








Recommender System

What is a recommender system?

- They are systems or techniques that recommend or suggest a particular product, service, or entity.
- Can be classified into the following two categories, based on their approach to providing recommendations:
 - **The prediction problem**
 - Aims to predict missing values using all the information it has at its disposal (the ratings recorded, data on movies, data on users, and so on). If it is able to predict the missing values accurately, it will be able to give great recommendations
 - **A** matrix of m users and n items are given
 - Each row of the matrix represents a user and each column represents an item.
 - The value of the cell in the i^{th} row and the j^{th} column denotes the rating given by user i to item j . This value is usually denoted as r_{ij} .

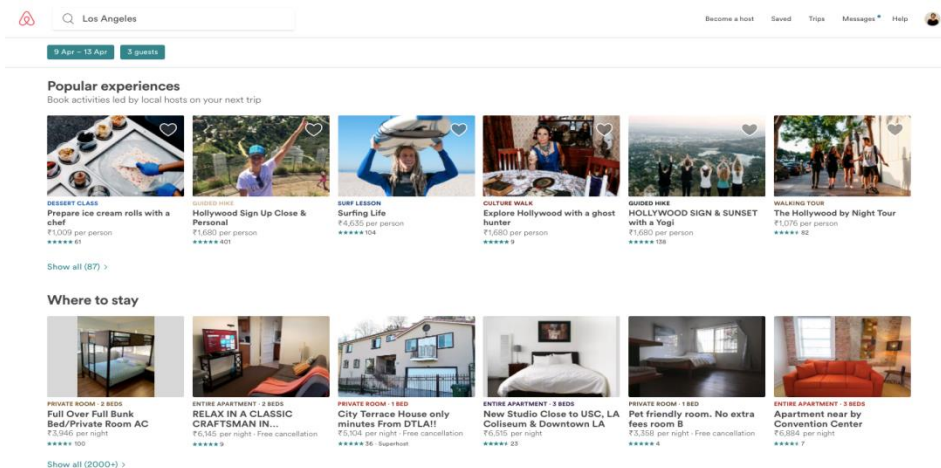
	i_1	i_2	i_3	i_4	i_5	i_6
 U1	4	?	3	?	5	?
 U2	?	2	?	?	4	1
 U3	?	?	1	?	2	5
 U4	?	?	3	?	?	1
 U5	1	4	?	?	2	5
 U6	5	?	2	1	?	4
 U7	?	2	3	?	4	5

- This matrix has seven users rating six items. Therefore, $m = 7$ and $n = 6$. User 1 has given the item 1 a rating of 4. Therefore, $r_{11} = 4$.

The ranking problem

- Ranking is the more intuitive formulation of the recommendation problem.
- Given a set of n items, the ranking problem tries to discern the top k items to recommend to a particular user, utilizing all of the information at its disposal

Example



- Imagine you are Airbnb, much like the preceding example. Your user has input the specific things they are looking for in their host and the space (such as their location, and budget). You want to display the top 10 results that satisfy those aforementioned conditions. This would be an example of the ranking problem

Types of recommender systems

- Collaborative filtering

- ☐ Collaborative filtering leverages the power of community to provide recommendations

- **User-based filtering**

- ☐ Find users that have bought and liked similar items in the past
they are more likely to buy similar items in the future too

- **Item-based filtering**

- ☐ Recommend items based on the past ratings of users

- ☐ Amazon makes good use of this model by recommending products to you based on your browsing and purchase history

- **Sorting and output**

- Sort our DataFrame on the basis of the score we just computed and output the list of top movies

Shortcomings

- Collaborative filters suffer from what we call the **cold start problem**
- Amazon is able to leverage collaborative filters so well because it has access to data concerning millions of purchases from millions of users.

Content-based systems

- Unlike collaborative filters, content-based systems do not require data relating to past activity
- Provides recommendations based on a user profile and metadata it has on particular item

Knowledge-based recommenders

- Used for items that are very rarely bought.
- It is simply impossible to recommend such items based on past purchasing activity or by building a user profile.
- Take real estate, for instance. Real estate is usually a once-in-a-lifetime purchase for a family. It is not possible to have a history of real estate purchases for existing users to leverage into a collaborative filter, nor is it always feasible to ask a user their real estate purchase history.

Quick Search

All ▾

Type of Play or Musical

Full Length Musical ▾

Genre

Dramatic Comedy ▾

Total Cast

Large C ▾

Females

Medium ▾

Males

Small C ▾

GO

[Use Advanced Search](#)

Manipulating Data with the Pandas Library

- The **Internet Movie Database (IMDB)** maintains a chart called the IMDB Top 250, which is a ranking of the top 250 movies according to a certain scoring metric.

The simple recommender

1. The first step in building our simple recommender is setting up our workspace.
2. Let's create a new directory named `Chapter3`.
3. Create a Jupyter Notebook in this directory named `Simple Recommender` and open it in the browser
4. Let's now load the dataset.

<https://www.kaggle.com/rounakbanik/the-movies-dataset/>

```
import pandas as pd
import numpy as np
```

```
#Load the dataset into a pandas dataframe
df = pd.read_csv('../data/movies_')
```

```
#Display the first five movies in the dataframe
df.head()
```

Upon running the cell, you should see a familiar table-like structure output in the notebook.

Building the simple recommender

- Choose a metric (or score) to rate the movies on
- Decide on the prerequisites for the movie to be featured on the chart
- Calculate the score for every movie that satisfies the conditions
- Output the list of movies in decreasing order of their scores

The metric

Numeric quantity based on which we rank movies

The choice of a metric is arbitrary

We are building an IMDB top 250 clone.

Therefore, we shall use IMDB's weighted rating formula as our metric. Mathematically, it can be represented as follows:

Weighted Rating (WR) =

$$\left(\frac{v}{v + m} \times R \right) + \left(\frac{m}{v + m} \times C \right)$$

The following apply:

- v is the number of votes garnered by the movie
- m is the minimum number of votes required for the movie to be in the chart (the prerequisite)
- R is the mean rating of the movie
- C is the mean rating of all the movies in the dataset

The prerequisites

- The IMDB weighted formula also has a variable m , which it requires to compute its score. This variable is in place to make sure that only movies that are above a certain threshold of popularity are considered for the rankings

The value of m :

#Calculate the number of votes garnered by the 80th percentile movie

```
m = df['vote_count'].quantile(0.80)
```

m

OUTPUT:

50.0

- Only 20% of the movies have gained more than 50 votes. Therefore, our value of m is 50.

- Another prerequisite is the runtime

- We will only consider movies that are greater than 45 minutes and less than 300 minutes in length

Let us define a new DataFrame, `q_movies`, which will hold all the movies that qualify to appear in the chart:

#Only consider movies longer than 45 minutes and shorter than 300 minutes

```
q_movies = df[(df['runtime'] >= 45) & (df['runtime'] <= 300)]
```

#Only consider movies that have garnered more than `m` votes

```
q_movies = q_movies[q_movies['vote_count'] >= m]
```

#Inspect the number of movies that made the cut

```
q_movies.shape
```

OUTPUT:

(8963, 24)

-

➤ We see that from our dataset of 45,000 movies approximately 9,000 movies (or 20%) made the cut.

Calculating the score

- The final value that we need to discover before we calculate our scores is C , the mean rating for all the movies in the dataset

```
# Calculate C
C = df['vote_average'].mean()
C
```

OUTPUT:

```
5.6182072151341851
```

- First, let us define a function that computes the rating for a movie, given its features and the values of m and C :

```
# Function to compute the IMDB weighted rating for each movie
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Compute the weighted score
    return (v/(v+m) * R) + (m/(m+v) * C)
```

The knowledge-based recommender

- Data that we have has information on the duration, genres, and timelines, but it isn't currently in a form that is directly usable.
- In other words, our data needs to be wrangled before it can be put to use to build this recommender

The `build_chart` function

- Get user input on their preferences
- Extract all movies that match the conditions set by the user
- Calculate the values of m and C for only these movies and proceed to build the chart as in the previous section
- The `build_chart` function will accept only two inputs: our `gen_df` DataFrame and the percentile used to calculate the value of m .

```
In [114]: #Generate the chart for top animation movies and display top 5.  
build_chart(gen_df).head()
```

```
Input preferred genre  
animation  
Input shortest duration  
30  
Input longest duration  
120  
Input earliest year  
1990  
Input latest year  
2005
```

Out[114]:

	title	runtime	vote_average	vote_count	year	genre	score
9698	Howl's Moving Castle	119.0	8.2	2049.0	2004	animation	7.994823
359	The Lion King	89.0	8.0	5520.0	1994	animation	7.926672
0	Toy Story	81.0	7.7	5415.0	1995	animation	7.637500
6232	Finding Nemo	100.0	7.6	6292.0	2003	animation	7.549423
546	The Nightmare Before Christmas	76.0	7.6	2135.0	1993	animation	7.460500

Building Content-Based Recommenders

- Simple recommender did not take into consideration an individual user's preferences.
- Knowledge-based recommender did take account of the user's preference for genres, timelines, and duration, but still generic
- Example:
- What sites like Netflix do. When you sign up on Netflix for the first time, it doesn't have any information about your tastes for it to build a profile, leverage the power of its community, and give you recommendation. Instead, what it does is ask you for a few movies you like and show you results that are most similar to those movies

Two types of content-based recommender

- **Plot description-based recommender:** Compares the descriptions and taglines of different movies, and provides recommendations that have the most similar plot descriptions.

- **Metadata-based recommender:** This model takes a host of features, such as genres, keywords, cast, and crew, into consideration and provides recommendations that are the most similar with respect to the aforementioned features

Document vectors

- Every document is depicted as a series of n numbers, where each number represents a dimension and n is the size of the vocabulary of all the documents put together.
- How do we numerically quantify the similarity between two bodies of text?
- Consider three movies: A, B, and C. How can we mathematically prove that the plot of A is more similar to the plot of B than to that of C (or vice versa)?
- But what are the values of these vectors?
- The two most popular vectorizers are CountVectorizer and TF-IDFVectorizer.

TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency) takes the aforementioned point into consideration and assigns weights to each word according to the following formula. For every word i in document j , the following applies:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

In this formula, the following is true:

- $w_{i,j}$ is the weight of word i in document j
- df_i is the number of documents that contain the term i

N is the total number of documents

The cosine similarity score

$$\text{cosine}(x, y) = \frac{x \cdot y^T}{||x|| \cdot ||y||}$$

- The cosine score can take any value between -1 and 1. The higher the cosine score, the more similar the documents are to each other.


























Plot description-based recommender

Getting Started with Data Mining Techniques

- In 2003, Linden, Smith, and York of Amazon.com published a paper entitled Item-to-Item Collaborative Filtering, which explained how product recommendations at Amazon work.



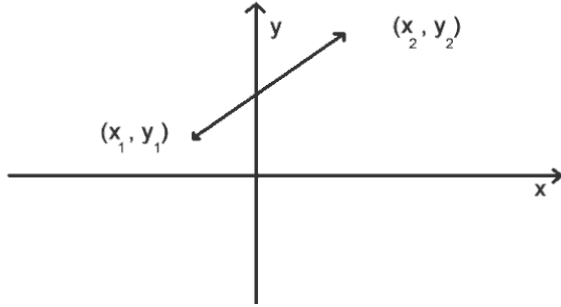
- Collaborative filters try to leverage the power of the community to give reliable, relevant content

- Collaborative filtering algorithms try to solve the prediction problem
- We are given a matrix of i users and j items. The value in the i th row and the j th column (denoted by r_{ij}) denotes the rating given by user i to item j

Euclidean distance

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



$$d(v_1, v_2) = \sqrt{\sum_{i=1}^n (q_i - r_i)^2}$$

- The Euclidean distance can be defined as the length of the line segment joining the two data points plotted on an n -dimensional Cartesian plane
- Euclidean scores can take any value between 0 and infinity.
- The lower the Euclidean score (or distance), the more similar the two vectors are to each other.

- Next, let's define three users who have rated five different movies:

```
#Define 3 users with ratings for 5 movies  
u1 = [5,1,2,4,5]  
u2 = [1,5,4,2,1]  
u3 = [5,2,2,4,4]
```

- From the ratings, we can see that users 1 and 2 have extremely different tastes, whereas the tastes of users 1 and 3 are largely similar. Let's see whether the Euclidean distance metric is able to capture this:

```
euclidean(u1, u2)
```

OUTPUT:

```
7.4833147735478827
```

- The Euclidean distance between users 1 and 2 comes out to be approximately 7.48:

```
euclidean(u1, u3)
```

OUTPUT:

```
1.4142135623730951
```

- Users 1 and 3 have a much smaller Euclidean score between them than users 1 and 2. Therefore, in this case, the Euclidean distance was able to satisfactorily capture the relationships between our users.

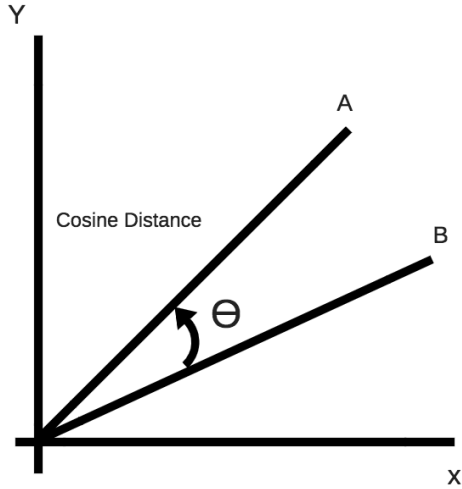
Pearson correlation

- Euclidean distances place emphasis on magnitude
- not able to gauge the degree of similarity or dissimilarity well.
- Pearson correlation is a score between -1 and 1, where -1 indicates total negative correlation and 1 indicates total positive correlation whereas 0 indicates that the two entities are in no way correlated with each other

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

Cosine similarity

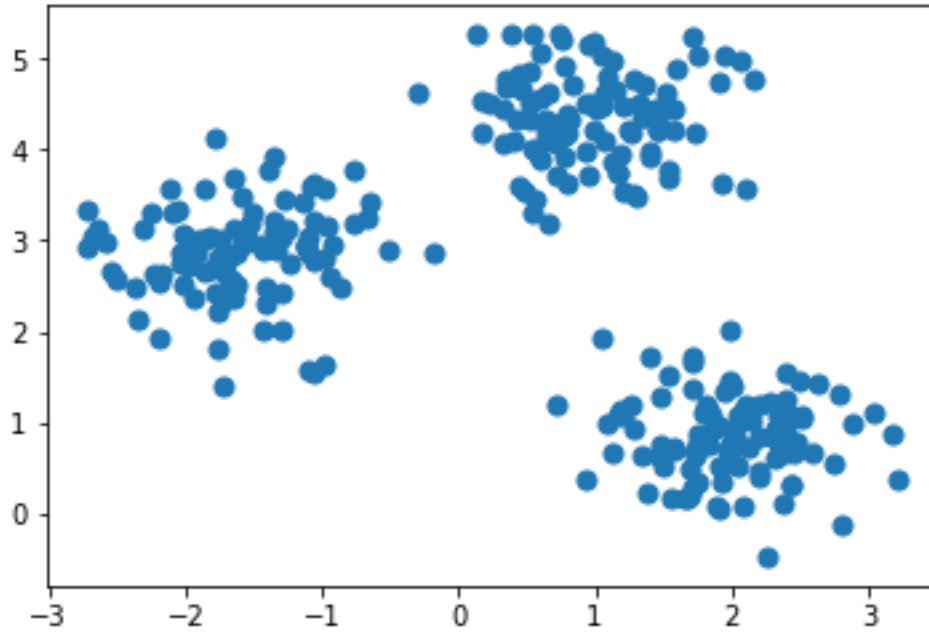
Cosine similarity is defined as follows:



$$\text{cosine}(x, y) = \frac{x \cdot y^T}{||x|| \cdot ||y||}$$

- Cosine similarity score computes the cosine of the angle between two vectors in an n -dimensional space
- When the cosine score is 1 (or angle is 0), the vectors are exactly similar
- Cosine score of -1 (or angle 180 degrees) denotes that the two vectors are exactly dissimilar to each other

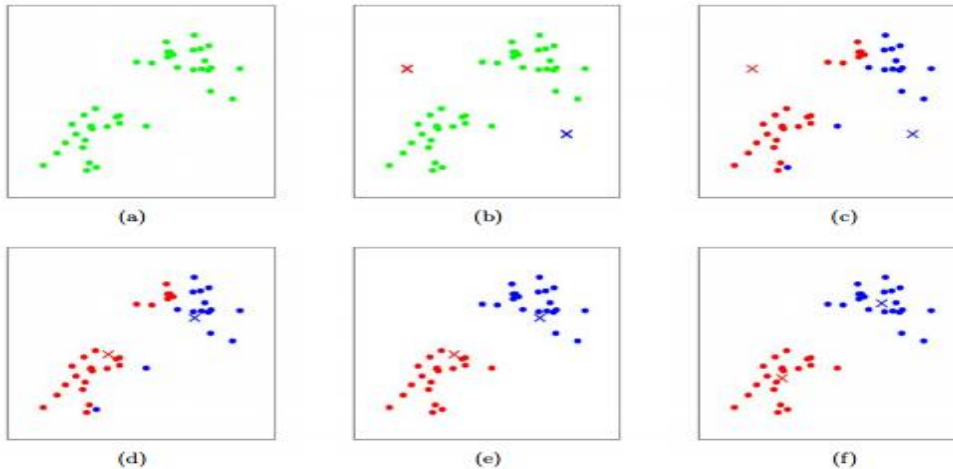
Clustering



- Clustering is one of the most popular techniques used in collaborative-filtering algorithms

k-means clustering

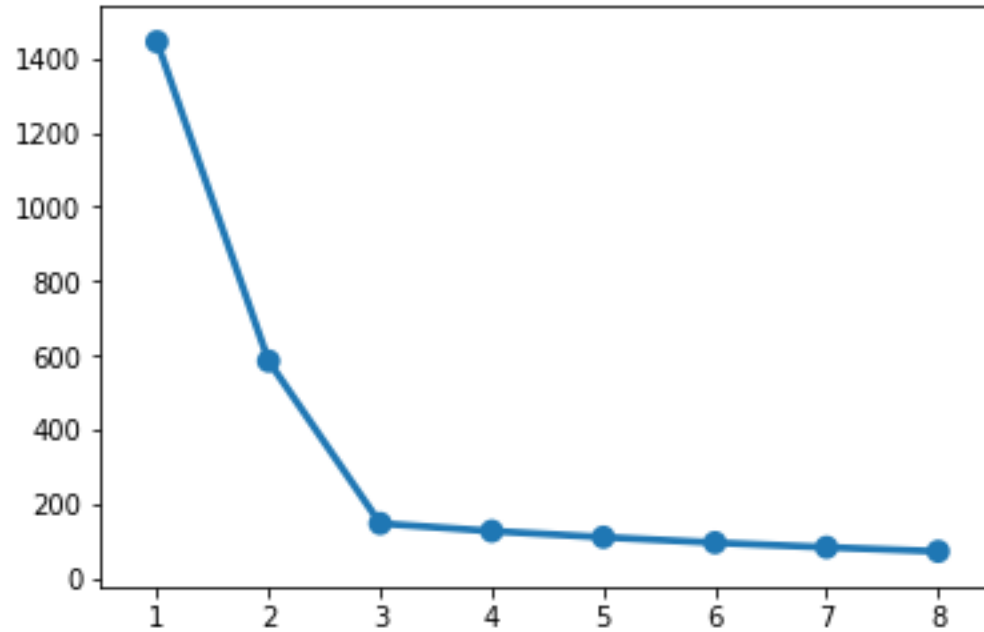
- Takes in the data points and the number of clusters (k) as input.
- Randomly plots k different points on the plane (called centroids)
- Assignment of points to the centroids: Every data point is assigned to the centroid that is the closest to it. The collection of data points assigned to a particular centroid is called a cluster. Therefore, the assignment of points to k centroids results in the formation of k clusters.
- Reassignment of centroids: In the next step, the centroid of every cluster is recomputed to be the center of the cluster (or the average of all the points in the cluster). All the data points are then reassigned to the new centroids



Choosing k

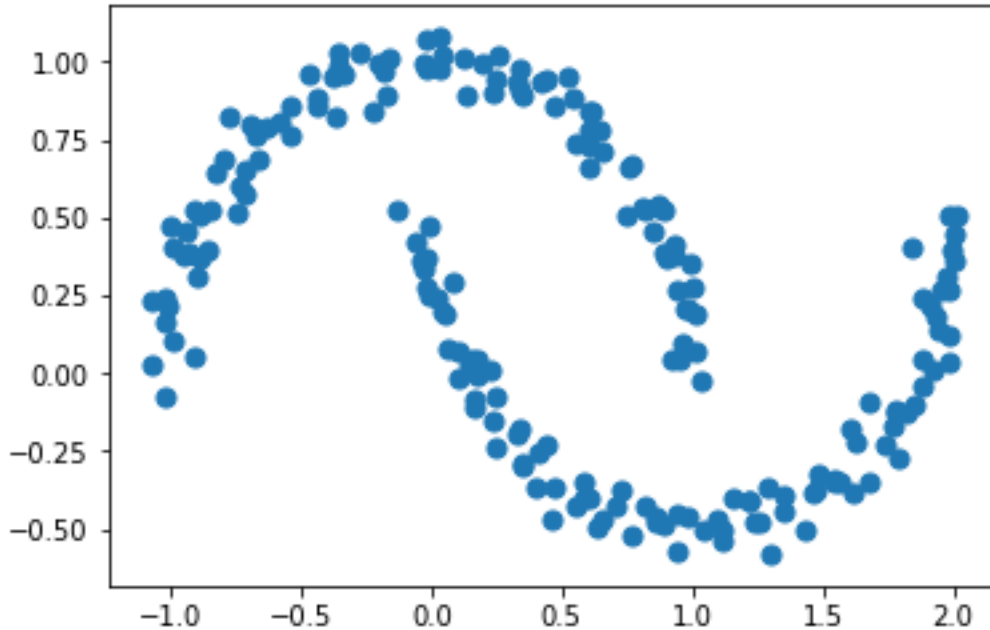
- Choosing a good value of k is vital to the success of the k-means clustering algorithm.
- Number of clusters can be anywhere between 1 and the total number of data points (where each point is assigned to its own cluster).
- Elbow method computes the sum of squares for each value of k and chooses the elbow point of the sum-of-squares v/s K plot as the best value for k. The elbow point is defined as the value of k at which the sum-of-squares value for every subsequent k starts decreasing much more slowly

$$SS = \sum_k \sum_{x_i \in C_k} (x_i - \mu_k)^2$$



- From the plot, it is clear that the Elbow is at $K=3$. From what we visualized earlier, we know that this is indeed the optimum number of clusters for this data

Other clustering algorithms



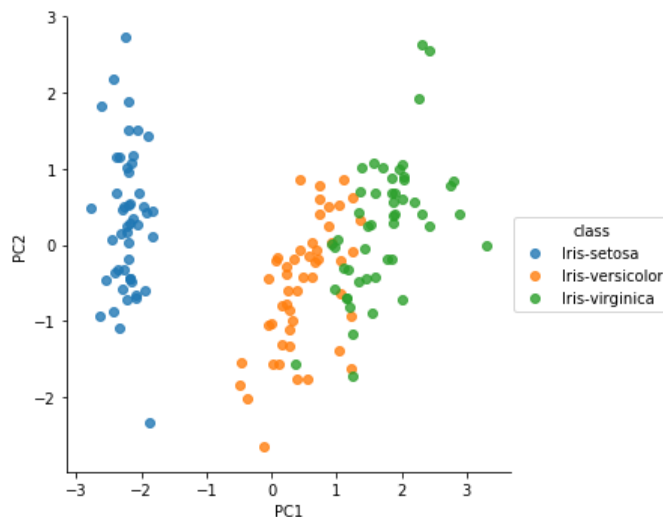
- k-means algorithm doesn't do a very good job of identifying the correct clusters like the one above.
- For clusters such as these half moons, another algorithm, called spectral clustering, with nearest-neighbor, affinity performs much better

Dimensionality reduction

- Most machine learning algorithms tend to perform poorly as the number of dimensions in the data increases. This phenomenon is often known as the curse of dimensionality. **Feature selection:** This method involves identifying the features that have the least predictive power and dropping them altogether.
- **Feature extraction:** Feature extraction takes in m -dimensional data and transforms it into an n -dimensional output space (usually where $m \gg n$), while retaining most of the information.
- Important feature-extraction method: **Principal component analysis** (or **PCA**).

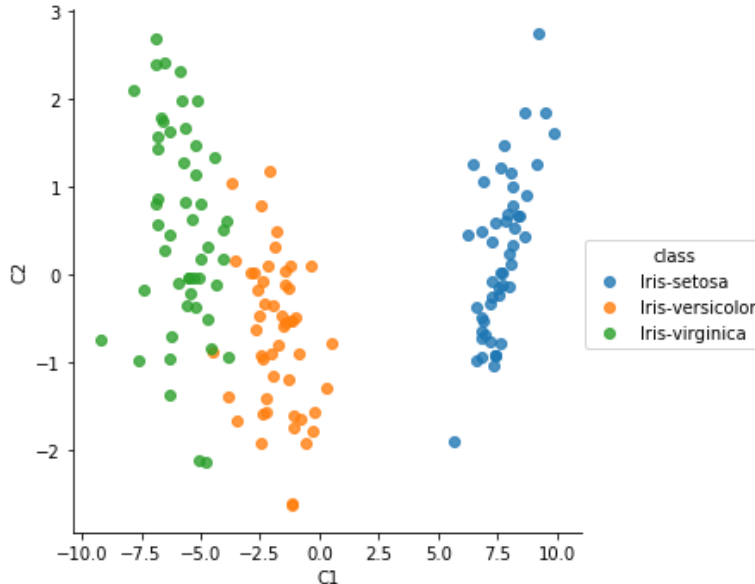
Principal component analysis

- **Principal component analysis** is an unsupervised feature extraction algorithm that takes in m -dimensional input to create a set of n ($m \gg n$) linearly uncorrelated variables (called principal components)
- The linear transformation in PCA is done in such a way that the first principal component holds the maximum variance (or information).
- considers those variables that are highly correlated to each other.
- Every principal component has more variance than every succeeding component and is orthogonal to the preceding com



Linear-discriminant analysis

- Like PCA, linear-discriminant analysis is a linear transformation method that aims to transform m -dimensional data into an n -dimensional output space.
- Unlike PCA, which tries to retain the maximum information, LDA aims to identify a set of n features that result in the maximum separation (or discrimination) of classes
- LDA requires labeled data in order to determine its components, it is a type of supervised learning algorithm.



Singular value decomposition

- Singular value decomposition, or SVD, is a type of matrix analysis technique that allows us to represent a high-dimensional matrix in a lower dimension
- SVD achieves this by identifying and removing the less important parts of the matrix and producing an approximation in the desired number of dimensions.

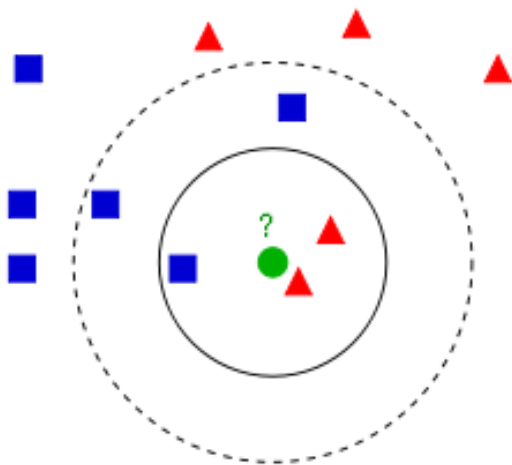
Supervised learning

- Supervised learning is a class of machine learning algorithm that takes in a series of vectors and their corresponding output (a continuous value or a class) as input, and produces an inferred function that can be used to map new examples.
- Precondition for using supervised learning is the availability of labeled data
- Supervised learning can be classified into two types: classification and regression.
- Classification problem has a discrete set of values as the target variable (for instance, a like and a dislike), whereas a regression problem has a continuous value as its target

k-nearest neighbors

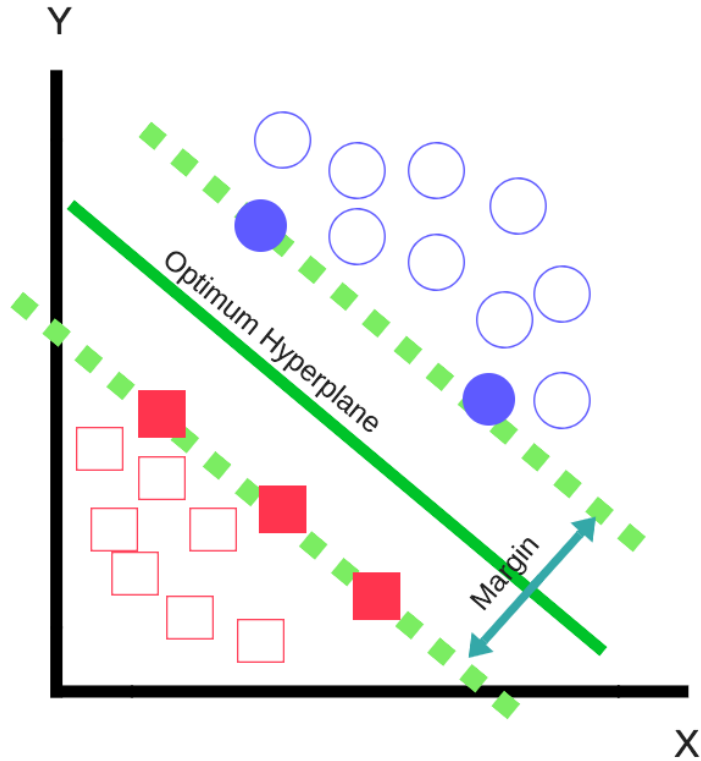
- **k-nearest neighbors (k-NN)** is perhaps the simplest machine learning algorithm
- Assigns a class to a particular data point by a majority vote of its k nearest neighbors
- In regression, it computes the average value for the target variable based on its k -nearest neighbors
- k-NN is non-parametric and lazy in nature
- k-NN does not make any underlying assumptions about the distribution of the data.

Classification

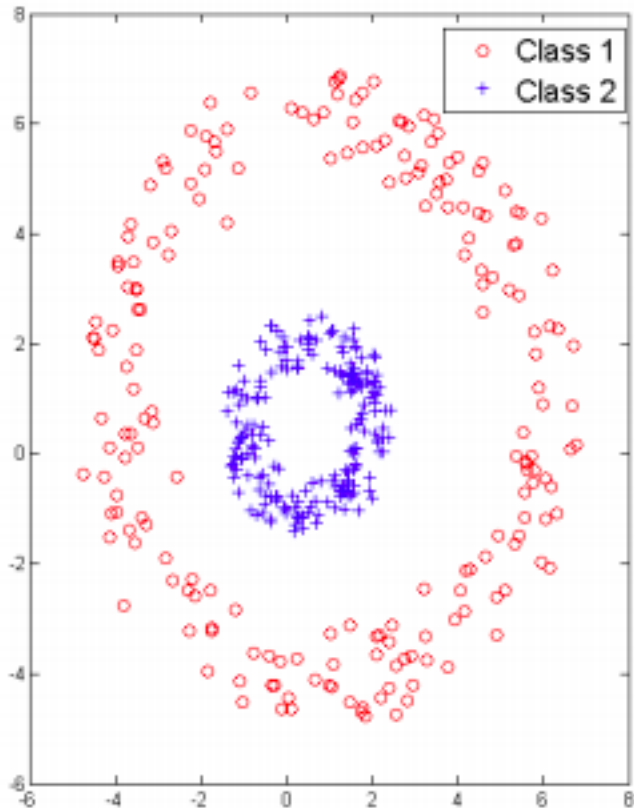


- ❖ Consider a dataset that has binary classes (represented as the blue squares and the red triangles).
- ❖ k-NN now plots this into n -dimensional space (in this case, two dimensions)
- ❖ k-NN computes the distance metric (usually the Euclidean distance) from the green circle to every other point in the training dataset and selects the three data points that are closest to it.

Support vector machines



- Most popular classification algorithms used in the industry.
- It takes in an n -dimensional dataset as input and constructs an $(n-1)$ dimensional hyperplane in such a way that there is maximum separation of classes
- Graph shows three possible hyperplanes (the straight lines) that separate the two classes. However, the solid line is the one with the maximum margin.
- it is the hyperplane that maximally separates the two classes.
- Also, it divides the entire plane into two regions. Any point below the hyperplane will be classified as a red square, and any point above will be classified as a blue circle



- SVM model is only dependent on support vectors; these are the points that determine the maximum margin possible between the two classes.
- In the preceding graph, the filled squares and circles are the support vectors

Decision trees

- Decision trees are extremely fast and simple tree-based algorithms that branch out on features that result in the largest information gain.
- Decision trees, although not very accurate, are extremely interpretable
- Decision trees have an element of randomness in their workings and come up with different conditions in different iterations

Ensembling

- Idea behind ensembling is that the predictive power of multiple algorithms is much greater than a single algorithm. Decision trees are the most common base algorithm used when building ensembling models

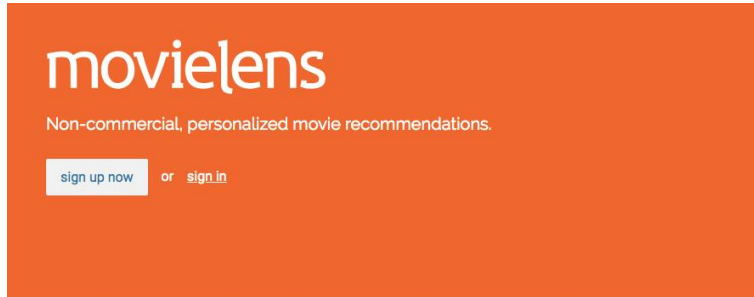
Bagging and random forests

- Bagging is short for bootstrap aggregating. Like most other ensemble methods, it averages over a large number of base classification models and averages their results to deliver its final prediction

Boosting

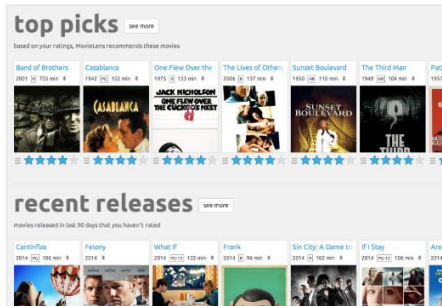
- Bagging and the random forest models train baseline models that are completely independent of each other
- Boosting models build a baseline model using a subset of samples and features.
- Boosting algorithms are extremely robust and routinely provide high performance

The MovieLens dataset



recommendations

MovieLens helps you find movies you will like. Rate movies to build a custom taste profile, then MovieLens recommends other movies for you to watch.



- The MovieLens dataset is made publicly available by GroupLens Research, a computer science lab at the University of Minnesota.
- It is one of the most popular benchmark datasets used to test the potency of various collaborative filtering models
- MovieLens gives us user ratings on a variety of movies and is available in various sizes.

Model-based approaches

- collaborative filters are known as memory-based filters
- make use of similarity metrics to come up with their results.
- They learn any parameters from the data or assign classes/clusters to the data.
- They do not make use of machine learning algorithms

Clustering

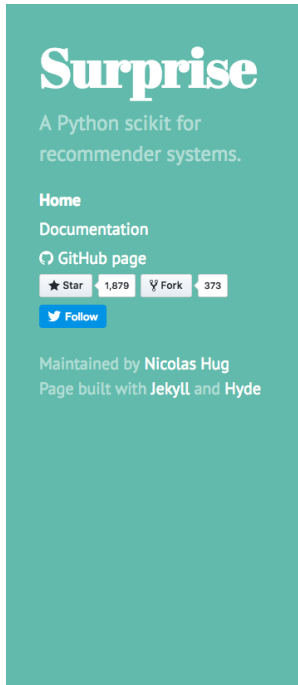
- Group users into a cluster and then take only the users from the same cluster into consideration when predicting ratings

➤ Example:

- Find the k -nearest neighbors of u who have rated movie m

Output the average rating of the k users for the movie m

1. Surprise is a scikit (or scientific kit) for building recommender systems in Python
2. According to its documentation, `surprise` stands for Simple Python Recommendation System Engine.
3. Within a very short span of time, `surprise` has gone on to become one of the most popularly used recommender libraries.
4. This is because it is extremely robust and easy to use.
5. It gives us ready-to-use implementations of most of the popular collaborative filtering algorithms and also allows us to integrate an algorithm of our own into the framework.



Overview

`Surprise` is a Python `scikit` building and analyzing recommender systems.

`Surprise` was designed with the following purposes in mind:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on [documentation](#), which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of [Dataset handling](#). Users can use both *built-in* datasets ([Movielens](#), [Jester](#)), and their own *custom* datasets.
- Provide various ready-to-use [prediction algorithms](#) such as [baseline algorithms](#), [neighborhood methods](#), matrix factorization-based ([SVD](#), [PMF](#), [SVD++](#), [NMF](#)), and [many others](#). Also, various [similarity measures](#) (cosine, MSD, pearson...) are built-in.
- Make it easy to implement [new algorithm ideas](#).
- Provide tools to [evaluate](#), [analyse](#) and [compare](#) the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by [scikit-learn](#) excellent tools), as well as [exhaustive search over a set of parameters](#).

The name *SurPRISE* (roughly :) stands for Simple Python Recommendation System Engine.

Getting started, example

Here is a simple example showing how you can (down)load a dataset, split it for 5-fold cross-validation, and compute the MAE and RMSE of the [SVD](#) algorithm.

- Makes use of a technique known as `fivefold cross-validation`.
- In a nutshell, this means that `surprise` divides the data into five equal parts.
- It then uses four parts as the training data and tests it on the fifth part.
- This is done five times, in such a way that every part plays the role of the test data once

Supervised learning and dimensionality reduction

- Most supervised learning algorithms do not work with missing data.
- In standard problems, it is common practice to impute the missing values with the mean or median of the column it belongs to
- Simon Funk came up with a solution that could be used to reduce the $m \times (n-1)$ matrix into a lower-dimensional $m \times d$ matrix where $d \ll n$.
- He used standard dimensionality-reduction techniques (in his case, the SVD) but with slight tweaks
- **PCA (Principal Component Analysis)** transforms an $m \times n$ matrix into n , m -dimensional vectors (called principal components) in such a way that each component is orthogonal to the next component

$$\begin{array}{ccccc}
 \begin{array}{|c|} \hline A \\ \hline n \times d \\ \hline \end{array} & = & \begin{array}{|c|} \hline \hat{U} \\ \hline n \times r \\ \hline \end{array} & \begin{array}{|c|} \hline \hat{\Sigma} \\ r \times r \\ \hline \end{array} & \begin{array}{|c|} \hline \hat{V}^T \\ r \times d \\ \hline \end{array} \\
 & & U & \Sigma & V^T \\
 & & n \times d & n \times d & d \times d
 \end{array}$$

Let's denote our ratings matrix as A . The transpose of this matrix would be A^T , which would be of the $n \times m$ shape and each row would represent a movie (instead of a user)

Use PCA to construct two new matrices, U and V , from A and A^T , respectively

Singular-value decomposition allows us to compute U and V in one go from A :

$$A = U\Sigma V^T$$

- The U matrix, which is essentially composed of user principal components, is typically called the user-embedding matrix
- Its counterpart, V , is called the movie-embedding matrix.
- The classic version of SVD, like most other machine learning algorithms, does not work with sparse matrices
- However, Simon Funk figured out a workaround for this problem, and his solution led to one of the most famous solutions in the world of recommender systems
- Funk's system took in the sparse ratings matrix, A , and constructed two dense user- and item-embedding matrices, U and V respectively. These dense matrices directly gave us the predictions for all the missing values in the original matrix, A .

Hybrid Recommenders

- Hybrid recommenders are extremely powerful, robust systems that combine various simpler models to give us prediction
- Some hybrids predict using content and collaborative filtering techniques separately to produce results. Some others introduce content-based techniques into collaborative filters and vice versa
- Netflix is a very good example of a hybrid recommender . Netflix employs both content- and collaborative-based techniques separately to produce results that are extremely satisfactory

Case study – Building a hybrid model

1. Imagine that you have built a website like Netflix.
2. Every time a user watches a movie, you want to display a list of recommendations in the side pane (like YouTube).
3. Let's say our user is watching *The Dark Knight*. Since this is a Batman movie, our content-based recommender is likely to recommend other Batman (or superhero) movies regardless of quality
4. Take in a movie title and user as input
5. Use a content-based model to compute the 25 most similar movies
6. Compute the predicted ratings that the user might give these 25 movies using a collaborative filter
7. Return the top 10 movies with the highest predicted rating

Download the following datasets from Kaggle and Google Drive:

- ratings_small.csv: https://www.kaggle.com/rounakbanik/the-movies-dataset/downloads/ratings_small.csv/7.
- movie_ids.csv: https://drive.google.com/drive/folders/1H9pnfVTzP46s7VwOTcC5ZY_VahRTr5Zv?usp=sharing.

- The `ratings_small.csv` file contains 100,000 ratings for 9,000 movies from 700 users. We use this file since it contains ratings for more recent movies (the dataset we used for collaborative filtering only contained movies released before 1998).
- The `links_small.csv` file contains the movie IDs of all the movies rated in the `ratings_small.csv` file and their corresponding titles. We can use these IDs to extract relevant metadata from the `movies_metadata.csv` file.
- The first step is to compute the `cosine_sim` matrix for our movies. In addition, we also need to map every movie to the indices in the `cosine_sim` matrix
- Next, let's use the `ratings.csv` file to build a collaborative filtering model. We will use the SVD model from the last chapter for this purpose

Build the SVD based Collaborative filter
from surprise import SVD, Reader, Dataset

```
reader = Reader()
ratings = pd.read_csv('../data/ratings_small.csv')
data = Dataset.load_from_df(ratings[['userId', 'movieId',
'rating']], reader)
data.split(n_folds=5)
svd = SVD()
trainset = data.build_full_trainset()
svd.train(trainset)
```

Next, let's load the `movie_ids.csv` file into a DataFrame and construct two mappings: one that returns the movie title for a given movie ID, and the other vice versa

```
#Build title to ID and ID to title mappings
id_map = pd.read_csv('../data/movie_ids.csv')
id_to_title = id_map.set_index('id')
title_to_id = id_map.set_index('title')
```

Now, let's import the metadata for our movies so that our recommender can display useful information, such as the IMDB rating and the year of release. This information can be extracted from the main `movies_metadata.csv` file, and is again left as an exercise for the reader.

You can download the required metadata file from the following

link: https://drive.google.com/drive/folders/1H9pnfVTzP46s7VwOTcC5ZY_VahRTr5Zv?usp=sharing

	title	vote_count	vote_average	year	id	est
1011	The Terminator	4208.0	7.4	1984	218	3.140748
974	Aliens	3282.0	7.7	1986	679	3.126947
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.079551
7705	Alice in Wonderland	8.0	5.4	1933	25694	3.054995
3060	Sinbad and the Eye of the Tiger	39.0	6.3	1977	11940	3.028386
8658	X-Men: Days of Future Past	6155.0	7.5	2014	127585	2.997411
2014	Fantastic Planet	140.0	7.6	1973	16306	2.957614
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	2.914548
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	2.844940
1668	Return from Witch Mountain	38.0	5.6	1978	14822	2.804012

```
hybrid(1, 'Avatar')
```

	title	vote_count	vote_average	year	id	est
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	3.943639
2834	Predator	2129.0	7.3	1987	106	3.866272
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.858491
1011	The Terminator	4208.0	7.4	1984	218	3.856029
7705	Alice in Wonderland	8.0	5.4	1933	25694	3.701565
922	The Abyss	822.0	7.1	1989	2756	3.676465
974	Aliens	3282.0	7.7	1986	679	3.672303
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	3.628234
1668	Return from Witch Mountain	38.0	5.6	1978	14822	3.614118
2014	Fantastic Planet	140.0	7.6	1973	16306	3.602051

hybrid(2, 'Avatar')

Summary

- Introduced to the world of recommender systems.
- Defined the recommendation problem mathematically and discussed the various types of recommendation engines that exist, as well as their advantages and disadvantages.
- Learned to perform data wrangling with the pandas library and familiarized ourselves with two of pandas, most powerful data structures: the series and the DataFrame.
- With our newly found data wrangling techniques, we proceeded to build an IMDB Top 250 clone.
- Learned how to build content-based recommenders using plot lines and subsequently more sophisticated metadata, such as the genre, cast, crew, and keywords.
- Collaborative filtering had us experimenting with a variety of models that used rating data, and also leveraged data mining techniques introduced in the previous chapter.
- Introduce to the `surprise` library, which made building recommender systems a breeze.
- In this final chapter, we briefly discussed the various kinds of hybrid recommender used in the industry today and built a model that incorporated collaborative filtering into a content-based engine to offer personalized recommendations to a user, while keeping the current movie being watched in mind.