Assignment 1: Sentiment Classification

Beto An

January 29, 2023

1 Q1

Perceptron Model with Unigram features: 30 epochs
Average Train Time: 7.4s
Average Training Set Accuracy: 6920/6920 = 1.0
Average Dev Set Accuracy: 661.6/872 = 0.759

2 Q2

Schedule 1: Constant Schedule The dev set accuracy on average is slightly worst, around 0.737. Furthermore, the dev set accuracy has greater variance.

Schedule 2: $\frac{1}{t}$ Learning schedule The dev set accuracy has, on average, higher accuracy, at 0.759. The model is able to attain this accuracy pretty consistently.

3 Q3

The ten words with the highest positive model weights:

1. refreshing
2. hilarious
3. appealing
4. summer
5. powerful
6. prose
7. half-bad
8. solid
9. remarkable

10. sharp

The ten words with the lowest negative model weights

1. stupid
2. failure
3. lousy
4. terrible
5. worst
6. sustain
7. lacking
8. thinks
9. flat
10. product

Here, notice that words with positive weights tend to have a more positive connotation: "I feel great after a **refreshing** drink" or "That cake you baked is quite **appealing**."

On the contrary, words with negative weights tend to have a more negative connotation: "Your ideas are **stupid**," or "The plan is **lacking**."

Therefore, positive weights should correspond to positive "happy" words whereas negative weights should correspond to negative "sad" words. The model weights trend with word connotations.

4 Q4

The training set accuracy is consistently 1.0, but the development set accuracy is significantly less, usually hovering around 0.75.

For the training set accuracy to be 1.0, the perceptron model must've correctly labeled all of the examples, which means the training examples are linearly seaprable (as the perceptron model is linear).

However, the lower accuracy on the dev set suggests that the dev set is not as separable. Similarly, it is also plausible that the perceptron model is overfitting. Another possibility is that the training set is not representative or does not generalize to the validation set. Lowering the number of epochs and regularization may reduce variance and overfitting.

5 Q5

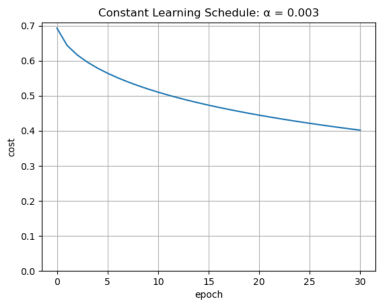
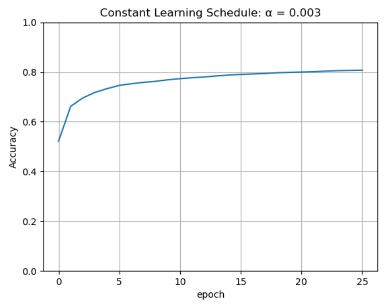
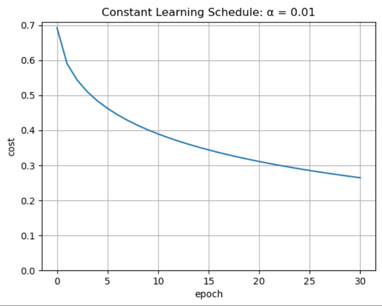
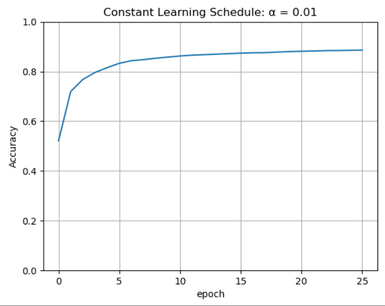
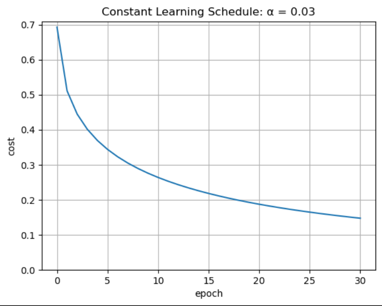
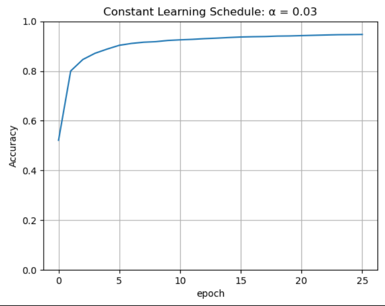
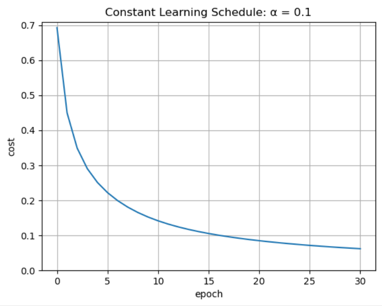
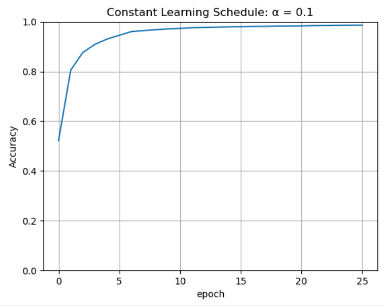
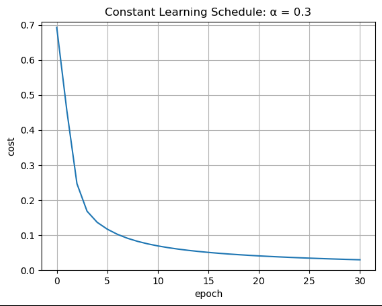
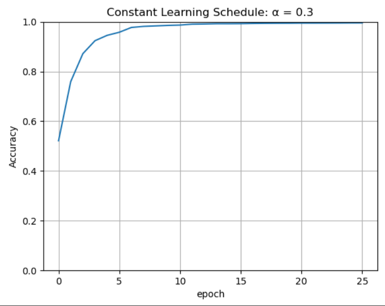
Logistic Regression Model with Unigram features: 30 epochs
Average Train Time: 10.1s
Average Training Set Accuracy: 6895/6920 = 0.996
Average Dev Set Accuracy: 674.4/872 = 0.773

6 Q6

Below are the cost vs. epoch graphs for various learning rates and two different learning schedules.

A coupel of observations I made

- As the learning rate increases, the cost comes down more quickly, and the model converges faster
- Similarly, as the learning rate increases, the accuracy increases much more quickly, and model accuracy converges faster
- At around five epochs, all of the models begin to taper off in progress: cost comes down much slower and accuracy increases only slightly even many epochs after.

Learning Rate	Objective (Log-likelihood Cost)	Accuracy
$\alpha = 0.003$		
$\alpha = 0.01$		
$\alpha = 0.03$		
$\alpha = 0.1$		
$\alpha = 0.3$		

7 Q7

Perceptron Model with Bigram features: 15 epochs
Average Train Time: 45.5s
Average Training Set Accuracy: 6920/6920 = 1.0
Average Dev Set Accuracy: 625.1/872 = 0.717

Logistic Regression Model with Bigram features: 15 epochs
Average Train Time: 51.2s
Average Training Set Accuracy: 6870/6920 = 0.993
Average Dev Set Accuracy: 633/872 = 0.726

8 Q8

Perceptron Model with Better features: 20 epochs
Average Train Time: 4.49s
Average Training Set Accuracy: 6896.9/6920 = 0.997
Average Dev Set Accuracy: 667.8/872 = 0.766

Logistic Regression Model with Better features: 25 epochs
Average Train Time: 8.9s
Average Training Set Accuracy: 6551/6920 = 0.9466
Average Dev Set Accuracy: 692.4/872 = 0.794

9 Q9

I made two modification in my better feature extractor:

1. Ignore common stop words that have neutral connotation
2. Propagate negativity in a sentence
   - In other words, when a negative word is encountered, append "_NOT" to the end of nearby words to denote their negative form. (2x potential number of features)

Encoding negativity, in my opinion, is quite important. The sentence "I am happy" and "I am not happy" have completely opposite sentiments.

A model which does not encode negativity directly into its features cannot predict both accurately: if it performs well on one, it will perform poorly on the other (no free lunch).