Content based filtering

A common approach when designing recommender systems is content-based filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preferences. These methods are best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on an item's features.

In this system, keywords are used to describe the items and a [user profile](https://en.wikipedia.org/wiki/User_profile" \o "User profile) is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present. It does not rely on a user sign-in mechanism to generate this often temporary profile. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in [information retrieval](https://en.wikipedia.org/wiki/Information_retrieval" \o "Information retrieval) and [information filtering](https://en.wikipedia.org/wiki/Information_filtering" \o "Information filtering) research.

To create a [user profile](https://en.wikipedia.org/wiki/User_profile" \o "User profile), the system mostly focuses on two types of information:

1. A model of the user's preference.

2. A history of the user's interaction with the recommender system.

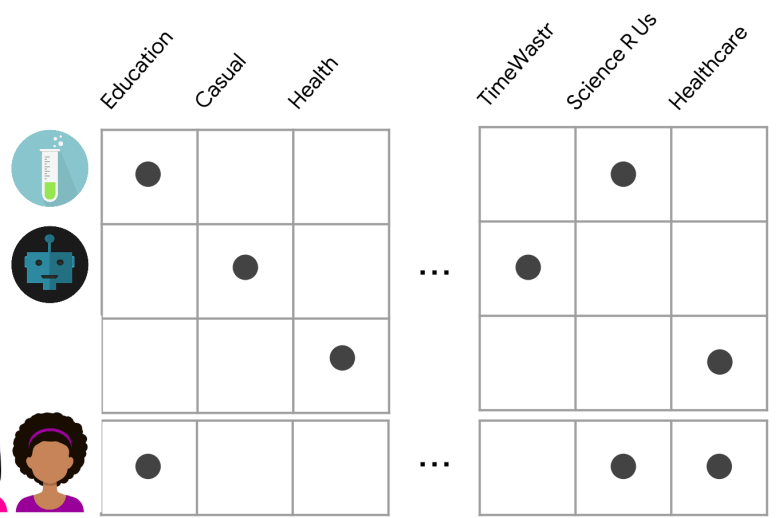
Basically, these methods use an item profile (i.e., a set of discrete attributes and features) characterizing the item within the system. To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the [tf–idf](https://en.wikipedia.org/wiki/Tf%E2%80%93idf" \o "Tf–idf) representation (also called vector space representation). The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as [Bayesian Classifiers](https://en.wikipedia.org/wiki/Naive_Bayes_classifier" \o "Naive Bayes classifier), [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis" \o "Cluster analysis), [decision trees](https://en.wikipedia.org/wiki/Decision_trees" \o "Decision trees), and [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_networks" \o "Artificial neural networks) in order to estimate the probability that the user is going to like the item.

A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but would be much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing. To overcome this, most content-based recommender systems now use some form of hybrid system.

To demonstrate content-based filtering, let’s hand-engineer some features for the Google Play store. The following figure shows a feature matrix where each row represents an app and each column represents a feature. Features could include categories (such as Education, Casual, Health), the publisher of the app, and many others. To simplify, assume this feature matrix is binary: a non-zero value means the app has that feature.

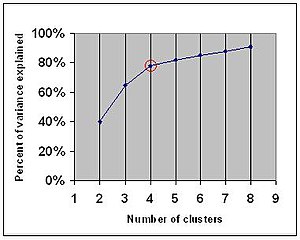
You also represent the user in the same feature space. Some of the user-related features could be explicitly provided by the user. For example, a user selects "Entertainment apps" in their profile. Other features can be implicit, based on the apps they have previously installed. For example, the user installed another app published by Science R Us.

The model should recommend items relevant to this user. To do so, you must first pick a similarity metric (for example, dot product). Then, you must set up the system to score each candidate item according to this similarity metric. Note that the recommendations are specific to this user, as the model did not use any information about other users.



Elbow method

In [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis" \o "Cluster analysis), the elbow method is a [heuristic](https://en.wikipedia.org/wiki/Heuristic" \o "Heuristic) used in [determining the number of clusters in a data set](https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set" \o "Determining the number of clusters in a data set). The method consists of plotting the [explained variation](https://en.wikipedia.org/wiki/Explained_variation" \o "Explained variation) as a function of the number of clusters, and picking the [elbow of the curve](https://en.wikipedia.org/wiki/Elbow_of_the_curve" \o "Elbow of the curve) as the number of clusters to use. The same method can be used to choose the number of parameters in other data-driven models, such as the number of [principal components](https://en.wikipedia.org/wiki/Principal_component" \o "Principal component) to describe a data set.



Explained variance. The "elbow" is indicated by the red circle. The number of clusters chosen should therefore be 4.

Using the "elbow" or "[knee of a curve](https://en.wikipedia.org/wiki/Knee_of_a_curve" \o "Knee of a curve)" as a cutoff point is a common heuristic in [mathematical optimization](https://en.wikipedia.org/wiki/Mathematical_optimization" \o "Mathematical optimization) to choose a point where [diminishing returns](https://en.wikipedia.org/wiki/Diminishing_returns" \o "Diminishing returns) are no longer worth the additional cost. In clustering, this means one should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data.

The intuition is that increasing the number of clusters will naturally improve the fit (explain more of the variation), since there are more parameters (more clusters) to use, but that at some point this is [over-fitting](https://en.wikipedia.org/wiki/Over-fitting" \o "Over-fitting), and the elbow reflects this. For example, given data that actually consist of *k* labeled groups – for example, *k* points sampled with noise – clustering with more than *k* clusters will "explain" more of the variation (since it can use smaller, tighter clusters), but this is over-fitting, since it is subdividing the labeled groups into multiple clusters. The idea is that the first clusters will add much information (explain a lot of variation), since the data actually consist of that many groups (so these clusters are necessary), but once the number of clusters exceeds the actual number of groups in the data, the added information will drop sharply, because it is just subdividing the actual groups. Assuming this happens, there will be a sharp elbow in the graph of explained variation versus clusters: increasing rapidly up to *k* ([under-fitting](https://en.wikipedia.org/wiki/Under-fitting" \o "Under-fitting) region), and then increasing slowly after *k* (over-fitting region).

In practice there may not be a sharp elbow, and as a heuristic method, such an "elbow" cannot always be unambiguously identified.

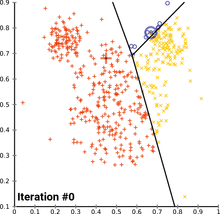
K means

*k*-means clustering is a method of [vector quantization](https://en.wikipedia.org/wiki/Vector_quantization" \o "Vector quantization), originally from [signal processing](https://en.wikipedia.org/wiki/Signal_processing" \o "Signal processing), that aims to [partition](https://en.wikipedia.org/wiki/Partition_of_a_set" \o "Partition of a set) *n* observations into *k* clusters in which each observation belongs to the [cluster](https://en.wikipedia.org/wiki/Cluster_(statistics)" \o ") with the nearest [mean](https://en.wikipedia.org/wiki/Mean" \o "Mean) (cluster centers or cluster [centroid](https://en.wikipedia.org/wiki/Centroid" \o "Centroid)), serving as a prototype of the cluster. This results in a partitioning of the data space into [Voronoi cells](https://en.wikipedia.org/wiki/Voronoi_cell" \o "Voronoi cell). It is popular for [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis" \o "Cluster analysis) in [data mining](https://en.wikipedia.org/wiki/Data_mining" \o "Data mining). *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult [Weber problem](https://en.wikipedia.org/wiki/Weber_problem" \o "Weber problem): the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, Better Euclidean solutions can be found using [k-medians](https://en.wikipedia.org/wiki/K-medians_clustering" \o "K-medians clustering) and [k-medoids](https://en.wikipedia.org/wiki/K-medoids" \o "K-medoids).

The problem is computationally difficult ([NP-hard](https://en.wikipedia.org/wiki/NP-hardness" \o "NP-hardness)); however, efficient [heuristic algorithms](https://en.wikipedia.org/wiki/Heuristic_algorithm" \o "Heuristic algorithm) converge quickly to a [local optimum](https://en.wikipedia.org/wiki/Local_optimum" \o "Local optimum). These are usually similar to the [expectation-maximization algorithm](https://en.wikipedia.org/wiki/Expectation-maximization_algorithm" \o "Expectation-maximization algorithm) for [mixtures](https://en.wikipedia.org/wiki/Mixture_model" \o "Mixture model) of [Gaussian distributions](https://en.wikipedia.org/wiki/Gaussian_distribution" \o "Gaussian distribution) via an iterative refinement approach employed by both *k-means* and *Gaussian mixture modeling*. They both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the *[k](https://en.wikipedia.org/wiki/K-nearest_neighbor" \o "K-nearest neighbor)*[-nearest neighbor classifier](https://en.wikipedia.org/wiki/K-nearest_neighbor" \o "K-nearest neighbor), a popular [machine learning](https://en.wikipedia.org/wiki/Machine_learning" \o "Machine learning) technique for classification that is often confused with *k*-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by *k*-means classifies new data into the existing clusters. This is known as [nearest centroid classifier](https://en.wikipedia.org/wiki/Nearest_centroid_classifier" \o "Nearest centroid classifier) or [Rocchio algorithm](https://en.wikipedia.org/wiki/Rocchio_algorithm" \o "Rocchio algorithm).

### Standard algorithm (naive k-means)

[](https://en.wikipedia.org/wiki/File:K-means_convergence.gif)

Convergence of *k*-means

The most common algorithm uses an iterative refinement technique. Due to its ubiquity, it is often called "the *k*-means algorithm"; it is also referred to as [Lloyd's algorithm](https://en.wikipedia.org/wiki/Lloyd's_algorithm" \o "Lloyd's algorithm), particularly in the computer science community. It is sometimes also referred to as "naive *k*-means", because there exist much faster alternatives.

Given an initial set of *k* means *m*1(1),...,*mk*(1) (see below), the algorithm proceeds by alternating between two steps:

Assignment step: Assign each observation to the cluster with the nearest mean: that with the least squared [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance" \o "Euclidean distance). (Mathematically, this means partitioning the observations according to the [Voronoi diagram](https://en.wikipedia.org/wiki/Voronoi_diagram" \o "Voronoi diagram) generated by the means.)

S i ( t ) = { x p : ‖ x p − m i ( t ) ‖ 2 ≤ ‖ x p − m j ( t ) ‖ 2   ∀ j , 1 ≤ j ≤ k } , {\displaystyle S\_{i}^{(t)}={\big \{}x\_{p}:{\big \|}x\_{p}-m\_{i}^{(t)}{\big \|}^{2}\leq {\big \|}x\_{p}-m\_{j}^{(t)}{\big \|}^{2}\ \forall j,1\leq j\leq k{\big \}},}

where each x p {\displaystyle x\_{p}} IMG_258is assigned to exactly one S ( t ) {\displaystyle S^{(t)}} IMG_259, even if it could be assigned to two or more of them.

Update step: Recalculate means ([centroids](https://en.wikipedia.org/wiki/Centroids" \o "Centroids)) for observations assigned to each cluster.

m i ( t + 1 ) = 1 | S i ( t ) | ∑ x j ∈ S i ( t ) x j {\displaystyle m\_{i}^{(t+1)}={\frac {1}{\left|S\_{i}^{(t)}\right|}}\sum \_{x\_{j}\in S\_{i}^{(t)}}x\_{j}}

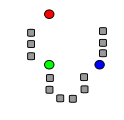
The algorithm has converged when the assignments no longer change. The algorithm does not guarantee to find the optimum.

The algorithm is often presented as assigning objects to the nearest cluster by distance. Using a different distance function other than (squared) Euclidean distance may stop the algorithm from converging. Various modifications of *k*-means such as spherical *k*-means and *[k](https://en.wikipedia.org/wiki/K-medoids" \o "K-medoids)*[-medoids](https://en.wikipedia.org/wiki/K-medoids" \o "K-medoids) have been proposed to allow using other distance measures.

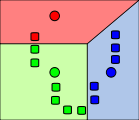
#### Initialization methods

Commonly used initialization methods are Forgy and Random Partition. The Forgy method randomly chooses *k* observations from the dataset and uses these as the initial means. The Random Partition method first randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points. The Forgy method tends to spread the initial means out, while Random Partition places all of them close to the center of the data set. According to Hamerly et al., the Random Partition method is generally preferable for algorithms such as the *k*-harmonic means and fuzzy *k*-means. For expectation maximization and standard *k*-means algorithms, the Forgy method of initialization is preferable. A comprehensive study by Celebi et al., however, found that popular initialization methods such as Forgy, Random Partition, and Maximin often perform poorly, whereas Bradley and Fayyad's approach performs "consistently" in "the best group" and *[k](https://en.wikipedia.org/wiki/K-means++" \o "K-means++)*[-means++](https://en.wikipedia.org/wiki/K-means++" \o "K-means++) performs "generally well".

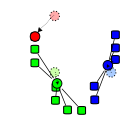
* Demonstration of the standard algorithm

[](https://en.wikipedia.org/wiki/File:K_Means_Example_Step_1.svg)

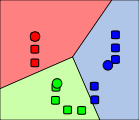
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

[](https://en.wikipedia.org/wiki/File:K_Means_Example_Step_2.svg)

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the [Voronoi diagram](https://en.wikipedia.org/wiki/Voronoi_diagram" \o "Voronoi diagram) generated by the means.

[](https://en.wikipedia.org/wiki/File:K_Means_Example_Step_3.svg)

3. The [centroid](https://en.wikipedia.org/wiki/Centroid" \o "Centroid) of each of the *k* clusters becomes the new mean.

[](https://en.wikipedia.org/wiki/File:K_Means_Example_Step_4.svg)

4. Steps 2 and 3 are repeated until convergence has been reached.

The algorithm does not guarantee convergence to the global optimum. The result may depend on the initial clusters. As the algorithm is usually fast, it is common to run it multiple times with different starting conditions. However, worst-case performance can be slow: in particular certain point sets, even in two dimensions, converge in exponential time, that is 2Ω(n). These point sets do not seem to arise in practice: this is corroborated by the fact that the [smoothed](https://en.wikipedia.org/wiki/Smoothed_analysis" \o "Smoothed analysis) running time of *k*-means is polynomial.

The "assignment" step is referred to as the "expectation step", while the "update step" is a maximization step, making this algorithm a variant of the *generalized* [expectation-maximization algorithm](https://en.wikipedia.org/wiki/Expectation-maximization_algorithm" \o "Expectation-maximization algorithm).

1. Sci-kit learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

* **NumPy**: Base n-dimensional array package
* **SciPy**: Fundamental library for scientific computing
* **Matplotlib**: Comprehensive 2D/3D plotting
* **IPython**: Enhanced interactive console
* **Sympy**: Symbolic mathematics
* **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named [SciKits](http://scikits.appspot.com/scikits). As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, [LAPACK](http://www.netlib.org/lapack/), [LibSVM](http://www.csie.ntu.edu.tw/~cjlin/libsvm/) and the careful use of cython.