NLP Lab Documentation

AKASH T
NSTI CALICUT

Lab 4 - Python Implementation for Word Embeddings using Word2vec

Importing Libraries

```
# Import necessary Libraries
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk import download
download("punkt")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

This code imports the necessary libraries:

- Word2Vec from gensim for training the word embedding model.
- word_tokenize from nltk for tokenizing sentences into words.
- download from nltk to download necessary data (in this case, the "punkt" tokenizer models).

Tokenizing Sentences

```
# Example sentences
sentences = [
    "Natural Language Processing is fun.",
    "Language models are improving every day."
]

# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
tokenized_sentences
```

```
[['natural', 'language', 'processing', 'is', 'fun', '.'],
 ['language', 'models', 'are', 'improving', 'every', 'day', '.']]
```

This code defines example sentences, tokenizes them, and converts each sentence to lowercase. The result is a list of tokenized sentences.

Training the Word2Vec Model

```
[3] # Train the Word2Vec model
    model = Word2Vec(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=0)
    # Here sg=0 means the model will use Continuous bag of words architecture and if sg=1 then it will use Skip-gram Model

    # Get word vectors
    word_vectors = model.wv
    print("Word Vector for 'language':", word_vectors['language'])

    Word Vector for 'language': [-0.14233617  0.12917745  0.17945977 -0.10030856 -0.07526743]
```

This code trains the Word2Vec model on the tokenized sentences:

- vector_size=5: Specifies the dimensionality of the word vectors.
- window=5: Specifies the maximum distance between the current and predicted word within a sentence.
- min_count=1: Ignores all words with a total frequency lower than this.
- workers=4: Uses 4 worker threads to train the model.
- sg=0: Uses the Continuous Bag of Words (CBOW) architecture. If sg=1, it would use the Skip-gram model.

Finally, it retrieves and prints the word vector for the word "language".

Lab 2 - Python Implementation for Word Embeddings using GloVe

Importing the Gensim Downloader

```
import gensim.downloader as api
```

This code imports the gensim downloader, which is used to load pre-trained word vectors.

Loading the 50-Dimensional GloVe Model

```
# Load the pre-trained GloVe model with 50 dimensions
glove_vectors_50d = api.load("glove-wiki-gigaword-50")
print("Dimensions of 50d GloVe vector:", len(glove_vectors_50d['language']))
```

```
[==================================================] 100.0% 66.0/66.0MB downloaded
Dimensions of 50d GloVe vector: 50
```

This code loads the pre-trained GloVe model with 50 dimensions using the gensim downloader and prints the dimensionality of the word vector for the word "language".

Loading the 100-Dimensional GloVe Model

```
# Load the pre-trained GloVe model with 100 dimensions
glove_vectors_100d = api.load("glove-wiki-gigaword-100")
print("Dimensions of 100d GloVe vector:", len(glove_vectors_100d['language']))
```

```
[==================================================] 100.0% 128.1/128.1MB downloaded
Dimensions of 100d GloVe vector: 100
```

This code loads the pre-trained GloVe model with 100 dimensions and prints the dimensionality of the word vector for the word "language".

Loading the 200-Dimensional GloVe Model

```
# Load the pre-trained GloVe model with 200 dimensions
glove_vectors_200d = api.load("glove-wiki-gigaword-200")
print("Dimensions of 200d GloVe vector:", len(glove_vectors_200d['language']))
```

```
[==================================================] 100.0% 252.1/252.1MB downloaded
Dimensions of 200d GloVe vector: 200
```

This code loads the pre-trained GloVe model with 200 dimensions and prints the dimensionality of the word vector for the word "language".

Loading the 300-Dimensional GloVe Model

```
# Load the pre-trained GloVe model with 300 dimensions
glove_vectors_300d = api.load("glove-wiki-gigaword-300")
print("Dimensions of 300d GloVe vector:", len(glove_vectors_300d['language']))
```

This code loads the pre-trained GloVe model with 300 dimensions and prints the dimensionality of the word vector for the word "language".

Lab 6 - Python Implementation for Word Embeddings using Fasttext

Importing Libraries

```
# Import necessary libarries
from gensim.models import FastText
from nltk.tokenize import word_tokenize
from nltk import download
# Download required NLTK data
download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

This code imports the necessary libraries:

- FastText from gensim for training the word embedding model.
- word_tokenize from nltk for tokenizing sentences into words.
- download from nltk to download necessary data (in this case, the "punkt" tokenizer models).

Training the FastText Model

```
# Example sentences
sentences = [
    "Natural Language Processing is fun.",
    "Language models are improving every day."
]

# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Train the FastText model
model = FastText(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=1)

# Get word vectors
word_vectors = model.wv
print("Word Vector for 'language':", word_vectors['language'])

# Get vector for an OOV word
print("Word Vector for 'NLPfun':", word_vectors['nlpfun'])
```

```
Word Vector for 'language': [-0.00461428  0.01921903 -0.00035116 -0.00750383 -0.02619313]
Word Vector for 'NLPfun': [ 0.01152632  0.00589536 -0.01608402 -0.00613909  0.00409522]
```

This code defines example sentences, tokenizes them, and converts each sentence to lowercase. Then, it trains the FastText model on the tokenized sentences:

- vector_size=5: Specifies the dimensionality of the word vectors.
- window=5: Specifies the maximum distance between the current and predicted word within a sentence.
- min_count=1: Ignores all words with a total frequency lower than this.
- workers=4: Uses 4 worker threads to train the model.
- sg=1: Uses the Skip-gram model. If sg=0, it would use the Continuous Bag of Words (CBOW) architecture.

Finally, it retrieves and prints the word vector for the word "language" and an out-of-vocabulary (OOV) word "NLPfun".