Python programming and data analysis

Lecture 10

Introduction to Machine Learning 3

Robert Szmurło

e-mail: robert.szmurlo@ee.pw.edu.pl 2019L

Lecture outline

1. K-Means Clustering
2. Recommender systems

Next lecture

- Optimizing Python Code (working with sparse matrices)

K-Means Clustering

K Means Clustering is an unsupervised learning algorithm that will attempt to group similar clusters together in your data. So what does a typical clustering problem look like?

- Cluster Similar Documents
- Cluster Customers based on Features
- Market Segmentation
- Identify similar physical groups

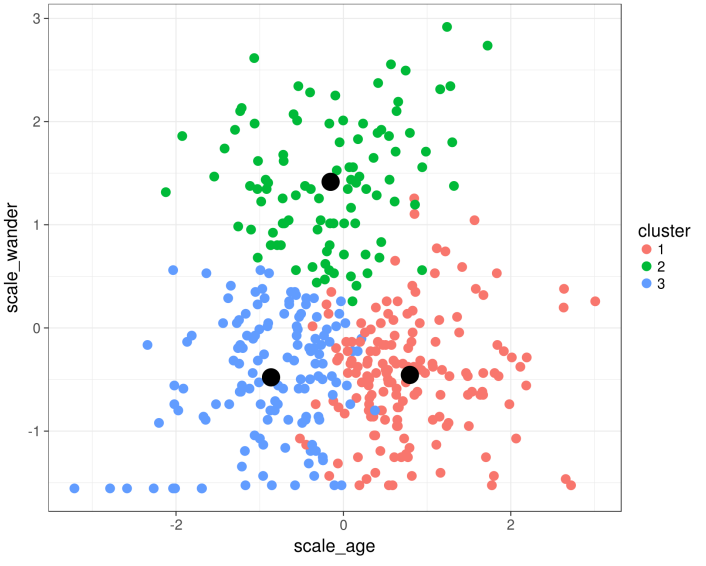
K Means Clustering

The overall goal is to divide data into distinct groups such that observations within each group are similar

Separate samples in n groups of equal variance, minimizing a criterion known as the **inertia** or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are. It suffers from various drawbacks:



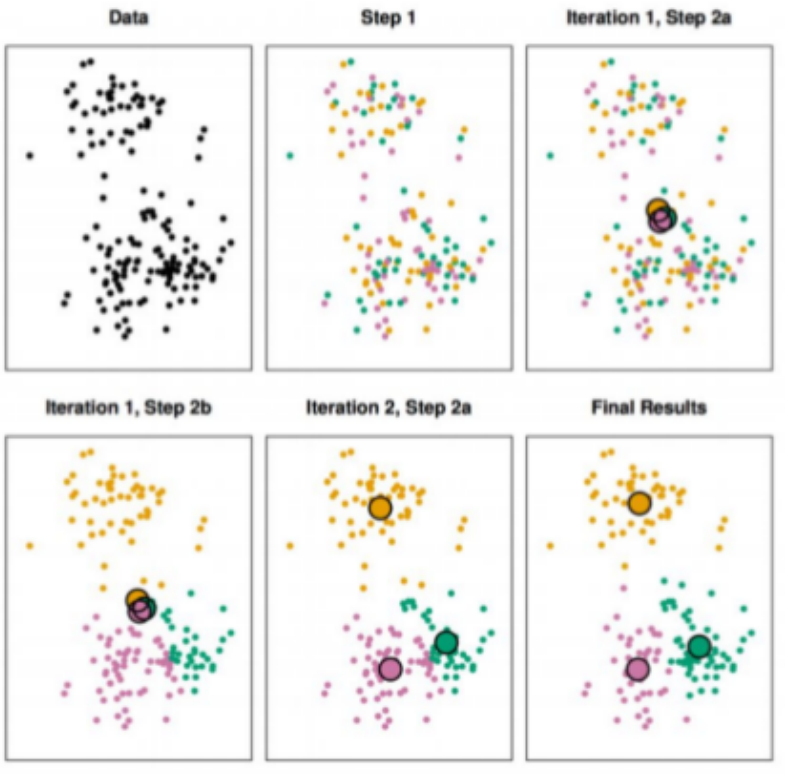
- Inertia makes the assumption that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters, or manifolds with irregular shapes.
- Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called “curse of dimensionality”). Running a dimensionality reduction algorithm such as PCA prior to k-means clustering can alleviate this problem and speed up the computations.

K-Means convergence

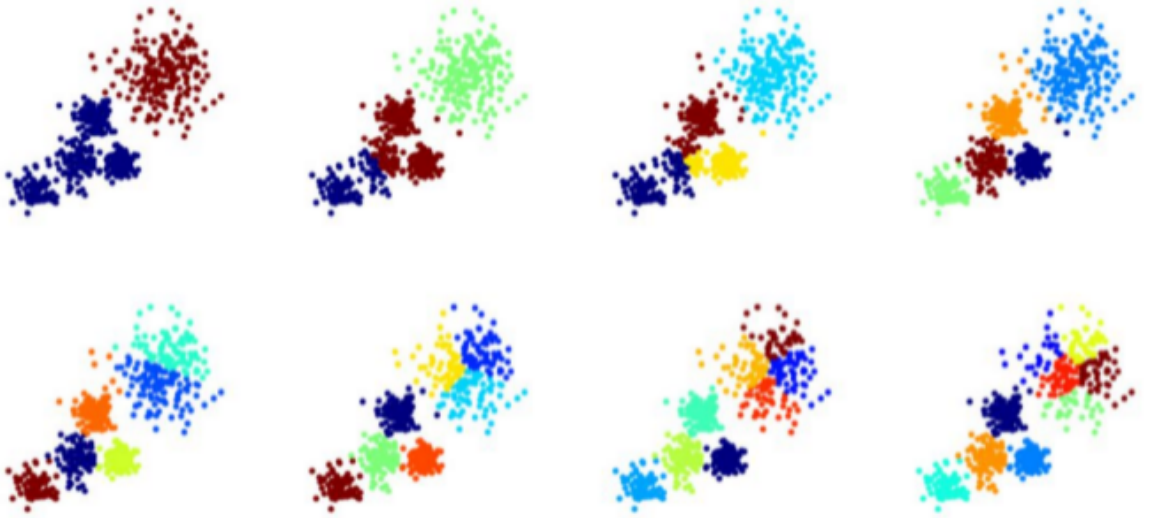
Given enough time, K-means will always converge, however this may be to a local minimum. This is highly dependent on the initialization of the centroids. As a result, the computation is often done several times, with different initializations of the centroids. One method to help address this issue is the k-means++ initialization scheme, which has been implemented in scikit-learn (use the init='k-means++' parameter). This initializes the centroids to be (generally) distant from each other, leading to provably better results than random initialization, as shown in the reference.

The K-Means algorithm

- Choose a number of Clusters “K”
- Randomly assign each point to a cluster
- Until clusters stop changing, repeat the following:
  - For each cluster, compute the cluster centroid by taking the mean vector of points in the cluster
  - Assign each data point to the cluster for which the centroid is the closest



Choosing a K Value

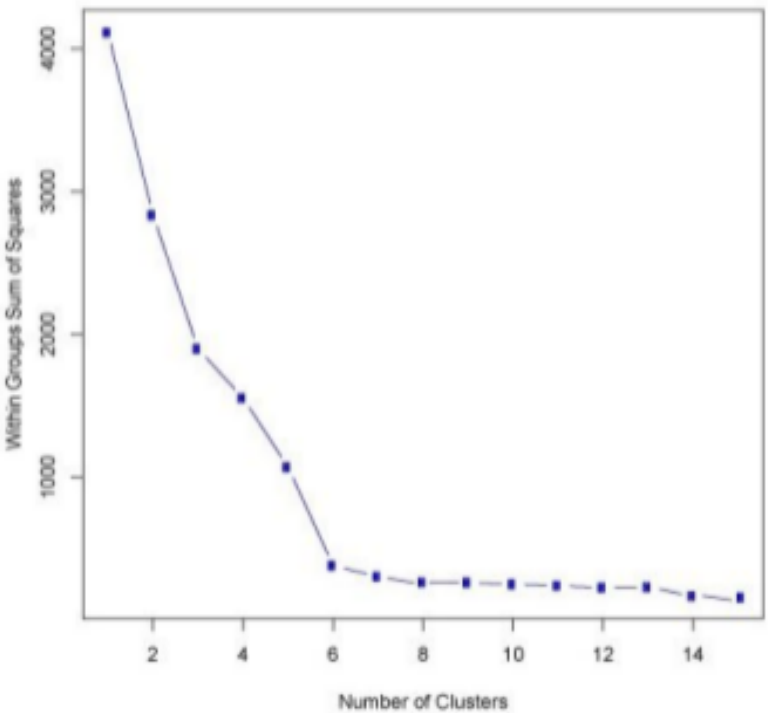


- There is no easy answer for choosing a “best” K value
- One way is the elbow method

First of all, compute the sum of squared error (SSE) for some values of k (for example 2, 4, 6, 8, etc.). The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid.

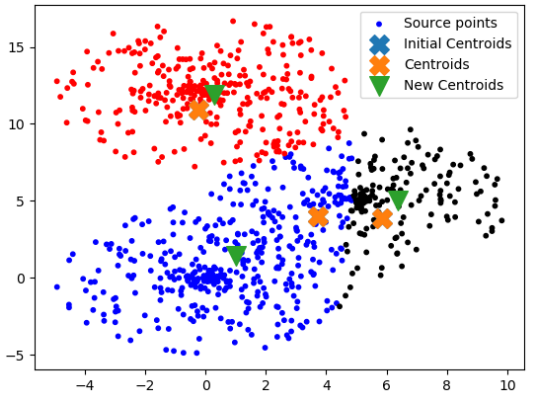
Choosing a K Value

If you plot k against the SSE, you will see that the error decreases as k gets larger; this is because when the number of clusters increases, they should be smaller, so distortion is also smaller. The idea of the elbow method is to choose the k at which the SSE decreases abruptly. This produces an "elbow effect" in the graph, as you can see in the following picture:



K-Means naive implementation

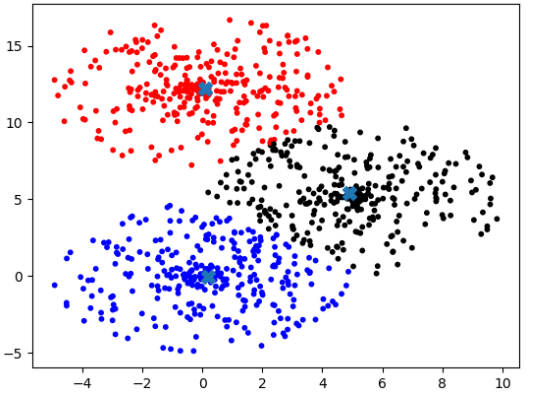
Initial clasification (randomly chosen centroids)



The training data blobs were initialized in:

```
blobs = np.array([  0, 0,
                    5, 5,
                    0, 12]).reshape(3,2)
```

Final classification of test data



Initial random centroids

```
[[ 2.64069926  2.34473848]
 [-2.80222435  9.59968344]
 [ 0.9015128  11.77231023]]
```

Centroids after training

```
[[ 0.09175458 12.2147556 ]
 [ 0.20403682 -0.02421538]
 [ 4.9077597   5.41107505]]
```

K-Means naive implementation Part 1

```
import numpy as np
import matplotlib.pyplot as plt
import math

# ...

def main():
    w = 5
    points = []
    blobs = np.array([  0,0,
                        5,5,
                        0, 12]).reshape(3,2)
    print(blobs)
    for i in range(3):
        for j in range(30):
            r = w * (np.random.random()-0.5)*2
            alpha = (np.random.random()-0.5)*2 * 2 * math.pi
            points.append([ blobs[i][0]+ r * math.sin(alpha), blobs[i][1]+r*math.cos(alpha), i ])
    #print(points)

    pointsz = list(zip(*points))

    x = np.array(pointsz[0:2]).T
    print(x)
    cents, classes = kmeans_naive(x, 3)
    colors = {0: 'red', 1: 'blue', 2: 'black', 3: 'yellow'}
    plt.scatter(pointsz[0], pointsz[1], s=10, marker="o", c = list(map(lambda i: colors[i], classes)),
                label="Source points")
    plt.scatter(cents[:,0], cents[:,1], s=100, marker="X", label="Centroids")

    plt.show()

if __name__ == "__main__":
    main()
```

K-Means naive implementation Part 2

```
def dist(p1,p2):
    return math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1]) ** 2)

def kmeans_naive(x,k):
    n = x.shape[0]
    initial = np.zeros((k, x.shape[1]))
    centroids = np.zeros((k, x.shape[1]))
    new_centroids = np.zeros((k, x.shape[1]))
    classes = [0] * n
    for i in range(k):
        ir = np.random.randint(0,n)
        centroids[i,:] = x[ir,:]
    initial[:] = centroids[:]

    diff = 10
    while (diff >0.1):
        classes_counts = np.zeros((k,))
        for i in range(n):
            cm = 0
            dm = dist(centroids[0,:], x[i,:])
            for c in range(1,k):
                d = dist(centroids[c,:], x[i,:])
                if (d <  dm):
                    dm  = d
                    cm = c
            classes[i] = cm
            classes_counts[cm] += 1
            new_centroids[cm,:] += x[i,:]

        for c in range(k):
            if (classes_counts[c] != 0):
                new_centroids[c,:] /= classes_counts[c]
            else:
                new_centroids[c,:] = centroids[c,:]
        print(new_centroids)
        diff = np.linalg.norm(new_centroids - centroids)
        print(f'diff={diff}')

        colors = {0: 'red', 1: 'blue', 2: 'black', 3: 'yellow'}
        plt.scatter(x[:,0], x[:,1], s=10, marker="o", c = list(map(lambda i: colors[i], classes)), label="Source points")
        plt.scatter(initial[:,0], initial[:,1], s=200, marker="X", label="Initial Centroids")
        plt.scatter(centroids[:,0], centroids[:,1], s=200, marker="X", label="Centroids")
        plt.scatter(new_centroids[:,0], new_centroids[:,1], s=200, marker="v", label="New Centroids")
```

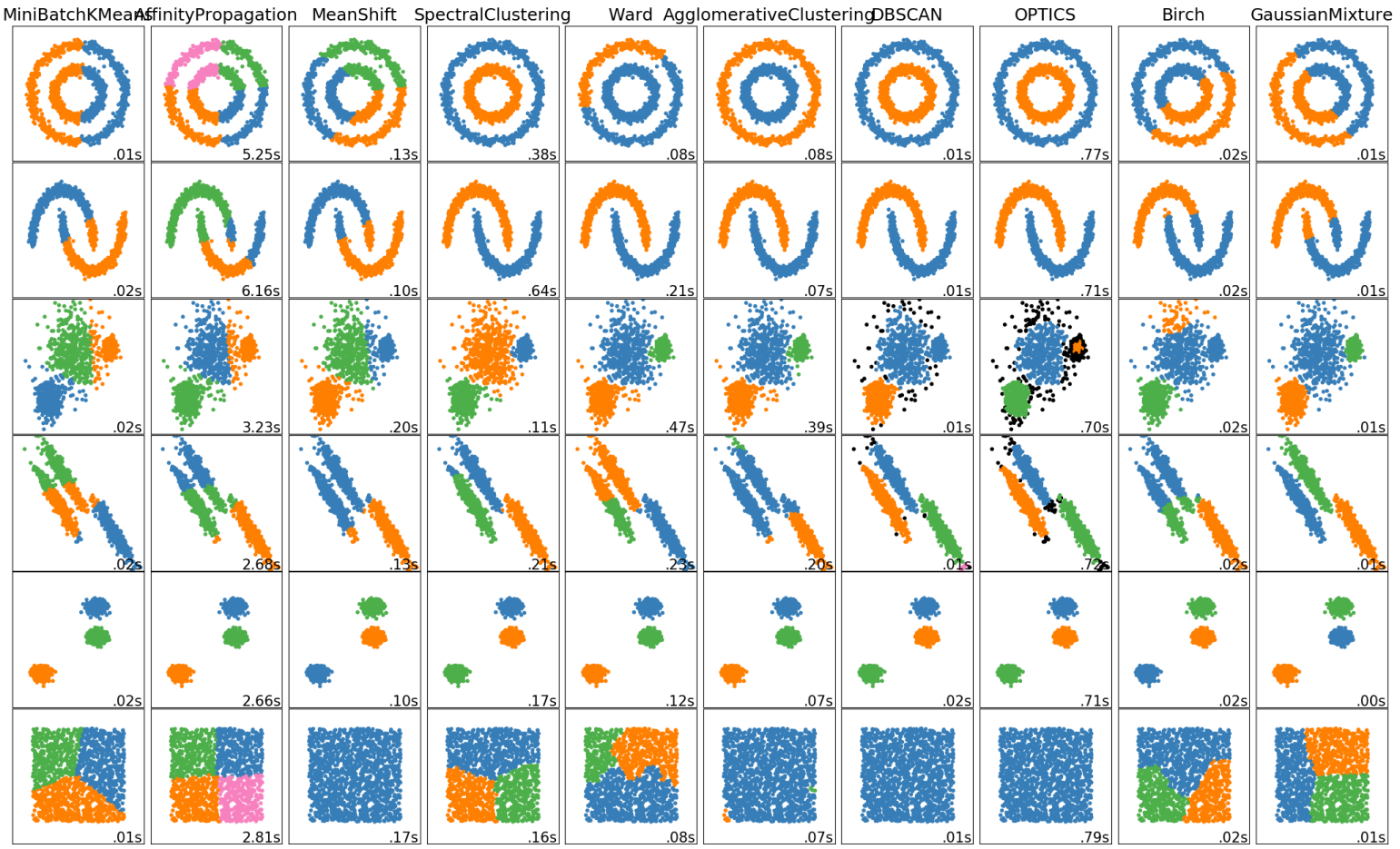
K-Means in SciKit library

```
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
# Create Data
# Returns: X : array of shape [n_samples, n_features]
data = make_blobs(n_samples=200, n_features=2,
                           centers=4, cluster_std=1.8,random_state=101)

plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(data[0])

kmeans.cluster_centers_

kmeans.labels_

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')
```

scikit learn

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

Overview of clustering methods



A comparison of the clustering algorithms in scikit-learn

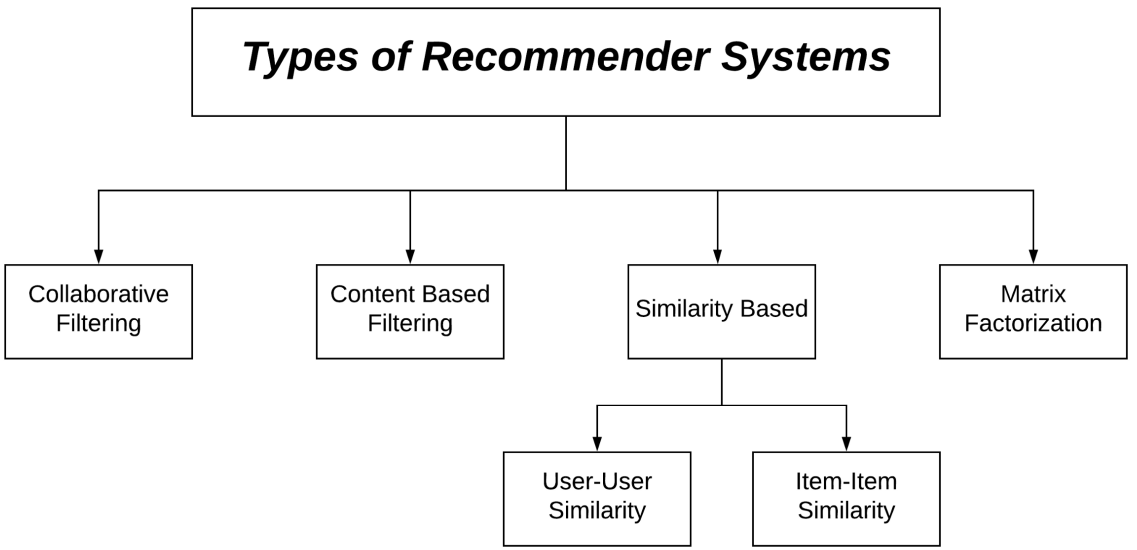
https://scikit-learn.org/stable/modules/clustering.html#clustering

Recommender Systems with Python

Recommender systems - methods

There are two most common types of recommender systems are **Content-Based** and **Collaborative Filtering (CF)**.

- Collaborative filtering produces recommendations based on the knowledge of users' attitude to items, that is it uses the "wisdom of the crowd" to recommend items.
  That is it uses the wisdom of the crowd to recommend items you can think of this as something like Amazon where based on other people's shopping experiences Amazon will suggest items that it believes you will enjoy.
- Content-based recommender systems focus on the attributes of the items and give you recommendations based on the similarity between them.

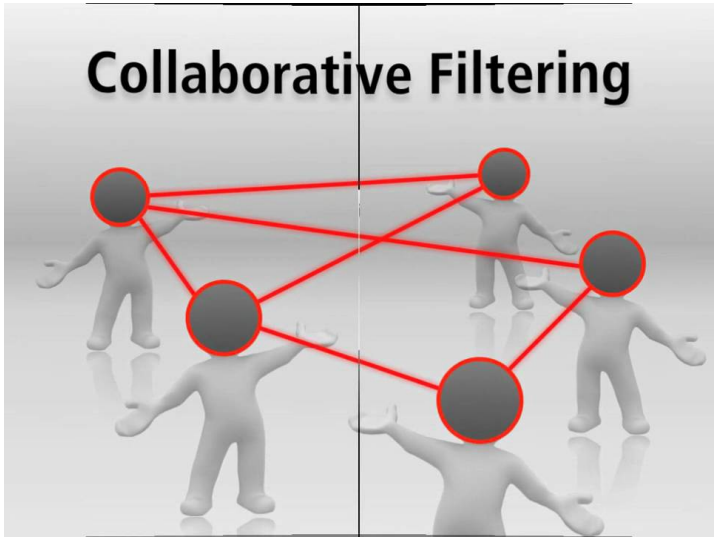


Collaborative Filtering

In general, Collaborative filtering (CF) is more commonly used than content-based systems because it usually gives better results and is relatively easy to understand (from an overall implementation perspective). The algorithm has the ability to do feature learning on its own, which means that it can start to learn for itself what features to use.

CF can be divided into **Memory-Based Collaborative Filtering** and **Model-Based Collaborative filtering**.

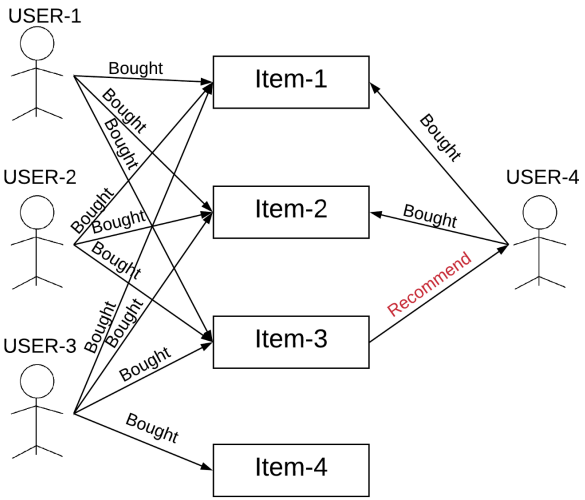
During this course, we will implement Model-Based CF by using singular value decomposition (SVD) and Memory-Based CF by computing cosine similarity.



Collaborative Filtering - explained

Core-idea/assumption here is that the users who have agreed in the past tend to also agree in the future. Here, all the three users namely USER-1, USER-2 and USER-3 agreed in the past that Item-3 is worth purchasing , hence in the future USER-4 may like Item-3 which is something USER-1, USER-2 and USER-3 agreed to purchase in the past.

If above assumption does not hold true then collaborative based filtering RS cannot be build.



Content Based Filtering

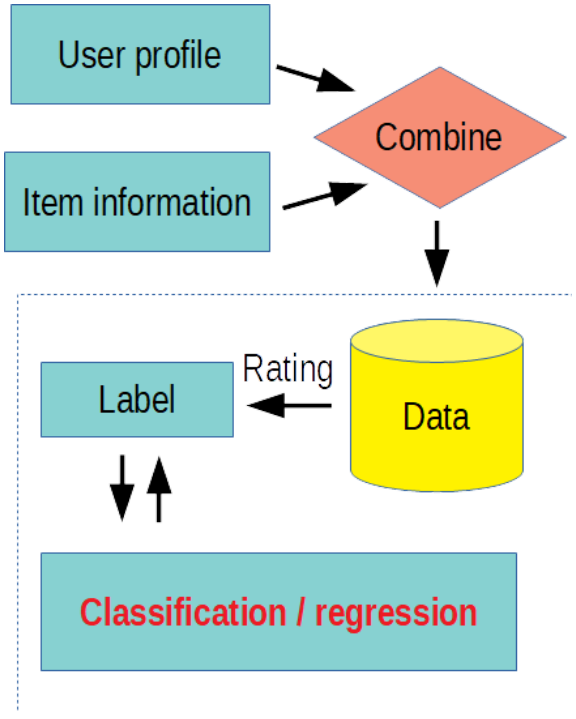
Content based filtering is similar in approach with classical machine learning techniques. We need to collect information about an item Ij and a user Ui then finally create features of both user Ui and Ij. Then we combine those features and feed them to a Machine Learning model for training. Here, label will be Aij, which is the corresponding rating given by a user Ui on item Ij.

Let say the item in our data-set is a movie. Now we can create its features like this:

- [Genre of Movie, Year of Release, Lead Actor, Director, Box Office Collection, Budget]

Similarly, features for user can be created as:

- [Likes and dislikes of users, Gender of a user, Age of user, Where user lives]

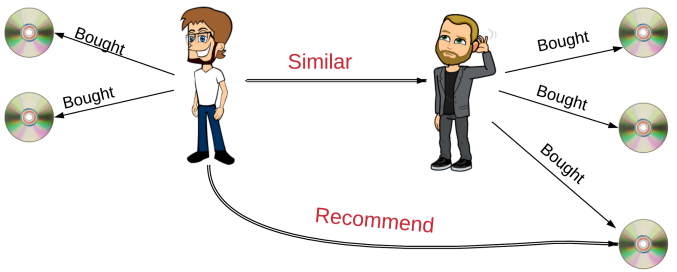


Now we can create an item vector which shall contain information about the item which is mentioned above. Then, we can similarly create a user vector which shall contain information about the user which is mentioned above. We can generate features for each user Ui and an item Ij. Finally we can combine these features and create a big data-set which can be suitable for feeding to Machine Learning model.

]

Similarity based

- USER-USER - similarity



- ITEM-ITEM - similarity

USER-USER SIMILARITY MATRIX

	U_1	U_2		U_j		U_{n-1}	U_n
U_1	1	Sim_{12}	...	Sim_{1j}	...		
U_2		1	...		...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
U_i		Sim_{i2}	...	Sim_{ij}	...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
U_{n-1}			...		...	1	
U_n			...		...		1

ITEM-ITEM SIMILARITY MATRIX

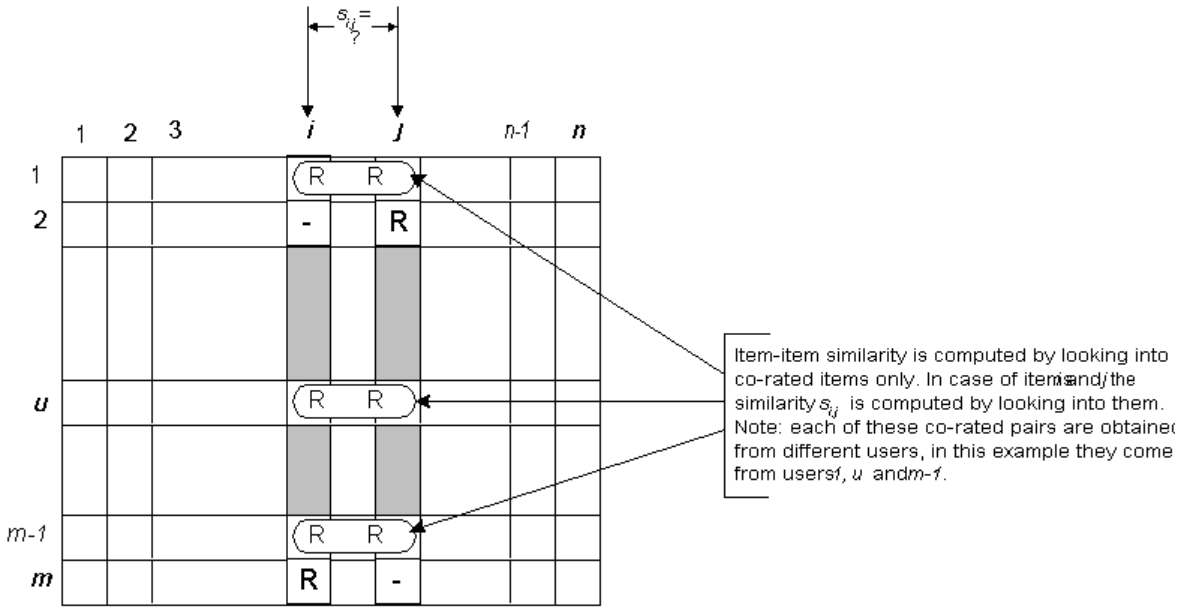
	I_1	I_2		I_j		I_{m-1}	I_m
I_1	1	Sim_{12}	...	Sim_{1j}	...		
I_2		1	...		...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
I_i		Simi2	...	Sim_{ij}	...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
I_{m-1}			...		...	1	
I_m			...		...		1

similarity measures

Isolation of the co-rated items and similarity computation

https://pdfs.semanticscholar.org/943a/e455fafc3d36ae4ce68f1a60ae4f85623e2a.pdf
http://www10.org/cdrom/papers/519/sdm2.html

- Cosine-based Similarity
- Correlation-based Similarity
- Adjusted Cosine Similarity

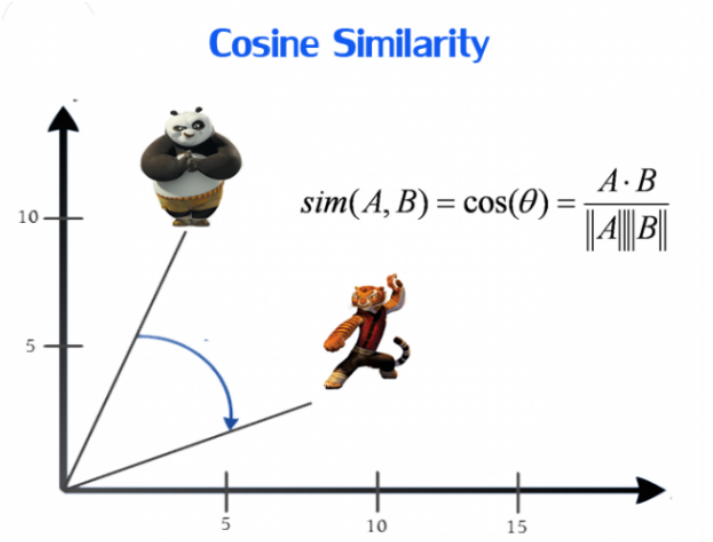


Cosine similarity

USER-USER or ITEM-ITEM similarity matrix]ces which will be square of size n*n. We can calculate similarity between two users using cosine similarity.

$$\cos(\theta) = \frac{A \cdot B}{||A||||B||} =$$

$$= \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2}\sqrt{\sum_i B_i^2}}$$



Offline

Rating matrix

	MI	ST	AV	BH	SS	LM
u_1	5	3		2	2	2
u_2	4	3	4		3	3
u_3	5	2	5	2	1	1
u_4	3	5	3		1	1
u_5	3	3	3	2	4	5
u_6	2	3	2	3	5	5

1. Calculate the similarity between all items.

Similarity

	MI	ST	AV	BH	SS	LM
MI	1	0.63	1	-0.21	-.088	-0.83
ST	0.63	1	0.35	-0.47	-0.64	-0.62
AV	1	0.35	1	0.01	-0.89	-0.83
BH	-0.21	-0.47	0.01	1	-0.23	-0.32
SS	-0.88	-0.64	-0.89	-0.23	1	0.96
LM	-0.83	-0.62	-0.83	-0.32	0.96	1

Correlation-based Similarity

Similarity between two items i and j is measured by computing the Pearson-r correlation $corr_{i,j}$. To make the correlation computation accurate we must first isolate the co-rated cases (i.e., cases where the users rated both i and j). Let the set of users who both rated i and j are denoted by U then the correlation similarity is given by

$$sim(i,j) = corr_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}}.$$

Here $R_{u,i}$ denotes the rating of user u on item i, $\bar{R}_i$ is the average rating of the i-th item.

Adjusted Cosine Similarity

One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the rows of the matrix but in case of the item-based CF the similarity is computed along the columns i.e., each pair in the co-rated set corresponds to a different user. Computing similarity using basic cosine measure in item-based case has one important drawback-the difference in rating scale between different users are not taken into account. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the similarity between items i and j using this scheme is given by

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}$$

Here $\bar{R_u}$ is the average of the u-th user's ratings.

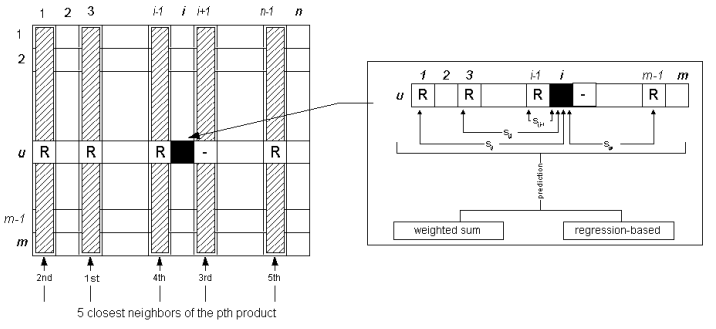
Prediction Computation

The most important step in a collaborative filtering system is to generate the output interface in terms of prediction. Once we isolate the set of most similar items based on the similarity measures, the next step is to look into the target users' ratings and use a technique to obtain predictions. Here we consider two such techniques.

Weighted Sum

As the name implies, this method computes the prediction on an item i for a user u by computing the sum of the ratings given by the user on the items similar to i. Each ratings is weighted by the corresponding similarity si,j between items i and j. Formally, using the notion shown in Figure 3 we can denote the prediction Pu,i as

$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$



Basically, this approach tries to capture how the active user rates the similar items. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range.

Prediction Computation - regression model

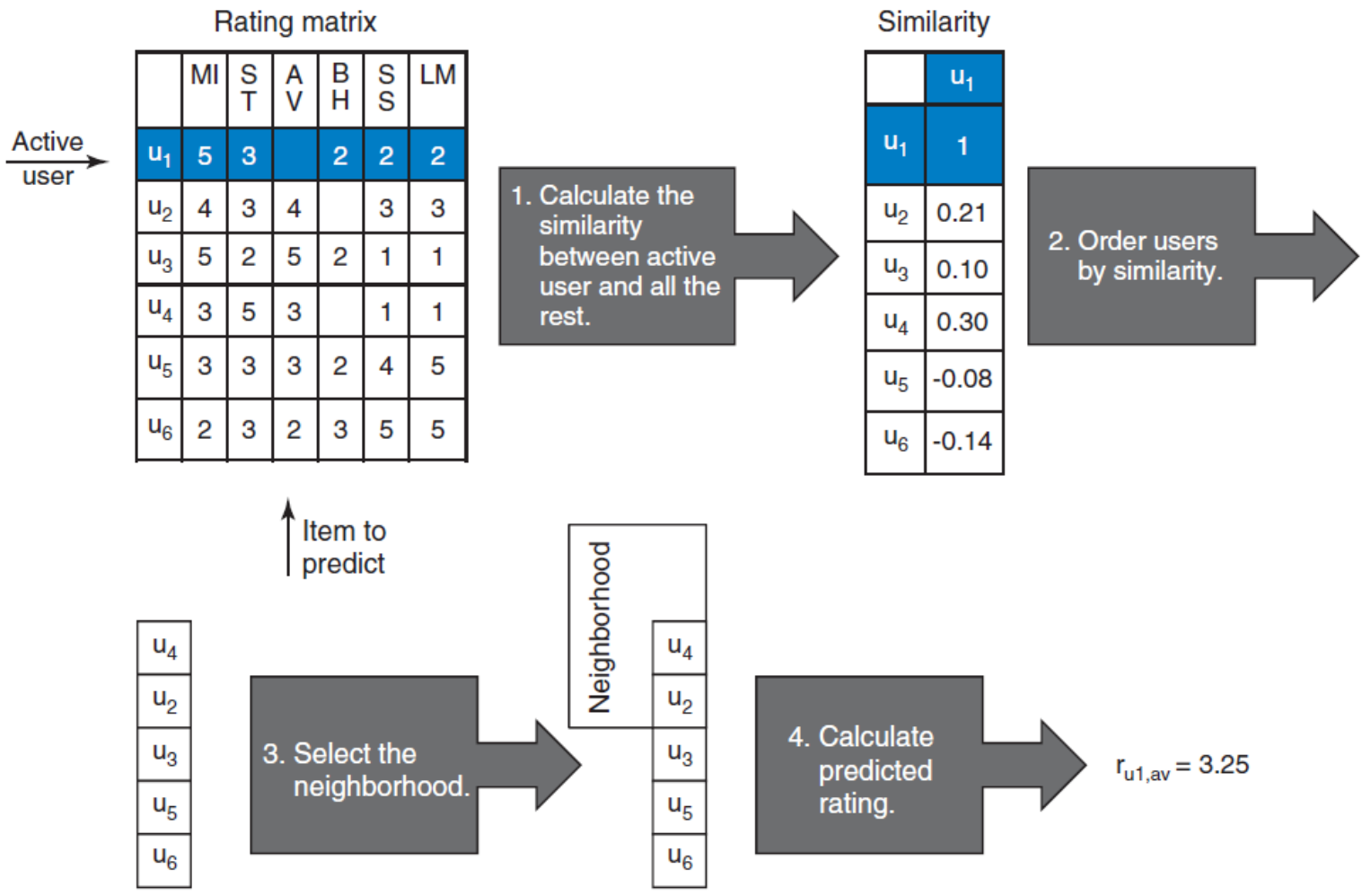
This approach is similar to the weighted sum method but instead of directly using the ratings of similar items it uses an approximation of the ratings based on regression model. In practice, the similarities computed using cosine or correlation measures may be misleading in the sense that two rating vectors may be distant (in Euclidean sense) yet may have very high similarity. In that case using the raw ratings of the *so called* similar item may result in poor prediction. The basic idea is to use the same formula as the weighted sum technique, but instead of using the similar item N's *raw* ratings values $R_{u,N}$'s, this model uses their approximated values $R_{u,N}$ based on a linear regression model. If we denote the respective vectors of the target item i and the similar item N by R_i and R_N the linear regression model can be expressed as

$$\bar{R'_N} = \alpha \bar{R}_i + \beta + \epsilon$$

The regression model parameters α and β are determined by going over both of the rating vectors. ϵ is the error of the regression model.

Pipeline for user-based neighborhood-based filtering

Kim Falk - Practical Recommender Systems



The Data

https://grouplens.org/

We will use famous MovieLens dataset, which is one of the most common datasets used when implementing and testing recommender engines. It contains 100k movie ratings from 943 users and a selection of 1682 movies.

You can download the dataset here.

This data set consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set. Detailed descriptions of the data file can be found at the end of this file.

Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering approaches can be divided into two main sections: user-item filtering and item-item filtering.

A user-item filtering will take a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked.

In contrast, item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

- Item-Item Collaborative Filtering: “Users who liked this item also liked …”
- User-Item Collaborative Filtering: “Users who are similar to you also liked …”

In both cases, you create a user-item matrix which built from the entire dataset.

Since we have split the data into testing and training we will need to create two [943 x 1682] matrices (all users by all movies). The training matrix contains 75% of the ratings and the testing matrix contains 25% of the ratings.

USER-ITEM Matrix

- In above USER-ITEM matrix, each row represents a user and each column represents an item and each cell represents rating given by a user to an item. A_{ij} is the rating given by a user Ui on item Ij. Aij can range from 1 to 5. Sometimes Aij can also be binary, if a matrix represents whether a user Ui has watched an item Ij or not. Here, Aij would be either 0 or 1.
- This USER-ITEM matrix is very sparse matrix, which means that many cells in this matrix are empty. Since, there are many items, and a single user cannot give rating to all of the items. These empty cells can be represented by "NaN" means not a number.
- Sparsity of Matrix = Number of Empty cells / Total Number of cells.
- Here, Sparsity of matrix = $(10^{10} — 5*10^{6}) / 10^{10} = 0.9995$
- It means that 99.95% of cells are empty. This is extreme sparsity actually.

USER-ITEM Matrix

	I_1	I_2		I_j		I_{m-1}	I_m
U_1			...		...		
U_2			...		...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
U_i			...	A_{ij}	...		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
U_{n-1}			...		...		
U_n			...		...		

Scenario for hands on part

- Import libraries and dataset

```
import numpy as np
import pandas as pd
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=column_names)
movie_titles = pd.read_csv("Movie_Id_Titles")
```

Resources

https://scikit-learn.org/stable/modules/clustering.html#k-means

Thank you