Classification of Arbitrary Data in Existing Databases

Steven Callahan

This application of program synthesis attempts to find patterns in existing databases to make the addition of further entries, particularly those with a different format than the original, automatic.

PRE-BUILT TYPES

Certain entries in databases have a very high frequency of appearing, so classification would greatly improve by specifying these specific categories of data instead of leaving it up to the generic typing to catch. Some of these types are as follows:

First Names
Last Names

For these entries, each entry in a column is compared to a list of common names.
If most entries in the column meet a variable similarity score, based on levenshtein distance to account for typos, the respective label will be applied.

More types can be easily added as necessary to further improve classification.

Specifying common types in databases to improve classification

ARBITRARY INPUT

Input can be shuffled in any order but is assumed to have a common delimiter, such as a space or comma, which can split it into *m* elements.
Further cleaning is also done to remove additional punctuation and blank space..

How arbitrary is the input?

THE PROBLEM

Given a database of *n* rows, each with *m* columns, how arbitrary can we make future entries that will still be entered into the database correctly?

What problem is being addressed?

CATEGORIES

To aid in classification, we use a “category” data structure, which is has the following attributes:

Items: Represents a column of data in the database.
Label: Represents the classification the synthesizer has given to the data. This could be a generic or pre-built type.

Each database is parsed into *m* categories, and it is assumed that each future entry will have *m* elements such that every category will have an item added for each new entry into the database.

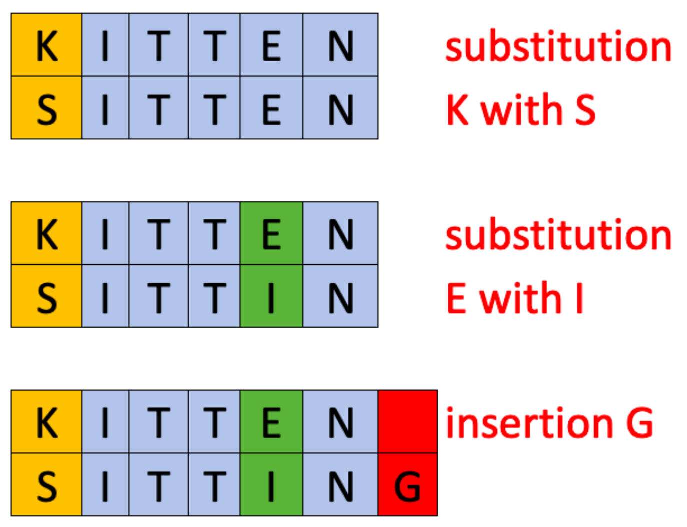
The data structure we use to manage the database

GENERIC STRINGS

To classify strings that don’t meet the criteria of a pre-built type, we loop through every item in every generic string category and place the new string in the category that contains the string with the highest similarity score.

How to classify string types that don’t fit other criteria

LEVENSHTEIN DISTANCE



A metric used to derive a similarity index between strings. This example has a levenshtein distance of 3.

SIMILARITY SCORE

Similarity(s1, s2) = 1 – (levenshtein(s1, s2) / (len(s1) + len(s2))

Similarity measures the levenshtein distance relative to the length of the two strings, so longer strings (which are more likely to have typos) will still match well.

How the similarity between strings is compared

GENERIC TYPES - NUMBERS

To classify numbers that don’t meet the criteria of a pre-built type, we model the numbers in a column as a distribution and assign the new number to the category in which it has the lowest standard deviation.

How to classify numeric types that don’t fit other criteria