Richa Gode

rg3006

**SLP Homework 3 Report**

1. **Feature description:**

Feature Set 1: Individual Parts of Speech and Trigrams

For this feature set, I used the cleaned text from the removal of the dysfluency text features to generate part of speech sequences using NLTK. Using these part of speech sequences, I initially constructed unigrams, bigrams, and trigrams using the n-grams function (and padded with START and STOP values) supported by NLTK. The features for this set include:

* individual POS counts per utterance
* Universal n-gram (and upon refinement, trigram) binary features

My motivation for selecting these features came from the J&M readings as well as the personality type classification paper that we read last week. Both texts suggest that extracting n-grams and then using them as features as part of a classification task are essential. I was also motivated by the fact that n-grams are a useful way to simplify the problem of estimating the language model from data, and since we were given such a plethora of data, this would fit the job well. Also in terms of background knowledge and skillset, I was taught how to generate n-grams and run a classification task in Natural Language Processing class, so I felt comfortable with extracting this particular feature set.

My methodology for building this feature set was the following: 1) generate the POS tags for each utterance, storing every single part of speech in a master set to do POS counts. 2) hen, construct n-grams using the n-grams function from NLTK, and store them as keys in a master n-gram dictionary which also stores the seen count. I knew that while there may be a manageable number of unigrams and bigrams, the number of unique trigrams would be massive, so storing the counts would mean that a bounding threshold could be applied later to manage the number of the features in the CSV.

However, upon closer look at the n-grams being generated, I felt that the punctuation in the sentence could be detracting from the proper generation. I also researched online and found that most companies that use trigrams as part of NLP tasks also strip punctuation (Google, Facebook, Amazon). Therefore, I changed my script to strip the cleaned text of punctuation as well, and then generate the POS tags and then the n-grams.

After that, I also decided to get rid of the unigram and bigram counts because I felt that trigrams were able to capture the language model better, and it is recommended to use n-grams of *n=3* or larger for datasets of the scale of the Switchboard Corpus. For each row in the CSV of parsed text, for each trigram in the master dictionary, I constructed a list of binary features—1 if the trigram is in the row’s generated set of trigrams, or 0 if not.

Finally, the POS counts and the trigram binary features were outputted to a CSV with the headers being the individual POS tags from the master set and the individual trigrams from the keys of the dictionary.

Feature Set 2: Dysfluency markers text features

For this feature set, I cleaned the text that was in the text field of the original training/test CSVs. I did this by removing the dysfluencies from the Dysfluency Annotation Stylebook that was cited on Piazza, as in I removed the following features from the text:

* non-sentence elements
* slash units
* restarts

My initial motivation for cleaning the text was that cleaned text is better to parse for parts of speech/parse trees, and in general NTLK and Stanford NLP tools have a tough time handling the dysfluency markers if they’re still in the utterances.

While cleaning the text, I realized that some of these features could be useful for the classifiers to determine dialogue acts; for example if a conversation has a lot of restarts without repairs or incompletes, it could indicate that it is a conversation that has a lot of fast back and forth, which can tell us something about the dialogue acts that are being used. Also, if the dysfluency was annotated initially, it must serve some role in the corpus.

In terms of methodology, I used a combination of regular expressions and pattern matching to run a script of housekeeping functions on each utterance, and then wrote the counts of when the parsing functions ran to a features CSV.

Specifically, because the non-sentence elements still indicate information about the sentence, I removed the element by parsing it out of the beginning of the utterance, but kept a count of which ones appeared (so for A, C, D, E, F, a count for each per utterance). I also kept a binary feature of if the utterance was marked incomplete, complete, or if it was marked by hashtags, which meant that the utterance was overlapping another in the conversation (i.e. if one person was talking over somebody else). I also attempted to parse out the restarts, and kept a count of which type of restart was seen in the utterance. Then, after parsing the text, I realized that one element was not parsed out—the background noise that was also annotated in the corpus, i.e. <Laughter> or <Thumping>. Not all of the background noises seemed relevant to detecting a dialogue act, i.e. if a baby was crying in the background, so I set a bounding threshold value to limit the number of features that would appear in the CSV.

The output CSV file has the following features: 1) non sentence elements, 2) restart types, 3) slash units (i.e. complete, incomplete, or overlapping), and 4) counts of the background noises for each utterance in the training or test file. I also wrote the parsed text to the output CSV to use it in the next feature set.

Feature Set 3: Vectorization by Unigram Model

For this feature, the initial thought process was to construct a feature set with individual parse trees generated by Stanford NLP. However, upon construction, it seemed impossible to generate these new sets because Stanford NLP takes in each row from the original train/test feature sets one by one, and generating the parse trees take up too much memory.

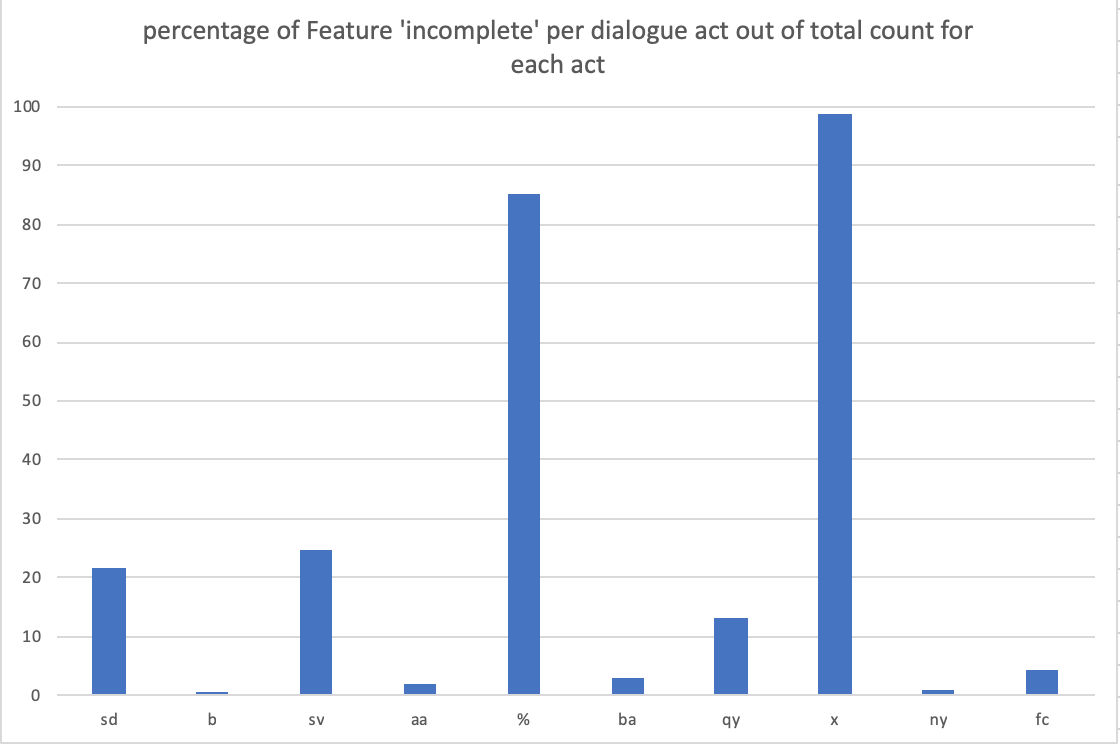
Therefore, based on office hours and Piazza, I chose to construct vectors based on the unigram counts. For each utterance, I add each word to a vocabulary if it hasn’t already been added. Then, I threshold that value because there were too many values to write to a CSV. Then, I construct a vector (a row in a CSV) for each utterance, placing a 1 if the utterance contains a matching word to the thresholded set, or 0 otherwise.

The reason that this feature set is useful is that it can help represent a corpus’s features in a more condensed way; in other words; it is similar to an n-gram model but simplified.

1. **Feature analysis:**

Hypothesis: The dysfluency feature “incomplete”, which captures sentences that are marked as cut off in the middle, will be most frequent for the dialogue act “%”, which signifies abandoned or turn exit.

Dysfluency Plot for the feature:



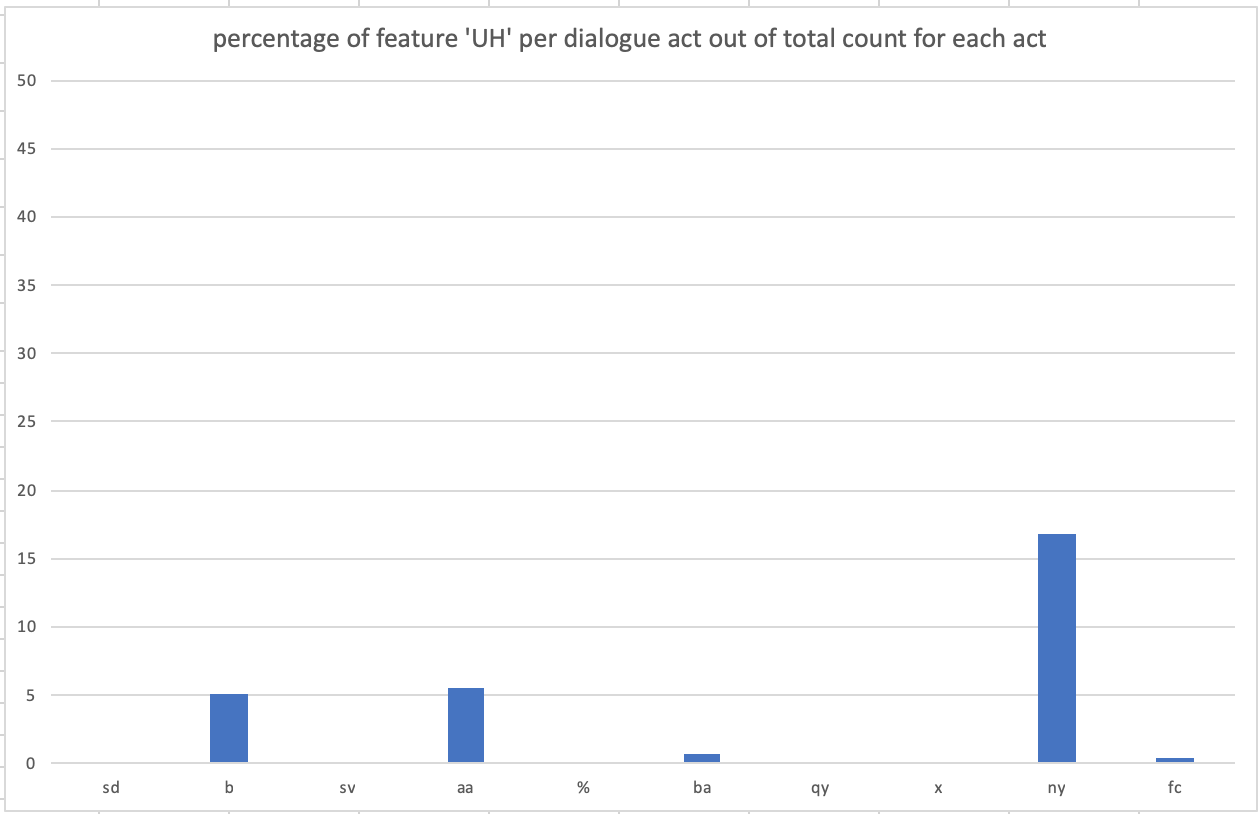
Conclusion:

The ‘incomplete’ feature shows up the most for the ‘x’ dialogue act, which is the non-verbal dialogue act. However, it shows up the second most for the dialogue act ‘%’, which is close to what the hypothesis stated. Because an abandoned dialogue act usually refers to a point in the conversation where somebody leaves a thought behind and either starts a new sentence or is cut off, then it would follow that the transcript would have a ‘-‘ symbol at the end of the sentence, which is how I created the feature incomplete. In terms of it showing up the most for the non-verbal act, ‘x’, I think this is because non-verbal acts were often marked as incomplete sentences because they were simply short sentences with no actual phrases, just background noises.

---------------

Hypothesis: The POS feature “UH”, which captures interjections, will be most frequent for the dialogue act “ny”, which signifies confirmation or yes answers.

POS/Trigram Set 2 Plot for the feature:

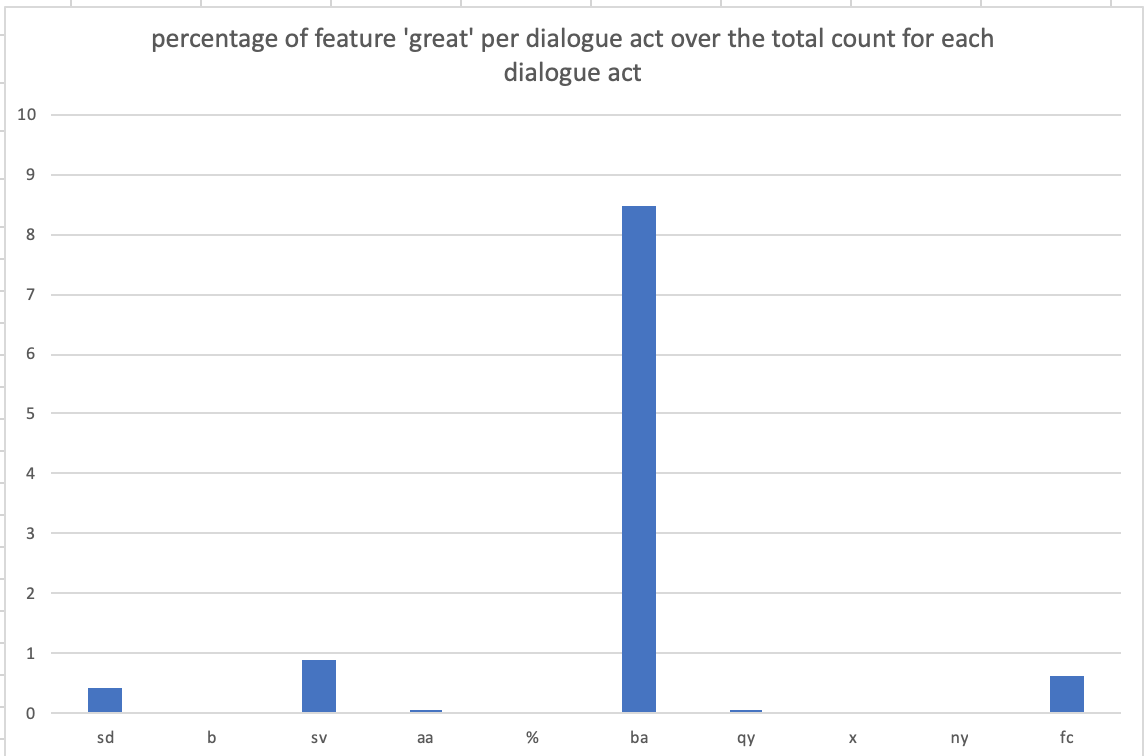


Conclusion:

Out of all of the dialogue acts, ‘UH’ does come up most often for the dialogue act ‘ny’ as a percentage of how many times the act appears in the total dataset. This proves the hypothesis correct, because many times in the set a confirmation answer contains the phrase “Uh-huh” or “Mmhmm”, which are marked by NLTK as the POS tag ‘UH’. I also noticed that it shows up for ‘b’, which is appreciation, which also has similar words in its vocabulary.

Vectorized Unigram Plot for the feature:

Hypothesis: The unigram feature “great”, will be most frequent for the dialogue act “ba”, which signifies appreciation.



Conclusion:

While the percentages themselves are quite low, if we compare them to each other, the feature ‘great’ does show up the highest for dialogue act ‘ba’, which signifies appreciation such as “I can imagine” or “that’s great”.

1. **Classification**

Accuracy over all classes:

LIWC: 76.04% (MLP)

Dysfluency: 54.06% (MLP)

Trigram + POS: 34.91% (RandomForest)

[[1397 0 309 64 0 0 5 396 269 0]

[ 618 0 7 13 0 0 21 75 125 0]

[3487 0 2 0 0 0 2526 89 53 0]

[ 316 2 6 3 0 0 4 187 238 0]

[ 156 0 7 25 0 0 64 113 26 0]

[ 339 0 0 0 0 0 70 3 1 0]

[ 126 8 21 2 0 1 3 282 73 0]

[1430 3 194 155 0 2 31 6822 1650 0]

[ 569 1 80 73 1 3 20 2365 1040 0]

[ 567 0 0 0 0 0 0 2 0 0]]

Vectorized Unigram: 42.30% (MLP)

[[1509 17 210 71 4 0 45 544 36 4]

[ 364 30 304 65 1 0 2 58 35 0]

[1220 2607 2215 42 0 0 6 62 5 0]

[ 443 23 40 26 1 0 7 167 49 0]

[ 206 10 13 0 0 0 2 158 2 0]

[ 81 85 246 1 0 0 0 0 0 0]

[ 60 50 47 5 1 0 37 278 38 0]

[1671 186 775 93 53 8 243 6324 932 2]

[ 593 109 339 56 34 3 138 2486 394 0]

[ 569 0 0 0 0 0 0 0 0 0]]

Dysfluency + (Trigram + POS) + Vectorized Unigram: 34.83% (RandomForest)

LIWC + Dysfluency + (Trigram + POS) + Vectorized Unigram: 71.14% (MLP)

The best classification I got was for the original LIWC features, and I used a RandomForestClassifier from sklearn to learn the class labels. However, I think it will be more pertinent to discuss the combination set of LIWC + Dysfluency + (Trigram + POS) + Vectorized Unigram, which performed with a 71.4% accuracy. For this, I used a Multilayer Perceptron Classifier from sklearn, with a OneVsRest Classifier after that. I originally used just an MLP classifier, but found after reading documentation that using a OneVsRest after that improves the accuracy.

Here is the confusion matrix:

[[1927 12 216 7 0 1 10 229 38 0]

[ 9 245 456 3 0 2 2 130 12 0]

[ 54 245 5800 21 8 1 1 21 2 4]

[ 29 142 34 335 2 1 2 176 35 0]

[ 6 3 86 6 189 0 2 85 14 0]

[ 7 13 391 0 0 2 0 0 0 0]

[ 13 3 3 0 0 0 395 56 45 1]

[ 293 50 41 44 20 0 128 7880 1829 2]

[ 105 56 18 65 13 0 104 2232 1558 1]

[ 0 1 7 5 1 0 0 0 1 554]]

Here is the same feature set with a RandomForestClassifier()’s confusion matrix:

Accuracy: 64.83%

[[1852 8 149 3 0 1 4 310 41 72]

[ 37 185 165 21 0 0 2 149 68 232]

[2395 156 1286 26 6 0 1 436 11 1840]

[ 31 48 53 213 1 0 2 172 186 50]

[ 15 8 30 5 79 0 4 153 33 64]

[ 91 8 127 0 0 0 0 0 0 187]

[ 11 1 0 0 0 0 298 150 56 0]

[ 157 11 6 20 1 0 13 9304 763 12]

[ 65 12 2 24 2 0 5 3025 1015 2]

[ 0 0 1 15 0 0 0 3 0 550]]

1. **Error Analysis**

Overall, the (second) best performing model was the combination of all of the feature sets. We can analyze the diagonal of the confusion matrix as it provides the true positive analysis of the set, aka the counts of all of the times that the act\_tag from the test was equivalent to what was predicted in the set.

* Accuracy for dialogue act 0: 78.87%
* Accuracy for dialogue act 1: 31.81%
* Accuracy for dialogue act 2: 82.22%
* Accuracy for dialogue act 3: 68.93%
* Accuracy for dialogue act 4: 81.18%
* Accuracy for dialogue act 5: 28.57%
* Accuracy for dialogue act 6: 61.35%
* Accuracy for dialogue act 7: 72.90%
* Accuracy for dialogue act 8: 44.08%
* Accuracy for dialogue act 9: 98.57%

Based on the accuracies above, dialogue acts 0, 2, 4, and 9 were the best, which correspond to the following in swda:

**%, b, fc, x, which are abandoned, acknowledgement, conventional closing, and non-verbal respectively.**

I believe these classes were the easiest to predict because the features generated in the sets leaned towards predicting them; as seen in the hypothesis section of report. For example, because in the dysfluency set I generated features for background noises detected, those are often marked as non-verbal or abandoned act\_tags in train\_merged.csv and test\_trained.csv. There are simply more features for picking up ‘x’ or ‘%’ than the other classes, such as the restart and incomplete features that had much higher counts for the ‘x’ or the ‘%’ classes. I also can see that ‘b’ and ‘fc’ are also high in terms of accuracy, however; neither the training nor testing set had a large count of the total fc tags in the first place (as seen on the swda documentation website), so the accuracy needs to be taken with a grain of salt. I think that ‘b’’s accuracy in this case comes from the LIWC + trigram feature sets the most, because this class came up as the most accurate in the confusion matrices for each respectively. Also, there are fewer words in the corpus that express ‘b’, such as “great”, or “amazing”, or “sure”, so their counts are higher and therefore weigh more, which is why the model may be generating a higher accuracy for that label.

Interestingly, because much of the accuracy of this model comes from the addition of LIWC features, which also acts similarly to vector unigrams (bag of words) feature set that I constructed, the model for LIWC + POS/trigram + dysfluency performed at 69%, which is only slightly lower than what this best model runs at.   
  
In terms of difficulty and common errors, I found across the board that it was difficult to predict classes such ‘sd’ and ‘qy’, because there were a) many examples of them in the training and testing sets, and many words in both of their vocabulary, and therefore a wide range of POS + trigram tags which made it harder to predict based on the features generated. Based on the confusion matrix, the classifier got confused on these classes because of the TP/FP/FN/TN ratios.

In the future, to improve this classifier, I would perhaps not make some of the features binary, like the trigrams or the vectorized unigrams, but rather keep counts of them like I did for POS or for the incomplete/restarts/nonsense features in the dysfluency set. I would also, given more time, try to apply more classifiers such as KNN to see if the accuracy improves.

Collaboration:

I spoke with some students who are in my undergraduate major (as I do not personally know many graduate students) about the feature sets (as in what they should be/how to refine them), but did not discuss code. The students are: Myra Deng, Vasudha Rengarajan, Ansh Kothary, and Vinay Ramesh. I am putting this note so as to respect the efforts and the input my classmates provided to me.

Cheers!