Word Embedding in Python

Word Embeddings in Python

Available in Gensim

- You can train them, or you can load pretrained embeddings
- Can be used to infer relations between words
- Can be used to train machine learning classifiers

Keras/TensorFlow, or pretrained models in HuggingFace

- To train neural networks
- Different models (word2vec, GloVE, but also alternative embeddings such as BERT)

Gensim example

```
from gensim.models import Word2Vec
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
             ['this', 'is', 'the', 'second', 'sentence'],
             ['yet', 'another', 'sentence'],
             ['one', 'more', 'sentence'],
             ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1,vector_size=50)

word="another"
print("another:",model.wv[word])
print("Model lenghth:",len(model.wv))

print("Most similar words:",model.wv.most_similar(positive=word,topn=3))

# save model
model.save('model.bin')

# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

Results

```
another: [-0.01427803  0.00248206 -0.01435343 -0.00448924  0.00743861
0.01166625
  0.00239636  0.00420546 -0.00822078  0.01445066 -0.01261408  0.00929443
 -0.01643995  0.00407293 -0.0099541  -0.00849538 -0.00621797  0.01131042
  0.0115968  -0.0099493   0.00154666 -0.01699156  0.01561961  0.01851458
 -0.00548466  0.00160045  0.0014933   0.01095577 -0.01721216  0.00116891
  0.01373884  0.00446319  0.00224935 -0.01864431  0.01696473 -0.01252826
 -0.00598475  0.00698757 -0.00154526  0.00282258  0.00356398 -0.0136578
 -0.01944963  0.01808117  0.01239611 -0.01382586  0.00680696  0.00041213
  0.00950749 -0.01423989]

Model lenghth: 14

Most similar words: [('second', 0.237352162599563 6), ('one',
0.18458434939384 46), ('is', 0.13940517604351044)]

Word2Vec(vocab=14, vector_size=50, alpha=0.025)
```

Discussion - I

- The Word2Vec constructor creates a word embedding model from the sentences

- By using model.wv[word] we can access the vector of a word

- model.wv.most_similar(positive=word,topn=3) returns the closest words if “another” is considered as a positive example

- Once you have created a model, you can store it with the save method, and then load it again with Word2Vec.load

Discussion - II

- That set of sentences was clearly not enough to train a model

- The training requires a careful calibration of the Word2vec parameters (e.g., the window size, the vector size, etc.)

- In the following, we will mainly use pre-trained models

Using pretrained models

```
import gensim.downloader
print(list(gensim.downloader.info()['models'].keys()))
```

- Result:
  ['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300', 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50', 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300', 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200', '__testing_word2vec-matrix-synopsis']

Using pretrained models - discussion

- We have models from different training data, and with vectors having different lengths

- A longer vector normally gives more accurate results, but also means a more expensive computation

Let’s use one…

```
import gensim.downloader
glove_vectors = gensim.downloader.load('glove-wiki-gigaword-50')

print(len(glove_vectors[0]))

print(glove_vectors['twitter'])

print(glove_vectors.most_similar('twitter'))

print(glove_vectors.most_similar(positive=['doctor','woman'],negative='man'))
```

Discussion

- gensim.downloader.load downloads a pretrained model
- glove_vectors[‘twitter'] lets you access to the vector for the word ‘twitter’ (note the structure is slightly different from Word2Vec where you should access model.wv instead)
- most_similar returns the most similar words. Positive words contribute positively towards the similarity, negative words negatively.

Result

- 50

- [0.55473   0.14251   1.577   0.44222   -0.40965   -0.24373
  -1.2366   -0.64589   0.31804   0.48623   -0.20947   0.019861
  -0.28046   -0.64705   0.87607   -0.28965   -1.1877   -0.22703
   0.73132   0.064986   0.34437   -0.044798   0.85787   1.0463
   1.3781   -0.21831   0.45545   -0.36639   -0.32279   -0.34018
   1.5663   -0.028824   0.0062708 -0.62084   -1.3351   0.082663
  -0.085856 -0.67657   -1.1872   -0.40016   1.1583   -0.50842
  -1.8528   0.49679   0.94368   -0.97676   0.30505   0.15514
   0.26331   -0.10485]

- [('facebook', 0.9333045482635498), ('myspace', 0.880136966705322 3), ('youtube', 0.8430657982826233), ('blog', 0.826205730438232 4), ('blogs', 0.806482434272766 1), ('blogging', 0.797067105770111 1), ('tumblr', 0.790108978748321 5), ('email', 0.778261125087738), ('tweets', 0.760453701019287 1), ('e-mail', 0.753872692584991 5)]

- [('nurse', 0.840464293956756 6), ('child', 0.766325950622558 6), ('pregnant', 0.757013022899627 7), ('mother', 0.751745760440826 4), ('patient', 0.751666307449340 8), ('physician', 0.750728070735931 4), ('dentist', 0.736034393310546 9), ('therapist', 0.734253704547882 1), ('parents', 0.728634595870971 7), ('surgeon', 0.716521322727203 4)]

You now see what AI bias means!!!!