Introduction

Welcome to lab #6. This week, we'll practice working with *Vector Space Models* (VSMs), by applying them to the domain of recommender engines. Rather than implementing the methods discussed in the lecture from scratch, we will work with *scikit-learn*, a popular Python toolkit for machine learning.

Follow-up Lab #5

Here are two example solutions for the queries from the Wikidata Task #1.1:

1. All member countries of the EU, with English labels:

```
SELECT ?countryLabel
WHERE
{
    ?country wdt:P463 wd:Q458 .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

2. Same as above, but now printed with their capital on a map view:

```
#defaultView:Map
SELECT ?countryLabel ?coord ?capital ?capitalLabel
WHERE
{
    ?country wdt:P463 wd:Q458 .
    ?country wdt:P36 ?capital .
    ?capital wdt:P625 ?coord .
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

Confused about the labels above, which look like they are unbound in the query (like the `countryLabel`)? This is done through the *automatic mode* of Wikidata's Label service. For more details on querying Wikidata, check out the online resources, starting from Wikidata Query Help.

Task #1: Cosine Similarity

Let's reproduce the results from the lecture worksheet. The movie data can be encoded (using NumPy) as:

```
import numpy as np
movies = np.array([
[4, 8, 6, 3, 0, 0],
```

```
[0, 5, 0, 8, 5, 0],
[1, 4, 0, 3, 0, 10]])
```
Now we can use the cosine similarity implementation from scikit-learn to compute the similarity matrix:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity_scores = cosine_similarity(movies)
```
Print out the similarity matrix and compare it to your worksheet results. You can continue computing the user/movie similarity matrix in the same style.

Task #2: A Movie Recommender Engine

So that works: Now let's try this with some real data. We'll use the MovieLens datasets; for development, it's recommended to start with the small version: Download the ml-latest-small.zip dataset and uncompress it. This dataset has 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

Now you have to load the `movies.csv` and `tags.csv` files into your program. We'll use pandas to import the data (here is a nice cheatsheet for Python data imports from DataCamp):

```
import pandas as pd
movies = pd.read_csv("ml-latest-small/movies.csv",header=0)
tags = pd.read_csv("ml-latest-small/tags.csv",header=0,na_filter=False)
```
If you want to print out some intermediate results, best prepare some small test versions of the data files with only a few movies in them.

You'll notice that each tag comes on its own line (with a timestamp). However, for creating feature vectors it's easier to first group all tags by movie:

```
groupedtags = tags.groupby(["movieId"])['tag'].apply(','.join).reset_index()
```
Now we can merge the tag data with the movie data:

```
movietags = pd.merge(movies, groupedtags, on='movieId', how='inner')
```
Now we have to create a *count* for each tag, similar to the matrix in Task #1 on the worksheet: Fortunately, *scikit-learn* has a helper function we can use, the CountVectorizer:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
count_matrix = cv.fit_transform(movietags['tag'])
```
Print out the `count_matrix` to check if everything looks right. Computing the similarity matrix works just like before:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity_scores = cosine_similarity(count_matrix)
```

Now you can start recommending movies. To access the dataframes, we'll use some helper functions to translate between movie titles and the index into the matrix:

```
def getMovieIdx(title):
    return movietags.loc[movietags['title'] == title].index[0]
def getMovieTitle(id):
    return movietags[movietags.index == id]["title"].values[0]
```

With this, we can find the index into the similarity matrix by title, like here:

```
movieId = getMovieIdx('Toy Story (1995)')
```

and find all similar movies in our matrix:

```
similar_movies = list(enumerate(similarity_scores[movieId]))
```

You still need to sort this list (in descending order), based on the similarity score:

```
sorted_similar_movies = sorted(similar_movies,key=lambda x:x[1],reverse=True)[1:]
```

Now you can print out the most similar movies that you would recommend for a movie, based on the tags that were given by other users. Even with the limited example dataset, this gives some workable results:

```
Top 5 similar movies to 'Toy Story (1995)' are:
1. Bug's Life, A (1998) (0.89)
2. Toy Story 2 (1999) (0.54)
3. Guardians of the Galaxy 2 (2017) (0.45)
4. Up (2009) (0.22)
5. The Lego Movie (2014) (0.16)
```

You could now extend this program to compute a user/movie similarity matrix and start giving personalized recommendations, as discussed in the lecture.

Bonus Task #2.1: An Intelligent Movie Agent

You can now build an intelligent movie agent, by combining your recommender code with the code from your smart university agent from the previous weeks: Note that the movie dataset also comes with a file `links.csv`, which provides externally resolvable ids. In particular, you can use the IMDBid to query Wikidata and find more information there (like Q171048 for "Toy Story"). Now you can retrieve lots of additional information, like the directors, awards, etc. and present it to the user.