TC 5033

Word Embeddings

Team Members:

- A01200230 - Armando Bringas Corpus

Activity 3a: Exploring Word Embeddings with GloVe and Numpy

- Objective:
  - To understand the concept of word embeddings and their significance in Natural Language Processing.
  - To learn how to manipulate and visualize high-dimensional data using dimensionality reduction techniques like PCA and t-SNE.
  - To gain hands-on experience in implementing word similarity and analogies using GloVe embeddings and Numpy.

- Instructions:
  - Download GloVe pre-trained vectors from the provided link in Canvas, the official public project: Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation https://nlp.stanford.edu/data/glove.6B.zip

  - Create a dictorionay of the embeddings so that you carry out fast look ups. Save that dictionary e.g. as a serialized file for faster loading in future uses.

- PCA and t-SNE Visualization: After loading the GloVe embeddings, use Numpy and Sklearn to perform PCA and t-SNE to reduce the dimensionality of the embeddings and visualize them in a 2D or 3D space.
- Word Similarity: Implement a function that takes a word as input and returns the 'n' most similar words based on their embeddings. You should use Numpy to implement this function, using libraries that already implement this function (e.g. Gensim) will result in zero points.
- Word Analogies: Implement a function to solve analogies between words. For example, "man is to king as woman is to ____". You should use Numpy to implement this function, using libraries that already implement this function (e.g. Gensim) will result in zero points.
- Submission: This activity is to be submitted in teams of 3 or 4. Only one person should submit the final work, with the full names of all team members included in a markdown cell at the beginning of the notebook.

- Evaluation Criteria:
  - Code Quality (40%): Your code should be well-organized, clearly commented, and easy to follow. Use also markdown cells for clarity.
  - Functionality (60%): All functions should work as intended, without errors.
    - Visualization of PCA and t-SNE (10% each for a total of 20%)
    - Similarity function (20%)
    - Analogy function (20%)

Import libraries

In [1]:
```
# Import libraries
import torch
import torch.nn.functional as F
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import norm
import pickle

from typing import Dict, List, Tuple
plt.style.use('ggplot')

import seaborn as sns
```

In [2]:
```
# Check torch version
torch.__version__
```

Out[2]:
```
'2.1.0'
```

In [3]:
```
# Use GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

Load file

In [4]:
```
# PATH = '/media/pepe/DataUbuntu/Databases/glove_embeddings/glove.6B.200d.txt'
PATH = 'data/glove_embeddings/glove.6B.50d.txt'
emb_dim = 50
```

In [5]:
```
# Create dictionary with embeddings
def create_emb_dictionary(path):
    embeddings_dict = {}
    with open(path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings_dict[word] = vector
    return embeddings_dict
```

In [6]:
```
# create dictionary
embeddings_dict = create_emb_dictionary(PATH)
```

In [7]:
```
# Serialize
with open('data/glove_embeddings/embeddings_dict_50D.pkl', 'wb') as f:
    pickle.dump(embeddings_dict, f)

# Deserialize
with open('data/glove_embeddings/embeddings_dict_50D.pkl', 'rb') as f:
    embeddings_dict = pickle.load(f)
```

See some embeddings

In [8]:
```
# Show some
def show_n_first_words(path, n_words):
        with open(path, 'r') as f:
            for i, line in enumerate(f):
                print(line.split(), len(line.split()[1:]))
                if i>=n_words: break
```

In [9]:
```
show_n_first_words(PATH, 5)
```

```
['the', '0.418', '0.24968', '-0.41242', '0.1217', '0.34527', '-0.044457', '-0.4968
8', '-0.17862', '-0.00066023', '-0.6566', '0.27843', '-0.14767', '-0.55677', '0.1465
8', '-0.0095095', '0.011658', '0.10204', '-0.12792', '-0.8443', '-0.12181', '-0.0168
01', '-0.33279', '-0.1552', '-0.23131', '-0.19181', '-1.8823', '-0.76746', '0.09905
1', '-0.42125', '-0.19526', '4.0071', '-0.18594', '-0.52287', '-0.31681', '0.0005921
3', '0.0074449', '0.17778', '-0.15897', '0.012041', '-0.054223', '-0.29871', '-0.157
49', '-0.34758', '-0.045637', '-0.44251', '0.18785', '0.0027849', '-0.18411', '-0.11
514', '-0.78581'] 50
[',', '0.013441', '0.23682', '-0.16899', '0.40951', '0.63812', '0.47709', '-0.4285
2', '-0.55641', '-0.364', '-0.23938', '0.13001', '-0.063734', '-0.39575', '-0.4816
2', '0.23291', '0.090201', '-0.13324', '0.078639', '-0.41634', '-0.15428', '0.1006
8', '0.48891', '0.31226', '-0.1252', '-0.037512', '-1.5179', '0.12612', '-0.02442',
'-0.042961', '-0.28351', '3.5416', '-0.11956', '-0.014533', '-0.1499', '0.21864', '-
0.33412', '-0.13872', '0.31806', '0.70358', '0.44858', '-0.080262', '0.63003', '0.32
111', '-0.46765', '0.22786', '0.36034', '-0.37818', '-0.56657', '0.044691', '0.3039
2'] 50
['.', '0.15164', '0.30177', '-0.16763', '0.17684', '0.31719', '0.33973', '-0.43478',
'-0.31086', '-0.44999', '-0.29486', '0.16608', '0.11963', '-0.41328', '-0.42353',
'0.59868', '0.28825', '-0.11547', '-0.041848', '-0.67989', '-0.25063', '0.18472',
'0.086876', '0.46582', '0.015035', '0.043474', '-1.4671', '-0.30384', '-0.023441',
'0.30589', '-0.21785', '3.746', '0.0042284', '-0.18436', '-0.46209', '0.098329', '-
0.11907', '0.23919', '0.1161', '0.41705', '0.056763', '-6.3681e-05', '0.068987', '0.
087939', '-0.10285', '-0.13931', '0.22314', '-0.080803', '-0.35652', '0.016413', '0.
10216'] 50
['of', '0.70853', '0.57088', '-0.4716', '0.18048', '0.54449', '0.72603', '0.18157',
'-0.52393', '0.10381', '-0.17566', '0.078852', '-0.36216', '-0.11829', '-0.83336',
'0.11917', '-0.16605', '0.061555', '-0.012719', '-0.56623', '0.013616', '0.22851',
'-0.14396', '-0.067549', '-0.38157', '-0.23698', '-1.7037', '-0.86692', '-0.26704',
'-0.2589', '0.1767', '3.8676', '-0.1613', '-0.13273', '-0.68881', '0.18444', '0.0052
464', '-0.33874', '-0.078956', '0.24185', '0.36576', '-0.34727', '0.28483', '0.07569
3', '-0.062178', '-0.38988', '0.22902', '-0.21617', '-0.22562', '-0.093918', '-0.803
75'] 50
['to', '0.68047', '-0.039263', '0.30186', '-0.17792', '0.42962', '0.032246', '-0.413
76', '0.13228', '-0.29847', '-0.085253', '0.17118', '0.22419', '-0.10046', '-0.4365
3', '0.33418', '0.67846', '0.057204', '-0.34448', '-0.42785', '-0.43275', '0.55963',
'0.10032', '0.18677', '-0.26854', '0.037334', '-2.0932', '0.22171', '-0.39868', '0.2
0912', '-0.55725', '3.8826', '0.47466', '-0.95658', '-0.37788', '0.20869', '-0.3275
2', '0.12751', '0.088359', '0.16351', '-0.21634', '-0.094375', '0.018324', '0.2104
8', '-0.03088', '-0.19722', '0.082279', '-0.09434', '-0.073297', '-0.064699', '-0.26
044'] 50
['and', '0.26818', '0.14346', '-0.27877', '0.016257', '0.11384', '0.69923', '-0.5133
2', '-0.47368', '-0.33075', '-0.13834', '0.2702', '0.30938', '-0.45012', '-0.4127',
'-0.09932', '0.038085', '0.029749', '0.10076', '-0.25058', '-0.51818', '0.34558',
'0.44922', '0.48791', '-0.080866', '-0.10121', '-1.3777', '-0.10866', '-0.23201',
'0.012839', '-0.46508', '3.8463', '0.31362', '0.13643', '-0.52244', '0.3302', '0.337
07', '-0.35601', '0.32431', '0.12041', '0.3512', '-0.069043', '0.36885', '0.25168',
'-0.24517', '0.25381', '0.1367', '-0.31178', '-0.6321', '-0.25028', '-0.38097'] 50
```

Plot some embeddings

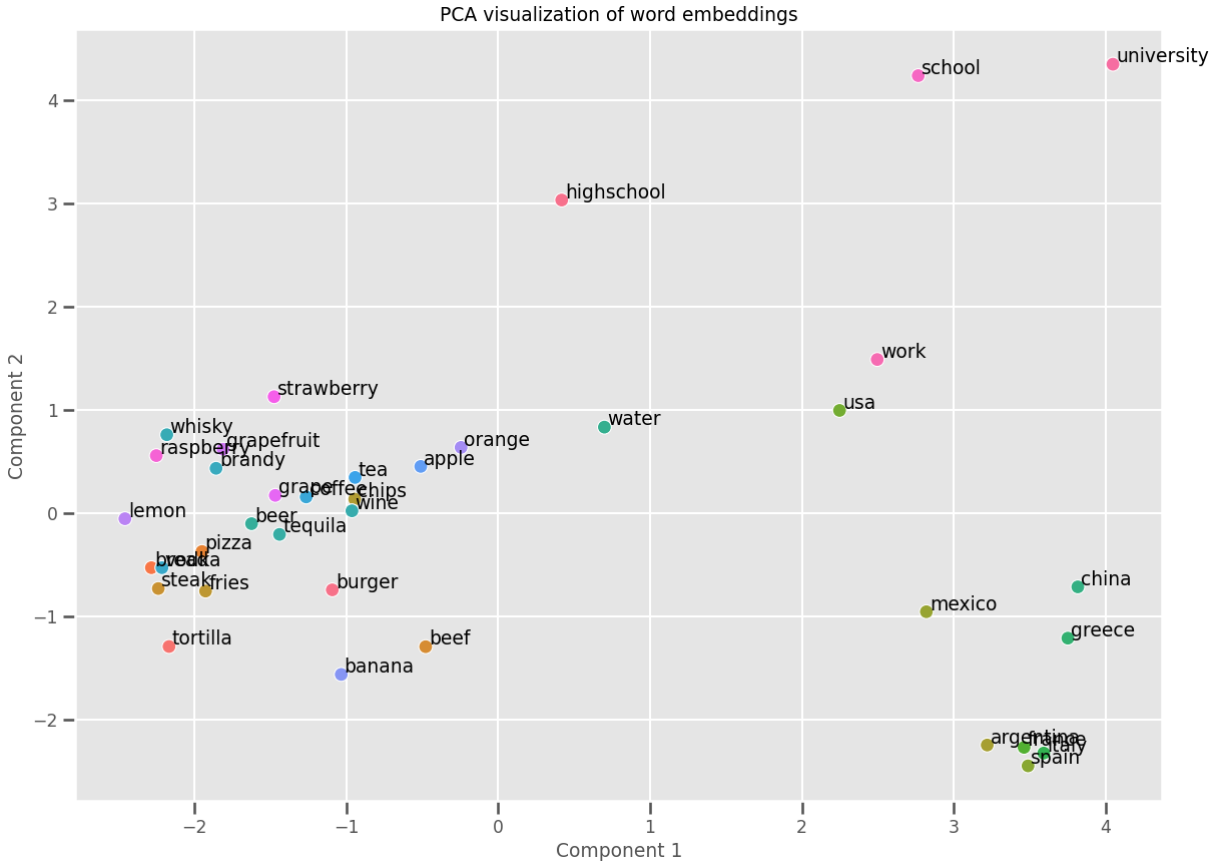
In [10]:

```python
def plot_embeddings(emb_path, words2show, emb_dim, embeddings_dict, func = PCA):

    # Extract the embeddings for the specified words
    selected_embeddings = np.array([embeddings_dict[word] for word in words2show if
```

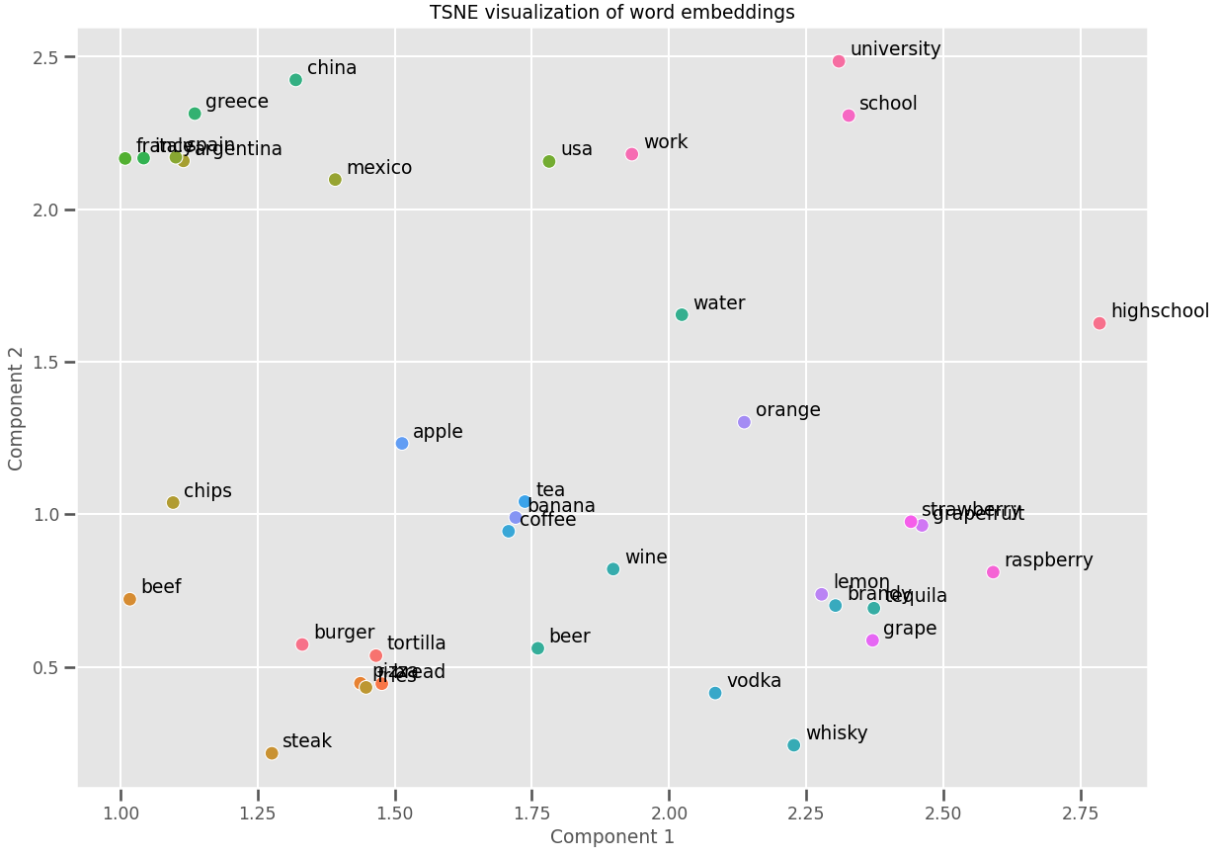
```
    # Fit the model and initialize the dimensionality reduction using PCA or TSNE
    if func == PCA:
        model = PCA(n_components=2)
        reduced_embeddings = model.fit_transform(selected_embeddings)
    elif func == TSNE:
        model = TSNE(n_components=2)
        reduced_embeddings = model.fit_transform(selected_embeddings)
    else:
        model = func(n_components=2)
        reduced_embeddings = model.fit_transform(selected_embeddings)

    # Use Seaborn for plotting
    selected_words = [word for word in words2show if word in embeddings_dict]
    sns.set_context("talk", font_scale=0.75)
    plt.figure(figsize=(14, 10))

    plot = sns.scatterplot(
        x=reduced_embeddings[:, 0], y=reduced_embeddings[:, 1],
        hue=np.arange(len(selected_embeddings)), palette="husl", legend=False, s=10
    )

    for i, word in enumerate(selected_words):
        plot.text(reduced_embeddings[i, 0] + 0.02, reduced_embeddings[i, 1] + 0.02,

    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.title(f'{func.__name__} visualization of word embeddings')
    plt.grid(True)
    plt.show()
```

In [11]:
```
words= ['burger', 'tortilla', 'bread', 'pizza', 'beef', 'steak', 'fries', 'chips',
            'argentina', 'mexico', 'spain', 'usa', 'france', 'italy', 'greece', 'ch
            'water', 'beer', 'tequila', 'wine', 'whisky', 'brandy', 'vodka', 'coffe
            'apple', 'banana', 'orange', 'lemon', 'grapefruit', 'grape', 'strawberr
            'school', 'work', 'university', 'highschool']
```

In [12]:
```
plot_embeddings(PATH, words, emb_dim, embeddings_dict, PCA)
```



In [13]:
```
# t-SNE dimensionality reduction for visualization
embeddings = plot_embeddings(PATH, words, emb_dim, embeddings_dict, TSNE)
```



Let us compute analogies

Analogy Function

In [14]:
```
def cosine_similarity(vec_a, vec_b):
    # Compute the cosine similarity between two vectors
    dot_product = np.dot(vec_a, vec_b)
    norm_a = np.linalg.norm(vec_a)
    norm_b = np.linalg.norm(vec_b)

    return dot_product / (norm_a * norm_b)
```

In [15]:
```
# analogy
def analogy(word_1, word_2, word_3, embeddings_dict):
    # Calculate the target vector for the analogy
    target_vector = embeddings_dict[word_2] - embeddings_dict[word_1] + embeddings_

    # Initialize variables to keep track of the closest word and highest similarity
    closest_word = None
    highest_similarity = -1  # Start with -1 as cosine similarity ranges from -1 to

    # Define exclusion set for input words
    exclusion_words = {word_1, word_2, word_3}

    # Iterate through the embeddings to find the closest word
    for word, vector in embeddings_dict.items():
        if word not in exclusion_words:
            similarity = cosine_similarity(target_vector, vector)
            if similarity > highest_similarity:
                highest_similarity = similarity
                closest_word = word

    return closest_word
```

In [16]:
```
sample_words = ['man', 'king', 'woman']

analogy_1 = analogy(sample_words[0], sample_words[1], sample_words[2], embeddings_d
print(f"{sample_words[0]} is to {sample_words[1]} as {sample_words[2]} is to {analo
```

```
man is to king as woman is to queen
```

For the sake of joy test on another example

In [17]:
```
sample_words = ['paris', 'france', 'rome']

analogy_2 = analogy(sample_words[0], sample_words[1], sample_words[2], embeddings_d
print(f"{sample_words[0]} is to {sample_words[1]} as {sample_words[2]} is to {analo
```

```
paris is to france as rome is to italy
```

Similarity function

In [18]:
```python
def find_most_similar(word, embeddings_dict, top_n=10):
    # Retrieve the embedding vector for the specified word
    target_vector = embeddings_dict[word]

    # Initialize a list to store tuples of (similarity score, word)
    similarities = []

    # Iterate over each word and its embedding in the dictionary
    for other_word, other_vector in embeddings_dict.items():
        # Skip the target word itself
        if other_word != word:
            # Compute the cosine similarity between the target word and other words
            similarity = cosine_similarity(target_vector, other_vector)
            # Append the similarity and other_word as a tuple to the list
            similarities.append((similarity, other_word))

    # Sort the list of tuples based on the similarity score in descending order
    # Take the top N similar words based on the similarity score
    most_similar_words = sorted(similarities, key=lambda x: x[0], reverse=True)[:to

    # Extract the words from the tuples and return them
    return most_similar_words # [word for _, word in most_similar_words]
```

In [19]:
```python
word = 'mexico'

most_similar = find_most_similar(word, embeddings_dict)

print(f"Words most similar to '{word}':\n")
for i, (score, similar_word) in enumerate(most_similar, 1):
    print(f"{i} ---> {similar_word} (score: {score:.4f})")
```

```
Words most similar to 'mexico':

1 ---> mexican (score: 0.8551)
2 ---> venezuela (score: 0.8497)
3 ---> colombia (score: 0.8490)
4 ---> peru (score: 0.8446)
5 ---> chile (score: 0.8439)
6 ---> puerto (score: 0.8363)
7 ---> rico (score: 0.8195)
8 ---> cuba (score: 0.8125)
9 ---> guatemala (score: 0.8114)
10 ---> panama (score: 0.8097)
```

For the sake of joy, test on another example

In [20]:
```python
word = 'cat'

most_similar = find_most_similar(word, embeddings_dict)

print(f"Words most similar to '{word}':\n")
for i, (score, similar_word) in enumerate(most_similar, 1):
    print(f"{i} ---> {similar_word} (score: {score:.4f})")
```

```
Words most similar to 'cat':

1 ---> dog (score: 0.9218)
2 ---> rabbit (score: 0.8488)
3 ---> monkey (score: 0.8041)
4 ---> rat (score: 0.7892)
5 ---> cats (score: 0.7865)
6 ---> snake (score: 0.7799)
7 ---> dogs (score: 0.7796)
8 ---> pet (score: 0.7792)
9 ---> mouse (score: 0.7732)
10 ---> bite (score: 0.7729)
```