INTRODUCTION

At the heart of many Natural Language Processing (NLP) tasks, there is the necessity to represent text data in a numeric way or more specifically as vectors. Different techniques exist to represent text data as vectors. One very popular model for this purpose is the Bag-of-Words model which represents documents as a vector of unique words from the vocabulary present in the document. Although this is a very popular model, there are certain drawbacks of this model. Word2Vec model, which represents each word in a document as a fixed-length feature vector, helps to overcome some of the drawbacks of the Bag-of-Words model. Doc2Vec is another vector representation model which enhances the Word2Vec model to provide fixed-length vector representation for variable length text data, such as a sentence, a paragraph or an entire document. In this article, we'll be reviewing the Doc2Vec model in more detail.

EVOLUTION

As mentioned earlier, despite its popularity, the Bag-of-Words (BOW) model has a few drawbacks as listed below:

- It loses the order of the words in the document
- It ignores the semantics of the words
- It suffers from data sparsity and high dimensionality

Bag-of-n-grams or n-grams model improves the BOW model somewhat by considering the word order in a short context. However, this model also suffers from the same data sparsity and high dimensionality problem and ignores the semantics of the words. Losing semantics of words is formally defined as all words being equally distant in a vector space, e.g. "Paris", "France", and "Powerful" are equally distant even though intuitively "Paris" should have been closer to "France" than to "Powerful".

Word2Vec model developed by Mikolov et al. in 2013, generates word embeddings from a large corpus of text by representing each word as a feature vector of multiple dimensions, typically about hundreds of dimensions. This model helps in capturing the syntactic and semantic relations of words such that similar words are placed close to each other in the vector space, e.g. vector representations of words "strong" and "powerful" are close to each other. Similarly, vectors of words that are not closely related will be placed far from each other, e.g. "strong" and "Paris" will be far from each other.

Success of the word vector representations led to experiments towards extending the model to achieve phrase-level or sentence-level vector representations. Doc2Vec model was the result of a similar effort to extend the Word2Vec model to variable length text.

Doc2Vec model introduced the concept of Paragraph Vector which is concatenated with several word vectors from a paragraph to predict the following word in the given context.

ALGORITHMS

Since the Doc2Vec model is an extension of the Word2Vec model, it becomes necessary to discuss the algorithms of the Word2Vec model before going into the algorithms of the Doc2Vec model.

Word2Vec Architecture

Word2Vec architecture is based on the following two models:

- Continuous Bag of Words (CBOW)
- Skip-gram

CBOW

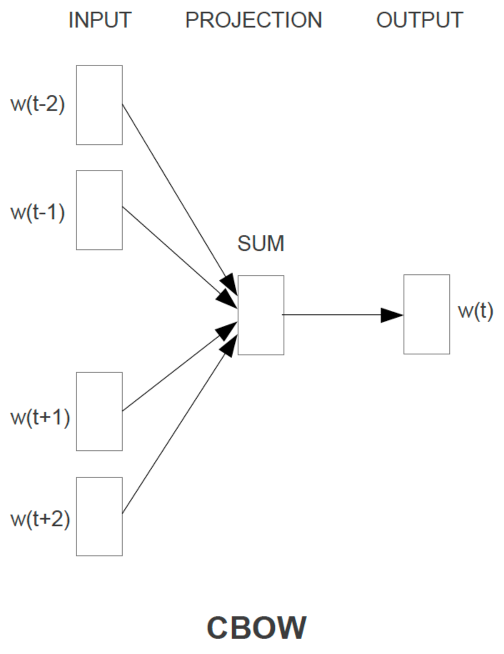


Fig. 1[1]: CBOW architecture

CBOW architecture is used to predict the current word based on words from the context. As shown in Fig. 1, words are selected from the context of the current word within a fixed-sized window, in this example two words before and two words after the current word. These context word vectors are then concatenated to predict the current word vector. This model is denoted as CBOW because unlike in BOW, this model uses continuous distributed representation of the context.

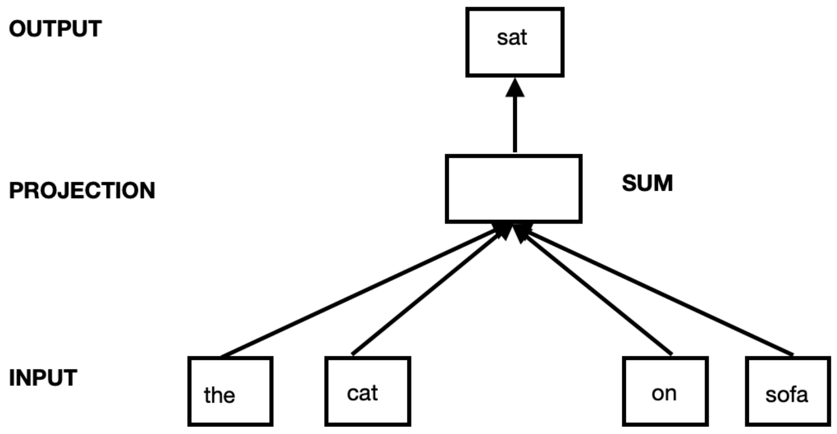


Fig. 2: Example of CBOW

E.g. given a sentence “the cat sat on sofa”, CBOW model would learn to predict the word “sat” based on the context words “the”, “cat”, “on”, and “sofa”.

Skip-gram

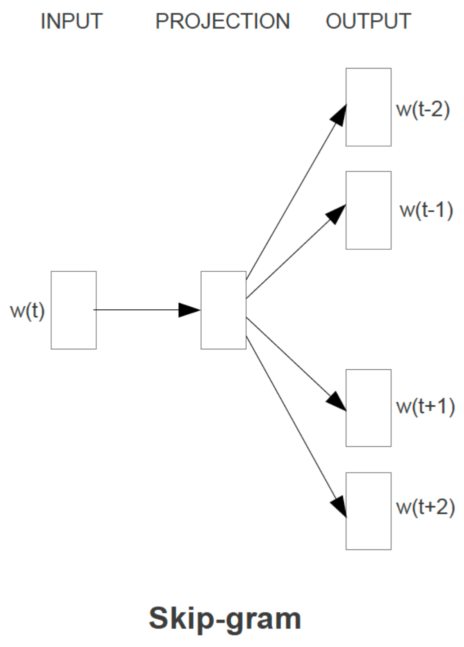


Fig. 3[2]: Skip-Gram architecture

Skip-Gram architecture is similar to the CBOW architecture, although the direction of prediction is opposite to that of CBOW. Instead of predicting the current word from context words, Skip-Gram model predicts the context words from the current word. As shown in Fig. 3, the current word is used to predict words in a fixed-size window around the current word, in this example two words before and two words after the current word.

Doc2Vec Architecture

Doc2Vec architecture is based on the following two models. Both of these models are inspired from the models of Word2Vec explained earlier.

- Distributed Memory Model of Paragraph Vector (PV-DM)
- Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

PV-DM

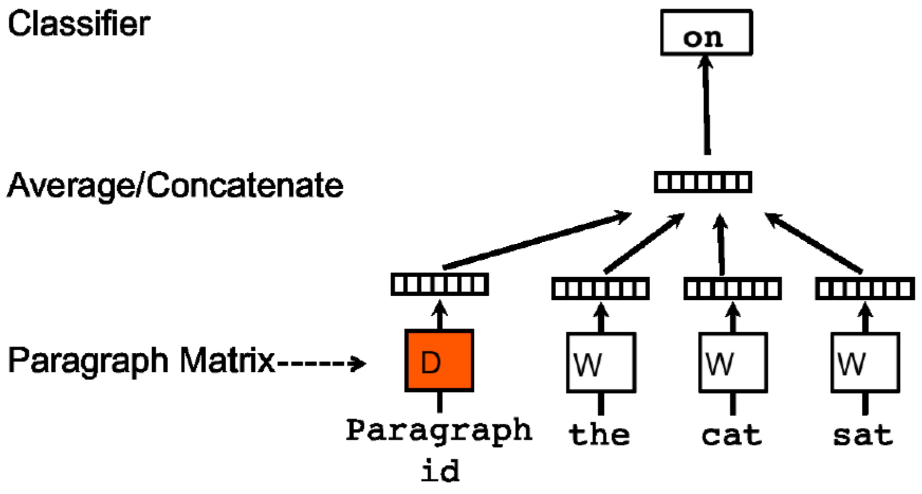


Fig. 4[3]: PV-DM framework for learning Paragraph Vector

PV-DM model derives from the CBOW model in Doc2Vec. The algorithm is almost similar to the CBOW model with the only difference that a paragraph vector is also added as an additional input to contribute to the task of predicting the next word in the current context given the context words. In Fig. 4, each of the context words is a word vector represented as a column in the word vector matrix W. Similarly, each of the paragraphs in the document is a paragraph vector represented as a column in the matrix D. The paragraph vector is concatenated with the word vectors to predict the next word. So, in effect the paragraph vector works as a memory of the topic of the paragraph wherein the paragraph vector remembers the missing word in the current context of the paragraph given the other context words in a fixed-size sliding window of context. Therefore, this model is also called the Distributed Memory Model of Paragraph Vectors, i.e. PV-DM.

After training the model to learn the word vectors W, paragraph vectors D for the known paragraphs, and softmax weights, the next step is to perform an “inference task”. The inference step computes the paragraph vectors for new paragraphs by adding new columns in D and by fixing other parameters of the model, i.e. the word vectors W and the softmax weights. D is then used to make prediction of the following word in the context by using a standard classifier, such as logistic regression.

PV-DBOW

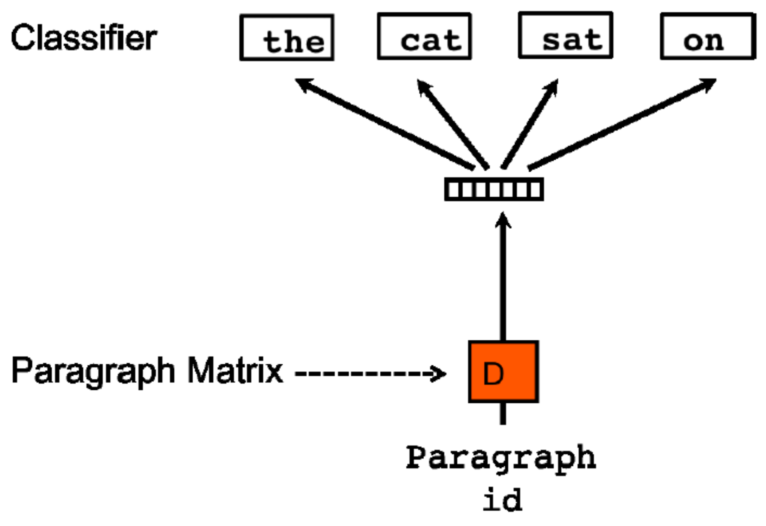


Fig. 5[4]: Distributed Bag of Words version of Paragraph Vector

In the PV-DBOW model, a random word from a random text window in a paragraph is sampled. Then a classification task is formed to predict this randomly sampled word in the output given a Paragraph Vector. This model is similar to the Skip-gram model in Word2Vec.

These two models can be combined to form a Paragraph Vector. PV-DM alone usually works well for most tasks with state-of-the-art performance. But experiments have shown that combination of PV-DM with PV-DBOW gives consistent results across many tasks. Therefore combining the two models is recommended.

HOW TO IMPLEMENT DOC2VEC IN APPLICATION

Gensim is a popular open source python library which is targeted for various Natural Language Processing (NLP) and Information Retrieval (IR) tasks and provides ready-to-use implementation of Doc2Vec, Word2Vec models.

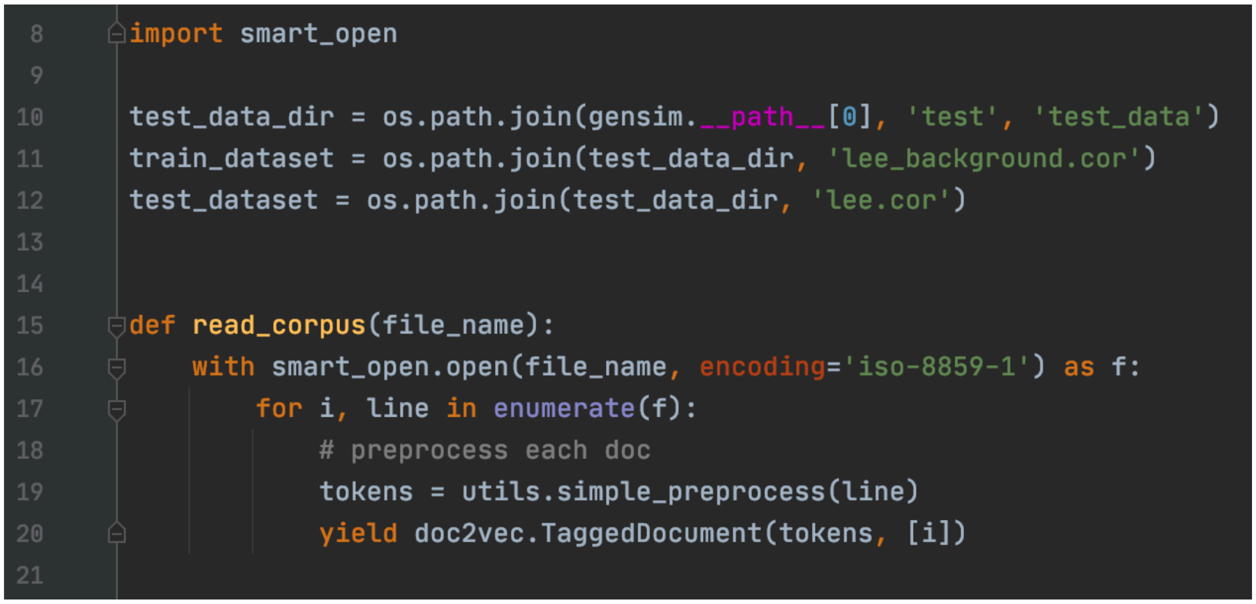
We'll now be reviewing some of the features provided by gensim for Doc2Vec model implementation using a code example.

Reading the corpus

We'll be using the *Lee Background Corpus* (lee_background.cor file) that comes with the installation of gensim to demonstrate how gensim can be used to build and train a Doc2Vec model.

We'll use the *Lee Corpus* (lee.cor file) which is also included with the gensim installation as our test dataset.

Consider that the file is a corpus and each line in the file is a document in that corpus. Define a function to read the corpus.

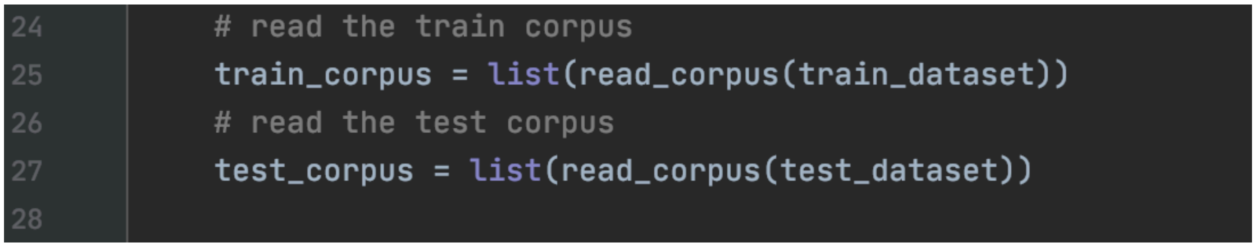


```python
import smart_open

test_data_dir = os.path.join(gensim.__path__[0], 'test', 'test_data')
train_dataset = os.path.join(test_data_dir, 'lee_background.cor')
test_dataset = os.path.join(test_data_dir, 'lee.cor')


def read_corpus(file_name):
    with smart_open.open(file_name, encoding='iso-8859-1') as f:
        for i, line in enumerate(f):
            # preprocess each doc
            tokens = utils.simple_preprocess(line)
            yield doc2vec.TaggedDocument(tokens, [i])

```

The read_corpus() function reads the corpus. For each line in the file, i.e. each document in the corpus, some preprocessing is done to achieve the following:

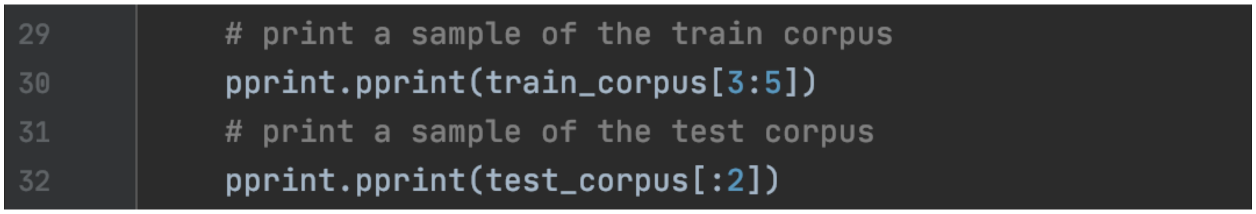
- Tokenize the document into a list of lowercase tokens
- Ignore tokens too small or too large
- By default tokens smaller than two characters, such as "I", "a", etc. or larger than 15 characters are ignored

Read the training dataset and the test dataset.



```python
    # read the train corpus
    train_corpus = list(read_corpus(train_dataset))
    # read the test corpus
    test_corpus = list(read_corpus(test_dataset))

```

Print a few samples from the train dataset and test dataset after they have been read. This is just to get an idea about how the corpus looks like in the application using gensim.



```python
    # print a sample of the train corpus
    pprint.pprint(train_corpus[3:5])
    # print a sample of the test corpus
    pprint.pprint(test_corpus[:2])
```

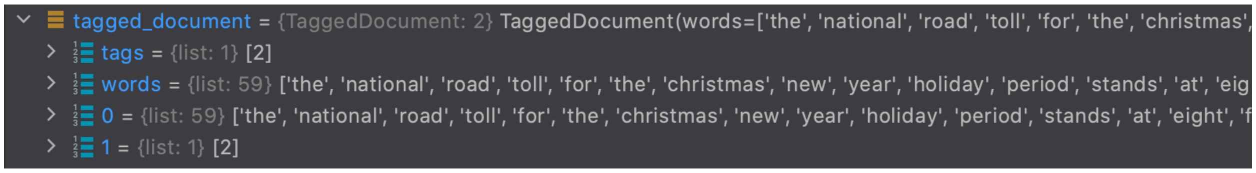
The following output shows one entry in the training corpus printed by the above lines of code.

```
[TaggedDocument(words=['argentina', 'political', 'and', 'economic', 'crisis',
'has', 'deepened', 'with', 'the', 'resignation', 'of', 'its', 'interim',
'president', 'who', 'took', 'office', 'just', 'week', 'ago', 'aldolfo',
'rodregiuez', 'saa', 'told', 'stunned', 'nation', 'that', 'he', 'could', 'not',
'rescue', 'argentina', 'because', 'key', 'fellow', 'peronists', 'would', 'not',
'support', 'his', 'default', 'on', 'massive', 'foreign', 'debt', 'repayment',
'or', 'his', 'plan', 'for', 'new', 'currency', 'it', 'was', 'only', 'week',
'ago', 'that', 'he', 'was', 'promising', 'million', 'new', 'jobs', 'to', 'end',
'four', 'years', 'of', 'recession', 'days', 'after', 'his', 'predecessor',
'resigned', 'following', 'series', 'of', 'failed', 'rescue', 'packages', 'after',
'announcing', 'that', 'the', 'senate', 'leader', 'ramon', 'puerta', 'would',
'assume', 'the', 'presidency', 'until', 'congress', 'appoints', 'new',
'caretaker', 'president', 'the', 'government', 'said', 'he', 'too', 'had',
'quit', 'and', 'another', 'senior', 'lawmaker', 'would', 'act', 'in', 'the',
'role', 'fresh', 'elections', 'are', 'not', 'scheduled', 'until', 'march',
'leaving', 'whoever', 'assumes', 'the', 'presidency', 'with', 'the', 'daunting',
'task', 'of', 'tackling', 'argentina', 'worst', 'crisis', 'in', 'years', 'but',
'this', 'time', 'isolated', 'by', 'international', 'lending', 'agencies'],
tags=[3]),
```

Generating tagged documents for the model

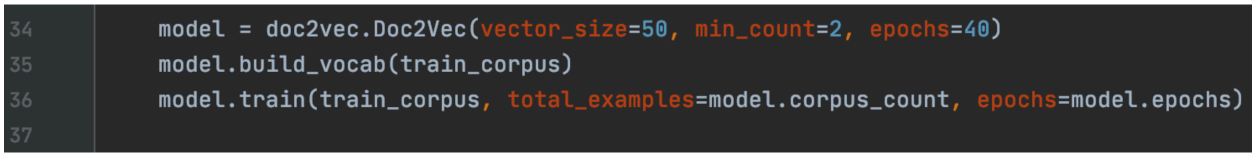
In the read_corpus() function above, line 20 generates a doc2vec.TaggedDocument object with the tokens retrieved from the document and assigns a numeric tag to the TaggedDocument object. TaggedDocument is the input document object for the Doc2Vec model.

A doc2vec.TaggedDocument object looks like below:



```
∨ tagged_document = {TaggedDocument: 2} TaggedDocument(words=['the', 'national', 'road', 'toll', 'for', 'the', 'christmas',
  > tags = {list: 1} [2]
  > words = {list: 59} ['the', 'national', 'road', 'toll', 'for', 'the', 'christmas', 'new', 'year', 'holiday', 'period', 'stands', 'at', 'eig
  > 0 = {list: 59} ['the', 'national', 'road', 'toll', 'for', 'the', 'christmas', 'new', 'year', 'holiday', 'period', 'stands', 'at', 'eight', 'f
  > 1 = {list: 1} [2]
```

Build and train a Doc2Vec model



```
34      model = doc2vec.Doc2Vec(vector_size=50, min_count=2, epochs=40)
35      model.build_vocab(train_corpus)
36      model.train(train_corpus, total_examples=model.corpus_count, epochs=model.epochs)
37
```

Gensim provides the Doc2Vec class to instantiate a Doc2Vec model. Here the model is instantiated with the following parameter values:

- ***vector_size*** = 50, which means the feature vector for the documents will have 50 dimensions
- ***min_count*** = 2, which means words that appear less than two times will be ignored
- ***epochs*** = 40, which means the model will run 40 iterations during training

Another important parameter for the Doc2Vec class is ***dm***. By default *dm* is set to 1. The possible values for the *dm* parameter are:

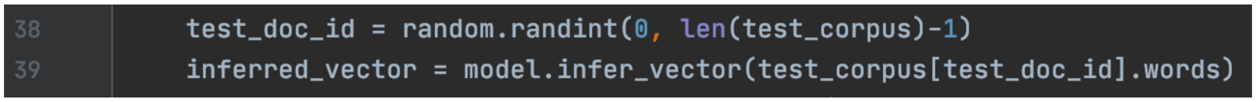
- 1, which sets the training algorithm of the model to be PV-DM
- 0, which sets the training algorithm of the model to be PV-DBOW

In our example, we have not explicitly set the value for *dm*. So we're using the value *dm*=1, i.e. we're using the PV-DM algorithm for training the model.

In line 35, build the vocabulary of the model from the training corpus. Essentially, the vocabulary is a list of all the unique words extracted from the training corpus.

In line 36, train the model on the training corpus.

Infer a document vector for a new document



```
38      test_doc_id = random.randint(0, len(test_corpus)-1)
39      inferred_vector = model.infer_vector(test_corpus[test_doc_id].words)
```

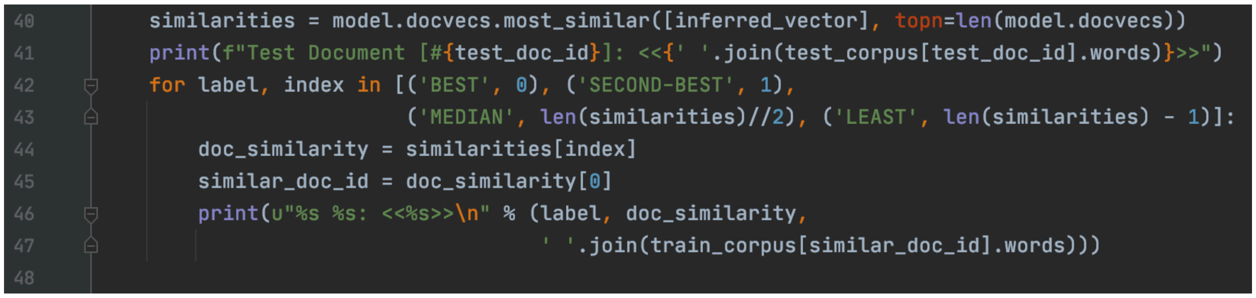
After training the model on the training corpus, we can now infer a feature vector for a document that has not been seen by the model yet.

In line 38, we're getting a random id for a document from the test corpus to infer a vector for the document from the trained model.

In line 39, we call infer_vector() to infer a vector for this new document from the test corpus.

An important parameter to the infer_vector() function is ***epochs***. The value of *epochs* specifies the number of times the new document should be trained. Large values may take more time to train but may improve the quality and stability of the inferred vector after each run. In our example, we haven't specifically set any value for *epochs*. The default value for *epochs* is None which means the *epochs* value from the trained model will be used to train the new document.

Find similar documents



```
40     similarities = model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))
41     print(f"Test Document [#{test_doc_id}]: <<{' '.join(test_corpus[test_doc_id].words)}>>")
42     for label, index in [('BEST', 0), ('SECOND-BEST', 1),
43                          ('MEDIAN', len(similarities)//2), ('LEAST', len(similarities) - 1)]:
44         doc_similarity = similarities[index]
45         similar_doc_id = doc_similarity[0]
46         print(u"%s %s: <<%s>>\n" % (label, doc_similarity,
47                                       ' '.join(train_corpus[similar_doc_id].words)))
48
```

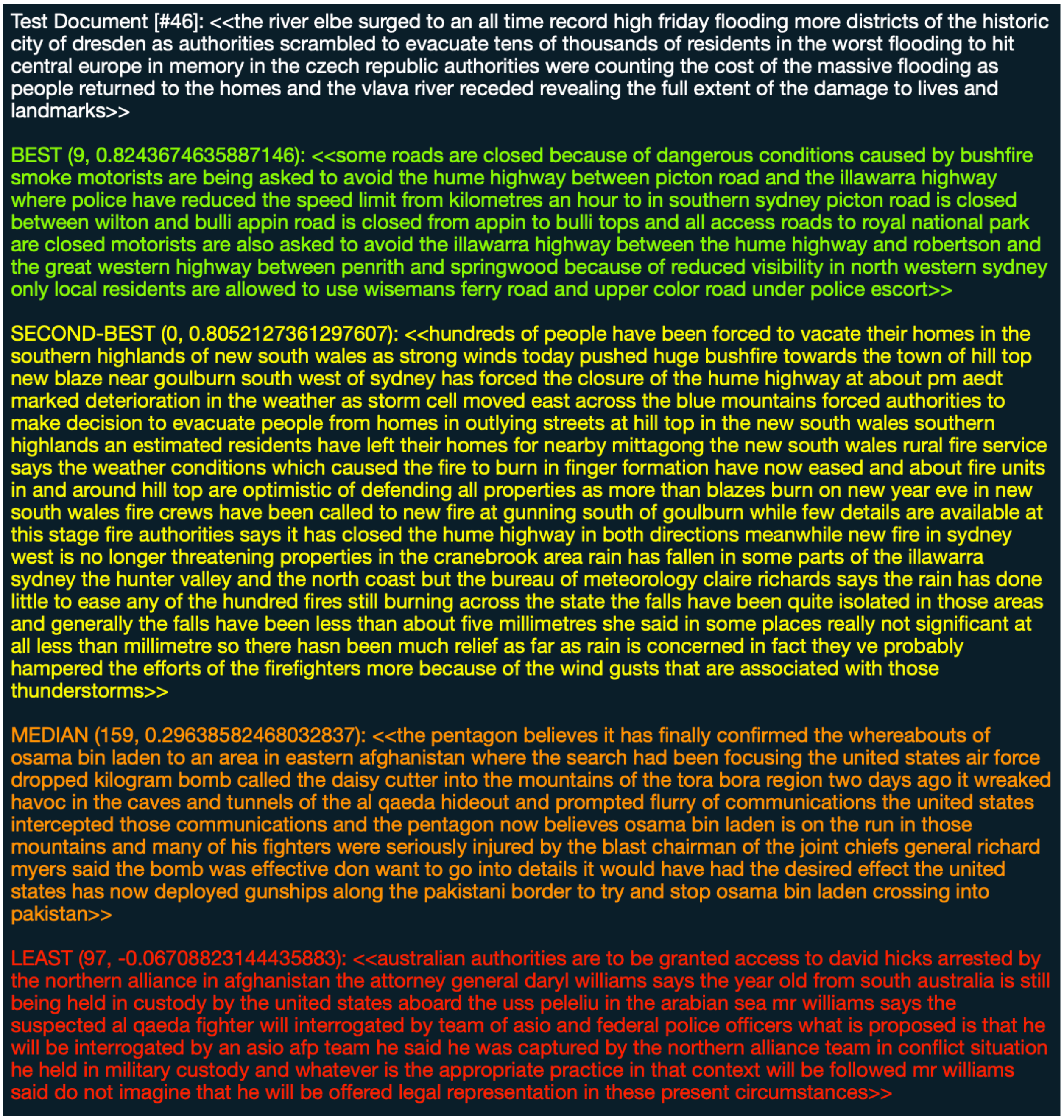
We'll use the inferred vector for a new document from the test corpus and use the trained model to find documents similar to the test document from the trained corpus.

In line 40, we're finding the similarities between the inferred vector of the test document and all the document vectors in the training corpus.

In lines 42-47, we're printing some of the similar documents and their contents in the order starting from 'BEST' to 'LEAST'.

We can then visually inspect the contents of the test document and the printed similar documents to determine whether the returned similar documents are actually similar to the test document and to what extent.

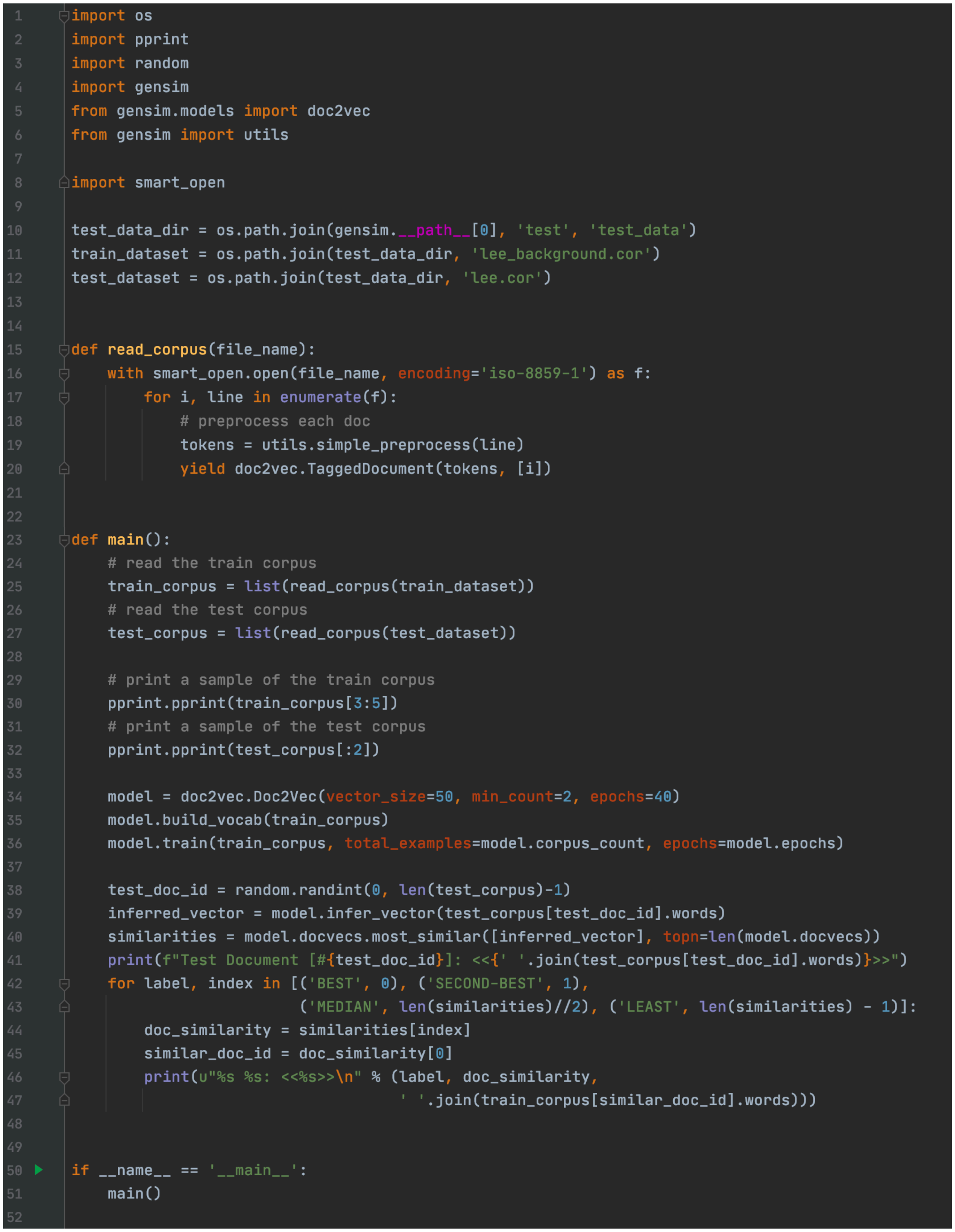
The output of the above lines of code is shown below. In the color coded output below, green represents the document best matching with the test document, yellow represents the second-best matching document, orange represents the median ranked document of all the training documents that were matched, and red represents the least matching document.



```
Test Document [#46]: <<the river elbe surged to an all time record high friday flooding more districts of the historic
city of dresden as authorities scrambled to evacuate tens of thousands of residents in the worst flooding to hit
central europe in memory in the czech republic authorities were counting the cost of the massive flooding as
people returned to the homes and the vlava river receded revealing the full extent of the damage to lives and
landmarks>>

BEST (9, 0.8243674635887146): <<some roads are closed because of dangerous conditions caused by bushfire
smoke motorists are being asked to avoid the hume highway between picton road and the illawarra highway
where police have reduced the speed limit from kilometres an hour to in southern sydney picton road is closed
between wilton and bulli appin road is closed from appin to bulli tops and all access roads to royal national park
are closed motorists are also asked to avoid the illawarra highway between the hume highway and robertson and
the great western highway between penrith and springwood because of reduced visibility in north western sydney
only local residents are allowed to use wisemans ferry road and upper color road under police escort>>

SECOND-BEST (0, 0.8052127361297607): <<hundreds of people have been forced to vacate their homes in the
southern highlands of new south wales as strong winds today pushed huge bushfire towards the town of hill top
new blaze near goulburn south west of sydney has forced the closure of the hume highway at about pm aedt
marked deterioration in the weather as storm cell moved east across the blue mountains forced authorities to
make decision to evacuate people from homes in outlying streets at hill top in the new south wales southern
highlands an estimated residents have left their homes for nearby mittagong the new south wales rural fire service
says the weather conditions which caused the fire to burn in finger formation have now eased and about fire units
in and around hill top are optimistic of defending all properties as more than blazes burn on new year eve in new
south wales fire crews have been called to new fire at gunning south of goulburn while few details are available at
this stage fire authorities says it has closed the hume highway in both directions meanwhile new fire in sydney
west is no longer threatening properties in the cranebrook area rain has fallen in some parts of the illawarra
sydney the hunter valley and the north coast but the bureau of meteorology claire richards says the rain has done
little to ease any of the hundred fires still burning across the state the falls have been quite isolated in those areas
and generally the falls have been less than about five millimetres she said in some places really not significant at
all less than millimetre so there hasn been much relief as far as rain is concerned in fact they ve probably
hampered the efforts of the firefighters more because of the wind gusts that are associated with those
thunderstorms>>

MEDIAN (159, 0.29638582468032837): <<the pentagon believes it has finally confirmed the whereabouts of
osama bin laden to an area in eastern afghanistan where the search had been focusing the united states air force
dropped kilogram bomb called the daisy cutter into the mountains of the tora bora region two days ago it wreaked
havoc in the caves and tunnels of the al qaeda hideout and prompted flurry of communications the united states
intercepted those communications and the pentagon now believes osama bin laden is on the run in those
mountains and many of his fighters were seriously injured by the blast chairman of the joint chiefs general richard
myers said the bomb was effective don want to go into details it would have had the desired effect the united
states has now deployed gunships along the pakistani border to try and stop osama bin laden crossing into
pakistan>>

LEAST (97, -0.06708823144435883): <<australian authorities are to be granted access to david hicks arrested by
the northern alliance in afghanistan the attorney general daryl williams says the year old from south australia is still
being held in custody by the united states aboard the uss peleliu in the arabian sea mr williams says the
suspected al qaeda fighter will interrogated by team of asio and federal police officers what is proposed is that he
will be interrogated by an asio afp team he said he was captured by the northern alliance team in conflict situation
he held in military custody and whatever is the appropriate practice in that context will be followed mr williams
said do not imagine that he will be offered legal representation in these present circumstances>>
```

Visually inspecting the output indicates that our test document is about a natural disaster and evacuation related to that. The best and second-best documents also indicate that they are about some natural disaster and evacuation. The median and the least matching documents show no similarity with the topic of the original test document.

Complete code for reference



```python
import os
import pprint
import random
import gensim
from gensim.models import doc2vec
from gensim import utils

import smart_open

test_data_dir = os.path.join(gensim.__path__[0], 'test', 'test_data')
train_dataset = os.path.join(test_data_dir, 'lee_background.cor')
test_dataset = os.path.join(test_data_dir, 'lee.cor')


def read_corpus(file_name):
    with smart_open.open(file_name, encoding='iso-8859-1') as f:
        for i, line in enumerate(f):
            # preprocess each doc
            tokens = utils.simple_preprocess(line)
            yield doc2vec.TaggedDocument(tokens, [i])


def main():
    # read the train corpus
    train_corpus = list(read_corpus(train_dataset))
    # read the test corpus
    test_corpus = list(read_corpus(test_dataset))

    # print a sample of the train corpus
    pprint.pprint(train_corpus[3:5])
    # print a sample of the test corpus
    pprint.pprint(test_corpus[:2])

    model = doc2vec.Doc2Vec(vector_size=50, min_count=2, epochs=40)
    model.build_vocab(train_corpus)
    model.train(train_corpus, total_examples=model.corpus_count, epochs=model.epochs)

    test_doc_id = random.randint(0, len(test_corpus)-1)
    inferred_vector = model.infer_vector(test_corpus[test_doc_id].words)
    similarities = model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))
    print(f"Test Document [#{test_doc_id}]: <<{' '.join(test_corpus[test_doc_id].words)}>>")
    for label, index in [('BEST', 0), ('SECOND-BEST', 1),
                         ('MEDIAN', len(similarities)//2), ('LEAST', len(similarities) - 1)]:
        doc_similarity = similarities[index]
        similar_doc_id = doc_similarity[0]
        print(u"%s %s: <<%s>>\n" % (label, doc_similarity,
                                    ' '.join(train_corpus[similar_doc_id].words)))


if __name__ == '__main__':
    main()
```

CONCLUSION

From our review of the Doc2Vec model, we observed that Doc2Vec or Paragraph Vector is capable of representing input text of variable length as a fixed-length feature vector. The paragraph vector representation of a document helps in many NLP tasks, such as text classification, document clustering, sentiment analysis, etc.

Because Doc2Vec is an unsupervised model, it is able to learn the paragraph vectors from unlabeled data and thus works well for tasks that do not have a lot of labeled data. The learned paragraph vectors can be used to predict surrounding words of a randomly sampled word in contexts sampled from the paragraph. Similarly, the paragraph vectors can also be used to predict the next word in a context given the other words in the context.

Paragraph vectors also address some of the weaknesses of the BOW model. By virtue of being an extension of Word2Vec model, they inherit an important property of word vectors which helps to capture the semantics of words. They also consider word order at least in the short context.

Gensim python library provides implementation of Doc2Vec model which can be readily used in python applications for various NLP tasks. As demonstrated in the example of this review, gensim library's Doc2Vec implementation can help to train a model on a training corpus to learn paragraph vectors for documents in the corpus, infer paragraph vectors for new documents using the trained model, and find documents similar to a new document from the training corpus.

REFERENCES:

Distributed Representations of Sentences and Documents:
Quoc Le, Tomas Mikolov, 22 May 2014.
arXiv:1405.4053 [cs.CL]
https://arxiv.org/abs/1405.4053

Efficient Estimation of Word Representations in Vector Space:
Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, 7 Sep 2013.
arXiv:1301.3781 [cs.CL]
https://arxiv.org/abs/1301.3781

Distributed Representations of Words and Phrases and their Compositionality:
Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, 16 Oct 2013.
arXiv:1310.4546 [cs.CL]
https://arxiv.org/abs/1310.4546

Gensim documentation:
https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py

Lee Background Corpus, Lee Corpus:
https://hekyll.services.adelaide.edu.au/dspace/bitstream/2440/28910/1/hdl_28910.pdf

[1][2] Fig. taken from https://arxiv.org/pdf/1301.3781.pdf

[3][4] Fig. taken from https://arxiv.org/pdf/1405.4053.pdf