Shiva Suri
May 14, 2018
Homework 10 [Extra Credit] Writeup

Semantic Textual Similarity

A. Run Instructions & Prerequisites

Download google's word embeddings (word2vec) here. Make sure the GoogleNews-vectors-negative300.bin file you have downloaded is in the same directory as the code (i.e. main.py).
Make sure you have installed the nltk library and WordNet; I use WordNet to find synonyms and antonyms.
>> conda install nltk
>> nltk.download()
In the pop-up window, select to download the "all" library.

Please be sure to install all libraries included as imports in the top of main.py
Make sure data/ contains en-train.txt, en-val.txt, and en-test.txt as provided.

To run my code, type in command line:
>> python main.py --inputfile [data/en-val.txt | data/en-test.txt]
This will run three functions, which correspond to the baseline and two extensions. The baseline and first extension will yield in two files: pred_simple.txt and pred_ex1.txt, respectively. Each of the two files contains predictions on the sentence pairs provided in the input file. However, the second extension automatically yields in two of its own files, for validation and test sets, and the outputs are in pred_val_ex2.txt and pred_test_ex2.txt.

Please be patient when running the script, as the supervised model in Extension 2 takes O(1.5min.) to run.

To evaluate predictions, run evaluate.py with
>> python evaluate.py --predfile [pred....txt] --goldfile [data/en-val.txt | data/en-test.txt]

Along with my code submission in code.zip, I have provided pred_simple.txt, pred_ex1.txt, pred_val_ex2.txt, and pred_test_ex2.txt for your convenience. These files can also be obtained by running:
>> python main.py --inputfile data/en-test.txt

B. Bag-of-words

For the baseline, we were to implement a bag-of-words representation for each sentence and compute the cosine similarity between these two sentences. To construct a vector for each sentence, I had to make a tf-idf matrix *M* where the rows represented each document (i.e. sentence in this case) and the columns represented each unique word across the two sentences. The value of any M_{ij} was the count of word *j* in sentence *i*. Now, I computed the cosine similarities across the vectors from each row (i.e. sentence). A cosine similarity is always between 0 and 1, so I had to scale this value by 5 to achieve a proper similarity measure. I followed a scaling procedure I found in http://www.aclweb.org/anthology/S17-2001 which normalizes the scaling via: $s_{new} = 5 * \frac{s_{old} - min(s)}{max(s) - min(s)}$. Via this approach, my baseline performed better than the project group's baseline, with a pearson correlation of 0.524188848 on the validation set and 0.669820456317 on the test set.

C. Word2Vec Aggregate Approach

For this extension, I decided to follow an approach outlined in Cer et al.'s paper on Semantic Textual Similarity (STS). Namely, I used Google's word2vec word embeddings to help vectorize each input sentence. Each word embedding in word2vec is an array of size 300, so I constructed a size 300 numpy vector for each sentence and aggregated it with each word's word2vec embedding, as per the paper's methodology. Then, I computed the cosine-similarity across the aggregated vectors for each sentence [both 300x1], and I scaled this cosine-similarity using the same equation listed in the paper: $s_{new} = 5 * \frac{s_{old} - min(s)}{max(s) - min(s)}$. Furthermore, if a word did not exist in Google's word embeddings, I just found a similar word via finding a word in Google's word embeddings with a low edit distance. This word2vec aggregate approach yielded in very strong results with a pearson correlation of 0.695343152 on the validation set and 0.722979906 on the test set. Again, this score surpasses the project group's improved performance pearson correlation of 0.69 on the test set.

D. Supervised Models — WordNet Synonyms/Antonyms & Word2Vec

For this enhancement, I tested several different combinations of models and features, and ultimately arrived at an SVM Support Vector Regression with features being the synonym count across the two sentences, the antonym count across both sentences, and the word2vec aggregate cosine similarity detailed in (C). I found inspiration from Maharjan et al. to incorporate synonym and antonym counts across sentences found from using nltk's WordNet as features for my model. At first, I tried using models such as Perceptron and Logistic Regression; however, this was naive as they were unable to predict float values. They ended up predicting integers, and

yielding in very poor results. Thus, I transitioned into regressor algorithms to solve this issue. Namely, I tried SVM along with several linear models, including: linear regression, bayesian ridge, SGD Regressor, TheilSen Regressor, Lasso Lars, and ARD Regression, and I started acquiring much better predictions. I achieved phenomenal results with SVM along with the three features of synonym counts, antonym counts, and word2vec aggregated sentence-vectors' cosine similarities: a pearson correlation of 0.716679029 on the validation set and 0.75767446 on the test set!

E. Results & Conclusions

Model	Validation Performance	Test Performance
Bag-of-Words, Scaled Cosine Similarity	0.524188848	0.669820456
Word2Vec Aggregate, Scaled Cosine Similarity	0.695343152	0.722979906
SVM with Features: Syn. Count, Ant. Count	0.526978276	0.399155292
SVM with Features: W2V Cosine Similarity	0.704301969	0.733123938
SVM with Features: Syn. Count, Ant. Count, Cosine Similarity	0.712647116	0.747268463
SVM with Features: Syn. Count, Ant. Count, W2V Cosine Similarity, BOW Cosine Similarity	**0.716679029**	**0.75767446**
SVM with Features: Syn. Count, Ant. Count, W2V Cosine Similarity, BOW Cosine Similarity, Sent. Len. Difference	0.713594237	0.719152873
Linear Regression (best combination of features)	0.354700731	0.208772586
Bayesian Ridge (best combination of features)	0.705612608	0.737395333
SGD Regressor (best combination of features)	0.691482535	0.700656816
TheilSen Regressor (best combination of features)	0.698474376	0.726241123

Overall, I found that aggregating Google's word2vec embeddings to vectorize sentences was key in my success of achieving high pearson correlations. A supervised approach that combined the cosine similarities obtained from vectorized sentences with synonym and antonym counts as features for my SVM model yielded in results better than an unsupervised method, or a supervised model with fewer features than the cosine similarities and synonym/antonym counts.

Adding features did not always help, as seen when I added features that included: checking for capitalization and sentence length difference in characters and/or words.