Homework2_Lab2_Rosenberg

October 25, 2019

Link to repo & branch: https://github.com/rrosenb1/rrosenb1_msia490_2019/tree/homework2

Remember to: - Submit .py file - If there are multiple files in the repo, include a README

[203]:
```
import os
import gensim
import string
import nltk
from nltk.stem.porter import PorterStemmer
from string import digits
import itertools
```

[204]:
```
# nltk.download('stopwords')
```

1 Lab 2 Assignment

[205]:
```
# Working with full dataset from last lab assignment
def retrieve_data(end_len):
    path = '/Users/rachelrosenberg/MSiA/490 - Text Analytics/20news-18828/'
    textdata = []
    files_parsed = 0

    for folder in os.listdir(path):
        try:
            for filename in os.listdir(str(path + folder)):
                with open(str(path + folder + '/' + filename), 'r') as f:
                    text = f.read()
                    textdata.append(text)
                    files_parsed += 1
        except:
            pass

    print(files_parsed, "files parsed.")
    print("Returning first", end_len, "files.")

    return(textdata[:end_len])
```

```
textdata_raw = retrieve_data(7000)
print(len(textdata_raw))
```

```
7522 files parsed.
Returning first 7000 files.
7000
```

[206]:
```
# Clean & normalize dataset
# normalization (e.g. convert to lowercase, remove non-alphanumeric chars,␣
 ↪numbers,
from nltk.corpus import stopwords

textdata = textdata_raw

def strip(textdata):
    textdata_stripped = []

    for l in textdata:
        # split into words by white space
        words = l.split()
        # remove punctuation from each word
        table = str.maketrans('', '', string.punctuation)
        textdata_stripped.append([w.translate(table) for w in words])

    return textdata_stripped

def to_lower(textdata):
    textdata_lower = []

    for l in textdata:
        # convert to lower case
        textdata_lower.append([word.lower() for word in l])

    return textdata_lower

def rm_stopwords(textdata):
    # filter out stop words
    words = []
    stop_words = set(stopwords.words('english'))

    for l in textdata:
        words.append([word for word in l if not word in stop_words])

    return words

def rm_numbers(textdata):
    words = []
```

```
    remove_digits = str.maketrans('', '', digits)

    for l in textdata:
        words.append([word.translate(remove_digits) for word in l])

    return words

def stem_words(textdata):
    words = []
    porter = PorterStemmer()

    for l in textdata:
        words.append([porter.stem(word) for word in l])

    return words

textdata = strip(textdata)
print("Removed punctuation. New length =", len(textdata))
textdata = to_lower(textdata)
print("Converted to lowercase. New length =", len(textdata))
textdata = rm_stopwords(textdata)
print("Removed stopwords. New length =", len(textdata))
textdata = rm_numbers(textdata)
print("Removed numbers. New length =", len(textdata))
textdata = stem_words(textdata)
print("Stemmed words. New length =", len(textdata))

print(textdata[0:2])
```

```
Removed punctuation. New length = 7000
Converted to lowercase. New length = 7000
Removed stopwords. New length = 7000
Removed numbers. New length = 7000
Stemmed words. New length = 7000
[['hmcsbrownedu', 'harri', 'mamayski', 'subject', 'heil', 'hernlem', 'articl',
'aprncsuedu', 'hernlemchessncsuedu', 'brad', 'hernlem', 'write', 'lebanes',
'resist', 'forc', 'deton', 'bomb', 'isra', 'occup', 'patrol', 'lebanes',
'territori', 'two', 'day', 'ago', 'three', 'soldier', 'kill', 'two', 'wound',
'retali', 'isra', 'israeliback', 'forc', 'wound', '', 'civilian', 'bombard',
'sever', 'lebanes', 'villag', 'iron', 'isra', 'govern', 'justifi', 'occup',
'lebanon', 'claim', 'necessari', 'prevent', 'bombard', 'isra', 'villag',
'congratul', 'brave', 'men', 'lebanes', 'resist', 'everi', 'isra', 'son',
'place', 'grave', 'underlin', 'moral', 'bankruptci', 'israel', 'occup', 'draw',
'attent', 'isra', 'govern', 'polici', 'reckless', 'disregard', 'civilian',
'life', 'brad', 'hernlem', 'hernlemchessncsuedu', 'nice', 'three', 'peopl',
'murder', 'bradli', 'overjoy', 'hear', 'death', 'middl', 'east', 'jewish',
'arab', 'death', 'feel', 'sad', 'hope', 'soon', 'stop', 'appar', 'view',
```

```
'point', 'accept', 'peopl', 'like', 'bradli', 'hernlem', 'disgust', 'harri'],
['waldocybernetcsefauedu', 'todd', 'j', 'dicker', 'subject', 'israel', 'expans',
'ii', 'abzvirginiaedu', 'andi', 'beyer', 'write', '', 'first', 'never', 'said',
'holocaust', 'said', '', 'holocaust', 'im', 'ignor', 'holocaust', 'know', '',
'nazi', 'germani', 'peopl', 'mayb', 'includ', 'uh', 'oh', 'first', 'sign',
'argument', 'without', 'meritth', 'state', 'one', 'qualif', 'area', 'know',
'someth', 'nazi', 'germani', 'show', 'dont', 'shut', 'simpl', '', 'dont',
'think', 'suffer', 'jew', 'wwii', '', 'justifi', 'crime', 'commit', 'isra',
'govern', '', 'attempt', 'call', 'civil', 'liberterian', 'like', 'antisemet',
'', 'appreci', 'jew', 'suffer', 'wwii', 'belov', 'perish', 'tortur', 'suffer',
'second', 'namecal', 'direct', 'civillibertarian', 'gener', 'namedrop', 'fanci',
'sound', 'polit', 'term', 'yet', 'anoth', 'attempt', 'cite', 'qualif', 'order',
'obfusc', 'glare', 'unprepared', 'argument', 'go', 'back', 'minor', 'junior']]
```

[207]:
```
# Output as a text file with one document per line
f = open("lab2results_Rosenberg.txt","w+")

for doc in textdata[0:10]:
    f.write(' '.join(doc))
    f.write('\n\n')

f.close()
```

[208]:
```
# Preview results doc
f = open("lab2results_Rosenberg.txt","r+")
output = []

for line in f:
    output.append(line)

print(output[0])
```

```
hmcsbrownedu harri mamayski subject heil hernlem articl aprncsuedu
hernlemchessncsuedu brad hernlem write lebanes resist forc deton bomb isra occup
patrol lebanes territori two day ago three soldier kill two wound retali isra
israeliback forc wound  civilian bombard sever lebanes villag iron isra govern
justifi occup lebanon claim necessari prevent bombard isra villag congratul
brave men lebanes resist everi isra son place grave underlin moral bankruptci
israel occup draw attent isra govern polici reckless disregard civilian life
brad hernlem hernlemchessncsuedu nice three peopl murder bradli overjoy hear
death middl east jewish arab death feel sad hope soon stop appar view point
accept peopl like bradli hernlem disgust harri
```

[ ]:

2 Homework 2 Assignment

2.1 Homework 2 Part 1

1. After completing the lab assignment, experiment with 2 or more sets of word2vec model parameters, for example, different embedding sizes, CBOW vs. skip-gram models, etc. Provide a paragraph or two of qualitative evaluation of your embeddings (no need to provide a quantitative evaluation). For example, you can evaluate the embeddings by hand-picking ~10 words and manually reviewing/evaluating their closest neighbors in terms of cosine similarity.

[186]:
```
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec
```

First model uses skip-gram and softmax tuning

[187]:
```
model1 = gensim.models.Word2Vec(
        textdata,
        size = 200,
        sg = 1, # use skip-gram
        hs = 1, # use hierarchical softmax
        iter = 20)
```

[188]:
```
w1 = 'polit' # stemmed version of 'politics'
model1.wv.most_similar(w1)
```

[188]:
```
[('societi', 0.44188982248306274),
 ('crux', 0.40291404724121094),
 ('klux', 0.38458943367004395),
 ('agenda', 0.38446104526519775),
 ('ideolog', 0.3736516833305359),
 ('parti', 0.3689371347427368),
 ('individu', 0.3670465350151062),
 ('keithccocaltechedu', 0.36505645513534546),
 ('invol', 0.358116090297699),
 ('hypocrisi', 0.35464829206466675)]
```

[189]:
```
w2 = 'thousand'
model1.wv.most_similar(w2)
```

[189]:
```
[('hundr', 0.6938508749008179),
 ('riflemen', 0.4558294117450714),
 ('twenti', 0.4538172781467438),
 ('thirti', 0.4189767837524414),
 ('year', 0.4153338074684143),
 ('million', 0.4071618616580963),
 ('bitli', 0.4050976037979126),
 ('refuge', 0.3979335427284241),
 ('erzincan', 0.392652690410614),
```

```
 ('acr', 0.3860400915145874)]
```

[190]:
```
w3 = 'countri'
model1.wv.most_similar(w3)
```

[190]:
```
[('land', 0.4887744188308716),
 ('arab', 0.4550645053386688),
 ('peopl', 0.44194215536117554),
 ('wealthi', 0.42973756790161133),
 ('thet', 0.4263495206832886),
 ('world', 0.4240114092826843),
 ('us', 0.4212038516998291),
 ('european', 0.41993874311447144),
 ('expansion', 0.41251832246780396),
 ('live', 0.40439772605895996)]
```

Second model uses CBOW tuning and negative sampling

[191]:
```
model2 = gensim.models.Word2Vec(
        textdata,
        size = 200,
        sg = 0, # use CBOW
        hs = 0, # use negative sampling
        iter = 20)
```

[192]:
```
w1 = 'polit' # stemmed version of 'politics'
model2.wv.most_similar(w1)
```

[192]:
```
[('ideolog', 0.479442298412323),
 ('agenda', 0.4705851078033447),
 ('econom', 0.4452664852142334),
 ('racial', 0.44172555208206177),
 ('stake', 0.4137029647827148 4),
 ('sole', 0.4134103059768677),
 ('divers', 0.40592777729034424),
 ('philosophi', 0.3986875116825104),
 ('prolif', 0.39447957277297974),
 ('coalit', 0.3930385112762451)]
```

[193]:
```
w2 = 'thousand'
model2.wv.most_similar(w2)
```

[193]:
```
[('hundr', 0.7453307509422302),
 ('twenti', 0.6143282651901245),
 ('fifti', 0.597360372543335),
 ('dozen', 0.57829749584198),
 ('million', 0.5748527646064758),
```

```
 ('approxim', 0.5406707525253296),
 ('hunder', 0.5385438203811646),
 ('refuge', 0.5160118341445923),
 ('half', 0.5053600072860718),
 ('deport', 0.5041437149047852)]
```

[194]:
```
w3 = 'countri'
model2.wv.most_similar(w3)
```

[194]:
```
[('economi', 0.4222654700279236),
 ('vast', 0.4147487282752991),
 ('democraci', 0.40389731526374817),
 ('britain', 0.4017581343650818),
 ('germani', 0.40147364139556885),
 ('succeed', 0.39928996562957764),
 ('citizenship', 0.3979325592517853),
 ('wealthi', 0.397931843996048),
 ('treat', 0.3917727470397949),
 ('european', 0.3860580325126648)]
```

Third model uses skip-gram and negative sampling.

[195]:
```
model3 = gensim.models.Word2Vec(
        textdata,
        size = 200,
        sg = 1, # use skip-gram
        hs = 0, # use negative sampling
        iter = 20)
```

[196]:
```
w1 = 'polit' # stemmed version of 'politics'
model3.wv.most_similar(w1)
```

[196]:
```
[('crux', 0.46794018149375916),
 ('uci', 0.44179970026016235),
 ('covert', 0.42880165576934814),
 ('academia', 0.4243655204772949),
 ('impot', 0.40385329723358154),
 ('invol', 0.40143439173698425),
 ('reshap', 0.40044814348220825),
 ('monarch', 0.3960481584072113),
 ('clout', 0.3960346579551697),
 ('irrespect', 0.393835186958313)]
```

[197]:
```
w2 = 'thousand'
model3.wv.most_similar(w2)
```

[197]:
```
[('hundr', 0.6340253353118896),
 ('riflemen', 0.5023699998855591),
 ('hunder', 0.4799419641494751),
 ('trabzon', 0.45165306329727173),
 ('rehir', 0.45126426219940186),
 ('livelihood', 0.4443967938423157),
 ('glyph', 0.4428291916847229),
 ('twenti', 0.4341743588447571),
 ('acr', 0.43135836720466614),
 ('deport', 0.42930537462234497)]
```

[198]:
```
w3 = 'countri'
model3.wv.most_similar(w3)
```

[198]:
```
[('america', 0.4656364917755127),
 ('armsnevernevernev', 0.4558003544807434),
 ('acr', 0.4442521929740906),
 ('telaviv', 0.4336003065109253),
 ('ecconomi', 0.42291903495788574),
 ('englishman', 0.41559451818466187),
 ('expansion', 0.41443145275115967),
 ('arabian', 0.4144164025783539),
 ('calam', 0.4112203121185303),
 ('hungari', 0.4111582040786743)]
```

Fourth model uses negative sampling and CBOW

[199]:
```
model4 = gensim.models.Word2Vec(
        textdata,
        size = 200,
        sg = 0, # use CBOW
        hs = 1, # use hierarchical softmax
        iter = 20)
```

[200]:
```
w1 = 'polit' # stemmed version of 'politics'
model4.wv.most_similar(w1)
```

[200]:
```
[('ideolog', 0.2981662154197693),
 ('econom', 0.2893720269203186),
 ('usca', 0.28250837326049805),
 ('involv', 0.27779242396354675),
 ('forc', 0.2678338885307312),
 ('reinvent', 0.2653023898601532),
 ('posit', 0.262771338224411),
 ('leftist', 0.26245975494384766),
 ('desir', 0.26115232706069946),
 ('implic', 0.2609318792819977)]
```

[201]:
```
w2 = 'thousand'
model4.wv.most_similar(w2)
```

[201]:
```
[('million', 0.3267165422439575),
 ('flee', 0.31789684295654297),
 ('earthli', 0.3094852566719055),
 ('twenti', 0.3011215329170227),
 ('compund', 0.2995664179325104),
 ('whereabout', 0.2929856479167938),
 ('hundr', 0.27984845638275146),
 ('hungarian', 0.27687346935272217),
 ('approxim', 0.274114191532135),
 ('refuge', 0.2739272117614746)]
```

[202]:
```
w3 = 'countri'
model4.wv.most_similar(w3)
```

[202]:
```
[('world', 0.3884241282939911),
 ('peopl', 0.3606693148612976),
 ('us', 0.3505263924598694),
 ('jew', 0.33863359689712524),
 ('russia', 0.31717199087142944),
 ('economi', 0.3112293481826782),
 ('nonjew', 0.29754510521888733),
 ('wealthi', 0.29508641362190247),
 ('societi', 0.2901819944381714),
 ('planet', 0.28953540325164795)]
```

2.1.1 Results

Model 1: Skip-Gram and Hierarchical Softmax tuning "polit" - does not do well. Captures similar words like "ideology" and "society" but its cosine similarity values are smaller than they should be.

"thousand" - model does much better here. Captures similar number-related words like "hundred", "twenty", and "million", and words that go with these numbers in military context like "riflemen" (as in, "the army had a thousand riflemen"). However, it also picks some nonsensical words, like "erzincan".

"countri" (country) - the model captures words that make sense, like "land", "european", and "world". However, its confidences of similarities are still very low.

Overall: Overall, this is not a great model. The combination of skip-gram and softmax is not ideal for this data.

Model 2: CBOW and Negative Sampling tuning "polit" - This model does somewhat better than the first, but the highest cosine similarity value is still a low 0.48. However, it does catch words like "economy", "ideology", and "agenda".

"thousand" - This model does much better than the first, capturing similar word "hundred" with a similarity value of 0.75. This is the most sensical vector so far, with words like "twenty", "dozen", and "fifty" being captured with high similarity values.

"countri" - This model does slightly worse than the first model, capturing similar words but with a max similarity value of 0.42. However, words like "european", "germany", and "britain" are in fact similar and are captured by this model.

Overall: Overall, this model performs about the same as the first model. "polit" and "countri" are consistently harder to train, with "thousand" being consistently the word with the best similarity vectors. Parameter-wise, this is the opposite of the first model; this may indicate that the dataset is just not large enough or is too sparse for most of these words to have high levels of context around them.

Model 3: Skip-Gram and Negative Sampling tuning "polit" - This is not a great result, with the word "uci" as the "most similar" with a similarity of only 0.43 (and being nonsensical). It does capture some other similar words, like "ideology" and "chancellor", but generally does a poor job.

"thousand" - The first several words in this similarity vector ("hundred", "riflemen") make sense, but it breaks down towards the end. Overall this is not a strong result, despite cosine similarity values being high.

"countri" - This is the strongest result for Model 3, with strong words like "telaviv" and "englishmen" and one funny entry of "armsnevernevernev" (which really should have been removed in tf-idf).

Overall: The combination of skip-gram and negative sampling is not a great one. It appears that skip-gram may be the problem; CBOW performs better in these experiments. The next experiment will determine whether hierarchical softmax or negative sampling performs better for this dataset.

Model 4: CBOW and Hierarchical Softmax tuning "polit" - The model does not have high cosine similarity values (max 0.31) here but does pick words that somewhat make sense around "political", such as "economy", "extremist", and "leadership".

"thousand" - The model still assigns a low max similarity value (0.35) but again chooses words that make sense in context, like "million", "fifteen", and "twenty".

"countri" - We see the pattern continue here; the max cosine similarity value is a low 0.36, but the model captures words that make sense in context like "world", "us", and "society".

Overall: This model does a decent job across all three word vectors despite assigning low cosine similarity values. It is interesting that it behaves so consistently across all three words, as models 1-3 behaved very differently across the three word vectors.

3 Homework 2 Part 2

2. Read the required embeddings papers (word2vec, Bert, optionally Elmo) and compare the approaches in less than one page, preferably in a table format. You are free to decide what aspects to compare. For example, you can include information such as: learning model details, summary of word context approaches, corpus size requirements, computational requirements,

ease of installation/use of source code, date of publication and number of google scholar citations :), … etc.

Word2Vec

- Learning Model Details:
  - The training objective is to learn word vector representations that are good at predicting the nearby words.
  - Simple extension from word basis to phrase basis (e.g. to pick out phrases like "Boston Globe", the newspaper).
  - Extension of the skip-gram model, but can also be modified to use CBOW.
  - Options for hierarchical softmax (an efficient approximation of full softmax) or negative sampling (NCE).
  - Subsamples frequent words. This is similar to TF-IDF in that it decreases the impact of words that are used over and over (and therefore have less meaning in context). This technique significantly improves the accuracy of the model on more rare words.
- Word Context Approach:
  - The model uses a skip-gram algorithm to train each word using the words around it. Phrases are formed based on unigram and bigram counts, so that the model can use some multi-grams without hugely increasing the size of vocabulary (as it would with n-gram use).
  - However, when you are testing the model or finding similarity results, it does not matter where in the sentence or phrase a word occurs. In this sense it is somewhat context-independent.
  - word2vec also does not attempt to classify parts of speech or otherwise place a word in its sentence.
- Corpus Size Requirement:
  - You need a large number of samples for word2vec. Generally a training corpus of 250,000 unique words is best.
  - This paper uses a corpus with dimensionality of 300 and context size of 5. For the highest amount of accuracy, it used a corpus of 33 billion words (huge) and achieved an accuracy of 72%.
  - A computationally efficient infrastructure (via skip-gram and hierarchical softmax) means that word2vec can train quickly on huge datasets.
- Ease of Installation:
  - word2vec is easiest to implement in C, but gensim produces a Python version as well which works well (above). The Python version works up to 70x faster if a C compiler is also installed, though for a relatively small corpus simple models still trained within a few seconds. If the C compiler is not installed, the gensim algorithm runs on Numpy implementations. Both gensim and numpy are simple to install.
- Date of Publication:
  - 7 Sep 2013
- Number of Google Scholar Citations:
  - 15320

BERT

- Learning Model Details:

  - BERT improves on fine-tuning approaches (as opposed to feature-based approaches) by building a deep learning model that uses bidirectional training (breaking a previously-held assumption that training for NLP must flow from left to right). It does this by using Masked Language Models (MLMs). The LMs used must be "masked" since they are bidirectional; each pass, a random number of words are omitted in order to keep the word from being able to "see" (and thus predict) itself.
  - BERT starts with a generalized pre-training step and then moves into context-specific fine-tuning steps that build on the pre-trained model with labelled data.
  - It is versatile for many tasks because it represents "sentences" as continuous strings of words, rather than strictly as linguistic sentences.
- Word Context Approach:
  - Model focuses on Next Sentence Prediction (NSP) so that it can understand the relationship between sentences. This is beneficial for downstream tasks like question answering and natural language inference.
  - BERT is therefore very contextual and very dependent on the Transform used. This means that it is fairly difficult to tune and implement, BUT that once tuned and fully trained it can answer questions and perform other more complex tasks than single-directional models like word2vec can.
- Corpus Size Requirement:
  - For each language, BERT must be pre-trained on a very large corpus (33 billion words for English). This must only be done once per language; afterwards, fine-tuning can be done by context and can be much faster and done with a much smaller dataset.
  - In the paper, BERT fine-tuning does fairly well even with fairly small datasets. For example, the MPRC dataset ranges from an accuracy of about 79% to about 88%, depending on the size of the model.
- Ease of Installation:
  - BERT models are available for download and should run out of the box (though I have not downloaded, as I do not have the memory or RAM to run these models).
  - The BERT documentation on Github (https://github.com/google-research/bert/blob/master/README.md) enumerates how computationally intensive its models are to train: it can 4 days on 4 to 16 TPUs to train BERT on a new language.
  - Fine-tuning can be done quickly - in one hour on a single TPU for most datasets.
- Date of Publication:
  - Nov 2, 2018
- Number of Google Scholar Citations:
  - 1851 (many fewer than word2vec)

[ ]:

[ ]:

[ ]: