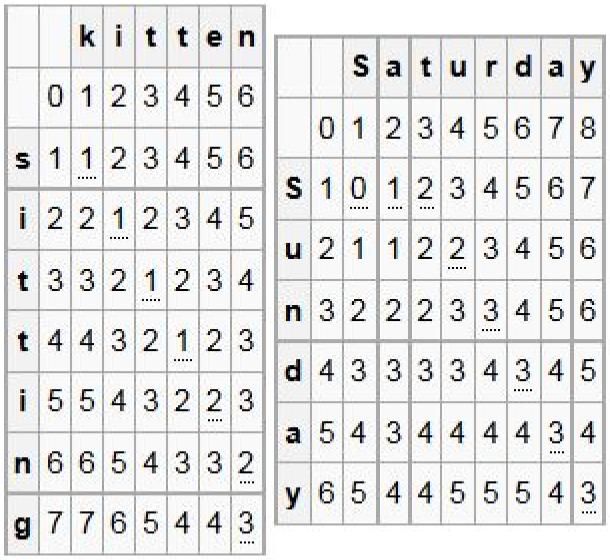
HW 5: Reflection

Maor Bernstein
Gabrielle Ariana Ewall

March 3, 2014

Project Overview Our goal was to compute the degree of similarity between two languages. To do this, we compared English words to their translations using a translation dictionary. We used levenshtein distance as a measure of string similarity. A levenshtein algorithm finds the minimum number of edits (i.e. insertions, deletions, or substitutions) necessary to transform one string into another string. The number of edits is the levenshtein distance between those two strings. We normalized levenshtein distance to word length (divide the distance by the longer word length) and then computed the average levenstein distance per word.
In the future this algorithm could be used with multiple languages compared with English to assess relative distances from English, which would be a more interesting measurement.

Implementation We used a German-English dictionary from Project Gutenberg. We parsed the dictionary into a list, wherein each element is a list of strings (i.e. the word being defined followed by its translations). The first string in each nested list is compared to each consecutive string in the nested list, and levenshtein distances are computed.
The dictionary was parsed using the regular expression method split to divide lines and words. The levenshtein distance function takes two strings as input and loops through each character of the shorter string and then through each character of the longer string and calculates the edit distance necessary to transform the shorter string up to that character into the longer string up to that character based on the previously computed edit distances. It does this by checking the edit distance when substitution, insertion, and deletion are respectively used and using the minimum of these. In essence, this code creates a levenshtein table (an example of which is shown below), storing in memory only the current and prior row (these are the only ones necessary to compute the edit distance of the current comparison. The final value calculated is the levenshtein distance. For a longer explanation of a the tabular method of computing levenshtein distance, see the Wikipedia article on Levenstein Distance, from which this picture was taken.



		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Figure 1: These tables can be used to visualize the calculation of Levenshtein Distance. In both situations the levenshtein distance is 3.

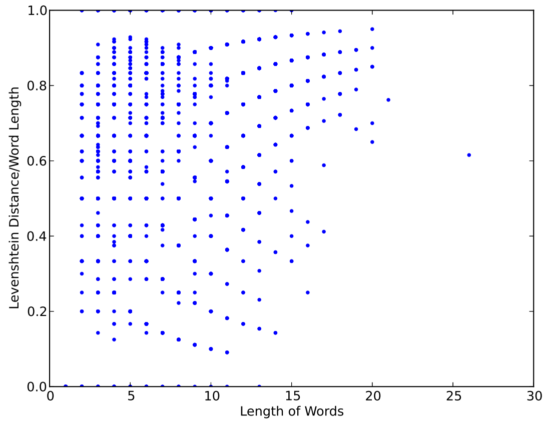


Figure 2: This shows how the different length of the German word effects it's similarity to English words

Results Our final results are clearly represented by matplotlib plot shown above. AS we can see, the weighted levenshtein distance bifurcates with length of the word. What this means is that as our word gets longer and longer, it either means that it is very likely to be very similar to an English word, or very dissimilar to an English word.

Reflection We underestimated the time we would need to spend parsing the dictionaries. We did not realize that text editors leave many artifacts which need to be removed during parsing. Removing these articles took a great deal of our time. In the future we recognize that we should allocate twice at least twice the amount of time we think that a task will take in order to avoid any last minute crush. We also learned about the value of creating new branches in github.