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:

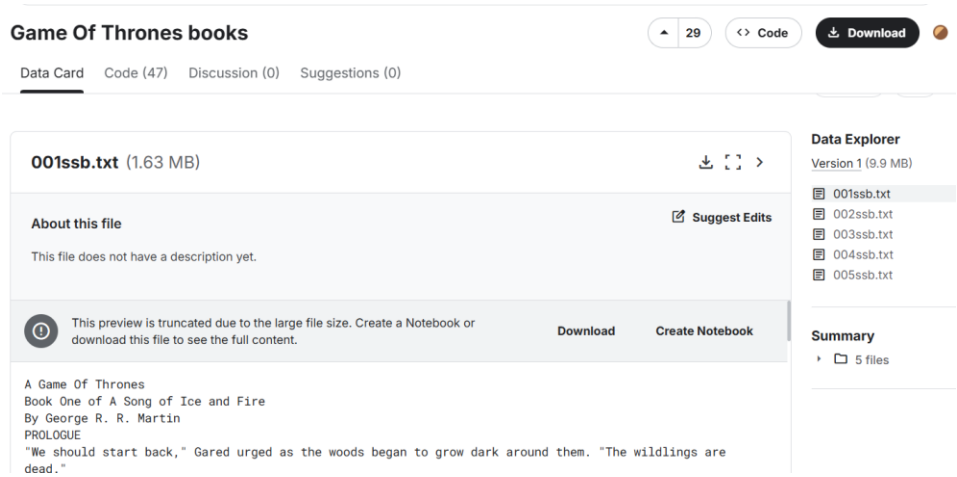
- Understand **distributional semantics**
- Learn how **Word2Vec** converts words into dense vectors
- Apply **Word2Vec (Skip-Gram / CBOW)** on real textual data (Game of Thrones)
- Explore **word similarity, analogy, and visualization**

game-of-thrones-word2vec

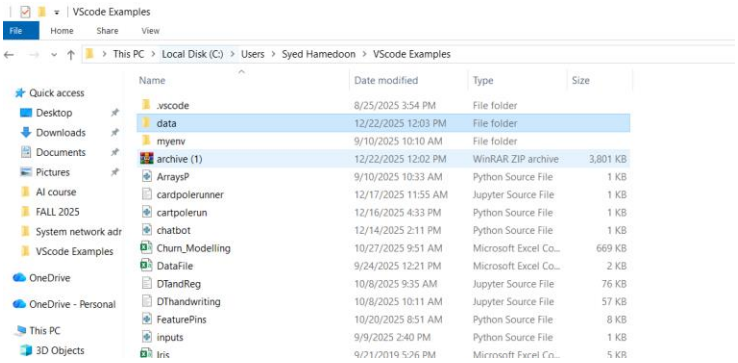
word2vec applied on game of thrones data

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())
```

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

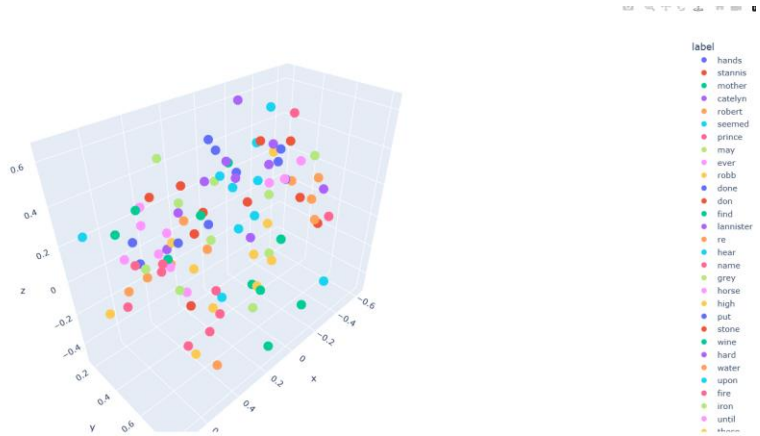
```
import pandas as pd
import plotly.express as px
import plotly.io as pio

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?

Answer:

The core idea behind **Word2Vec** is that **words appearing in similar contexts tend to have similar meanings**.
Word2Vec learns **dense numerical vectors (embeddings)** for words by analyzing surrounding words in large text data.
These vectors capture **semantic and syntactic relationships** between words.

Example:
king and queen appear in similar contexts → their vectors become close in space

2. Difference between CBOW and Skip-Gram?

Answer:

Feature	**CBOW**	**Skip-Gram**
Prediction	Predicts target word from context	Predicts context from target word
Speed	Faster	Slower
Best for	Large datasets	Small datasets
Rare words	Poor performance	Better performance

3. Why is one-hot encoding inefficient?

Answer:

One-hot encoding is inefficient because:

- Vectors are **very large** (equal to vocabulary size)
- Mostly **zeros**, wasting memory
- Cannot capture **semantic similarity**
- No relationship between words

Example:
king and queen have completely different vectors although meanings are related.

---

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

Answer:

Character names appear close because:

- They occur in **similar contexts**
- Appear together in scenes, families, or storylines
- Share roles (e.g., Stark family members)

Example:
arya, sansa, bran → frequently mentioned together → similar embeddings

5. How does window size affect semantic learning?

Answer:

- Small window size (2–5):
    - Learns syntactic relationships
    - Focuses on nearby words
- Large window size (8–15):
    - Learns semantic relationships
    - Captures broader context

6. Why might rare characters have poor embeddings?

Answer:

Rare characters have poor embeddings because:

- They appear **few times** in the corpus
- Model cannot learn enough context
- Limited co-occurrence with other words

---

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

Answer:

Skip-Gram performed better for this dataset because:

- Game of Thrones has many **rare character names**
- Skip-Gram handles rare words more effectively
- Produces more meaningful embeddings for names

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

Answer:

Too small vector size:

- Cannot capture enough semantic information
- Poor similarity results

Too large vector size:

- Overfitting
- Slower training
- Higher memory usage

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

Answer:

Yes, **Word2Vec learns word meaning without labels** because:

- It uses **unsupervised learning**
- Learns from **word co-occurrence patterns**
- Discovers semantic relationships automatically