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:

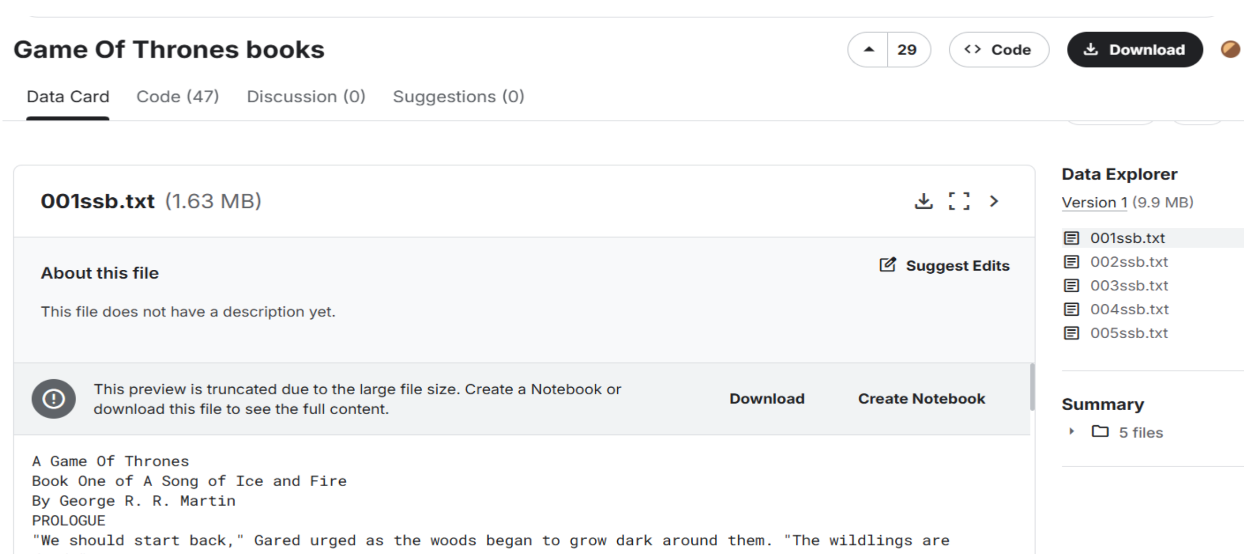
- Understand **distributional semantics**
- Learn how **Word2Vec** converts words into dense vectors
- Apply **Word2Vec (Skip-Gram / CBOW)** on real textual data (Game of Thrones)
- Explore **word similarity, analogy, and visualization**

game-of-thrones-word2vec

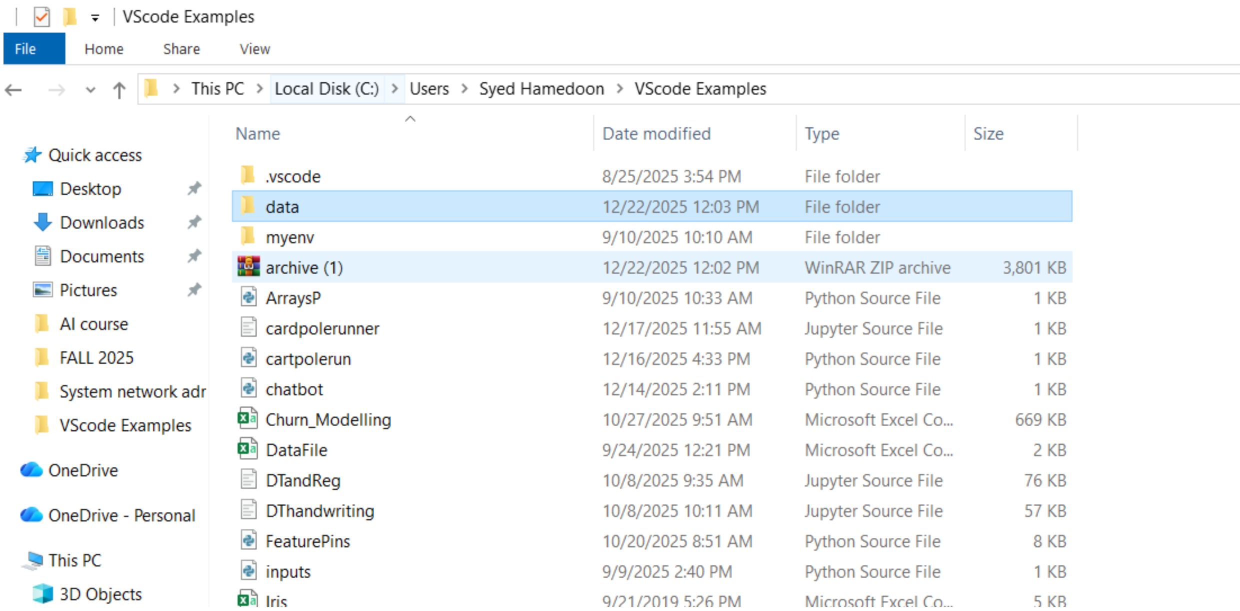
word2vec applied on game of thrones data

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

Download the data set from Kaggle



Add the dataset in folder data where VS code directory present



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())
```

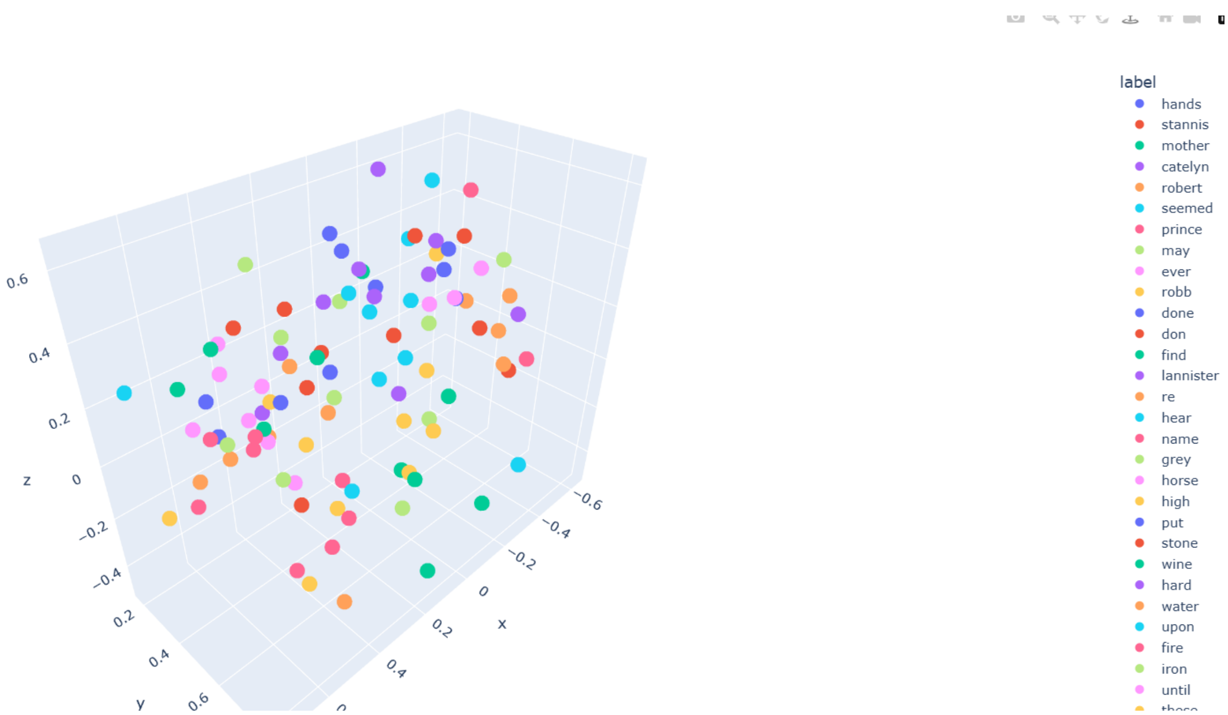
```
X.shape
```

```
!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:



Code:

```
import numpy as np
import pandas as pd
import gensim
import os
import nltk
```

[1] ✓ 8.0s

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

[2] ✓ 11.9s

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\h\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\h\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt_tab.zip.
```

```
True
```

```
DATA_PATH = r"C:\Users\h\Desktop\AI\data"
```

[5] ✓ 0.0s

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

story = []
```

[6] ✓ 13.6s

```python
story = []

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

        try:    (class) UnicodeDecodeError
            wit                          ing="utf-8") as f:
                Unicode decoding error.
        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))
```

✓ 13.6s Python

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

✓ 0.0s Python

```
145020
[['game', 'of', 'thrones', 'book', 'one', 'of', 'song', 'of', 'ice', 'and', 'fire', 'by', 'george', 'martin', 'prologue', 'we', 'should', 'start', 'back', 'gared', 'urged', 'as', 'the', 'woods', 'be
```

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

✓ 0.0s Python

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

✓ 2.9s

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

✓ 13.0s

```
(6569050, 8628190)
```

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

✓ 0.0s

```
[('stormborn', 0.7747085690498352),
 ('myrcella', 0.7417674660682678),
 ('princess', 0.7254637479782104),
 ('targaryen', 0.7166107296943665),
 ('viserys', 0.6705787777900696),
 ('unburnt', 0.6640740633010864),
 ('margaery', 0.6635355949401855),
 ('queen', 0.6590381264686584),
 ('dorne', 0.6493701934814453),
 ('prince', 0.6415064930915833)]
```

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

✓ 0.0s

```
'jon'
```

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

0.0s

```
'bronn'
```

```
model.wv['king']
```

0.0s

```
array([ 0.26891482,  0.6001591 ,  2.6924682 ,  1.9977348 , -2.0928023 ,
        1.6741775 , -0.74811924,  1.1452627 , -1.9751002 ,  0.2568304 ,
       -1.0746068 ,  0.86136997,  0.500974  ,  2.8437057 , -3.4520917 ,
       -1.4782377 ,  1.6148216 ,  2.1974611 ,  0.72354144,  1.636202  ,
        0.8321633 , -0.9486615 ,  2.9643846 , -3.2509696 , -2.0574193 ,
        2.2818778 , -0.37232143, -1.0752884 , -0.47045296,  1.9520614 ,
       -2.7735584 , -0.2271765 ,  0.5828542 , -0.68766874,  1.1737362 ,
       -1.5275186 , -0.4710071 , -1.7312647 ,  0.64836687, -2.2143753 ,
        0.1098459 ,  1.5535835 ,  1.6918998 ,  0.23302521, -0.33263117,
       -1.9727484 ,  0.02945553, -1.9833945 ,  2.4889457 , -3.0467474 ,
       -2.7870016 , -2.498347  , -1.981903  , -2.5230064 ,  3.9048395 ,
       -2.4079874 ,  1.5737748 ,  1.081367  ,  1.0208126 ,  1.1831667 ,
        1.2238128 ,  1.3786469 ,  0.9803622 ,  1.2135874 , -0.08005533,
        1.9607065 , -0.71088463, -1.6886363 , -1.1306278 , -1.1948683 ,
       -0.2185671 , -0.37058043,  3.992105  , -2.2785435 ,  2.4370081 ,
       -0.2633271 , -1.7713969 , -1.2305627 , -1.8899138 , -3.097366  ,
        1.600244  ,  2.0607677 , -1.9047579 , -2.1947439 , -2.5791237 ,
       -3.3174608 ,  0.8148284 ,  4.4358478 ,  1.1345456 , -0.6091975 ,
        1.0613319 ,  1.5266478 , -0.29228923, -1.6843776 ,  0.23084323,
       -3.454302  , -1.7525158 , -0.992976  , -1.433621  ,  0.1680581 ],
      dtype=float32)
```

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

0.0s

```
np.float32(0.8439563)
```

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

0.0s

```
np.float32(0.254941)
```

```
vectors = model.wv.get_normed_vectors()
words = model.wv.index_to_key

len(words)
```

0.0s

```
17453
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
X = pca.fit_transform(vectors)

X.shape
```

1.0s

```
(17453, 3)
```

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
X = pca.fit_transform(vectors)

X.shape
```

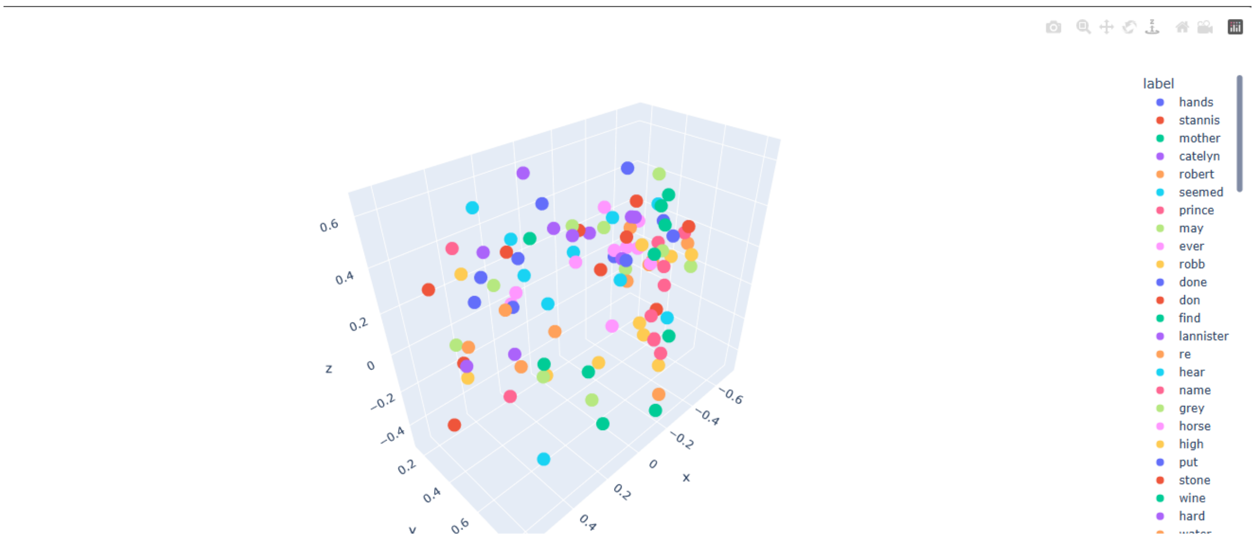
✓ 1.0s

```
(17453, 3)
```

```python
import plotly.express as px
import plotly.io as pio

pio.renderers.default = "browser"

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

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

fig.show()
```

✓ 9.1s

Output:



LAB Questions

Q1.

What is the core idea behind Word2Vec?

Answer:

Word2Vec is a technique that converts words into dense vectors which capture both semantic and syntactic relationships. The main idea is that words appearing in similar contexts have similar meanings. It learns vector representations such that similar words are close together in the vector space.

Q2.

Difference between CBOW and Skip-Gram?

Answer:

Feature	CBOW	Skip-Gram
Goal	Predict target word from context	Predict context words from target
Input	Context words	Target word
Output	Target word	Context words
Speed	Faster, good for frequent words	Slower, better for rare words
Use Case	Frequent words	Rare words

Q3.

Why is one-hot encoding inefficient?

Answer:

One-hot encoding creates very large and sparse vectors that do not show any similarity between words. Every word is independent in this representation, and it requires a lot of memory and computation for large vocabularies.

Q4.

Why do character names appear close in vector space?

Answer:

- Character names often appear in similar contexts like same scenes or dialogues.
- Word2Vec places them close together because it learns similarity from context.

Q5.

How does window size affect semantic learning?

Answer:

Window size determines how many words around the target word are considered while training. A small window focuses more on syntactic relations like grammar, while a large window captures broader semantic relations like topics. If the window is too large, irrelevant words may add noise, while a very small window may miss context.

Q6.

Why might rare characters have poor embeddings?

Answer:

Rare words or characters appear infrequently in the text. Because Word2Vec relies on word co-occurrence in contexts to learn embeddings, rare words get very little information, leading to vectors that do not accurately capture their meaning.

Q7.

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

Answer:

Skip-Gram usually performs better than CBOW, especially for rare words. It predicts multiple context words for each target word, so even rare words get more training examples. CBOW is faster and works well for frequent words, but it may not handle rare words effectively.

Q8.

What happens if vector size is too small or too large?

Answer:

- Too small vector size cannot capture enough information, making embeddings less useful.
- Too large vector size may overfit the training data, capture noise, and require more computation.
- Choosing an optimal size, usually between 100 to 300 dimensions, balances expressiveness and efficiency.

Q9.

Can Word2Vec understand word meaning without labels? Explain.

Answer:

Yes, Word2Vec works in an unsupervised way. It learns the meaning of words purely from the contexts they appear in, without any predefined labels. Semantic similarity is derived from co-occurrence patterns in the text.