CPSC 330 - Applied Machine Learning

Homework 7: Clustering and recommender systems

Associated lectures: Lectures 14 and 15

Due date: Thursday, June 16, 2022 at 18:00

In [1]:
```
import os

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.cluster import DBSCAN, KMeans
from sklearn.feature_extraction.text import CountVectorizer

pd.set_option("display.max_colwidth", 0)
```

Instructions

---

rubric={points:2}

Follow the homework submission instructions.

You may work with a partner on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.
- 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.

Exercise 1: Document clustering toy example

In lecture 14, we looked at a popular application of clustering: customer segmentation. In this homework, we will work on another popular application, **document clustering**. A large amount of unlabeled text data is available out there (e.g., news, recipes, online Q&A), and clustering is a commonly used technique to organize this data in a meaningful way.

In this exercise, we will create a toy dataset with sentences from Wikipedia articles and cluster these sentences. In the next exercise, we'll move to a real corpus.

1.1 Sample sentences from Wikipedia articles

rubric={points:2}

The code below extracts first sentences of Wikipedia articles on a set of queries. You will need the `wikipedia` package installed in the course environment to run the code below.

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

You also need `nltk` library in the course environment.

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

Your tasks:

Run the code below and answer the following question.

1. Given this dataset, how many clusters would you expect a clustering algorithm to identify? How would you manually label these clusters?

> *Note: Feel free to experiment with queries of your choice. But stick to the provided list for the final submission so that it's easier for the TAs when they grade your submission.*

In [2]:
```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
# nltk.download("punkt")  # you need to run this line once
```

In [3]:
```
import wikipedia

queries = [
    "Orange Tree",
    "Grapefruit Tree",
    "Hexagon Shape",
    "English Language",
    "German Language",
    "Planet Jupiter",
    "Planet Venus"
]

wiki_dict = {"wiki query": [], "text": [], "n_words": []}
for i in range(len(queries)):
    sent = sent_tokenize(wikipedia.page(queries[i]).content)[0]
    wiki_dict["text"].append(sent)
    wiki_dict["n_words"].append(len(word_tokenize(sent)))
    wiki_dict["wiki query"].append(queries[i])
```

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

Out[3]:

	wiki query	text	n_words
0	Orange Tree	Citrus × sinensis (sometimes written Citrus sinensis), also known as the sweet oranges, is a commonly cultivated family of oranges that includes blood oranges and navel oranges.	32
1	Grapefruit Tree	The grapefruit (Citrus × paradisi) is a subtropical citrus tree known for its relatively large, sour to semi-sweet, somewhat bitter fruit.	26
2	Hexagon Shape	In geometry, a hexagon (from Greek ἕξ, hex, meaning "six", and γωνία, gonía, meaning "corner, angle") is a six-sided polygon or 6-gon.	36
3	English Language	English is a West Germanic language of the Indo-European language family, originally spoken by the inhabitants of early medieval England.	22
4	German Language	German (Deutsch, pronounced [dɔʏtʃ] (listen)) is a West Germanic language of the Indo-European language family, mainly spoken in Central Europe.	29
5	Planet Jupiter	Jupiter is the fifth planet from the Sun and the largest in the Solar System.	16
6	Planet Venus	Venus is the second planet from the Sun and is named after the Roman goddess of love and beauty.	20

BEGIN SOLUTION

I would expect a clustering algorithm to identify 3 to 4 clusters: fruit trees, natural languages, celestial objects, and possibly a geometric shapes category for Hexagon Shape or it could be considered to be an outlier.

END SOLUTION

1.2 `KMeans` with bag-of-words representation

rubric={points:4}

We have seen that before we pass text to machine learning models, we need to encode it into a numeric representation. So let's encode our toy dataset above (`wiki_df`) to a numeric representation.

First, let's try our good old friend: bag-of-words representation. The code below creates dense bag-of-words representation of Wikipedia sentences from 1.1 with `CountVectorizer` .

Your tasks:

Run the code below and answer the following questions.

1. Run `KMeans` clustering on the transformed data (`bow_sents`) with K = the number of clusters you identified in 1.1.
2. Examine clustering labels assigned by `KMeans` . Is `KMeans` doing a reasonable job in clustering the sentences?

> You can access cluster label assignments using `labels_` attribute of the clustering object.

In [4]:
```
vec = CountVectorizer(stop_words='english')
bow_sents = vec.fit_transform(wiki_df["text"]).toarray()
bow_df = pd.DataFrame(
    data=bow_sents, columns=vec.get_feature_names_out(), index=wiki_df.index
)
bow_df
```

Out[4]:

	angle	beauty	bitter	blood	central	citrus	commonly	corner	cultivated	deutsch	...	spoken	subtropical
0	0	0	0	1	0	2	1	0	1	0	...	0	0
1	0	0	1	0	0	2	0	0	0	0	...	0	1
2	1	0	0	0	0	0	0	1	0	0	...	0	0
3	0	0	0	0	0	0	0	0	0	0	...	1	0
4	0	0	0	0	1	0	0	0	0	1	...	1	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0
6	0	1	0	0	0	0	0	0	0	0	...	0	0

7 rows × 69 columns

BEGIN SOLUTION

In [5]:
```
kmeans_bow = KMeans(n_clusters=3, random_state=42)
kmeans_bow.fit(bow_sents)
wiki_df["bow_labels"] = kmeans_bow.labels_
wiki_df
```

Out[5]:

	wiki query	text	n_words	bow_labels
0	Orange Tree	Citrus × sinensis (sometimes written Citrus sinensis), also known as the sweet oranges, is a commonly cultivated family of oranges that includes blood oranges and navel oranges.	32	1
1	Grapefruit Tree	The grapefruit (Citrus × paradisi) is a subtropical citrus tree known for its relatively large, sour to semi-sweet, somewhat bitter fruit.	26	0
2	Hexagon Shape	In geometry, a hexagon (from Greek ἕξ, hex, meaning "six", and γωνία, gonía, meaning "corner, angle") is a six-sided polygon or 6-gon.	36	0
3	English Language	English is a West Germanic language of the Indo-European language family, originally spoken by the inhabitants of early medieval England.	22	2
4	German Language	German (Deutsch, pronounced [dɔʏtʃ] (listen)) is a West Germanic language of the Indo-European language family, mainly spoken in Central Europe.	29	2
5	Planet Jupiter	Jupiter is the fifth planet from the Sun and the largest in the Solar System.	16	0
6	Planet Venus	Venus is the second planet from the Sun and is named after the Roman goddess of love and beauty.	20	0

I would say `KMeans` is doing a good job here, given that we are training only on seven sentences. One could argue that it's only putting two sentences (regarding "Grapefruit Tree" and "Hexagon Shape") in the wrong clusters. Other than that it's correctly identifying the expected clusters.

> *Extra information: Similarity based on bag-of-words representation is based on the idea that when documents have similar column vectors, they tend to have similar meaning. Since we are training K-Means only on six examples, it's possible that we do not find any meaning patterns in the representation of the documents.*

END SOLUTION

1.3 Sentence embedding representation

rubric={points:6}

Clustering is sensitive to what kind of representation we use for the given data. Bag-of-words representation is limited in that it does not take into account word ordering and context. There are other richer representations of text, and we are going to use one such representation in this exercise.

The code below creates an alternative and a more expressive representation of sentences. We will call it *sentence embedding representation*. We'll use sentence transformer to extract these representations. At this point it's enough to know that this is an alternative representation of text which usually works better than simple bag-of-words representation. We will talk a bit more about embedding representations next week. You need to install `sentence-transformers` in the course conda environment to run the code below.

`conda install -n cpsc330 -c conda-forge sentence-transformers`

Your tasks:

Run the code below and answer the following questions.

1. How many dimensions (features associated with each example) are present in this representation?
2. Run `KMeans` clustering with sentence embedding representation of text (`emb_sents`) and examine cluster labels.
3. How well the sentences are clustered together?

In [6]:
```
from sentence_transformers import SentenceTransformer

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

In [7]:
```
emb_sents = embedder.encode(wiki_df["text"])
emb_sent_df = pd.DataFrame(emb_sents, index=wiki_df.index)
emb_sent_df
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	
0	-0.169783	0.328721	0.349129	0.160305	-0.001064	-0.042220	0.046996	0.344551	-0.096110	0.152743	...	-0
1	-0.246986	0.277209	0.370583	0.099333	-0.366400	0.168401	0.050352	0.403484	-0.127596	0.240492	...	0
2	0.245753	0.292599	-0.055407	0.269905	-0.148924	0.058780	0.037858	0.228929	0.197867	0.406216	...	0
3	-0.043948	0.300679	0.027356	0.522667	0.220695	0.083796	-0.030243	0.330987	0.268973	0.128816	...	0
4	-0.152196	0.336464	-0.008689	0.666325	0.257241	0.056739	0.087526	0.288248	0.229120	0.134044	...	0
5	-0.124168	0.215485	0.013107	0.143409	-0.636150	0.189037	0.114603	0.075671	0.000973	-0.188109	...	-0
6	0.071988	0.221728	0.100429	0.005925	-0.170285	-0.222390	0.293238	0.210783	0.178689	0.293481	...	-0

7 rows × 768 columns

BEGIN SOLUTION

With this representation each example is represented with 768 dimensions.

In [8]:
```
kmeans_emb = KMeans(n_clusters=3)
kmeans_emb.fit(emb_sents)
wiki_df["emb_labels"] = kmeans_emb.labels_
wiki_df
```

Out[8]:

	wiki query	text	n_words	bow_labels	emb_labels
0	Orange Tree	Citrus × sinensis (sometimes written Citrus sinensis), also known as the sweet oranges, is a commonly cultivated family of oranges that includes blood oranges and navel oranges.	32	1	2
1	Grapefruit Tree	The grapefruit (Citrus × paradisi) is a subtropical citrus tree known for its relatively large, sour to semi-sweet, somewhat bitter fruit.	26	0	2
2	Hexagon Shape	In geometry, a hexagon (from Greek ἕξ, hex, meaning "six", and γωνία, gonía, meaning "corner, angle") is a six-sided polygon or 6-gon.	36	0	1
3	English Language	English is a West Germanic language of the Indo-European language family, originally spoken by the inhabitants of early medieval England.	22	2	0
4	German Language	German (Deutsch, pronounced [dɔʏtʃ] (listen)) is a West Germanic language of the Indo-European language family, mainly spoken in Central Europe.	29	2	0
5	Planet Jupiter	Jupiter is the fifth planet from the Sun and the largest in the Solar System.	16	0	1
6	Planet Venus	Venus is the second planet from the Sun and is named after the Roman goddess of love and beauty.	20	0	1

Cluster assignments is working better than BOW. Now, "Hexagon Shape" is categorized with celestial objects, which is the only unclear categorization here.

> *Extra information: Unlike bag-of-words representation in the previous exercise, we are not learning the representation from the given data in this case. We are using transfer learning. The pre-trained model (paraphrase-distilroberta-base-v1) provides a rich sentence representation for a given query sentence which encodes meaningful similarities between words. For example it*

> *would know that "Jupiter" and "Venus" are similar. Because of this, K-Means is finding meaningful clusters.*

END SOLUTION

1.4 DBSCAN with cosine distance

rubric={points:8}

Let's 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 (sparse data), cosine similarities or cosine distances tend to work better. Given vectors u and v, the **cosine distance** between the vectors is defined as:

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

In this exercise, you'll use DBSCAN with cosine distances.

Your tasks

1. Use DBSCAN to cluster our toy data using sentence embedding representation (`emb_sents`) and `metric='cosine'` .
2. Briefly comment on the number of clusters identified and the cluster assignment given by the algorithm.

> *Note: You will also have to set appropriate values for the hyperparamters `eps` and `min_samples` to get meaningful clusters, as default values for these hyperparameters won't work on this toy dataset. In order to set appropriate value for `eps` , you may want to examine the distances given by sklearn's `cosine_distance` .*

BEGIN SOLUTION

In [9]:
```
from sklearn.metrics.pairwise import cosine_distances, cosine_similarity

print(pd.DataFrame(cosine_distances(emb_sents)))

dbscan_emb = DBSCAN(eps=0.65, min_samples=2, metric="cosine")
dbscan_emb.fit(emb_sents)
wiki_df["dbscan_emb"] = dbscan_emb.labels_
wiki_df
```

```
          0         1         2         3         4         5         6
0  0.000000  0.372911  0.788142  0.893928  0.955026  0.880982  0.724729
1  0.372911  0.000000  0.790605  0.916904  0.862229  0.725848  0.700491
2  0.788142  0.790605  0.000000  0.779869  0.850694  0.806560  0.739264
3  0.893928  0.916904  0.779869  0.000000  0.344403  0.848152  0.744589
4  0.955026  0.862229  0.850694  0.344403  0.000000  0.910820  0.828599
5  0.880982  0.725848  0.806560  0.848152  0.910820  0.000000  0.413087
6  0.724729  0.700491  0.739264  0.744589  0.828599  0.413087  0.000000
```

Out[9]:

	wiki query	text	n_words	bow_labels	emb_labels	dbscan_emb
0	Orange Tree	Citrus × sinensis (sometimes written Citrus sinensis), also known as the sweet oranges, is a commonly cultivated family of oranges that includes blood oranges and navel oranges.	32	1	2	0
1	Grapefruit Tree	The grapefruit (Citrus × paradisi) is a subtropical citrus tree known for its relatively large, sour to semi-sweet, somewhat bitter fruit.	26	0	2	0
2	Hexagon Shape	In geometry, a hexagon (from Greek ἕξ, hex, meaning "six", and γωνία, gonía, meaning "corner, angle") is a six-sided polygon or 6-gon.	36	0	1	-1
3	English Language	English is a West Germanic language of the Indo-European language family, originally spoken by the inhabitants of early medieval England.	22	2	0	1
4	German Language	German (Deutsch, pronounced [dɔʏtʃ] (listen)) is a West Germanic language of the Indo-European language family, mainly spoken in Central Europe.	29	2	0	1
5	Planet Jupiter	Jupiter is the fifth planet from the Sun and the largest in the Solar System.	16	0	1	2
6	Planet Venus	Venus is the second planet from the Sun and is named after the Roman goddess of love and beauty.	20	0	1	2

In our toy dataset, `DBSCAN` with `eps=0.65` and `min_samples=2` or `min_samples=1` is giving us desirable clustering on sentence embedding representations.

END SOLUTION

1.5 Visualizing clusters

rubric={points:5}

One thing we could do with unlabeled data is visualizing it. That said, our data is high dimensional (each example is represented with 768 dimensions) and high-dimensional data is hard to visualize. One way to visualize high-dimensional data is applying dimensionality reduction to get the most important (2 or 3) components of the dataset and visualizing this low-dimensional data.

Given data as a `numpy` array and cluster assignments, the `plot_pca_clusters` function below transforms the given data by applying dimensionality reduction and plots the transformed data into corresponding clusters.

> *Note: At this point we are using this function only for visualization and you are not expected to understand the PCA part. Feel free to modify the function as you see fit.*

Your tasks:

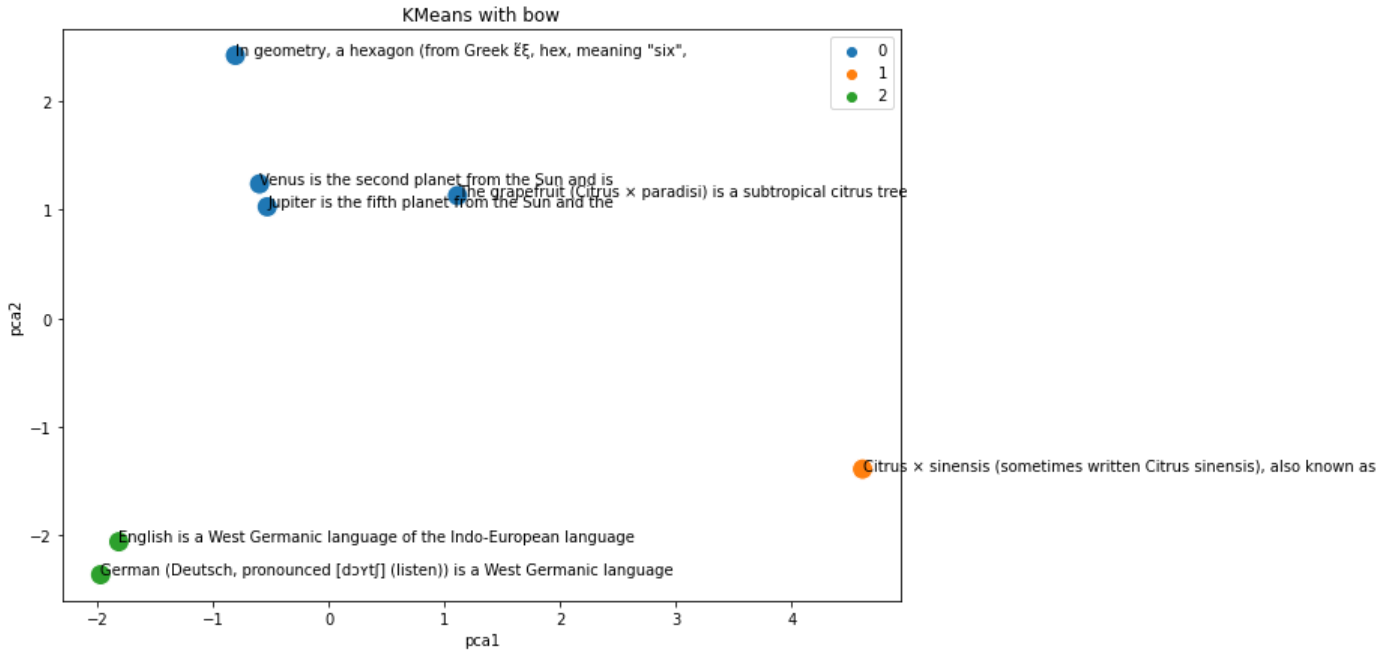
1. Call the function `plot_pca_clusters` to visualize the clusters created by the three models above:
   - KMeans with bag-of-words representation

- KMeans with sentence embedding representation.
- DBSCAN with sentence embedding representation.

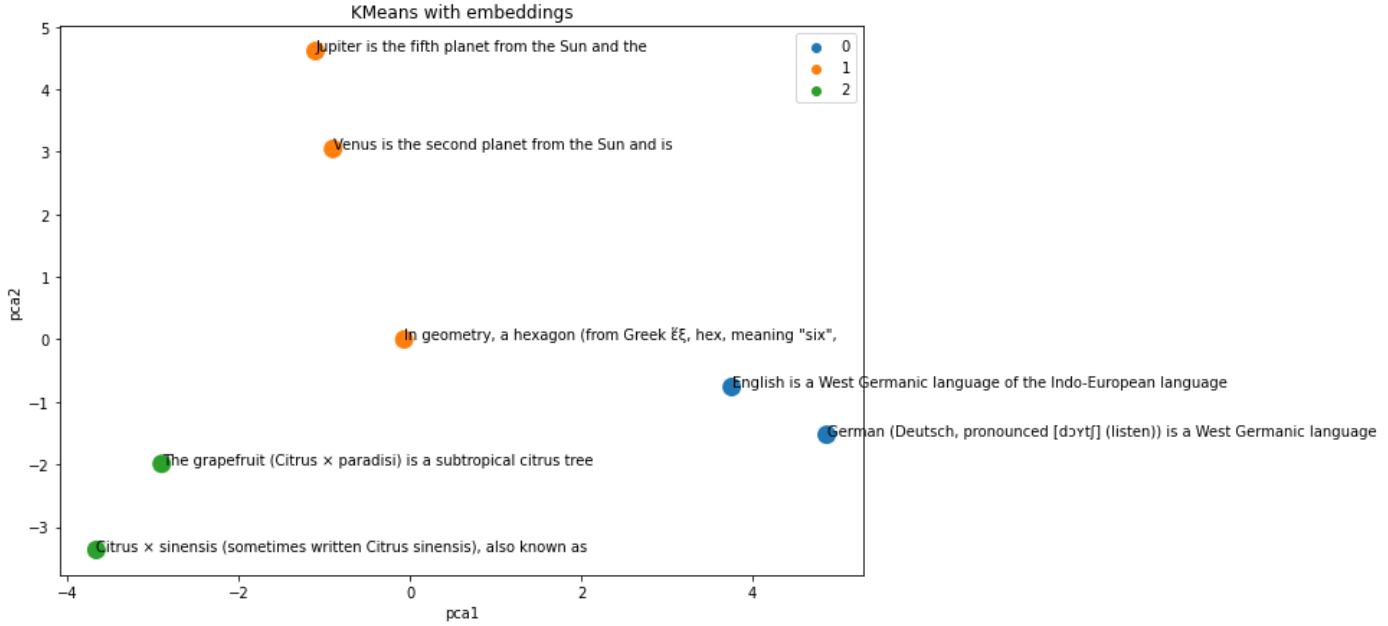
In [10]:
```
from sklearn.decomposition import PCA # Obtain the principal components

def plot_pca_clusters(
    data,
    cluster_labels,
    raw_sents=wiki_df["text"],
    show_labels=False,
    size=100,
    title="PCA visualization",
):
    """
    Carry out dimensionality reduction using PCA 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
    title : str
        title for the visualization plot

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

    pca = PCA(n_components=2)
    principal_comp = pca.fit_transform(data)
    pca_df = pd.DataFrame(data=principal_comp, columns=["pca1", "pca2"])
    pca_df["cluster"] = cluster_labels

    plt.figure(figsize=(10, 7))
    plt.title(title)
    ax = sns.scatterplot(
        x="pca1", y="pca2", hue="cluster", data=pca_df, palette="tab10", s=size
    )

    x = pca_df["pca1"].tolist()
    y = pca_df["pca2"].tolist()
    if show_labels:
        for i, txt in enumerate(raw_sents):
            plt.annotate(" ".join(txt.split()[:10]), (x[i], y[i]))
        ax.legend(loc="upper right")

    plt.show()
```

BEGIN SOLUTION

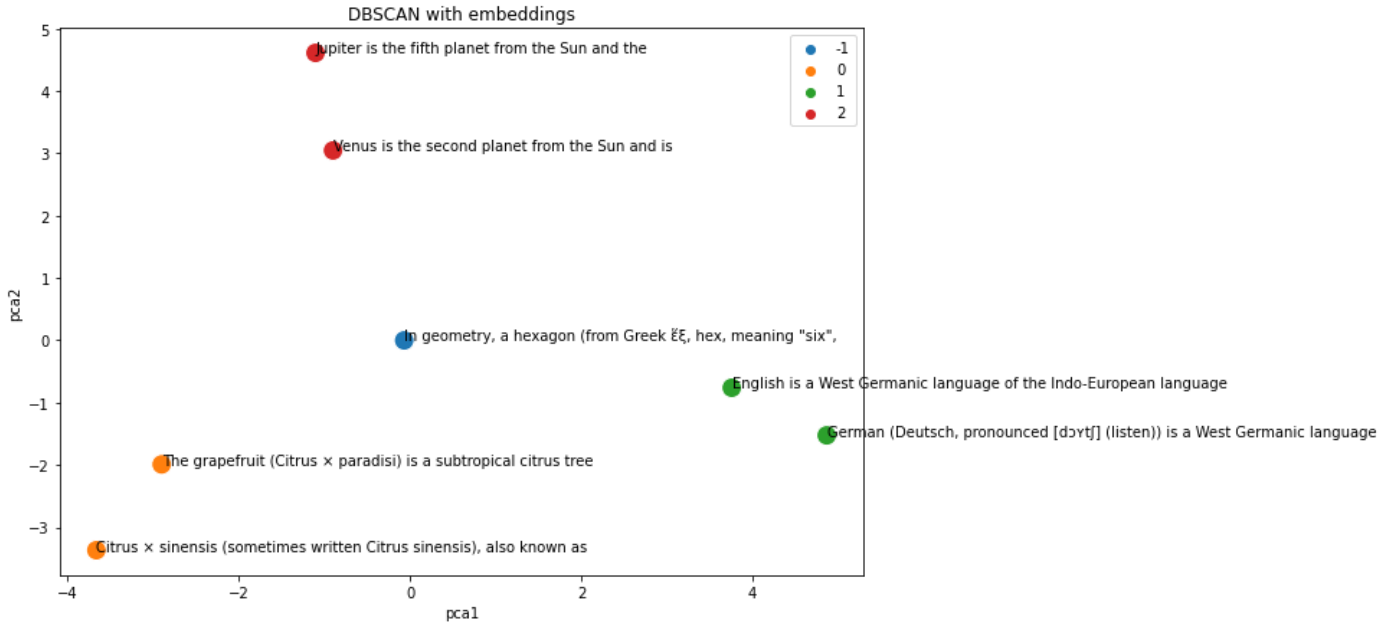
In [11]:
```
labels = kmeans_bow.labels_
plot_pca_clusters(
    bow_sents, labels, show_labels=True, size=200, title="KMeans with bow"
)
```



In [12]:
```
labels = kmeans_emb.labels_
plot_pca_clusters(
    emb_sents, labels, show_labels=True, size=200, title="KMeans with embeddings"
)
```



In [13]:
```
labels = dbscan_emb.labels_
plot_pca_clusters(
    emb_sents, labels, show_labels=True, size=200, title="DBSCAN with embeddings"
)
```



END SOLUTION

Exercise 2: Movie recommendations

Let's build simple movie recommendation systems using the MovieLens dataset. The original source of the data is here, and the structure of the data is described in the README that comes with it. The code below reads the data as a CSV assuming that it's under `ml-100k/` directory under your homework folder.

In [14]:
```
r_cols = ["user_id", "movie_id", "rating", "timestamp"]
ratings = pd.read_csv(
    os.path.join("ml-100k", "u.data"),
    sep="\t",
    names=r_cols,
    encoding="latin-1",
)
ratings.head()
```

Out[14]:

	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

In [15]:
```
# We'll be using these keys later in the starter code
user_key = "user_id"
item_key = "movie_id"
```

2.1 Terminology

rubric={points:3}

Here is some notation we will be using in this homework.

Constants:

- N: the number of users, indexed by n
- M: the number of movies, indexed by m
- $\mathcal{R}$: the set of indices (n, m) where we have ratings in the utility matrix Y
  - Thus $|\mathcal{R}|$ is the total number of ratings

The data:

- Y: the utility matrix containing ratings, with a lot of missing entries
- `train_mat` and `valid_mat`: Utility matrices for train and validation sets, respectively

Your tasks:

1. What are the values of N and M in movie ratings data?
2. What would be the shape of the dense utility matrix Y?
3. What would be the fraction of non missing ratings in the utility matrix Y?

In [16]:
```
N = None
M = None
```

BEGIN SOLUTION

In [17]:
```
N = len(np.unique(ratings[user_key]))
M = len(np.unique(ratings[item_key]))
print("Number of users (N)                : %d" % N)
print("Number of movies (M)               : %d" % M)
```

```
Number of users (N)                : 943
Number of movies (M)               : 1682
```

The shape of utility matrix is: $N \times M$. In this case it would be 943 by 1682.

In [18]:
```
print("Number of ratings (|R|)            : %d" % len(ratings))
print("Fraction of nonzero                : %.1f%%" % (len(ratings) / (N * M) * 100))
```

```
Number of ratings (|R|)            : 100000
Fraction of nonzero                : 6.3%
```

END SOLUTION

2.2 Splitting the data

rubric={points:5}

Your tasks:

1. Split the ratings data with `test_size=0.2` and `random_state=42`.

BEGIN SOLUTION

In [19]:
```
from sklearn.model_selection import train_test_split

ratings = ratings.drop(columns="timestamp")
X = ratings.copy()
y = ratings["user_id"]
X_train, X_valid, y_train, y_valid = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

END SOLUTION

2.3 Utility matrix

rubric={points:10}

Your tasks

1. Create utility matrices for train and validation sets.
2. Briefly explain the difference between the train and validation utility matrices.

> You may use the code from lecture notes with appropriate attributions.
>
> You won't do it in real life but since our dataset is not that big, create a dense utility matrix in this assignment. You are welcome to try sparse matrix but then you may have to change some started code provided in the later exercises.

In [20]:
```
user_mapper = dict(zip(np.unique(ratings[user_key]), list(range(N))))
item_mapper = dict(zip(np.unique(ratings[item_key]), list(range(M))))
user_inverse_mapper = dict(zip(list(range(N)), np.unique(ratings[user_key])))
item_inverse_mapper = dict(zip(list(range(M)), np.unique(ratings[item_key])))
```

BEGIN SOLUTION

In [21]:
```
def create_Y_from_ratings(data, N, M):
    Y = np.zeros((N, M))
    Y.fill(np.nan)
    for index, val in data.iterrows():
        n = user_mapper[val[user_key]]
        m = item_mapper[val[item_key]]
        Y[n, m] = val["rating"]

    return Y
```

END SOLUTION

In [22]:
```
train_mat = None
```

```
valid_mat = None
```

BEGIN SOLUTION

In [23]:
```
train_mat = create_Y_from_ratings(X_train, N, M)
valid_mat = create_Y_from_ratings(X_valid, N, M)

print("train_mat shape: ", train_mat.shape)
print("valid_mat shape: ", valid_mat.shape)
print("Number of non-nan elements in train_mat: ", np.sum(~np.isnan(train_mat)))
print("Number of non-nan elements in valid_mat: ", np.sum(~np.isnan(valid_mat)))
```

```
train_mat shape:  (943, 1682)
valid_mat shape:  (943, 1682)
Number of non-nan elements in train_mat:  80000
Number of non-nan elements in valid_mat:  20000
```

Both the training utility matrix `train_mat` and validation utility matrix `valid_mat` are of shape N by M but `train_mat` only has the ratings from `X_train`, whereas `valid_mat` only has the ratings from `X_valid` and all other ratings missing.

END SOLUTION

2.4 Evaluation and baseline

rubric={points:4}

To compare different models you build in this homework, let's write a couple of functions for evaluation.

- The `error` function returns RMSE.
- The `evaluate` function prints the train and validation RMSEs.

Your task:

1. Briefly explain what exactly we are comparing to evaluate recommender systems.
2. Implement the global average baseline, where you predict everything as the global average rating. What's the RMSE of the global average baseline?

In [24]:
```
def error(Y1, Y2):
    """
    Returns the root mean squared error (RMSE).
    """
    return np.sqrt(np.nanmean((Y1 - Y2) ** 2))


def evaluate(pred_Y, train_mat, valid_mat, model_name="Global average"):
    print("%s train RMSE: %0.2f" % (model_name, error(pred_Y, train_mat)))
    print("%s valid RMSE: %0.2f" % (model_name, error(pred_Y, valid_mat)))
```

BEGIN SOLUTION

Once we have train and validation utility matrices, we predict ratings in the utility matrix. Once we have predictions, here is our plan for evaluation:

- Compare predictions with the non-NaN ratings in the train set to get train root mean squared error (RMSE)
- Compare predictions with the non-NaN ratings in the validation set to get validation RMSE.

RMSE is a standard way to benchmark recommender systems. But it doesn't necessarily measure what we want in recommender systems. First of all, there is no ground truth in the context of recommender systems. The overall goal is to find items that the user is actually interested in, and along with the rating the user would give to the item, there are many other considerations such as diversity, freshness, and trust, which are not really captured by RMSE.

In [25]:
```
avg = np.nanmean(train_mat)
pred_g = np.zeros(train_mat.shape) + avg
pd.DataFrame(pred_g).head()
```

Out[25]:

	0	1	2	3	4	5	6	7	8	9	...	1672
0	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	...	3.531262
1	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	...	3.531262
2	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	...	3.531262
3	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	...	3.531262
4	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	3.531262	...	3.531262

5 rows × 1682 columns

In [26]:
```
evaluate(pred_g, train_mat, valid_mat, model_name="Global average")
```

```
Global average train RMSE: 1.13
Global average valid RMSE: 1.12
```

In [27]:
```
### END SOLUTION
```

(Optional) 2.5 k-nearest neighbours imputation

rubric={points:1}

Your tasks:

Try KNNImputer to fill in the missing entries. Discuss your observations.

BEGIN SOLUTION

In [28]:
```
from sklearn.impute import KNNImputer

num_neighs = [10, 15, 20, 40]
for n_neighbors in num_neighs:
    print("\nNumber of neighbours: ", n_neighbors)
```

```
    imputer = KNNImputer(n_neighbors=n_neighbors)
    pred_knn = imputer.fit_transform(train_mat)
    keep_cols = ~np.isnan(train_mat).all(axis=0)
    print(keep_cols.shape)
    evaluate(pred_knn,
             train_mat[:, keep_cols],
             valid_mat[:, keep_cols],
             model_name=f"k-nearest neighbours imputation, k={n_neighbors}")
```

```
Number of neighbours:  10
(1682,)
k-nearest neighbours imputation, k=10 train RMSE: 0.00
k-nearest neighbours imputation, k=10 valid RMSE: 0.98

Number of neighbours:  15
(1682,)
k-nearest neighbours imputation, k=15 train RMSE: 0.00
k-nearest neighbours imputation, k=15 valid RMSE: 0.97

Number of neighbours:  20
(1682,)
k-nearest neighbours imputation, k=20 train RMSE: 0.00
k-nearest neighbours imputation, k=20 valid RMSE: 0.97

Number of neighbours:  40
(1682,)
k-nearest neighbours imputation, k=40 train RMSE: 0.00
k-nearest neighbours imputation, k=40 valid RMSE: 0.98
```

The training RMSE is zero here because KNN imputer only imputes missing values. With `n_neighbors = 15 or 20`, we are getting the best validation RMSE so far although the difference between average of per-user and per-movie baseline and this one is not much.

Also, in order for the imputer to word, we had to remove the columns where all entries were NaN because KNN imputer is not able to find an average of ratings for such columns. We could impute ratings of such columns with the global average similar to how we did it with other baselines.

END SOLUTION

2.6 Use collaborative filtering with the `surprise` package

rubric={points:6}

Use the `surprise` package which has implementation of SVD algorithm for collaborative filtering. You can install it as follows in your conda environment.

`conda install -n cpsc330 -c conda-forge scikit-surprise`

Your tasks:

1. Carry out cross-validation using SVD algorithm in the package, similar to how we did it in the lecture on Jester dataset. Report mean RMSE and compare it with global baseline.

BEGIN SOLUTION

In [29]:
```
from surprise import SVD, Dataset, KNNBasic, Reader
from surprise.model_selection import cross_validate

reader = Reader()
data = Dataset.load_from_df(ratings, reader)
trainset = data.build_full_trainset()
algo = SVD()
res = cross_validate(algo, data, measures=["RMSE"], cv=5, verbose=True)
pd.DataFrame(res)
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9366  0.9399  0.9344  0.9386  0.9348  0.9368  0.0021
Fit time          4.03    4.10    4.51    7.76    7.61    5.60    1.71
Test time         0.10    0.10    0.31    0.20    0.22    0.19    0.08
```

Out[29]:

	test_rmse	fit_time	test_time
0	0.936620	4.034111	0.098305
1	0.939868	4.096569	0.104959
2	0.934352	4.514900	0.311502
3	0.938609	7.764040	0.202858
4	0.934759	7.605111	0.219171

In [30]:
```
pd.DataFrame(res).mean()
```

Out[30]:
```
test_rmse    0.936842
fit_time     5.602946
test_time    0.187359
dtype: float64
```

END SOLUTION

Exercise 3: Short answer questions

---

rubric={points:5}

Answer the following short-answer questions:

1. What's the main difference between unsupervised and supervised learning?
2. When choosing k in k-means, why not just choose the k that leads to the smallest inertia (sum of squared distances within clusters)?
3. You decide to use clustering for *outlier detection*; that is, to detect instances that are very atypical compared to all the rest. How might you do this with k-means?
4. You decide to use clustering for *outlier detection*; that is, to detect instances that are very atypical compared to all the rest. How might you do this with DBSCAN?

5. How might you apply clustering to recommendation systems?

BEGIN SOLUTION

1. Supervised has target values (y), unsupervised does not.
2. Because inertia decreases with k, so you'd just choose $k = n$, which is not interesting.
3. Look for examples that are very far away from their cluster mean.
4. Look for examples that were not assigned to any cluster.
5. If you cluster items, you could recommend other items in the same cluster when someone looks at a particular item. You could also try clustering users and recommending items to a user based on users in the same cluster.

END SOLUTION

Submission instructions

PLEASE READ: 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.