This version edited: 14th September 2019

Character matching and scoring

Documentation for the FSC Identikit

Original development funded by the Esmée Fairbairn Foundation and the Heritage Lottery Fund

1 Contents

2 Introduction

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

Most users of the Identikit 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 Identikit visualisations or to create new knowledge-bases for use within the Identikit. However, some people may want to have a better understanding of how the Identikit matching and scoring works and this document is aimed at them.

3 General principles

Every matching operation in the Identikit 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 Identikit, 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.

(Note that if the `tombiovis.opts.ignoreNegativeScoring` option is set to `true`, negative scores are *not* included. The default, if this option is not set, is to include negative scoring.)

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 (0.7-0.3). 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 Identikit 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.

When the user specifies or changes the state for a character, the Identikit 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 Identikit 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 'Latitude' attribute of numeric characters as specified in characters worksheet. Latitude for numeric characters can take any numeric value and is specified in the same units in which the character values themselves are expressed. 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 latitude value of 0 – 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 higher latitude value, values that are close but not an exact match can also score if they fall within the specified latitude.

The scoring algorithm for the numeric character matching requires three pieces of information: the latitude value (call it *latitude*), the numeric value specified by the user (call it *userValue*), the range of values specified for this character and taxon (call it *taxonRange*). Note that the value of 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 *latitude*.

If *userValue* is equal to or less than the lower bound of *taxonRange* minus *latitude*, it scores 0 *for* (and -1 *against*). Likewise, if *userValue* is equal to or 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.

Knowledge-base values for an ordinal character can, like text values, have more than one alternative. And like numeric values, they can be expressed as a range. When calculating matching scores for ordinal characters, ranges are resolved to individual values. Then all the possible values for the character state for the taxon are scored for a match against the user-specified value and the best matching score is the one used.

For each possible value of the character state for the taxon, the rank of that value is derived – call it *taxonValueRank*. Call the rank of the user-selected value *userValueRank*. The *latitude* specified by the kb developer is expresses the limit of values *that must score*. Therefore one is added to the latitude value so that it works in the same way as latitude expressed for numeric characters (i.e. the first value *that doesn't score*) – call this value *latitude*. Now these values are simply passed to the same scoring algorithm that assesses numeric matches as previously described, so that:

- *taxonValueRank* is equivalent to *taxonRange* (with upper and lower bounds the same);
- *userValueRank* is equivalent to *userValue; and*
- *latitude* is what it is*.*

For circular ordinals the scoring is carried out a second time, first changing the value of *taxonValueRank* by either subtracting or adding the number of all possible values (i.e. the value of the highest rank), whichever brings the value of *taxonValueRank* and *userValueRank* closer together. If this scores better than the unmodified *taxonValueRank* then this score is used instead.

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;

5 Character matching between taxa

One visualisation – 'Side by side comparison' – facilitates the comparison of knowledge-base values between taxa. The 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').

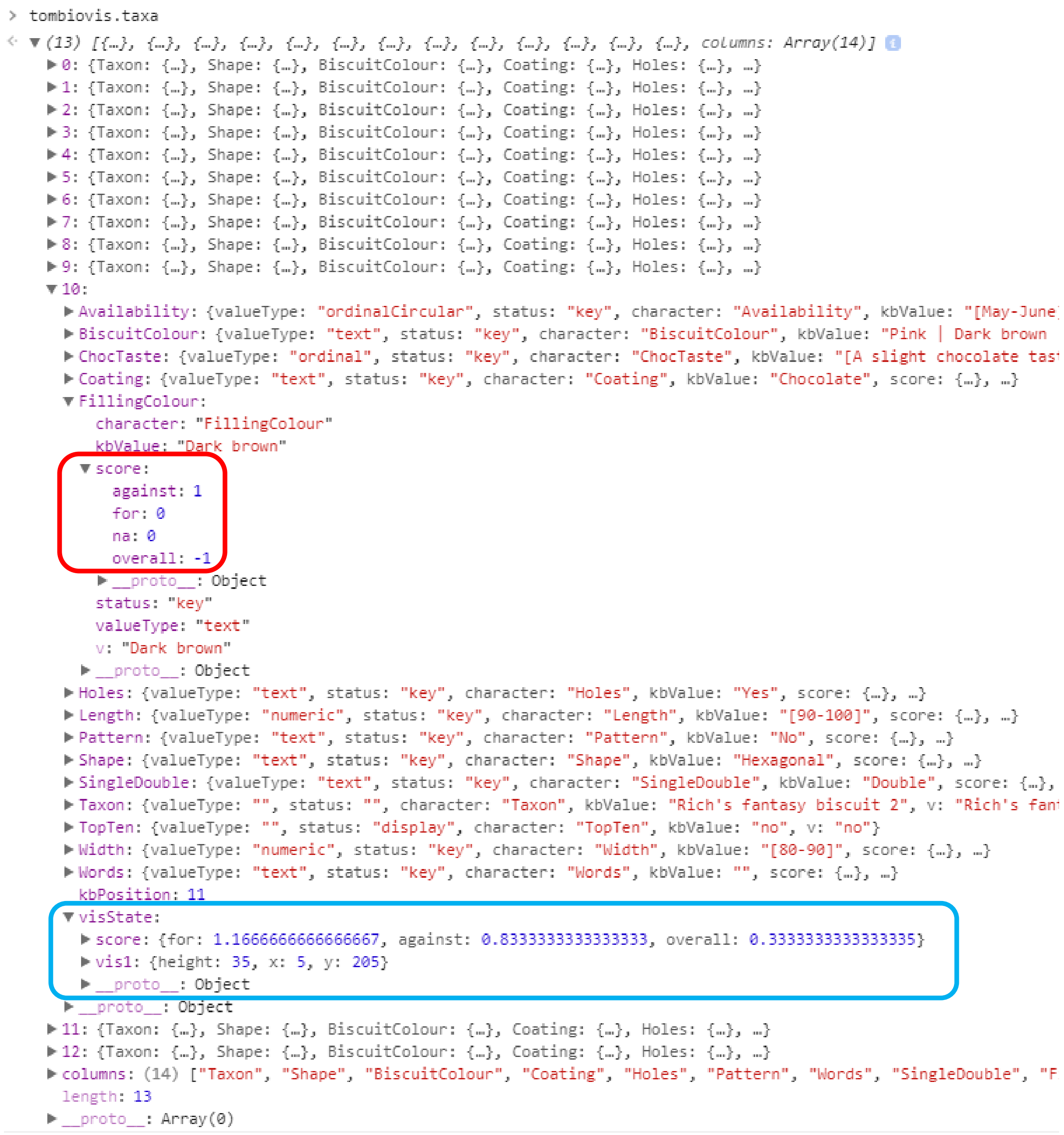
Note that this is quite different to matching a user input value against the character states for a taxon (see section 'Text character matching') where any intersection at all leads to a perfect match.

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.



```
> tombiovis.taxa
< ▼(13) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, columns: Array(14)]
   ▶0: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶1: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶2: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶3: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶4: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶5: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶6: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶7: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶8: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶9: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▼10:
     ▶Availability: {valueType: "ordinalCircular", status: "key", character: "Availability", kbValue: "[May-June
     ▶BiscuitColour: {valueType: "text", status: "key", character: "BiscuitColour", kbValue: "Pink | Dark brown
     ▶ChocTaste: {valueType: "ordinal", status: "key", character: "ChocTaste", kbValue: "[A slight chocolate tas
     ▶Coating: {valueType: "text", status: "key", character: "Coating", kbValue: "Chocolate", score: {…}, …}
     ▼FillingColour:
        character: "FillingColour"
        kbValue: "Dark brown"
       ▼score:
          against: 1
          for: 0
          na: 0
          overall: -1
         ▶__proto__: Object
        status: "key"
        valueType: "text"
        v: "Dark brown"
       ▶__proto__: Object
     ▶Holes: {valueType: "text", status: "key", character: "Holes", kbValue: "Yes", score: {…}, …}
     ▶Length: {valueType: "numeric", status: "key", character: "Length", kbValue: "[90-100]", score: {…}, …}
     ▶Pattern: {valueType: "text", status: "key", character: "Pattern", kbValue: "No", score: {…}, …}
     ▶Shape: {valueType: "text", status: "key", character: "Shape", kbValue: "Hexagonal", score: {…}, …}
     ▶SingleDouble: {valueType: "text", status: "key", character: "SingleDouble", kbValue: "Double", score: {…},
     ▶Taxon: {valueType: "", status: "", character: "Taxon", kbValue: "Rich's fantasy biscuit 2", v: "Rich's fan
     ▶TopTen: {valueType: "", status: "display", character: "TopTen", kbValue: "no", v: "no"}
     ▶Width: {valueType: "numeric", status: "key", character: "Width", kbValue: "[80-90]", score: {…}, …}
     ▶Words: {valueType: "text", status: "key", character: "Words", kbValue: "", score: {…}, …}
      kbPosition: 11
     ▼visState:
       ▶score: {for: 1.1666666666666667, against: 0.8333333333333333, overall: 0.3333333333333335}
       ▶vis1: {height: 35, x: 5, y: 205}
       ▶__proto__: Object
     ▶__proto__: Object
   ▶11: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶12: {Taxon: {…}, Shape: {…}, BiscuitColour: {…}, Coating: {…}, Holes: {…}, …}
   ▶columns: (14) ["Taxon", "Shape", "BiscuitColour", "Coating", "Holes", "Pattern", "Words", "SingleDouble", "F
    length: 13
   ▶__proto__: Array(0)
```