

Data Science Toolkit - The Bonsai language

Article • 02/13/2024

The Bonsai language is designed to write decision tree logic for custom predictive models. This page explains the syntax and core semantics of the Bonsai Language. For additional reference, we've also provided the full grammar in Backus-Naur Form notation.

Expressions

A decision tree written in Bonsai is structured as a `branch` or a series of branches written as `if/elif/else` and/or `switch` expressions. The outcome of a branch is known as a `leaf` or `leaf value`. What variable receives the `leaf value` depends on the custom model being used. For example, if you used a [Custom Model](#) with a `model_output` that has a type of `bid` the `leaf` would be applied to the CPM bid price of the using the custom model.

ⓘ Note

Bids that are negative or zero (0) are not supported unless explicitly specified in the feature description. Do not use these expressions or unexpected results may occur. It is best to specify `no_bid` explicitly using [Bonsai Smart Leaves](#).

If/Elif/Else

Every `if` must have a matching `else`. There can be 0 or more `elif` expressions between the `if` and the matching `else`.

Example

```
if country = "US":  
    1  
elif user_hour range (8, 12):  
    if domain = "news.com":  
        0.85  
    else:  
        0.2  
else:  
    0.1
```

This example shows an `if/elif/else` expression with a nested `if/else` expression. The decision path of this example is as follows:

1. Does the user reside in the United States?
 - If yes, bid 1.
 - If no, go to step 2.
2. Is it between 8 am and 12 pm in the user's timezone?
 - If yes, go to step 3.
 - If no, bid 0.1.
3. Is the domain "news.com"?
 - If yes, bid 0.85.
 - If no, bid 0.2.

Switch

A `switch` expression can be used as an alternative to many chained `if/elif` statements. This can be useful if you have many values you want to test for a single feature.

Some things to note about `switch` expressions:

- Each case must begin with the keyword `case`.
- A case can be a single value, a list of values, or a range. Ranges can only be used for numerical values.
- The value, list of values, or range of values for each case must be wrapped in parentheses.
- A range is denoted by two periods (`..`). You can use a range with a one or two bounds. Bounds are inclusive. For example, (`.. 3`) applies to all cases numbered up to and including 3.
- A case can lead to a leaf or to another expression.
- A `switch` expression must conclude with a `default` case. This ensures that every leaf of the tree has a value.

Example

```
switch user_hour:
  case (1 .. 3):
    0.4
  case (4, 6, 8):
    if country present:
      0.8
    else:
      0.4
  default: 0.1
```

The decision path of this example is as follows:

1. Is the user's timezone between 1am and 3am?
 - If yes, bid 0.4.
 - If no, go to step 2.
2. Is it 4, 6, or 8 am in the user's timezone?

- If yes, go to step 3.
- If no, bid 0.1.

3. Is the country present?

- If yes, bid 0.8.
- If no, bid 0.4.

Conditions

In `if` and `elif` statements, you can use the `any` and `every` conditions to apply Boolean logic to a series of features.

ⓘ Note

It's not possible to combine `any` and `every` conditions in a single `if` or `elif` statement.

Any

When you use the `any` condition in an `if` or `elif` statement, the comma separators between feature values are effectively "ors". This means that if **any one** of the features in the statement matches the impression, the leaf will be evaluated.

In this example, if `hour` is present OR `country` is present OR and it is Sunday, Monday, or Tuesday, the `1` leaf will be evaluated:

```
if any user_hour present, country present, user_day < 4:
    1
else:
    0.2
```

Every

When you use the `every` condition in an `if` or `elif` statement, the comma separators between feature values are effectively "ands". This means that if **all** features in the statement match the impression, the leaf will be evaluated.

In this example, if `hour` is present AND `country` is present AND it is Sunday, Monday, or Tuesday, the `1` leaf will be evaluated:

Features

A central syntactic element of Bonsai is the **feature**. A feature is composed of:

1. A keyword
2. A set of valid values

In the example below, the feature keyword is `user_day`, and its valid values are integers between `0` - `6`, where `0` is Sunday and `6` is Saturday.

```
if user_day < 6:
    1
else:
    2
```

When you use an invalid value for a feature keyword, the API will return an error informing you of the exact location and nature of the problem in your decision tree:

```
if user_day < 100:
    1
else:
    2
```

```
ERROR: Invalid value on line 1 at position 13; user_day must be between 0 and 6
```

For more information about Bonsai validation, see [Decision Tree Validation and Error Messages](#).

For a full list of available features and their valid values, see [Bonsai Features](#). Note that some features are numeric (such as `user_hour`) while others are not (like `country`). Certain numeric operators are only valid for numeric types.

ⓘ Note

The `segment` feature is handled differently than all others. It can be tested only based on:

- Segment presence
- Number of minutes since the user was added to the segment
- User-defined value passed in with the segment

For more details, see Segment Presence, Segment Age, and Segment Value in [Bonsai Features](#).

Operators

Presence/Absence


The `present` and `absent` operators are used to test for feature presence/absence:

```
if country present:
```

```
else:
    2
    1
```

Comparison

The following comparison operators are supported.

 Expand table

Operator	Meaning
=	Equal to
!=	Not equal to
/=	Not equal to
<	Less than (can be used with integers only)
>	Greater than (can be used with integers only)
<=	Less than or equal to (can be used with integers only)
>=	Greater than or equal to (can be used with integers only)

```
if country = "US":
    2
else:
    1
```

In

The `in` operator is used to test whether a feature value matches any value in a list of possible values. Note that the list must be wrapped in parentheses.

```
if country in ("US", "MX", "CA"):
    2
else:
    1
```

Range

The `range` operator is used to test a feature value for membership in an inclusive range of values.

- This operator can be used with integer values only.
- The integer range must be wrapped in parentheses.

```
if user_hour range (1, 12):
    1
else:
    0.2
```

The example above would test if the hour is anywhere between 1 am and 12 pm in the user's time zone.

ⓘ Note

When using a range to specify the cases for a switch expression, use two periods (`..`) inside the range, rather than a comma. For more information, see [Switch](#).

Not

The `not` keyword is used to negate a condition.


```
if not country present:
    1
else:
    0.2
```

Leaf

Every tree branch leads to a leaf value.

- Each leaf can be an integer or float.
- For custom models with the "bid" model_output:
 - Each leaf is a cpm bid price in the currency of the using the custom model.
- For custom models with the "bid_modifier" model_output:
 - Each leaf is a multiplier to be applied to a Xandr optimization-derived CPM bid.

In the "bid" model_output example below, when used in a in USD, the leaves result in bid values of \$1.00 CPM and \$0.25 CPM.

```
if not country present:
    1
else:
    0.25
```

Smart leaf

Smart leaves dynamically modify bids based on a number of estimated ad performance and delivery metrics.

- Smart leaves support both the "bid" and "bid_modifier" model_output.
- Smart leaves allow you to specify a "no_bid" value.

- Smart leaves allow you to name leaves with the `leaf_name` field.
- Bid values can be derived from:
 - Estimated IAB viewthrough rate
 - Estimated average price
 - Estimated clear price
 - Uniform random numbers

In the example below, the bid price is the estimated average price multiplied by 1.5 and offset by \$0.03, with a minimum value of \$1 and a maximum value of \$5:

```
if country = "US":  
    leaf_name: "10000"  
    value: compute(estimated_average_price, 1.50, 0.03, 1.00,  
5.00)
```

For more information, see [Bonsai Smart Leaves](#).

Indentation

The tab character (`\t`) is used to indicate the indentation of a line, which in turn is used to determine the grouping of expressions (similar to Python). At the moment, only tabs are supported (spaces are ignored). The newline character (`\n`) is used to mark the end of a line.

Comments

The `#` character is for comments. A comment is a line of text that Bonsai won't try to run as code. It's just for humans to read.

Comments make your code easier to understand. When you look back at your code or others want to collaborate with you, they can read your comments and easily figure out what your code does.

- Each comment must be on its own line and must start with the # character.
- If you write a multi-line comment, each line must start with the # character.
- At this time, it is not possible to insert comments on the same line as other Bonsai code.

Valid comment

```
# Evaluate if hour is between 1am and 12pm in the user's time
zone. If so, bid $1.00. If not, bid $0.20.
if user_hour range (1, 12):
    1
else:
    0.2
```

Valid comment

```
# Evaluate if hour is between 1am and 12pm in the user's time
zone.
# If so, bid $1.00. If not, bid $0.20.
if user_hour range (1, 12):
    1
else:
    0.2
```

Invalid comment

```
if user_hour range (1, 12): # Evaluate if hour is between 1am and
12pm in the user's time zone. if so, bid $1.00. If not, bid
$0.20.
    1
else:
    0.2
```

Full grammar

This full grammar is in Backus-Naur Form notation.

Expand source

```
expression : ifExpression
            | switchExpression
            | leaf
            ;
ifExpression : 'if' condition consequent alternative
            ;
condition : anyCondition
          | everyCondition
          | notCondition
          | segmentCondition
          | scalarCondition
          ;
conditionList : simpleCondition
              | simpleCondition ',' conditionList
              ;
simpleCondition : simpleNotCondition
               | segmentCondition
               | scalarCondition
               ;
simpleNotCondition : 'not' simpleCondition
                  ;
anyCondition : 'any' conditionList
              ;
everyCondition : 'every' conditionList
               ;
notCondition : 'not' condition
              ;
segmentCondition : 'segment' INT segmentSubconditionList
                  | 'segment' INT
                  ;
segmentSubconditionList : segmentSubcondition segmentSubconditionList
                        | segmentSubcondition
                        ;
segmentSubcondition : age < INT
                    | age > INT
```

```

        | value < INT
        | value > INT
    ;
    scalarCondition : feature_keyword 'present'
        | feature_keyword 'absent'
        | feature_keyword 'in' valueList
        | feature_keyword '=' value
        | feature_keyword '>' value
        | feature_keyword '<' value
        | feature_keyword '/=' value
        | feature_keyword 'range' '(' value ',' value ')'
    ;
    valueList : value
        | '(' value ',' valueList ')'
    ;
    value | determind by feature;
    feature_keyword : 'country'
        | 'region'
        | 'city'
        | 'value'
        | 'cookie_age'
        | 'user_hour'
        | 'size'
        | 'user_day'
        | 'os_family'
        | 'os_extended'
        | 'browser'
        | 'language'
        | 'domain'
        | 'position'
        | 'placement'
        | 'placement_group'
        | 'publisher'
        | 'seller_member_id'
        | 'mobile_app'
        | 'advertiser_day_freq'
        | 'advertiser_life_freq'
        | 'advertiser_recency'
        | 'supply_type'
        | 'carrier'
        | 'device_type'
        | 'device_model'
        | 'zip'
        | 'user_gender'
        | 'dma'
    ;
    consequent : ':' INDENT expression DEINDENT

```

```

        ;
alternative : 'elif' condition consequent alternative
            | 'else' ':' INDENT expression DEINDENT
switchExpression : 'switch' feature_keyword ':' INDENT caseList DEIN-
DENT
        ;
caseList : case caseList
        | default
        ;
case : 'case' valueList ':' INDENT expression DEINDENT
default : 'default' ':' INDENT expression DEINDENT
leaf : INT
    | FLOAT
    | smartLeaf
    ;
smartLeaf : keyVal smartLeaf
    | keyVal
    ;
keyVal : value
    | learn
    | name
    ;
value : 'value' ':' valueVals
    ;
learn : 'is_learn' ':' BOOL
    ;
name : 'leaf_name' ':' ID
    ;
valueVals : computeLeaf
    | 'no_bid'
    | numeric
    ;
computeLeaf : 'compute' '(' inputField ',' computeVal ',' computeVal
',' computeVal ',' computeVal ')'
inputField : 'estimated_iab_viewthrough_rate'
    | 'estimated_video_completion_rate'
    | 'estimated_average_price'
    | 'estimated_clearing_price'
    | 'estimated_click_rate'
    | 'uniform'
    ;
computeVal : numeric
    | '_'
    ;
numeric : INT
    | FLOAT
    ;

```

Related topics

- [Custom Model Parser Service](#)
- [Custom Model Service](#)
- [Bonsai Language Features](#)
- [Bonsai Smart Leaves](#)
- [Create a Bonsai Decision Tree Custom Model](#)
- [Bonsai Custom Model Workflow](#)
- [Custom Models](#)