This version edited: 28th November 2016

CC BY

Character matching and scoring

Documentation for the Tom.bio ID Visualisation Framework

Dr Richard Burkmar
Biodiversity Project Officer
Field Studies Council
Head Office
Montford Bridge
Shrewsbury
SY4 1HW

richardb@field-studies-council.org
Tel: (01743) 852125

Tomorrow's Biodiversity Project
funded by the Esmée Fairbairn
Foundation

CC BY

1 Contents

2 Introduction

Many of the visualisations within the framework that match characters – either against user input or between taxa. This document describes how character matching and scoring is implemented in the Tom.bio ID Visualisation Framework (referred to as 'the framework').

Most users of the framework do not need to read this document. It is not necessary to have a detailed understanding of how scoring works in order to either use the framework visualisations or to create new knowledge-bases for use within the framework. However, some people may want to have a better understanding of how the framework matching and scoring works and this document is aimed at them.

3 General principles

Every matching operation in the framework is capable of providing evidence both *for* a match and *against* a match between two things. So if a user specifies value X for character A based on a specimen they have at hand, then that can provide evidence for or against a match between their specimen (one thing) and a taxon in the knowledge-base (the second thing) for which a value is specified for character A.

In the framework, evidence *for* a match based on a comparison of two values scores between 0 and 1 (positive 1) whilst evidence *against* a match for the same comparison scores between 0 and -1 (negative 1). A single comparison of two values produces both evidence for *and* evidence against a match. The relationship between them is: *against* = *for* – 1.

As an example, consider a character 'body length' for which a user specifies a value of 20 (units are unimportant for our example). The value ranges recorded in the knowledge-base for 'body length' for taxon A, taxon B and taxon C are 6 to 8, 18 to 20 and 21 to 22 respectively.

The evidence for a match with taxon B, based on this character, is strong. For this comparison, evidence *for* would score 1.0 (since the specified value falls within the valid range for the taxon). The corollary of this is that it does not furnish any evidence against a match with taxon B; evidence *against* would score 0 (1-1).

The evidence for a match with taxon A, based on the values for this character, is weaker. Let's suppose that it's so weak that the evidence *for* scores 0. (Just how far apart the two values would have to be to score zero will be discussed later.) Evidence *against* would therefore score -1 (0-1).

Relative to taxon A, taxon C is a pretty close and, depending on some properties of the character (which we will discuss later), would likely furnish some evidence *for* a match. Let's say evidence *for* scores 0.7. That means that evidence against would score -0.3 (0.7-1).

It may seem counter intuitive that any given comparison should provide both evidence for *and* against a match, but it does make sense when you think about the match for taxon C. The user specified value of 20 is very close to the range for the taxon of 21 to 22 – that's evidence for a match – but it does not fall within the range – that's evidence against a match.

For any given character the *overall* matching score is obtained by summing the *for* score and the *against* score. So in our example the *overall* score for the character for taxon C would be 0.4 (0.7-

CC BY

0.3). Mathematically, we are simply transforming the *for* score, which ranges between 0 and 1, to the *overall* score which ranges between -1 and 1. The significance is semantic rather than mathematic – it provides for a more intuitive interpretation of characters and taxa where the overall balance of evidence for is greater than the balance of evidence against (positive values vs negative values).

4 Matching taxa against user input

Users of the framework visualisations can, where permitted (e.g. for the visualisations 'Single-column key' and 'Two-column key') describe a specimen or a taxon by entering character state values using the controls provided for that purpose. The image on the right shows the character state input controls for the example biscuits knowledge-base provided with the framework.

When the user specifies or changes the state for a character, the framework cycles through all the taxa in the knowledge base and calculates the matching scores (*for*, *against* and *overall*) for that character for each of the taxa as outlined previously and described in more detail, for each character type, below.

4.1 Total overall scores and character weighting

The total score for a taxon is the sum of all the *overall* scores for each specified character. But before they are summed, the overall score for each character is multiplied by a factor derived from the value of the 'Weight' specified against the character in the knowledge-base (characters worksheet).

Character weight is expressed as a value between 1 and 10 and the factor is derived thus: weight / 10. Scores for characters with a weighting of 10 are therefore not reduced at all whilst those for characters with a weighting of 1 are reduced to a tenth of their unweighted value. This enables knowledge-base authors to assign relative importance to characters.

4.2 Missing and 'not applicable' values

When a user specifies a value for a character (of any type) any taxon which has a value of 'not applicable' recorded against that character in the knowledge-base ('n/a' on the taxa worksheet) is deemed to score -1 *against* and therefore 0 *for*. Taxa with missing values for that character (" or '?' in the knowledge-base) do not score at all (i.e. *for* is 0 and *against* is 0).

4.3 Text character matching

When a user specifies a value for a text character it can take one of two forms – a single text string or multiple text strings (the latter where the knowledge-base author has specified a 'ControlType' value of 'multi' on the characters worksheet). For the purposes of matching, multiple values are considered as alternatives (think of them as being joined by 'or'). Call this set the *input values* (regardless of whether there is one or several alternatives).

For any given taxon a text-type character can be specified as a single text string or multiple alternative text strings (specified in the knowledge base using the 'or' character ('|') in the taxon worksheet. Call this set the *taxon values* (regardless of whether there is one or several alternatives).

Text character matching in the framework is simple: if at least one of the characters in the *input values* set matches any one of values in the *taxon value* set (i.e. the intersecting set is not empty),

the character scores 1 *for* and 0 *against* for that taxon. If there is no intersection at all between the two sets then the character scores 0 *for* and -1 *against* for the taxon.

4.4 Numeric character matching

In order to understand how numeric scoring works, we need to understand the concept of the 'Strictness' attribute of numeric characters as specified in characters worksheet. Strictness can take a value of between 0 and 10. It is used to indicate how much latitude to give to user-specified values for a character that are not exact matches for a taxon but are relatively close.

A strictness value of 10 – the strictest value – requires an exact match between character values specified by the user and those recorded in the knowledge-base for a taxon to score any sort of match for that character. But with a lower strictness value, values that are close but not an exact match can also score. The lower the strictness value, the more latitude there is for a match of some sort.

The scoring algorithm for the numeric character matching requires four pieces of information: the strictness value (call it *strictness*), the numeric value specified by the user (call it *userValue*), the range of values specified for this character and taxon (call it *taxonRange*) and the full range of values for this character represented across all taxa in the knowledge-base (call it *wholeRange*). Note that the value for a numeric character specified for a taxon might be a single value (as opposed to a range), but where that's the case, we just treat the lower and upper bounds of the range as equal to each other (and the value).

If *userValue* lies within *taxonRange*, the character scores 1 *for*. If outside the range, the score depends on the value of *strictness*.

A value called *latitude* is calculated thus: (1 - *strictness* / 10) * *wholeRange*. If *userValue* is less than the lower bound of *taxonRange* minus *latitude*, it scores 0 *for* (and -1 *against*). Likewise, if *userValue* is more than the upper bound of *taxonRange* plus *latitude*, it scores 0 *for* (and -1 *against*).

If *userValue* falls between the lower bound of *taxonRange* (call it *lowerTaxonRange*) and *taxonRange* - *latitude*, then score *for* is calculated as:

1 – ((*lowerTaxonRange* – *userValue*) / *latitude*)

Likewise, if *userValue* falls between the upper bound of *taxonRange* (call it *upperTaxonRange*) and *taxonRange* + *latitude*, then score *for* is calculated as:

1 – ((*userValue* – *upperTaxonRange*) / *latitude*)

4.5 Ordinal character matching

Ordinal character matching works in a similar way to numeric character matching except that it works on the ranks of the characters instead of their values.

The scoring algorithm for ordinal character matching requires four pieces of information: the strictness value (call it *strictness*), the text state selected by the user (call it *userValue*), the set of states recorded for this character against the taxon (call it *taxonValues*) and the set of states recorded for this character across all the taxa in the knowledge-base (call this *allValues*).

Notes:

- unlike simple text values, multiple values can't be specified for an ordinal character - only single select controls can be specified so *userValue* is just a single value;
- *taxonValues* is most likely to consist of a single value, but it is possible to specify more than one for a taxon in the knowledge-base (separated by 'or') in which case it is deemed to represent an ordinal range represented by the lowest and highest ranking values from the set;
- *allValues* is a *ranked* set of possible values for the character (specified on the values worksheet).

After some manipulation on the ordinal character values (see table below) the ordinal character matching algorithm simply passes its data to the numeric matching algorithm in order to carry out the matching.

Numeric matching algorithm	Derivation from the ordinal
userValue	The value of *userValue* is converted to a numeric value representing it's ordinal rank in *allValues*.
taxonRange	The *taxonValues* set is converted to a numeric range representing the lowest and highest ranks its members represent in *allValues*.
wholeRange	The length of the *allValues* set.
strictness	*strictness*

5 Character matching between taxa

One visualisation – 'Side by side comparison' – facilitates the comparison of knowledge-base values between taxa. They way in which this operates is outline below. For the purposes of the descriptions below, we will call the taxon against which the second is compared and scored as the *primary taxon* and the second we shall called the *compared taxon*.

5.1 Text character comparison between taxa

For every text character, the two sets of possible values – one from the *primary taxon* and one from the *compared taxon* – are compared using a Jaccard coefficient which measures similarity between finite sample sets defined as the size of the intersection divided by the size of the union of the two sets. Note that, for many characters, each set will consist of a single value, larger sets occurring where a taxon has alternative values for a character state (separated by 'or').

Note that this is quite different to matching a user input value against the character states for a taxon (see sectionv'Text character matching') where any intersection at all leads to a perfect match.

5.2 Numeric character comparison between taxa

For every numeric character, the lower and upper bounds of the value range specified for the *compared taxon* is compared to the value range of the *primary taxon* using the same method described in 'Numeric character matching' above. The matching score is the average of the two overall scores from these two comparisons.

5.3 Ordinal character comparison between taxa

For ordinal character comparison, each possible ordinal value for the *compared taxon* is compared against those allowed for the *primary taxon* using the same method described in 'Ordinal character matching' above. The matching score is the average from all of these comparisons.

6 Notes for programmers

Most of the processing for the matching described above under 'Matching taxa against user input' is to be found in a javascript code module dedicated to this alone: `score.js` (in the tombio subfolder of the main framework folder).

However, the `score.js` module does *not* include adjustment for character weighting – that's done in the calling routines. In the case of comparing user character input against taxa, that's the function `scoreTaxa` in the module `tombiovis.js` (in the tombio subfolder).

The module responsible for the 'Side by side comparison' visualisation (`vis3.js` in the tombio/vis3 subfolder) makes use of the routines in `score.js` but carries out the higher level stuff described in 'Character matching between taxa' itself.

Scores for individual characters and taxa are stored within javascript objects representing the taxa. The complete set of taxa objects is found in an array assigned to the `core.taxa` variable. The image below is taken from the Javascript console and shows, on the left, one of the taxon objects from `core.taxa`. You can see that the taxon has a property named after each of the knowledge-base characters, each holding an object. One of these, Taxon, is expanded to show the taxon name.

Further down, there are more properties including those, marked in blue, holding the summed values of all the scored characters for this taxon. The *for* and *against* scores are summed separately.

Above these scores you can see an object called 'matchscore' which is expanded in the image on the right. The 'matchscore' object holds the matching scores for each individual character for the taxon. In the image, an object representing one of these characters – 'Width' – has been expanded. The individual scores for the character 'Width' for this taxon are held here (outlined in red).

Note that under the section 'General principles' in this document *for* and *against* scores were described as ranging between 0 and 1 and 0 and -1 respectively, in fact the *against* scores are not negated until they are actually used to calculate overall scores.

You may also note that a separate score is held for 'not applicable' characters. This is a bit of a development artefact. If a taxon character is 'not applicable' for a character whose value is supplied by the user, then the value of '`scorena`' will be 1 and the framework will use it to drive an *against* score of -1.


1: Object
BiscuitColour: StateValue
ChocTaste: StateValue
Coating: StateValue
FillingColour: StateValue
Holes: StateValue
Length: StateValue
Pattern: StateValue
Shape: StateValue
SingleDouble: StateValue
Taxon: StateValue
character: "Taxon"
kbValue: "Romany Vanilla Cream"
status: ""
value: (...)
valueType: ""
__proto__: Object
TopTen: StateValue
Width: StateValue
Words: StateValue
charcount: 1
height: 35
lastPosInTaxain: 5
lastPosInTaxaout: -95
matchscore: Object
scoreagainst: 0.15999999999999998
scorefor: 0.6400000000000001
scoreoverall: 0.4800000000000001
x: 5
y: 205
matchscore: Object
BiscuitColour: Object
ChocTaste: Object
Coating: Object
FillingColour: Object
Holes: Object
Length: Object
Pattern: Object
Shape: Object
SingleDouble: Object
Width: Object
label: "Width"
scoreagainst: 0.19999999999999996
scorefor: 0.8
scorena: 0
scoreoverall: 0.6000000000000001