Andre Holder

HW 15 – Checking code similarities.

My code is designed to check similarities between 8 codes (from deid HW assignment) using two string similarity metrics (Jaro similarity and Absolute Levenshtein distance) and a Universal Sentence Similarity metric.

The string metrics were used to test how similar the raw text output is for 8 codes (converted from text files to strings, tabs substitute for new lines). The Jaro distance is a normalized value such that 0 equates to no **similarity** and 1 is an exact match. The Levenshtein distance between two words (or strings) is the minimum number of **single-character edits** (i.e. insertions, deletions or substitutions) required to change one word into the other. While these metrics are good measures of how similar literal text could be between two strings, it does not account for the similarity in the *meaning* behind them.

In addition to noting similarity in characters, the Universal Sentence Encoders (such as that used in Tensorflow) also creates a context in which two sentences with the similar meaning are judged to be “similar”. Because computer code can exhibit a similar behavior, I created a framework in which the intent of the code is similar. I chose to ignore the code comments as this was the most likely to have semantic variation, and contributed little to code functionality. After review of the code content for each, I broke the code down into seven major tasks: (1) Describe the identifier (since everyone chose different types of identifiers); (2) Name the compiler; (3) Find the iterations throughout each string (most commonly done using a for loop); (4) Create the string to write to a file; (5) Write the string; (6) Open the output file; and (7) Export/print the string values to the output file. I also included the different variable names to ensure that this variation was not a factor in determining code similarity. The differences were then visualized using a semantic textual similarity heat map.