There are 7222 labels so there must be that many docs / examples

This is the row index of the train.X

train.X.shape[0]

This line determines shape of train.X

self.X = csr\_matrix((X\_values, (X\_row\_indices, X\_col\_indices)), shape=(max(X\_row\_indices)+1, self.vocab.GetVocabSize()))

Remember perceptron is just a one layer nn

Trying to classify documents as positive or negative so think of it as the doc is the dot / datapoint itself on an xy plane and the counts of words are x and y coords of the data point however of course they are many many more dimensional

**So we want weights equal to the vocab size**

Each word is a feature

You have 80 samples with 2 features. For each feature you will train one weight if you use a single layer perceptron. Hence you will need only 2 weights. But it's also common to introduce a bias layer with another weight. Thus you would use 3 weights. Also It is possible that your error will never reach 0 as a single layer perceptron can not solve xor for example.

The weight w d is increased by yx d and the bias is increased by y.

Generally there is a weight vector for each output class or just one if binary

Tips for implementing normal perceptron:

ciml.info/dl/v0\_99/ciml-v0\_99-ch04.pdf

Thesis paper for implementing averaged perceptron

<http://www.umiacs.umd.edu/~hal/docs/daume06thesis.pdf>

Incase you have sparse matrix mult issues later

<https://piazza.com/class/kj7vngax6ni7lt?cid=25>

Below are the minimum test set accuracies for different models:

Perceptron:

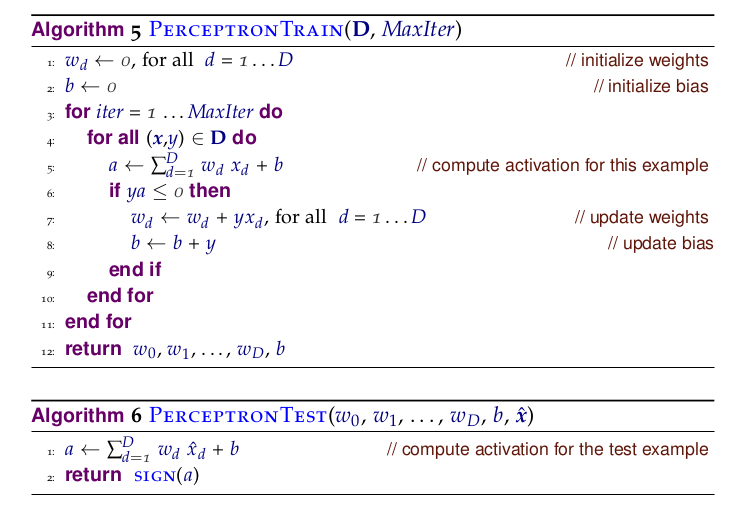
Basic: 80%

Parameter Averaging: 82%

Neural Bag of Words: 83%

CNN: 82%

If your models are able to achieve the test accuracies listed above, it is likely that your implementations are correct, so we will give you full credits. You are welcome to tune the hyperparameters to make your models perform better on those test sets.



def Train(self, X, Y):

for iter in range(self.max\_iterations):

print("Iteration: ", iter)

for example\_ind in range(X.shape[0]):

example = X[example\_ind].transpose()

print(example.shape)

example\_gold = Y[example\_ind]

activation = self.weights.transpose().dot(example)[0,0]

target = 1.0 if (y > 0) else 0.0

delta = (label.item(0,0) - target)

if(delta): # misclassified

w += (delta \* x)

if (example\_gold \* activation) <= 0: # ya <= 0

self.weights = self.weights + example\_gold \* example # w = w + yx

self.avg\_weights = self.avg\_weights + self.c \* example\_gold \* example

self.c += 1

return self.weights

def Predict(self, X):

results = np.zeros((X.shape[0]))

for example\_ind in range(X.shape[0]):

example = X[example\_ind].transpose()

activation = self.weights.transpose().dot(example)[0,0]

# print(activation)

results[example\_ind] = 1 if activation >= 0 else 0

return results