Name: HSU, Chia hong; Stid: 20562037

Data Cleaning

Before data cleaning:

Number of words	266282
Max sentence length	165
Number of sentences	20000
Number of vocab	34389
Top 10 most frequent words	['for', 'you', 'is', 'my', 'and', 'a', 'the', 'to', 'i', '@']

After data cleaning:

Number of words	264125
Max sentence length	156
Number of sentences	20000
Number of vocab	19269
Top 10 most frequent words	['for', 'is', 'you', 'my', 'and', 'it', 'a', 'the', 'to', 'i']

Data cleaning method steps:

1. Lowercase string
2. Eliminate words with containing digits (0-9) in it
3. Eliminate punctuation characters “:.?@;,&*-+’()” and any whitespaces.

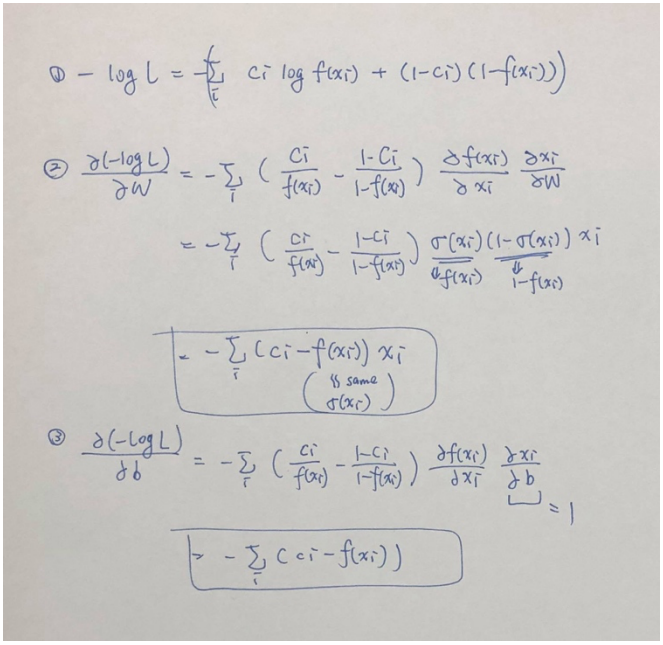
Feature Extraction

Input: Function feature_extraction_bow() takes 2 arguments, “revs”: list of (ground truth, cleaned text) pairs for all data, “word2idx”: map of key representing unique vocabs and value assigning unique indices.
Output: Function feature_extraction_bow() returns 2 things, “data array”: 2-d array with each sentence’s BoW vector stored inside, “label”: each sentence’s ground truth.

Normalization

Find mean, std-dev of each column in the 2-d array “data”. Then, rescale new data array as data = (data – mean)/std-dev. We choose column normalization since it shows the relative emphasis of the same BoW element with respect to the same BoW element in other sentences. If we choose row-wise normalization, the computed emphasis on certain words will be diluted by the weight of other words within the same sentence, which is apparently not our desire result.

Logistic Regression



What is the effect of the learning rate? What is the best learning rate you found?
Learning step, gradient descent step size.

When the learning rate is too big, our training accuracy improves faster. However, our development accuracy dropped significantly compared to lower learning rate.

[lr=10]

iteration: 1 , time {0:.2f} 8.253018856048584
    Training accuracy: 90.3687 %, cost: 90.3687
    Dev accuracy: 71.9250 %
logistic_regression.py:40: RuntimeWarning: divide by zero encountered in log
  loss = -(1.0/m) * (Y @ np.log(A).T + (1-Y) @ np.log(1-A).T)
logistic_regression.py:40: RuntimeWarning: invalid value encountered in matmul
  loss = -(1.0/m) * (Y @ np.log(A).T + (1-Y) @ np.log(1-A).T)
iteration: 2 , time {0:.2f} 9.093153715133667
    Training accuracy: 93.6813 %, cost: 93.6813
    Dev accuracy: 72.3250 %
…
iteration: 10 , time {0:.2f} 16.232189893722534
    Training accuracy: 97.0250 %, cost: 97.0250
    Dev accuracy: 70.7500 %

When the learning rate is too small, our training accuracy improves slower. However, our development accuracy dropped less significantly compared to higher learning rate.

[lr=0.01]

iteration: 1 , time {0:.2f} 6.6825950145721436
    Training accuracy: 89.9625 %, cost: 89.9625
    Dev accuracy: 74.4750 %
iteration: 2 , time {0:.2f} 8.162430047988892
    Training accuracy: 89.9750 %, cost: 89.9750
    Dev accuracy: 74.4750 %

…
iteration: 10 , time {0:.2f} 13.521360874176025
Training accuracy: 90.1625 %, cost: 90.1625
Dev accuracy: 74.5750 %

Best: lr=0.1 works fine.

What is the best accuracy you can get from the training set? How about in the development set?
Best training: about 95.5%. Best dev: about 75%.

Describe whether the data normalization helps. Why or why not?
[Without normalization result]
iteration: 1 , time {0:.2f} 9.828789710998535
Training accuracy: 62.4188 %, cost: 62.4188
Dev accuracy: 62.9000 %
iteration: 2 , time {0:.2f} 17.733616828918457
Training accuracy: 62.5187 %, cost: 62.5187
Dev accuracy: 63.0000 %
[With normalization result]
iteration: 1 , time {0:.2f} 9.041015863418579
Training accuracy: 90.2687 %, cost: 90.2687
Dev accuracy: 73.2500 %
iteration: 2 , time {0:.2f} 12.5433828830719
Training accuracy: 90.5687 %, cost: 90.5687
Dev accuracy: 73.5000 %
Yes it helps. Reason is that we get better accuracy in both training and developer set. Rescaling data significance among different features helps a smoother gradient descent, and therefore our loss function is easier to converge.

List at least 10 sentences in the dataset that your model gives you a wrong prediction. Explain why.
10134 “almost bedtime”, neg predicted pos
10167 “thanks for bursting my bubble”, neg predicted pos
10223 “aww, good luck paula!! please don't work too hard but i hope you have fun your new album is gonna be amazing! xx” neg predicted pos
10356 “URL - i love you, buck.” neg predicted pos
10368 “wonder if jon lost the net” neg predicted pos
10403 “hey! you just changed your default.” neg predicted pos
10496 “geez ur no fun are you” neg predicted pos
10498 “that's a lot of angst for a tuesday afternoon” neg predicted pos
11024 “new post! URL” neg predicted pos
11068 “rachel and jessy r making me work out thanks you guy” neg predicted pos
BoW cannot understand the reverse meaning of certain vocabs, for example, “not happy” is definitely negative but BoW captures “not” and “happy” separately, so this might result in positive because “not” could be a lighter word compared to “happy”.