Lecture 19: Deep Text Learning

CS 167: Machine Learning

Things to get started downloading

Download/install `gensim` library:
`http://radimrehurek.com/gensim/install.html`

This might require updated versions of other libraries like NumPy which in turn might require a fortran compiler. I recommend gfortran:
`https://gcc.gnu.org/wiki/GFortranBinaries`

Get the `punkt` tokenizer models from the `nltk` downloader (under the Models tab)

```
import nltk
nltk.download()
```


Extra Credit Opportunity

For *5 extra credit* homework points: attend the **Math and Computer Science Department Capstone Poster Session** and fill out evaluation form.

- Friday, December 9th (dead day) from 3:30-5:00 pm
- Upper Olmstead, 310-311
- Talk to at least 3 presenters
- Ask them an insightful question
- Fill out evaluation form (will be provided in that room)
- Folder for completed evaluations will be available

Deep Learning

Deep Learning: (Manley's definition) something that involves learning algorithms and at least one of the following:

- nonlinear stuff - like multilayer neural nets or SVMs with nonlinear kernels?
- abstractions of attributes - like when we did PCA?
- you need funding and want it to sound cool

Word2vec

Word2vec

- Developed by Google
- *unsupervised* - doesn't look at target column
- 2-layer neural network
- given a word, guesses nearby words, order unimportant (like bag-of-words)
- words mapped to a numerical vector (the neural network's hidden layer) representing relations to other words

libraries you need

```
import pandas
from bs4 import BeautifulSoup
import re
import nltk
import numpy
from gensim.models import word2vec
```


Format needed by Word2vec

To train **Word2vec**, we need a list of lists

- each inner list is the words from one sentence
- the more sentences, the better

```
[[u'with', u'all', u'this', u'stuff', u'going', u'down',
u'at', u'the', u'moment', u'with', u'mj', u'i', u've',
u'started', u'listening', u'to', u'his', u'music',
u'watching', u'the', u'odd', u'documentary', u'here', u'and',
u'there', u'watched', u'the', u'wiz', u'and', u'watched',
u'moonwalker', u'again'],
[u'maybe', u'i', u'just', u'want', u'to', u'get', u'a',
u'certain', u'insight', u'into', u'this', u'guy', u'who',
u'i', u'thought', u'was', u'really', u'cool', u'in', u'the',
u'eighties', u'just', u'to', u'maybe', u'make', u'up', u'my',
u'mind', u'whether', u'he', u'is', u'guilty', u'or',
u'innocent'],
...
```


Cleaning one sentence

Just like before, except don't take out the stop words

return the list of words instead of joining them back together

```
def clean_sentence( raw ):
    bs = BeautifulSoup(raw)
    letters_only = re.sub("[^a-zA-Z]"," ",bs.get_text())
    lower_case = letters_only.lower()
    words = lower_case.split()
    return words
```

Breaking up a review into sentences

```
def review_to_sentences( review, tokenizer):
    #didn't seem to work without it, thanks StackOverflow
    review = review.decode('utf-8')
    #strip out whitespace at beginning and end
    review = review.strip()
    raw_sentences = tokenizer.tokenize(review)
    sentences_list = []

    for sentence in raw_sentences:
        if len(sentence) > 0: #skip it if the sentence is empty
            cl_sent = clean_sentence(sentence)
            sentences_list.append(cl_sent)

    return sentences_list
```


Exercise

Check out what you get for the first review:

```
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
data = pandas.read_csv("imdb_reviews.tsv", delimiter="\t")
print( review_to_sentences(data['review'][0],tokenizer) )
```

Exercise: Do this for all the reviews (it might take a while). You need to end up with a list of lists (not a list of lists of lists) with the sentences for all reviews all on the same level. Print it out to make sure.

Training Word2vec

```
num_attributes = 300   # Word vector dimensionality
min_word_count = 40    # Minimum word frequency
num_workers = 4        # Number of threads to run in parallel
context = 10           # Context window size
downsampling = 1e-3    # Downsample setting for frequent words

# Initialize and train the model (this will take some time)
model = word2vec.Word2Vec(sentences_for_all_reviews, \
            workers=num_workers, size=num_attributes, \
            min_count = min_word_count, \
            window = context, sample = downsampling)

#saves memory if you're done training it
model.init_sims(replace=True)
```


Fun with the model

Print out the results of each

```
model.vocab
'chicago' in model.vocab
'iowa' in model.vocab
model.similarity('england','france')
model.similarity('england','paris')
model.most_similar('king')
model.most_similar('awful')
model.doesnt_match(['man','woman','child','kitchen'])
model.doesnt_match(['france','england','germany','berlin'])
model['king']
model['queen']
model['man']
model['woman']
(model['king'] - model['man'] + model['woman'])
model.most_similar(positive=['woman', 'king'], negative=['man'])
```

Vector for a full review

To get a vector for a full review, we could just average the vectors for each of the words

```
def make_attribute_vec(words, model, num_attributes):
    # Pre-initialize an empty numpy array (for speed)
    attribute_vec = numpy.zeros((num_attributes,),dtype="float32")

    nwords = 0.0

    # Loop over each word in the review and, if it is in the model's
    # vocaublary, add its attribute vector to the total
    for word in words:
        if word in model.vocab:
            nwords = nwords + 1.0
            attribute_vec = numpy.add(attribute_vec,model[word])

    # Divide the result by the number of words to get the average
    attribute_vec = numpy.divide(attribute_vec,nwords)
    return attribute_vec
```


Vector for a review

```
print('original:',data['review'][0])
clean_review = clean_sentence(data['review'][0])
print('clean:',clean_review)
review_vector = make_attribute_vec(clean_review, model, \
                        num_attributes)
print('vector:',review_vector)
```


Learning with word vectors

Exercise: Convert the reviews into vectors and use the vectors for learning

1. loop through each review
   1. convert review to vector using `make_attribute_vec`
   2. append onto running list
2. split into train and test sets (target column is still in the original dataFrame, hopefully in the same order as your list)
3. train a classifier and test accuracy as usual
4. lots of parameters to tweak

I got 70% accuracy with GaussianNB off the shelf

Try more algorithms and tweak word2vec parameters (and learning algorithm parameters), you should be able to beat that