Tristan Cecil

CS373

8:30 a.m.

PUID:0029627240

4/1/19

1 late day used

CS373 HW3 Write-Up:

Perceptron

1. $f(x) = \begin{cases} 1 \; if \; \sum w_j x_j + b > 0 \\ 0 \; if \; \sum w_j x_j + b \leq 0 \end{cases}$

   Using a bias term is more expressive because it does not anchor the perceptron line to the origin of the graph. Allowing perceptron to find any line that will divide labels most cleanly.

2. A) i: low accuracy (not linearly dividable) ii: low accuracy (not linearly dividable)
   B) i: low accuracy (not linearly dividable) ii: low accuracy (not linearly dividable)
   C) i: high accuracy (dividing line passes through origin ii: high accuracy
   D) i:low accuracy ii: high accuracy

3. When the classifier label doesn't match the gold label during training, the update rule for bias is b' = b + ry where r is the learning rate, y is the gold label, b' is the new bias, and b is the old bias. There is no update rule for bias when the classifier label matches the gold label because perceptron doesn't update itself on a correct guess.

Naïve Bayes

1. P(c+ | d) = P(d|c+)P(c+) / P(d)

2. If V is the vocab size and L is the length of each review,, then we need to keep track of every word and how it effects the probability of classifying a label as positive or negative. Therefore, there would be V^L parameters to keep track of

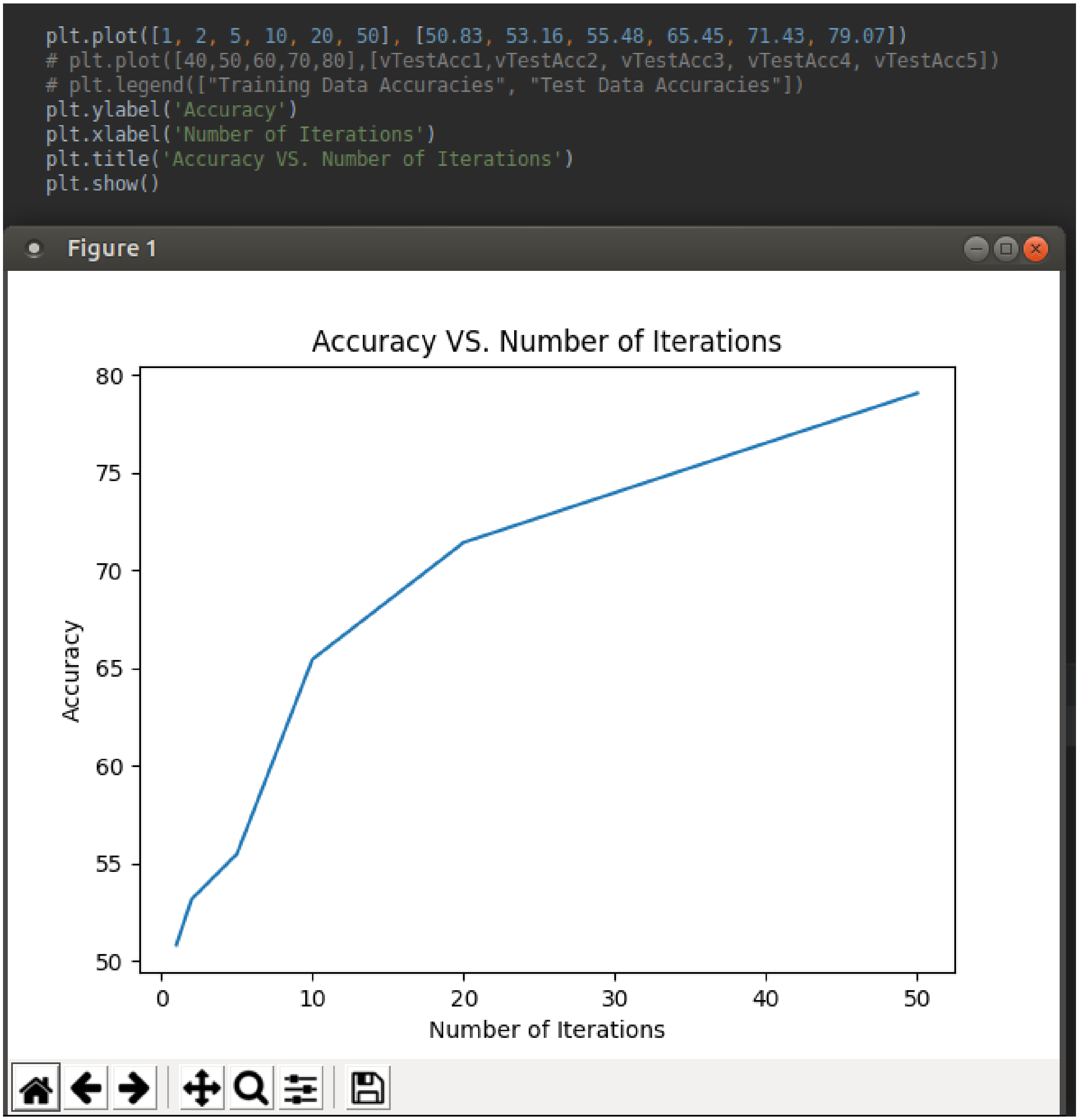
3. It we make the unigram assumption, we would only need to know the probability that a word would show up in a positive review and a negative review. Because we can find this by just using a feature vector that describes a trait that each word in the vocab shows, we only need V parameters to classify.

4. To find the probability that a review will be positive, just find the number of reviews in the training data that are given positive, and divide by the total number of reviews. To find negative probability, find the number of negative reviews and divide by the total number of reviews.

Analysis

1. The accuracy is worse, because we are constraining the dividing line to go through the origin, therefore not expressing the best way to separate the two label groupings

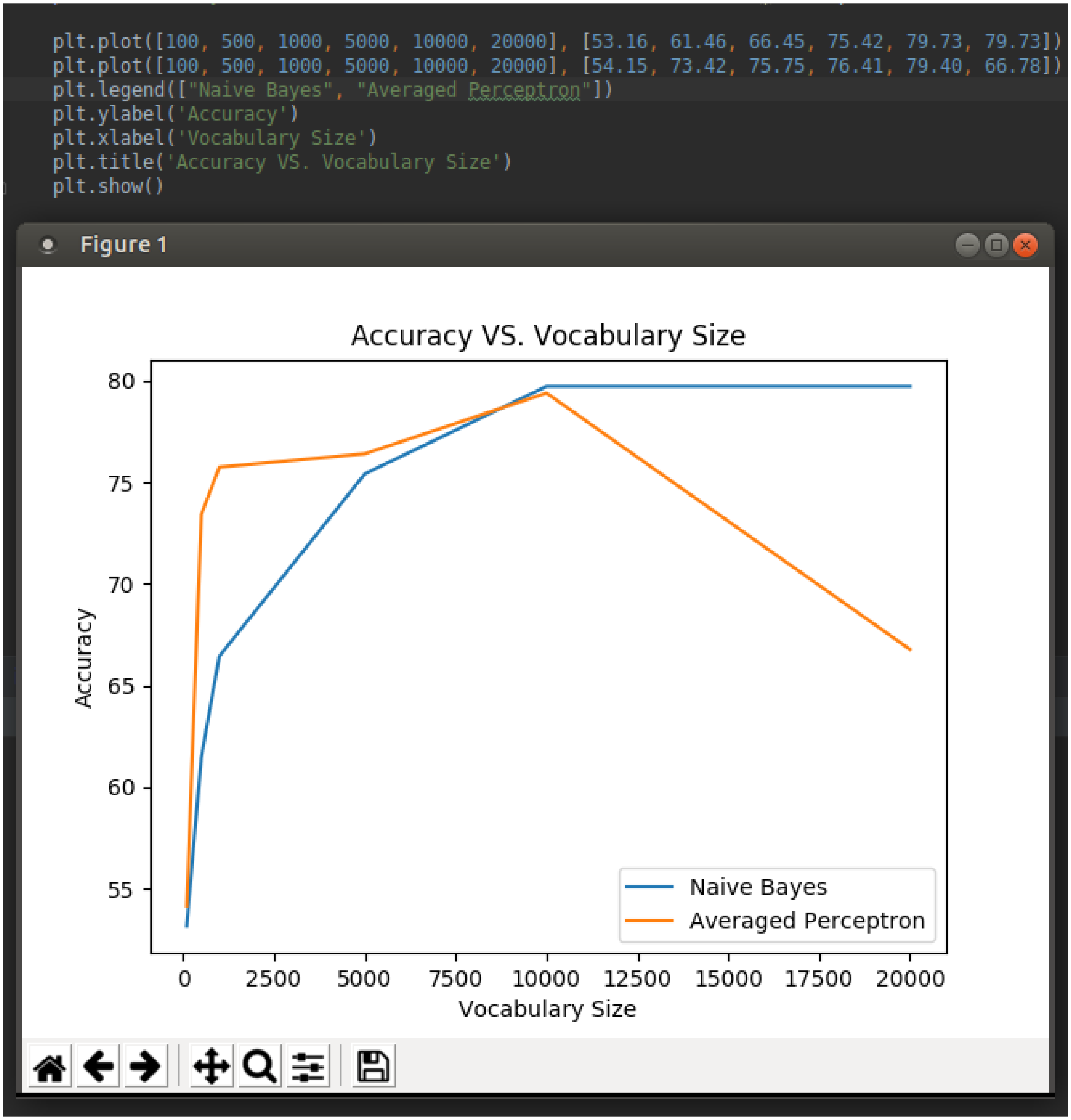
2.



```
plt.plot([1, 2, 5, 10, 20, 50], [50.83, 53.16, 55.48, 65.45, 71.43, 79.07])
# plt.plot([40,50,60,70,80],[vTestAcc1,vTestAcc2, vTestAcc3, vTestAcc4, vTestAcc5])
# plt.legend(["Training Data Accuracies", "Test Data Accuracies"])
plt.ylabel('Accuracy')
plt.xlabel('Number of Iterations')
plt.title('Accuracy VS. Number of Iterations')
plt.show()
```

No, perceptron will not converge. This is because the test data is not perfectly linearly dividable.

3.



```
plt.plot([100, 500, 1000, 5000, 10000, 20000], [53.16, 61.46, 66.45, 75.42, 79.73, 79.73])
plt.plot([100, 500, 1000, 5000, 10000, 20000], [54.15, 73.42, 75.75, 76.41, 79.40, 66.78])
plt.legend(["Naive Bayes", "Averaged Perceptron"])
plt.ylabel('Accuracy')
plt.xlabel('Vocabulary Size')
plt.title('Accuracy VS. Vocabulary Size')
plt.show()
```

Naive Bayes will gradually get more accurate when more of the vocab is looked at, because it can better gauge how each word effects a review's classification. Perceptron benefits from using a larger vocab size up to a certain point, but since the reviews likely don't hold that many words and the feature vector and weight vector are initialized with zeros, this many zeros likely corrupts the classification.