CSE 5525

HOMEWORK I – Report

Pragya Arora

arora.170

1. Naïve Bayes

Naïve Bayes Classifier is used to train IMDB dataset with smoothing hyperparameter ALPHA.

1.1. Classification and Evaluation

Two methods are implemented in this section.

1.1.1. Train method

In this method model is trained by using Naïve Bayes classification. Two dictionaries for positive and negative words in the IMDB review vocabulary are maintained which will be used to predict labels. Positive and negative word counts are also calculated in this method.

```
1.  def Train(self, X, Y):
2.      data = X.data
3.      indptr = X.indptr
4.      indices = X.indices
5.
6.      for i in range(len(Y)):
7.          #Calculate total positive and negative words
8.          if Y[i]==1.0:
9.              self.pos += 1
10.         else:
11.             self.neg += 1
12.
13.         index=indices[indptr[i]:indptr[i+1]]
14.         for k in range(0,len(index)):
15.             wordId=index[k]
16.             v=data[indptr[i]+k]
17.
18.             #Calculate wordId counts for positive and negative reviews
19.             if Y[i]==1.0:
20.                 self.posDict[wordId]=self.posDict.get(wordId,0)+v
21.                 self.totalPos+=v
22.             else:
23.                 self.negDict[wordId] = self.negDict.get(wordId, 0) + v
24.                 self.totalNeg+=v
25.     return
```

1.1.2. PredictLabel method

This method is used to predict 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 so as to avoid floating point underflows. Various values of hyperparameter ALPHA are tried to calculate accuracy of the classifier.

```
def PredictLabel(self, X):
    data = X.data
    indptr = X.indptr
    indices = X.indices
    vocabSize=self.data.vocab.GetVocabSize()
    Y = []

    #Calculate positive and negative review probabilities
    probPos = (self.pos*1.0)/(self.pos+self.neg)
    probNeg = (self.neg*1.0)/(self.pos+self.neg)

    #Predict class of review
    for i in range(0,len(indptr)-1):
        index = indices[indptr[i]:indptr[i + 1]]
        posClass=math.log(probPos)
        negClass=math.log(probNeg)
        for k in range(0, len(index)):
            wordId = index[k]
            posClass += math.log((self.posDict.get(wordId, 0) + self.ALPHA) / (self.totalPos + (self.ALPHA*vocabSize)))
            negClass += math.log((self.negDict.get(wordId,0) + self.ALPHA)/(self.totalNeg + (self.ALPHA*vocabSize)))

        if posClass>negClass:
            Y.append(+1.0)
        else:
            Y.append(-1.0)
    return Y
```

The following table illustrates accuracy on the IMDB data for various values of ALPHA. The maximum accuracy is attained at ALPHA=5

ALPHA	ACCURACY (%)
0.1	81.688
0.5	82.668
1.0	83.04
5.0	83.632
10.0	83.712

1.2. Probability Prediction

In this section, following two methods are implemented.

1.2.1. LogSum method

In this method log-sum-exp trick is implemented which is used to predict probabilities. The maximum among two values is calculated and then both are added using log-sum-exp trick.

```
def LogSum(self, logx, logy):
    # log sum exp trick
    if logx>logy:
        m=logx
    else:
        m=logy
    total=math.exp(logx-m)+ math.exp(logy - m)
    return  m+math.log(total)
```

1.2.2. PredictProb method

In this method probabilities are estimated for first 10 reviews in test data. Positive and negative probabilities are listed for positive and negative reviews respectively i.e predicted class probability is displayed.

```
def PredictProb(self, test, indexes):
    # predict the probabilities of reviews being positive (first 10 reviews in the test set)
    data = test.X.data
    indptr = test.X.indptr
    indices = test.X.indices
    vocabSize = self.data.vocab.GetVocabSize()
    probPos = (self.pos * 1.0) / (self.pos + self.neg)
    probNeg = (self.neg * 1.0) / (self.pos + self.neg)

    for i in indexes:
        index = indices[indptr[i]:indptr[i + 1]]
        posClass = math.log(probPos)
        negClass = math.log(probNeg)
        for k in range(0, len(index)):
            wordId = index[k]
            posClass += math.log((self.posDict.get(wordId, 0) + self.ALPHA) / (self.totalPos + (self.ALPHA * vocabSize)))
            negClass += math.log((self.negDict.get(wordId, 0) + self.ALPHA) / (self.totalNeg + (self.ALPHA * vocabSize)))

        posProb = posClass-self.LogSum(posClass,negClass)
        negProb = negClass-self.LogSum(posClass,negClass)

        #Print predicted probabilities of respective classes
        if posProb>negProb:
            print test.Y[i], +1.0, math.exp(posProb), test.X_reviews[i]
        else:
            print test.Y[i], -1.0, math.exp(negProb), test.X_reviews[i]
```

The following table illustrates the predicted probabilities of first 10 reviews in the test data for ALPHA as 0.5 and 10.0. Reviews are shuffled once so same reviews are evaluated.

ALPHA : 0.5

Given Class	Predicted Class	Probability	Reviews
-1.0	-1.0	0.99747	I saw this at my city's independent cinema - no surprise....
-1.0	-1.0	0.99999	I wasted 35 minutes of my life on this turkey before I .....
-1.0	-1.0	0.73310	This is a very low budget film, set in one location in a .....
-1.0	-1.0	0.99999	I rented a copy of this one from Netflix -- big mistake.....
-1.0	-1.0	0.99986	An updated version of a theme which has been done be...
-1.0	-1.0	1.0	I got this movie as a buy one get one deal at troma.com....
-1.0	-1.0	1.0	After some internet surfing, I found the "Homefront" ser..
1.0	1.0	0.99999	HOUSE CALLS was an amusing 1978 comedy about a wid..
-1.0	-1.0	0.99999	I would not recommend it whatsoever. It was like gettin...
-1.0	-1.0	0.90341	I rented this movie primarily because it had Meg Ryan in..

ALPHA : 10.0

Given Class	Predicted Class	Probability	Reviews
-1.0	-1.0	0.99813	I saw this at my city's independent cinema - no surprise....
-1.0	-1.0	0.99999	I wasted 35 minutes of my life on this turkey before I .....
-1.0	-1.0	0.82497	This is a very low budget film, set in one location in a .....
-1.0	-1.0	0.99999	I rented a copy of this one from Netflix -- big mistake.....
-1.0	-1.0	0.99041	An updated version of a theme which has been done be...
-1.0	-1.0	1.0	I got this movie as a buy one get one deal at troma.com....
-1.0	-1.0	0.99999	After some internet surfing, I found the "Homefront" ser..
1.0	1.0	0.99999	HOUSE CALLS was an amusing 1978 comedy about a wid..
-1.0	-1.0	0.99999	I would not recommend it whatsoever. It was like gettin...
-1.0	-1.0	0.87030	I rented this movie primarily because it had Meg Ryan in..

2. Perceptron

2.1. Vanilla Perceptron

The vanilla perceptron was implemented using EPOCH as hyperparameter. In this following method are implemented.

2.1.1. Train method

In this method perceptron parameters i.e. weights and bias are calculated which will be used to predict label of reviews.

```
def Train(self, X, Y):
    #Estimate perceptron parameters- weights and bias
    data = X.data
    indptr = X.indptr
    indices = X.indices

    for n in range(self.N_ITERATIONS):
        for i in range(len(Y)):
            index = indices[indptr[i]:indptr[i + 1]]
            a=self.b
            for k in range(0, len(index)):
                wordId = index[k]
                x = data[indptr[i] + k]
                a+=(self.weights.get(wordId,0.0)*x)

    #Update weights and bias
            if Y[i]*a <=0:
                for k in range(0, len(index)):
                    wordId = index[k]
                    x = data[indptr[i] + k]
                    self.weights[wordId]=self.weights.get(wordId,0.0)+(Y[i]*x)
                self.b+=Y[i]
    return
```

2.1.2. Predict method

In this method labels for IMDB reviews are predicted using weights and bias.

```
def Predict(self, X):
    #Predict label of reviews by perceptron classification
    data = X.data
    indptr = X.indptr
    indices = X.indices
    Y=[]

    for i in range(0,len(indptr)-1):
        index = indices[indptr[i]:indptr[i + 1]]
        a = self.b
        for k in range(0, len(index)):
            wordId = index[k]
            v = data[indptr[i] + k]
            a += (self.weights.get(wordId, 0.0) * v)
```

```
15.
16.         if a>=0:
17.             Y.append(+1.0)
18.         else:
19.             Y.append(-1.0)
20.
21.     return Y
```

The following table illustrates the accuracy on the IMDB dataset for various values of hyperparameter EPOCH. The maximum accuracy is attained at EPOCH=50.

EPOCH	ACCURACY (%)
1	74.196
10	86.100
50	86.184
100	86.180
500	86.152
1000	86.152

2.2. Parameter Averaging

In this section, vanilla perceptron is modified to implement parameter averaging.

2.2.1. ComputeAverageParameters method

In this method, average of the parameters from every iteration is used to classify test data.

```
1.  def ComputeAverageParameters(self):
2.      #Compute average parameters
3.      X=self.traindataX
4.      Y=self.traindataY
5.      data = X.data
6.      indptr = X.indptr
7.      indices = X.indices
8.      self.weights = {w: 0.0 for w in range(X.shape[1])}
9.      cachedWeights = {u: 0.0 for u in range(X.shape[1])}
10.     self.b = 0.0
11.     cachedBias = 0.0
12.     c=1.0
13.
14.     for n in range(self.N_ITERATIONS):
15.         for i in range(len(Y)):
16.             index = indices[indptr[i]:indptr[i + 1]]
17.             a=self.b
18.             for k in range(0, len(index)):
19.                 wordId = index[k]
20.                 x = data[indptr[i] + k]
21.                 a+=(self.weights.get(wordId, 0.0) * x)
```

```
22.
23.     #Update average parameters
24.             if (Y[i] * a) <=0:
25.                 for k in range(0, len(index)):
26.                     wordId = index[k]
27.                     x = data[indptr[i] + k]
28.                     self.weights[wordId] = self.weights.get(wordId, 0.0) + (Y[i] * x)
29.                     cachedWeights[wordId] = cachedWeights.get(wordId, 0.0) + (Y[i] * c
    * x)
30.                 self.b += Y[i]
31.                 cachedBias += (Y[i] * c)
32.             c += 1
33.
34.     self.b-=(cachedBias/c)
35.     self.weights = {key: self.weights[key] - (cachedWeights.get(key, 0.0)/c) for key in
     self.weights.keys()}
36.     return
```

The following table illustrates the accuracy on the IMDB dataset for various values of EPOCH. The maximum accuracy is attained at EPOCH=10. As compared to vanilla perceptron average perceptron attains higher accuracy for all the values of EPOCH.

EPOCH	ACCURACY (%)
1	83.836
10	87.184
50	87.072
100	86.888
500	86.340
1000	86.296

2.3. Features

In this section, 20 most positive and 20 most negative words in the vocabulary sorted by their weight are presented.

2.3.1. getWords method

The most positive and negative words are extracted from vocabulary by using argsort function of numpy.

```
1.  def getWords(self,vocab):
2.      values=self.weights.values()
3.      indices=np.argsort(values)
4.      c=1
5.      print "---------Positive words---------"
6.      for i in xrange(indices.size-1,indices.size-21,-1):
```

```
            print c,' ',vocab.GetWord(indices[i]),' ',self.weights.get(indices[i])
            c+=1

    c=1
    print "---------Negative words---------"
    for i in indices[0:20]:
        print c,' ',vocab.GetWord(i),' ',self.weights.get(i)
        c+=1
```

2.3.2. Top Positive Words

In this section, top 20 positive words are calculated by using weights from vanilla and average perceptron.

The following table illustrates top 20 positive words for vanilla perceptron with EPOCH as 500.

Count	Word	Weight
1	wonderfully	204
2	7/10	189
3	7	182
4	rare	170
5	refreshing	169
6	perfect.	162
7	subtle	155
8	amazing.	155
9	noir	153
10	excellent.	150
11	highly	150
12	perfect,	149
13	funniest	147
14	favorite	145
15	8	145
16	delightful	144
17	surprisingly	144
18	driven	142
19	captures	139
20	8/10	139

The following table illustrates top 20 positive words for average perceptron with EPOCH as 500.

Count	Word	Weight
1	wonderfully	200.6535958
2	7/10	182.926403
3	7	178.4257487
4	rare	166.7243001
5	refreshing	165.1751086
6	perfect.	157.7598265
7	subtle	153.3869543
8	amazing.	151.3327963
9	noir	149.1677679
10	highly	146.7521211
11	excellent.	145.7333547
12	perfect,	145.3483808
13	funniest	144.3544898
14	favorite	143.111058
15	8	141.7445868
16	surprisingly	141.1543495
17	delightful	141.0813612
18	driven	137.8365044
19	captures	136.9216547
20	8/10	135.9877102

2.3.3. Top Negative Words

In this section, top 20 negative words are calculated by using weights from vanilla and average perceptron.

The following table illustrates top 20 negative words for vanilla perceptron with EPOCH as 500.

Count	Word	Weight
1	poorly	-295
2	worst	-268
3	waste	-260
4	awful.	-223
5	fails	-220
6	boring	-196
7	disappointing	-193

8	lacks	-189
9	annoying	-183
10	terrible.	-183
11	4/10	-182
12	lame	-180
13	annoying.	-169
14	disappointment	-167
15	mildly	-162
16	badly	-161
17	weak	-161
18	/>4/10	-159
19	awful	-159
20	pointless	-157

The following table illustrates top 20 negative words for average perceptron with EPOCH as 500.

Count	Word	Weight
1	poorly	-290.3632417
2	worst	-267.9204972
3	waste	-259.2966419
4	awful.	-218.9446145
5	fails	-215.3333051
6	boring	-194.2974997
7	disappointing	-190.4272653
8	lacks	-185.7092635
9	annoying	-181.3020378
10	terrible.	-179.2478998
11	lame	-177.6706873
12	4/10	-175.8519476
13	annoying.	-165.1686566
14	disappointment	-163.3755129
15	mildly	-159.0392117
16	badly	-158.6270424
17	weak	-156.9824645
18	awful	-156.7171673
19	pointless	-155.9368313
20	dull	-153.686866