65070501037
Paweekorn Soratyathorn

Install dependencies

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os

os.chdir('/content/drive/MyDrive/CPE393/week10_text-clustering')
!ls
```

```
consumer_complaint_dataset.data  lab7_text-clustering.ipynb  requirements.txt
L9-+Text+Clustering.pdf          menuitems.csv               restaurants_clustered.csv
```

```
%pip install -r requirements.txt
!pip install sentence_transformers
```

```
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sentence_transformers import SentenceTransformer

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

import nltk
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer

from collections import Counter
```

```
/usr/local/lib/python3.10/dist-packages/sentence_transformers/cross_encoder/CrossEncoder.py:13: TqdmExperimentalWarning: Using `tqdm.aut
  from tqdm.autonotebook import tqdm, trange
```

```
nltk.download('all')
```

Read data from CSV

```
DATA = 'menuitems.csv'
df = pd.read_csv(DATA)
df.describe()
```

	Restaurant Name	Original food Item
count	4524	4524
unique	341	4330
top	Vocelli Pizza	Chocolate Chip Cookie
freq	48	11

```
df.head()
```

	Restaurant Name	Original food Item
0	Coffee Time	Garden Vegetable Sandwich
1	HoSan	Roasted Seaweed Snack
2	Weight Watchers	Muffins
3	Natural Sea	Albacore Tuna, Solid White
4	World of Beer	Caesar Salad with Grilled Flat Iron Steak

Next steps: Generate code with df   View recommended plots   New interactive sheet

```
# Dataframe columns:
RESTAURANT = 'Restaurant Name'
ITEM = 'Original food Item'
EMBEDDING = 'Embedding'
CLUSTER = 'Cluster'
```

Data cleaning

Very basic, remove non-alphabetic characters and measure like fl oz and inch

```
def clean_item(item):
    # Remove these measures as they are creating spurious clusters
    rem = ['oz', 'fl', 'inch']
    for r in rem:
        p = r'\b' + re.escape(r) + r'\b'
        item = re.sub(p, '', item, flags=re.IGNORECASE)

    # Keep only alphabetical characters
    item = re.sub(r'[^a-zA-Z\s]', '', item)

    return item

df[ITEM] = df[ITEM].apply(lambda x: clean_item(x))
```

Create sentence embeddings for menu items

Using sentence-transformers library and a popular BERT based model from Hugging Face to generate the vector representations of the menu items.

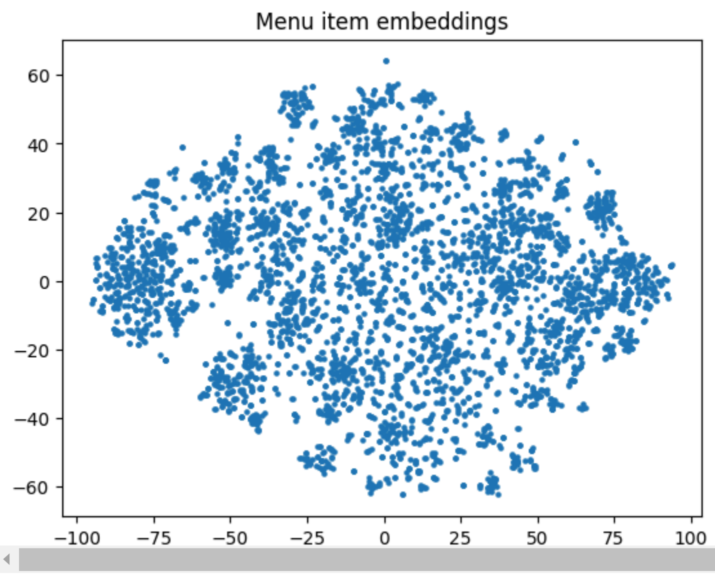
```
bert = 'bert-base-nli-stsb-mean-tokens'

sentence_transformer = SentenceTransformer(bert)
embeddings = sentence_transformer.encode(df[ITEM], show_progress_bar=True)
print(f"Embedding shape: {embeddings.shape}")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json:  100%                                  229/229  [00:00<00:00,  16.0kB/s]
config_sentence_transformers.json:  100%                                  122/122  [00:00<00:00,  8.46kB/s]
README.md:  100%                                  4.01k/4.01k  [00:00<00:00,  375kB/s]
sentence_bert_config.json:  100%                                  53.0/53.0  [00:00<00:00,  4.05kB/s]
config.json:  100%                                  630/630  [00:00<00:00,  47.3kB/s]
model.safetensors:  100%                                  438M/438M  [00:08<00:00,  54.3MB/s]
tokenizer_config.json:  100%                                  409/409  [00:00<00:00,  26.6kB/s]
vocab.txt:  100%                                  232k/232k  [00:00<00:00,  2.69MB/s]
tokenizer.json:  100%                                  466k/466k  [00:00<00:00,  23.6MB/s]
added_tokens.json:  100%                                  2.00/2.00  [00:00<00:00,  155B/s]
special_tokens_map.json:  100%                                  112/112  [00:00<00:00,  8.36kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was
  warnings.warn(
1_Pooling/config.json:  100%                                  190/190  [00:00<00:00,  11.5kB/s]
Batches:  100%                                  142/142  [00:05<00:00,  59.05it/s]
Embedding shape: (4524, 768)
```

Visualize embeddings

Visualize the embedding of the menu items using t-SNE. Couldn't see any clearly separated clusters in 2D using PCA, it was only capturing ~30% of the variance. So tried t-SNE, which shows more clearly separated clusters.

```
def visualize_embeddings(embeddings, size=10, labels=None, title=''):
    tsne = TSNE(n_components=2, learning_rate='auto')
    tsne_embeddings = tsne.fit_transform(embeddings)

    plt.scatter(tsne_embeddings[:, 0], tsne_embeddings[:, 1], s=size, c=labels)
    plt.title(title)
    plt.show()

visualize_embeddings(embeddings, size=5, title='Menu item embeddings')
```



Dimensionality reduction using PCA

Use PCA to reduce dimensionality while retaining 90% of variance in data.

```
pca = PCA(n_components=0.9, svd_solver='full')
dim_reduced_embeddings = pca.fit_transform(embeddings)
print(f"Dimension reduced embedding shape: {dim_reduced_embeddings.shape}")

df[EMBEDDING] = list(dim_reduced_embeddings)
```

```
Dimension reduced embedding shape: (4524, 119)
```

Vector Representation of restaurants

Average the vectors of the menu items of each restaurant to create a vector representation of the restaurant.

```
restaurant_df = df.drop(columns=[ITEM])
restaurant_df = restaurant_df.groupby(RESTAURANT).mean().reset_index()
```

Find the best number of clusters

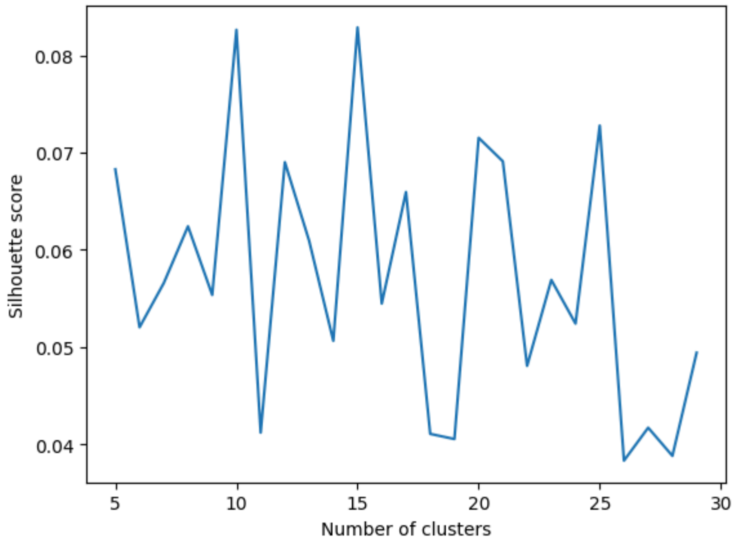
Use silhouette score (higher the better) to find the best k.

https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient

```
restaurant_embeddings = [embedding for embedding in restaurant_df[EMBEDDING].values]

candidate_k_values = list(range(5, 30))
slht_scores, kmeans_labels = [], []
for k in candidate_k_values:
    kmeans = KMeans(n_clusters=k, n_init=20)
    kmeans.fit(restaurant_embeddings)

    kmeans_labels.append(kmeans.labels_)
    slht_scores.append(silhouette_score(restaurant_embeddings, kmeans.labels_))

silhouette_score_dict = {candidate_k_values[i]: slht_scores[i] for i in range(len(slht_scores))}

print(f"Silhouette scores: {silhouette_score_dict}")

```

```
plt.plot(candidate_k_values, slht_scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.show()
```

```
Silhouette scores: {5: 0.06828255, 6: 0.052038793, 7: 0.056582134, 8: 0.0624203, 9: 0.055361707, 10: 0.08262198, 11: 0.04120551, 12: 0.0
```



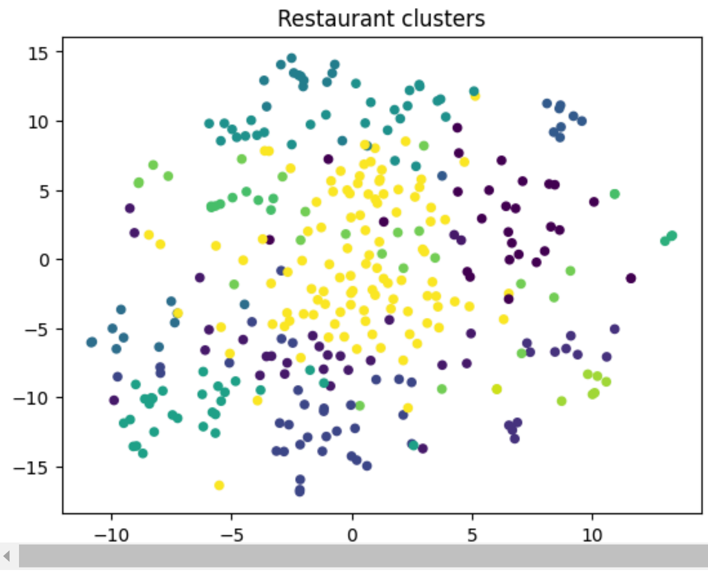
Label restaurants with clusters

```
best_k = 15
restaurant_df[CLUSTER] = kmeans_labels[best_k - candidate_k_values[0]]
```

Visualize restaurant clusters

Not very well defined clusters, other than few of them at the edge.

```
restaurant_embeddings = np.array([embedding for embedding in restaurant_df[EMBEDDING].values])

visualize_embeddings(restaurant_embeddings, size=20, labels=restaurant_df[CLUSTER].values, title='Restaurant clusters')
```



```
restaurant_df.sort_values(by=CLUSTER, inplace=True)
restaurant_df.to_csv('restaurants_clustered.csv', index=False, columns=[RESTAURANT, CLUSTER])
```

Inspect clusters

Tried to look at what the most common words in the menu items for restaurants in each cluster are.

```
def most_common_words(x):
    word_salad = ' '.join(x.values).lower().split()
    ctr = Counter(word_salad)

    return ctr.most_common(3)

menuitems_clustered_df = df.merge(restaurant_df, on=RESTAURANT, how='left')
menuitems_clustered_df.groupby(CLUSTER)[ITEM].apply(most_common_words).reset_index()
```

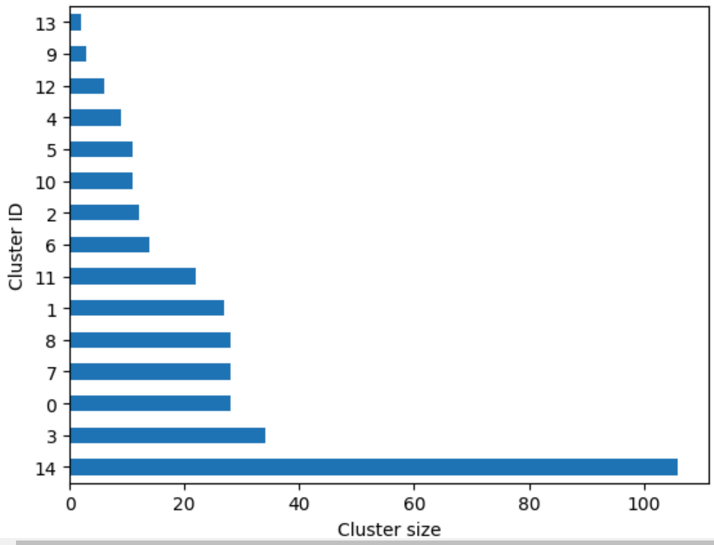
	Cluster	Original food Item
0	0	[(soup, 29), (sauce, 27), (organic, 19)]
1	1	[(cheese, 39), (chocolate, 20), (with, 19)]
2	2	[(corn, 15), (starch, 5), (bread, 4)]
3	3	[(smoothie, 75), (strawberry, 35), (milk, 29)]
4	4	[(salmon, 24), (wild, 14), (tuna, 8)]
5	5	[(yogurt, 45), (frozen, 29), (milk, 23)]
6	6	[(chicken, 117), (beef, 25), (breast, 22)]
7	7	[(pizza, 193), (crust, 72), (chicken, 63)]
8	8	[(cream, 122), (ice, 112), (chocolate, 90)]
9	9	[(salsa, 3), (classic, 1), (gourmet, 1)]
10	10	[(shrimp, 39), (breaded, 9), (butterfly, 8)]
11	11	[(cheese, 6), (tortellini, 6), (dressing, 4)]
12	12	[(rye, 5), (bread, 5), (crispbread, 3)]
13	13	[(peanuts, 3), (salted, 2), (roasted, 1)]
14	14	[(chicken, 257), (with, 224), (salad, 161)]

Cluster sizes

Few oddly small clusters (5, 7), and couple of large clusters (4, 13).

```
restaurant_df[CLUSTER].value_counts().plot(kind='barh')
plt.xlabel('Cluster size')
plt.ylabel('Cluster ID')
```

```
Text(0, 0.5, 'Cluster ID')
```



Lab part

- implement sentence embeddings with complaint data

```
complaint = pd.read_pickle('consumer_complaint_dataset.data', compression='gzip')
complaint.drop_duplicates(inplace=True)
complaint.head()
```

	topic	input
0	Debt collection	transworld systems inc. \nis trying to collect...
1	Credit reporting, credit repair services, or o...	I would like to request the suppression of the...
2	Debt collection	Over the past 2 weeks, I have been receiving e...
3	Credit reporting, credit repair services, or o...	I HAD FILED WITH CFPB ON XX/XX/XXXX19 TO HAVE ...
4	Credit reporting, credit repair services, or o...	I have several accounts that the balance is in...

```
complaint.describe()
```

	topic	input
count	464561	464561
unique	18	464287
top	Credit reporting, credit repair services, or o...	This particular account situation that is late...
freq	123323	7

```
df_prep = pd.DataFrame(columns=complaint.columns)

for topic in complaint['topic'].unique():
    temp = complaint[ complaint['topic'] == topic ].copy()

    if(len(temp) < 1000):
        df_prep = pd.concat([df_prep, temp])
    else:
        df_prep = pd.concat([df_prep, temp.sample(1000)])


df_prep.reset_index(drop=True, inplace=True)
df_prep.shape
```

```
(16305, 2)
```

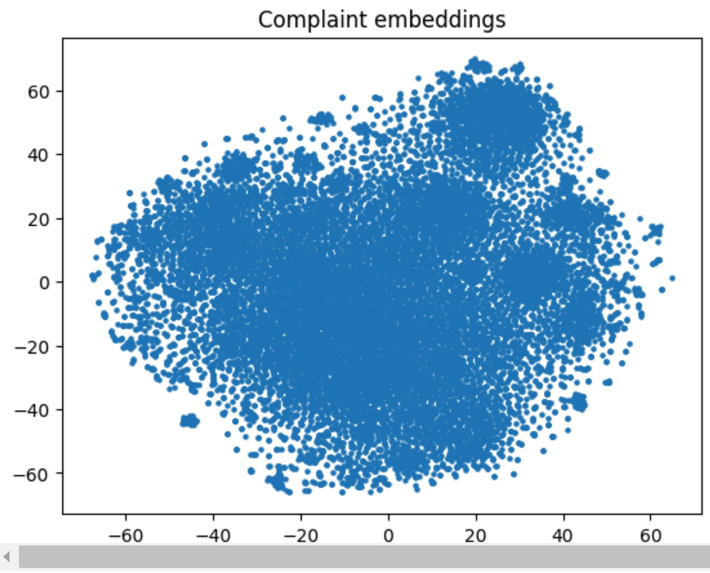
```
stop_words = stopwords.words('english')
def clean_sentence(df, label):
    for ind, sentence in enumerate(df[label]):
        # remove punctuation
        punctuation_pattern = re.compile(r'[^\w\s]|_')
        text = re.sub(punctuation_pattern, '', sentence)

        # remove censored sensitive data
        pattern = re.compile(r'x+|X+')
        text = re.sub(pattern, '', text)

        # trim sentence and change to lowercase
        text = re.sub('\n', '', text)
        words = text.split(' ')
        words = [word.lower() for word in words if word.lower() not in stop_words]

        # lemmatization
        lemmatizer = WordNetLemmatizer()
        cleaned_text = ' '.join([lemmatizer.lemmatize(word, pos='v') for word in words])
        cleaned_text = ' '.join([lemmatizer.lemmatize(word, pos='n') for word in cleaned_text.split(' ')])

        df.loc[ind, label] = cleaned_text


clean_sentence(df_prep, 'input')
```

Create sentence embeddings from complaint

```
bert = 'bert-base-nli-stsb-mean-tokens'

sentence_transformer = SentenceTransformer(bert)
embeddings = sentence_transformer.encode(df_prep['input'], show_progress_bar=True)
print(f"Embedding shape: {embeddings.shape}")
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was
  warnings.warn(
Batches: 100%                                    510/510 [01:16<00:00, 31.19it/s]
Embedding shape: (16305, 768)
```

```
visualize_embeddings(embeddings, size=5, title='Complaint embeddings')
```



```
pca = PCA(n_components=0.9, svd_solver='full')
dim_reduced_embeddings = pca.fit_transform(embeddings)
print(f"Dimension reduced embedding shape: {dim_reduced_embeddings.shape}")

df_prep['embedding'] = list(dim_reduced_embeddings)
```

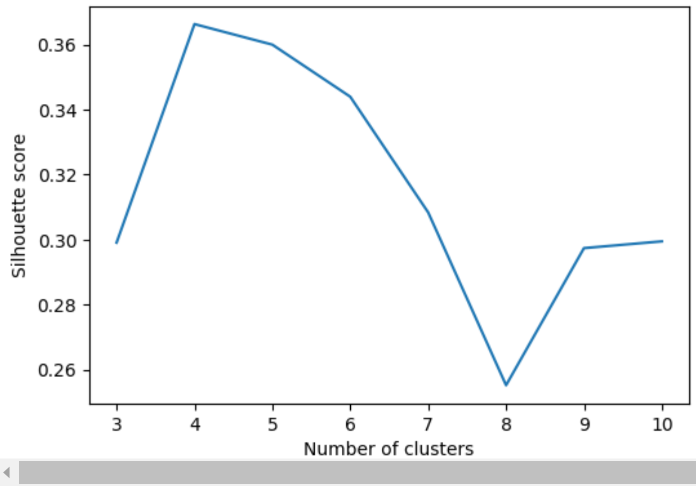
```
Dimension reduced embedding shape: (16305, 131)
```

Vector representation of Topics

```
topic_df = df_prep.drop(columns=['input'])
topic_df = topic_df.groupby('topic').mean().reset_index()
```

```
topic_embeddings = np.array([embedding for embedding in topic_df['embedding'].values])

candidate_k_values = list(range(3, 11))
slht_scores, kmeans_labels = [], []
for k in candidate_k_values:
    kmeans = KMeans(n_clusters=k, n_init=20)
    kmeans.fit(topic_embeddings)

    kmeans_labels.append(kmeans.labels_)
    slht_scores.append(silhouette_score(topic_embeddings, kmeans.labels_))

silhouette_score_dict = {candidate_k_values[i]: slht_scores[i] for i in range(len(slht_scores))}

print(f"Silhouette scores: {silhouette_score_dict}")

plt.figure(figsize=(6, 4))
plt.plot(candidate_k_values, slht_scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.show()
```

```
Silhouette scores: {3: 0.2990773, 4: 0.3663051, 5: 0.35997707, 6: 0.34395307, 7: 0.30837342, 8: 0.25520167, 9: 0.29738295, 10: 0.2994643
```



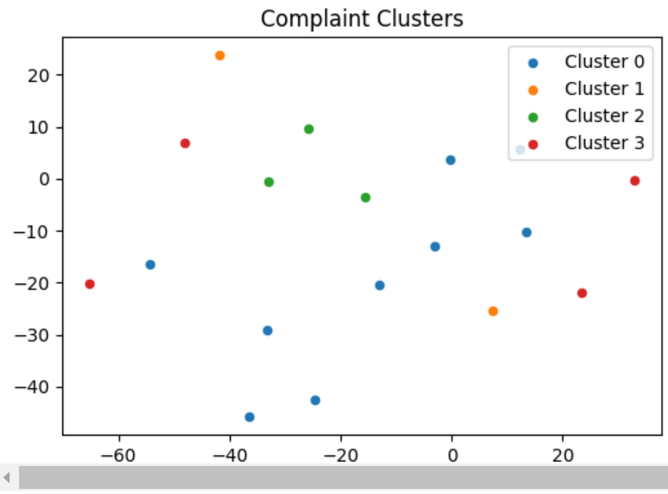
```
best_k = 4
topic_df['Cluster'] = kmeans_labels[best_k - candidate_k_values[0]]
topic_df.sort_values(by='Cluster', inplace=True)
```

Cluster Evaluation

```
complaint_clustered_df = df_prep.merge(topic_df, on='topic', how='left')
complaint_clustered_df.groupby('Cluster')['input'].apply(most_common_words).reset_index()
```

	Cluster	input
0	0	[(account, 18544), (bank, 11199), (card, 10815)]
1	1	[(payment, 4715), (call, 3092), (loan, 3090)]
2	2	[(report, 6591), (credit, 6233), (account, 4342)]
3	3	[(loan, 10844), (payment, 8242), (pay, 5841)]

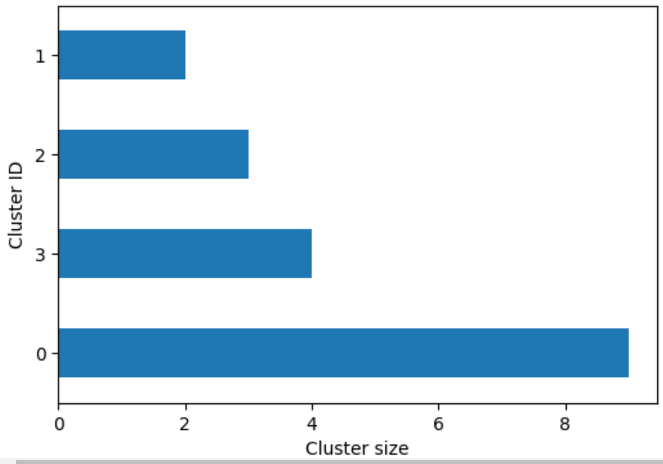
```
def visualize_embeddings(embeddings, size=10, labels=None, title=''):
    tsne = TSNE(n_components=2, perplexity=10, learning_rate='auto')
    tsne_embeddings = tsne.fit_transform(embeddings)

    plt.figure(figsize=(6, 4))
    unique_labels = np.unique(labels)
    for i, label in enumerate(unique_labels):
        indices = (labels == label)
        plt.scatter(tsne_embeddings[indices, 0], tsne_embeddings[indices, 1],
                    s=size, label=f'Cluster {label}')

    plt.title(title)
    plt.legend(loc='upper right')
    plt.show()


visualize_embeddings(topic_embeddings, size=20, labels=topic_df['Cluster'].values, title='Complaint Clusters')
```



Didn't well separate much

```
plt.figure(figsize=(6, 4))
topic_df['Cluster'].value_counts().plot(kind='barh')
plt.xlabel('Cluster size')
plt.ylabel('Cluster ID')
```

```
Text(0, 0.5, 'Cluster ID')
```



Interpret Result

- **TSNE:** cluster 0 and 2 is well-defined at the middle. On the other hand, cluster 1 and 3 is not quite separated. They are mixed at the edges.
- **Cluster size:** large cluster of group 0 while other groups didn't difference much.