hw6

November 5, 2024

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw6.ipynb")
```

1 CPSC 330 - Applied Machine Learning

1.1 Homework 6: Clustering

1.1.1 Associated lectures: Lectures 15 and 16

Due date: Check the Calendar

1.2 Imports

[2]:
```
import os
from hashlib import sha1

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

%matplotlib inline
pd.set_option("display.max_colwidth", 0)
```

1.3 Submission instructions

rubric={points:6}

Please be aware that this homework assignment requires installation of several packages in your course environment. It's possible that you'll encounter installation challenges, which might be frustrating. However, remember that solving these issues is not wasting time but it is an essential skill for anyone aspiring to work in data science or machine learning.

Follow the homework submission instructions.

You may work in a group on this homework and submit your assignment as a group. Below are some instructions on working as a group.
- The maximum group size is 4. - Use group work as an opportunity to collaborate and learn new things from each other. - Be respectful to each other and make sure you understand all the concepts

in the assignment well. - It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline. - You can find the instructions on how to do group submission on Gradescope here.

When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.
4. Make sure that the plots and output are rendered properly in your submitted file.
5. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb. If the pdf or html also fail to render on Gradescope, please create two files for your homework: hw6a.ipynb with Exercise 1 and hw6b.ipynb with Exercises 2 and 3 and submit these two files in your submission.

Points: 6

1.4 Exercise 1: Document clustering warm-up

In this homework, we will explore a popular application of clustering called **document clustering**. A large amount of unlabeled text data is available out there (e.g., news, recipes, online Q&A, tweets), and clustering is a commonly used technique to organize this data in a meaningful way.

As a warm up, in this exercise you will cluster sentences from a toy corpus. Later in the homework you will work with a real corpus.

The code below extracts introductory sentences of Wikipedia articles on a set of queries. To run the code successfully, you will need the `wikipedia` package installed in the course environment.

```
conda activate cpsc330
conda install -c conda-forge wikipedia
```

Your tasks:

Run the code below which - extracts content of Wikipedia articles on a set of queries - tokenizes the text (i.e., separates sentences) and - stores the 2nd sentence in each article as a document representing that article

> Feel free to experiment with Wikipedia queries of your choice. But stick to the provided list for the final submission so that it's easier for the TAs to grade your submission.
>
> For tokenization we are using the `nltk` package. If you do not have this package in the course environment, you will have to install it.

```
conda activate cpsc330
conda install -c anaconda nltk
```

Even if you have the package installed via the course `conda` environment, you might have to download `nltk` pre-trained models, which can be done with the code below.

[3]:
```
import nltk

nltk.download("punkt")
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\docto\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\docto\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

[3]:
```
True
```

[4]:
```
import wikipedia
from nltk.tokenize import sent_tokenize, word_tokenize

queries = [
    "Artificial Intelligence",
    "Deep learning",
    "Unsupervised learning",
    "Quantum Computing",
    "Environmental protection",
    "Climate Change",
    "Renewable Energy",
    "Biodiversity",
    "French Cuisine",
    "Bread food",
    "Dumpling food"
]

wiki_dict = {"wiki query": [], "text": [], "n_words": []}
for i in range(len(queries)):
    text = sent_tokenize(wikipedia.page(queries[i]).content)[1]
    wiki_dict["text"].append(text)
    wiki_dict["n_words"].append(len(word_tokenize(text)))
    wiki_dict["wiki query"].append(queries[i])

wiki_df = pd.DataFrame(wiki_dict)
wiki_df
```

[4]:

	wiki query	\
0	Artificial Intelligence	
1	Deep learning	
2	Unsupervised learning	
3	Quantum Computing	
4	Environmental protection	
5	Climate Change	

```
6   Renewable Energy
7   Biodiversity
8   French Cuisine
9   Bread food
10  Dumpling food

          text  \
0   It is a field of research in computer science that develops and studies
methods and software that enable machines to perceive their environment and use
learning and intelligence to take actions that maximize their chances of
achieving defined goals.
1   The field takes inspiration from biological neuroscience and is centered
around stacking artificial neurons into layers and "training" them to process
data.
2   The training data is processed, building a function that maps new data to
expected output values.
3   On small scales, physical matter exhibits properties of both particles and
waves, and quantum computing leverages this behavior using specialized hardware.
4   Its objectives are to conserve natural resources and the existing natural
environment and, where it is possible, to repair damage and reverse trends.
5   Climate change in a broader sense also includes previous long-term changes
to Earth's climate.
6   The most widely used renewable energy types are solar energy, wind power,
and hydropower.
7   It can be measured on various levels.
8   In the 14th century, Guillaume Tirel, a court chef known as "Taillevent",
wrote Le Viandier, one of the earliest recipe collections of medieval France.
9   Throughout recorded history and around the world, it has been an important
part of many cultures' diet.
10  The dough can be based on bread, wheat or other flours, or potatoes, and it
may be filled with meat, fish, tofu, cheese, vegetables, or a combination.

    n_words
0   40
1   25
2   18
3   24
4   26
5   16
6   17
7   8
8   31
9   20
10  36
```

Our toy corpus has six toy documents (`text` column in the dataframe) extracted from Wikipedia queries.

1.4.1 1.1 How many clusters?

rubric={points}

Your tasks:

1. If you are asked to cluster the documents from this toy corpus manually, how many clusters would you identify and how would you label each cluster?

Solution_1.1

Points: 1

I would expect to see the following 3 clusters: 1. "Computer Science" cluster: contains the documents for "Artificial Intelligence", "Deep learning", "Unsupervised learning", and "Quantum Computing" 2. "Environmental Science" cluster: contains the documents for "Environmental protection", "Climate Change", "Renewable Energy", and "Biodiversity" 3. "Food" cluster: contains the documents for "French Cuisine", "Bread food", and "Dumpling food"

1.4.2 1.2 `KMeans` with bag-of-words representation

rubric={points}

In the lecture, we saw that data representation plays a crucial role in clustering. Changing flattened representation of images to feature vectors extracted from pre-trained models greatly improved the quality of clustering.

What kind of representation is suitable for text data? We have used bag-of-words representation to numerically encode text data before, where each document is represented with a vector of word frequencies.

Let's try clustering documents with this simplistic representation.

Your tasks:

1. Create bag-of-words representation using `CountVectorizer` with default arguments for the `text` column in `wiki_df` above.
2. Cluster the encoded documents with `KMeans` clustering. Use `random_state=42` (for reproducibility) and set `n_clusters` to the number you identified in the previous exercise.
3. Store the clustering labels in `kmeans_bow_labels` variable below.

Solution_1.2

Points: 4

[5]:
```
# count vectorizer
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
bag_of_words = pd.DataFrame(vectorizer.fit_transform(wiki_df["text"]).
 ↪toarray(), columns=vectorizer.get_feature_names_out())
bag_of_words
```

[5]:

	14th	achieving	actions	also	an	and	are	around	artificial	as	…	\
0	0	1	1	0	0	4	0	0	0	0	…	
1	0	0	0	0	0	2	0	1	1	0	…	
2	0	0	0	0	0	0	0	0	0	0	…	
3	0	0	0	0	0	2	0	0	0	0	…	
4	0	0	0	0	0	3	1	0	0	0	…	
5	0	0	0	1	0	0	0	0	0	0	…	
6	0	0	0	0	0	1	1	0	0	0	…	
7	0	0	0	0	0	0	0	0	0	0	…	
8	1	0	0	0	0	0	0	0	0	1	…	
9	0	0	0	0	1	1	0	1	0	0	…	
10	0	0	0	0	0	1	0	0	0	0	…	

	vegetables	viandier	waves	wheat	where	widely	wind	with	world	\
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	
4	0	0	0	0	1	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	1	1	0	0	
7	0	0	0	0	0	0	0	0	0	
8	0	1	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	1	
10	1	0	0	1	0	0	0	1	0	

	wrote
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	0
10	0

[11 rows x 159 columns]

[6]:
```
from sklearn.cluster import KMeans

kmeans_bow_labels = KMeans(n_clusters=3, random_state=42).
 ↪fit_predict(bag_of_words)
kmeans_bow_labels
```

[6]:
```
array([2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1])
```

[7]:
```
wiki_df["bow_kmeans"] = kmeans_bow_labels
wiki_df
```

[7]:

	wiki query	text	n_words	bow_kmeans
0	Artificial Intelligence	It is a field of research in computer science that develops and studies methods and software that enable machines to perceive their environment and use learning and intelligence to take actions that maximize their chances of achieving defined goals.	40	2
1	Deep learning	The field takes inspiration from biological neuroscience and is centered around stacking artificial neurons into layers and "training" them to process data.	25	1
2	Unsupervised learning	The training data is processed, building a function that maps new data to expected output values.		
3	Quantum Computing	On small scales, physical matter exhibits properties of both particles and waves, and quantum computing leverages this behavior using specialized hardware.		
4	Environmental protection	Its objectives are to conserve natural resources and the existing natural environment and, where it is possible, to repair damage and reverse trends.		
5	Climate Change	Climate change in a broader sense also includes previous long-term changes to Earth's climate.		
6	Renewable Energy	The most widely used renewable energy types are solar energy, wind power, and hydropower.		
7	Biodiversity	It can be measured on various levels.		
8	French Cuisine	In the 14th century, Guillaume Tirel, a court chef known as "Taillevent", wrote Le Viandier, one of the earliest recipe collections of medieval France.		
9	Bread food	Throughout recorded history and around the world, it has been an important part of many cultures' diet.		
10	Dumpling food	The dough can be based on bread, wheat or other flours, or potatoes, and it may be filled with meat, fish, tofu, cheese, vegetables, or a combination.		

2	18	1
3	24	1
4	26	0
5	16	1
6	17	1
7	8	1
8	31	1
9	20	1
10	36	1

1.4.3 1.3 Sentence embedding representation

rubric={points}

Bag-of-words representation is limited in that it does not take into account word ordering and context. There are other richer and more expressive representations of text which can be extracted using transfer learning. In this lab, we will use one such representation called sentence embedding representation, which uses deep learning models to generate dense, fixed-length vector representations for sentences. We will extract such representations using sentence transformer package. Sentence embedding takes into account context of words and semantic meaning of sentences and it is likely to work better when we are interested in clustering sentences based on their semantic similarity.

```
conda activate cpsc330
conda install pytorch::pytorch torchvision torchaudio -c pytorch
conda install -c conda-forge sentence-transformers
```

Your tasks:

1. Run the code below to create sentence embedding representation of documents in our toy corpus.
2. Cluster documents in our toy corpus encoded with this representation (`emb_sents`) and `KMeans` with following arguments:
   - `random_state=42` (for reproducibility)
   - `n_clusters`=the number of clusters you identified in 1.1
3. Store the clustering labels in `kmeans_emb_labels` variable below.

[8]:
```
from sentence_transformers import SentenceTransformer

embedder = SentenceTransformer("paraphrase-distilroberta-base-v1")

# If this cell gives an error, try updating transformers with
# pip install transformers -U
```

```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-
packages\sentence_transformers\cross_encoder\CrossEncoder.py:13:
TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use
`tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  from tqdm.autonotebook import tqdm, trange
```

[9]:
```
emb_sents = embedder.encode(wiki_df["text"])
emb_sent_df = pd.DataFrame(emb_sents, index=wiki_df.index)
emb_sent_df
```

[9]:

	0	1	2	3	4	5	6	\
0	0.102875	0.201959	0.044091	0.281749	0.321483	-0.281129	0.042515	
1	0.000322	0.428834	0.152298	-0.161278	0.224354	-0.363829	0.110951	
2	0.236465	-0.282463	-0.258300	0.300584	0.234605	0.061746	-0.072744	
3	0.276844	0.657946	0.106466	0.290566	0.803929	0.023764	0.136675	
4	0.200328	0.157551	0.093484	0.120533	-0.439307	0.148569	-0.003543	
5	0.189106	0.406864	0.172560	0.273776	0.058933	0.224642	-0.056590	
6	-0.066224	0.465512	-0.135840	-0.229255	-0.144746	0.013772	-0.122810	
7	-0.139882	0.207129	-0.127446	0.214821	-0.099096	0.063319	-0.347634	
8	-0.112771	-0.259073	0.172584	-0.149188	-0.074585	0.222288	-0.213039	
9	-0.022418	0.217159	0.022694	0.003616	0.240856	0.358046	-0.053310	
10	-0.123723	0.190113	-0.064433	0.206614	0.198812	0.156827	0.040764	

	7	8	9	…	758	759	760	761	\
0	0.083172	0.155722	-0.114267	…	0.310223	-0.141897	-0.153256	-0.058627	
1	0.042096	0.188454	0.188299	…	-0.092609	-0.117086	0.133018	0.207266	
2	0.045550	0.271853	0.054201	…	0.531341	-0.095102	0.316027	0.170600	
3	-0.030063	0.115825	0.244903	…	0.069148	-0.183727	0.217584	0.140797	
4	-0.211189	-0.006088	-0.102894	…	0.187549	-0.430333	-0.350163	0.251643	
5	-0.242728	0.056221	0.127506	…	0.099107	-0.350095	0.070398	0.469786	
6	-0.172322	-0.278583	0.002542	…	-0.090149	-0.305184	-0.048772	-0.008466	
7	-0.203128	0.325749	0.624156	…	0.311010	-0.036291	0.004326	0.078006	
8	0.512004	0.216943	-0.200929	…	0.083372	0.196123	0.030323	-0.308583	
9	-0.328075	0.190012	0.244470	…	0.265413	-0.415594	0.003036	0.052484	
10	0.202014	-0.195558	0.422269	…	-0.037450	-0.172986	0.167084	-0.012104	

	762	763	764	765	766	767
0	0.005015	-0.005808	0.630392	-0.023694	-0.071900	-0.115921
1	-0.395182	0.085347	0.677385	-0.405457	0.092258	-0.100411
2	-0.213746	0.110625	0.463024	-0.658667	0.246860	0.171115
3	-0.216237	0.128014	0.181907	-0.054305	0.131837	-0.016249
4	0.115400	-0.053312	0.088485	0.307340	0.355386	0.024684
5	0.003972	-0.130359	-0.105738	-0.091219	0.210350	0.072002
6	0.145859	-0.153741	0.012771	0.507823	0.239722	0.015132
7	0.321024	-0.180906	0.553141	-0.010740	0.382759	0.032368
8	0.061457	-0.008511	0.350080	0.456677	-0.352978	-0.053868
9	0.345947	0.110091	0.405441	0.197791	-0.058254	0.212376
10	0.387072	0.218355	0.290221	0.407192	0.209376	-0.189508

[11 rows x 768 columns]

Solution_1.3

Points: 3

[10]:
```
kmeans_emb_labels = KMeans(n_clusters=3, random_state=42).
↪fit_predict(emb_sent_df)
```

[11]:
```
wiki_df["emb_kmeans"] = kmeans_emb_labels
wiki_df
```

[11]:

	wiki query	text	n_words	bow_kmeans	emb_kmeans
0	Artificial Intelligence	It is a field of research in computer science that develops and studies methods and software that enable machines to perceive their environment and use learning and intelligence to take actions that maximize their chances of achieving defined goals.	40	2	2
1	Deep learning	The field takes inspiration from biological neuroscience and is centered around stacking artificial neurons into layers and "training" them to process data.			
2	Unsupervised learning	The training data is processed, building a function that maps new data to expected output values.			
3	Quantum Computing	On small scales, physical matter exhibits properties of both particles and waves, and quantum computing leverages this behavior using specialized hardware.			
4	Environmental protection	Its objectives are to conserve natural resources and the existing natural environment and, where it is possible, to repair damage and reverse trends.			
5	Climate Change	Climate change in a broader sense also includes previous long-term changes to Earth's climate.			
6	Renewable Energy	The most widely used renewable energy types are solar energy, wind power, and hydropower.			
7	Biodiversity	It can be measured on various levels.			
8	French Cuisine	In the 14th century, Guillaume Tirel, a court chef known as "Taillevent", wrote Le Viandier, one of the earliest recipe collections of medieval France.			
9	Bread food	Throughout recorded history and around the world, it has been an important part of many cultures' diet.			
10	Dumpling food	The dough can be based on bread, wheat or other flours, or potatoes, and it may be filled with meat, fish, tofu, cheese, vegetables, or a combination.			

1	25	1	2
2	18	1	2
3	24	1	2
4	26	0	0
5	16	1	0
6	17	1	0
7	8	1	1
8	31	1	1
9	20	1	1
10	36	1	1

1.4.4 1.4 DBSCAN with cosine distance

rubric={points}

Now try `DBSCAN` on our toy dataset. K-Means is kind of bound to the Euclidean distance because it is based on the notion of means. With `DBSCAN` we can try different distance metrics. In the context of text data, cosine similarities or cosine distances tend to work well. Given vectors u and v, the **cosine distance** between the vectors is defined as:

$$distance_{cosine}(u, v) = 1 - (\frac{u \cdot v}{\|u\|_2 \|v\|_2})$$

Your tasks

1. Cluster documents in our toy corpus encoded with sentence embedding representation (`emb_sents`) and DBSCAN with `metric='cosine'`. You will have to set appropriate values for the hyperparamters `eps` and `min_samples` to get meaningful clusters, as default values of these hyperparameters are unlikely to work well on this toy dataset.
2. Store the clustering labels in the `dbscan_emb_labels` variable below.

Solution_1.4

Points: 4

[12]:
```
# DBSCAN
from sklearn.cluster import DBSCAN
```

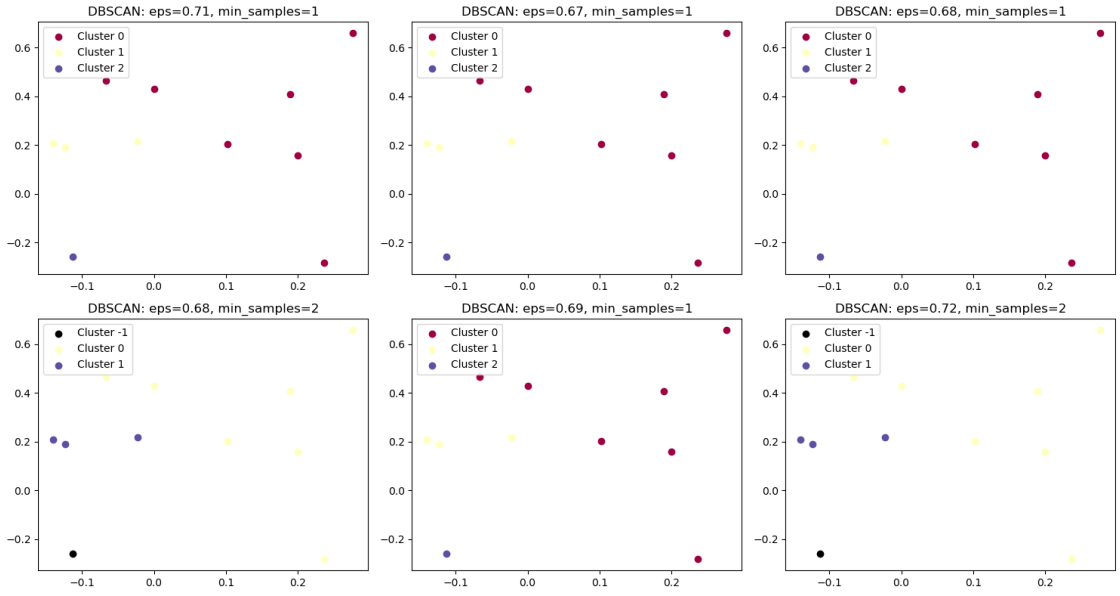
[13]:
```
import numpy as np
from sklearn.cluster import DBSCAN
from matplotlib import pyplot as plt
import random

from random import uniform, randint

random.seed(42)
np.random.seed(42)

n_samples = 6
```

```
fig, axes = plt.subplots(2, 3, figsize=(15, 8))
plot_count = 0

for ax in axes.flatten():
    while plot_count < n_samples:
        eps = uniform(0.2, 1)
        min_samples = randint(1, 3)

        dbscan_emb_labels = DBSCAN(eps=eps, min_samples=min_samples,␣
 ↪metric='cosine').fit_predict(emb_sent_df)
        unique_labels = np.unique(dbscan_emb_labels)

        print(f"eps={eps}, min_samples={min_samples}: {unique_labels}")

        if len(unique_labels) == 3:
            print(f"eps={eps}, min_samples={min_samples}: {unique_labels}")

            colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))

            for k, col in zip(unique_labels, colors):
                if k == -1:
                    col = [0, 0, 0, 1]

                class_member_mask = (dbscan_emb_labels == k)
                xy = emb_sent_df[class_member_mask]
                ax.scatter(xy.iloc[:, 0], xy.iloc[:, 1], c=[col],␣
 ↪label=f'Cluster {k}')

            ax.set_title(f'DBSCAN: eps={eps:.2f}, min_samples={min_samples}')
            ax.legend()
            plot_count += 1
            break

plt.tight_layout()
plt.show()
```

```
eps=0.711541438766307, min_samples=1: [0 1 2]
eps=0.711541438766307, min_samples=1: [0 1 2]
eps=0.7932403998078663, min_samples=1: [0]
eps=0.3785685905190582, min_samples=3: [-1]
eps=0.28199614093720604, min_samples=3: [-1]
eps=0.9137436541638764, min_samples=1: [0]
eps=0.6723940099592318, min_samples=1: [0 1 2]
eps=0.6723940099592318, min_samples=1: [0 1 2]
eps=0.22383777555045628, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.3861287147125917, min_samples=3: [-1]
eps=0.2212287757470909, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
```

```
eps=0.7728156903379229, min_samples=3: [-1  0]
eps=0.6359531844825734, min_samples=1: [0 1 2 3 4 5]
eps=0.5593672370270829, min_samples=2: [-1  0]
eps=0.8475443653422614, min_samples=1: [0]
eps=0.8070458937038139, min_samples=1: [0]
eps=0.7585115159905815, min_samples=2: [-1  0]
eps=0.4222970733373135, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.9657704577654249, min_samples=2: [0]
eps=0.28176822121587897, min_samples=2: [-1]
eps=0.27737310146677124, min_samples=2: [-1]
eps=0.6829808250935129, min_samples=1: [0 1 2]
eps=0.6829808250935129, min_samples=1: [0 1 2]
eps=0.7837854293550544, min_samples=3: [0]
eps=0.2998609302825675, min_samples=2: [-1]
eps=0.26304015846276657, min_samples=2: [-1]
eps=0.863523731402396, min_samples=3: [0]
eps=0.9083613983494454, min_samples=2: [0]
eps=0.6618817162054096, min_samples=3: [-1  0]
eps=0.25564411905899276, min_samples=3: [-1]
eps=0.3823186205212375, min_samples=2: [-1]
eps=0.9881772165286062, min_samples=1: [0]
eps=0.8931869334042157, min_samples=2: [0]
eps=0.4223788824880737, min_samples=3: [-1]
eps=0.8672883413126002, min_samples=1: [0]
eps=0.4961447736935061, min_samples=1: [0 1 2 3 4 5 6 7 8 9]
eps=0.7361401395021547, min_samples=3: [-1  0]
eps=0.9493236701699952, min_samples=3: [0]
eps=0.25712200766283866, min_samples=3: [-1]
eps=0.3369109185584776, min_samples=3: [-1]
eps=0.395848740121141, min_samples=2: [-1]
eps=0.5035643534061183, min_samples=3: [-1]
eps=0.7505295667466505, min_samples=1: [0 1]
eps=0.7476914007918998, min_samples=1: [0 1]
eps=0.3832384575712835, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.8440366406169715, min_samples=2: [0]
eps=0.4141927007805622, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.9305088993484361, min_samples=3: [0]
eps=0.901094101178135, min_samples=2: [0]
eps=0.3701012352433125, min_samples=2: [-1]
eps=0.5165055208485314, min_samples=3: [-1]
eps=0.567081482069919, min_samples=2: [-1  0]
eps=0.31170425562040505, min_samples=3: [-1]
eps=0.6490945073305207, min_samples=2: [-1  0  1  2]
eps=0.7976110493218063, min_samples=2: [0]
eps=0.9182583068819816, min_samples=2: [0]
eps=0.4895971549708888, min_samples=1: [0 1 2 3 4 5 6 7 8 9]
eps=0.6076210349411717, min_samples=1: [0 1 2 3 4 5 6 7 8 9]
eps=0.8046257260269902, min_samples=1: [0]
```

```
eps=0.32227305547860885, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.8336634914903713, min_samples=2: [0]
eps=0.6771272245855791, min_samples=2: [-1  0  1]
eps=0.6771272245855791, min_samples=2: [-1  0  1]
eps=0.5052954292052294, min_samples=2: [-1  0]
eps=0.6232914760793096, min_samples=3: [-1  0]
eps=0.8886237617875985, min_samples=1: [0]
eps=0.7442267282052148, min_samples=1: [0 1]
eps=0.7453682952212599, min_samples=3: [-1  0]
eps=0.8007022396732879, min_samples=3: [0]
eps=0.4721403784248362, min_samples=2: [-1]
eps=0.547812200535284, min_samples=2: [-1  0]
eps=0.20259625288138344, min_samples=3: [-1]
eps=0.9006823523025553, min_samples=2: [0]
eps=0.9775104790960916, min_samples=1: [0]
eps=0.6061453396834602, min_samples=1: [0 1 2 3 4 5 6 7 8 9]
eps=0.8964148558694136, min_samples=2: [0]
eps=0.8733354785379737, min_samples=3: [0]
eps=0.6871761691505378, min_samples=1: [0 1 2]
eps=0.6871761691505378, min_samples=1: [0 1 2]
eps=0.4991310820729184, min_samples=1: [0 1 2 3 4 5 6 7 8 9]
eps=0.6315032240957006, min_samples=3: [-1  0]
eps=0.9347947955165619, min_samples=3: [0]
eps=0.4593248456037385, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.2894941802567161, min_samples=2: [-1]
eps=0.9029775022585473, min_samples=2: [0]
eps=0.39156184827044704, min_samples=1: [ 0  1  2  3  4  5  6  7  8  9 10]
eps=0.9024076793632325, min_samples=1: [0]
eps=0.26852276165430305, min_samples=2: [-1]
eps=0.8528186027360709, min_samples=3: [0]
eps=0.8126675434455903, min_samples=1: [0]
eps=0.727800658197894, min_samples=3: [-1  0]
eps=0.33210156717930217, min_samples=3: [-1]
eps=0.8979464328682061, min_samples=2: [0]
eps=0.9714903248059643, min_samples=3: [0]
eps=0.8042121573086445, min_samples=3: [0]
eps=0.3609208507117567, min_samples=2: [-1]
eps=0.5191938399979433, min_samples=3: [-1]
eps=0.7199024461115628, min_samples=2: [-1  0  1]
eps=0.7199024461115628, min_samples=2: [-1  0  1]
```



[14]:
```
dbscan_emb_labels = DBSCAN(eps=0.70, min_samples=1, metric='cosine').
↪fit_predict(emb_sent_df)
```

[15]:
```
wiki_df["emb_dbscan"] = dbscan_emb_labels
wiki_df
```

[15]:

	wiki query	text
0	Artificial Intelligence	It is a field of research in computer science that develops and studies methods and software that enable machines to perceive their environment and use learning and intelligence to take actions that maximize their chances of achieving defined goals.
1	Deep learning	The field takes inspiration from biological neuroscience and is centered around stacking artificial neurons into layers and "training" them to process data.
2	Unsupervised learning	
3	Quantum Computing	
4	Environmental protection	
5	Climate Change	
6	Renewable Energy	
7	Biodiversity	
8	French Cuisine	
9	Bread food	
10	Dumpling food	

```
2   The training data is processed, building a function that maps new data to
expected output values.
3   On small scales, physical matter exhibits properties of both particles and
waves, and quantum computing leverages this behavior using specialized hardware.
4   Its objectives are to conserve natural resources and the existing natural
environment and, where it is possible, to repair damage and reverse trends.
5   Climate change in a broader sense also includes previous long-term changes
to Earth's climate.
6   The most widely used renewable energy types are solar energy, wind power,
and hydropower.
7   It can be measured on various levels.
8   In the 14th century, Guillaume Tirel, a court chef known as "Taillevent",
wrote Le Viandier, one of the earliest recipe collections of medieval France.
9   Throughout recorded history and around the world, it has been an important
part of many cultures' diet.
10  The dough can be based on bread, wheat or other flours, or potatoes, and it
may be filled with meat, fish, tofu, cheese, vegetables, or a combination.
```

	n_words	bow_kmeans	emb_kmeans	emb_dbscan
0	40	2	2	0
1	25	1	2	0
2	18	1	2	0
3	24	1	2	0
4	26	0	0	0
5	16	1	0	0
6	17	1	0	0
7	8	1	1	1
8	31	1	1	2
9	20	1	1	1
10	36	1	1	1

1.4.5 1.5 Hierarchical clustering with sentence embedding representation

rubric={points}

Your tasks:

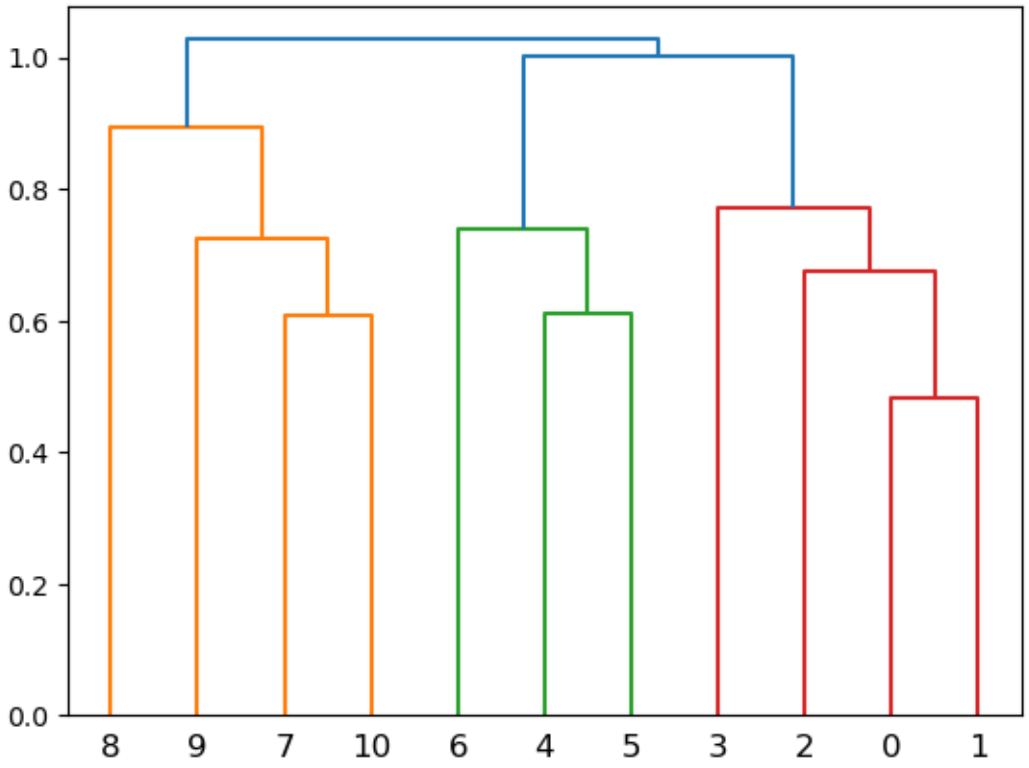
Try hierarchical clustering on `emb_sents`. In particular 1. Create and show a dendrogram with `complete` linkage and `metric='cosine'` on this toy dataset. 2. Create flat clusters using `fcluster` with appropriate hyperparameters and store cluster labels to `hier_emb_labels` variable below.

Solution_1.5

Points: 3

[16]:
```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

dendro = linkage(emb_sents, method='complete', metric='cosine')
dendrogram(dendro, color_threshold=1)
```

```
plt.show()
```



[17]:
```
hier_emb_labels = fcluster(dendro, 0.9, criterion='distance')
hier_emb_labels
```

[17]:
```
array([3, 3, 3, 3, 2, 2, 2, 1, 1, 1, 1], dtype=int32)
```

[18]:
```
# hier_emb_labels = fcluster(Z, 3, criterion="maxclust") # alternative solution
```

[19]:
```
wiki_df["emb_hierarchical"] = hier_emb_labels
wiki_df
```

[19]:

	wiki query	\
0	Artificial Intelligence	
1	Deep learning	
2	Unsupervised learning	
3	Quantum Computing	
4	Environmental protection	
5	Climate Change	
6	Renewable Energy	
7	Biodiversity	
8	French Cuisine	

```
9   Bread food
10  Dumpling food

          text  \
0   It is a field of research in computer science that develops and studies
methods and software that enable machines to perceive their environment and use
learning and intelligence to take actions that maximize their chances of
achieving defined goals.
1   The field takes inspiration from biological neuroscience and is centered
around stacking artificial neurons into layers and "training" them to process
data.
2   The training data is processed, building a function that maps new data to
expected output values.
3   On small scales, physical matter exhibits properties of both particles and
waves, and quantum computing leverages this behavior using specialized hardware.
4   Its objectives are to conserve natural resources and the existing natural
environment and, where it is possible, to repair damage and reverse trends.
5   Climate change in a broader sense also includes previous long-term changes
to Earth's climate.
6   The most widely used renewable energy types are solar energy, wind power,
and hydropower.
7   It can be measured on various levels.
8   In the 14th century, Guillaume Tirel, a court chef known as "Taillevent",
wrote Le Viandier, one of the earliest recipe collections of medieval France.
9   Throughout recorded history and around the world, it has been an important
part of many cultures' diet.
10  The dough can be based on bread, wheat or other flours, or potatoes, and it
may be filled with meat, fish, tofu, cheese, vegetables, or a combination.

    n_words  bow_kmeans  emb_kmeans  emb_dbscan  emb_hierarchical
0   40       2           2           0           3
1   25       1           2           0           3
2   18       1           2           0           3
3   24       1           2           0           3
4   26       0           0           0           2
5   16       1           0           0           2
6   17       1           0           0           2
7   8        1           1           1           1
8   31       1           1           2           1
9   20       1           1           1           1
10  36       1           1           1           1
```

1.4.6 1.6 Discussion

rubric={points}

Your tasks:

1. Reflect on and discuss the clustering results of the methods you explored in the previous

exercises, focusing on the following points:

- effect of input representation on clustering results
- whether the clustering results match with your intuitions and the challenges associated with getting the desired clustering results with each method

Solution_1.6

Points: 4

1. KMeans (BOW): the `kmeans_bow` clustering did not result in similar clusters as manual clustering, the grouping does not make sense. This is likely due to the fact that the BOW representation does not capture the semantic information of the sentence well, as it only captures the frequency of words in the sentence.
2. KMeans (Sentence Embeddings): the `emb_kmeans` clustering nearly resulted in the same clusters as the manual clustering, which is a good sign. With the only misclassification where biodiversity was grouped with food (perhaps because the text is so short). This method performed better compared to the BOW representation, as it captures the semantic information of the sentence better in a higher dimension vector space.
3. DBScan: the `dbscan_emb` clustering did not result in similar clusters as manual clustering, the grouping was not able to separate "Computer Science" and "Environmental science" clusters at all. Despite tuning hyperparameters, the clustering results were not as expected. This is likely caused by the fact that clusters are not well separated in the high dimensional space.
4. Heirarchical: the `hier_emb` clustering resulted in the same clusters as `emb_kmeans`, which is good. The dendrogram shows relatively even splits. The cosine distance metric was able to separate the clusters well. Using sentence embeddings seemed to be effective at matching our intuitions as sentences with similar semantics were merged together successfully.

Overall, the sentence embedding representation performed better than the BOW representation. The KMeans clustering was the easiest and quickest to use since the number of clusters was known. DBScan performed worse and was the most difficult to use as it required tuning of hyperparameters to get meaningful clusters. Moreover, the data lacked clear separation for DBScan to be effective. Hierarchical clustering performed well and was not to difficult as its possible to restrict by `maxclust` to get the desired number of clusters.

1.4.7 1.7 Visualizing clusters

rubric={points:4}

One approach to working with unlabeled data is visualization. That said, our data is high-dimensional, making it challenging to visualize. Take sentence embedding representation as an example: each instance is depicted in 768 dimensions. To visualize such high-dimensional data, we can employ dimensionality reduction techniques to extract the most significant 2 or 3 components, and then visualize this low-dimensional data.

Given data as a `numpy` array and corresponding cluster assignments, the `plot_umap_clusters` function below transforms the data by applying dimensionality reduction technique called UMAP to it and plots the transformed data with different colours for different clusters.

> *Note: At this point we are using this function only for visualization and you are not expected to understand the UMAP part.*

You'll have to install the `umap-learn` package in the course conda environment either with `conda` or `pip`, as described in the documentation.

```
> conda activate cpsc330
> conda install -c conda-forge umap-learn
```

or

```
> conda activate cpsc330
> pip install umap-learn
```

If you get an error with the import below try

```
pip install --upgrade numba umap-learn
```

Your tasks:

1. Visualize the clusters created by the methods above using `plot_umap_clusters` function below. In other words, visualize clusters identified by each of the methods below.
   - K-Means with bag-of-words representation
   - K-Means with sentence embedding representation
   - DBSCAN with sentence embedding representation
   - Flat cluster of hierarchical clustering with sentence embedding representation

[20]:
```
import umap
```

[21]:
```
def plot_umap_clusters(
    data,
    cluster_labels,
    raw_sents=wiki_df["text"],
    show_labels=False,
    size=50,
    n_neighbors=15,
    title="UMAP visualization",
    ignore_noise=False,
):
    """
    Carry out dimensionality reduction using UMAP and plot 2-dimensional␣
 ↪clusters.

    Parameters
    ----------
    data : numpy array
        data as a numpy array
    cluster_labels : list
        cluster labels for each row in the dataset
    raw_sents : list
        the original raw sentences for labeling datapoints
    show_labels : boolean
        whether you want to show labels for points or not (default: False)
    size : int
```

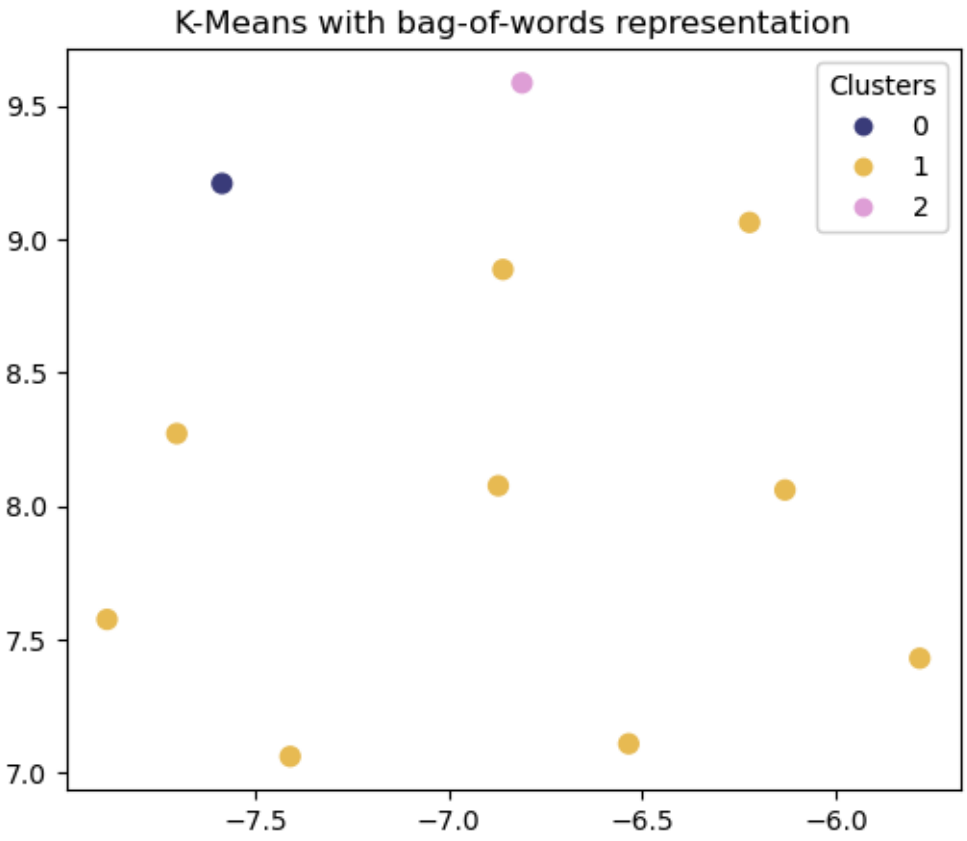
```
        size of points in the scatterplot
    n_neighbors : int
        n_neighbors hyperparameter of UMAP. See the documentation.
    title : str
        title for the visualization plot

    Returns
    -----------
    None. Shows the clusters.
    """

    reducer = umap.UMAP(n_neighbors=n_neighbors, random_state=42)
    Z = reducer.fit_transform(data)  # reduce dimensionality
    umap_df = pd.DataFrame(data=Z, columns=["dim1", "dim2"])
    umap_df["cluster"] = cluster_labels

    if ignore_noise:
        umap_df = umap_df[umap_df["cluster"] != -1]

    labels = np.unique(umap_df["cluster"])

    fig, ax = plt.subplots(figsize=(6, 5))
    ax.set_title(title)

    scatter = ax.scatter(
        umap_df["dim1"],
        umap_df["dim2"],
        c=umap_df["cluster"],
        cmap="tab20b",
        s=size,
        #edgecolors="k",
        #linewidths=0.1,
    )

    legend = ax.legend(*scatter.legend_elements(), loc="best", title="Clusters")
    ax.add_artist(legend)

    if show_labels:
        x = umap_df["dim1"].tolist()
        y = umap_df["dim2"].tolist()
        for i, txt in enumerate(raw_sents):
            ax.annotate(" ".join(txt.split()[:10]), (x[i], y[i]))
    plt.show()
```

Solution_1.7

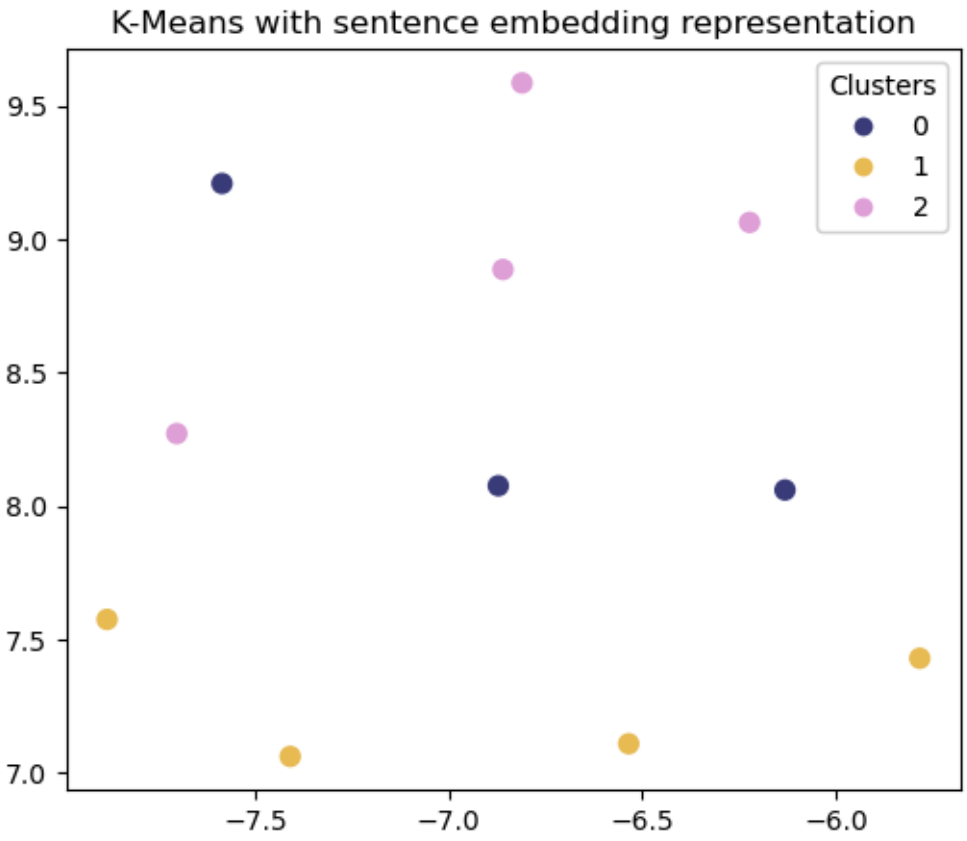
Points: 4

[22]:
```
# 1. Visualize the clusters created by the methods above using␣
 ↪`plot_umap_clusters` function below. In other words, visualize clusters␣
 ↪identified by each of the methods below.
#     - K-Means with bag-of-words representation
#     - K-Means with sentence embedding representation
#     - DBSCAN with sentence embedding representation
#     - Flat cluster of hierarchical clustering with sentence embedding␣
 ↪representation

plot_umap_clusters(
    emb_sents,
    wiki_df["bow_kmeans"],
    title="K-Means with bag-of-words representation",
)

plot_umap_clusters(
    emb_sents,
    wiki_df["emb_kmeans"],
    title="K-Means with sentence embedding representation",
)

plot_umap_clusters(
    emb_sents,
    wiki_df["emb_dbscan"],
    title="DBSCAN with sentence embedding representation",
)

plot_umap_clusters(
    emb_sents,
    wiki_df["emb_hierarchical"],
    title="Flat cluster of hierarchical clustering with sentence embedding␣
 ↪representation",
)
```

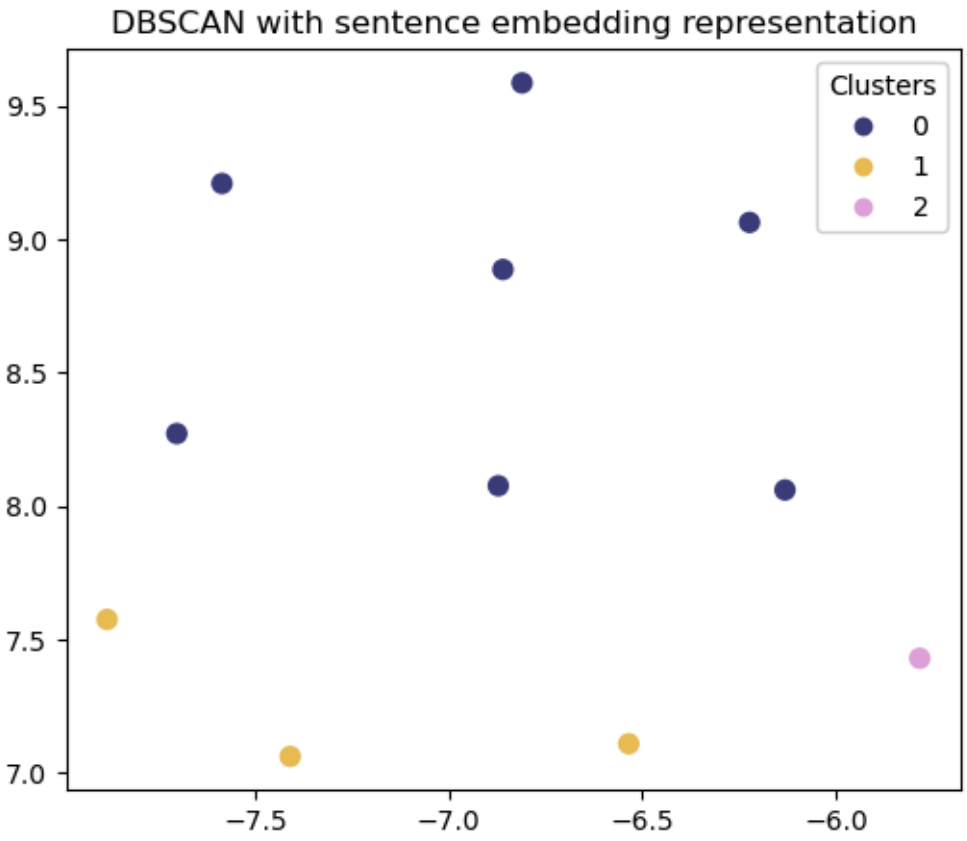
```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:2462:
UserWarning: n_neighbors is larger than the dataset size; truncating to
X.shape[0] - 1
  warn(
```



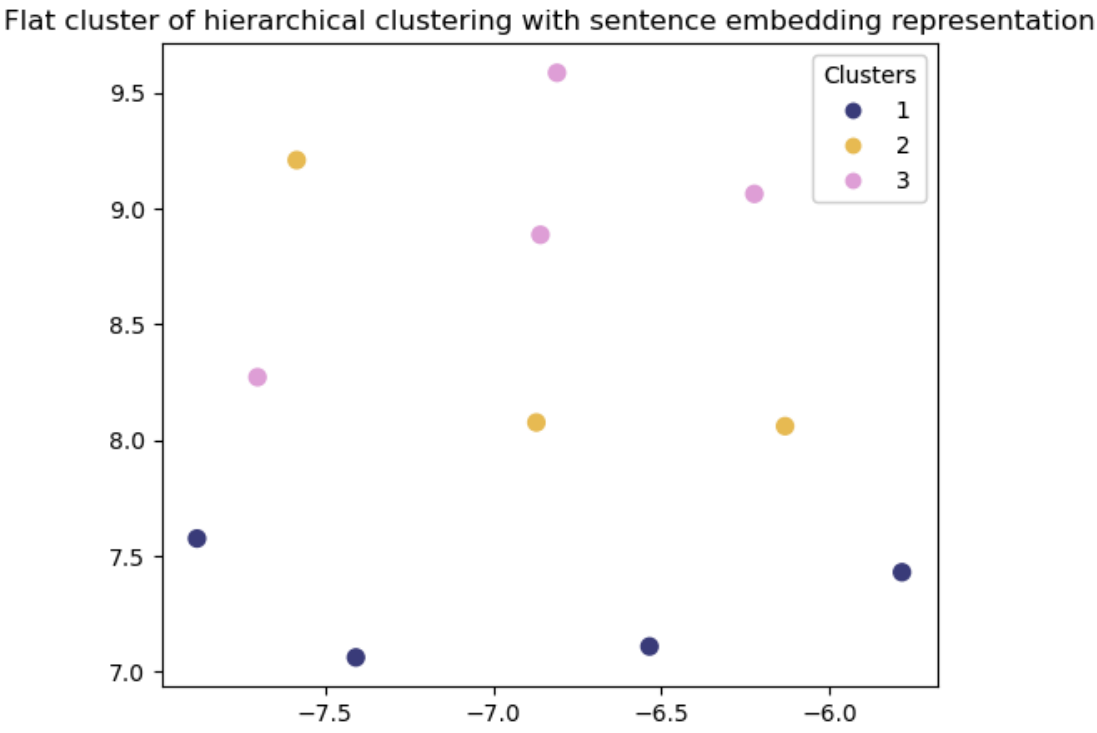

```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:2462:
UserWarning: n_neighbors is larger than the dataset size; truncating to
X.shape[0] - 1
  warn(
```




```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:2462:
UserWarning: n_neighbors is larger than the dataset size; truncating to
X.shape[0] - 1
  warn(
```




```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:2462:
UserWarning: n_neighbors is larger than the dataset size; truncating to
X.shape[0] - 1
  warn(
```



1.5 Exercise 2: Food.com recipes

Now that we have applied document clustering on a toy corpus, let's move to a more realistic corpus.

In the lecture, we worked on an activity of manually clustering food items and discussed challenges associated with it. We also applied different clustering algorithms to cluster food images. We'll continue this theme of clustering food items in this lab. But instead of images we will cluster textual description of food items, i.e., recipe names.

In this lab, we will work with a sample of Kaggle's Food.com recipes corpus. This corpus contains 180K+ recipes and 700K+ recipe reviews. In this lab, we'll only focus on recipes and **not** on reviews. The recipes are present in `RAW_recipes.csv`. Our goal is to find categories or groupings of recipes from this corpus based on their names.

Your tasks:

- Download `RAW_recipes.csv` and put it under the `data` directory in the homework folder.
- Run the code below. The dataset is quite large, and in this assignment, for speed, you will work with a sample of the dataset. The function `get_recipes_sample` below carries out some preliminary preprocessing and returns a sample of the recipes with most frequent tags.

  Note: Depending upon the capacity of your computer, feel free to increase or decrease the size of this sample by changing the value for `n_tags`. If you decide to go with a different value of `n_tags`, state it clearly in Exercise 2.1 so that the grader knows about it.

[23]:
```
orig_recipes_df = pd.read_csv("data/RAW_recipes.csv")
orig_recipes_df.shape
```

[23]:
```
(231637, 12)
```

[24]:
```
def get_recipes_sample(orig_recipes_df, n_tags=300, min_len=5):
    orig_recipes_df = orig_recipes_df.dropna()  # Remove rows with NaNs.
    orig_recipes_df = orig_recipes_df.drop_duplicates(
        "name"
    )  # Remove rows with duplicate names.
    # Remove rows where recipe names are too short (< 5 characters).
    orig_recipes_df = orig_recipes_df[orig_recipes_df["name"].apply(len) >=␣
 ↪min_len]
    # Only consider the rows where tags are one of the most frequent n tags.
    first_n = orig_recipes_df["tags"].value_counts()[0:n_tags].index.tolist()
    recipes_df = orig_recipes_df[orig_recipes_df["tags"].isin(first_n)]
    return recipes_df
```

[25]:
```
recipes_df = get_recipes_sample(orig_recipes_df)
recipes_df.shape
```

[25]:
```
(9100, 12)
```

[26]:
```
recipes_df["name"]
```

[26]:
```
42          i yam what i yam  muffins
101         to your health  muffins
129         250 00 chocolate chip cookies
138         lplermagronen
163         california roll   salad
                    …
231430      zucchini wheat germ cookies
231514      zucchini blueberry bread
231547      zucchini salsa burgers
231596      zuppa toscana
231629      zydeco salad
Name: name, Length: 9100, dtype: object
```

In the rest of the homework, we will use `recipes_df` above, which is a subset of the original dataset.

1.5.1 2.1 Longest and shorter recipe names

rubric={points:2}

Your tasks:

1. Print the shortest and longest recipe names (length in terms of number of characters) from `recipes_df`. If there is more than one recipe with the same shortest/longest length, store **one** of them in `shortest_recipe` and/or `longest_recipe` as a **string**.

Solution_2.1

Points: 2

[27]:
```
shortest_recipe = recipes_df.loc[recipes_df["name"].apply(len).idxmin(), "name"]
longest_recipe = recipes_df.loc[recipes_df["name"].apply(len).idxmax(), "name"]

shortest_recipe, longest_recipe
```

[27]:
```
('bread', 'baked tomatoes with a parmesan cheese crust and balsamic drizzle')
```

1.5.2 2.2 More EDA

rubric={points:2}

Your tasks: 1. Create a word cloud for the recipe names. You can use the `wordcloud` package for this, which you will have to install in the course environment.

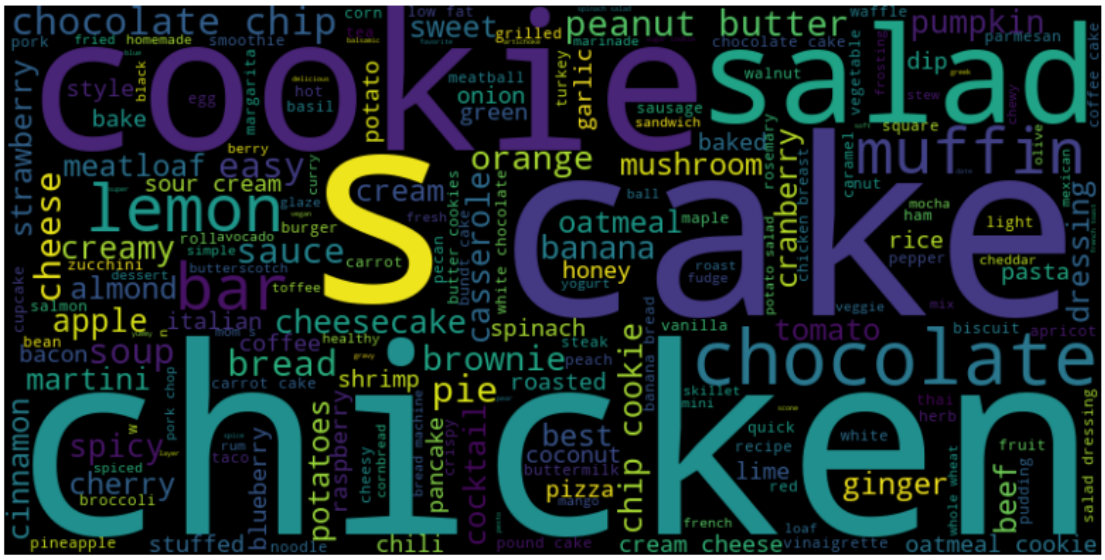
```
> conda activate cpsc330
> conda install -c conda-forge wordcloud
```

Solution_2.2

Points: 2

[28]:
```
import wordcloud

def plot_wordcloud(text, title=""):
    wc = wordcloud.WordCloud(width=800, height=400, max_words=200).
 ↪generate(text)
    plt.figure(figsize=(10, 10))
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(title)
    plt.show()

plot_wordcloud(" ".join(recipes_df["name"]), title="Word cloud of recipe names")
```

Word cloud of recipe names



1.5.3 2.3 Representing recipe names

rubric={points:3}

The next step is creating a representation of recipe names.

Your tasks:

1. Similar to Exercise 1, create sentence embedding representation of recipe names (`name` column in `recipes_df`). For the rest of the homework, we'll stick to the sentence embedding representation of recipe names.

   You might have to convert the recipe names to a list (`recipes_df["name"].tolist()`) for the embedder to work *If you create a dataframe with sentence embedding representation, set the index to* ***`recipes_df.index`*** *so that the indices match with the indices of the sample we are working with.*
   This might take a while to run.

Solution_2.3

Points: 3

[29]:
```
embedder = SentenceTransformer("paraphrase-distilroberta-base-v1")
embeddings = embedder.encode(recipes_df["name"].tolist())
emb_recipe_df = pd.DataFrame(embeddings, index=recipes_df.index)
```

[30]:
```
emb_recipe_df
```

[30]:

	0	1	2	3	4	5	6	\
42	-0.333474	0.227865	-0.307339	0.410549	0.917103	-0.345506	0.305810	
101	-0.024523	0.246223	-0.055708	0.358273	0.454786	-0.088054	0.260368	

```
129    -0.026562  0.194671  0.038102 -0.099181  0.653784 -0.230868  0.064517
138    -0.168002 -0.219218  0.330761  0.302196 -0.173169  0.204557  0.192390
163     0.061076 -0.333798  0.242905  0.395977 -0.466468  0.496505 -0.136754
…          …       …       …       …       …       …       …
231430 -0.009714  0.200162  0.018329  0.237816  0.748988  0.121919  0.087918
231514 -0.106532 -0.034574  0.160070  0.258504  0.882480  0.091899  0.008815
231547  0.016149 -0.053035 -0.035097 -0.021835  0.735161  0.245519 -0.151837
231596  0.026659  0.202483  0.344633 -0.138708  0.514424  0.630948 -0.003165
231629 -0.031947 -0.258189 -0.079766 -0.507932 -0.155580  0.635225 -0.127390

               7         8         9  …       758       759       760  \
42      0.256676 -0.147712  0.040727  … -0.053025 -0.209463  0.362418
101     0.231947  0.093013 -0.007835  …  0.144860 -0.435459  0.208344
129     0.001376 -0.061280  0.232094  …  0.169605  0.239354  0.392649
138     0.140975  0.303174  0.038063  …  0.019094 -0.002448  0.437833
163     0.122097  0.114248  0.065317  …  0.044438 -0.213657  0.410353
…          …       …       …  …       …       …       …
231430  0.216755 -0.291041  0.074165  … -0.120459  0.162877  0.480064
231514  0.372013 -0.266542  0.085018  …  0.063240 -0.099132  0.332694
231547  0.003954  0.102566  0.001363  …  0.102296  0.023821  0.602685
231596 -0.102897 -0.434900 -0.192056  … -0.162457  0.130696  0.113645
231629  0.062607  0.146174 -0.004416  …  0.043947 -0.066851  0.167459

             761       762       763       764       765       766       767
42     -0.246085  0.208379 -0.024874  0.506561  0.197738 -0.386421 -0.002184
101    -0.002655 -0.081200 -0.130489  0.452378  0.186310 -0.278500  0.028580
129     0.271398 -0.093971  0.060101 -0.281418  0.133211 -0.122345 -0.109863
138     0.563785  0.033442  0.187658  0.529252 -0.233786  0.509931  0.410715
163    -0.233599 -0.224422  0.094330  0.173351  0.105781 -0.114749 -0.150677
…          …       …       …       …       …       …       …
231430  0.235902  0.146278  0.252980 -0.063639 -0.066044 -0.133067 -0.034040
231514  0.031869  0.582796  0.216874  0.241107  0.091758 -0.261551 -0.082310
231547 -0.067070  0.177535  0.253982  0.242439 -0.436000 -0.053910  0.012423
231596  0.186650 -0.155393  0.130507  1.146238 -0.240861  0.236830  0.177123
231629 -0.087434 -0.097304  0.265510 -0.099823 -0.569383 -0.219148 -0.033525

[9100 rows x 768 columns]
```

1.6 Exercise 3: Clustering recipe names

In this exercise you'll cluster recipe names with some of the clustering algorithms we have seen in class. This will also involve making some attempts to pick reasonable hyperparameter values for each clustering method based on the quality of the resulting clusters. For example, for KMeans, you need to specify the number of clusters in advance, which is often challenging on real-world datasets. For DBSCAN, you need to pick appropriate `eps` and `min_samples`. For hierarchical clustering, you need to pick a suitable linkage criterion, distance metric, and prune the tree so that it's possible to visualize and interpret it.

Here are some methods which may help you with picking reasonable values for the hyperparameters. - Visualize the Elbow plot (KMeans). - Visualize Silhouette plots. - Visualize resulting clusters using `plot_umap_clusters` function from Exercise 1. - Sample some recipes from each cluster, manually inspect whether there are coherent semantic themes. (For this, you may use the function `print_clusters` given below.)

> You may use the `yellowbrick` package for visualizing the Elbow plot and the Silhouette plots. You can intall it with

```
conda install -c districtdatalabs yellowbrick
```

Note that the process of picking reasonable hyperparameter values will be exploratory, iterative, and will involve manual inspection and judgment, as there is no ground truth to verify how well the model is doing. In your solutions, please do not include everything you try. Only present the results of the most informative trials. Add a narrative to your answer so that it's easy for the grader to follow your choices and reasoning.

[31]:
```
def print_clusters(recipes_df, cluster_labels, n_recipes=10, replace=False,␣
↪random_state=None):
    """
    Given recipes_df containing recipe names and cluster assignment (labels),
    sample and print n_recipes recipes per cluster.

    Parameters
    -----------
    recipe_df : pandas dataframe
        recipes dataframe containing recipe names in the "name" column
    cluster_labels : ndarray or a list
        cluster labels for each row in recipes_df
    n_recipes : int
        number of examples to sample from each cluster
    replace: bool
        replace flag to pass to the sampling of recipe names

    Returns
    -----------
    None
    """

    grouped = (
        pd.DataFrame(
            {
                "name": recipes_df["name"],
                "cluster_label": cluster_labels,
            }
        )
        .sort_values("cluster_label")
        .groupby("cluster_label")
```

```
    )

    for name, group in grouped:
        print(f"Cluster {name}")
        print(("----------").format(""))
        sample_size = min(n_recipes, len(group))
        print("\n".join(group.sample(sample_size,␣
↪random_state=random_state)['name'].tolist()))
        print("\n\n")
```

1.6.1 3.1 K-Means

rubric={points:6}

Your tasks:

1. Cluster recipe titles using KMeans. Make some attempts to determine the optimal number of clusters.
2. Pick one or two best models and justify your choice.

Solution_3.1

Points: 6

We selected the k-value 18 by using the elbow method, where the error decreases the most within the range, but also avoids overfitting on the training data. The metric used to evaluate the clustering quality is the "within-cluster sum of squares" (or "distortion")

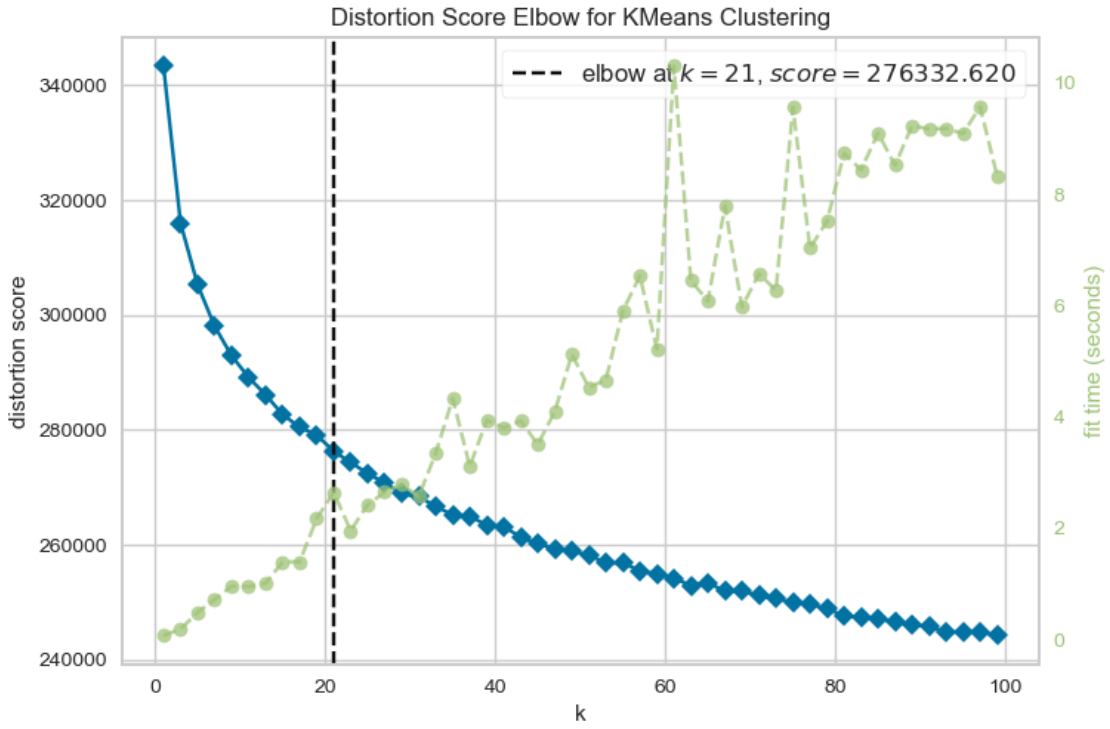
Note: the results from the elblow method fluctuated, and the most optimal k-value fluctuated around 18. So we decided to go with 18 as the k-value, even though the elbow in the graph shows 21 as the most optimal choice.

From the printed cluster results, we can see that the clusters are somewhat coherent, which suggest that our selected number of clusters is reasonable. Moreover, in the UMAP plot, we can see clear groups of clusters – although some clusters are overlapping, which is expected given the high-dimensional space.

[32]:
```
from sklearn.metrics import silhouette_score

def get_silhouette_score(x, labels):
    return -1 if len(set(labels)) <= 1 else silhouette_score(x, labels)

def silhouette_scorer(estimator, x):
    return get_silhouette_score(x, estimator.fit_predict(x))
```

[ ]:
```
from yellowbrick.cluster import KElbowVisualizer, kelbow_visualizer

k_values = range(1, 100, 2)

visualizer = KElbowVisualizer(KMeans(n_init='auto'), k=k_values)
visualizer.fit(emb_recipe_df)
```

```
visualizer.show()

best_k = visualizer.elbow_value_
best_k
```



[ ]: 21

[34]:
```
kmeans = KMeans(n_init='auto', n_clusters = 18, random_state=42)
clusters = kmeans.fit_predict(emb_recipe_df)
```

[35]:
```
print_clusters(recipes_df, clusters)
```

```
Cluster 0
----------
olive marinate
dry martini
champagne martini
rubytini
candy corn martini
houidini bars
muddled basil martini
nilla wafer martini
pallini fresco
```

pumpkin martini recipe

Cluster 1
---
vanilla coke alcoholic
road warrior spicy real hot chocolate mix
lemon and lime tango cocktail
best ever cocktail sauce
nuclear ice tea
herb infused salt and pepper
sweet   spicy dressing
zesty orange coffee
caramel coffee indulgence
root beer sloppy joes

Cluster 2
---
the next best thing to brad pitt
57 t bird with hawaiian plates
nicole s teriyaki salmon
down home shepherd s pie
skinny bride s guide to ham and potato casserole
broccoli casserole jane and michael stern
haitian diri ak pwa rice and beans
my momma s meatloaf
my fcd sammie fresh colourful and delish
freddy fudpucker

Cluster 3
---
creamy no fat coleslaw
dora s prize winning cheesecake
my ultimate cheeseburger
no bake cheesecake
low sugar   low fat lemon cheesecake
pistachio nut swirl cake or coffeecake
farmhouse triple layer pumpkin cheesecake pie
pumpkin cheesecake with caramel swirl
cheesecake shot
cheesecake lemon bars

Cluster 4
---
health nut s demise cookies
melt in your mouth sugar cookies
norwegian almond cookies
dream cookies
ciasteczka z orzechami polish walnut cookies
peanut butter chocolate chip cookie bars
minty cocoa fudge sandwich cookies
lovely lemon bar cookies
scratch me back cookies
lemon nut refrigerator cookies

Cluster 5
---
blue tail fly
orange brandy cooler
cape codder
onion lover s twist
fruit fantasy
biscuit sliders
deliciously soft gingersnaps
cuban breeze
white dragon
five star bars

Cluster 6
---
baked salmon patties or muffins
cranberry cheesecake muffins
lemon blueberry poppy seed muffins
favorite rhubarb muffins
healthy banana muffins
strawberry macadamia nut muffins
cocoa oatmeal muffins
easy raisin bran muffins
cherry yogurt muffins
low fat baking mix muffins

Cluster 7
---

corn in creamy dill sauce
chicken   roasted red potatoes
creamy herb mashed potatoes
warm mushroom spread
rainy day collards and lentil soup
best ever green bean casserole   fresh
easy fruit soup   whole wheat scones
crockpot pork chops and potatoes in sauce
moroccan vegetable soup
slow cooker north woods wild rice soup

Cluster 8
----
salty chihuahua
branston pickle balsamic vinaigrette
american flag
cranberry rita
toasted coco colada
mango avocado margarita
lima bean cassoulet
vegetable fred
rigatoni campagnolo like carrabba s
all purpose hawaiian marinade

Cluster 9
----
yellow sour cream cupcakes
double rich chocolate cake
chocolate cake to die for
lemon cake with lemon filling and lemon butter frosting
triple malted chocolate cake with vanilla malted frosting
bundt cheese cake with strawberry sauce
crepe cake with espresso chocolate glaze
lemon and yogurt cake   simple and best
lemon cornmeal cake with lemon glaze and crushed blueberry sauce
caramel latte cake

Cluster 10
----
buttery chocolate crinkles
warm nutty caramel brownies
butter chocolate bliss martini

chocolate raisin nut crunch bars
chocolate raspberry layer bars
rich chocolate drops
peanut butter paisley brownies
sweetheart bars
honey chocolate chippers
oatmeal peanut butter chocolate chip bars

Cluster 11
----
the real deal caesar salad
jenn s egg salad
onion n orange green salad
grilled steak and onion salad
almost empty dijon mustard jar vinaigrette salad dressing
shells warm chicken salad
grilled chicken salad with raspberry vinaigrette
romaine salad
red skin potato salad
dilled potato and grilled corn salad

Cluster 12
----
butter bundt cake
the ultimate carob cake
praline turtle cake
1 2 3 4 cake orange
almond lemon cake
jack robinson cake
lemon frosted lemon cake
jimmy carter cake
east 62nd street lemon cake
healthy berry bramble cake

Cluster 13
----
waffled french toast
emeril s i love gaaaaahlic garlic bread spread
excellent gingerbread
rye batter bread
cordon bleu loaf
poor boy loaf

starbucks banana walnut bread
authentic indian taco fry bread
egg bread
lemon almond tea bread

Cluster 14
---
pork medallions with fig port wine sauce
best monte cristo sandwich
grilled portobello mushroom sandwiches with basil aioli
spanish main stuffed poppers
sage scented ziti with broccoli pine nuts and orange zest
orecchiette with spicy sausage and broccoli rabe
gnocchi with chicken sausage bell pepper and fennel
creamy baked lemon pasta
chili cheese croissant corndogs
rustic country potato sausage pizza 5fix

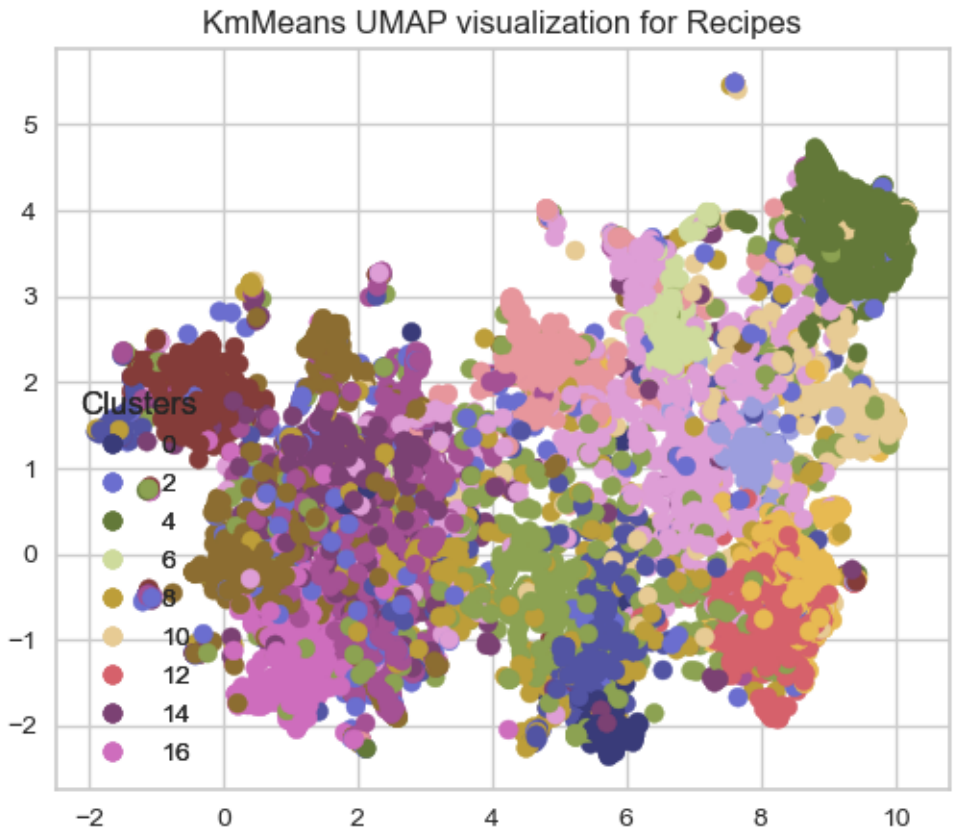
Cluster 15
---
beyond basic burgers 4 different burgers
szechuan noodles with pork
meatloaf squared
hunt s bruschetta chicken grill
breaded fried cube steak and milk gravy
smothered steak with mushroom gravy
salmon burgers patties with a herb sauce
teriyaki meatloaf
meatballs in creamy mushroom sauce
hungry man pork tenderloin

Cluster 16
---
raspberry chicken
valarie s chicken bruschetta
pesto chicken roll ups
creme de volaille french cream of chicken soup
barefoot contessa s tequila lime chicken
creamy chicken cabbage casserole
louisiana style blackened chicken
pho ga chicken noodle soup
tequila honey glazed chicken with jalapeno

```
prosciutto stuffed chicken



Cluster 17
----------
fat blasting berry smoothie
lemon poppy seed tart
real  strawberry pie with french cream topping
reverse whoopie pies
rhubarb cream pie
new england corn pudding
oriental almond ponzu slaw
root beer pancakes
chocolate french toast  pain perdu  by melissa d arabian
apple pumpkin tart
```

[36]:
```python
from sklearn.metrics import silhouette_score

recipes_df["emb_KMeans"] = clusters

plot_umap_clusters(
    emb_recipe_df,
    clusters,
    title="KmMeans UMAP visualization for Recipes",
)

kmeans_silueta = silhouette_score(emb_recipe_df, clusters)
print(f"Silhouette score for KMeans: {kmeans_silueta}")
```

```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
```




```
Silhouette score for KMeans: 0.01470062043517828
```

1.6.2 3.2 DBSCAN

rubric={points:6}

Your tasks:

1. Cluster recipe names using `DBSCAN` with `metric="cosine"`. Make some attempts to tune the hyperparameters `eps` and `min_samples`.

Solution_3.2

Points: 6

We opted to use a Randomized search that is scored on silhouette score here to strike a good balance between speed, and accuracy for finding good hyperparameters to train DBSCAN on.

Even with hyperparameter optimization the DBSCAN model created two clusters, with one cluster being much larger. However, the UMAP visualization clearly shows that the results are not intuitive, and it is not clear how to interpret the clusters.

Due to the fact that there is no clear separation in the high-dimensional space, DBSCAN is not a good choice for this dataset. The uniform density of the dataset makes it difficult for DBSCAN to

find meaningful clusters.

It seems that due to the way the silhouette score is calculated, the DBSCAN model can achieve a higher score despite producing obviously flawed results. This is likely due to the fact that the silhouette score is calculated based on the distance between the points and the distance between the points and the nearest cluster. Since the points are so close together, the silhouette score is high, even though the clustering is not meaningful.

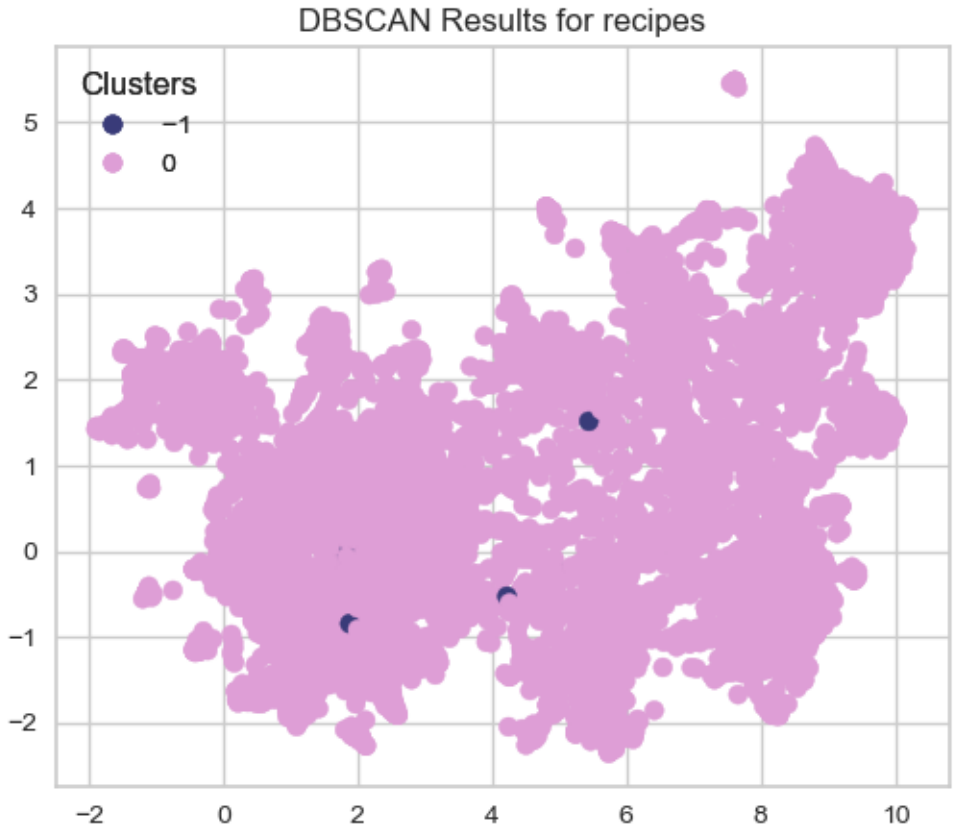
[37]:
```
from sklearn.model_selection import ParameterGrid
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.metrics import silhouette_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import make_scorer
from scipy.stats import uniform, randint

param_dist = {
    'eps': uniform(0.05, 2),
    'min_samples': randint(1, 100)
}

random_search = RandomizedSearchCV(
    DBSCAN(metric='cosine'),
    param_distributions=param_dist,
    n_iter=50,
    scoring=silhouette_scorer,
    n_jobs=-1,
    cv=3,
    random_state=42
)

random_search.fit(emb_recipe_df)

best_params = random_search.best_params_
best_score = random_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Silhouette Score: {best_score:.2f}")
```

```
Best Parameters: {'eps': 0.6584844859190755, 'min_samples': 22}
Best Silhouette Score: 0.17
```

[38]:
```
best_dbscan = random_search.best_estimator_
dbscan_clusters = best_dbscan.labels_
recipes_df["emb_dbscan"] = dbscan_clusters

plot_umap_clusters(
    emb_recipe_df,
    dbscan_clusters,
    title="DBSCAN Results for recipes",
```

```
)

silhouette_score = silhouette_score(emb_recipe_df, dbscan_clusters)
print(f"Silhouette Score: {silhouette_score:.2f}")
```

```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
```




```
Silhouette Score: 0.17
```

[39]:
```
print_clusters(recipes_df, dbscan_clusters)
```

```
Cluster -1
----------
the roosters call
2000 flushes
tie me to the bedpost
mob witnesses
an elbow in the boobs
the undercurrent
```

```
profiteroles



Cluster 0
----------
orange blossom honey madeleines
sassy middle eastern fish rub
miss kitty s oven roasted potatoes
pork steak w roasted potatoes and carrots
apricot munch
crispy nut cookies
melon with sweet lime dressing
island spam
french silk pie
homemade soup
```

1.6.3 3.3 Hierarchical clustering

rubric={points:6}

Your tasks:

1. Try hierarchical clustering with `metric="cosine"` on this problem. Show a dendrogram by using a suitable truncation method.
2. Create flat clusters by cutting the tree at the appropriate level.

   Note: Try orientation="left" of `dendrogram` for better readability of the dendrogram.

Solution_3.3

Points: 6

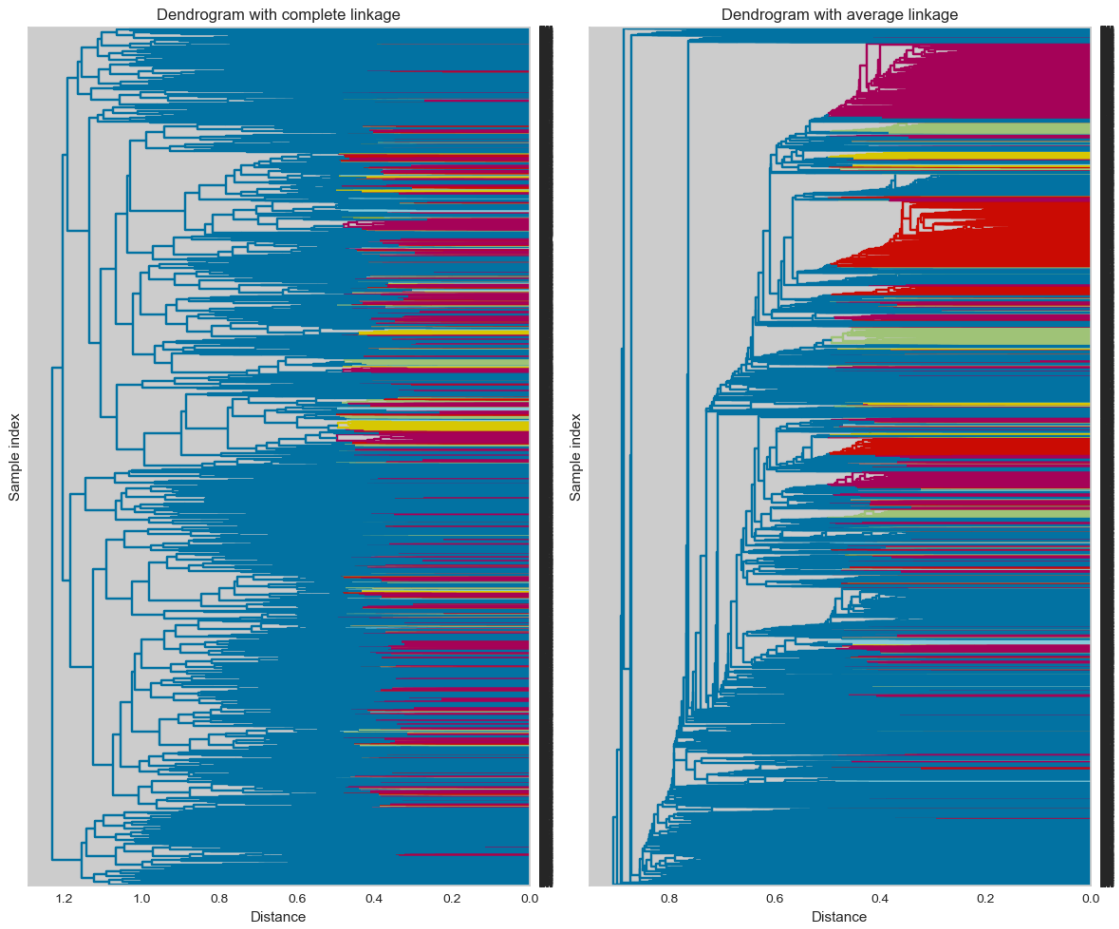
Given the reasonable results from the KMeans clustering, we decided to use limit the number of cluster to be around 18 for hierarchical clustering. We test the "complete" and "average" linkage methods with cosine distance (we couldnt add more since the kernel would crash for some reason)

We designed a custom function to find the best hyperparameters for the hierarchical clustering model. We perform grid search over the hyperparameters and keep the values that result in the largest silhouette score. It is also parallized to speed up the process. We found that the "average" linkage method with 10 clusters performed the best, and we decided to use this method for the final model.

Unfortunately, the hierarchical clustering model did not perform as well as the KMeans model. The clusters are not as coherent, and the UMAP visualization shows that the clusters are not well separated. This is likely due to the fact that the hierarchical clustering model is not able to separate the clusters well in the high-dimensional space. Heirarchical clustering is also sensitive to noise and outliers, which could be another reason for the poor performance.

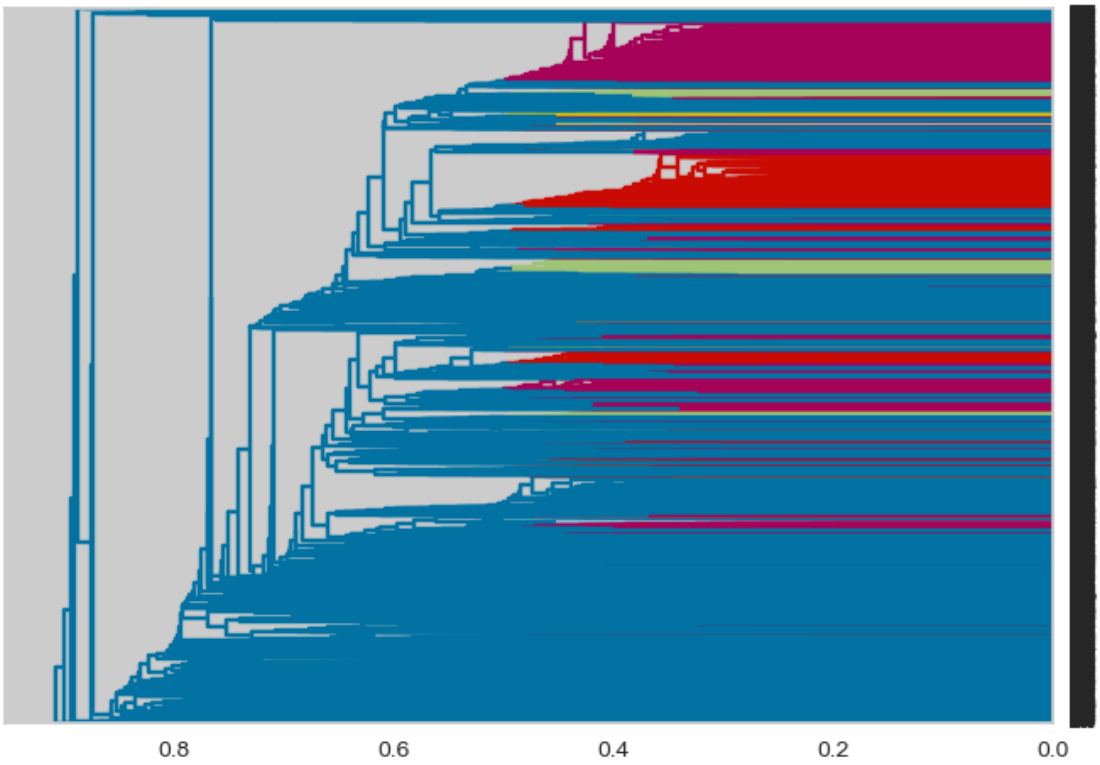
[40]:
```
from tqdm.notebook import tqdm
from joblib import Parallel, delayed
from sklearn.metrics import silhouette_score as sk_silhouette_score

final_silhouette = -1
final_max_clusters = -1
final_cluster_labels = None
final_linkage_method = None

fig, axes = plt.subplots(1, 2, figsize=(12, 10))
linkage_methods = ["complete", "average"]  # ward won't work with cosine

def compute_silhouette(dendro, max_clusters):
    hier_labels = fcluster(dendro, max_clusters, criterion='maxclust')
    if len(set(hier_labels)) > 1:
        silhouette = sk_silhouette_score(emb_recipe_df, hier_labels)
        return silhouette, max_clusters, hier_labels
    return -1, max_clusters, None

for ax, linkage_method in zip(axes.flatten(), tqdm(linkage_methods,␣
 ↪desc="Linkage Methods")):
    dendro = linkage(emb_recipe_df, method=linkage_method, metric='cosine')

    dendrogram(dendro, color_threshold=0.5, orientation='left', p=50, ax=ax)
    ax.set_title(f"Dendrogram with {linkage_method} linkage")
    ax.set_xlabel('Distance')
    ax.set_ylabel('Sample index')

    results = Parallel(n_jobs=-1)(delayed(compute_silhouette)(dendro,␣
 ↪max_clusters) for max_clusters in np.arange(10, 30))

    for silhouette, max_clusters, hier_labels in results:
        if silhouette > final_silhouette:
            final_silhouette = silhouette
            final_max_clusters = max_clusters
            final_cluster_labels = hier_labels
            final_linkage_method = linkage_method

plt.tight_layout()
plt.show()

print(f"Best Linkage Method: {final_linkage_method}")
print(f"Best Number of Clusters: {final_max_clusters}")
print(f"Best Silhouette Score: {final_silhouette:.2f}")
```

```
Linkage Methods:   0%|          | 0/2 [00:00<?, ?it/s]
```




```
Best Linkage Method: average
Best Number of Clusters: 10
Best Silhouette Score: 0.11
```

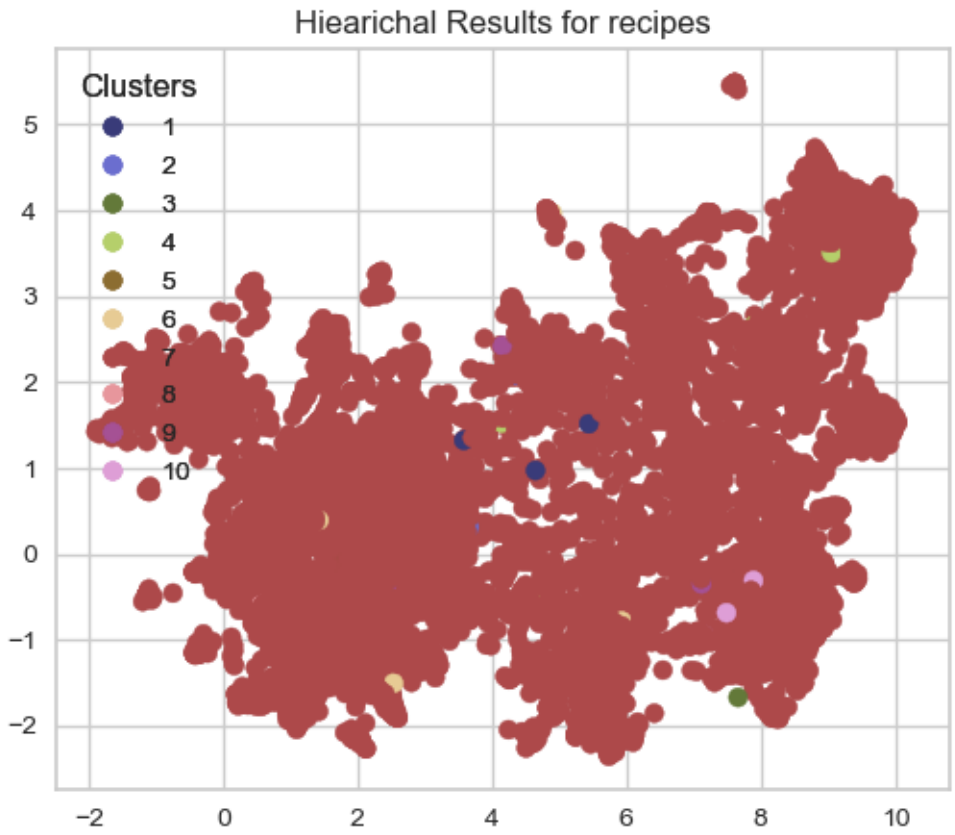
[41]:
```
dendro = linkage(emb_recipe_df, method='average', metric='cosine')
dendrogram(dendro, color_threshold=0.5, orientation='left')
plt.show()
```



[42]:
```
hier_emb_labels_recipes = fcluster(dendro, 10, criterion='maxclust')
recipes_df["emb_hierarchical"] = hier_emb_labels_recipes
```

[43]:
```
plot_umap_clusters(
    emb_recipe_df,
    hier_emb_labels_recipes,
    title="Hiearichal Results for recipes",
)

heir_silhouette = sk_silhouette_score(emb_recipe_df, hier_emb_labels_recipes)
print(f"Silhouette score for Hierarchical Clustering: {heir_silhouette}")
```

```
c:\Users\docto\miniconda3\envs\cpsc330\Lib\site-packages\umap\umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed
for parallelism.
  warn(
```




```
Silhouette score for Hierarchical Clustering: 0.10845261067152023
```

[44]:
```
print_clusters(recipes_df, hier_emb_labels_recipes)
```

```
Cluster 1
----------
parker house rolls  made by hand
an elbow in the boobs
tie me to the bedpost
90 minute batter rolls
yeast rolls
slow roll



Cluster 2
----------
betsy clear
just peachy
essential boy bribes
```

Cluster 3
---
the next best thing to brad pitt

Cluster 4
---
absolut trouble
betty crocker s hermits
2000 flushes
holyfield s ear
lemon jumbles
hermits
mincemeat hermits
pumpkin hermits
loved up
tvp crumbles

Cluster 5
---
mob witnesses

Cluster 6
---
schuylkill river punch
orange julius knock off
critter crunch
black jack cocktails or black jacks
claim jumper asian pear salad with citrus dressing
health chews
afternoon pick me up
black and whites
take out tabbouleh
claim jumper s crunchy spinach salad

Cluster 7
---
sage biscuits
kitchen sink cookies

```
potato wedges with cilantro lime mayonnaise
grapefruit and basil martini
baked french swirl toast
plain jane s brownies
quick shrimp and pasta
hidden veggie mac and cheese
lime ginger butter cookies
prose s protein packed power porridge



Cluster 8
----------
snap peas for parties



Cluster 9
----------
dragonsbreath slabs
blarney stones
herbed spareribs
razberri kazi
strawbreeze



Cluster 10
----------
wagon wheel slice
captain morgan s treasure
wrecked golf cart
island plunder rum cake
```

1.6.4 3.4 Manual interpretation of clusters

rubric={points:6}

Your tasks:

1. Label the topics/themes you see in the clusters created by different clustering methods.

2. Do you see a common theme across clusters created by different clustering methods? Do you see any differences between the clusters created by different clustering methods?

Solution_3.4

Points: 6

Generally, KNN grouped clusters based on recipes with similar whole words in its names for example:

Summary of Cluster Labels:

- Cluster 0: Cocktails and Drinks
- Cluster 1: Beverages and Sauces
- Cluster 2: Main Dishes
- Cluster 3: Cheesecakes and Desserts
- Cluster 4: Cookies
- Cluster 5: Mixed Drinks and Snacks
- Cluster 6: Muffins
- Cluster 7: Vegetable and Soup Dishes
- Cluster 8: Mixed Drinks and Marinades
- Cluster 9: Cakes
- Cluster 10: Chocolate and Brownies
- Cluster 11: Salads
- Cluster 12: Specialty Cakes
- Cluster 13: Breads
- Cluster 14: Gourmet Dishes
- Cluster 15: Meat and Poultry Dishes
- Cluster 16: Chicken Dishes
- Cluster 17: Desserts and Pies

Overall, KNN did a very good job separating what humans would consider 'similar' clusters together. For example, cluster 0 contains cocktails and drinks, cluster 1 contains beverages and sauces, cluster 2 contains main dishes, and so on. However, there are some clusters that are not as coherent. For example, cluster 5 contains mixed drinks and snacks, which is not a very coherent grouping. The confusion is likely due to the fact that the text is so short, and the model is not able to capture the semantic information of the sentence well. Moreover, some of the names are very misleading and do not accurately represent the recipe (literally 90% of the cocktail names).

In comparison, DBSCAN basically did not produce much of substance at all. Our best model only produced two clusters:

Summary of Cluster Labels:

- Cluster -1: Names of cocktails? – Generally weird and shorter names
- Cluster 0: Everything else

DBSCAN really struggled to find meaningful clusters out of the data, since this is density based perhaps this suggests a distinct lack of clear dense regions that the text data has. This is likely due to the curse of dimensionality, as we have a lot of features, and as a result, it may be very sparsely distributed, which makes it difficult to find pockets of high density to group together.

Hierarichal clustering created some interesting results, some clusters were significantly imbalenced, containing only a couple of recipes.

Summary of Cluster Labels:

- Cluster 1: Baked Goods and Rolls
- Cluster 2: Beverages and Drinks
- Cluster 3: Unique or Singular Items
- Cluster 4: Snacks and Miscellaneous
- Cluster 5: Miscellaneous or Outlier
- Cluster 6: Cocktails and Salads
- Cluster 7: Homemade Recipes and Cookies
- Cluster 8: Party Snacks
- Cluster 9: Themed Snacks and Drinks
- Cluster 10: Themed Desserts and Cocktails

Overall, the clusters are not as coherent as the KMeans model. For example, cluster 3 contains unique or singular items, which is not a very coherent grouping. The confusion is likely due to the fact that the text is so short. Moreover, there a good clusters but ones that can clearly be separated further, like Cluster 7 and 9.

In hiearichal clustering, similar clusters are grouped together first, and so the clusters with single items suggest that they are very significantly different from the other items that they get their own cluster. Because we only see a small sample from each cluster, it at first seems like hiearichal clustering tends to group in themes rather than specific key words unlike K-Means. But this is likely because some clusters are simply way larger than others, and so keywords that are similar like “cookies”, and “cakes” will be pooled together first, while the really weird clusters like mob witnesses, and brad pitt end up being pooled together last. As a result, we lose a lot of resolution on a majority of clusters as only outliers are specifically highlighted.

Being a distance based method, heiarichal clustering also suffers from the same issues as DBSCAN, where the high dimensionality of the data makes it difficult to find meaningful clusters.

In summary, KMeans likes to pool words by keywords – making it likely the best choice to use in a real world scenario, as people will probably want to know different recipes of things that are directly related.

DBSCAN has no clue what the clusters are, and likely needs furthur tuning and refinement.

And Hiearchical clustering tends to pool words that are the most different from each other, giving us a lot of imbalenced clusters, with very niche categories that may not be very large.

Before submitting your assignment, please make sure you have followed all the instructions in the Submission instructions section at the top.


WELL
DONE!