Lab No. 13

Natural Language Processing (NLP)

This laboratory session introduces students to Word2Vec, a popular technique in Natural Language Processing (NLP) used to represent words as meaningful dense vectors. Using textual data from the *Game of Thrones* series, students will learn how word embeddings capture semantic relationships between words based on their context. Through preprocessing, model training, and similarity analysis, this lab helps students understand how machines learn word meanings and relationships from large text corpora in an unsupervised manner.

LAB Objectives:

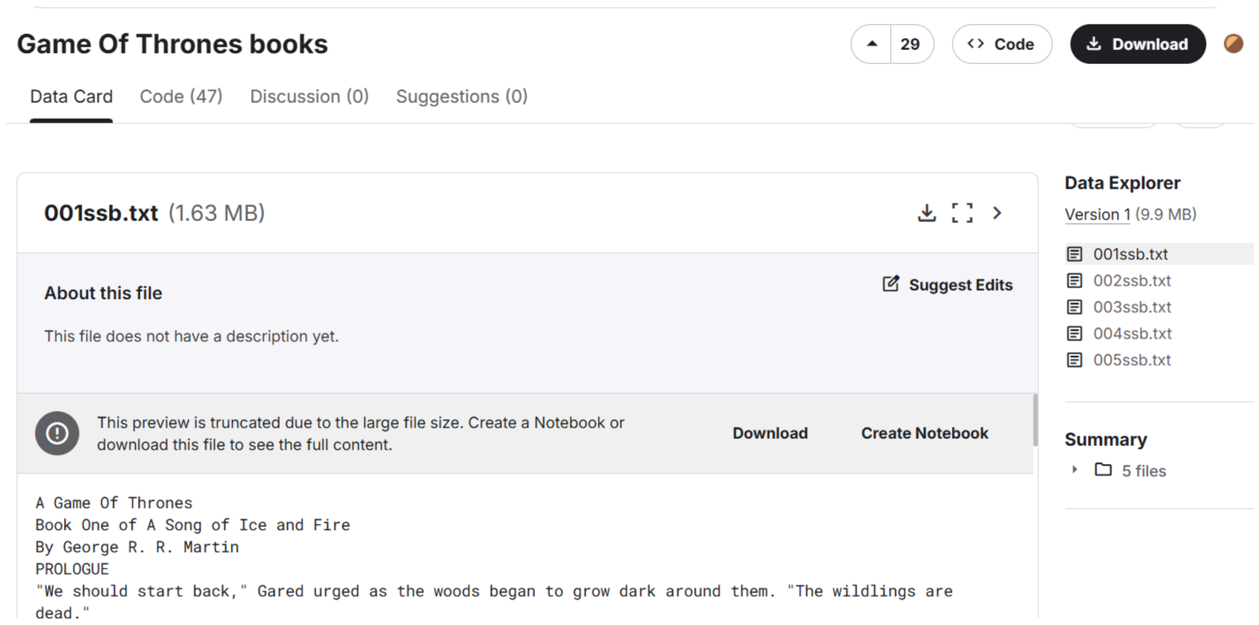
- Understand **distributional semantics**
- Learn how **Word2Vec** converts words into dense vectors
- Apply **Word2Vec (Skip-Gram / CBOW)** on real textual data (Game of Thrones)
- Explore **word similarity, analogy, and visualization**

game-of-thrones-word2vec

word2vec applied on game of thrones data

Dataset Link: https://www.kaggle.com/khulasasndh/game-of-thrones-books

Download the data set from Kaggle



Game Of Thrones books

001ssb.txt (1.63 MB)

About this file

This file does not have a description yet.

This preview is truncated due to the large file size. Create a Notebook or download this file to see the full content.

```
A Game Of Thrones
Book One of A Song of Ice and Fire
By George R. R. Martin
PROLOGUE
"We should start back," Gared urged as the woods began to grow dark around them. "The wildlings are
dead."
```

Data Explorer

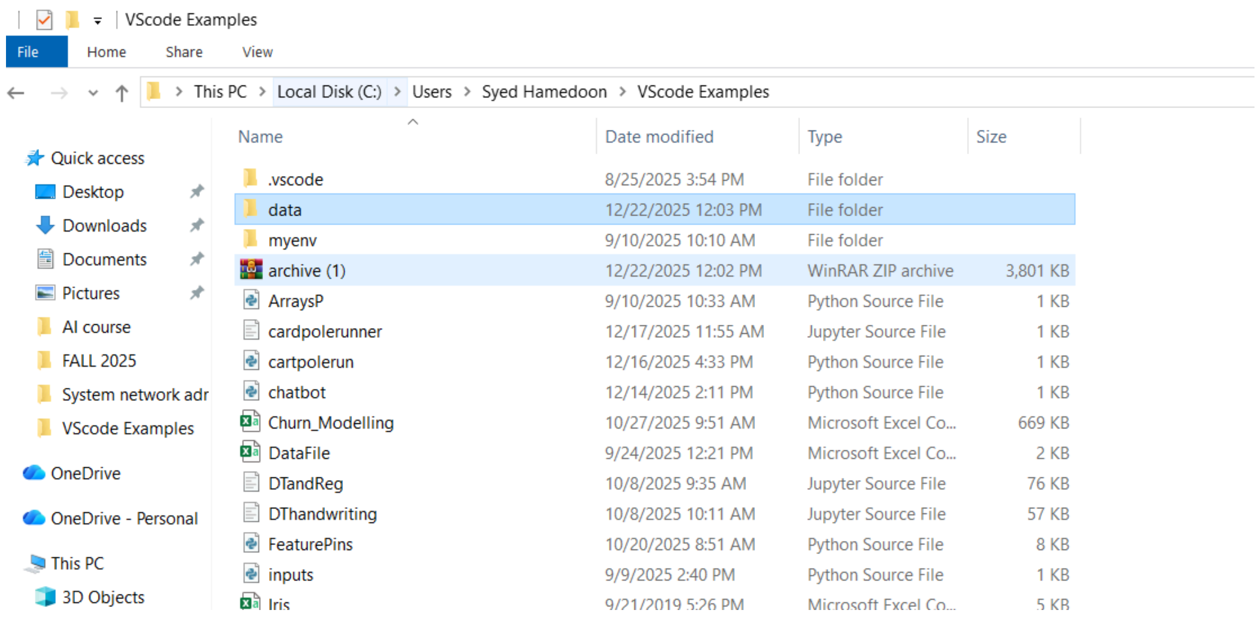
Version 1 (9.9 MB)

- 001ssb.txt
- 002ssb.txt
- 003ssb.txt
- 004ssb.txt
- 005ssb.txt

Summary

5 files

Add the dataset in folder data where VS code directory present



Name	Date modified	Type	Size
.vscode	8/25/2025 3:54 PM	File folder	
data	12/22/2025 12:03 PM	File folder	
myenv	9/10/2025 10:10 AM	File folder	
archive (1)	12/22/2025 12:02 PM	WinRAR ZIP archive	3,801 KB
ArraysP	9/10/2025 10:33 AM	Python Source File	1 KB
cardpolerunner	12/17/2025 11:55 AM	Jupyter Source File	1 KB
cartpolerun	12/16/2025 4:33 PM	Python Source File	1 KB
chatbot	12/14/2025 2:11 PM	Python Source File	1 KB
Churn_Modelling	10/27/2025 9:51 AM	Microsoft Excel Co...	669 KB
DataFile	9/24/2025 12:21 PM	Microsoft Excel Co...	2 KB
DTandReg	10/8/2025 9:35 AM	Jupyter Source File	76 KB
DThandwriting	10/8/2025 10:11 AM	Jupyter Source File	57 KB
FeaturePins	10/20/2025 8:51 AM	Python Source File	8 KB
inputs	9/9/2025 2:40 PM	Python Source File	1 KB
Iris	9/21/2019 5:26 PM	Microsoft Excel Co...	5 KB

Code in jupter VS code:

```
import numpy as np
import pandas as pd
```

```
!pip install gensim
```

```
import gensim
import os
```

```
!pip install nltk
```

```
data = "C:/Users/Syed Hamedoon/VScode Examples/data"
```

```
import nltk

nltk.download('punkt')
nltk.download('punkt_tab')
```

```
import os
from nltk import sent_tokenize
from gensim.utils import simple_preprocess

DATA_PATH = r"C:\Users\Syed Hamedoon\VScode Examples\data"

story = []

for filename in os.listdir(DATA_PATH):
    if filename.endswith(".txt"):
        file_path = os.path.join(DATA_PATH, filename)

        try:
            with open(file_path, "r", encoding="utf-8") as f:
                corpus = f.read()
        except UnicodeDecodeError:
            with open(file_path, "r", encoding="cp1252") as f:
                corpus = f.read()

        for sent in sent_tokenize(corpus):
            story.append(simple_preprocess(sent))
```

```
print(len(story))
print(story[:2])
```

```
model = gensim.models.Word2Vec(
    window=10,
    min_count=2
)
```

```
model.build_vocab(story)
```

```
model.train(story, total_examples=model.corpus_count, epochs=model.epochs)
```

```
model.wv.most_similar('daenerys')
```

```
model.wv.doesnt_match(['jon','rikon','robb','arya','sansa','bran'])
```

```
model.wv.doesnt_match(['cersei', 'jaime', 'bronn', 'tyrion'])
```

```
model.wv['king']
model.wv.similarity('arya','sansa')
```

```
model.wv.similarity('tywin','sansa')
```

```
model.wv.get_normed_vectors()
```

```
y = model.wv.index_to_key
```

```
len(y)
```

```
y
```

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=3)
```

```
X = pca.fit_transform(model.wv.get_normed_vectors())
```

```
X.shape
```

```
!pip install --upgrade nbformat
```

```
import pandas as pd
import plotly.express as px
import plotly.io as pio
```

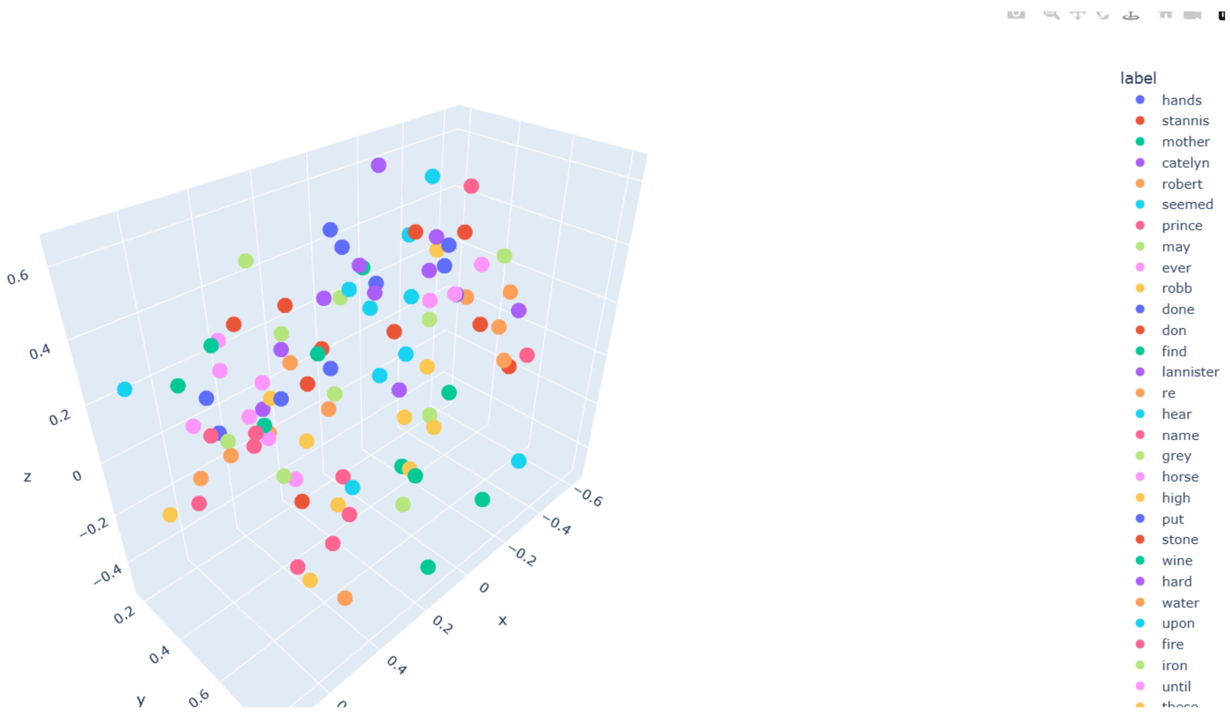
```
pio.renderers.default = "browser"  # ← IMPORTANT

df = pd.DataFrame(X[200:300], columns=["x", "y", "z"])
df["label"] = y[200:300]

fig = px.scatter_3d(
    df,
    x="x",
    y="y",
    z="z",
    color="label"
)

fig.show()
```

Output:



LAB Questions

1. **What is the core idea behind Word2Vec?**

   Word2Vec is a method to turn words into numbers (vectors) so that a computer can understand them. Words that have similar meanings or are used in similar situations are placed closer together in this number space. For example, “king” and “queen” are close to each other, and the difference between “king” and “man” is similar to the difference between “queen” and “woman.” Word2Vec learns these relationships by looking at lots of text and understanding which words appear together. This helps AI find word meanings, similarities, and relationships, making it useful for tasks like language understanding, search, or text analysis.

2. **Difference between CBOW and Skip-Gram?**

   CBOW (Continuous Bag of Words) and Skip-Gram are two methods used in Word2Vec to learn word meanings, but they work in opposite ways. CBOW predicts a word based on the words around it (the context), so it looks at surrounding words to guess the target word. On the other hand, Skip-Gram predicts the context based on a given word, meaning it takes one word and tries to guess which words are likely to appear nearby. CBOW is generally faster and works well with large datasets, while Skip-Gram is better at learning rare words and capturing detailed relationships.

3. **Why is one-hot encoding inefficient?**

   One-hot encoding is a way to represent words as vectors where each word is turned into a vector of mostly zeros and a single one at the position representing that word. This method is inefficient because the vectors are usually very large when the vocabulary is big, which takes up a lot of memory. It also doesn’t show any relationship between words—words like “king” and “queen” are just as different as “king” and “apple,” even though their meanings are related. Because of this, one-hot encoding is not very useful for capturing word meanings in AI applications, which is why methods like Word2Vec are preferred.

4. **Why do character names appear close in vector space?**

   In Word2Vec or similar models, character names often appear close together in vector space because they are used in similar contexts in the text. For example, in a story, names like “Harry” and “Ron” might appear together in many sentences. The model learns that these words often show up near the same words, so it places their vectors near each other. This helps the AI understand that these names are related or connected in the text, even though it doesn't know anything about the real people

5. **How does window size affect semantic learning?**

   In Word2Vec and similar models, the window size is the number of words the model looks at around a target word to learn its meaning. A small window focuses on closer words, which helps the model learn syntactic relationships like grammar or word order. A large window includes more surrounding words, which helps the model learn broader semantic meanings and relationships between concepts. Choosing the right window size is important because it affects how well the AI can understand both specific word usage and overall meaning in text.

6. **Why might rare characters have poor embeddings?**

   Rare characters or words appear very few times in the training text, so the model doesn't get enough examples to learn their meaning well. As a result, their vectors (embeddings) might not be accurate or meaningful. This makes it harder for the AI to understand or relate these rare words to other words in the text. Methods like Skip-Gram or using larger datasets can help improve embeddings for rare words by giving the model more context to learn from.

7. **Which model performed better: CBOW or Skip-Gram? Why?**

   There is no single model that is always better; it depends on the task and data. CBOW is usually faster and works well for common words in large datasets. Skip-Gram often performs better for rare words because it captures more detailed relationships between words.

   Why:
   The difference comes from how the models learn. CBOW predicts a word based on its surrounding words, which averages the context and makes it less precise for rare words. Skip-Gram predicts the surrounding words from a given word, giving the model more opportunities to learn relationships, especially for words that appear less frequently. That's why Skip-Gram usually gives better embeddings for rare words, while CBOW is faster and works well with frequent words.

8. **What happens if vector size is too small or too large?**

The vector size (dimension) decides how much information a word’s embedding can store. If the vector size is too small, it cannot capture enough details about words, so embeddings may be too simple and less accurate. If the vector size is too large, the embeddings may store too much unnecessary information, making the model slower and sometimes overfit the training data. Choosing a moderate vector size helps balance accuracy and efficiency.

9. **Can Word2Vec understand word meaning without labels? Explain.**

   Yes, Word2Vec can learn word meanings without labels because it uses the context of words in text instead of pre-defined categories. It looks at which words appear near each other in sentences and learns patterns from these relationships. Words that appear in similar contexts end up having similar vectors, so the model captures meaning automatically. This makes Word2Vec an unsupervised learning method, meaning it doesn’t need labeled data to understand word relationships.