CS 580

Homework 1 Report

Yash Agarwal
CWID: A20392372

Classification and Evaluation

Train Method

Code 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_P = np.ix_(positive_indices)
    self.count_N = np.ix_(negative_indices)

    self.count_positive = self.count_positive + csr_matrix.sum(X[self.count_P], axis=0) + self.ALPHA
    self.count_negative = self.count_negative + csr_matrix.sum(X[self.count_N], axis=0) + self.ALPHA

    self.total_positive_words = csr_matrix.sum(X[self.count_P])
    self.total_negative_words = csr_matrix.sum(X[self.count_N])

    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])


    return
```

PredictLabel Method

This method is used to predict the class of all the reviews. The class is predicted by comparing only the numerator for a given review. The numerator is calculated by using log to avoid floating point underflows.

Code for PredictLabel Method:

```
def PredictLabel(self, X):
    #TODO: Implement Naive Bayes Classification

    P_review= float(self.num_positive_reviews)
    N_review= float(self.num_negative_reviews)
    P_review= math.log(P_review)
    N_review= math.log(N_review)
    self.P_positive = P_review - (P_review + N_review)
    self.P_negative = N_review - (P_review + N_review)
    pred_labels = []
```

```
    sh = X.shape[0]
    for i in range(sh):
        z = X[i].nonzero()
        P_positive_c= self.P_positive
        P_negative_c=self.P_negative
        for j in range(len(z[0])):
            #col = z[1][j]
            x = X[i, z[1][j]]
            p = self.count_positive[0, z[1][j]]
            n = self.count_negative[0, z[1][j]]
            positive_score = math.log(p) - math.log(self.deno_pos)
            negative_score = math.log(n) - math.log(self.deno_neg)
            P_positive_c = P_positive_c + x * positive_score
            P_negative_c = P_negative_c + x * negative_score


        if P_positive_c > P_negative_c:
            pred_labels.append(1.0)
        else:
            pred_labels.append(-1.0)

    return pred_labels
```

The table below illustrates accuracy for various values of ALPHA. We observe the maximum accuracy when ALPHA is 10.0 :

ALPHA	ACCURACY
0.1	0.81131
0.5	0.82148
1.0	0.82284
5.0	0.82272
10.0	0.82812

Probability Prediction

PredictProb Method

The probabilities are estimated for first 10 reviews.

Code for PredictProb Method:

```
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()
        P_positive_c= self.P_positive
        P_negative_c=self.P_negative
        for j in range(len(z[0])):
          x = test.X[i, z[1][j]]
          p = math.log(self.count_positive[0, z[1][j]])
          n = math.log(self.count_negative[0, z[1][j]])
          P_positive_c = P_positive_c + x * p
          P_negative_c = P_negative_c + x * n

        #predicted_prob_positive = 0.5
        #predicted_prob_negative = 0.5
        predicted_prob_positive = exp(P_positive_c - self.LogSum(P_positive_c, P_negative_c))
        predicted_prob_negative = exp(P_negative_c - self.LogSum(P_positive_c, P_negative_c))

        sum_positive=P_positive_c
        sum_negative=P_negative_c

        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])
```

The table below shows the predicted probabilities for the 10 movie reviews when

ALPHA=0.5

Given Class	Predicted Class	Predicted Probability Positive	Predicted Probability Negative	Reviews
-1.0	-1.0	0.00927	0.990729	I went and saw this at my movie last night after being coaxed…
-1.0	-1.0	2.979127e-08	0.99999	Actors turned directed Bill Paxton follows up his promosing….
-1.0	-1.0	0.986705	0.0132944	As a recreational golfer with some knowledge of the soprt’s…

-1.0	-1.0	0.0503312767	0.9496687	I saw this film in a sneak preview, and it is delightful…
-1.0	-1.0	0.000640133	0.9993598	Bill Paxton has taken the true story of the 1913…
-1.0	-1.0	2.66900e-19	1.0	I saw this film on September 1st, 2005 in Indianapolis…
-1.0	-1.0	1.151108e-14	1.0	Maybe I'm reading into this too much, but I wonder how much….
1.0	1.0	1.0	5.371760e-17	I felt this film did have many good qualities. The cinematography was certainly….
-1.0	-1.0	1.82320641e-08	0.9999	This movie is amazing because the fact that the real people portray themselves and…
-1.0	-1.0	0.3840485	0.61595149	"Quitting" may be as much about exiting a pre-ordained identity as about drug withdrawal…

Precision and Recall

EvalPrecision

Code for EvalPrecision Method:

```
def EvalPrecision(self, target, predict):

    tp=0
    fp=0
    for i in range(len(target)):
        if (target[i] ==1 and predict[i] == 1):
            tp= tp+1
        elif (target[i] ==1 and predict[i] == -1):
            fp = fp+1
```

```
    self.precision= tp/(tp+fp)
    return self.precision
```

EvalRecall

Code for EvalRecall Method:

```
def EvalRecall(self, target, predict):

    fn=0
    tn=0
    tp=0
    for i in range(len(target)):
      if (target[i] == -1 and predict[i] == 1):
        fn= fn+1
      elif (target[i] == -1 and predict[i] == -1):
        tn = tn+1
      elif (target[i] == 1 and predict[i] == 1):
        tp= tp+1
    self.recall= tp/(tp+fn)
    return self.recall
```

Modified PredictLabel for handling Threshold

```
def PredictLabel_threshold(self, X,probThresh):
    #TODO: Implement Naive Bayes Classification

    P_review= float(self.num_positive_reviews)
    N_review= float(self.num_negative_reviews)
    P_review= math.log(P_review)
    N_review= math.log(N_review)
    self.P_positive = P_review - (P_review + N_review)
    self.P_negative = N_review - (P_review + N_review)

    pred_labels = []

    sh = X.shape[0]
    for i in range(sh):
      z = X[i].nonzero()
      P_positive_c= self.P_positive
      P_negative_c=self.P_negative
      for j in range(len(z[0])):
        #col = z[1][j]
        x = X[i, z[1][j]]
        p = self.count_positive[0, z[1][j]]
        n = self.count_negative[0, z[1][j]]
        positive_score = math.log(p) - math.log(self.deno_pos)
        negative_score = math.log(n) - math.log(self.deno_neg)
```

```
            P_positive_c = P_positive_c + x * positive_score
            P_negative_c = P_negative_c + x * negative_score
            #percentage = P_positive_c / (P_positive_c + P_negative_c)
            percentage = exp(P_positive_c - self.LogSum(P_positive_c,P_negative_c))



        if percentage > probThresh:
            pred_labels.append(1.0)
        else:
            pred_labels.append(-1.0)

    return pred_labels
```

Modified PredictProb for implementing EvalPrecision and EvalRecall:

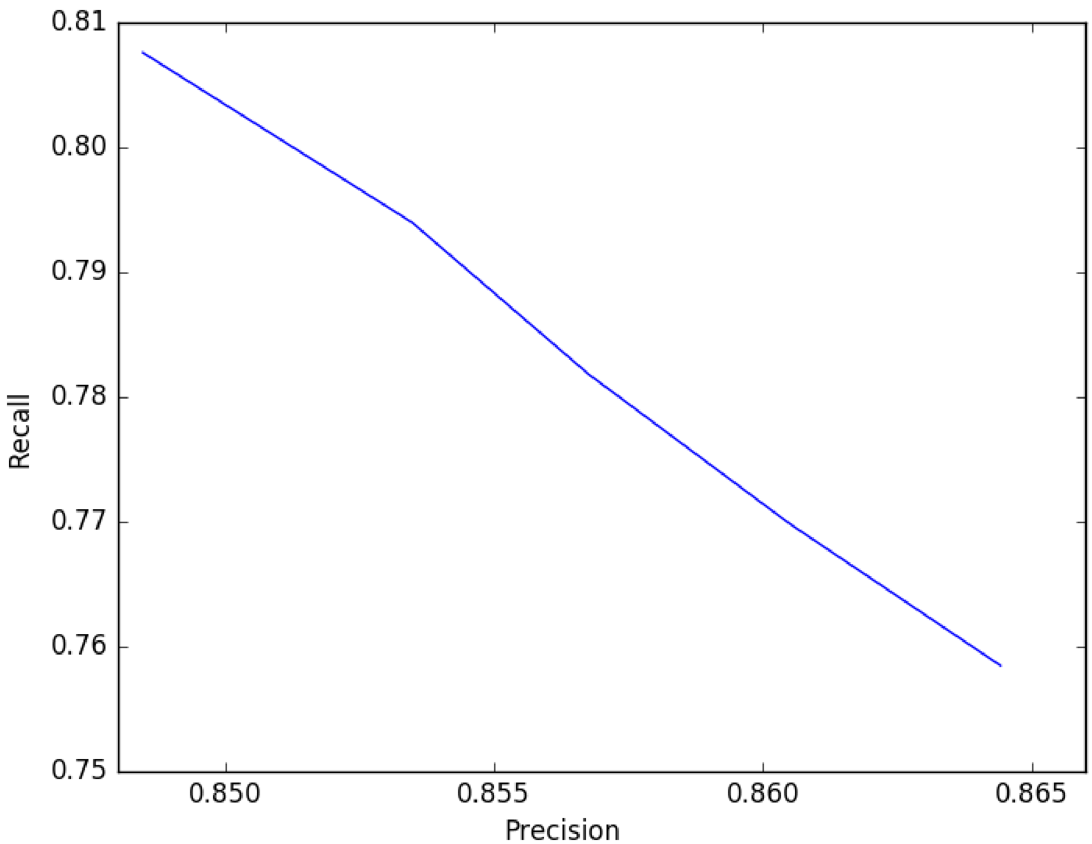
```
l= (0.22,0.32,0.42,0.52,0.62)
    precision_list=[]
    recall_list=[]

    for i in range(len(l)):
        Y_pred_1 = self.PredictLabel_threshold(test.X,l[i])
        #ev = Eval(Y_pred_1, test.Y)
        #Y_pred1 = self.PredictProb(test.X)

        precision_prob = self.EvalPrecision(Y_pred_1,test.Y)
        recall_prob = self.EvalRecall(Y_pred_1,test.Y)
        #print ("Precision:",precision_label)
        #print ("Recall:",recall_label)
        print ("Precision:",precision_prob)
        print ("Recall:",recall_prob)
        precision_list.append(precision_prob)
        recall_list.append(recall_prob)
    plt.plot(precision_list,recall_list)
    plt.xlabel("Precision")
    plt.ylabel("Recall")
    plt.savefig("graph.png")
```

Below is plot for precision vs recall, when we vary the threshold for the negative and the positive classes, I have used the threshold as 0.22,0.32,0.42,0.52,0.62 for ALPHA=1.0.

We see that when the threshold value increases, there is a increase in precision and a decrease in recall:



Features

getWords Method:

The getWords method is implemented to get top most 20 positive and negative words with their weights.

Code for getWords Method:

```
def getWords(self, X, vocab):
	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))
	ft_wt_pos = {}
	ft_wt_neg = {}
```

```
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
      word = vocab.GetWord(int(col_index))
      if word not in ft_wt_pos:
         ft_wt_pos[word] = P_pos
      else:
         ft_wt_pos[word] += P_pos
      if word not in ft_wt_neg:
         ft_wt_neg[word] = P_neg
      else:
         ft_wt_neg[word] += P_neg


twenty_largest_p = sorted(ft_wt_pos, key=ft_wt_pos.get)[:100]
twenty_largest_n = sorted(ft_wt_neg, key=ft_wt_neg.get)[:100]

print("Top 20 Positive Words: ")
for i in twenty_largest_p:
   print(i, ft_wt_pos[i])
print("Top 20 Negative Words: ")
for i in twenty_largest_n:
   print(i, ft_wt_neg[i])
```

The following table illustrates the top 20 positive words with their weights:

Positive Words	**Weight**
wonderfully	187.354
7	164.304
07/10	158.717
rare	153.551
refreshing	150.008
subtle	147.056

perfect.	140.869
amazing.	136.819
favorite	135.566
noir	134.068
funniest	133.956
highly	133.914
perfect	130.863
surprisingly	129.797
delightful	129.407
8	128.823
excellent.	128.662
captures	128.608
excellent	127.284
perfect	126.323

The following table illustrates the top 20 negative words with their weights:

Negative Words	Weight
Poorly	-271.62
Worst	-267.19
Waste	-256.61
awful.	-202.82
Fails	-197
Boring	-187.27
disappointing	-180.15
annoying	-174.56
Lacks	-172.55
Lame	-168.66
terrible.	-164.46
Dull	-152.54
pointless	-151.53
04/10	-151.26
annoying.	-149.82
Badly	-149.17
disappointment	-148.84
Awful	-147.96
Mildly	-147.29
worse	-142.78