CAP 6640 – Natural Language Processing

Homework 2

February 11, 2019

Due: in 7 days. Preferred typesetting system LaTeX

Answer all of the following questions. Keep in mind the following instructions: individual work, cite all references, document your code, type in all answers, and in mathematical expression, use standard notations.

1. Explain how ANN works and how multiple layers can be used for improving the accuracy.

   An ANN works by taking input vector x, where these inputs x_i are passed to a hidden layer. While in the hidden layer, each input x_i are multiplied by weight w_i and bias b is added to it to form a new vector h(x) for every h_i. The hidden layer h(x) can then be passed through an activation functions for the final output layer. This can be seen mathematically by:

   Hidden layer composed of Input, Weight and bias vector

   $$h_{w,b}(x) = (w^T x + b)$$

   Activation Layer

   $$a = f(h) = \frac{1}{1 + e^{-h}}$$

   Scoring function for final output:

   $$s = U^T a$$

   The loss functions job will be to determine future weights in the next layer. So if we have more hidden layers, the lose function will allows us to take more information from original given input. Adding more hidden layers would add more cost in calculations but again would be able to generalize more with test data since the granularity of the data would be finer than a smaller hidden layer ANN. Essentially adding more layers will capture non-linear features easier with the extra layers.

2. Explain ANN using matrix and sum-based representation.

   ANN will have an input layer x for x_i to x_n. The next layer is the hidden layer which is composed of inputs, weights, and biases. Each input is multiplied by a weight, summed together to be passed to each row of the hidden layer, and passed through an activation function to the next hidden layer or output. So if we had an input vector and wanted to determine the output of the activation layer we would calculate that by:

   $$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

   $$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

   In Matrix representation we take:

   $$z = Wx + b; HiddenLayer$$

$$a = f(z); Activation function$$

The Activation function is applied element wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

3. Explain what is the problem in named entities and how window classification can be used to solve the problem.

   The task of named entity recognition(NER) is to find and classify names. For example if the word is a name or a city. The problems in NER can be that words are ambiguous, it can be difficult to tell if something is a new entity, and can be hard to determine boundary of the entity. The window classification can be used to solve this problem by classifying a word in its context window of neighboring words. We could train a soft max classifier that would classify a center word based on its surrounding words of some window size.

4. Explain how back propagation works and what it is used for.

   In back propagation we have source nodes: inputs, interior nodes: operations, and edges that pass along the result of the operation. Back propagation consist of passing the gradients backwards through the interior nodes along the edges. At each node, there are two stream directions: left of the node is the downstream gradients and right of the node is the upstream gradients. Additionally each node has a local gradient where it is the rate of change of hidden layer h with respect to its previous layer z. And so node takes this upstream gradient with its own local gradients to pass long the downstream gradients to the previous layer. To determine the down stream gradient we multiply the local and the upstream gradients together using the chain rule. This can be seen visually:

$$\frac{\partial s}{\partial z} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}$$

   Where,
   $\frac{\partial s}{\partial z}$ is the Down Stream Gradients being sent to the previous layer of node N

   $\frac{\partial h}{\partial z}$ is the Local Gradient at the Node N

   $\frac{\partial s}{\partial h}$ is the Upstream Gradient from the forward layer.

   This process is used for updating the loss function which allows for more precise weights to be used, resulting in more precise accuracy.Back propagation helps to reduce the error with respect to the baseline by propagating the value of W. With this value we can minimize the expected output with actual output.

5. Explain why we need structures to analyze language dependency.

We need dependency structures because they inform us which words depend on which words or the rules of the language essentially. So this will help us understand sentence structure in any language. For example organizing words into starting words, combine words into a phrase, and combine phrases into bigger phrases. Humans have the ability to generate vast ideas or visions by joining words together into bigger elements to describe high level meanings. Not having an accurate dependency structure can give non accurate meanings in of phrases or words. So we will nee to know what type of words are connected each other in sentences.

6. Explain how n-gram technique is used for:

   (a) language modeling: Finds the probability for each subsequent word for a given stem to find the probability of the next word. So language modeling, the n-gram technique is used for predicting the next word given n words.

   $$P(word|previouswords) = \frac{P(word)}{P(previouswords)}$$

   This can also be translated to being:

   $$P(word|previouswords) = \frac{count(word)}{count(previouswords)}$$

   So for a given corpus, we would determine what words will come next given some n set of words. This could be used for text predictions in text messaging or typing.

   (b) feature representation of text
   In feature representation of text we can use ngrams to determine the probabilities by counting. We can find the frequencies of a word given a stem, and we can even normalize the probability as needed. This will allow us to develop up representations of our features with a probability and can use that as needed for our project goals.

7. Using python, implement n-gram approach starting from $n = 1$ to $n = 5$ for feature representation. Let $f_1 \dots f_k$ be the vectors representation of the n-gram features, whereby f_1 is bag of words, f_2 is the frequency of pairs of words, as defined for the n-gram model, and so forth. Using the n-gram representations, build and test a binary classifier on the three datasets attached (the datasets are sentiments; binary labels). Study the impact of n on the performance of the classifier (more details are below).

   Details. Evaluate your classifier on the three datasets, using the features extract with the n-gram model and 10-fold cross-validation. In reporting the results, calculate 7 metrics: false positive rate, false negative rate, true positive rate, true negative rate, accuracy, precision and recall, and report them in that order for each experiment. Report the results for each dataset, and for $n = 1 \dots 5$, **combined** and **separated**. That is, report the results for features representation of $f_1, f_2, \dots, f_5$ (5 experiments for each dataset; in total you'll have 15 experiments), and $[f1f2]$, $[f_1f_2f_3]$, $[f_1f_2f_3f_4]$, $[f_1f_2f_3f_4f_5]$ (4 experiments for each dataset; in total you'll have 12 experiments). Provide those 27 tables in your answer sheet.

   More information on the evaluation metrics is here: `https://en.wikipedia.org/wiki/Sensitivity_and_specificity`

```
import numpy as np
import pandas as pd
import string
from nltk import sent_tokenize, word_tokenize
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import confusion_matrix
import imp

#importing Data
data1 = '/Users/marctheshark/Documents/NLP/HW2/data/amazon_cells_labelle
data2 = '/Users/marctheshark/Documents/NLP/HW2/data/imdb_labelled.txt'
data3 = '/Users/marctheshark/Documents/NLP/HW2/data/yelp_labelled.txt'


#gathering the data and closing the file
def getfile(address):
    file = open(address, 'rt')

    data = file.readlines()
    file.close()

    return data
#processing the data from the txt file to get the sentiment
# labels, corpus, and the unique ngram tokens
def preprocessing(your_data):
    data = getfile(your_data)
    sentiment = []
    corpus = []
    #looping over the length of the data
    for i in range(len(data)):
        index = data[i]
        # print(index, 'i')

        #reverse looping through each index to find the label 0 or 1
        for j in reversed(range(len(index))):
            single_index = index[j]


            try:

                #if the single_index is 0 or 1 lets store it
                if int(single_index) <= 1:
```

```
                    sentiment.append(int(single_index))
                    #stop this from looping through
                    #the rest of the index
                    break
            except ValueError:
                w = 0
##Building sentences from the data and removing
##punctuation and lowering captilizations.
        words = word_tokenize(index)
        #print(words)

        words = [word.lower() for word in words]
        # removing any punctuation
        matrix = str.maketrans('', '', string.punctuation)
        removed = [word.translate(matrix) for word in words]
        filter_words = [word for word in removed if word.isalpha()]

        corpus.append(filter_words)

#building the unique word bag

        token=[]

        #findthing the unique ngram token for the given corpus
        for words in corpus:
            token.extend(words)
        unique_token = list(set(token))

    return sentiment, corpus, unique_token

#creating the ngram to compare with the ngram dictionary
def n_gram_GetTraining(data, n):

    nothing = preprocessing(data)
    labels, corpus, tokens = nothing
    corpus_ngram =[]

    #looking through each sentence in the corpus
    for i in range(len(corpus)):

        sentences = corpus[i]
        sentence_ngram =[]

        #looking at each word of the sentence
        for word in range(len(sentences)):
```

```
            #storing the ngram
            sentence_ngram.append(sentences[word:word+n])

        #storing all ngrams in their respective sentences
        corpus_ngram.append(sentence_ngram)


    #splitting into unique ngram tokens
    tokens = []

    for j in range(len(corpus_ngram)):

        index = corpus_ngram[j]

        for w in range(len(index)):

            next_index = index[w]

            if next_index not in tokens:
                tokens.append(next_index)

    tokens.sort() #sorting alphabetically
    #print(tokens)

    #creating the training data for this selection of n
    #looping over each unqiue token and then over each ngram
    # with respective of its index.
    encoded_data = np.zeros([len(corpus_ngram),len(tokens)])
    count = 0
    for z in range(len(tokens)):
        unique_token = tokens[z]
        for e in range(len(corpus_ngram)):
            if unique_token in corpus_ngram[e]:
                count += 1
                encoded_data[e,z] = count

    return encoded_data , labels


#                ----------<<<<< Classification >>>>>----------
#splitting the data so they can be equally distributed to the
# testing and training
```

```
def split_data(encoded_data, labels):

    zero =[]
    one = []
    zero_data = []
    one_data = []

    #looping through the encoded date
    for i in range(len(labels)):

        #if the label is 0 store it in its respective element
        if labels[i] == 0:

            zero.append(labels[i])
            zero_data.append(encoded_data[i])

        #if the label is 1 store it in its respective element
        elif labels[i] == 1:

            one.append(labels[i])
            one_data.append(encoded_data[i])

    return zero , zero_data , one , one_data

#classifying the data with the labels
def classifcation(encoded_data, labels , k,s_or_c,n_gram):

   zero, zero_data, one, one_data = split_data(encoded_data, labels)

   #initializing the 7 stats needed for assessment
    accuracy = []
    true_p = []
    true_n =[]
    false_p =[]
    false_n = []

    length = int(len(zero))
    folds =  int(length / (k))

    #print(folds)

    #looping through the kfolds for classifying
    for i in range(k):

        zero, zero_data, one, one_data = split_data(encoded_data, labels
```

```
#creating the testing data for the labels and observations
test_zero = zero_data[folds* i: folds*(i)+folds]
test_zero_l = zero[folds* i: folds*(i)+folds]
test_one = one_data[folds* i: folds*(i)+folds]
test_one_l = one[folds* i: folds*(i)+folds]

#checking size
'''
print(test_zero)
print(len(test_zero), 'zd')
print(len(test_one), 'od')
print(len(test_zero_l), 'zd1')
print(len(test_one_l), 'od1')
print(len(zero_data[folds*(i-1): folds*(i-1)+folds]))

  '''
#adding the data together
testing_data = np.concatenate((test_zero,test_one))
testing_labels = np.concatenate((test_zero_l,test_one_l))

#deteting the data that the was stored for the test data
del zero_data[folds* i: folds*(i)+folds]
del one_data[folds* i: folds*(i)+folds]
del zero[folds* i: folds*(i)+folds]
del one[folds* i: folds*(i)+folds]

#storing the rest of the data used into the
# new training variables
training_data = np.concatenate((zero_data,one_data))
training_labels = np.concatenate((zero, one))

#testing size
'''


print(testing_data.shape, 'testing shape')
print(testing_labels.shape, ' testing label shape')

print(training_data.shape, 'traing data')
print(training_labels.shape , 'train label')
  '''


#developing the classifier
classifier = LogisticRegression(solver='newton-cg'
,n_jobs=-1, max_iter=1000)
```

```
#calling the fit to train the model
classifier.fit(training_data,training_labels)

#predicting based on the model
prediction = classifier.predict(testing_data)

#print(prediction)
#print(testing_labels)

#variables to use as counters
everything = 0
count = 0
yes0 = 0
yes1 = 0
no0 = 0
no1 = 0
# true will be 0 positive will be 1 negative
        for i in range(len(prediction)):
    everything += 1
    if prediction[i] == testing_labels[i]:

        #predicted a 0 correctly true positive
        if prediction[i] == 0:
            yes0 +=1
            count += 1
        #predicted a 1 correct false positive
        else:
            yes1 +=1
            count += 1

    else:
        #predicted 0 instead of 1 true negative
        if prediction[i] == 0:

            no1 +=1
        #predict 1 instead of a 0 false negative
        else:
            no0 +=1
#storing values for stats
tp = yes0
fp = yes1
tn = no1
fn = no0
```

```
        #need to average up the tptnfn and so on and
        # all the other stats per 1 run of this function
        accuracy1 = (tp + tn) / (tp + tn + fp + fn)

        #storing the values to be averaged later
        true_p.append(tp)
        true_n.append(tn)
        false_p.append(fp)
        false_n.append(fn)
        accuracy.append(count/everything)

    #averaging values
    tp = (np.mean(true_p))
    tn = (np.mean(true_n))
    fp = (np.mean(false_p))
    fn = ((np.mean(false_n)))
    acc = np.mean(accuracy)

    #acc precision recall calculation
    precision = round((tp /(tp+fp)),5)
    accuracy = round(((tp+tn)/(tp+tn+fp +fn)),5)
    recall = round((tp / (tp + fn)),5)

    #formating for printing output
    if s_or_c == 's':
        if n_gram == 1:

            print("Results for uni-gram:" )
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                  precision)
        elif n_gram == 2:
            print("Results for bi-gram:")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                  precision)
        elif n_gram == 3:
            print("Results for tri-gram:")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                  precision)
        elif n_gram == 4:
            print("Results for quad-gram:")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
```

```
                precision)
        elif n_gram == 5:
            print("Results for penta-gram:")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                 precision)
    else:
        if n_gram ==1:
            print("Results for uni-gram & bi-gram::")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                 precision)
        elif n_gram == 2:
            print("Results for uni-gram & bi-gram & tri-gram:")
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                 precision)
        elif n_gram == 3:
            print("Results for uni-gram & bi-gram & tri-gram & quad-gram
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                 precision)
        elif n_gram == 4:
            print("Results for uni-gram & bi-gram & tri-gram & quad-gram
            print('TP', tp, 'TN', tn, 'FP', fp, 'FN', fn,
             'accuracy', accuracy, 'recall', recall, 'precision',
                 precision)
    print("")


    return tp,tn,fp,fn,accuracy,recall,precision


#    <<<<<Gathering the data for the necessary experiements >>>>>
#a function use to run all of the experiments
def get_results(data,number):
    #Generating printing outputs per dataset
    if number == 1:
        print("Generating results for Amazon dataset")
    elif number == 2:
        print("Generating results for IMDB dataset")
    else:
        print("Generating results for YELP dataset")
    #              ----------<<<<<Seperate >>>>>----------
    encode1, l1 = n_gram_GetTraining(data, 1)
```

```
    encode2, l2 = n_gram_GetTraining(data, 2)
    encode3, l3 = n_gram_GetTraining(data, 3)
    encode4, l4 = n_gram_GetTraining(data, 4)
    encode5, l5 = n_gram_GetTraining(data, 5)

    #                 ----------<<<<<Combined >>>>>----------
    c1 = np.hstack((encode1, encode2))
    c2 = np.hstack((c1, encode3))
    c3 = np.hstack((c2, encode4))
    c4 = np.hstack((c3, encode5))
    #calling the classification functions with respect to the data and
    classifcation(encode1, l1, 10, 's', 1)
    classifcation(encode2, l1, 10, 's', 2)
    classifcation(encode3, l1, 10, 's', 3)
    classifcation(encode4, l1, 10, 's', 4)
    classifcation(encode5, l1, 10, 's', 5)

    classifcation(c1, l1, 10, 'c', 1)
    classifcation(c2, l1, 10, 'c', 2)
    classifcation(c3, l1, 10, 'c', 3)
    classifcation(c4, l1, 10, 'c', 4)
    return

#               <<<<<Getting the results>>>>>
d1 = get_results(data1,1)
d2 = get_results(data2,2)
d3 = get_results(data3,3)
```

1 Results

The results for table 1,2,3 can be seen below:

To clarify True refers to 0 and 1 is False in the labeling. Assessing the amazon data set we can see an overall increase as we get more context. For example Table 1 penta-gram value for accuracy is 60.5 percent. It seems for using the combination of words on this data set it doesn't help the accuracy but it does increase the True positive rate or Recall.

Looking at Table 2 with the IMDB data we see quite the opposite of what took place in the first data set. The one word model is best separate model calculated for this data set. Its important to note for the separate trials for the n-gram $n \geq 3$ showed extremely low values for True positive rate. Thus resulting in the false negative value to be high because the model would not correctly identify the sentence in the corpus being true or 0. As the combined experiments increase in combination we see an increase in accuracy like we do in the amazon data set in Table 1.

From Table 3 with the Yelp Data set, we see again a negative increase in the accuracy as we increase n in the separate experiments. The true postive rate for the higher n-grams again arent

Table 1: Amazon

	TP	FP	TN	FN	Accuracy	Recall	Precision
uni-gram	37.9	40.1	12.1	9.9	.478	.758	.486
bi-gram	35.6	37.2	12.8	14.4	.484	.712	.49
tri-gram	35.3	33.7	16.3	14.7	.516	.706	.512
quad-gram	36.4	27.7	22.3	13.6	.587	.728	.57
penta-gram	36.6	26.1	23.9	13.4	.605	.732	.584
combined 1	39.4	41.2	8.8	10.6	.482	.788	.489
combined 2	39.1	40.1	9.9	10.9	.49	.782	.494
combined 3	38.7	40.2	9.8	11.3	.485	.774	.49
combined 4	38.7	39.6	10.4	11/3	.491	.774	.49425

Table 2: IMDB

	TP	FP	TN	FN	Accuracy	Recall	Precision
uni-gram	33.1	35.3	14.7	16.9	.478	.662	.484
bi-gram	22.4	38.5	11.5	27.6	.339	.448	.368
tri-gram	9.7	46.3	3.7	40.3	.134	.194	.173
quad-gram	5.7	47.7	2.3	44.3	.08	.114	.107
penta-gram	5.8	47.7	2.3	44.2	.081	.116	.108
combined 1	31.5	37.1	12.9	18.5	.444	.63	.46
combined 2	30.5	36.6	13.4	19.9	0.431	0.594	0.448
combined 3	29.7	36.6	13.4	20.3	.485	.774	.49
combined 4	28.9	36.4	13.6	21.1	0.425	0.578	0.44257

Table 3: YELP

	TP	FP	TN	FN	Accuracy	Recall	Precision
uni-gram	38.0	39.9	10.1	12.0	0.481	0.76	0.488
bi-gram	34.9	37.3	12.7	15.1	0.476	0.698	0.483
tri-gram	26.2	40.2	9.8	23.8	0.36	0.524	0.395
quad-gram	23.1	39.8	10.2	26.9	0.333	0.462	0.36725
penta-gram	21.8	40.0	10.0	28.2	0.318	0.436	0.353
combined 1	39.0	39.9	10.1	11.0	0.491	0.78	0.494
combined 2	39.5	39.1	10.9	10.5	0.504	0.79	0.503
combined 3	38.9	39.2	10.8	11.1	0.497	0.778	0.498
combined 4	38.4	38.7	11.3	11.6	0.497	0.768	0.498

as high and that seems to be what is the issue. When we increase the conncatenation for the combination experiments the accuracy hovers between .49 and .5

These results are interesting and lead me to suspect possibly issues in the code. I also would need to look at each dataset and compare how different each are in their use of language. It was

interesting that the IMDB data set took the longest to run, so again leads me to think there could be some issue with developing the n-grams. Which could help explain why the accuracy would decrease when n got bigger. One issue could be that when the classifer was trained it trained the model with all of the 0's and then all of the 1's. I'm not sure if this has issue with our results but I would like to see what the results are if we were to shuffle the data but still keeping even proportions in the training and testing sets.

2 references

Reference

1. Class Slides

2. "Albert Au Yeung," Generating N-grams from Sentences Python — Albert Au Yeung. [Online]. Available: http://www.albertauyeung.com/post/generating-ngrams-python/. [Accessed: 19-Feb-2019].

3. granprofacigranprofaci 3, dave mankoffdave mankoff 8, Franck DernoncourtFranck Dernoncourt 37.7k32195344, SpaceghostSpaceghost 4, GunjanGunjan 1, Joel CornettJoel Cornett 17.7k44269, r11r11 8714, JKCJKC 816530, and Yann DuboisYann Dubois 17125, "Computing N Grams using Python," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/13 n-grams-using-python. [Accessed: 19-Feb-2019].

4. $\backslash numpy.hstack, "scipy.stats.trim_meanSciPyv1.1.0Reference$ Guide. [Online]. Available: https://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.hstack.html. [Accessed: 19-Feb-2019].

5. "Python: How to Sort a List? (The Right Way)," Afternerd, 13-Jan-2019. [Online]. Available: https://www.afternerd.com/blog/python-sort-list/. [Accessed: 19-Feb-2019].

6. yemuyemu 5, rootroot 39k1379103, Nasir ShahNasir Shah 1, Jan-Philip GehrckeJan-Philip Gehrcke 19.1k46094, MeminMemin 655718, NikNik 18829, CatbuiltsCatbuilts 9061015, Prakhar AgarwalPrakhar Agarwal 1, Rohit MalhotraRohit Malhotra 7612, debodebo 11315, AllenAllen 9, and VladVlad 658521, "How do I get the row count of a Pandas dataframe?," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/15943769/how-do-i-get-the-row-count-of-a-pandas-dataframe. [Accessed: 19-Feb-2019].