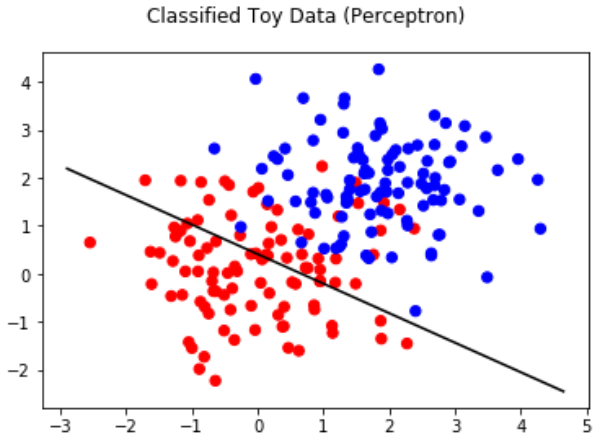
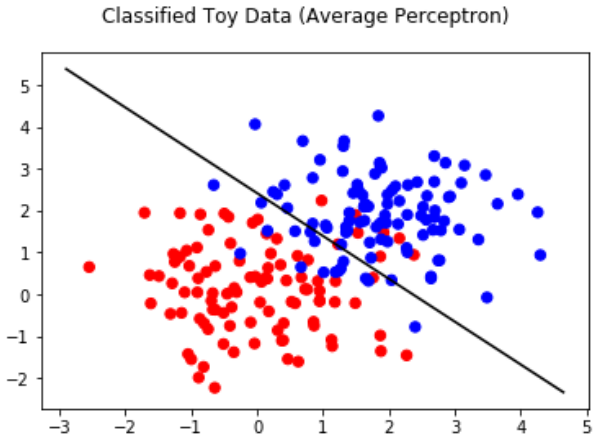
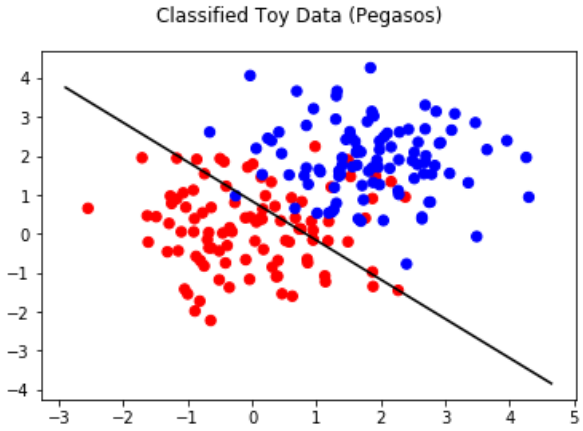
7.

These algorithms each provide a different decision boundary because they function differently in finding the boundary. The regular perceptron iterates until it finds a separation. In this case there is no separation so it gets bounced between a collection of values until it reaches T. The average perceptron is a bit smarter. Realizing that perceptron won't converge, but rather get pushed between values where it should be for a long time, it averages the resulting thetas to get a bit closer to where it should be. Pegasos on the other hand tries to optimize the margins by minimizing the penalty accrued by misclassified points. At the same time it is balancing over-fitting to the training data with a lambda term. This should allow it to be more generalizable for the test cases.







9ab.

Training accuracy for perceptron:	0.9605
Validation accuracy for perceptron:	0.8160
Training accuracy for average perceptron:	0.9760
Validation accuracy for average perceptron:	0.8420
Training accuracy for Pegasos:	0.9155
Validation accuracy for Pegasos:	0.8300

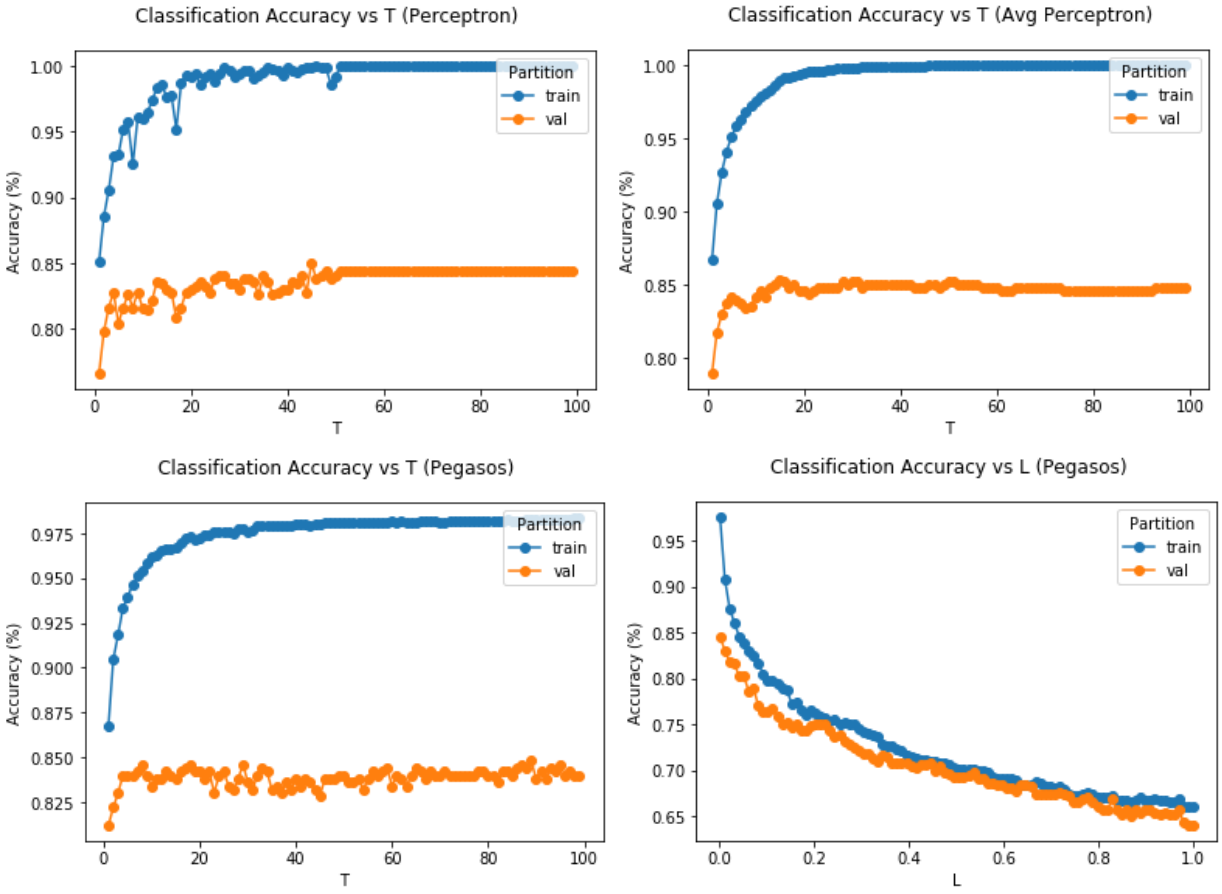
10.

a) The training and validation accuracies behave vaguely similarly as a function of T and as a function of lambda. This is because increasing T allowed the algorithms more time to fit the data which somewhat accurately described the validation set so a better fit meant a better fit for the other. I expected increasing lambda to make the validation set increase in accuracy, while

decreasing the accuracy of the training set. However, this didn't happen. This further supported my hypothesis that the training set was a good representation of the validation set.

b) The algorithm that performed the best of the three on the validation set was Average Perceptron, but not by a significant amount.

c) Optimal T for perceptron was 45. Optimal T for average perceptron was 15. Optimal T for pegasos was 89 and optimal lambda was 0.001.



11.

a. My best function was Average Perceptron, but I still wanted to check how each one did on the test data because one may be good for the validation set but not as good for the test set. However in the real world, we wouldn't know this. The results generated are below.

Best T = 45

Training accuracy for perceptron:	0.9990
Validation accuracy for perceptron:	0.8500
Test accuracy for perceptron:	0.7720

Best T = 15

Training accuracy for Average Perceptron:	0.9890
Validation accuracy for Average Perceptron:	0.8540
Test accuracy for Average Perceptron:	0.7920

Best T = 89 Best L = 0.001

Training accuracy for Pegasos:	0.9830

Validation accuracy for Pegasos:	0.8440
Test accuracy for Pegasos:	0.7820

b. The top ten classified words are reported below using average perceptron. I was surprised that it included & and) as useful when expanded the top 100.

['bland', 'delicious', 'awful', 'itself', 'amazing', 'date', 'toy', 'worst', 'ok', 'threw']

12.

I tried a variety of the suggested changes. Almost none of them seemed to improve classifications. At best, they would improve the ability to fit to the training set, but decrease the ability to fit the validation set. To check if they were truly better, I would need to re run the code that chose the optimum parameters, but I avoided doing this because it was really slow because of the space I searched. I searched from T=1 to 99 checking each integer and L = 0.001 to 1 checking 100 points in the range for L. I used the best values from each algorithm for testing modifications.

- I tried removing punctuation because I thought that they were sometimes unnecessary since almost all reviews had punctuation. However I expanded the range of useful words in part 11 and noticed that ! : and) were among the most useful for classifying. This is likely because happy people would be more ecstatic and use smiley faces, so I included these three punctuation. This didn't help though. This may be because one type of post is more likely to use other types of punctuation that I didn't notice only looking at the top 70 useful words. There were 12688 entries in the dictionary so looking at only the top 70 was not enough to tell which punctuation are not needed.
- I tried removing stop words from the list given without modification. I expected this to have improved classification because some words aren't too useful with classifying things if they appear in almost all reviews good and bad, like the word 'the'. This did not work too well. My hypothesis is that we are including too many useful words in our stop words. Instead of trying to optimize which stop words to use, I tried different methods of improving accuracy.
- I tried only including a word in the dictionary if it appeared in the reviews more than some threshold number of times (two and three tried). This surprisingly decreased accuracy. My hypothesis is that when there is some word that only appeared in one post in the train set, there is a small chance that it would appear in the validation set still and have the same label. Removing it from training would make it slightly harder to classify the review in the validation set.
- I tried including groups of words. For this one I tried it in combination with removed punctuation because I thought they might interrupt useful phrases if they are considered their own word, but it turns out that it worked better with punctuation by a fraction of a percent. The values in the table are done with only spacing punctuation to be their own words. However, the dictionary grew to sizes my computer couldn't handle so I had to also incorporate the previous method where I only included words or phrases if they appeared more than twice in the whole training set. This allowed me to look up to phrases of size two and three. I noticed that this would allow all algorithms to perfectly classify the training set. Because of this, I decided to rerun the hyperparameter fitting for pegasos with this method because it was likely to improve the validation accuracy if I increased the lambda. I did this with pairs of two words because it ran much faster than groups of three and there was actually a slight drop in validation accuracy for three words at a time. However this could have been due to a lambda that was too great.

	Perceptron		Avg Perceptron		Pegasos	
Method tried	**Train**	**Validation**	**Train**	**Validation**	**Train**	**Validation**
Baseline	0.9990	0.8500	0.9890	0.8540	0.9830	0.8440
No Punctuation	0.9982	0.8120	0.9922	0.7960	0.9848	0.8260
No Punctuation, include !:)	1.0000	0.8120	0.9932	0.8160	0.9862	0.8240
Remove stop words	0.9978	0.8200	0.9942	0.8300	0.9775	0.8400
Dictionary threshold = 2	0.9988	0.8320	0.9825	0.8480	0.9802	0.8420
Dictionary threshold = 3	0.9988	0.8280	0.9822	0.8340	0.9785	0.8400
Including two word phrases (+ dict threshold = 2)	1.0000	0.8400	1.0000	0.8580	1.0000	0.8460
Including three word phrases (+ dict threshold = 2)	1.0000	0.8440	1.0000	0.8580	1.0000	0.8420

Raising L to 0.01 and keeping T at 89, was able to increase validation accuracy to 0.8720 at the cost of decreasing training accuracy to 0.9892 using Pegasos when combined with including two word phrases.