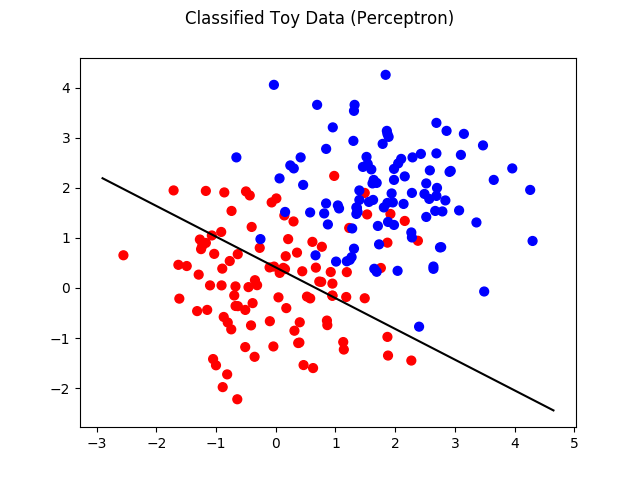
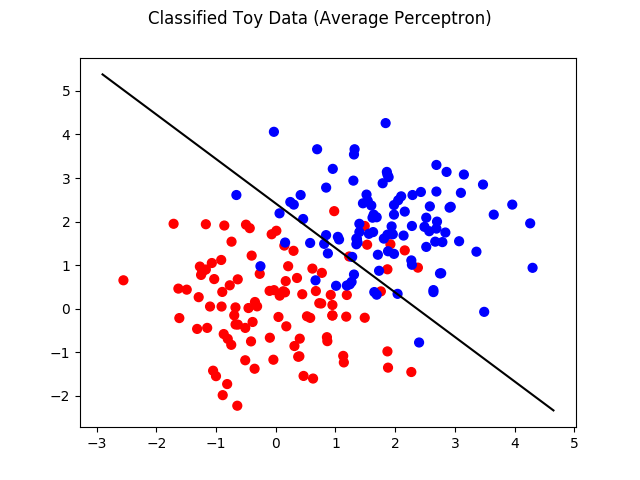
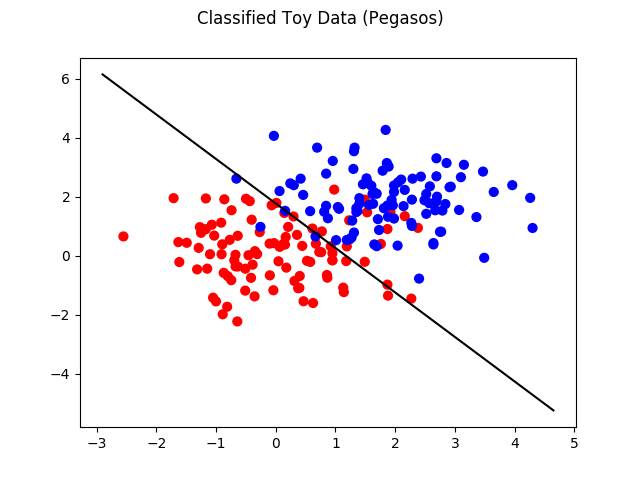
Project 1 Write Up

**Problem 1.7**

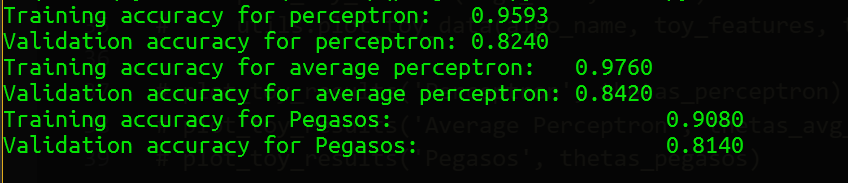
All three algorithms provide different decision boundaries as they all have different single step update methods. While perceptron updates the theta vector according to a standard rule, the average perceptron keeps track of all the different updates to theta and averages it out over the number of updates, thus providing a more accurate classification. Pegasos algorithm has an altogether different update rule and also relies on random numbers while picking the order in which to traverse the training data. *Graphs are attached below--*

****



**Problem 2.9b**

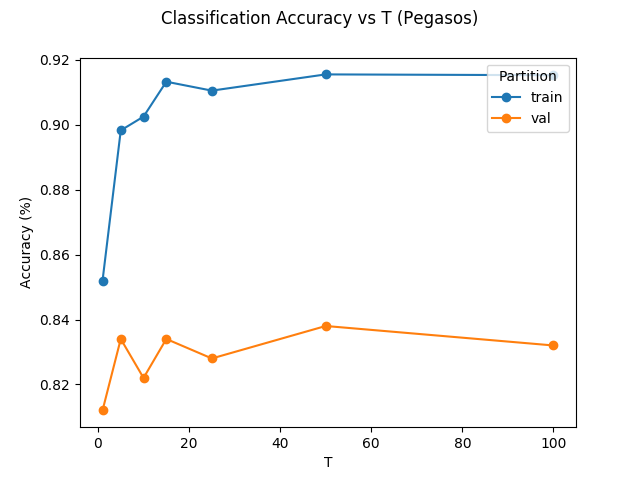
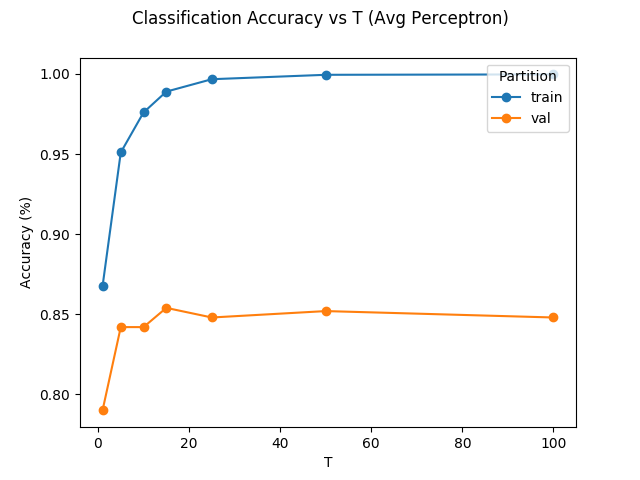
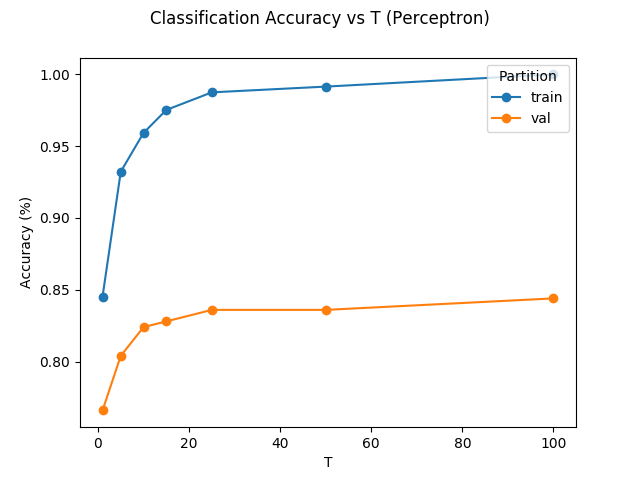
Relevant screenshot reporting the accuracies--

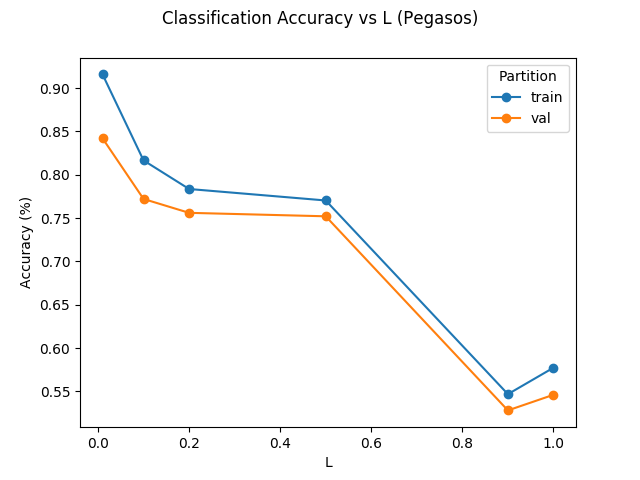


**Problem 2.10**

1. Inferring from the plots, while it does seem that training and validation accuracies have roughly the same shapes, there are deviations from between the two as well. Therefore, I’d say that training and validation accuracies don’t exactly behave similarly as a function of L and T. The reason for this is that while the training set is used explicitly to train the linear classifier, the validation set acts as an external data set that we use to fine tune the hyper parameters. Thus, we can expect the training accuracy to have a high direct correlation between the parameters T and L, it doesn’t necessarily translate to high validation accuracies in general.
2. The average perceptron the best among three.
3. **T optimal values:** 
   1. Perceptron: >=90
   2. Average perceptron: 15
   3. Pegasos: 50 (*with ~0 L value*)

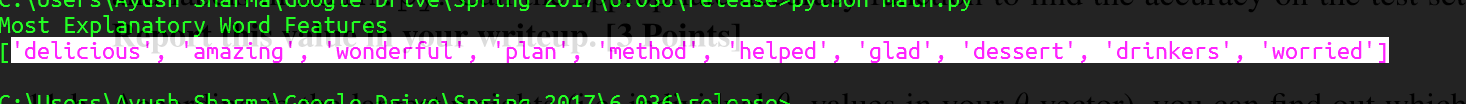
**See plots below:**



****

**Problem 2.11**

1. Accuracy function on test set using average perceptron = 78.4%



**Problem 3.12**

**My approach for optimizing accuracies:**

1. Normalizing the feature matrix: This was the first change I made. For this, took the 2-norm of the feature matrix and divided the feature vectors by that magnitude. Result: No significant improvements in the accuracy on test\_set.
2. Using Term Frequencies (TF): Originally, the feature matrix was augmented with just 0’s and 1’s where 1 would correspond to a “word” being present in the document. However, I modified the code to count the frequency of words and augment the feature\_matrix with that instead. Result: ~1% improvement in accuracies as measured on test\_set.
3. Stop Words Removal: I removed the stop words by modifying the bag\_of\_words function using the stopwords.txt file that was provided. This has significant impact on accuracy: ~3.5%