PFA: Model Validation Proposal

The idea of model validation is to provide a way to ensure that a model produces expected output. We support this by adding an optional top-level field called `validate`.

`validate` is a record with five defined fields: `margin`, `inputs`, `outputs`, `zeroTolerance`, and `infinityTolerance`. The field `margin` is of Avro type `double`, constrained to be within the range $[0, 1)$, and represents the margin of error used when comparing numeric fields for validation. The fields `inputs` and `outputs` are arrays of JSON objects whose types match the Avro schema defined in the top-level `input` and `output` fields, respectively. The fields `zeroTolerance` and `infinityTolerance` are of Avro type `double`, and used for comparing numeric fields that are near zero or near infinity, respectively. These fields are optional: when not specified, they have the default values $1.0E-8$ and $1.0E80$, respectively.

Behavior

Model validation occurs at the end of the third execution phase of a PFA scoring engine (as the last set of additional checks performed, but prior to the standard initialization of the engine), as described in Section 3.1 of the PFA standard.

Validation consists of a "mock scoring run" with some additional checks:

First, the PFA scoring engine iterates over `validate`'s `outputs` array, and checks that each element of this array conforms to the Avro schema specified in the top-level `output` field.

If all outputs pass validation, next the engine checks that there are the same number of elements in the `inputs` array as in the `outputs` array. If this is the case, the engine is initialized as it would be at the beginning of a scoring run (phase 4). The engine then executes the PFA document's `begin` method (if defined).

Next the engine iterates over `validate`'s `inputs` array, doing the following for each input datum:

1. Checks that the input datum conforms to the Avro schema specified by `input`.
2. Executes the model's `action` method on the input datum.
3. Compares the produced output datum to the corresponding expected output datum defined in the `outputs` array, as described below.

Finally, the engine executes the `end` step (if defined). If no errors are produced at any step, the engine resets its state, repeats engine initialization, and begins regular scoring behavior.

Comparing JSON objects

Produced outputs are compared to the expected outputs (specified by the `outputs` array) based on the Avro type of the object:

- Any non-numeric primitive type (e.g. a string) must be identical.
- Integer-valued types (`int`, `long`, `bytes`, etc.) must also be identical.
- Fractional-valued types (`float` and `double`) are compared using the percentage difference rule described below.
- Complex types are compared by recursively comparing each of their elements and fields, according to the rules above.

Fractional-valued type comparison

The `margin` field in `validate` is used to specify a percentage margin of error used to determine whether two fractional-valued objects (`double` or `float`) are the same. Let y denote an expected output (or field thereof), and $\hat{y}$ the corresponding output produced by the engine. Then, y and $\hat{y}$ are compared using the following rules:

1. If $|y| \geq$ `infinityTolerance`, $|\hat{y}| \geq$ `infinityTolerance`, and $sign(y) = sign(\hat{y})$, then $|y|$ and $|\hat{y}|$ are considered equal.
2. If $|y| \leq$ `zeroTolerance` and $|\hat{y}| \leq$ `zeroTolerance`, then y and $\hat{y}$ are considered equal.
3. If both y and $\hat{y}$ are between `zeroTolerance` and `infinityTolerance` and they satisfy the inequality:

$$\frac{|y - \hat{y}|}{|y|} \leq \texttt{margin}$$

   then they are considered equal.
4. In any other case, y and $\hat{y}$ are not considered equal.

Syntax Examples

This section provides some examples of the syntax described above.

YAML

```yaml
input: double
output: double
validate:
  margin: 0.01
  zeroTolerance: 0.005
  infinityTolerance: 10000.0
  inputs:
    - 3.14
    - -3.14
  outputs:
    - 3.14
    - 3.14
action:
  - {m.abs: input}
```

JSON

```json
{
  "input": "double",
  "output": "double",
  "validate": {
    "margin": 0.05,
    "inputs": [
      1.0,
      2.0,
      3.0
    ],
    "outputs": [
      2.0,
      3.0,
      4.0
    ]
  },
  "action": {
    "+": [
      "input",
      1
    ]
  }
}
```