Group Members:

- John Le
- Chen Kai Zhang
- Jay Patel
- Brad Byun

Problem 1: Naive Bayes

a.

```
total words in positive = 23        total words in negative = 22

P(great|+) = 0.26                   P(great|-) = 0.09
P(amazing|+) = 0.35                 P(amazing|-) = 0.05
P(epic|+) = 0.20                    P(epic|-) = 0.18
P(boring|+) = 0.09                  P(boring|-) = 0.23
P(terrible|+) = 0.04                P(terrible|-) = 0.18
P(disappointing|+) = 0.04           P(disappointing|-) = 0.27
```

POS = P(+) * P(great|+) * P(amazing|+) * P(terrible|+) * P(disappointing|+)
NEG = P(-) * P(great|-) * P(amazing|-) * P(terrible|-) * P(disappointing|-)

Log Space Probabilities

log(P(w|S = +)) = log(POS/(POS+NEG)) = log(0.44) = −0.357
log(P(w|S = -)) = log(NEG/(POS+NEG)) = log(0.56) = −0.252

Normal Space Probabilites

POS = 0.50 * 0.26 * 0.35 * 0.04 * 0.04 = 0.0000858
NEG = 0.50 * 0.09 * 0.05 * 0.18 * 0.27 = 0.0001094
P(w|S = +) = POS/(POS+NEG) = 0.0000858 / (0.0000858 + 0.0001094) = 0.44
P(w|S = -) = NEG/(POS+NEG) = 0.0001094 / (0.0000858 + 0.0001094) = 0.56

Since P(w|S = -)) > P(w|S = +), we can assume that the model will apply the *NEGATIVE* label on S.

b. add-1 smoothing

```
total words in positive (add1) = 29        total words in negative (add1) = 28

P(great|+) = 0.24                          P(great|-) = 0.11
P(amazing|+) = 0.31                        P(amazing|-) = 0.07
P(epic|+) = 0.21                           P(epic|-) = 0.18
P(boring|+) = 0.10                         P(boring|-) = 0.21
P(terrible|+) = 0.07                       P(terrible|-) = 0.18
P(disappointing|+) = 0.07                  P(disappointing|-) = 0.25
```

POS = P(+) * P(great|+) * P(amazing|+) * P(terrible|+) * P(disappointing|+)
NEG = P(-) * P(great|-) * P(amazing|-) * P(terrible|-) * P(disappointing|-)

Log Space Probabilities

log(P(w|S = +)) = log(POS/(POS+NEG)) = log(0.51) = −0.292
log(P(w|S = -)) = log(POS/(POS+NEG)) = log(0.49) = −0.310

Normal Space Probabilities

POS = 0.50 * 0.24 * 0.31 * 0.07 * 0.07 = 0.0001823
NEG = 0.50 * 0.11 * 0.07 * 0.18 * 0.25 = 0.0001733
P(w|S = +) = POS/(POS+NEG) = 0.0001823 / (0.0001823 + 0.0001733) = 0.51
P(w|S = -) = NEG/(POS+NEG) = 0.0001733 / (0.0001823 + 0.0001733) = 0.49

Since P(w|S = -) < P(w|S = +), we can assume that the model will apply the *POSITIVE* label on S, which is different from if we had not applied *add-1 smoothing*.

c.

We could have `not_word` as one of the features such as `not_disappointing` and `not_great`. This allows the model to notice that a not was place before an adjective and therefore, treat it as a different feature. In the case of the sentence S given, this should improve our positive probability since it will no longer see `disappointing` as a negative feature, but a positive which should increase our accuracy and improve classification.

Problem 2: Programming Hate Speech Detection

2.1 - Naives Bayes

1. Report the accuracy on the training, dev, and test sets.

```
===== Train Accuracy =====
Accuracy: 1353 / 1413 = 0.9575
===== Test Accuracy =====
Accuracy: 191 / 250 = 0.7640
Time for training and test: 3.51 seconds

===== Train Accuracy =====
Accuracy: 1353 / 1413 = 0.9575
===== Dev Accuracy =====
Accuracy: 189 / 250 = 0.7560
Time for training and test: 7.78 seconds
```

2. Look at the output of your model on some randomly selected examples in the dev set. What do you observe? Why do you think the model has predicted the way it has for those specific examples?

Out of the 5 random examples taken from the dev set, it seems the model was only able to accuratly predict 1/3 of them. This prediction is due to the fact that most of the words found wihtin these sentences, had a higher occurance in sentences labeled "negative" (1) from the test set. For example, the word "divide" which was in a postive (0) sentence only appeared in the negativily (1) labeled sentence in the test case, causing the probability that the sentence with this word included had a higher chance of being negative (1) than positive (0).

3. List the 10 words that, under your model, have the highest ratio of P(w|1)/P(w|0) (the most distinctly hatespeech words) and list the 10 words with the lowest ratio. What trends do you see?

The 10 words under my model that are distincly hatespeech words are:

- scum
- sweden
- hate
- different
- ape
- crap
- jew
- filth
- genocide
- Negro

The 10 words opposite are:

- whites
- black
- Jew
- culture
- history
- city
- stormfront
- home
- world
- years

A trend we noticed is that the words that are interpreted as distincly hatespeech, are words that by themselves imply a negative conotation. While those that can be mixed between postive and negative are just neutral words like city and history, which don't really add to the emotions of a sentence.

Logistic Regression

1. Report the accuracy without L2 regularization on the train, dev, and test sets. How does it compare to Naive Bayes?

```
===== Train Accuracy =====
Accuracy: 1412 / 1413 = 0.9993
===== Test Accuracy =====
Accuracy: 190 / 250 = 0.7600
===== Dev Accuracy =====
Accuracy: 173 / 250 = 0.6920
```

Compared to Naive Bayes, the Logistic Regression without L2 regularization did 4.18% better on the training data set, 0.4% worse on the test set, and 6.4% worse on the dev set. I suspect that Logistic Regression without L2 regularization overfits the training data set. This is shown in the 99% train accuracy that is produced and the comparatively lower test and dev accuracies.

2.Add L2 regularization with different weights, such as λ = {0.0001,0.001,0.01,0.1,1,10}. In your writeup, describe what you observed

L2 regularization slightly improved the training data by 0.14% and test data accuracy by 0.4% with, λ = 0.1, and learning rate, α = 0.03. The accuarcies with, λ = 0.1, and learning rate, α = 0.03, showed the best results in our code. Other values of λ, did not have better results than our logistic regression without L2 regularization results. We observed that at higher values of λ like, λ = {10, 1} , the logistic regression numbers tend to overflow and the accuracy of all sets drop significantly; at lower values of λ like, λ = {0.01, 0.001, 0.0001}, the accuracy of all sets drop slightly, but still had fairly high accuracies. We chose α = 0.03 because it had achieved the best results for our dataset compared to lower or higher values of α. The best accuarcy at a different α was at α = 0.01 and λ = 0.001, which had a 99.65% training accuracy and a 75.20% test accuracy. Through this testing, we observed that that α and λ are somewhat related to each other. Since lower values of α enabled a lower λ values, meaning that these new lower α and λ values maintained fairly high accuracy results. We also encountered numeric overflow errors while testing high values of α with λ. However as shown in the results, higher values of λ would lead to more inaccurate classification. So we can expect that higher values of α and or λ will lead to more inaccuarate results and more numeric overflow.

```
λ = 10
===== Train Accuracy =====
Accuracy: 716 / 1413 = 0.5067
===== Test Accuracy =====
Accuracy: 118 / 250 = 0.4720
===== Dev Accuracy =====
Accuracy: 124 / 250 = 0.4960

λ = 1
===== Train Accuracy =====
Accuracy: 806 / 1413 = 0.5704
===== Test Accuracy =====
Accuracy: 132 / 250 = 0.5280
===== Dev Accuracy =====
Accuracy: 134 / 250 = 0.5360

λ = 0.1
===== Train Accuracy =====
Accuracy: 1410 / 1413 = 0.9979
===== Test Accuracy =====
Accuracy: 191 / 250 = 0.7640
===== Dev Accuracy =====
Accuracy: 178 / 250 = 0.7120

λ = 0.01
===== Train Accuracy =====
Accuracy: 1412 / 1413 = 0.9993
===== Test Accuracy =====
Accuracy: 187 / 250 = 0.7480
===== Dev Accuracy =====
Accuracy: 173 / 250 = 0.6920

λ = 0.001
===== Train Accuracy =====
Accuracy: 1412 / 1413 = 0.9993
===== Test Accuracy =====
Accuracy: 189 / 250 = 0.7560
===== Dev Accuracy =====
Accuracy: 173 / 250 = 0.6920

λ = 0.0001
===== Train Accuracy =====
Accuracy: 1412 / 1413 = 0.9993
===== Test Accuracy =====
Accuracy: 190 / 250 = 0.7600
===== Dev Accuracy =====
Accuracy: 172 / 250 = 0.6880
```