**1**.

-------**EXPERIMENT 1**-------

tank: 0.75

plant: 0.88

perplace: 0.815

smsspam: 0.3471223021582734

-------**EXPERIMENT 2**-------

tank: 0.475

plant: 0.8275

perplace: 0.52

smsspam: 0.3147482014388489

-------**EXPERIMENT 3**-------

tank: 0.805

plant: 0.89

perplace: 0.8475

smsspam: 0.3471223021582734

-------**EXPERIMENT 4**-------

tank: 0.775

plant: 0.8475

perplace: 0.7475

smsspam: 0.3471223021582734

-------**EXPERIMENT 5**-------

tank: 0.76

plant: 0.91

perplace: 0.85

smsspam: 0.3471223021582734

-------**EXPERIMENT 6**-------

tank: 0.805

plant: 0.885

perplace: 0.82

smsspam: 0.3471223021582734

-------**EXPERIMENT 7**-------

tank: 0.8875

plant: 0.88

perplace: 0.8075

smsspam: 0.9676258992805755

-------**EXPERIMENT 8**-------

tank: 0.825

plant: 0.91

perplace: 0.6425

smsspam: 0.947841726618705

-------**EXPERIMENT 9**-------

tank: 0.8975

plant: 0.9025

perplace: 0.85

smsspam: 0.9676258992805755

-------**EXPERIMENT 10**-------

tank: 0.93

plant: 0.915

perplace: 0.7525

smsspam: 0.9676258992805755

-------**EXPERIMENT 11**-------

tank: 0.9025

plant: 0.9225

perplace: 0.85

smsspam: 0.9676258992805755

-------**EXPERIMENT 12**-------

tank: 0.875

plant: 0.8825

perplace: 0.8525

smsspam: 0.9676258992805755

**2**.

In part 2, I implemented my own weighting scheme (“ertman\_weighting”) inspired from stepped weighting. However, my approach considers more than just the words that are a distance 3 from the target and refines the weights of these ranges. Here are the weights for these words:

Adjacent words: 11

2 away: 10

3 away: 8

4 away: 5

5 away: 1

My thought process behind these values were rooted in the assumption that the closer a word is to the target, the more relevant/impactful it is. I assumed that once one has gotten to a context of 10 words (5 away in each direction), that the rest of the sentence should not be weighted. By tweaking the values (both how many words away and their corresponding weights), I saw my results differ each time; this weighting can be tested over larger data and ultimately be refined to the optimal values.

**BEST MODEL:**

Standard term frequency

Overlap similarity

No removal of stopwords

While testing out models, the first thing that stood out to me was how drastically the smsspam task improved by using overlap similarity instead of cosine. Previously getting results around %35, they jumped up to about 95%. Also, the other categories saw improvement, albeit not nearly as significant. Jaccard and Dice performed identically to cosine. So, from this evidence, I determined that overlap had to be the best choice.

Testing out term frequencies, boolean was by far the worst. Tf\_IDF, which I was surprised to find was not optimal, helped ssmspam noticeably (an increase of about 5-7% in some cases), however it performed worse in all other categories. This downgrade in performance outweighted the better results in smsspam (i.e., more than 7% reduction in the 3 other categories), and so for the sake of being holistic, I concluded standard tf was the best. However, from this, I realized that if I was specifically considered with the smsspam task, I would switch to a TF\_IDF model.

A similar situation occurred when testing whether stopwords should be removed. By removing stopwords, tank and plant saw some slight improvement (around 1-2%). However, smsspam and perplace performed worse and in a larger magnitude (sometimes up to 2% and 8% worse respectively). Again, for the sake of holisticness, I decided then to not remove stopwords. However, if I was not considered with the perplace or smsspam task, I would have removed them undoubtedly.

While not a parameter I could change, in my testing, I also discovered that stemming performed significantly worse than a model that does not stem.

**3**.

The largest difference is the fact that a targeted task has a target term, e.g., an ambiguous word or a named entity. This target term is treated separately from the rest of the model (i.e., it does not carry a weight) and is useful for making improvements to the model. For example, it can be used to scale the weights of the words surrounding it. It can also be used to extract context and special adjacent tokens which is almost certainly going to be useful when performing the task.

Conversely, in a non-targeted task, all the terms are treated equally in the sense that not one is distinguished from the rest initially. This does not allow for a context or a way to weight based off position.

**4**.

The most questionable assumption I made was regarding the weighting of terms in the smsspam task. Since there was no target word like ‘plant’, position weighting was not a viable approach (what would be the ambigious word?). Therefore, I uniformly assigned a weight of 1 to every term, not putting emphasis on position/context. However, I can see how this approach could be flawed. For example, it could be the case that the words at the start of a text message are far more important and thus should be weighted more heavily. This would be because on a text notification, only these first couple words are shown. If you are sending a text spam, you want your victims to be intrigued and click on your message right away as it pops up on their screen. It could also be the case that the words at the end of the text are more important, and that key words common to spam texts (e.g., ‘chat’ ‘site’) should be weighted more heavily.