Natural Language Processing Assignment 1

Marc Mailloux

February 27, 2019

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 the limitations of one-hot representation of words and the trade-off between count-based vs. probability-based representation of words.

One-hot vectors limitations lie with in the size of the data for processing purposes. For example if a corpus has 10 words in it, the the one-hot representation would then be encoded to a 10 by 10 matrix. So if we had a 10,000 or 100,000 word corpus the data would grow very large in a one-hot representation. Also through this representation we lose context of the words themselves.

Count based allows for fast training, used to obtain word similarity, and the frequency of each word in a given corpus. While probability based representation allows for scaling with the corpus, able to capture complex patterns beyond similarity, and words are weighted based on meaning or context. One main difference between count and probability based is that count is used on one corpus and probability based is over an entire population of corpus's. So for a smaller data set count based might be better while larger data containing multiple corpus's probability based would be better.

2. Explain (including mathematically) the operation of the followings techniques:

(a) Word2vec - Builds a dense vector for each word chosen so that it is similar to a vector of words that appear in similar contexts. In other words the algorithm goes through each position t in the text which predicts context words with in a given window size m, with a given center word w_j . This process will continue adapting the word vector to the optimized solution.

An example of this can be seen here: *I am going to the beach Molly* ***would*** *you like to join me?* , we choose our center word to be **would** , with a window size m=2, then we would essentially have four probabilities to calculate. From the given example we then have these probabilities from the word2vec model: $P(beach|would), P(Molly|would), P(you|would), P(like|would)$ This can be translated Mathematically:

$$P(w_{t-2}|w_t), P(w_{t-1}|w_t), P(w_{t+1}|w_t), P(w_{t+2}|w_t)$$

To calculate $P(w_{t+j}|w_t)$we will need to take the similarities of the chosen center word with respect to all outer words. Once we find the similarity we

use exponentiation to make all similarities positive and then divide over the entire vocabulary to give us a probability distribution. Mathematically this can be seen as:

$$P(context|centerword) = \frac{\exp(u_{context}^T \cdot v_{centerword})}{\sum_{contextword \in V} \exp(u_{contextword}^T \cdot v_{centerword})}$$

To further clarify: $u_{context}^T \cdot v_{centerword}$ is the dot product comparing in the context and center word. $\sum_{contextword \in V} \exp(u_{contextword}^T \cdot v_{centerword})$ normalizes over the entire vocabulary to generate a probability distribution.

These probabilities are then fed to calculate the likelihood function which then used in the objective function. **The Likelihood function:**

$$L(\theta) = \prod_{T=1}^{T} \prod_{-m \leqslant j \leqslant m} P(w_{t+j}|w_t; \theta) \, where \, j \neq 0$$

The objective function takes $L(\theta)$to find the negative log likelihood, this can be seen mathematically:

$$J(\theta) = -\frac{1}{T} \cdot logL(\theta) = -\frac{1}{T} \cdot \prod_{T=1}^{T} \prod_{-m \leqslant j \leqslant m} log(P(w_{t+j}|w_t; \theta)) \, ; where \, j \neq 0$$

By minimizing the objective function we are then maximizing the accuracy for the predictions. The objective function could also be viewed as the Loss function.

Mathematically this due to θ finding a lower and lower minima in the objective function $J(\theta)$ We do this by taking the derivative after each update and if the sign changes we now we found potentially the global minima. Stochastic Gradient descent help find the optimized solution which allows for θ jump through high inclines/declines that might not be the global minima on the objective function.

(b) Window classification - Uses a "window" of some size w around each word and is able to obtain syntactic and semantic information. So this method classifies a word in its context "window" of neighboring words. In the word2vec example the window was of size 2 meaning to word to the left and right of the center word is contained with in the window. Due to the size of the window can ultimately determine the context of the word representation. Again in our example: *I am going to the beach Molly would you like to join me?* if we are trying to define Molly with a window size of 3 our window vector would be:

$$X_{window} = [X_{to}, X_{the}, X_{beach}, X_{Molly}, X_{would}, X_{you}, X_{like}]$$

Based off this structure and other training data we could determine Molly is a Name of a Person from this context.

(c) Logistic regression - is modeled very similarly to Neural Networks. [1] Where a neuron of an ANN can be a binary logistic regression unit. This methodology works by estimating some training set x_i, y_i wherex_i are the inputs, word vectors, sentences, corpus, etc. The y_i are the labels of the data we are trying to predict things like named entities, center words given outer words or phrases identification like bullying or non-bullying. The mode has inputs being passed through to a hidden layer that has weights and some biases. This layer is then passed through an activation function and ultimately some output vector. Mathematically the Logistic Regression Model is given by:

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

Where f is a non-linear activation function, w is the weight, b is the bias, h is the hidden input and x is the input. In ANN this process is happening for every input at once.

(d) Soft max classification - is used to take our input vector and to normalize it over all inputs so that resulting vector will be a vector of probabilities that sum to 1. It must be noted that Logistic regression can use a soft max function and it results in a linear boundary which can be noted from the linear relationship of β_0 and β_1. The soft max function is represented mathematically by:

$$P(y|x) = \frac{exp^(W_y \cdot x)}{\sum_{c=1}^{C} W_y \cdot x}$$

Breaking down this equation we have our Weight vector W_y which is used to determine the decision boundary. So for each training set (x,y) we want to maximize the the probabilities of the correct labels y_i by changing their weights. *In simpler terms: the soft max function maps arbitrarily values to x_i to a probability distribution p_i. "Max" because it increases the probabilities of largest x_i and "soft" because it still assignments some probability to a smallerx_i unlike other methodology like Lasso.* A direct example of this could be used in Named Entity Recognition where to find P(center word — surrounding words). A drawback of soft max is that is gives only linear decision boundaries and as a result is inflexible. A solution to this is to use a Neural Network Classifier to become more flexible with a non-linear boundary.

(e) Stochastic gradient descent **- Add about theta going down slope and what happens when the sign is flipped (it found the minima)** this methodology performs a parameter update for each training example and its label: (x_i, y_i) using the objective function $J(\theta)$. [3] The main reason to constantly update is to be able to make the necessary jump from local to global minima. So for each example we are calculating :

$$P(context|centerword) = \frac{\exp(u_{context}^T \cdot v_{centerword})}{\sum_{contextword \in V} \exp(u_{contextword}^T \cdot v_{centerword})}$$

and updating after each observation. Another reason to update after set is because it is computationally expensive to update after the entire data set and sometime might not be able to find global minima. We are able to establish an optimized global minima after a steady state or number of executes. has been reach.

3. The followings can be applied to improve representation. Explain how they work.

(a) Meaning encoding can help improve the representation of words by rationing the co-occurrence probabilities where the label could be Large or Small for the given input. Mathematically we can use the Log-bi-linear model: $w_i \cdot w_j = log \cdot P(i|j)$ or with vector differences: $w_x \cdot (w_i - w_j) = log \frac{P(x|i}{P(x|j)}$ With the co-occurrence matrix we can take a known dictionary and compare our corpus relatively easily.

(b) Distribution truncation The main is idea is that a words meaning is given by the words that frequently appear close-by with in a fixed sized window giving a distribution.[s] Using this method we cut of the ends of the distribution to push the outliers. In the end we will remove the words that are uncommon or very common. In count based methods high or low frequencies just tell us they are used alot or not at all, which isnt useful.

(c) Limited memory (e.g., smaller window size in window-based representation). Generally speaking the window size should roughly be between: 5-10. [s] Going any smaller and we can fail to grasp the actual context and the larger the window size the more likely we will lose the quality of the context. One problem in the window-based representation is that it can create sparse data where eighty to ninety percent of the data could be zero's. A solution to this is to store most of the important info into a fixed small number of dimensions : dense vector.Having a dense vector helps reduce the dimensions since only the important words are in the matrix. Another way

to reduce the dimensionality is we can use Singular Value Decomposition, factor X into UST^T such that U and V are orthogonal.

(d) Scaling can help the count of X, but removing the words that are too frequent like: the, has, he, her,they,them,etc.

4. Evaluation is challenging in NLP. Explain two methods for evaluation and explain their limitations. The primary methods use for evaluation are are intrinsic(internal) and extrinsic(external). *Intrinsic* evaluation's aim is to look into sub components, extract, and evaluate, to determine how individual components work together. This can be fast to compute and helps understand the given system. If the tasks relevancy isn't properly established then the results could not be clear. *Extrinsic* looks at the the task end to end, which can take a long time to compute. It can become unclear if a subsystem is the problem or the iteration of the algorithm. The solution would be to tackle one subsystem at a time and then update to see if the problem was fixed, which again could take a rather long time to fix.

5. Write a python code to implement Word2vec from scratch. Given a file of words (in a long phrase; each token in the file is a word) output a matrix of size $w \times n$, where w is the window size and n is the number of words in the input file. While generating the output making sure of the following rules:

(a) use decimal precision of $5(i.e..0.00001)$
(b) tab character as a separator for the matrix elements
(c) write the output in a file named out.txt

```
import numpy as np
import pandas as pd
import imp
from nltk import sent_tokenize, word_tokenize
import string
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from decimal import *
```

```
import sys


#creating a function to open the file and close it to store the data
def getfile(address):
    file = open(address, 'rt')
    data = file.read()
    file.close()

    return data

#different datasets to play with
mylocation = '/Users/marctheshark/Documents/NLP/HW1/sample.txt'
#mylocation = '/Users/marctheshark/Documents/NLP/HW1/shakespeare.txt'
d = getfile(mylocation)

#print(d)
#creating the matrix to output a n x m matrix
def create_W(address, window):
        #gathering the data
    data = getfile(address)



    # splitting data into a variable called words to tokenize
    sentences = sent_tokenize(data)

    #sentences = sentences[:10]
    #creating a corpus from that lines of words/sentences
    corpus= []

    for i in range(len(sentences)):
        #turning each sentence into a words
        t = word_tokenize(sentences[i])
        t = [word.lower() for word in t]
        #removing any punctuation
        matrix = str.maketrans('', '', string.punctuation)
        removed = [word.translate(matrix) for word in t]
        filter_words = [word for word in removed if word.isalpha()]
        final_words = filter_words
        #removing stop words
```

```
    #sw = stopwords.words('english')
    corpus.append(final_words)
    #final_words = [a for a in filter_words if not a in sw]



    #creating a vector with all unique words in the corpus
token =[]
for words in corpus:
        token.extend(words)
unique_token = list(set(token))
#print(unique_token)
#print(len(unique_token))
#print(len(token))


#creating the matrix size n by n
matrix = np.zeros([len(unique_token), len(unique_token)])
#print(matrix)
#print(len(unique_token))

#for each token
for token in range(len(unique_token)):
    #print(unique_token[token])
    #looking at every sentence
    for sentence in range(len (corpus)):
        data_sentences = corpus[sentence]
        print(data_sentences)

        #setting counters to track
        count = 0
        tracking = 0

        #looking at every word in the sentence
        for word in range(len(data_sentences)):
            center_word = data_sentences[word]
            tracking += 1
                            #defining the right and left windows
            right_window = data_sentences[word+1:word + window+1]
            left_window = data_sentences[word - window:word]
            #print('l',left_window,center_word,'c', right_window ,'r
```

```
                    #if the token is in the window of the center word
                if unique_token[token] in left_window:

                    count += 1
                if unique_token[token] in right_window:
                    count += 1
                avg = count / tracking

                matrix[token,word] = round(count,5)
    print(len(unique_token))

    #I wasnt 100% sure how to create the n x w matrix but here is the co
    '''
    la = np.linalg
    U, s , Vh, = la.svd(matrix,full_matrices= False)

    print(U)
    '''

    return matrix


def one_shot_encoder(data):

    encoded_data=[]
    label_encoder = LabelEncoder()

    for i in range(len(data)):
        #print(i)

        label_encoder.fit(data)
        newval_encoded = label_encoder.transform(data)
        onehot_encoded =  np_utils.to_categorical(newval_encoded)
        encoded_data.append(onehot_encoded)
    return encoded_data

#a = one_shot_encoder(d)
#telling the system that we will have script outputs
script = sys.argv[0]
location = sys.argv[1]
input_window = int( sys.argv[2])
```

```
#location = mylocation
#input_window = 2
#print(input_window)
calling the variable cw that uses the terminal inputs
cw = create_W(location,input_window)
#cw[0].apply(',  '.join)
import math

#print(cw)
#turning the array into a string
cw = '/'.join(map(str, cw))
#outputting the file out.txt
outFile = open('out.txt', 'w')

outFile.write(cw)
outFile.close()
```

References: 1: https://web.stanford.edu/ jurafsky/slp3/7.pdf

Code:

1. https://norvig.com/ngrams/
2. https://machinelearningmastery.com/clean-text-machine-learning-python/
3. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
4.https://nathanrooy.github.io/posts/2018-03-22/word2vec-from-scratch-with-python-and-numpy/