Pranav Darbha (pd353)
Kane Tian (kxt3)

Kaggle Group Name: why a guy great until he gotta be great

Screenshots:

Language Model Based -

Naive Bayes -

Implementation Details:

For the Language Model Based Classification portion of the project, we followed the conditional probability formulas for our implementation. First, we generate dictionaries of the training dataset mapping each word (for unigrams) or phrase (for bigrams) to the number of times that word or phrase appears in the training corpus. We did one extra processing step by considering the beginning of the document as a token, so that the first word in the text document would also be considered in the bigram dictionary. We thought this would be better than simply ignoring that token. From these dictionaries, we can then calculate the conditional probability of each word or phrase occurring since we have the number of times each occurs and the total number of words in the training corpus. When calculating these probabilities, we decided to use additive smoothing because using Laplacian smoothing produced worse results. We used a grid search to find the best smoothing hyperparameter when calculating these probabilities to maximize our results. We added a processing step using the interpolation technique learned in class to combine unigram and bigram models. We tuned the weights for the unigram and bigram models using another grid search. We then use these probabilities to calculate the perplexity of each test review, and we compare perplexities obtained from training on the deceptive and truthful training sets. We choose the correct class based on the values for perplexity.

For the Naive Bayes portion of the project, we generated the unigram and bigram dictionaries for each training review. We then created a bag of unigrams and a bag of bigrams and combined these two for the vector representation of each review. We also created a new feature, part-of-speech counts, that went into our vectors using some of the parts-of-speeches obtained from the nltk library. We discarded parts-of-speeches we felt were useless or overpowering the others (e.g. prepositions, while we kept things like adjectives). (We also did experiments during this phase to choose which parts-of-speeches produced the best results. We discarded subsets of prepositions and checked how many validation reviews were correct.) We ran another

experiment to tune the hyperparameter that weights the parts-of-speech tags relative to the bags of n-grams. We did this because we observed much worse results when we weighted them equally, probably because some unimportant words such as prepositions were overpowering the results. We created the training vector representations using the scikit-learn library. Deceptive reviews were labeled with a 1, while truthful reviews were labeled with a 0. Any words that appeared in the validation set that did not appear in the training corpus were discarded, as we felt that smoothing would take care of it. These vector representations and labels were then fed into the Multinomial Naive Bayes model we obtained using scikit-learn. We had another hyperparameter tuning phase to find the best results on the validation set, since the scikit-learn Naive Bayes model has a smoothing hyperparameter. We used this model to make predictions for every line, or review, in the test dataset and reported the better class corresponding to the higher classification probability.

Hyperparameter Tuning:

As was mentioned above, we had to tune our hyperparameters for both the language modelling and the Naive Bayes approaches. The hyperparameters of our language-modelling approach included our smoothing parameter and weights on the unigram and bigram probabilities. To decide on the values assigned to these hyperparameters, we used grid search to pick the combination of parameters that gave us the highest accuracy on the validation set. We tested using values from 0.1 to 1 for the smoothing parameter and 0.1 to 0.9 for the weight on the unigram probability. The 10 highest combined accuracies are shown in the table below.

Smoothing parameter	Unigram weight	Truth accuracy	Deceptive accuracy	Combined accuracy
0.1	0.9	0.601563	0.960938	1.5625
0.1	0.8	0.5625	0.929688	1.492188
0.2	0.9	0.523438	0.96875	1.492188
0.1	0.7	0.5625	0.921875	1.484375
0.1	0.6	0.5625	0.882813	1.445313
0.2	0.8	0.5	0.945313	1.445313
0.1	0.4	0.578125	0.859375	1.4375
0.3	0.9	0.453125	0.984375	1.4375
0.1	0.5	0.570313	0.859375	1.429688
0.1	0.1	0.570313	0.84375	1.414063

From this table we could see that as the unigram weight increased, the accuracy increased so we ran grid search again, this time using values of 0.9 to 0.99 as the weight of unigrams. Our results are shown in the table below.

Unigram weight (with smoothing parameter 0.1)	Combined accuracy (truth + deceptive)
0.91	1.578125
0.92	1.570313
0.93	1.585938
0.94	1.59375
0.95	1.609375
0.96	1.625
0.97	1.671875
0.98	1.679688
0.99	1.71875

From this, we decided to have our smoothing parameter be 0.1 and our weight on the unigram property be 0.99.

The hyperparameters for our Naive Bayes approach included the smoothing parameter and the weight put on part-of-speech tag features. After including part of speech tags our accuracy severely declined, but we kept reducing the weight put on the part-of-speech tags until they improved accuracy. We came to the conclusion that the appropriate weight for the part-of-speech tags was 0.01. To determine the best value of the smoothing parameter, we ran grid search before and after including part-of-speech tags and saw that 0.4 was the value that produced the lowest error rate for both. The results of grid search are found in the table below.

Smoothing parameter	# errors (unigrams and bigrams only)	# errors (including part-of-speech tags)
0.1	19	18
0.2	17	16
0.3	16	16
0.4	15	16

0.5	15	17
0.6	17	19
0.7	16	20
0.8	20	20
0.9	19	19

Motivations for feature choices:

We decided on adding the extra feature of parts-of-speech counts for the Naive Bayes model. We used the nltk library to generate counts of parts-of-speeches in the training corpus, and we added these features to the bag of unigrams and bag of bigrams. The nltk library has functions that count the following parts-of-speeches: CC (coordinating conjunction), CD (cardinal digit), DT (determiner), EX (existential there), FW (foreign word), IN (preposition/subordinating conjunction), JJ (adjective), JJR (comparative adjective), JJS (superlative adjective), LS (list marker), MD (modal), NN (singular noun), NNS (plural noun), NNP (singular proper noun), NNPS (plural proper noun), PDT (predeterminer), POS (possessive ending), PRP (personal pronoun), PRP$ (possessive pronoun), RB (adverb), RBR (comparative adverb), RBS (superlative adverb), RP (particle), TO (‘to’), UH (interjection), VB (verb), VBD (past tense verb), VBG (gerund), VBN (past participle verb), VBP (singular present verb), VBZ (3rd person singular verb), WDT (wh-determiner), WP (wh-pronoun), WP$ (possessive wh-pronoun), WRB (wh-adverb).

We thought that using parts-of-speeches would improve our results because our Naive Bayes would now have many more features to work with. However, we found that our results significantly worsened (produced many more misclassifications) because we had weighted them equally. The addition of this feature caused the distribution of the word counts to change such that our unigrams were weighted more (since we can only do parts-of-speeches tagging with singular words), but it also changed its distribution to become less representative of the corpus. This was problematic, so we decided to scale down the weight of the parts-of-speeches tagging to 1% after experiments on tuning this parameter. We also got rid of some of the parts-of-speeches because we felt they were unnecessary or actually harmful. We ran more experiments trying out different combinations of parts-of-speeches, and we settled on keeping FW, JJ, LS, NNP, PDT, RB, and UH because they produced the best results we got using this feature. (They also seemed like the most logical differentiators between deceptive and truthful reviews, although we realize that this isn’t a reliable method to choose.)

Overall, unfortunately, this feature did not help improve our classification accuracy, despite our varied attempts at tuning the hyperparameters involved. We had trouble with part-of-speech tagging because each word in the bag of unigrams would be counted twice if it was also in the

subset of part-of-speech tags we wanted. However, we did not want to reduce the bag of unigrams because it produced the best results for the Naive Bayes classifier.

Other ideas we had for features were sentiment analysis using nltk and counts for punctuation, but we felt that these would not be great features given our issues with part-of-speech tagging. Sentiment analysis had to involve using a separate Naive Bayes model because it gives back probabilities instead of counts, so we can't use multinomial Naive Bayes. We thought of using a model ensemble with bags of unigrams and bigrams and then a separate sentiment analysis model, but we thought that sentiment was not really useful for detecting if a review was deceptive or legitimate because the datasets were generated by people, and there were similarly negative and positive reviews in each category. We also decided not to use counts for punctuation because we similarly observed excessive usage of punctuation in both deceptive and truthful reviews.

Error Analysis:

We ran a script to detect classification mismatches in the validation set. Each number corresponds to a review # (line).

Truthful (that was classified as deceptive) intersection:
64,39,104,17,114,83,52,84,25,94 - 10
Deceptive (that was classified as truthful) intersection:
6,71 - 2

Unique truthful Naive Bayes:
64,39,104,17,114,83,52,84,120,25,94 - 11
Unique deceptive Naive Bayes:
122,6,71,90,62 - 5

Unique truthful language based:
9,11,12,16,17,21,22,25,39,43,48,50,52,55,57,58,62,64,67,72,83,84,91,94,102,104,114,117,118,121,123,124 - 32
Unique deceptive language based:
26,58,6,71 - 4

<u>Naive Bayes</u>
False positive (truthful that was classified as deceptive) rate = 21/128 = 16.4%
False negative rate = 7/128 = 5.5%

<u>Language Based</u>
False positive (deceptive) rate = 42/128 = 32.8%
False negative rate = 6/128 = 4.7%

An example of a review that the Naive Bayes failed that the Language Based model did not:

- 122: We are not sure why it was a false negative, but it was a very short review, so it is more likely to be similar to many word distributions. This could've caused our Naive Bayes to fail, since it assumes certain distributions to estimate P(y | x).

An example of a review that the Language Based model failed that Naive Bayes did not:

- 9: There were very unique words that appeared in the review, which could've appeared more often in the deceptive corpus because deceptive reviews are probably more emotionally charged.

Details of Programming Library Usage:

We used the basic Math python library, Pandas, Numpy, scikit-learn, and nltk. We used the basic Math python library to handle basic math operations, we used the Pandas library to write our results to csv files, we used Numpy for more efficient array storage, we used scikit-learn's feature extraction and Naive Bayes models to create the input vectors for the Naive Bayes model and to train the Naive Bayes models, and we used nltk to do the part-of-speech tagging.

Description of contributions (Workflow):

We both worked equally on coding and writing the report. We worked together in person while sharing ideas on how to program our solutions, and divided up the report separately based on section. We finished the Language Based Model first and then reused some of the methods to finish the Naive Bayes Model.

Feedback:

The project was just right in terms of difficulty. We spent 3 evenings on it, and we felt we have a much better understanding of n-grams and Naive Bayes applied to language analysis.