

General comments

1. The word similarity experiments showed that, as always, reweighting has a huge effect. The correlation for the raw counts is negative, and just running a version of PMI brings them to the mid 50s.
2. The weighting schemes we looked at share a common intuition: each cell value is adjusted based on something about its associated row and column. TF-IDF amplifies values that are column-wise frequent and row-wise unusual. For PMI, the t-test, and expected values calculations, the cell value is adjusted based on what we would expect based on the row and column totals.
3. It's always worth considering the distribution of values in the matrix used as input. The count distributions will be roughly Zipfian (very far from normal). TF-IDF and PMI will create quite normal distributions, but with different means and variances. Positive PMI will reduce all negative values to 0 and so create a skewed distribution. A lot of groups saw the effects of these difference when applying LSA after reweighting. LSA does best when its inputs are normally distributed.
4. TF-IDF works best with sparse matrices. Its IDF term is extremely punishing for words that occur in many documents.
5. PMI over-values infrequent events. This seemed to have real practical consequences in the word similarity evaluation. To address this, consider the contextual discounting method that Turney and Pantel discuss (also covered in the homework).
6. GloVe can't take negative values as inputs. It includes a term $\log M[i,j]$ where $M[i,j]$ is the current cell value.
7. GloVe can be followed by additional manipulations. These might entail a departure from the intuitions guiding the model, but that could be okay. For instance, LSA after GloVe might capture interconnections that GloVe could not.

The word similarity bake-off

1. Solid published numbers on WordSim353 are .60–.70, with the very best results in the mid-.70s. So we were all doing pretty well!
2. It's my understanding that winning team used a version of contextual discounting, perhaps mixed with a kind of Laplacian smoothing. This re-enforces the point that PMI needs to be supplemented with methods for identifying low but misleading values.
3. Here's the very best approach I've found: train a shallow neural network with the two target word vectors as concatenated inputs, a single hidden layer, and the word similarity values as outputs to predict. (Making this work well using the codelab implementation seems to require doing challenging problem 7.) It's sort of cheating to look at the labels like this, but it's still illuminating, and one could consider creating a train/test split to make it more rigorous. My best score this way is .94, and I am sure perfection is attainable in this setting.