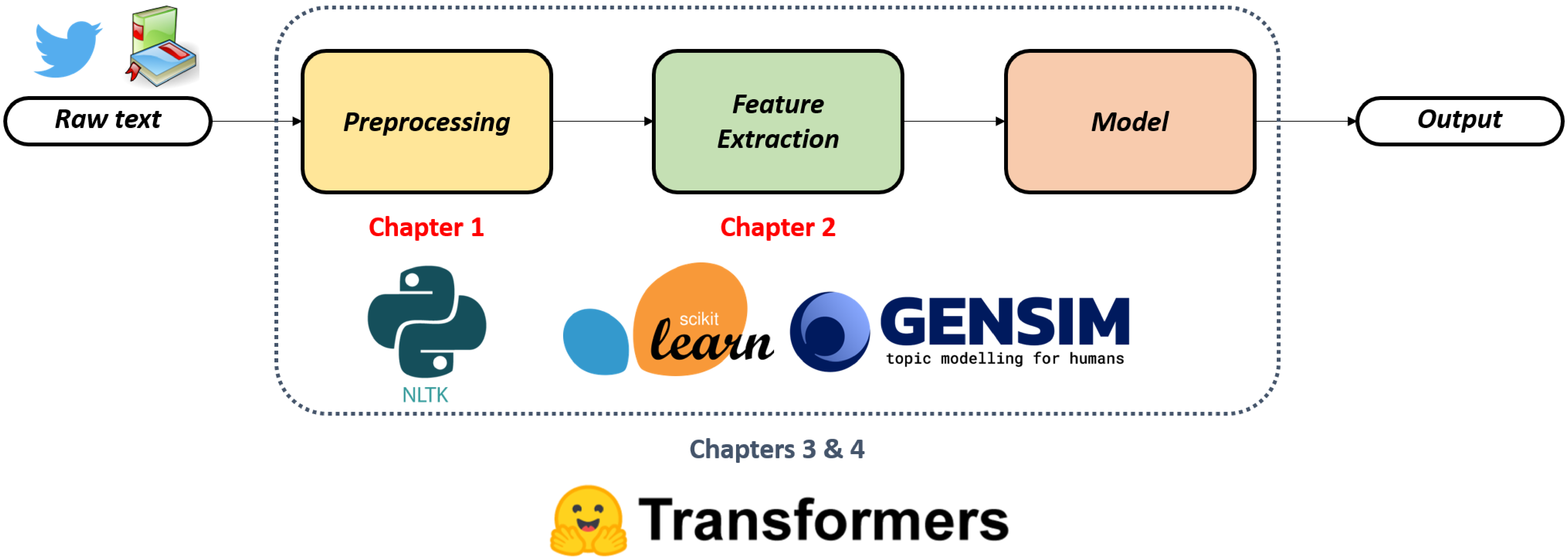
Bag-of-Words representation

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON

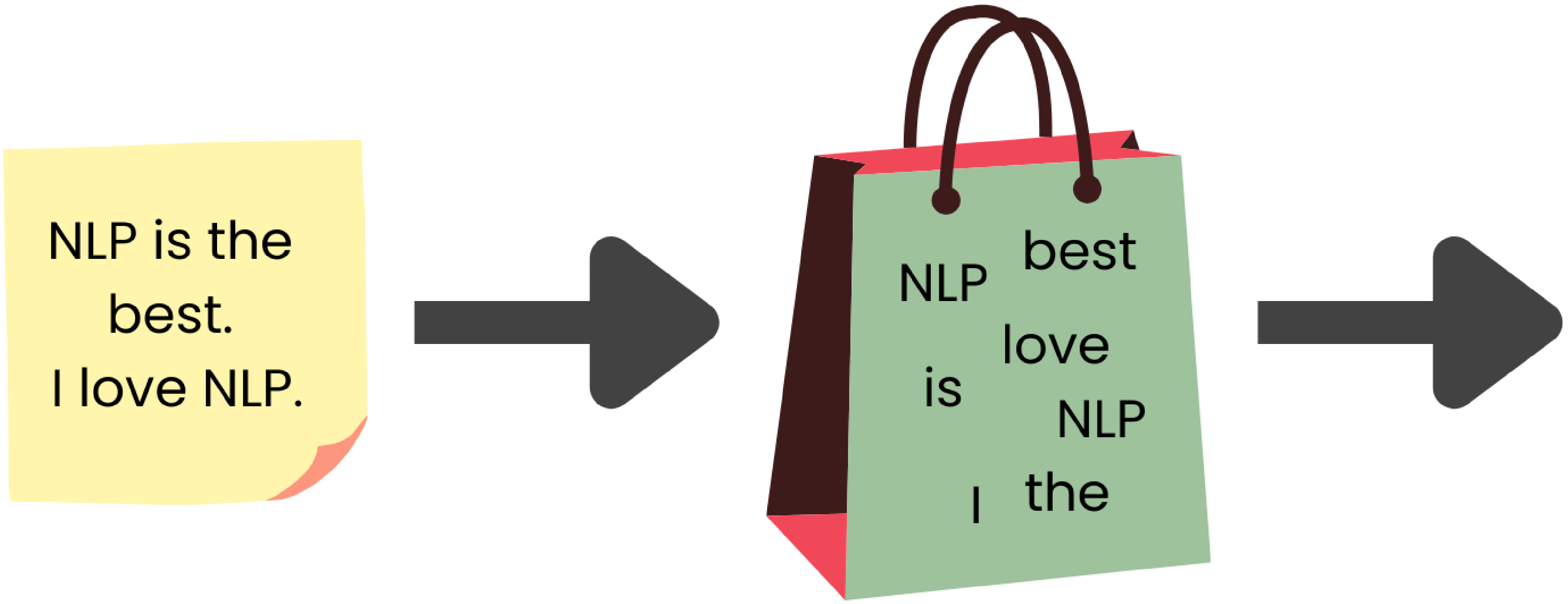
Fouad Trad
Machine Learning Engineer

NLP workflow recap



Bag-of-Words (BoW)

- Foundational technique to represent text as numbers
- Represent text by counting how often each word appears
- Throws words in a bag and counts them
- Ignores grammar and order



NLP	2
I	1
love	1
best	1
is	1
the	1

BoW example

Sentence
I love NLP
I love machine learning

BoW example

Sentence	I	love	NLP	Machine	Learning
I love NLP					
I love machine learning					

- Build a vocabulary of all unique words

BoW example

Sentence	I	love	NLP	Machine	Learning
I love NLP	1	1	1	0	0
I love machine learning	1	1	0	1	1

- Build a vocabulary of all unique words
- Count how many times each word from the vocabulary appears

BoW with code

```
reviews = ["I loved the movie. It was amazing!",
           "The movie was okay.",
           "I hated the movie. It was boring."]
def preprocess(text):
    text = text.lower()
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in string.punctuation]
    return " ".join(tokens)
cleaned_reviews = [preprocess(review) for review in reviews]
print(cleaned_reviews)
```

```
['i loved the movie it was amazing',
 'the movie was okay',
 'i hated the movie it was boring']
```

BoW with code

```python
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(cleaned_reviews)
print(vectorizer.get_feature_names_out())
```

```
['amazing' 'boring' 'hated' 'it' 'loved' 'movie' 'okay' 'the' 'was']
```

BoW output

```
X = vectorizer.transform(cleaned_reviews)
# OR
X = vectorizer.fit_transform(cleaned_reviews)
print(X)
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
    with 16 stored elements and shape (3, 9)>
```

Sparse matrix: table mostly filled with zeros

BoW output

```
print(X.toarray())
```

```
[[1 0 0 1 1 1 0 1 1]
 [0 0 0 0 0 1 1 1 1]
 [0 1 1 1 0 1 0 1 1]]
```

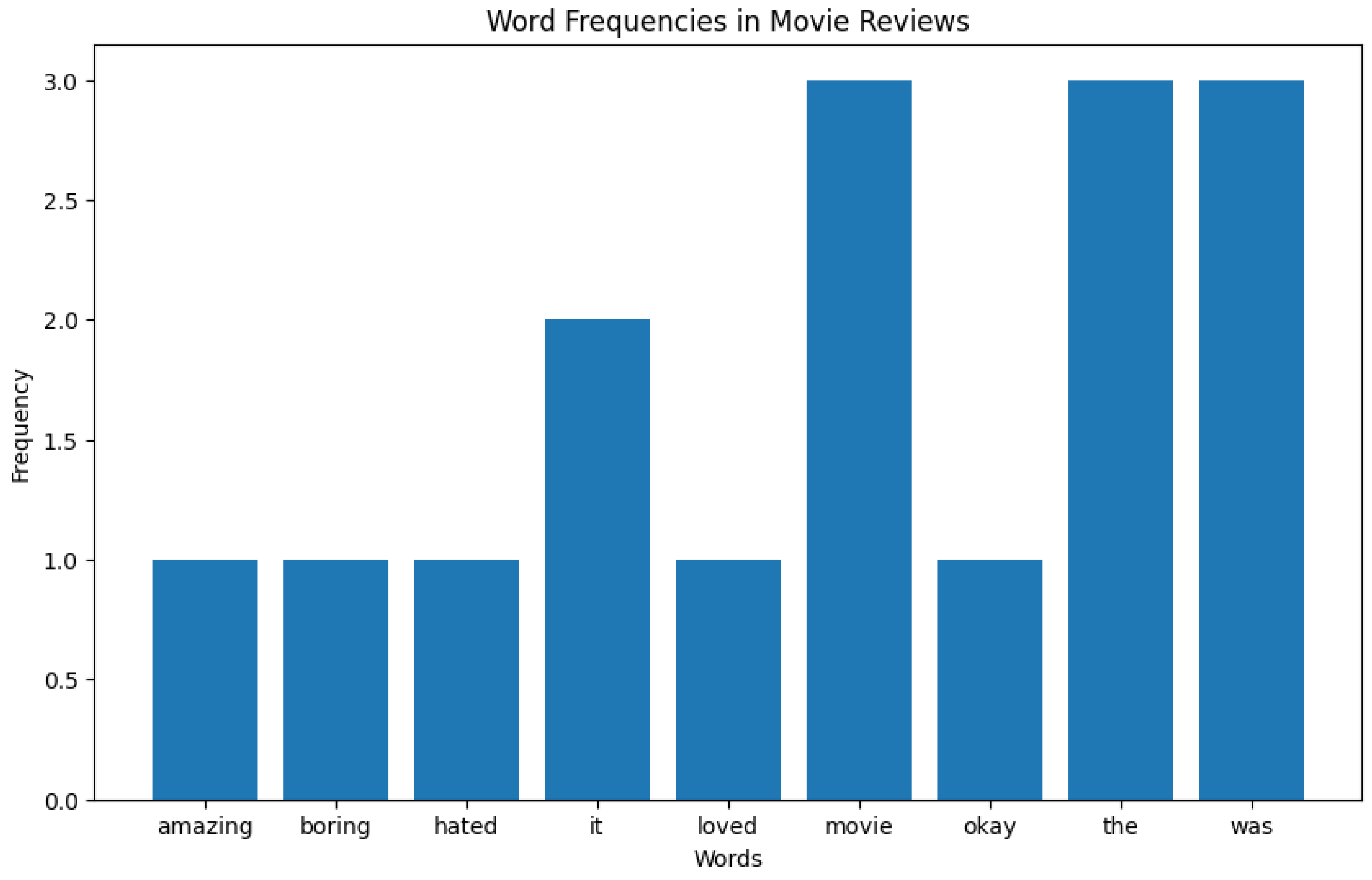
```
print(vectorizer.get_feature_names_out())
```

```
['amazing' 'boring' 'hated' 'it' 'loved' 'movie' 'okay' 'the' 'was']
```

Word frequencies

```
import numpy as np

word_counts = np.sum(X.toarray(), axis=0)
words = vectorizer.get_feature_names_out()
```

```
import matplotlib.pyplot as plt

plt.bar(words, word_counts)
plt.title("Word Frequencies in Movie Reviews")
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.show()
```



Let's practice!

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON

TF-IDF vectorization

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON

Fouad Trad
Machine Learning Engineer

From BoW to TF-IDF

- BoW treats all words as equally important
- TF-IDF fixes this by telling:
  - how often a word appears in a document
  - how meaningful that word is across a collection

Sentence	I	love	this	NLP	course	enjoyed	project
I love this NLP course	1	1	1	1	1	0	0
I enjoyed this project	1	0	1	0	0	1	1

TF-IDF

$$TF\ IDF = TF \times IDF$$

TF-IDF

$$TF\ IDF = TF \times IDF$$

- TF: Term Frequency
  - How many times a word appears in a document

TF-IDF

$$TF\ IDF = TF \times IDF$$

- TF: Term Frequency
  - How many times a word appears in a document
- IDF: Inverse Document Frequency
  - How rare that word is across all documents

- Word shows up in one document, not in others → high score
- Word appears in every document → low score

TF-IDF with code

```
reviews = [
    "I loved the movie. It was amazing!",
    "The movie was okay.",
    "I hated the movie. It was boring."
]
cleaned_reviews = [preprocess(review) for review in reviews]
print(cleaned_reviews)
```

```
['i loved the movie it was amazing',
 'the movie was okay',
 'i hated the movie it was boring']
```

TF-IDF with code

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(cleaned_reviews)
print(tfidf_matrix)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
    with 16 stored elements and shape (3, 9)>
```

TF-IDF output

```
print(tfidf_matrix.toarray())
```

```
[[0.52523431 0.         0.         0.39945423 0.52523431 0.31021184   0.         0.31021184 0.31021184]
 [0.         0.         0.         0.         0.         0.41285857   0.69903033 0.41285857 0.41285857]
 [0.         0.52523431 0.52523431 0.39945423 0.         0.31021184   0.         0.31021184 0.31021184]]
```

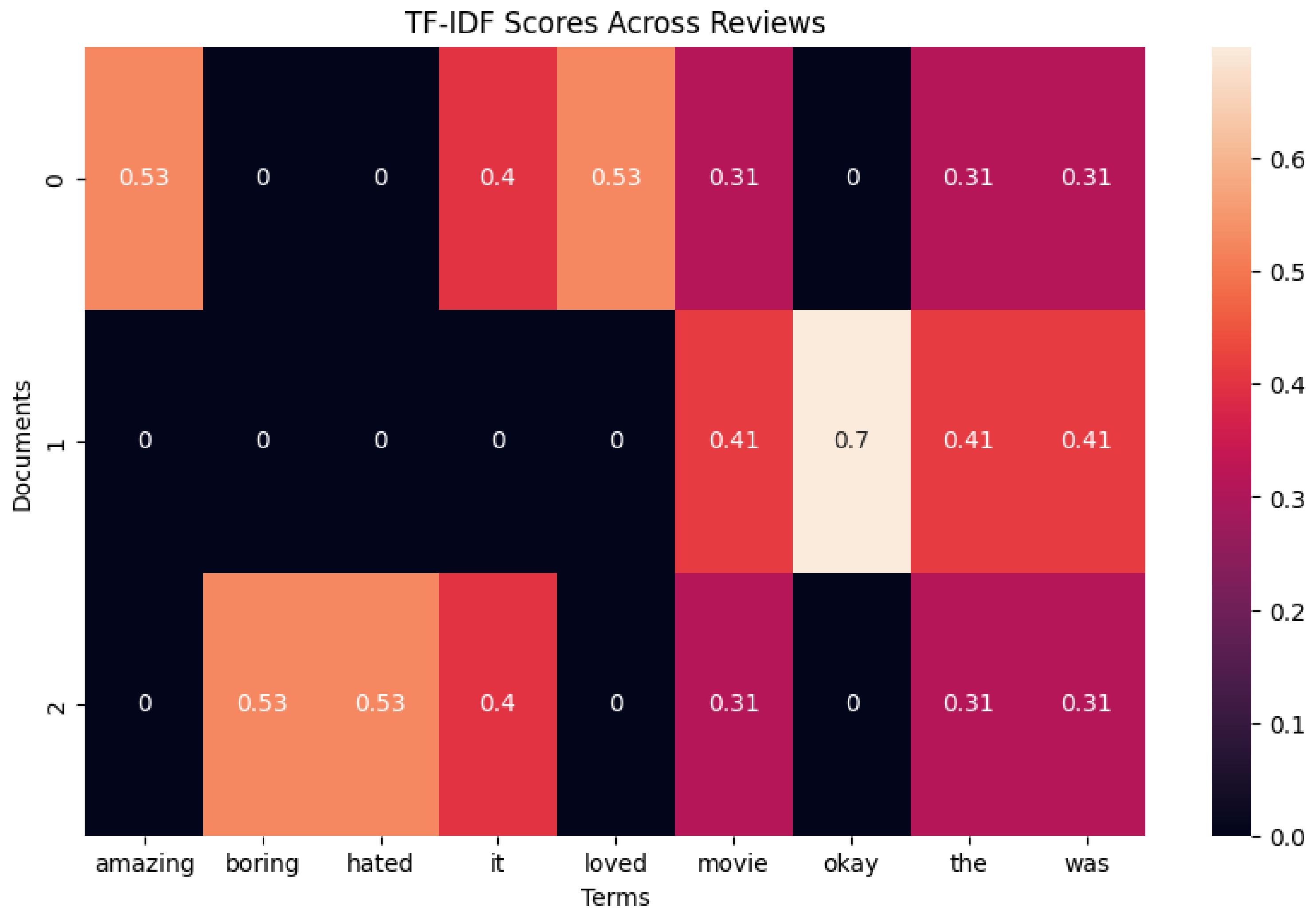
```
vectorizer.get_feature_names_out()
```

```
['amazing' 'boring' 'hated' 'it' 'loved' 'movie' 'okay' 'the' 'was']
```

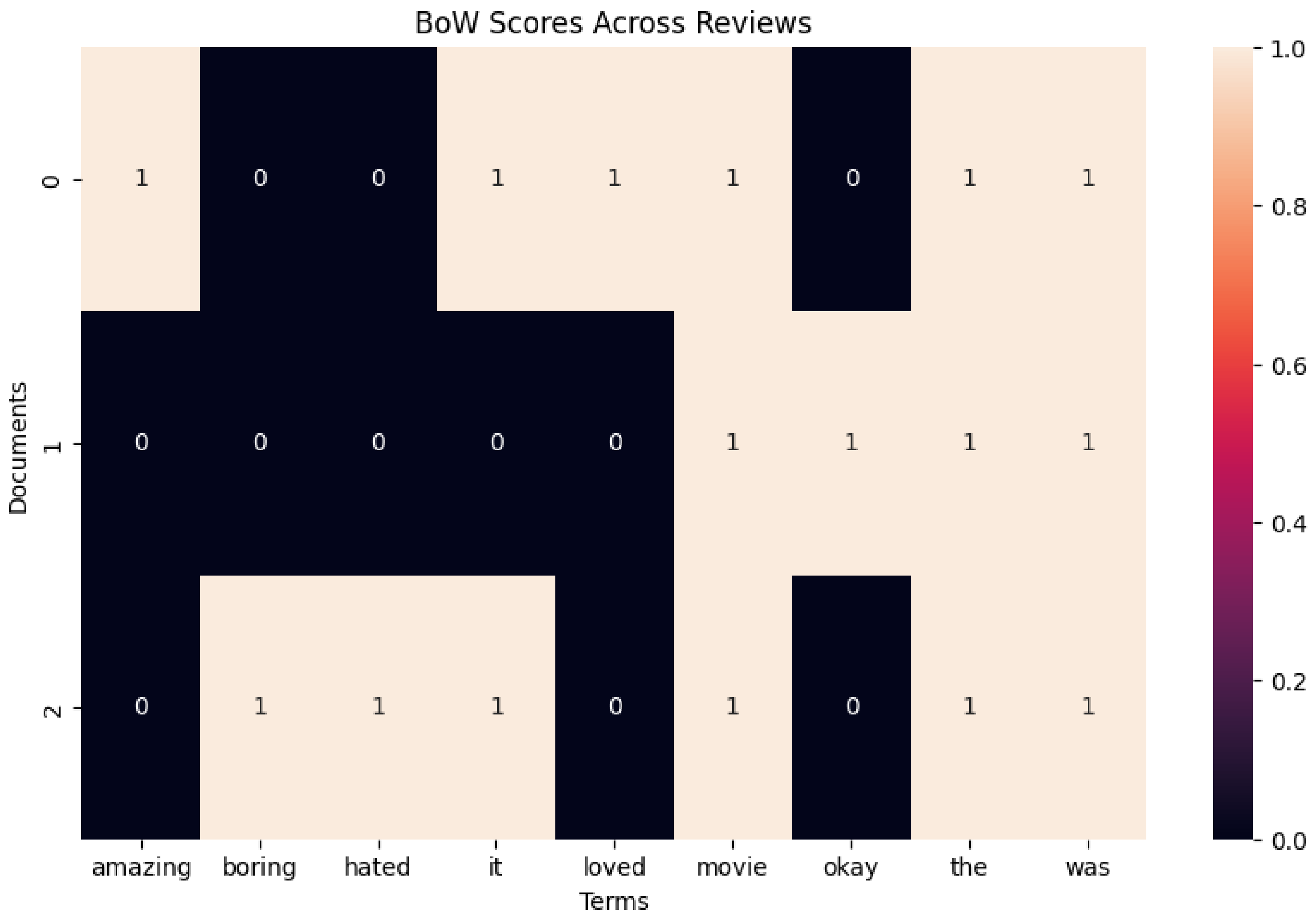
Visualizing scores as heatmap

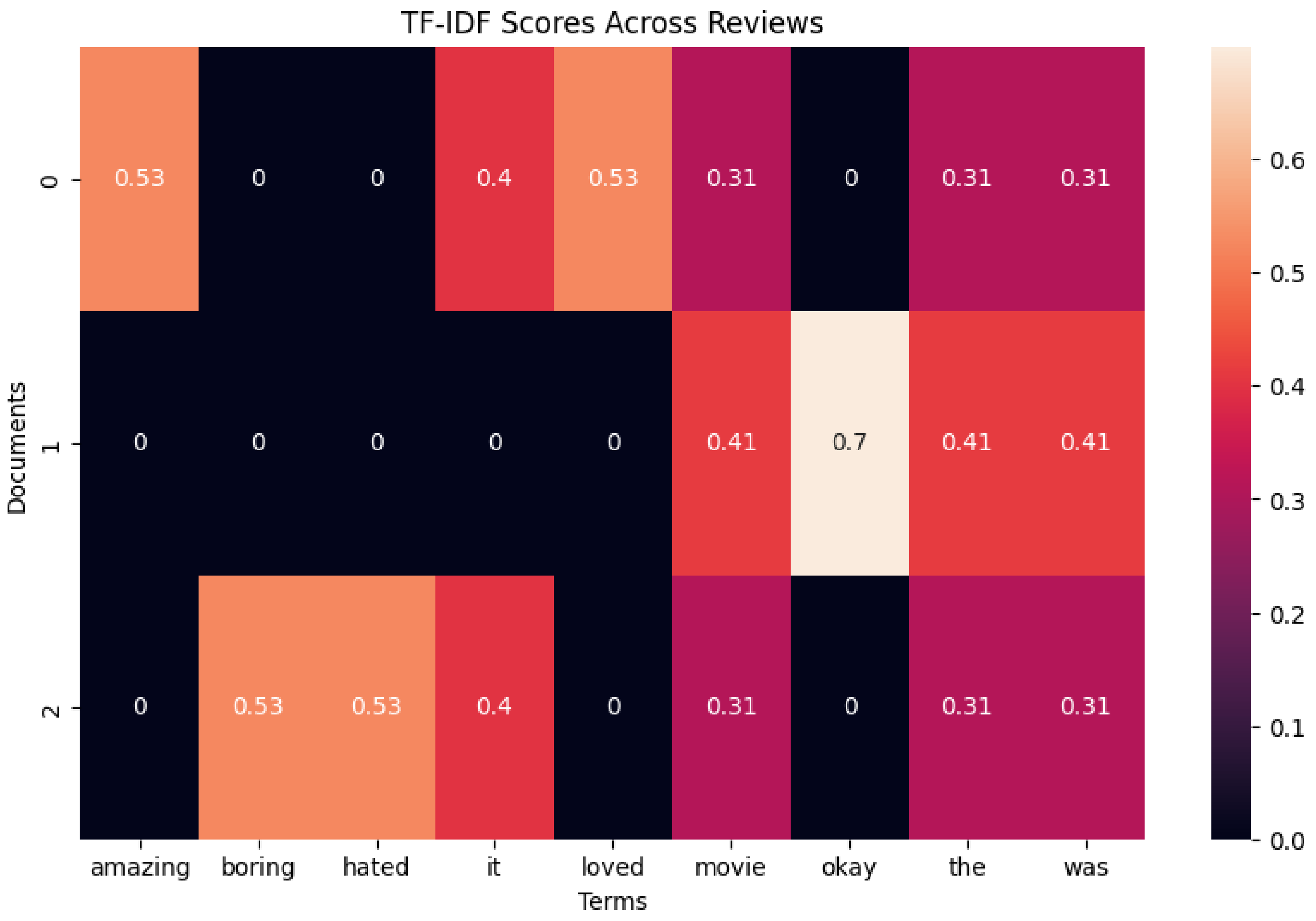
```
import pandas as pd

df_tfidf = pd.DataFrame(
    tfidf_matrix.toarray(),
    columns=vectorizer.get_feature_names_out()
)
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df_tfidf, annot=True)
plt.title("TF-IDF Scores Across Reviews")
plt.xlabel("Terms")
plt.ylabel("Documents")
plt.show()
```



Comparing with BoW





Let's practice!

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON

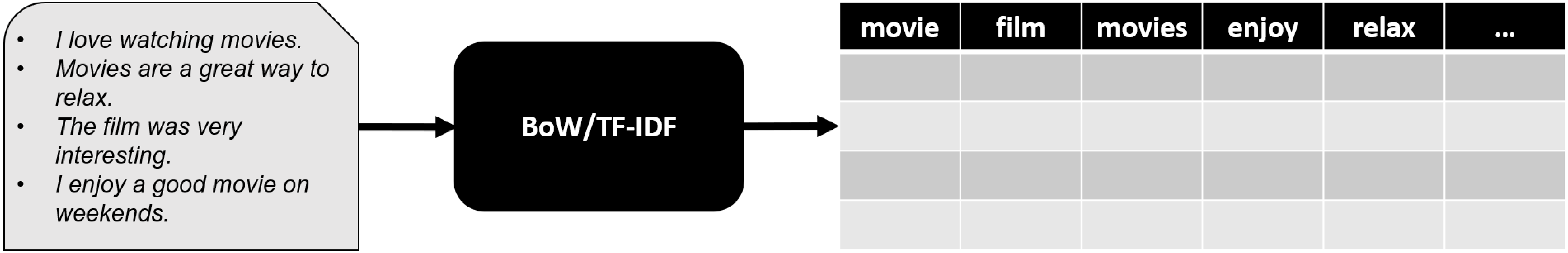
Embeddings

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON

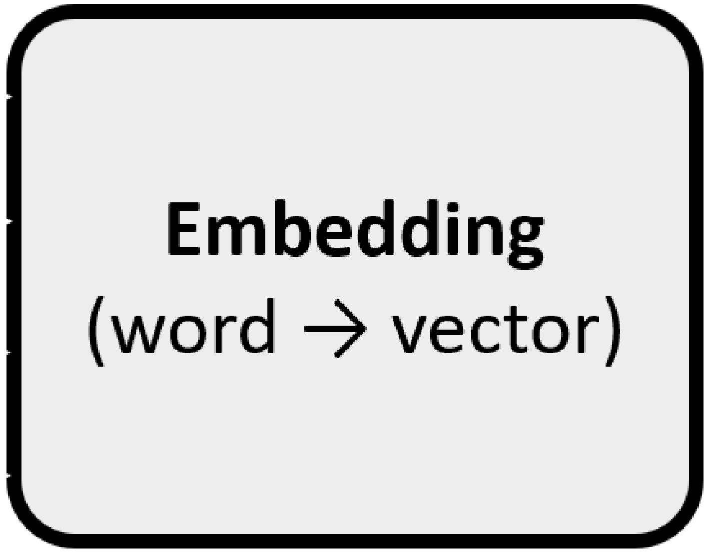
Fouad Trad
Machine Learning Engineer

Limitations of BoW and TF-IDF

- Treating similar words as completely unrelated
- Failing to capture meaning of text

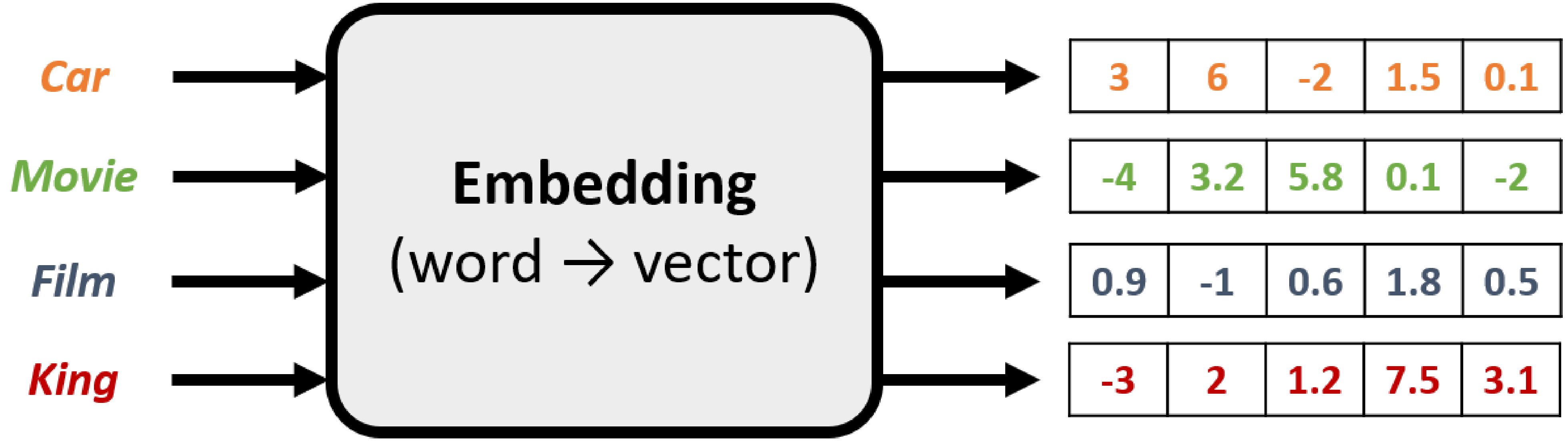


Embeddings



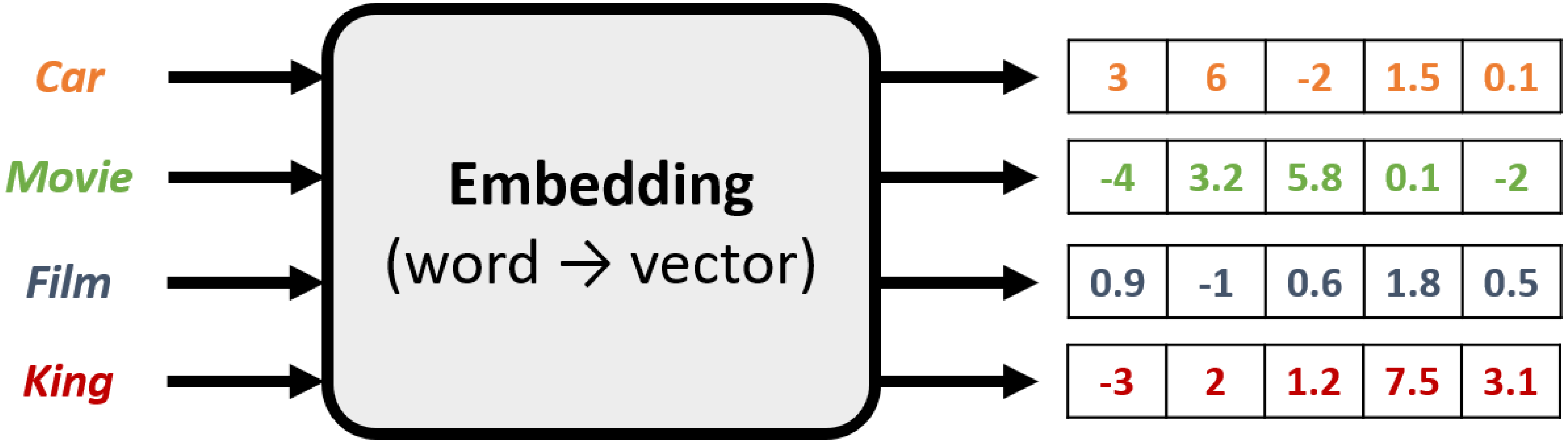
- Represent a word with a vector that captures its meaning

Embeddings



- Represent a word with a vector that captures its meaning
  - Assigns random values to each word

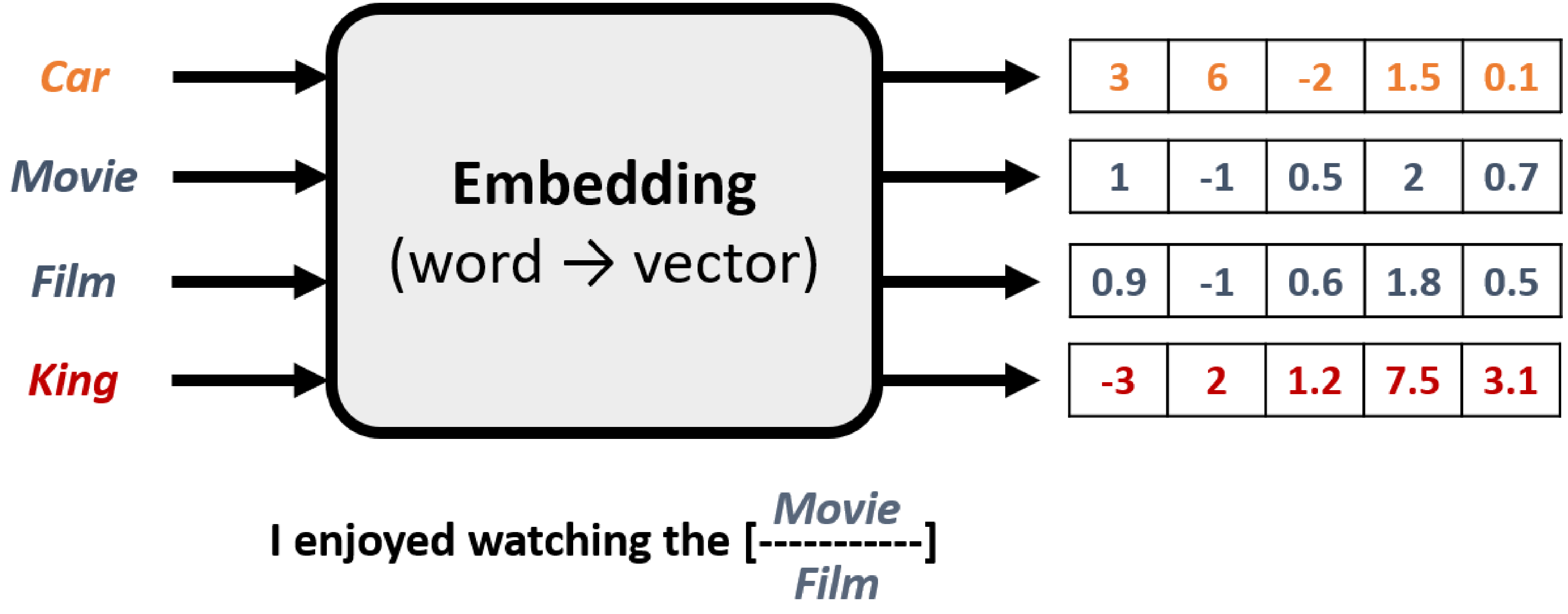
Embeddings



I enjoyed watching the [-----------]

- Represent a word with a vector that captures its meaning
  - Assigns random values to each word
  - Refines values by predicting missing words in sentences

Embeddings



- Represent a word with a vector that captures its meaning
  - Assigns random values to each word
  - Refines values by predicting missing words in sentences
  - Words appearing in similar contexts end up with similar representations

Embeddings as GPS coordinates for words

Gensim

- Provides popular embedding models
  - Word2Vec
  - GloVe

```
word2vec-ruscorpora-300
word2vec-google-news-300
glove-wiki-gigaword-50
glove-wiki-gigaword-100
glove-wiki-gigaword-200
glove-wiki-gigaword-300
...
```

GENSIM
topic modelling for humans

Loading an embedding model

```
import gensim.downloader as api
model = api.load('glove-wiki-gigaword-50')
print(type(model))
print(model['movie'])
```

```
<class 'gensim.models.keyedvectors.KeyedVectors'>
[ 0.30824   0.17223  -0.23339   0.023105  0.28522   0.23076  -0.41048
 -1.0035   -0.2072    1.4327   -0.80684   0.68954  -0.43648   1.1069
  1.6107   -0.31966   0.47744   0.79395  -0.84374   0.064509  0.90251
  0.78609   0.29699   0.76057   0.433    -1.5032   -1.6423    0.30256
  0.30771  -0.87057   2.4782   -0.025852  0.5013   -0.38593  -0.15633
  0.45522   0.04901  -0.42599  -0.86402  -1.3076   -0.29576   1.209
 -0.3127   -0.72462  -0.80801   0.082667  0.26738  -0.98177  -0.32147
  0.99823 ]
```

Computing similarity

```
similarity = model.similarity("film", "movie")
print(similarity)
```

```
0.9310100078582764
```

Finding most similar words

```
similar_to_movie = model.most_similar('movie', topn=3)
print(similar_to_movie)
```

```
[('movies', 0.9322481155395508),
 ('film', 0.9310100078582764),
 ('films', 0.8937394618988037)]
```

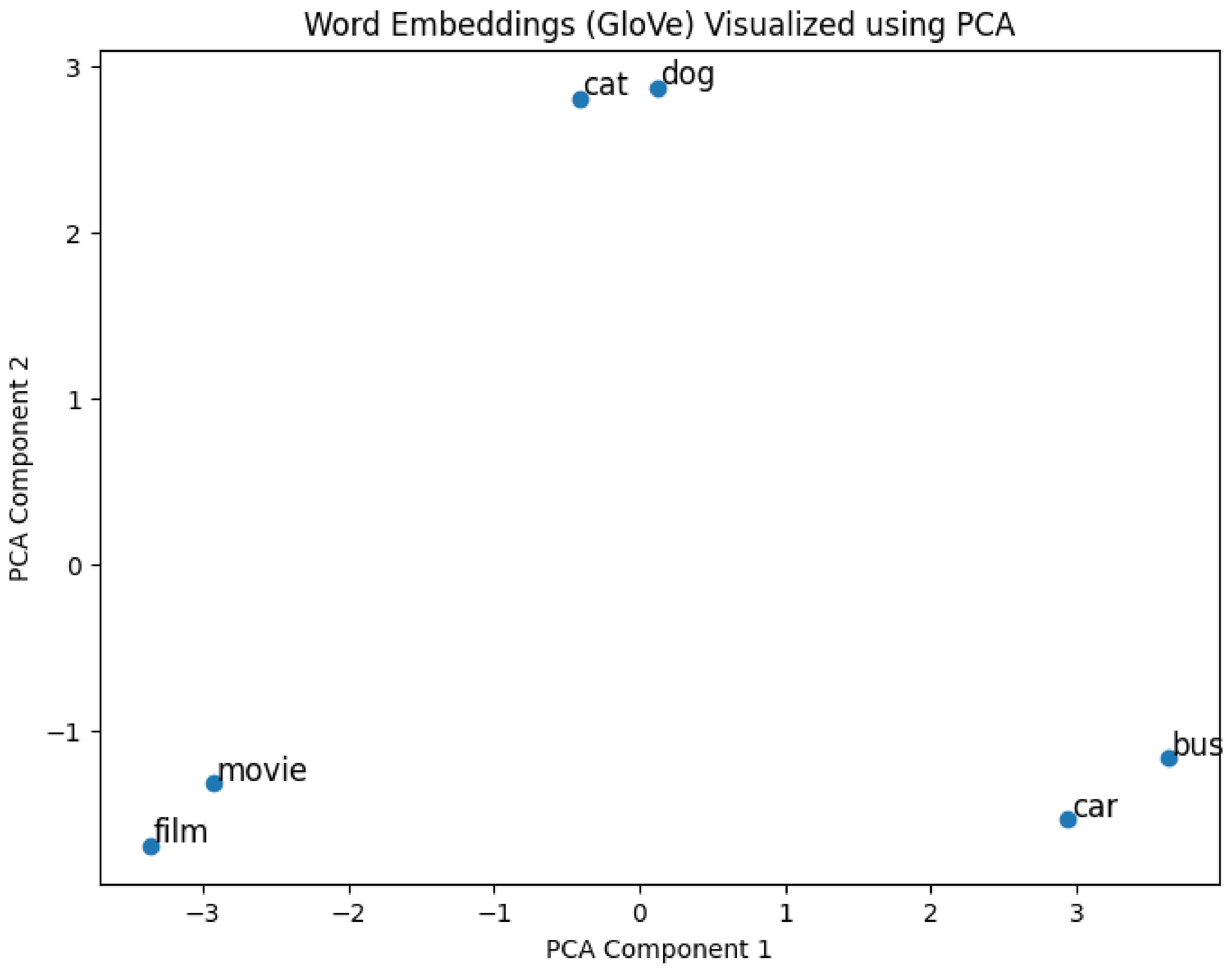
Visualizing embeddings

- Principal Component Analysis (PCA):
  - High-dimensional vectors → 2D or 3D vectors

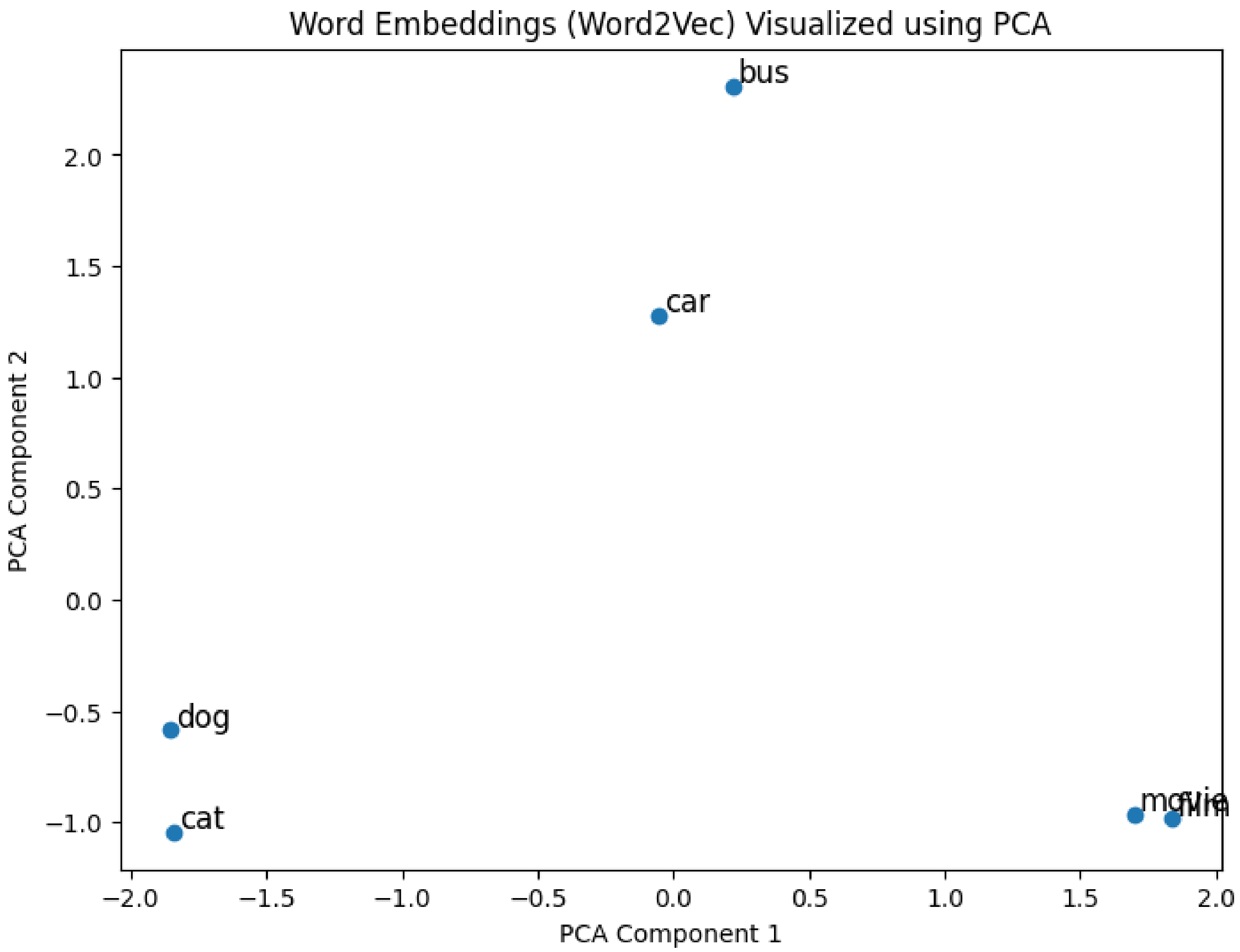
[1] Image generated by DALL-E

Visualizing embeddings with PCA

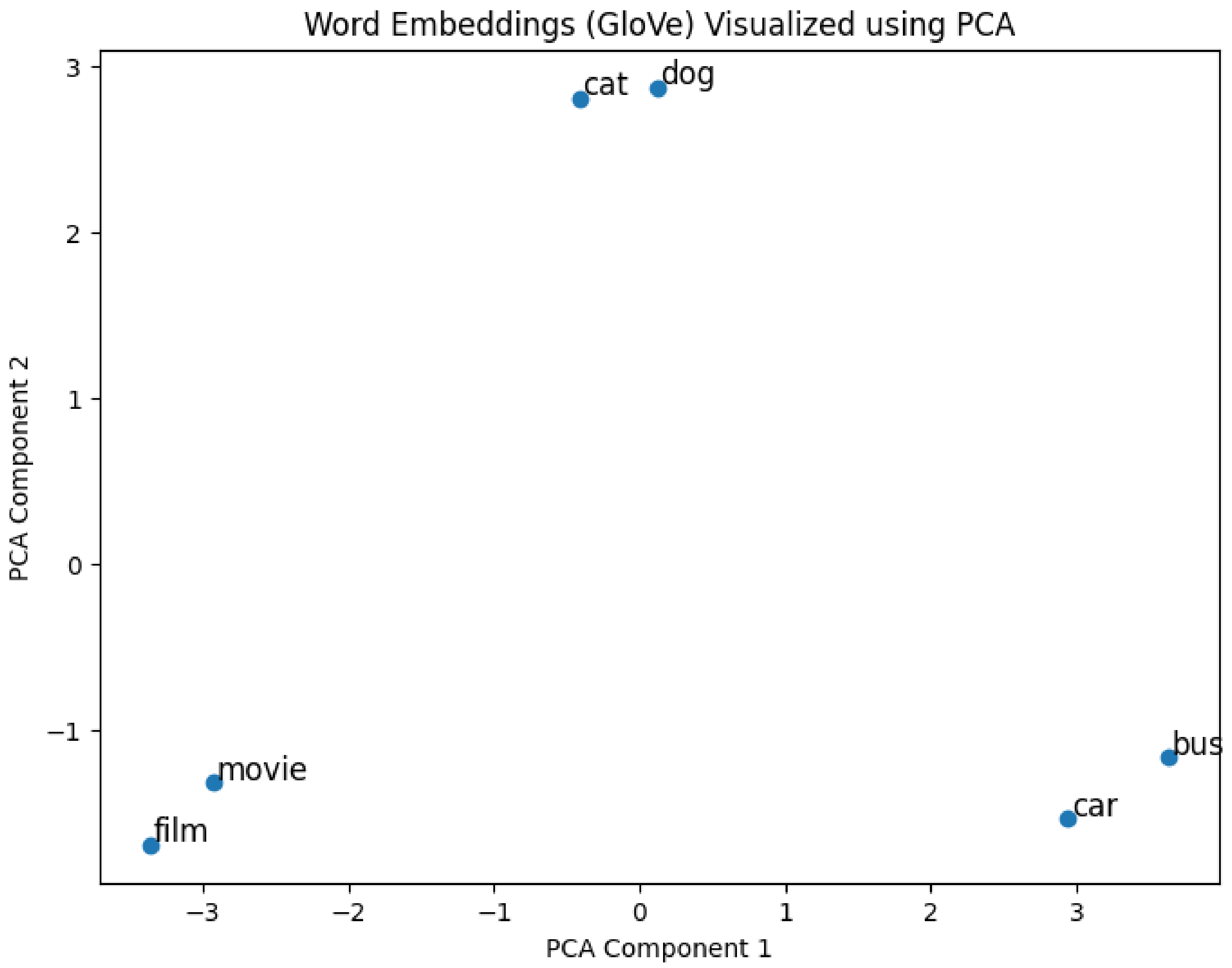
```
from sklearn.decomposition import PCA

words = ["film", "movie", "dog", "cat", "car", "bus"]
word_vectors = [model[word] for word in words]

pca = PCA(n_components=2)
word_vectors_2d = pca.fit_transform(word_vectors)

plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1])
for word, (x, y) in zip(words, word_vectors_2d):
    plt.annotate(word, (x, y))
plt.show()
```



Comparison of embeddings



`word2vec-google-news-300`



`glove-wiki-gigaword-50`

Let's practice!

NATURAL LANGUAGE PROCESSING (NLP) IN PYTHON