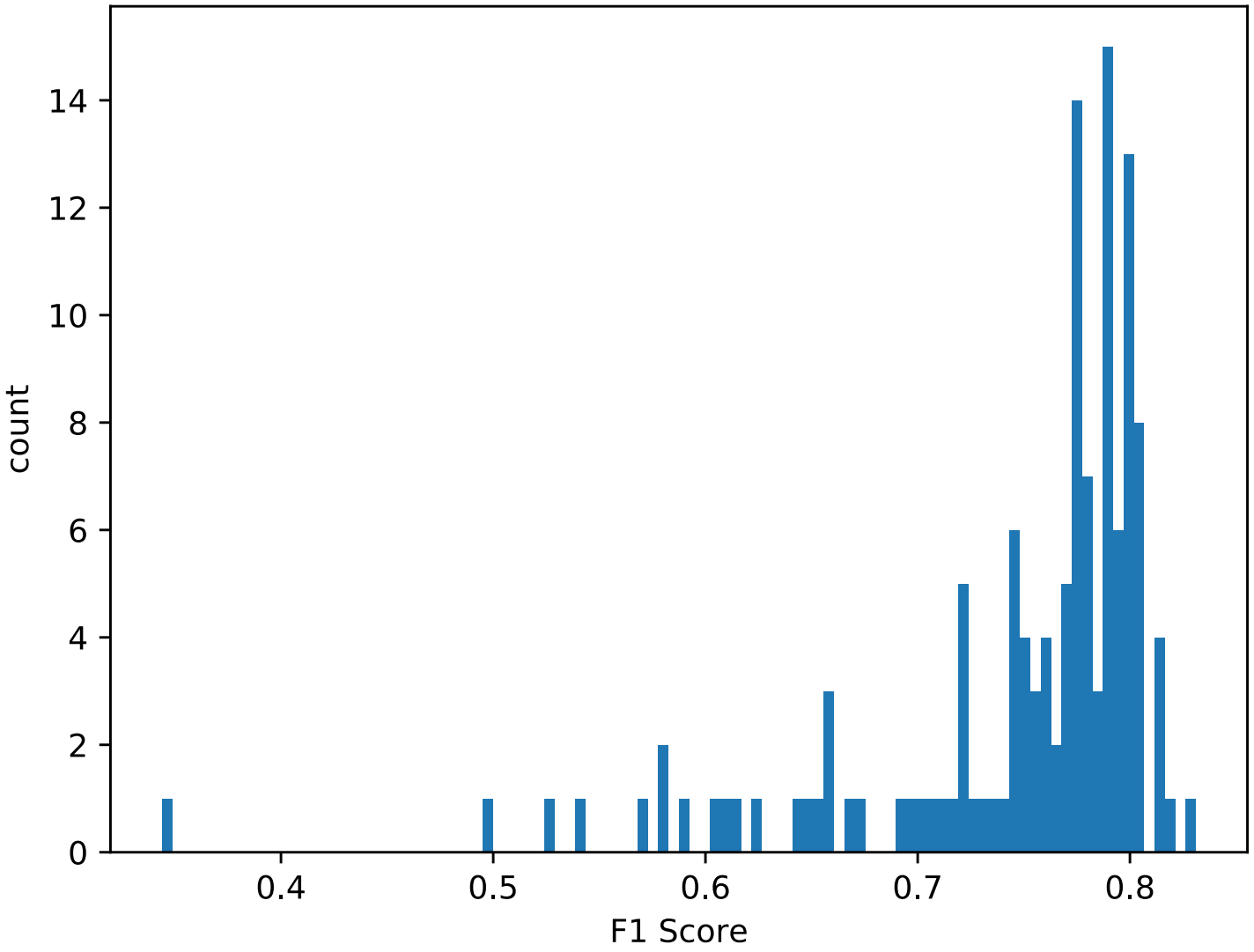
Bake-off 2: Stanford Sentiment Treebank

Results Discussion

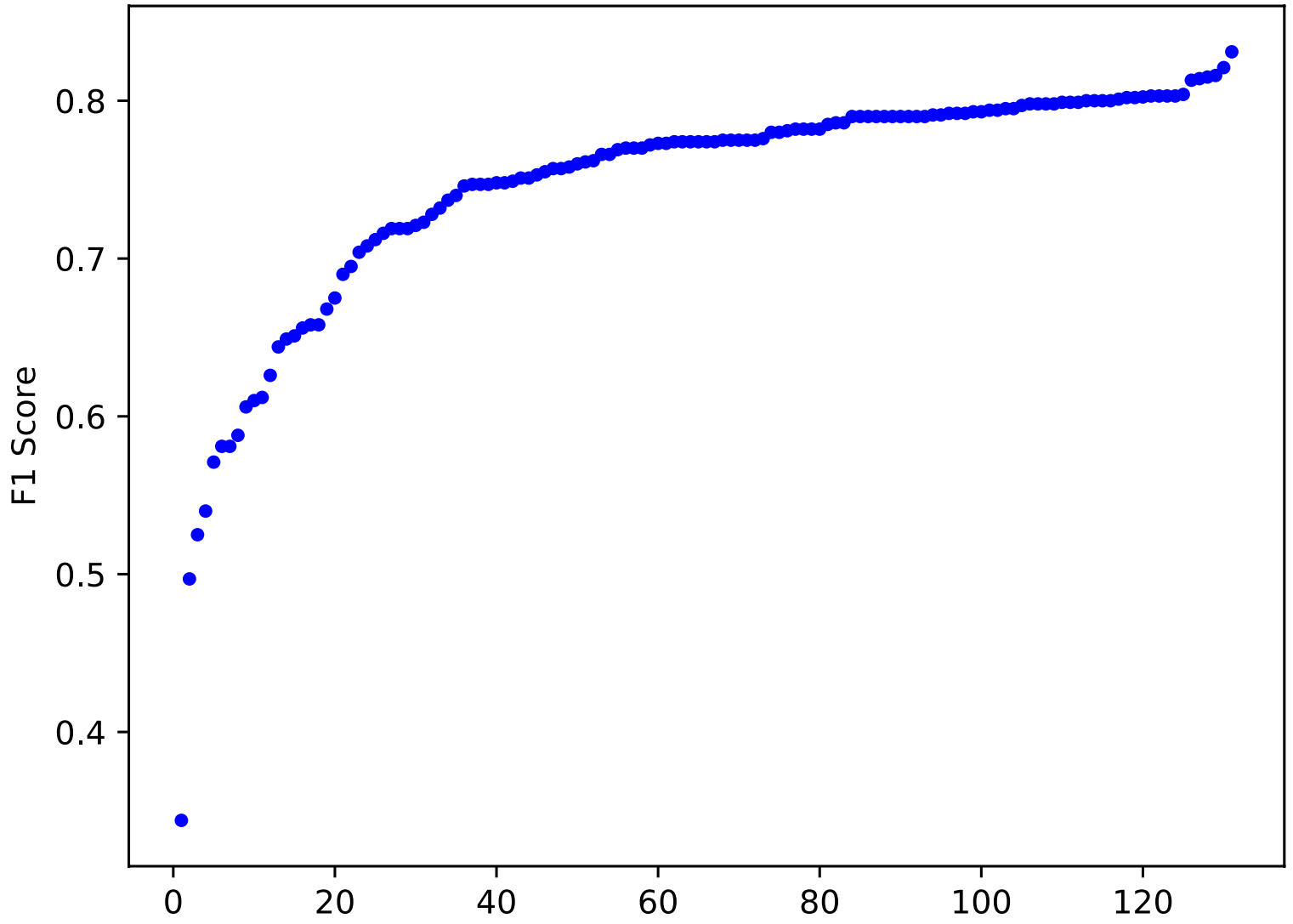
 or  ?

CS224U
Xin Zhou

Histogram of F1 Scores



Scatter Plot of F1 Scores



Top 3

F1 Score: 0.831

(jayadev,keshav2)

First, we removed all punctuation. Then we looked at all character n-grams (with n ranging from 3 to 8) as features, applied TFIDF weighting with sublinear TF weight. We used logistic regression with a balanced class weight (to penalize misclassifications differently based on class weights) and changed the regularization parameter (default in sklearn is C=1, we made it C=1.5 where C is the inverse of the usual lambda).

We don't know why many of these things worked, but they just did.

Top 3

F1 Score: 0.821

(lucy3, jmendels)

Our model used hand-built features. We first converted all words to lowercase, removed stopwords (using NLTK default English stopword list), and extracted unigram and bigram features. We accounted for negation in unigrams by creating 'NOT_X' tokens (where X is some word that follows an odd number of negation terms, such as "not", "no", or "neither"). Then, we added two additional sentiment features based on Bing Liu's opinion lexicon: the proportion of leaves that were positive words, and the proportion that were negative words. We also accounted for negation in creating these features. We also added features to account for part-of-speech (each feature was the proportion of words that were of a certain POS category, as tagged by the NLTK tagger). Then, we added a feature for sentence length. Finally, we averaged each word's 300d GloVe vector representation in order to get a GloVe-based representation for each tree. We concatenated these dense GloVe-based vectors with our feature vectors, and classified sentiment with the Logistic Regression classifier.

Top 3

F1 Score: 0.816

(wolmsted, inaccare)

We used a bidirectional RNN combined with 300-dimensional glove vectors. We also used the relu activation function. We ran it for 500 iterations with a batch size of 1024 and it took about 2 hours to train.

Preprocessing and feature selection

1. Remove punctuation, stopwords
2. Positive/Negative words
3. 'NOT_X' tokens (not, no, neither, …)
4. Intensifiers (very, so , really, …)/Downtoners (kind of, slightly, nearly)
5. N-grams
6. Part-of-speech (POS) tagging
7. Glove embeddings
8. TF-IDF weighted word frequency
9. $NUM token for numerical values
10. PCA
11. Tree parameters: number of leaves, heights, and etc
12. Retrofit

…

Commonly used classifiers

1. Logistic regression
2. Neural Network
3. MultinomialNB
4. Random forest
5. SVM

Other interesting features and models

(tucker)

My best performing model was a word-level Convolutional Neural Network, based on http://www.aclweb.org/anthology/D14-1181. The model uses GloVe embeddings for the words in a sentence, and convolves a number of "filters" over the word vectors of the sentence. A max pooling operation is performed on the filter outputs over the sentence length. This produces a fixed-sized sentence representation, which is fed to a softmax classifier. I found that normalizing the case of the words and using dropout before the softmax layer improved performance.

(avanloon,jtsheng, sastew)

First we uploaded dictionaries from two social psychological traditions based off similar work in survey research in the 1950s. The first, Affect Control Theory, measures the meaning of a word by its evaluation (how good/bad it is), power (how strong/weak it is), and its activity (how active/passive it is) all in the range (-3.5, 3.5). The second measures the same dimensions, but the scores are binarized. The first has more rich information, while the second has more publicly available data. We use the values of both as features (as well as first-order interactions). The problem with these dictionaries, of course, is that their vocabularies are highly limited. To augment the dictionaries, we use glove representations of words in the dictionaries and the words in the corpus. Specifically, if a word in the corpus has a glove representation in glove.6B.50d.txt but not in the dictionaries described above, we looked for the "nearest neighbor" in glove-space (via euclidean distance) which did exist in the dictionaries described above, allowing us to approximate these values for words not in the dictionary. Importantly, we also measured the "glove-distance" between this substitute word and the word that actually appeared in the corpus and weighted the features via this propagation method by the add-one-inverse of this distance (1/(1+dist)). We also included unigrams as features. We used a Bernoulli Naive Bayes classifier, which performed better than random forest with 10 trees (F1 score = 0.717) or logistic regression (F1 score = .787). We also tried including the sum of glove vectors into the model, but this decreased performance (F1 score = 0.792). Bigrams also tended to decrease performance. This is only a modest increase from just unigrams with Bernoulli NB (F1 score = 0.794), but I think it's telling that in this model bigrams and glove-vector sums both decreased performance.

Other interesting features and models

(From Professor Christopher Potts' suggestion)

Took a look at https://github.com/erickrf/treernn and played around with the code to get it running on python3/tf1.7, with no previous tf experience (https://github.com/bsparkes/treernn). Then, modified the code to return precision, recall, and f1 scores and then further to allow for pre-trained vectors. Played around with a few from glove, and also fasttext and lexvec. Fairly consistent f1 scores around 75 with pre-trained vectors and 78 for default vectors (with embedding sizes of 30/50/100). Training used 'extreme' trees with sentiment either 0 or 4, with 350 training trees, 50 dev trees, and 50 eval trees for each sentiment (default values from the repo). Number of epochs was always 30. Not quite by the rules (sorry), but hope is fine anyway. Final f1 score is from 350 test, 50 dev, 100 train for each sentiment with the crawl-300d-2 vectors from https://fasttext.cc/docs/en/english-vectors.html.

Thanks~