

✓ Represent word use model GloVe

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

import kagglehub

# Download latest version
path = kagglehub.dataset_download("danielwillgeorge/glove6b100dtxt")

print("Path to dataset files:", path)

Using Colab cache for faster access to the 'glove6b100dtxt' dataset.
Path to dataset files: /kaggle/input/glove6b100dtxt
```

```
!unzip /content/glove6b100dtxt.zip

Archive: /content/glove6b100dtxt.zip
replace glove.6B.100d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: glove.6B.100d.txt      y
```

```
def load_glove_embeddings(file_path):
    embeddings = {}
    with open(file_path, 'r', encoding='utf8') as f:
        for line in f:
            values = line.strip().split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings[word] = vector
    return embeddings

glove = load_glove_embeddings("/content/glove.6B.100d.txt")
print(f"Number of words in GloVe: {len(glove)}")

Number of words in GloVe: 400000
```

```
vectors = np.array([e for e in list(glove.values())])
```

✓ Reduce dimension

Using PCA

```
pca = PCA(n_components=2)
reduced_pca = pca.fit_transform(vectors)

def visualization(reduced, method="PCA", draw_vectors=True):
    plt.figure(figsize=(10, 8))

    plt.scatter(reduced[:, 0], reduced[:, 1], color='steelblue')

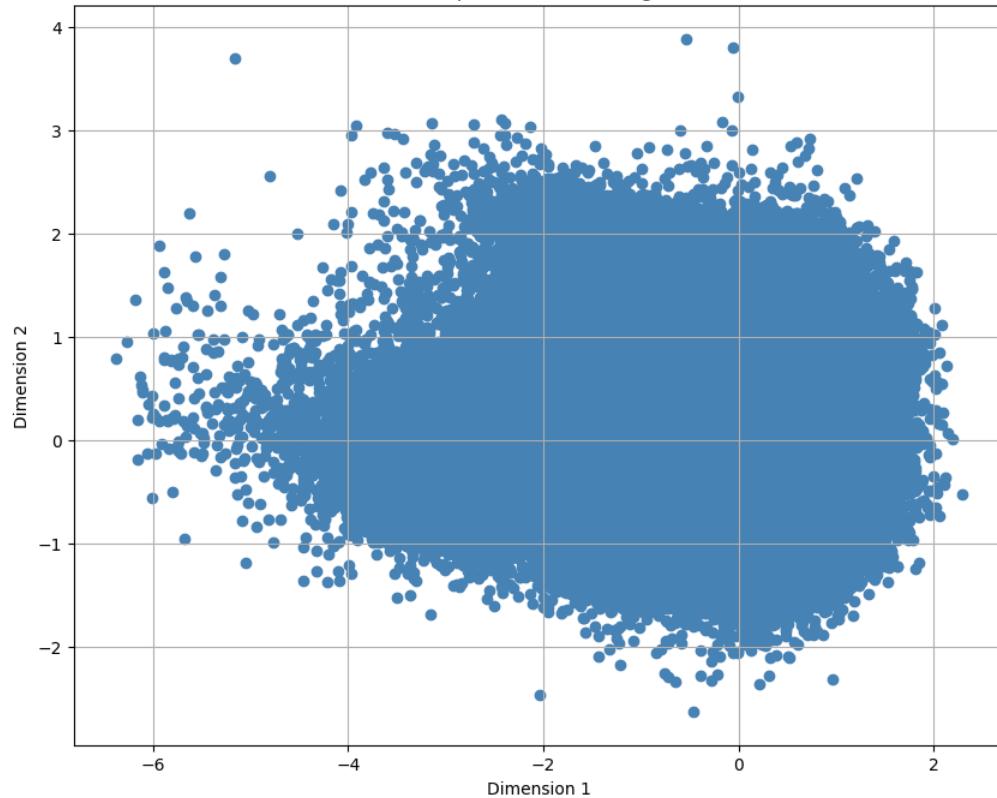
    plt.title(f"Word representations using {method}")
    plt.xlabel("Dimension 1")
    plt.ylabel("Dimension 2")

    plt.grid(True)
    plt.show()

visualization(reduced_pca, "PCA")
```



Word representations using PCA



Nhận xét về phân bố của vector 2D của các từ trong từ điển:

- Các từ phân bố tập trung ở gần điểm (-1, 0)
- Số ít các từ nằm rải rác ở phía bên ngoài
- Sau khi giảm số chiều còn 2 chiều bằng PCA, không có điểm nào có toạ độ x vượt quá 3 (do đã chuẩn hoá khi áp dụng PCA)

Using TSNE

```
from sklearn.manifold import TSNE

def reduce_to_2d(embeddings_index, n_words=20000):
    words = list(embeddings_index.keys())[:n_words]
    vectors = np.array([embeddings_index[word] for word in words])

    tsne = TSNE(n_components=2, random_state=42)
    reduced_vectors = tsne.fit_transform(vectors)

    reduced_embeddings = dict(zip(words, reduced_vectors))
    return reduced_embeddings

# Reduce dimensionality
glove_2d = reduce_to_2d(glove, n_words=15000)
```

Clustering with Kmeans

```
from sklearn.cluster import KMeans

embedding_matrix = glove_2d.copy()

# Prepare data for K-means clustering
words = list(embedding_matrix.keys())
embeddings = np.array(list(embedding_matrix.values()))

# Perform K-means clustering
num_clusters = 30
kmeans = KMeans(n_clusters=num_clusters, random_state=42).fit(embeddings)
labels = kmeans.labels_
```

```
import plotly.graph_objs as go
import plotly.express as px
```

```

interesting_clusters = [1, 3, 6, 7, 10, 11, 15, 20, 19, 24, 29]

def plot_embedding_space(embedding_matrix, labels, clusters_to_plot=[1, 2, 3, 4, 5]):
    fig = go.Figure()

    for cluster_idx in clusters_to_plot:
        cluster_indices = np.where(labels == cluster_idx - 1)[0]
        cluster_points = embeddings[cluster_indices]
        cluster_words = [words[i] for i in cluster_indices]

        scatter = go.Scatter(
            x=cluster_points[:, 0],
            y=cluster_points[:, 1],
            mode='markers',
            text=cluster_words,
            textposition='top center',
            marker=dict(size=8, opacity=0.8),
            name=f'Cluster {cluster_idx}',
        )
        fig.add_trace(scatter)

    # Customize the plot
    fig.update_layout(
        title='2D Word Embeddings with K-means Clustering (15000 words)',
        xaxis_title='Dimension 1',
        yaxis_title='Dimension 2',
        legend_title='Clusters',
    )

    # Show the plot
    fig.show()

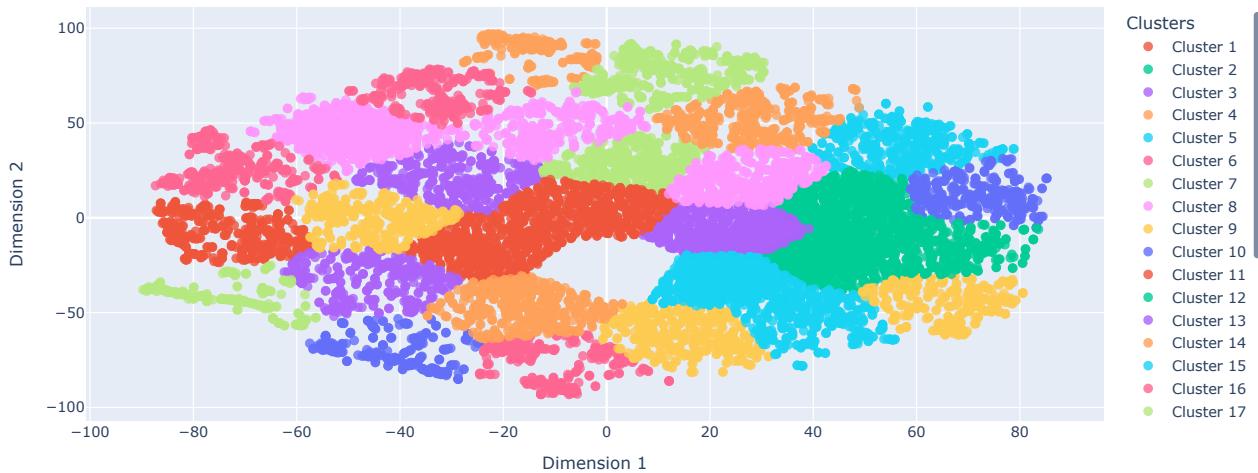
```

```

clusters = [i for i in range(30)]
plot_embedding_space(embedding_matrix, labels, clusters_to_plot=clusters)

```

2D Word Embeddings with K-means Clustering (15000 words)



Nhận xét:

- Các cụm có phân bố tương đối tách biệt, nhưng một số cụm giao nhau ở vùng trung tâm, cho thấy một số các vector từ có nghĩa gần nhau
- Một số cụm nằm rải rác ở biên, thể hiện những nhóm từ có nghĩa đặc trưng hoặc ít liên hệ với phần còn lại
- Có thể sử dụng thêm phương pháp EBOW để tìm số cụm phù hợp khi phân cụm với K-Means

❖ Cosine similarity search word

```

def cosine_similarity(vec1, vec2):
    return np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2))

def find_similar_words(target_word, embeddings, top_k=10):
    if target_word not in embeddings:
        raise ValueError(f"{target_word} is not in vocabulary.")

```

```

target_vec = embeddings[target_word]
similarities = {}

for word, vec in embeddings.items():
    if word == target_word:
        continue
    sim = cosine_similarity(target_vec, vec)
    similarities[word] = sim

sorted_words = sorted(similarities.items(), key=lambda x: x[1], reverse=True)
similar_words = sorted_words[:top_k]
antonym_words = sorted_words[-top_k:]
return similar_words, antonym_words

```

```

target_word = "king"
top_similar, top_antonym = find_similar_words(target_word, glove, top_k=10)

print(f"Top 10 closest word with '{target_word}':\n")
for word, sim in top_similar:
    print(f"{word:<15} similarity = {sim:.4f}")

print(f"\nTop 10 antonymest word with '{target_word}':\n")
for word, sim in top_antonym:
    print(f"{word:<15} similarity = {sim:.4f}")

Top 10 closest word with 'king':

prince      similarity = 0.7682
queen       similarity = 0.7508
son          similarity = 0.7021
brother     similarity = 0.6986
monarch     similarity = 0.6978
throne       similarity = 0.6920
kingdom     similarity = 0.6811
father       similarity = 0.6802
emperor     similarity = 0.6713
ii           similarity = 0.6676

Top 10 antonymest word with 'king':

higher-grade  similarity = -0.5601
s.s.d.        similarity = -0.5649
neuters       similarity = -0.5652
cheminformatics  similarity = -0.5657
nanophotonics  similarity = -0.5747
nizhal        similarity = -0.5763
biohazards     similarity = -0.5795
asie          similarity = -0.5809
recreative     similarity = -0.5941
in-process     similarity = -0.5990

```

▼ Visualize in 2D plot

```

closest_word = [w[0] for w in top_similar]
closest_word.append(target_word)

vectors = np.array([glove[w] for w in closest_word if w in glove])
reduced_vector_pca = pca.fit_transform(vectors)

vectors = np.array([glove[word] for word in closest_word])

tsne = TSNE(n_components=2, random_state=42, perplexity=5)
reduced_vector_tsne = tsne.fit_transform(vectors)

reduced_embeddings = dict(zip(closest_word, reduced_vector_tsne))

import matplotlib.pyplot as plt

def visualization(reduced, labels, method="PCA", keyword=None, draw_vectors=True):
    plt.figure(figsize=(10, 8))

    if keyword and keyword in labels:
        idx_keyword = labels.index(keyword)
        x_kw, y_kw = reduced[idx_keyword]
    else:
        idx_keyword = None

    plt.scatter(reduced[:, 0], reduced[:, 1], color='steelblue', label='Words')

    if idx_keyword is not None:
        plt.scatter(x_kw, y_kw, color='crimson', s=150, edgecolor='black', label=f"Keyword: {keyword}", zorder=5)

```

```

for i, word in enumerate(labels):
    color = 'crimson' if word == keyword else 'black'
    weight = 'bold' if word == keyword else 'normal'
    plt.text(reduced[i, 0] + 0.02, reduced[i, 1] + 0.02, word, fontsize=12, color=color, fontweight=weight)

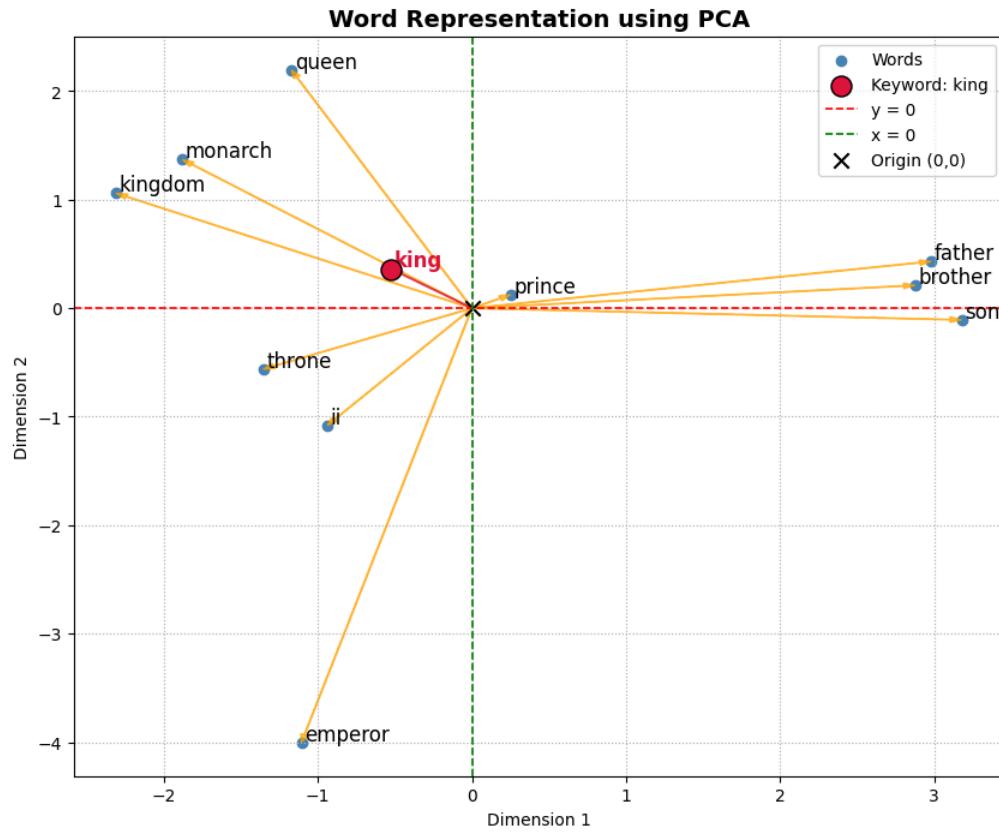
if draw_vectors:
    for i in range(len(labels)):
        color = 'orange' if labels[i] != keyword else 'crimson'
        plt.arrow(
            0, 0,
            reduced[i, 0], reduced[i, 1],
            color=color, alpha=0.7, width=0.005,
            head_width=0.05, length_includes_head=True,
            zorder=1
        )
plt.axhline(0, color='red', linewidth=1.2, linestyle='--', label='y = 0')
plt.axvline(0, color='green', linewidth=1.2, linestyle='--', label='x = 0')

plt.scatter(0, 0, color='black', s=80, marker='x', label='Origin (0,0)')

plt.title(f"Word Representation using {method}", fontsize=14, fontweight='bold')
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.grid(True, linestyle=':')
plt.legend()
plt.show()

```

visualization(reduced_vector_pca, closest_word, "PCA", target_word)

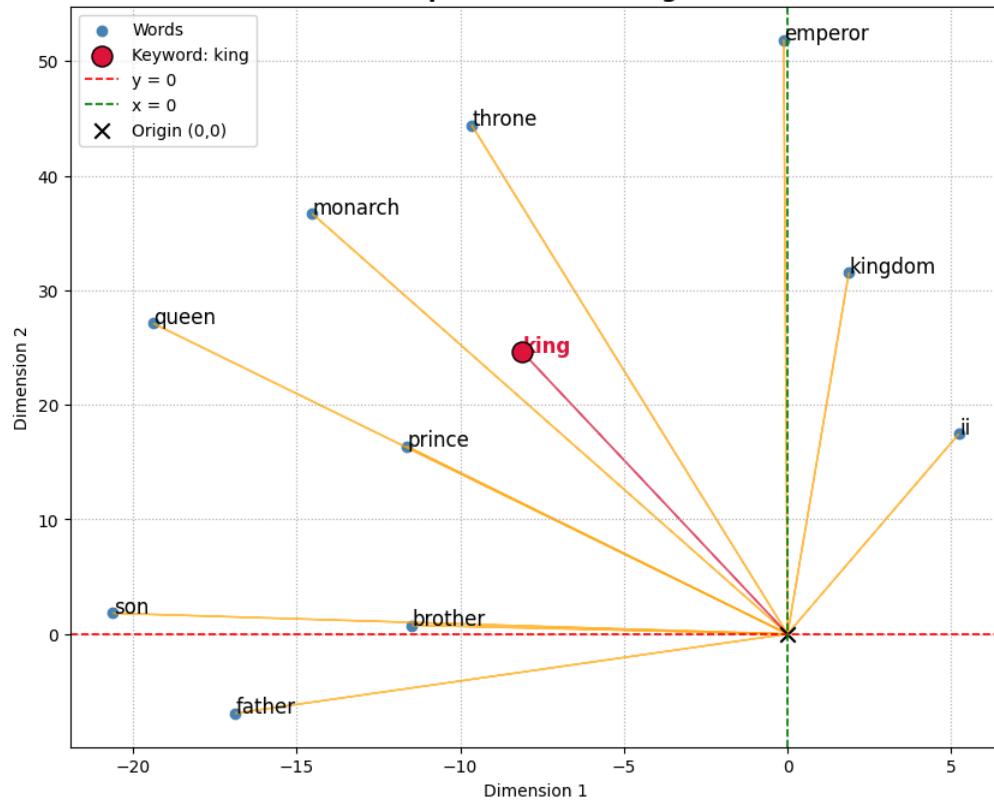


Nhận xét về biểu diễn các từ gần nghĩa thông qua giảm chiều bằng PCA:

- Các từ gần nghĩa với từ "king" khi biểu diễn thành các vector 2D thông qua việc giảm chiều bằng PCA không cho ra kết quả quá tốt về giá trị cosine similarity.
- Vector của một số từ gần nghĩa khi giảm số chiều còn 2 chiều bằng PCA không đảm bảo có giá trị cosine similarity tốt (góc với vector của từ "king" không nhỏ với một số từ) như: prince, father, brother, son, emperor
- Vector của các từ gần nghĩa khi giảm số chiều còn 2 chiều bằng PCA có giá trị cosine similarity tốt là: queen, throne, kingdom, monarch, ii

visualization(reduced_vector_tsne, closest_word, "TSNE", target_word)

Word Representation using TSNE



Nhận xét về biểu diễn các từ đồng nghĩa thông qua giảm chiều bằng phương pháp TSNE:

- Vector của các từ gần nghĩa khi giảm chiều về 2 chiều bằng TSNE có giá trị cosine similarity tốt hơn so với PCA vì các góc đã nhỏ hơn
- Các từ gần nghĩa với từ "king" đảm bảo vector embedding có giá trị góc phù hợp (không quá lớn) so với vector embedding của từ "king"