Text Analytics HW2 - Finn Qiao

October 26, 2019

0.1 Text Analytics - HW 2

0.2 Finn Qiao

0.3 1. Implementation of different CBOW and Skip-gram models

```
In [39]: import pandas as pd
         import matplotlib
         import matplotlib.pyplot as plt
         import seaborn as sb
         from gensim.models import Word2Vec
         from nltk.corpus import stopwords
         from nltk.tokenize import word_tokenize
         from nltk.stem.snowball import SnowballStemmer
```

```
In [40]: df = pd.read_csv('/Users/finn/Downloads/amazon-fine-food-reviews/Reviews.csv')
```

```
In [41]: df.head(2)
```

```
Out[41]:    Id   ProductId          UserId ProfileName  HelpfulnessNumerator  \
         0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian                     1
         1   2  B00813GRG4  A1D87F6ZCVE5NK      dll pa                     0

            HelpfulnessDenominator  Score        Time                Summary  \
         0                       1      5  1303862400  Good Quality Dog Food
         1                       0      1  1346976000      Not as Advertised

                                                         Text
         0  I have bought several of the Vitality canned d...
         1  Product arrived labeled as Jumbo Salted Peanut...
```

```
In [42]: df['Text'][1]
```

```
Out[42]: 'Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small si
```

```
In [43]: # remove symbols and numbers
         import re
         def cleaned(x):
             return re.sub(r'[^a-zA-Z ]',' ',x)
```

```
df['text_cleaned'] = df['Text'].apply(cleaned)

# lower case transform
df['text_cleaned'] = df['text_cleaned'].str.lower()

# stem and filter out stop words

stemmer = SnowballStemmer('english')
stop_words = set(stopwords.words('english'))

def wordfilter(string, filtwords):
    filtered = []
    tokens = word_tokenize(string)
    for word in tokens:
        if word not in filtwords:
            filtered.append(stemmer.stem(word))
    return filtered

for item, row in df.iterrows():
    df.at[item, 'text_cleaned'] = wordfilter(row['text_cleaned'], stop_words)
```

```
In [44]: df[['Text','text_cleaned']][:2]
         print(df['text_cleaned'][1])
```

```
['product', 'arriv', 'label', 'jumbo', 'salt', 'peanut', 'peanut', 'actual', 'small', 'size',
```

```
In [45]: # base model with all default parameters
         model_base = Word2Vec(df['text_cleaned'])

In [46]: model_base['peanut']
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning
  """Entry point for launching an IPython kernel.
```

```
Out[46]: array([-0.96061814, -2.5470355 ,  0.47732458, -3.2855825 ,  1.5646207 ,
                -1.4697863 ,  1.405279  ,  0.45318598, -2.8434055 ,  0.00930381,
                 0.52161384,  1.8094522 , -0.01619773,  1.0487529 , -0.07392865,
                -2.1169996 ,  2.1019816 ,  2.6989503 ,  0.8351442 , -1.3653778 ,
                -2.0431027 , -0.61818767,  0.20307854,  2.1668055 , -2.1621969 ,
                 1.1635364 ,  0.9542173 ,  0.92139494, -4.118723  ,  0.63691616,
                -2.399269  ,  0.43543836,  0.5743402 , -0.96405   ,  0.543317  ,
                -1.1213058 ,  2.9285383 ,  0.3826324 ,  0.7615425 ,  1.6145648 ,
                -1.5099298 ,  3.4179177 , -4.099808  ,  0.99413687,  1.383523  ,
                -2.2991269 ,  0.48511776, -0.13238025,  0.7414131 , -2.4375052 ,
                 3.5757852 , -5.8970757 ,  0.5406961 , -3.5622094 ,  0.20784187,
                 2.5606496 ,  2.5368028 , -0.40096894,  2.2933888 , -1.3364791 ,
                -0.7300951 , -0.73663986,  0.32252777,  0.58058625, -2.1335118 ,
```

```
             -0.96290153,  1.0450556 , -2.0887127 ,  6.347577  ,  1.4676281 ,
             -0.9922591 , -1.5251837 , -2.1877568 ,  1.2209108 , -1.0709122 ,
             -1.5383404 ,  1.5912632 ,  2.0822806 , -2.8919606 ,  1.6814653 ,
             -0.03178216, -0.41509455, -0.4948197 , -0.7211618 , -1.3816364 ,
              1.6435943 , -1.2333945 , -0.07880394, -3.979678  , -3.2428255 ,
             -0.18529986, -1.5009769 ,  0.59417874,  0.7418685 , -2.6063294 ,
              1.5369993 , -1.5209758 , -2.6312735 , -2.8596246 , -3.0852969 ],
            dtype=float32)
```

In [47]:
```python
model_base.most_similar('caviar')[:5]
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning
  """Entry point for launching an IPython kernel.
```

Out[47]:
```
[('lumpfish', 0.7236908078193665),
 ('roe', 0.7017508745193481),
 ('sturgeon', 0.6961816549301147),
 ('malossol', 0.6563703417778015),
 ('capelin', 0.6562642455101013)]
```

In [48]:
```python
# additional models with changed parameters

# skip gram base model
model_skip = Word2Vec(df['text_cleaned'], sg=1)

# CBOW with a larger window and larger min_count
model_large_cbow = Word2Vec(df['text_cleaned'], window=30, min_count = 30, sg=0)

# skip gram with a larger window and larger min_count
model_large_skip = Word2Vec(df['text_cleaned'], window=30, min_count = 30, sg=1)

# CBOW with a smaller window and smaller min_count
model_small_cbow = Word2Vec(df['text_cleaned'], window=2, min_count = 2, sg=0)

# skip gram with a smaller window and smaller min_count
model_small_skip = Word2Vec(df['text_cleaned'], window=2, min_count = 2, sg=1)
```

In [49]:
```python
all_models = [model_base, model_skip, model_large_cbow, model_large_skip, model_small_

pd.DataFrame([[tup[0] for tup in model.most_similar('caviar')[:5]] for model in all_mo
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: DeprecationWarning
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[49]:

	0	1	2	3	4
model_base	lumpfish	roe	sturgeon	malossol	capelin
model_skip	sturgeon	lumpfish	roe	malossol	osetra

```
model_large_cbow       roe    beluga            sockey   russian   oyster
model_large_skip       roe      keta  gourmetfoodstor   russian   salmon
model_small_cbow       roe  lumpfish              keta  sturgeon    sprat
model_small_skip  lumpfish  sturgeon               roe  malossol  capelin
```

As this is a fine foods review dataset, the first word we use to judge the different models qualitatively is 'caviar'.

The base model performs relatively well with the default parameters and CBOW implementation. Lumpfish is a relatively cheaper and available type of caviar but 'sturgeon' and 'roe' are words that almost exactly describe what caviar is.

When all model implementations are compared, there doesn't seem to be a huge difference in the top similar words to 'caviar'.

When the window for words is increased and the minimum word count for words is increased, there are more words that aren't as immediately applicable that show up. For example, oyster and salmon are common words that would exceed the minimum word count but might not be immediately applicable to caviar.

Meanwhile, reducing the window and minimum word count yields hyperspecific words that relate very well to caviar like 'keta' and 'malossol'.

The effect of skip gram models is most apparent when the window size is large. The large skip gram model yields similar tokens like gourmetfoodstor and salmon which are not seen in the other implementations.

```
In [59]: pd.DataFrame([[tup[0] for tup in model.most_similar('beef')[:5]] for model in all_mode
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning
  """Entry point for launching an IPython kernel.
```

```
Out[59]:                            0        1        2        3        4
         model_base            turkey  chicken     meat  poultri   jerkey
         model_skip           chicken    jerki   turkey    nesco     meat
         model_large_cbow  stroganoff  chicken  bullion  poultri  ostrich
         model_large_skip       jerki   turkey  chicken    beefi     meat
         model_small_cbow      turkey  chicken     meat     pork     fowl
         model_small_skip     chicken   turkey   jerkey    jerki     pork
```

```
In [58]: pd.DataFrame([model.most_similar('beef')[:5] for model in all_models], index = ['model
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: DeprecationWarning
  """Entry point for launching an IPython kernel.
```

```
Out[58]:                                             0  \
         model_base            (turkey, 0.7575222849845886)
         model_skip           (chicken, 0.7951224446296692)
         model_large_cbow  (stroganoff, 0.709730327129364)
         model_large_skip       (jerki, 0.7749670743942261)
         model_small_cbow      (turkey, 0.7531991600990295)
         model_small_skip     (chicken, 0.7986695766448975)
```

```
                                            1                            2  \
model_base        (chicken, 0.7324066162109375)    (meat, 0.7079607248306274)
model_skip          (jerki, 0.7668876647949219)  (turkey, 0.7523636817932129)
model_large_cbow  (chicken, 0.6739753484725952)  (bullion, 0.660982608795166)
model_large_skip   (turkey, 0.7477177381515503)  (chicken, 0.743026852607727)
model_small_cbow  (chicken, 0.7388721108436584)    (meat, 0.6888665556907654)
model_small_skip   (turkey, 0.7917496562004089)  (jerkey, 0.7655914425849915)

                                            3                            4
model_base        (poultri, 0.6909288763999939)  (jerkey, 0.6680481433868408)
model_skip          (nesco, 0.7369696497917175)    (meat, 0.7326550483703613)
model_large_cbow  (poultri, 0.6266921162605286)  (ostrich, 0.621371865272522)
model_large_skip    (beefi, 0.7388867139816284)    (meat, 0.7284117937088013)
model_small_cbow     (pork, 0.6174705624580383)    (fowl, 0.6131404638290405)
model_small_skip    (jerki, 0.7493690252304077)    (pork, 0.7479978799819946)
```

It was also interesting to note that cosine similarity for skip gram is generally slightly higher than that returned by CBOW with the same hyperparameters.

For a token that is much more common like 'beef', there isn't a noticeable difference between skipgram and CBOW implementations for the default hyperparameters and the reduced window size and reduced minimum count. However, it is interesting to note that for a more common word like 'beef', the most similar words to 'beef' for the CBOW model are rarer words like 'stroganoff' and 'ostrich'.

0.4 2. Skip-gram vs. Elmo vs. Bert

Recent advances in NLP has resulted in multiple new techniques by which similarities between words are calculated by representing words as vectors.

A few of the most prominent methods include the skip gram, elmo, and bert methods. Each of these methods offers up their respective advantages detailed in the below sections.

1. Contextual understanding
   - For skip gram, there is no contextual understanding of the words. The word 'play' for example would have the same vector representation no matter what sentence it is used in and what words there are around it.
   - For both Elmo and Bert, they include deep contextual understanding of the words with relation to the words around them and type of usage. Both Elmo and Bert have bi-directional representations which help them to create context for each use case of the word.
2. Computational efficiency
   - Bigrams and phrases for the skip gram model are selectively chosen to not use too much storage and compute. The subsampling of frequent words not just helps with better representing under-represented words but also helps with efficiency as well. Furthermore, there are no dense matrix multiplications used, making training extremely efficient.

- For Elmo, there is a constant need to balance overall language model perplexity with model size and computational requirements while maintaining a purely character based input representation. This has led to a halving of all embedding and hidden dimensions from the single best model.

3. Explainability

   - Skip gram wins out on explainability in terms of ease of visualization due to the nature of linear representations of words. This is best shown in the famous king - man = queen example. The linear structure makes analogical reasoning easy. It is also easy to plot out different word clusters.
   - For Elmo, there are 3 layers of representations for each input token, and there is added complexity that comes at the expense of being able to explain a token with a combination of these layers. The higher levels capture context dependent aspects of word meaning and lwer levels captures aspects of syntax. It is certainly more granular and provides better results but could be harder to explain.
   - Similarly for Bert, the masked language model is an additional predictive layer not used in other models. The ability to capture phrases and pairs of sentences in a single token sequence adds to model performance but could be hard for explainability at scale.

4. Out of vocabulary

   - Skip gram can only represent words and tokens within the training vocabulary.
   - Other models are character based and could help with out of vocabulary instances. Furthermore, selected phrases or sentence pairs can be represented as tokens.

5. Customizability and ease of use

   - Skip gram is very easy to use out of the box and a representation can be trained easily. You can also be highly case specific for hyperparameters and tune the model as you wish for the use case.
   - Elmo is pretrained on a large scale and can be easily incorporated into a wide range of existing neural NLP architectures. For domain specific use, the biLM can be fine tuned on domain sepcific data.
   - There is a minimal different between pre trained architecture and final downstream architecture and the same pre trained model can be used by a variety of tasks.