CSE 5525 Speech and Language Processing (Spring 2017)

Homework #1: Text Classification

Akshay Mehra

mehra.42@osu.edu

Introduction:

This is the report for the first homework for Speech and Language Processing. In this homework, I have implemented Naïve Bayes and Perceptron Algorithms as presented in class. The report contains my results of running Naïve Bayes for various values of ALPHA (the smoothing factor) and number of iterations for the perceptron.

I have utilized the starter code provided for this assignment along with functions from python's Numpy and Scipy packages.

PART 1: Naïve Bayes

The first part of the assignment asked to implement a Naïve Bayes Classifier and evaluate its performance on the Large Movie review dataset, for different values of the hyper-parameter alpha.

Below are my results

Hyper-Parameter (ALPHA)	Train Accuracy (%)	Test Accuracy (%)
0.1	97.768	81.132
0.5	95.876	82.148
1.0	94.512	82.284
5.0	90.665	82.772
10.0	88.912	82.816

In the second part of the assignment we had to print the probability of the reviews being positive or negative as predicted by the classifier (results have been rounded), for first 10 reviews in the test dataset.

S. No.	Movie Review (shortened to 7 lines for presenting in the report)	Actual Label	Hyper-Parameter (ALPHA)	Probability of being Positive	Probability of being Negative
1	I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions.	-1.0	0.1	0.2465	0.7535
			0.5	0.009	0.991
			1.0	0.004	0.996
			5.0	0.002	0.998
			10.0	0.002	0.998
2	Actor turned director Bill Paxton follows up his promising debut, the Gothic-horror "Frailty", with this family friendly sports drama about the 1913 U.S. Open where a young American caddy rises from his humble background to play against his Bristish idol in what was dubbed as "The Greatest Game Ever Played." I'm no fan of golf, and these scrappy underdog sports flicks are a dime a dozen (most recently done to grand effect with "Miracle" and "Cinderella Man…	-1.0	0.1	0.01	0.99
			0.5	0.01	0.99
			1.0	0.01	0.99
			5.0	0.01	0.99
			10.0	0.01	0.99
3	As a recreational golfer with some knowledge of the sport's history, I was pleased with Disney's sensitivity to the issues of class in golf in	-1.0	0.1	0.996	0.004
			0.5	0.986	0.014

	the early twentieth century. The movie depicted well the psychological battles that Harry Vardon fought within himself, from his childhood trauma of being evicted to his own inability to break that glass ceiling that prevents him from being equal, …		1.0	0.979	0.021
			5.0	0.949	0.051
			10.0	0.928	0.072
4	I saw this film in a sneak preview, and it is delightful. The cinematography is unusually creative, the acting is good, and the story is fabulous. If this movie does not do well, it won't be because it doesn't deserve to. Before this film, I didn't realize how charming Shia Lebouf could be. He does a marvelous, self-contained, job as the lead. There's something incredibly sweet about him, and it makes the movie even better …	-1.0	0.1	0.306	0.694
			0.5	0.05	0.95
			1.0	0.04	0.966
			5.0	0.027	0.973
			10.0	0.024	0.976
5	Bill Paxton has taken the true story of the 1913 US golf open and made a film that is about much more than an extra-ordinary game of golf. The film also deals directly with the class tensions of the early twentieth century and touches upon the profound anti-Catholic prejudices of both the British and American establishments. But at heart the film is about that perennial favourite of triumph against the odds	-1.0	0.1	0.001	0.999
			0.5	0.001	0.999
			1.0	0.003	0.997
			5.0	0.017	0.983
			10.0	0.028	0.972
6	I saw this film on September 1st, 2005 in Indianapolis. I am one of the judges for the Heartland Film Festival that screens films for their Truly Moving Picture Award. A Truly Moving Picture "...explores the human journey by artistically expressing hope and respect for the positive values of life." Heartland gave that award to this film. This is a story of golf in the early part of the 20th century. At that time, it was the game of upper class and …	-1.0	0.1	4.911e-20	1.0
			0.5	2.66e-19	1.0
			1.0	6.15e-19	1.0
			5.0	8.74e-18	1.0
			10.0	4.69e-17	1.0
7	Maybe I'm reading into this too much, but I wonder how much of a hand Hongsheng had in developing the film. I mean, when a story is told casting the main character as himself, I would think he would be a heavy hand in writing, documenting, etc. and that would make it a little biased. But...his family and friends also may have had a hand in getting the actual details about Hongsheng's life. I think the best view would have been told from Hongsheng's family and friends' perspectives.	-1.0	0.1	3.83e-31	1.0
			0.5	1.15e-14	1.0
			1.0	1.42e-08	1.0
			5.0	0.994	0.004
			10.0	0.999	0.001
8	I felt this film did have many good qualities. The cinematography was certainly different exposing the stage aspect of the set and story. The original characters as actors was certainly an achievement and I felt most played quite convincingly, of course they are playing themselves, but definitely unique. The cultural aspects may leave many disappointed as a familiarity with the Chinese and Oriental culture will answer a lot of questions regarding parent/child relationships and the stigma that goes with any drug use…	1.0	0.1	1.0	3.43e-18
			0.5	1.0	5.37e-17
			1.0	1.0	3.02e-16
			5.0	1.0	8.8e-14
			10.0	1.0	2.3e-12
9	This movie is amazing because the fact that the real people portray themselves and their real life experience and do such a good job it's like they're almost living the past over again. Jia Hongsheng plays himself an actor who quit everything except music and drugs struggling with depression and searching for the meaning of life while being angry at everyone especially the people who care for him most. There's moments in the movie that will make you wanna cry because the family especially the father did such a good job	-1.0	0.1	0.001	0.999
			0.5	0.001	0.999
			1.0	0.001	0.999
			5.0	0.001	0.999
			10.0	0.001	0.999
10	"Quitting" may be as much about exiting a pre-ordained identity as about drug withdrawal. As a rural guy coming to Beijing, class and success must have struck this young artist face on as an appeal to separate from his roots and far surpass his peasant parents' acting success. Troubles arise, however, when the new man is too new, when it demands too big a departure from family, history, nature, and personal identity. The ensuing splits, and confusion between the imaginary and the real and the dissonance between …	-1.0	0.1	0.386	0.614
			0.5	0.384	0.616
			1.0	0.382	0.618
			5.0	0.3621	0.6379
			10.0	0.344	0.656

Code Snippet for Training Naïve Bayes

```
def Train(self, X, Y):
    #TODO: Estimate Naive Bayes model parameters
    positive_indices = np.argwhere(Y == 1.0).flatten()
    negative_indices = np.argwhere(Y == -1.0).flatten()
    self.num_positive_reviews = len(positive_indices)
    self.num_negative_reviews = len(negative_indices)
    self.count_positive = csr_matrix.sum(X[np.ix_(positive_indices)], axis=0)
    self.count_negative = csr_matrix.sum(X[np.ix_(negative_indices)], axis=0)
    self.total_positive_words = csr_matrix.sum(X[np.ix_(positive_indices)])
    self.total_negative_words = csr_matrix.sum(X[np.ix_(negative_indices)])
    self.deno_pos = float(self.total_positive_words+ self.ALPHA * X.shape[1])
    self.deno_neg = float(self.total_negative_words+ self.ALPHA * X.shape[1])
    self.count_positive = (self.count_positive + self.ALPHA)
    self.count_negative = (self.count_negative + self.ALPHA)
    return
```

Code Snippet for predicting using Naïve Bayes

```
def PredictLabel(self, X):
    #TODO: Implement Naive Bayes Classification
    self.P_positive = log(float(self.num_positive_reviews)) -
log(float(self.num_positive_reviews + self.num_negative_reviews))
    self.P_negative = log(float(self.num_negative_reviews)) -
log(float(self.num_positive_reviews + self.num_negative_reviews))
    pred_labels = []

    sh = X.shape[0]
    for i in range(sh):
        z = X[i].nonzero()
        sum_positive = self.P_positive
        sum_negative = self.P_negative
        for j in range(len(z[0])):
            row_index = i
            col_index = z[1][j]
            times = X[row_index, col_index]

            P_pos = log(self.count_positive[0, col_index]) -
log(self.deno_pos)
            sum_positive = sum_positive + times * P_pos

            P_neg = log(self.count_negative[0, col_index]) -
log(self.deno_neg)
            sum_negative = sum_negative + times * P_neg

        if sum_positive > sum_negative:
            pred_labels.append(1.0)
        else:
            pred_labels.append(-1.0)

    return pred_labels
```

Code Snippet for predicting probability

```
def LogSum(self, logx, logy):
    # TO Do: Return log(x+y), avoiding numerical underflow/overflow.
    m = max(logx, logy)
    return m + log(exp(logx - m) + exp(logy - m))

def PredictProb(self, test, indexes):
    for i in indexes:
        # TO DO: Predict the probability of the i_th review in test being
positive review
        # TO DO: Use the LogSum function to avoid underflow/overflow
        predicted_label = 0
        z = test.X[i].nonzero()
        sum_positive = self.P_positive
        sum_negative = self.P_negative
        for j in range(len(z[0])):
            row_index = i
            col_index = z[1][j]
            times = test.X[row_index, col_index]

            P_pos = log((self.count_positive[0, col_index]))
            sum_positive = sum_positive + times * P_pos

            P_neg = log((self.count_negative[0, col_index]))
            sum_negative = sum_negative + times * P_neg

        predicted_prob_positive = exp(sum_positive -
self.LogSum(sum_positive, sum_negative))
        predicted_prob_negative = exp(sum_negative -
self.LogSum(sum_positive, sum_negative))

        if sum_positive > sum_negative:
            predicted_label = 1.0
        else:
            predicted_label = -1.0

        #print test.Y[i], test.X_reviews[i]
        # TO DO: Comment the line above, and uncomment the line below
        print test.Y[i], predicted_label, predicted_prob_positive,
predicted_prob_negative, test.X_reviews[i]
```

PART 2: Perceptron

This part of the assignment required to implement the Perceptron and Averaged Algorithm and evaluate its performance on the Large Movie Review dataset. It also asked to tune the hyperparameter (the number of iterations).

Below are the results for the Perceptron and Average Perceptron

Iterations	**Perceptron Accuracy (%)**		**Averaged Perceptron Accuracy (%)**	
	Train	**Test**	**Train**	**Test**
1	75.04	74.23	85.92	83.84
10	95.04	86.11	94.16	87.2
50	99.7	86.2	98.9	87.1
100	99.9	86.2	99.8	86.8
1000	100	86.1	99.89	86.77

The last part of the assignment asks to print the most positive and the most negative words in the corpus using the weights assigned to each word by the averaged perceptron algorithm.

Below are the results for it (after 10 iteration)

Positive Words	**Weight**		**Negative Words**	**Weight**
excellent	132.173		worst	-263.23
perfect	129.01		waste	-236.81
favorite	114.624		poorly	-177.46
wonderful	105.269		boring	-152.52
amazing	104.369		awful	-147.64
loved	102.437		annoying	-142.74
subtle	97.541		worse	-131.96
easy	96.8802		fails	-128.31
7	95.4269		dull	-126.78
wonderfully	93.8305		lame	-126.49
rare	93.4065		poor	-126.36
highly	92.8701		disappointing	-115.51
superb	92.5654		pointless	-115.23
funniest	89.3295		awful.	-114.96
!	87.8333		save	-111.95
noir	85.9518		bad.	-110.99
tony	85.7079		lacks	-110.15
great	83.2553		badly	-109.81
fantastic	82.425		supposed	-109.33
atmosphere	82.2412		nothing	-107.09

After 50 iterations

Positive Words	Weight		Negative Words	Weight
wonderfully	169.175		worst	-261.07
7	152.197		waste	-254.65
subtle	138.92		poorly	-247.11
rare	138.124		awful.	-183.52
7/10	134.585		fails	-183.52
refreshing	132.403		boring	-179.55
favorite	131.246		disappointing	-169.69
perfect	129.97		annoying	-166.26
excellent	128.012		lame	-164.19
funniest	127.514		lacks	-157.73
perfect.	122.003		terrible.	-151.71
amazing.	121.787		dull	-151.32
noir	121.472		awful	-146.65
highly	120.425		pointless	-146.2
superb	119.219		badly	-144.01
captures	118.911		worse	-139.67
8	117.234		mildly	-137.54
delightful	116.613		save	-135.18
surprisingly	116.351		annoying.	-134.06
perfect	115.446		disappointment	-128.61

After 100 iterations

Positive Words	Weight		Negative Words	Weight
wonderfully	187.354		poorly	-271.62
7	164.304		worst	-267.19
07/10	158.717		waste	-256.61
rare	153.551		awful.	-202.82
refreshing	150.008		fails	-197
subtle	147.056		boring	-187.27
perfect.	140.869		disappointing	-180.15
amazing.	136.819		annoying	-174.56
favorite	135.566		lacks	-172.55
noir	134.068		lame	-168.66
funniest	133.956		terrible.	-164.46
highly	133.914		dull	-152.54
perfect	130.863		pointless	-151.53
surprisingly	129.797		04/10	-151.26
delightful	129.407		annoying.	-149.82
8	128.823		badly	-149.17
excellent.	128.662		disappointment	-148.84
captures	128.608		awful	-147.96
excellent	127.284		mildly	-147.29
perfect	126.323		worse	-142.78

After 1000 iterations

Positive Words	Weight		Negative Words	Weight
wonderfully	189.82		poorly	-275.35
7	166.855		worst	-267.66
07/10	163.264		waste	-257.02
rare	156.12		awful.	-205.82
refreshing	152.793		fails	-200.23
subtle	148.165		boring	-188.79
perfect.	144.033		disappointing	-182.1
amazing.	139.461		annoying	-175.81
favorite	136.996		lacks	-175.06
noir	136.762		lame	-170.13
highly	136.238		terrible.	-167.1
funniest	135.79		04/10	-155.95
perfect	133.527		annoying.	-152.77
surprisingly	131.942		dull	-152.67
excellent.	131.921		pointless	-152.5
delightful	131.633		disappointment	-151.64
8	131.206		badly	-150.95
captures	130.193		mildly	-149.45
superb	127.372		awful	-149.33
excellent	127.108		weak	-143.98

Code Snippet for Training Perceptron and Averaged Perceptron

```
def ComputeAverageParameters(self):
    #TODO: Compute average parameters (do this part last)
    self.weight = self.weight - (self.for_avg_weight / float(self.penalty))
    return

def Train(self, X, Y):
    #TODO: Estimate perceptron parameters
    ite = self.N_ITERATIONS
    examples = self.examples
    activation = 0
    weight_transpose = np.zeros([X.shape[1],1])
    for i in range(ite) :
        converged = 1
        for j in range(examples):
            term = (X[j].dot(weight_transpose))
            activation = 0
            if(term > 0.0) :
                activation = 1.0
            elif term < 0.0 :
                activation =-1.0
            if Y[j] != activation:
                weight_transpose += (Y[j] * X[j].transpose())
                self.for_avg_weight += Y[j] * self.penalty * X[j]
                converged = 0
            self.penalty +=  1
        if converged == 1:
            break
    self.weight = weight_transpose.transpose()
    return
```

Code Snippet for Predicting

```
def Predict(self, X):
    #TODO: Implement perceptron classification
    pred_labels = []
    examples = X.shape[0]
    weight_transpose = self.weight.transpose()
    for j in range(examples):
        if (X[j].dot(weight_transpose)) > 0 :
            pred_labels.append(1.0)
        else:
            pred_labels.append(-1.0)

    return pred_labels
```

Conclusion

From the above results using ALPHA = 10 yields the highest accuracy of about 82% in the Naïve Bayes Classifier. For the perceptron, we notice that the algorithm converges at about 118 iterations indicating that the data is linearly separable. The most positive and negative words also give us a good indication that the algorithm is correctly able to learn positive and negative words present in the reviews. In general, the performance of the average perceptron is higher than Naïve Bayes and is about 87%. We notice in the perceptron that increasing number of iterations does not always improve the test accuracy which is in line with the results of early stopping.