[1]
        oneStepConfidence = allPredictions[1][oneStep]
        fiveStep = bestPredictions[5]
        fiveStepConfidence = allPredictions[5][fiveStep]

        result = (oneStep, oneStepConfidence * 100,
                  fiveStep, fiveStepConfidence * 100)
        print "1-step: {:16} ({:4.4}%)\t 5-step: {:16} ({:4.4}%)".format(*result)
        results.append(result)
      return results


  if __name__ == "__main__":
    runHotgym(_NUM_RECORDS)
```

Model Parameters

Before you can create an OPF model, you need to have model parameters defined in a file. These model parameters contain many details about how the HTM network will be constructed, what encoder configurations will be used, and individual algorithm parameters that can drastically affect how a model operates. The model parameters we're using in this Quick Start can be found here.

To use model parameters, they can be written to a file and imported into your script. In this example, our model parameters existing in a YAML file called `params.yaml` and are identical to those linked above.

Create an OPF Model

The easiest way to create a model once you have access to model parameters is by using the **`ModelFactory`**.

```
import yaml
from nupic.frameworks.opf.model_factory import ModelFactory

_PARAMS_PATH = "/path/to/model.yaml"

with open(_PARAMS_PATH, "r") as f:
  modelParams = yaml.safe_load(f)

model = ModelFactory.create(modelParams.MODEL_PARAMS)

# This tells the model the field to predict.
model.enableInference({'predictedField': 'consumption'})
```

The resulting `model` will be an instance of **`HTMPredictionModel`**.

Feed the Model Data

The raw input data file is described here in detail.

Our model parameters define how this data will be encoded in the `encoders` section:

```
  # List of encoders and their parameters.
  encoders:
    consumption:
      fieldname: consumption
      name: consumption
      resolution: 0.88
      seed: 1
      type: RandomDistributedScalarEncoder
    timestamp_timeOfDay:
      fieldname: timestamp
      name: timestamp_timeOfDay
      timeOfDay: [21, 1]
      type: DateEncoder
    timestamp_weekend:
      fieldname: timestamp
      name: timestamp_weekend
      type: DateEncoder
      weekend: 21
```

Notice that three semantic values are being encoded into the input space. The first is the scalar energy `consumption` value, which is being encoded with the **`RandomDistributedScalarEncoder`**. The next two values represent two different aspects of time using the **`DateEncoder`**. The encoder called `timestamp_timeOfDay` encodes the time of day, while the `timestamp_weekend` encoder will output different representations for weekends vs weekdays. The **`HTMPredictionModel`** will combine these encodings using the **`MultiEncoder`**.

> For details about encoding and how these encoders work, see the HTM School episodes on encoders.

Now that you see the raw input data and how it is configured to be encoded into binary arrays for the HTM to process it, let's see the code that actually reads the CSV data file and runs it through our `model`.

```
import csv
import datetime

# Open the file to loop over each row
with open ("gymdata.csv") as fileIn:
  reader = csv.reader(fileIn)
  # The first three rows are not data, but we'll need the field names when
  # passing data into the model.
  headers = reader.next()
  reader.next()
  reader.next()

  for record in reader:
    # Create a dictionary with field names as keys, row values as values.
    modelInput = dict(zip(headers, record))
    # Convert string consumption to float value.
    modelInput["consumption"] = float(modelInput["consumption"])
    # Convert timestamp string to Python datetime.
    modelInput["timestamp"] = datetime.datetime.strptime(
      modelInput["timestamp"], "%m/%d/%y %H:%M"
    )
    # Push the data into the model and get back results.
    result = model.run(modelInput)
```

Extract the results

In the classifier configuration of our model parameters, identified as `modelParams.clParams`, the `steps` value tells the model how many steps into the future to predict. In this case, we are predicting both one and five steps into the future as shown by the value `1,5`.

This means the `results` object will have prediction information keyed by both `1` and `5`.

```
result = model.run(modelInput)
bestPredictions = result.inferences['multiStepBestPredictions']
allPredictions = result.inferences['multiStepPredictions']
oneStep = bestPredictions[1]
fiveStep = bestPredictions[5]
# Confidence values are keyed by prediction value in multiStepPredictions.
oneStepConfidence = allPredictions[1][oneStep]
fiveStepConfidence = allPredictions[5][fiveStep]

result = (oneStep, oneStepConfidence * 100,
          fiveStep, fiveStepConfidence * 100)
print "1-step: {:16} ({:4.4}%)\t 5-step: {:16} ({:4.4}%)".format(*result)
```

As you can see in the example above, the `results` object contains an `inferences` property that contains all the information about predictions. This includes the following keys:

- `multiStepBestPredictions`: Contains information about the *best* prediction that was returned for the last row of data.
- `multiStepPredictions`: Contains information about *all* predictions for the last row of data, including confidence values for each prediction.

Each of these dictionaries should have a key corresponding to the *steps ahead* for each prediction. In this example, we are retrieving predictions for both `1` and `5` steps ahead (which was defined in the Model Parameters).

In order to get both the best prediction as well as the confidence in the prediction, we need to find the value for the best prediction from the `multiStepBestPredictions` structure, then use it to find the confidence in the `multiStepPredictions` (for both `1` and `5` step predictions).

When this example program is run, you can see both predictions and their confidences in the console output, which should look something like this:

```
  1-step:             35.7 (65.53%)   5-step:             35.7 (99.82%)
  1-step:             38.9 (65.73%)   5-step:             23.5 (99.82%)
  1-step:             36.6 (99.11%)   5-step:             35.7 (99.81%)
  1-step:             38.9 (85.73%)   5-step:             36.6 (99.96%)
  1-step:             38.2 (89.59%)   5-step:             38.2 (92.61%)
```

Congratulations! You've got HTM predictions for a scalar data stream!

NuPIC

Navigation

Quick search

Go