Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Understanding Data vs. Machine Training

Malte Luttermann – Institute for Humanities-Centered Artificial Intelligence (CHAI)

December 19, 2025 and January 9, 2026

Version from January 9, 2026

Overview of Contents

(1) Programming language Python
  (a) Introduction and first steps
  (b) Basics
  (c) Advanced
(2) Markup languages
  (a) LaTeX
  (b) Markdown
(3) Development environments
  (a) Jupyter notebooks
(4) Version control
  (a) Git and GitHub
(5) Scientific computing
  (a) NumPy and SciPy
(6) Data processing and visualisation
  (a) Pandas, matplotlib, and NLTK
(7) Machine learning (scikit-learn)
  (a) Basics (datasets, analysis)
  (b) Simple methods (clustering, . . .)
(8) Deep learning
  (a) PyTorch

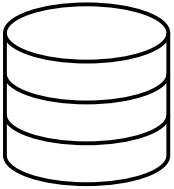
Topics for Today (and Next Time)

(1) Terminology
   (a) Artificial intelligence, machine learning, data science
   (b) Agents
(2) Clustering
(3) Recommendations
(4) Natural language processing
   (a) TF.IDF
   (b) Clustering of documents
   (c) Recommendation of documents

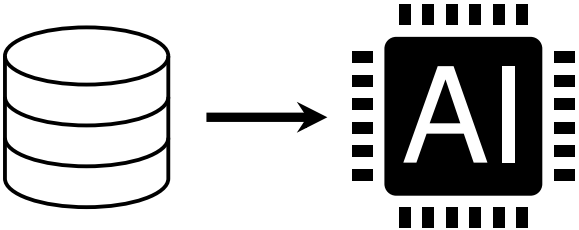
Acknowledgement

- The upcoming slides are taken from the following lecture and have been translated and partially modified:
  - Dr. Magnus Bender: »Python für Machine Learning und Data Science« as part of the German course »Werkzeuge für das wissenschaftliche Arbeiten«

»Artificial Intelligence«

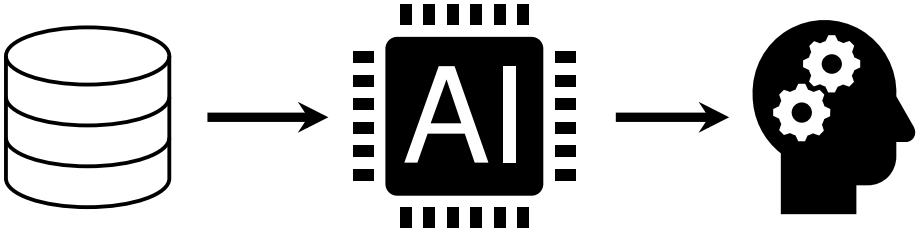


»Artificial Intelligence«



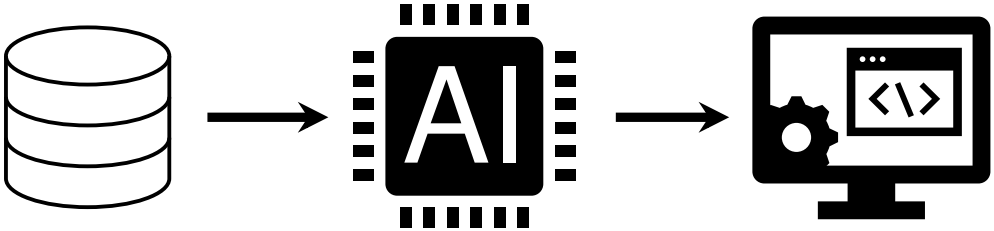
»The AI«

»Artificial Intelligence«



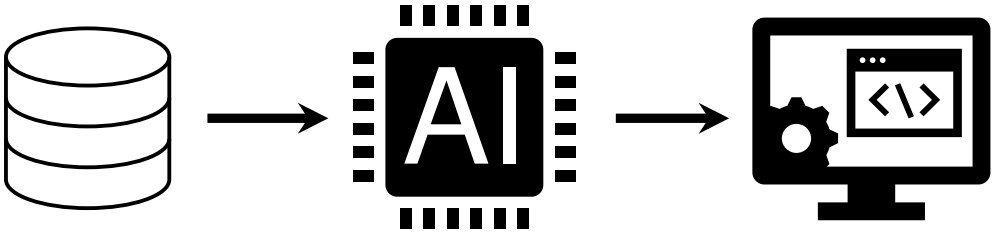
»The AI«

»Artificial Intelligence«



»The AI«

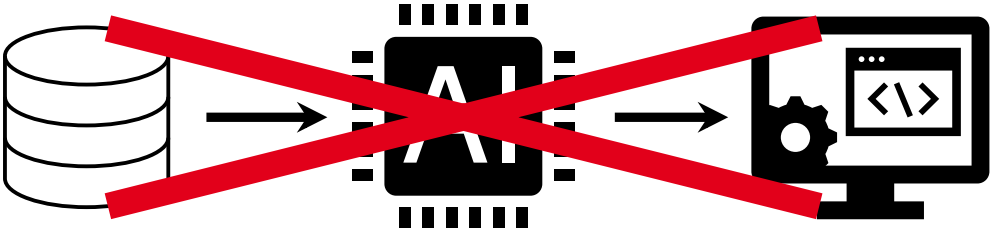
»Artificial Intelligence«



»The AI«

For example:
Classification, regression, prediction, diagnosis, . . .

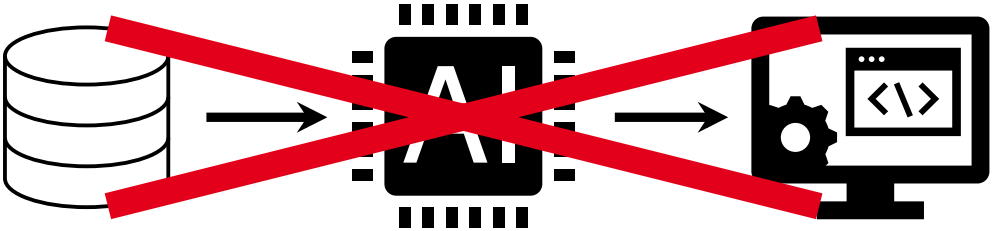
»Artificial Intelligence«



»The AI«

For example:
Classification, regression, prediction, diagnosis, . . .

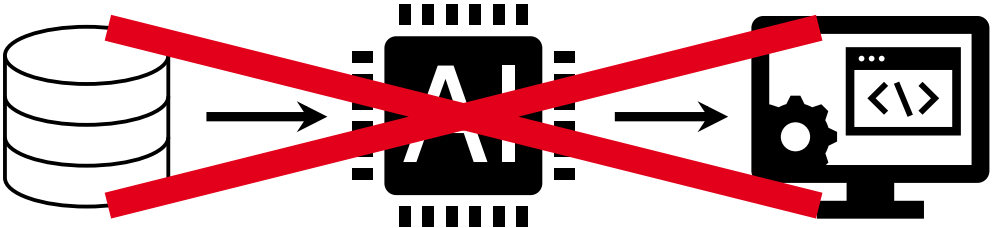
»Artificial Intelligence«



~~» The AI «~~

For example:
Classification, regression, prediction, diagnosis, . . .

Fairy Tale

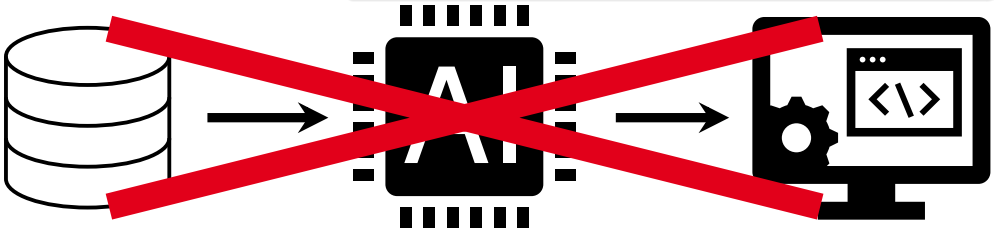


~~» The AI «~~

For example:
Classification, regression, prediction, diagnosis, . . .

Fairy Tale

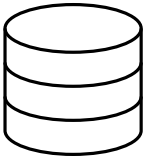
»AI« as a building block operating like the brain and then brings »intelligence« to a computer.



~~»The AI«~~

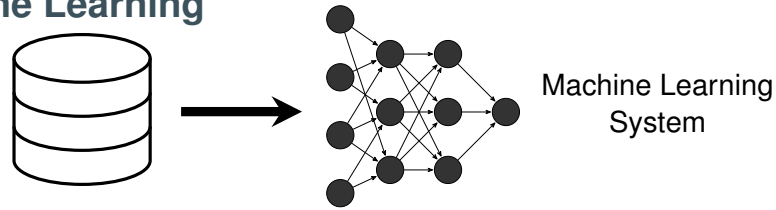
For example:
Classification, regression, prediction, diagnosis, . . .

Machine Learning

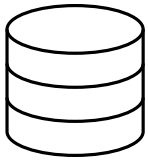


Data

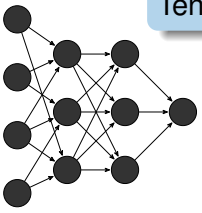
Machine Learning



Machine Learning



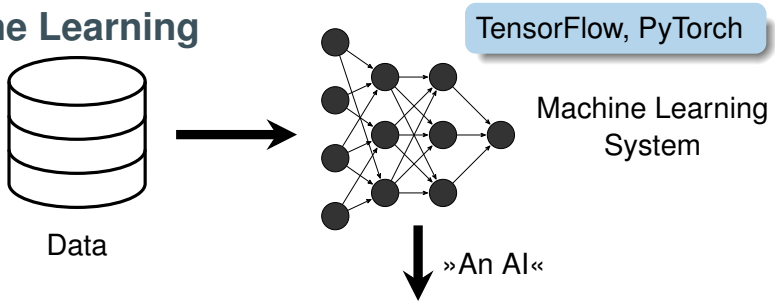
Data



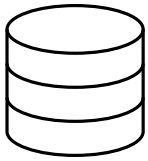
TensorFlow, PyTorch

Machine Learning System

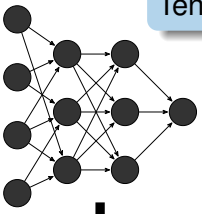
Machine Learning



Machine Learning



Data



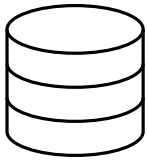
Machine Learning System

TensorFlow, PyTorch

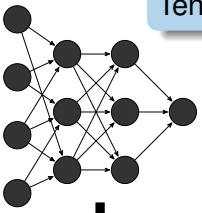
»An AI«

Program: Decision tree evaluator, cascade of linear and piecewise linear functions

Machine Learning



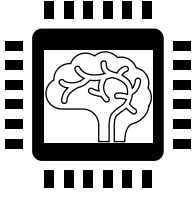
Data



TensorFlow, PyTorch

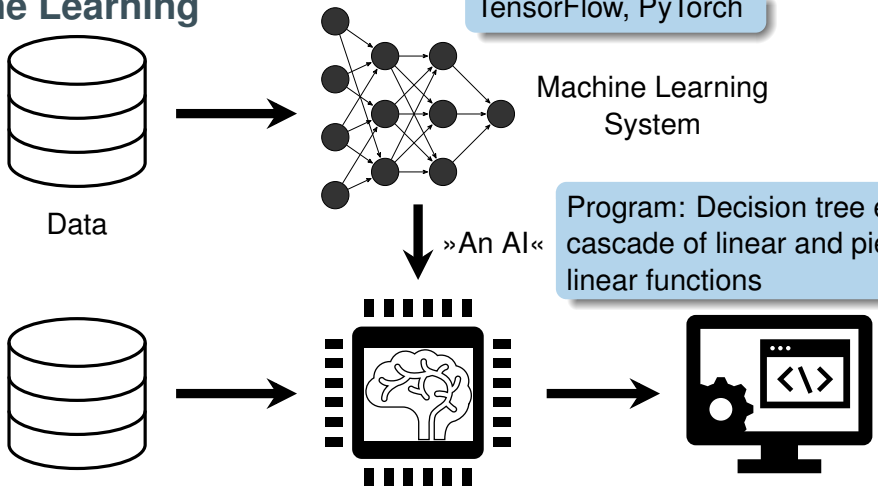
Machine Learning System

»An AI«

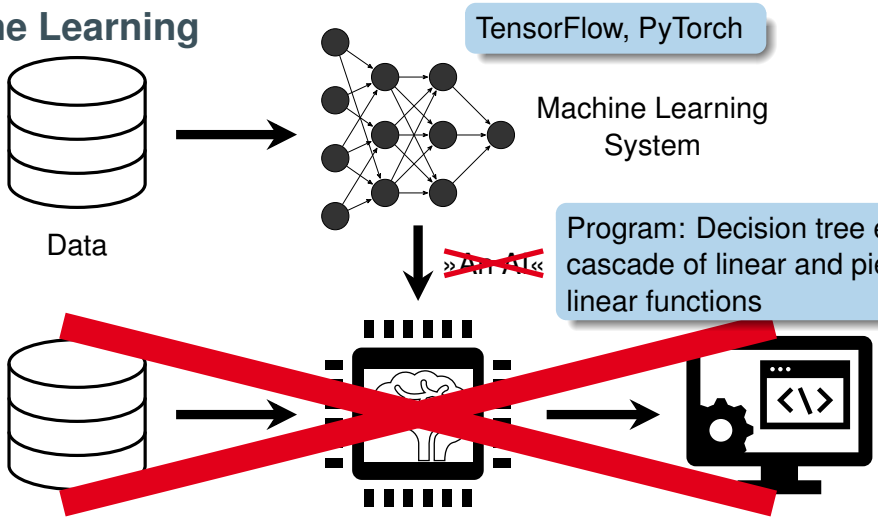


Program: Decision tree evaluator, cascade of linear and piecewise linear functions

Machine Learning



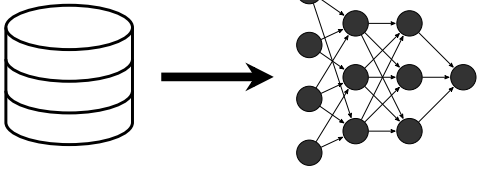
Machine Learning



Machine Learning

Training Data

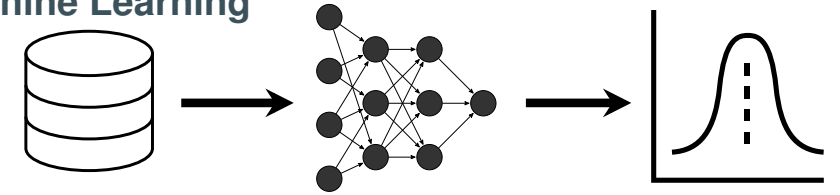
Machine Learning



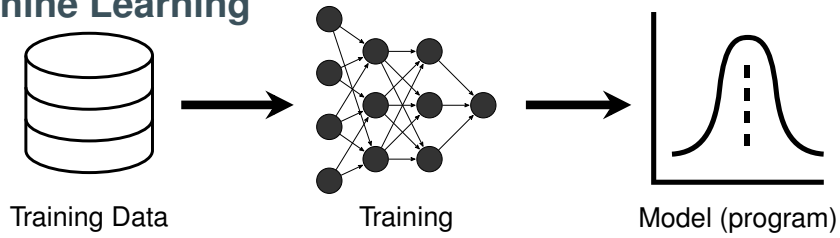
Training Data

Training

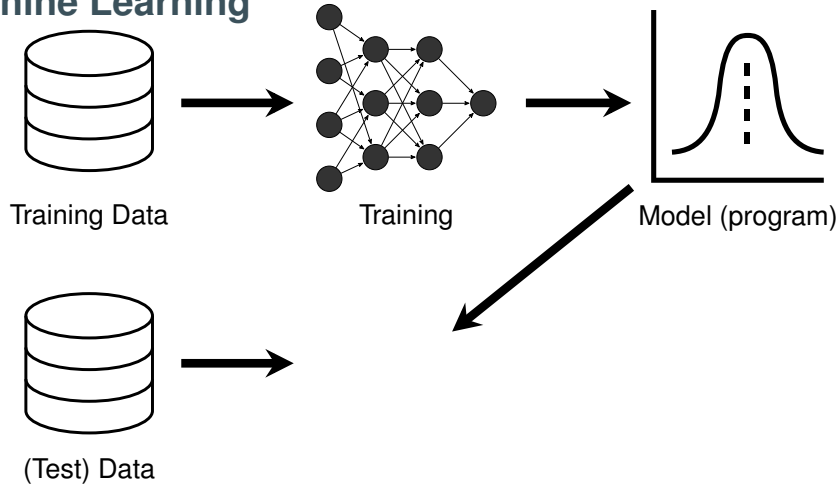
Machine Learning



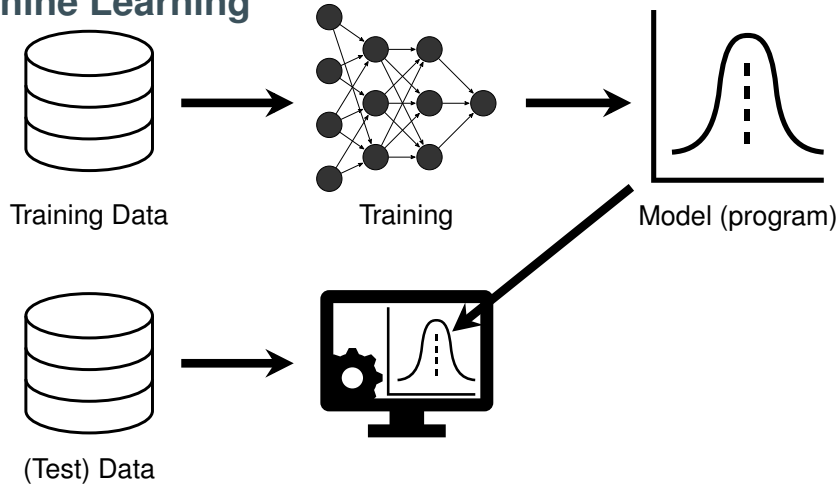
Machine Learning



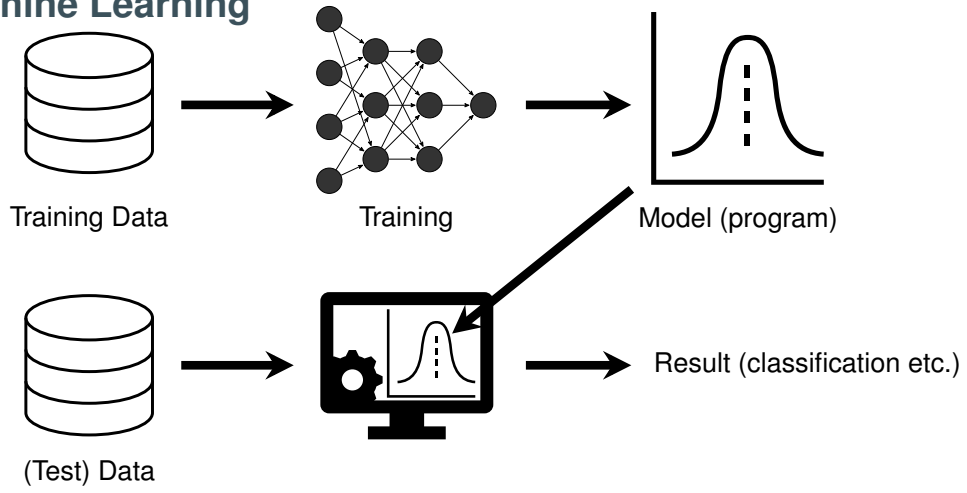
Machine Learning



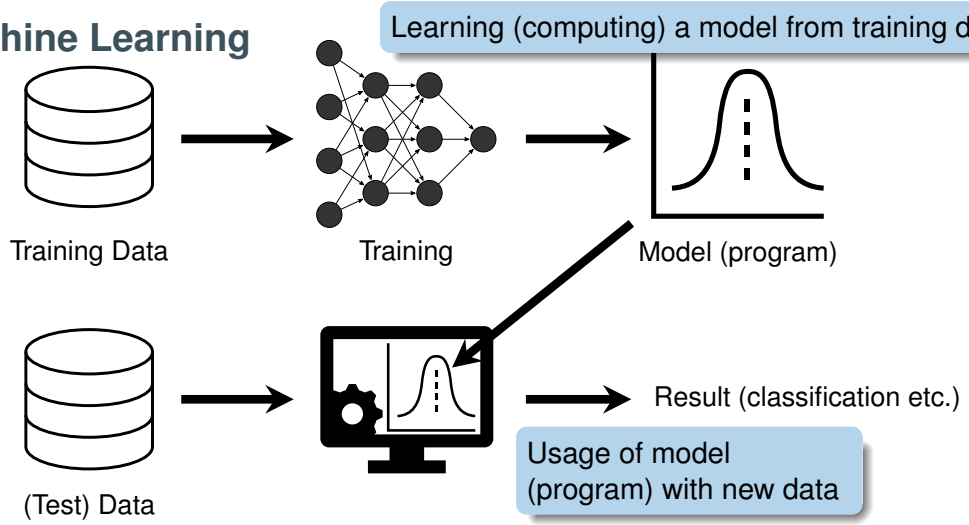
Machine Learning



Machine Learning

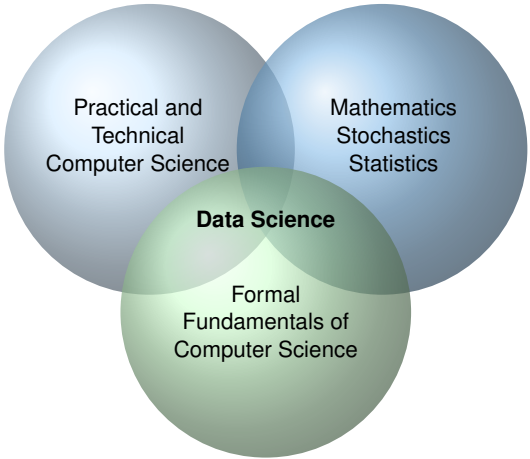


Machine Learning

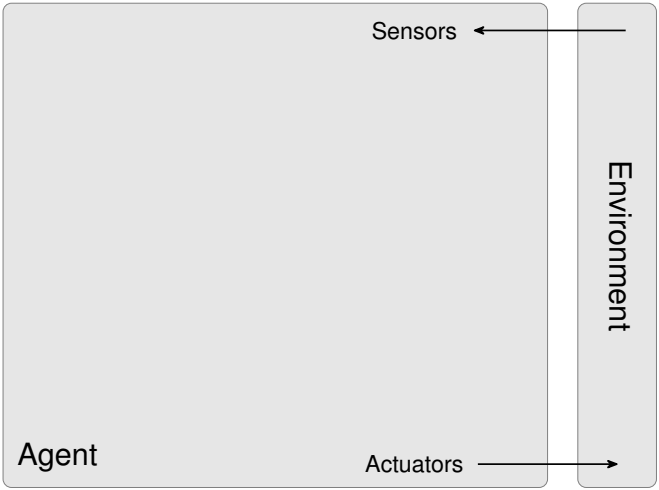


Data Science

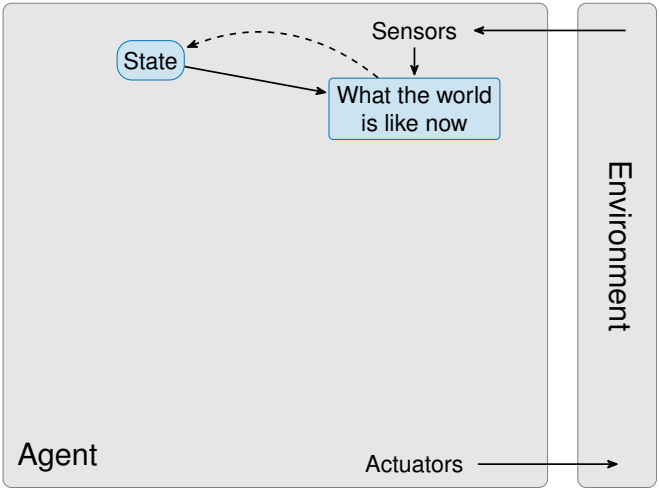
- Extraction of knowledge from data (including graph data)
- Term proposed already 60 years ago for computer science
- Development of innovative concepts in the fields of logic, databases and stochastics/statistics (data analysis and knowledge discovery)
- Use of linear algebra and analysis



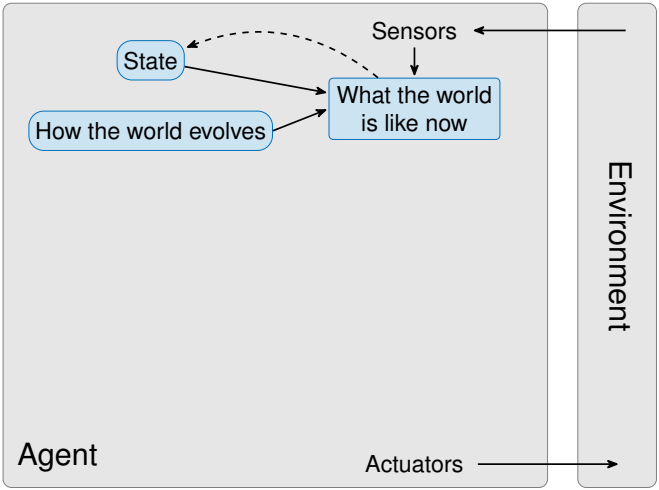
Agents



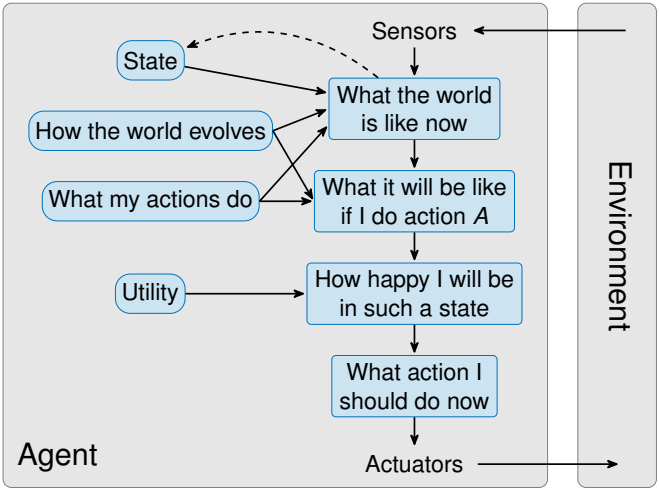
Agents


Sensors
State
What the world is like now
Environment
Agent
Actuators

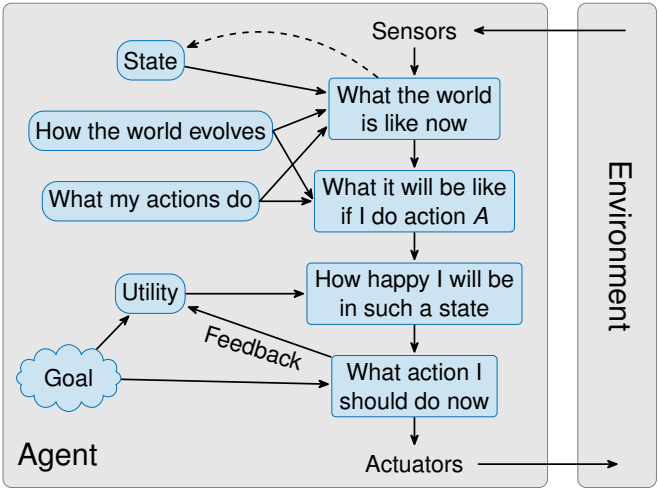
Agents


Sensors
State
What the world is like now
How the world evolves
Environment
Agent
Actuators

Agents

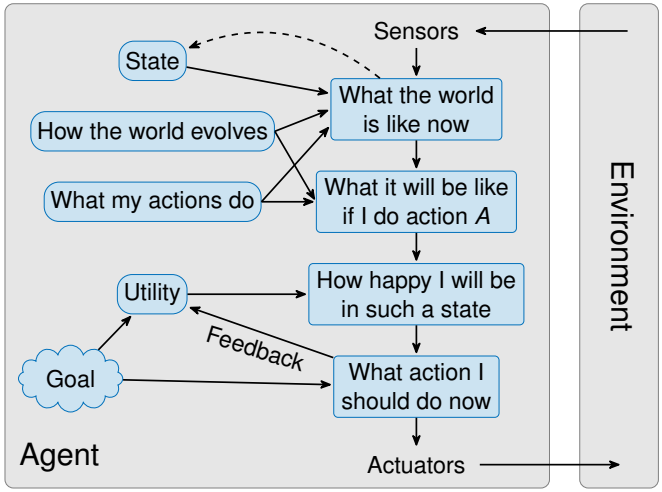

Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Agents

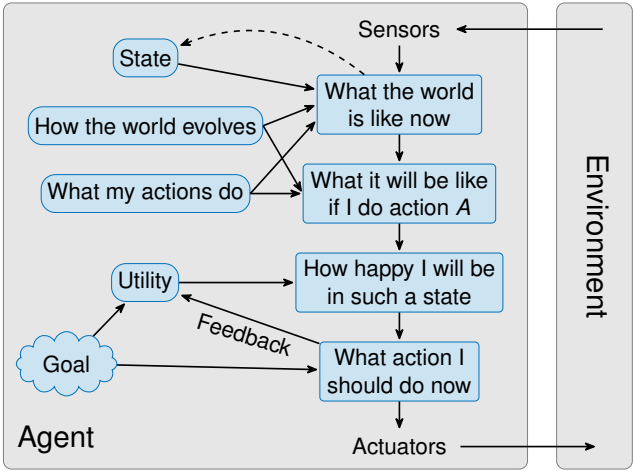

Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action A
Utility
How happy I will be in such a state
Feedback
Goal
What action I should do now
Agent
Actuators
Environment

Agents

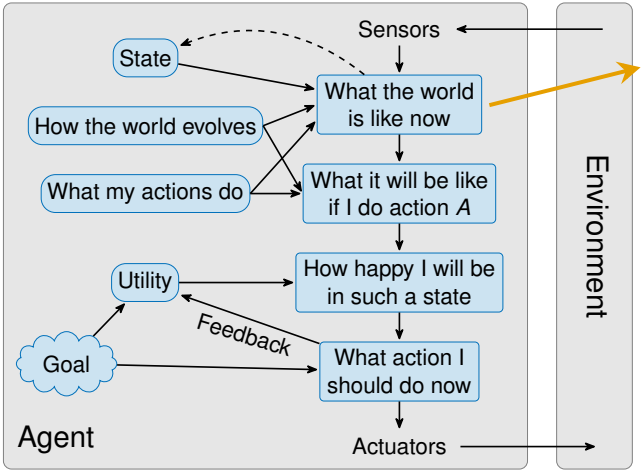
- Science of intelligent systems
- Agents
  - Have/form goals
  - Sensors/actuators
  - Action planning
  - Learning at runtime
- Agents interact with people (and other agents)
  - Agents' objectives (goals) can be influenced



Agent: Machine Learning



Agent: Machine Learning



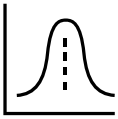
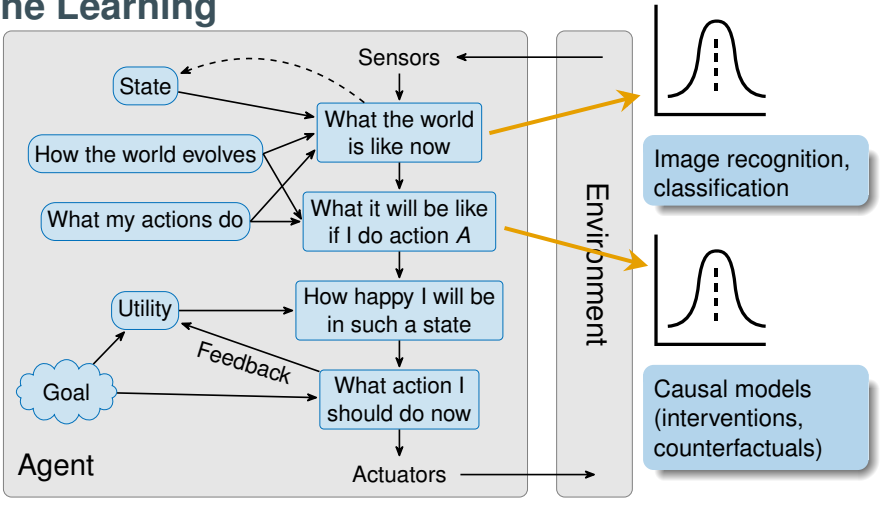
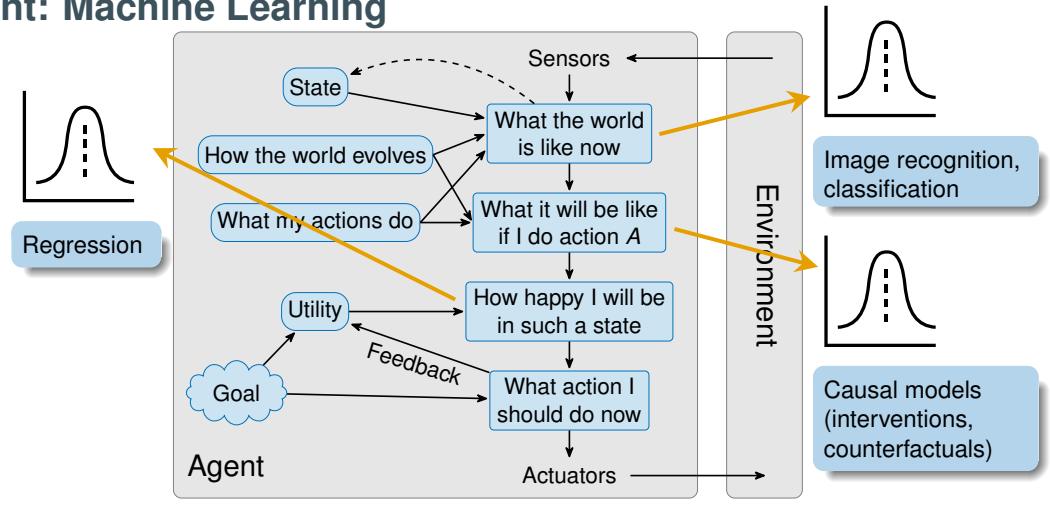


Image recognition, classification

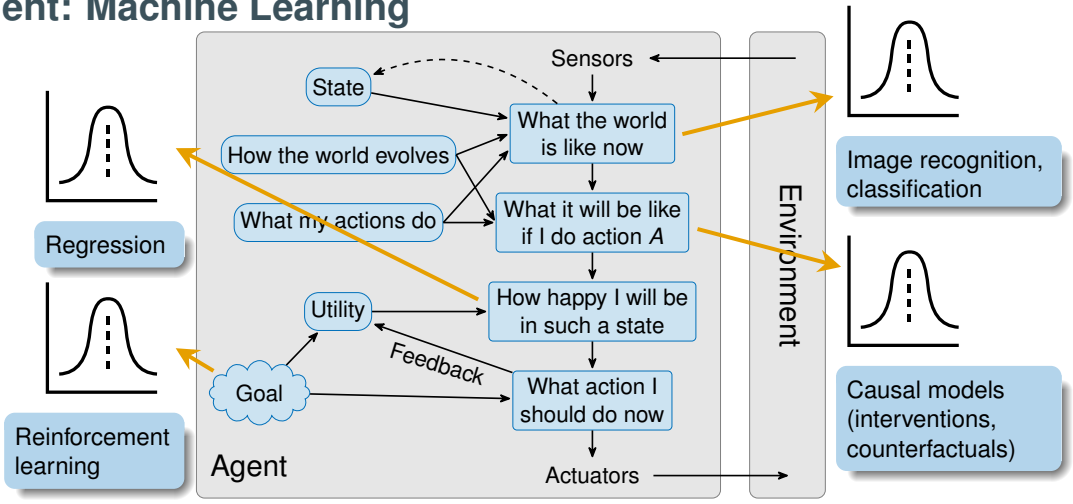
Agent: Machine Learning



Agent: Machine Learning



Agent: Machine Learning



Concept

- Focus on the tool Python
- Application of machine learning and data science
  - Presentation of a selection of methods
- Machine learning and data science methods
  - Problem
  - Idea of the theory
  - Solution via Python and libraries

Clustering

Problem	Clustering, partitioning
Method	k-Means clustering
Type	Unsupervised learning
Hyperparameter	Initial centroids, number of centroids
Python package	SKLearn
Class	`sklearn.cluster.KMeans`

Clustering

Problem	Clustering, partitioning
Method	k-Means clustering
Type	Unsupervised learning
Hyperparameter	Initial centroids, number of centroids
Python package	SKLearn
Class	`sklearn.cluster.KMeans`

- Unsupervised: No labelled data available
  - Find a »meaningful« grouping of the data
  - E.g., by minimising an error function

k-Means

- Form of unsupervised learning
- Search for natural groupings of objects
  - Determine classes directly from data
    - High intra-class similarity
    - Low inter-class similarity

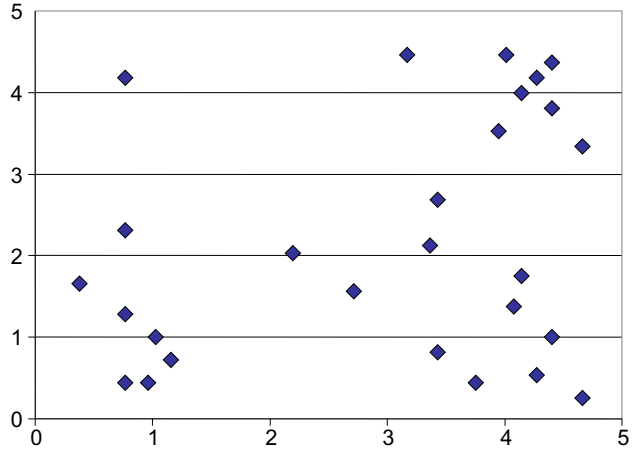
k-Means

- Form of unsupervised learning
- Search for natural groupings of objects
  - Determine classes directly from data
    - High intra-class similarity
    - Low inter-class similarity

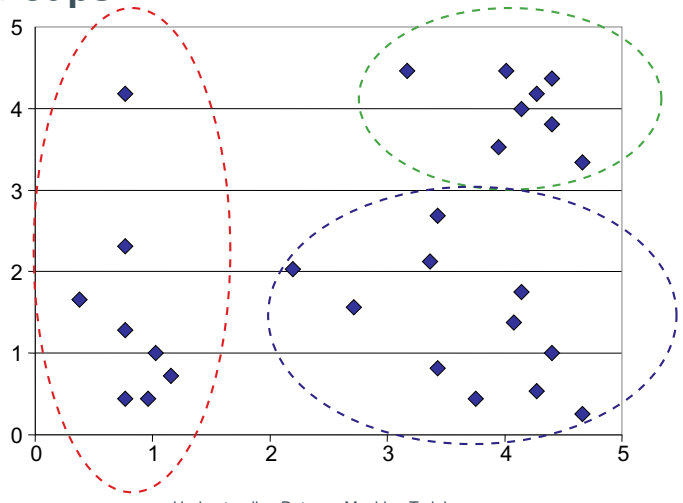
$$\|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Distance measure, e.g., Euclidean distance

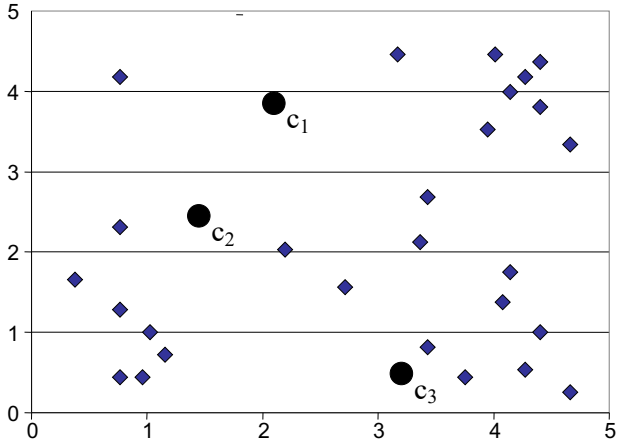
Looking for Groups



Looking for Groups



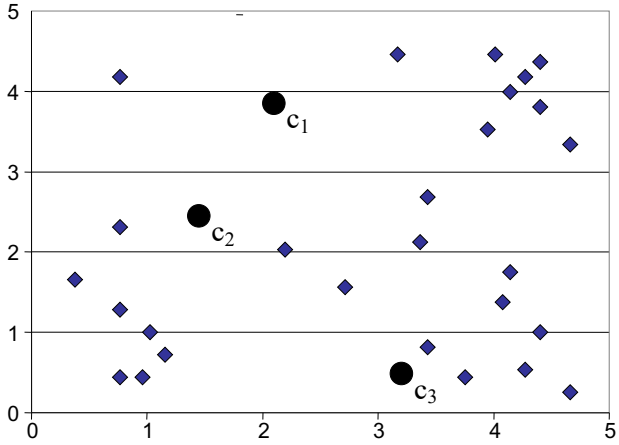
k-Means, $t = 0$



$$C_i^t = \{x_j : \|x_j - c_i^t\|_2 \leq \|x_j - c_r^t\|_2 \text{ for all } r = 1, \ldots, k, r \neq i\}$$

Centroid $i = 1, \ldots, k$ and iteration starting with $t = 0$

k-Means, $t = 0$

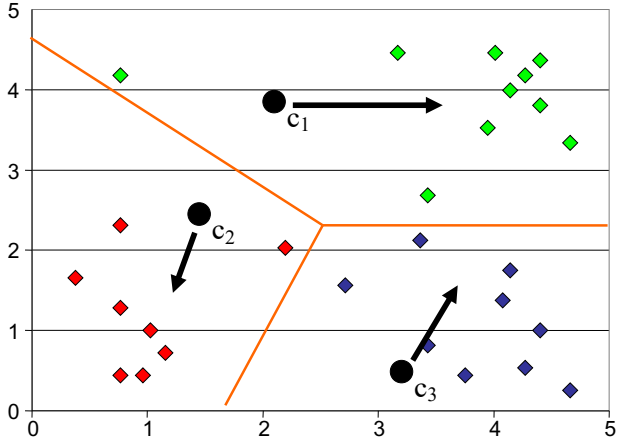


$$C_i^t = \{x_j : \|x_j - c_i^t\|_2 \leq \|x_j - c_r^t\|_2 \text{ for all } r = 1, \ldots, k, r \neq i\}$$

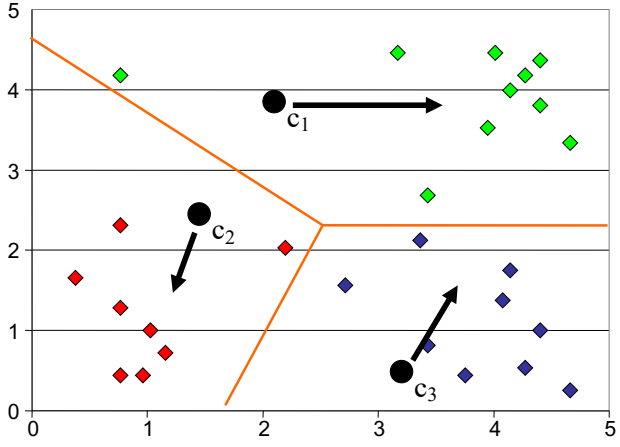
Centroid $i = 1, \ldots, k$ and iteration starting with $t = 0$

- Three initial centroids
- Assignment of each data point to the nearest centroid forms clusters

k-Means, $t = 0 + 1$

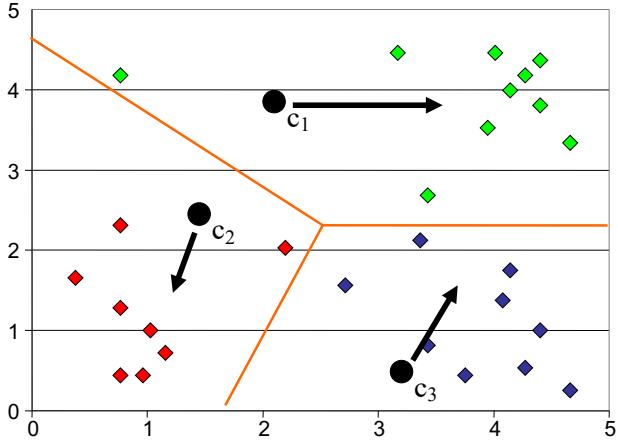


k-Means, $t = 0 + 1$



$$c_i^{t+1} = \frac{1}{|C_i^t|} \sum_{x_j \in C_i^t} x_j$$

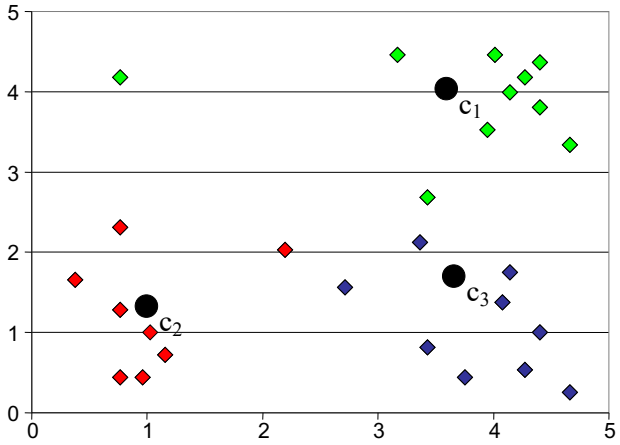
k-Means, $t = 0 + 1$



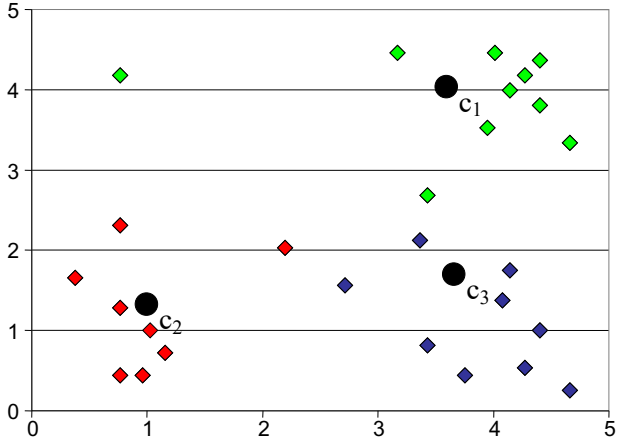
$$c_i^{t+1} = \frac{1}{|C_i^t|} \sum_{x_j \in C_i^t} x_j$$

- Three clusters (marked by colours)
- Move centroids to the center of each cluster

k-Means, $t = 1$



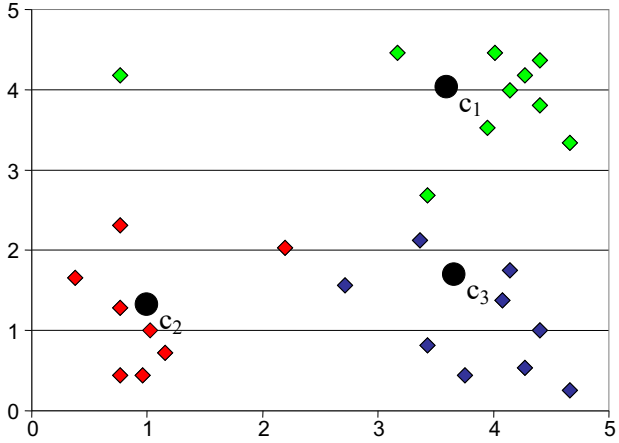
k-Means, $t = 1$



$$C_i^t = \{x_j : \|x_j - c_i^t\|_2 \leq \|x_j - c_r^t\|_2 \text{ for all } r = 1, \ldots, k, r \neq i\}$$

Centroid $i = 1, \ldots, k$ and iteration starting with $t = 0$

k-Means, $t = 1$

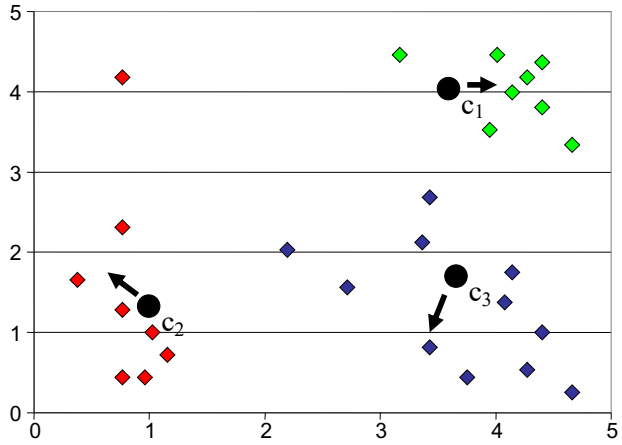


$$C_i^t = \{x_j : \|x_j - c_i^t\|_2 \leq \|x_j - c_r^t\|_2 \text{ for all } r = 1, \ldots, k, r \neq i\}$$

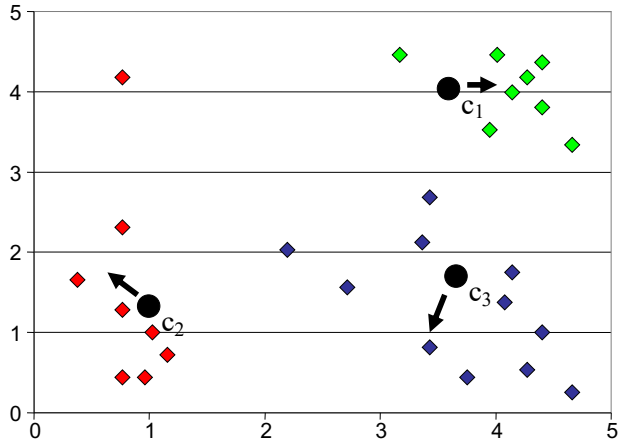
Centroid $i = 1, \ldots, k$ and iteration starting with $t = 0$

- Centroids at new positions
- Reassign data points to centroids, clusters change

k-Means, $t = 1 + 1$

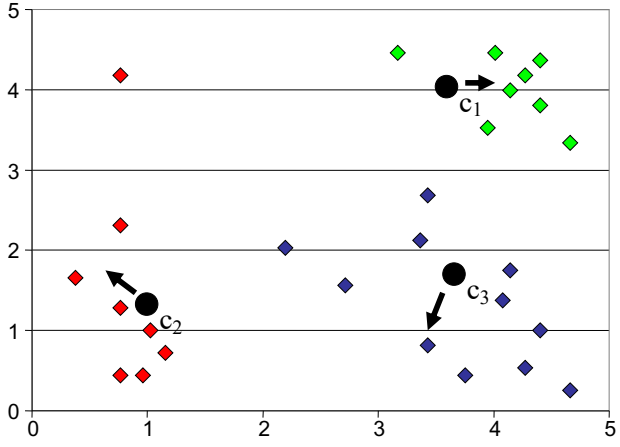


k-Means, $t = 1 + 1$



$$c_i^{t+1} = \frac{1}{|C_i^t|} \sum_{x_j \in C_i^t} x_j$$

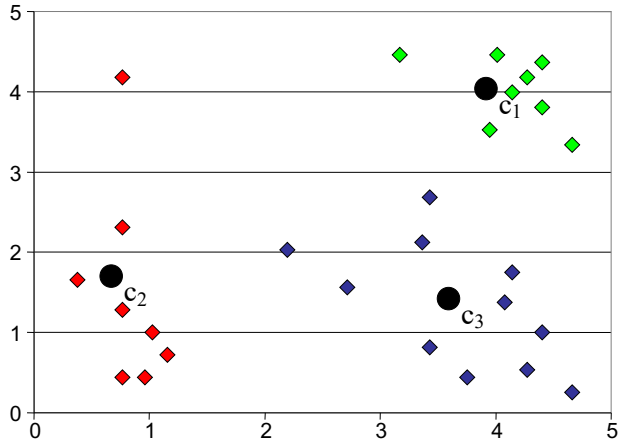
k-Means, $t = 1 + 1$



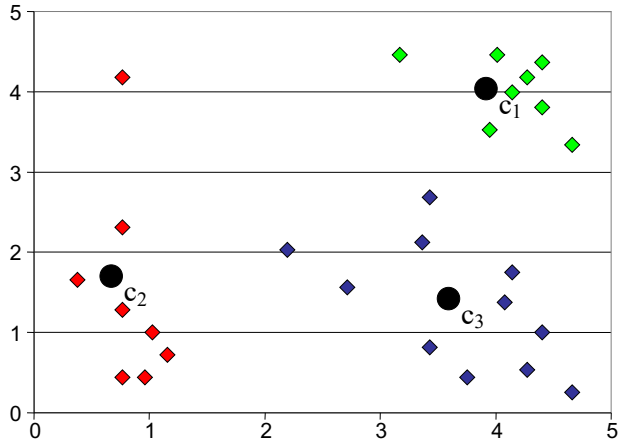
$$c_i^{t+1} = \frac{1}{|C_i^t|} \sum_{x_j \in C_i^t} x_j$$

- Move centroids again to the center of each cluster

k-Means, $t = 2$



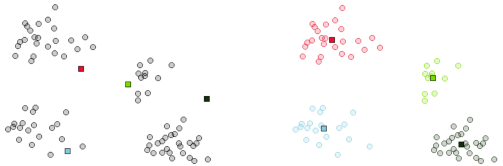
k-Means, $t = 2$



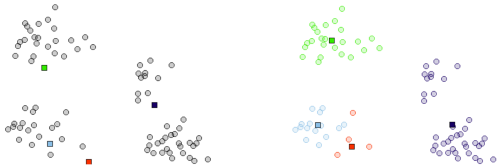
- Centroids again at new positions
- No more changes in cluster assignments, stop algorithm

k-Means – Result Depends on Initialisation

Good result



Bad result



Scikit-Learn (SKLearn) – Installation

- Scikit-Learn (SKLearn) is a Python package
- https://scikit-learn.org/
- Installation, e.g., via `pip3 install scikit-learn`
- Import via

```
from sklearn.cluster import KMeans
```

Scikit-Learn (SKLearn) – Installation

- Scikit-Learn (SKLearn) is a Python package
- https://scikit-learn.org/
- Installation, e.g., via `pip3 install scikit-learn`
- Import via

```
from sklearn.cluster import KMeans
```

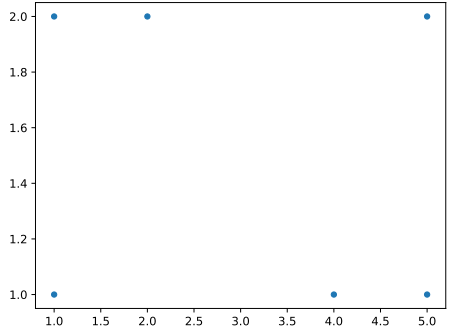
Uses NumPy and C internally and integrates with Matplotlib and Pandas

KMeans Class

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
  [1, 1], [1, 2], [2, 2],
  [4, 1], [5, 1], [5, 2]
])

plt.scatter(X[:, 0], X[:, 1], s=20)
plt.show()

km = KMeans(n_clusters=2, init='random', tol=1e-4)
km.fit(X)

print(km.cluster_centers_)
print(km.labels_)

print(km.predict([[1, 1.5]]))
```

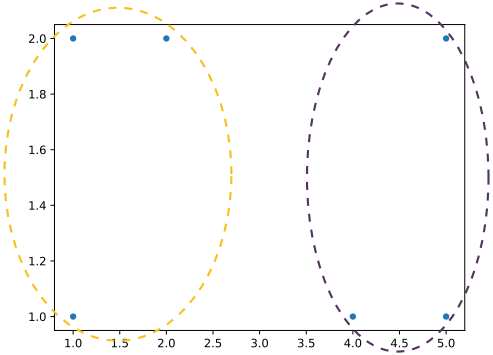
KMeans Class

```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
  [1, 1], [1, 2], [2, 2],
  [4, 1], [5, 1], [5, 2]
])

plt.scatter(X[:, 0], X[:, 1], s=20)
plt.show()

km = KMeans(n_clusters=2, init='random', tol=1e-4)
km.fit(X)

print(km.cluster_centers_)
print(km.labels_)

print(km.predict([[1, 1.5]]))
```



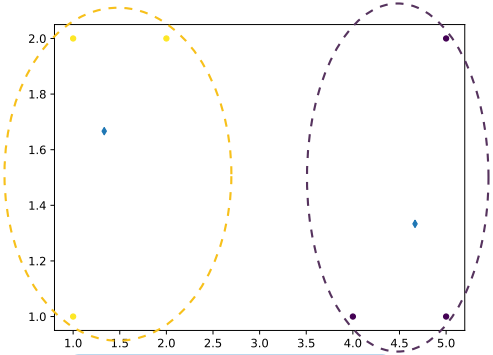
KMeans Class

```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
  [1, 1], [1, 2], [2, 2],
  [4, 1], [5, 1], [5, 2]
])

plt.scatter(X[:, 0], X[:, 1], s=20)
plt.show()

km = KMeans(n_clusters=2, init='random', tol=1e-4)
km.fit(X)

print(km.cluster_centers_)
print(km.labels_)

print(km.predict([[1, 1.5]]))
```



KMeans Class

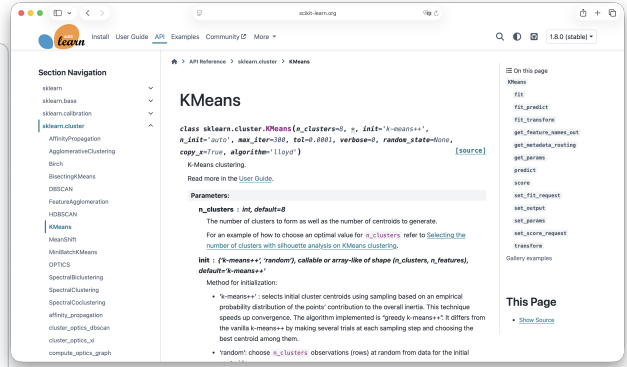
```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
  [1, 1], [1, 2], [2, 2],
  [4, 1], [5, 1], [5, 2]
])

plt.scatter(X[:, 0], X[:, 1], s=20)
plt.show()

km = KMeans(n_clusters=2, init='random', tol=1e-4)
km.fit(X)

print(km.cluster_centers_)
print(km.labels_)

print(km.predict([[1, 1.5]]))
```



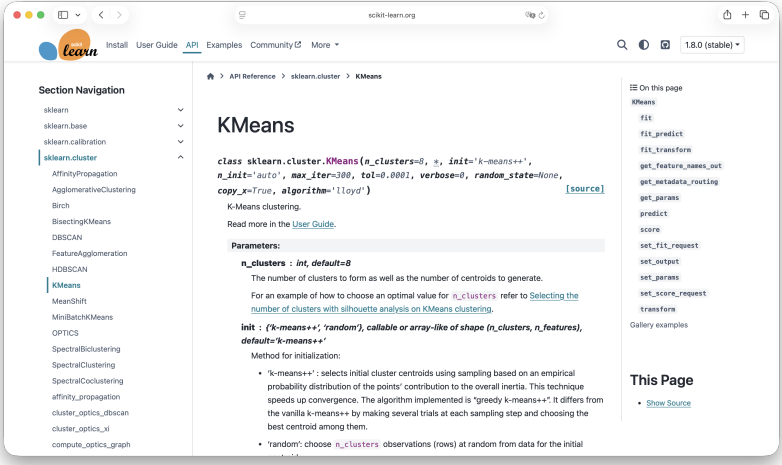

```
$> python3 script.py

[[4.66666667 1.33333333]
[1.33333333 1.66666667]]
[1 1 1 0 0 0]
[1]
```

Method Structure of SKLearn

```
km = sklearn.cluster.KMeans(
  n_clusters=8, init='k-means++',
  n_init='warn', max_iter=300,
  tol=0.0001, verbose=0, random_state=None,
  copy_x=True, algorithm='lloyd'
)

# Attributes
km.cluster_centers_
km.labels_
km.n_iter_

# Methods
fit(X[, y, sample_weight])
fit_predict(X[, y, sample_weight])
fit_transform(X[, y, sample_weight])
get_feature_names_out([input_features])
predict(X[, sample_weight])
transform(X)
```



Method Structure of SKLearn



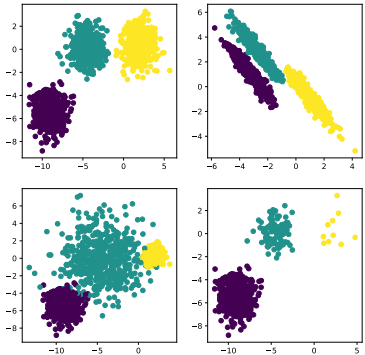
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Example

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X_a, y_a = make_blobs(n_samples=1500, random_state=170)
X_b, y_b = np.dot(X_a, [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]), y_a
X_c, y_c = make_blobs(n_samples=1500, cluster_std=[1.0, 2.5, 0.5], random_state=170)
X_d, y_d = np.vstack((X_a[y_a == 0][:500], X_a[y_a == 1][:100], X_a[y_a == 2][:10])), [0] * 500 + [1] * 100 + [2] * 10

# Plot Data
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
axs[0, 0].scatter(X_a[:, 0], X_a[:, 1], c=y_a)
axs[0, 1].scatter(X_b[:, 0], X_b[:, 1], c=y_b)
axs[1, 0].scatter(X_c[:, 0], X_c[:, 1], c=y_c)
axs[1, 1].scatter(X_d[:, 0], X_d[:, 1], c=y_d)

km_a = KMeans(n_clusters=2, random_state=170).fit(X_a)
km_b = KMeans(n_clusters=3, random_state=170).fit(X_b)
km_c = KMeans(n_clusters=3, random_state=170).fit(X_c)
km_d = KMeans(n_clusters=3, random_state=170).fit(X_d)

# Plot Cluster
```

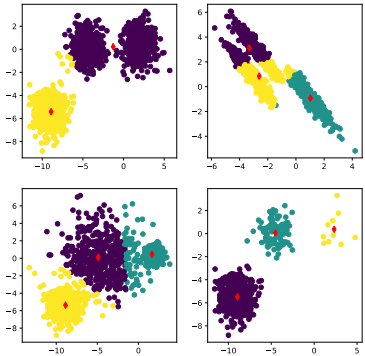
Example

```python
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X_a, y_a = make_blobs(n_samples=1500, random_state=170)
X_b, y_b = np.dot(X_a, [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]), y_a
X_c, y_c = make_blobs(n_samples=1500, cluster_std=[1.0, 2.5, 0.5], random_state=170)
X_d, y_d = np.vstack((X_a[y_a == 0][:500], X_a[y_a == 1][:100], X_a[y_a == 2][:10])), [0] * 500 + [1] * 100 + [2] * 10

# Plot Data
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
axs[0, 0].scatter(X_a[:, 0], X_a[:, 1], c=y_a)
axs[0, 1].scatter(X_b[:, 0], X_b[:, 1], c=y_b)
axs[1, 0].scatter(X_c[:, 0], X_c[:, 1], c=y_c)
axs[1, 1].scatter(X_d[:, 0], X_d[:, 1], c=y_d)

km_a = KMeans(n_clusters=2, random_state=170).fit(X_a)
km_b = KMeans(n_clusters=3, random_state=170).fit(X_b)
km_c = KMeans(n_clusters=3, random_state=170).fit(X_c)
km_d = KMeans(n_clusters=3, random_state=170).fit(X_d)

# Plot Cluster
```

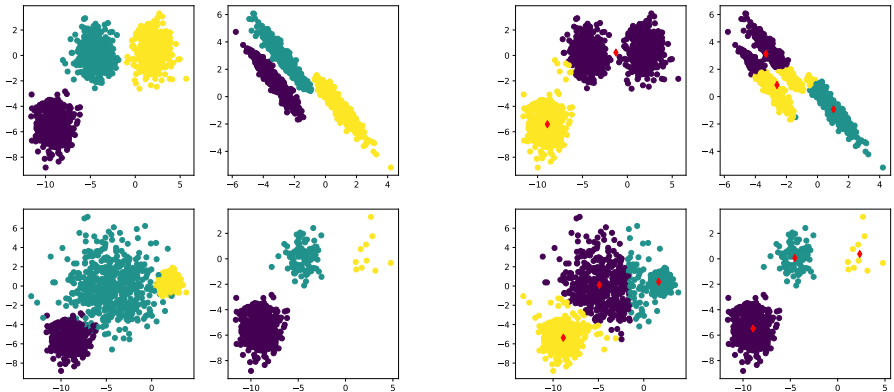


Example

```python
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X_a, y_a = make_blobs(n_samples=1500, random_state=170)
X_b, y_b = np.dot(X_a, [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]), y_a
X_c, y_c = make_blobs(n_samples=1500, cluster_std=[1.0, 2.5, 0.5], random_state=170)
X_d, y_d = np.vstack((X_a[y_a == 0][:500], X_a[y_a == 1][:100], X_a[y_a == 2][:10])), [0] * 500 + [1] * 100 + [2] * 10

# Plot Data
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
axs[0, 0].scatter(X_a[:, 0], X_a[:, 1], c=y_a)
axs[0, 1].scatter(X_b[:, 0], X_b[:, 1], c=y_b)
axs[1, 0].scatter(X_c[:, 0], X_c[:, 1], c=y_c)
axs[1, 1].scatter(X_d[:, 0], X_d[:, 1], c=y_d)

km_a = KMeans(n_clusters=2, random_state=170).fit(X_a)
km_b = KMeans(n_clusters=3, random_state=170).fit(X_b)
km_c = KMeans(n_clusters=3, random_state=170).fit(X_c)
km_d = KMeans(n_clusters=3, random_state=170).fit(X_d)

# Plot Cluster
```



Example



Discussion k-Means

- Usually relatively few steps required
- However, may only find local optimum
- Only applicable if mean is defined
- Based on a specified number of clusters k
- Clusters are usually the same size
- Problems with non-convex shapes

Discussion k-Means

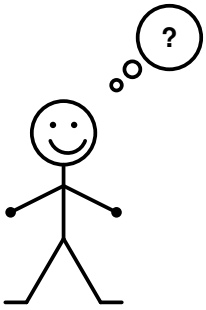
- Usually relatively few steps required
- However, may only find local optimum
- Only applicable if mean is defined

Result

- Based on a specified number of clusters k
- Clusters are usually the same size
- Problems with non-convex shapes

Desired result

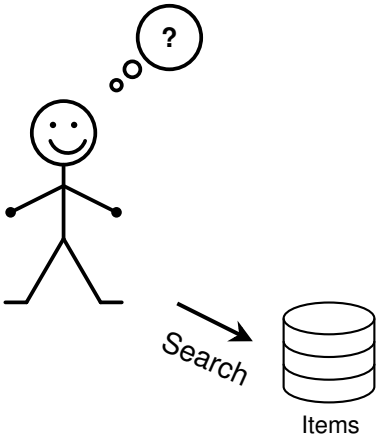
Generating Recommendations



Items

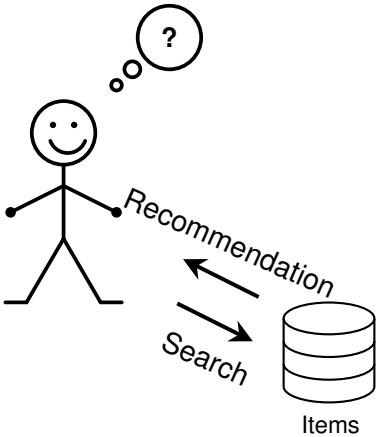
- What are good recommendations?
  - Increase customer satisfaction
  - Increase revenue of vendor

Generating Recommendations



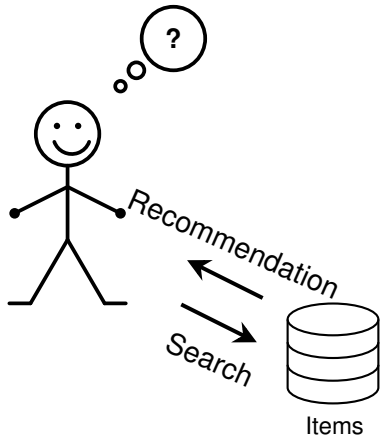
- What are good recommendations?
  - Increase customer satisfaction
  - Increase revenue of vendor

Generating Recommendations



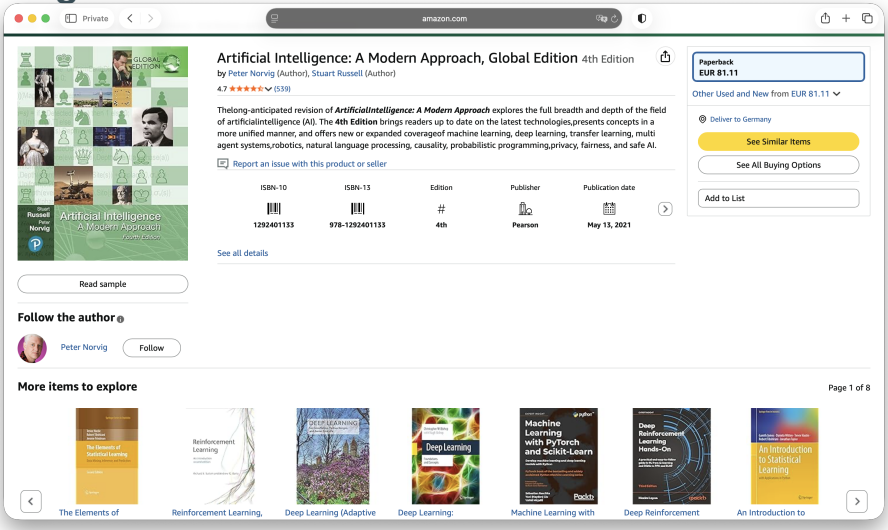
- What are good recommendations?
  - Increase customer satisfaction
  - Increase revenue of vendor

Generating Recommendations

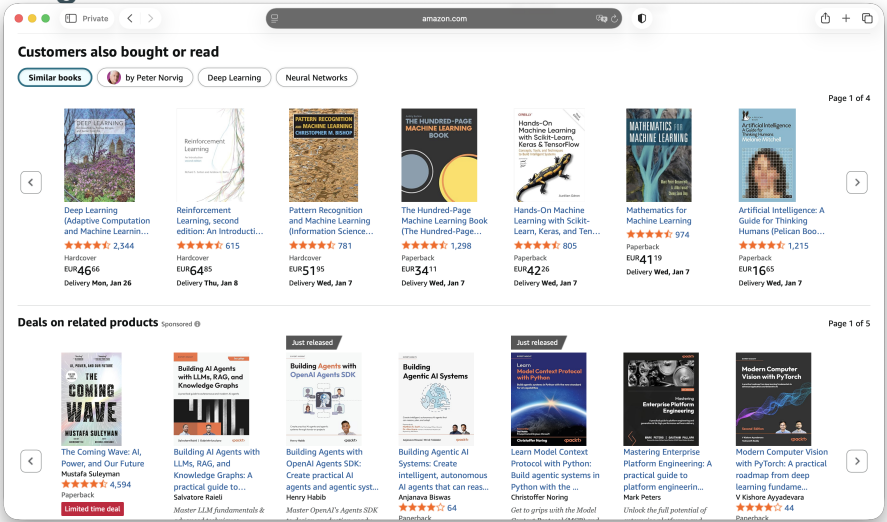


- What are good recommendations?
  - Increase customer satisfaction
  - Increase revenue of vendor
- Predicting how strong a customer's interest in an item is
- Recommend precisely those items from the set of all available items that are of most interest to the customer

Generating Recommendations



Generating Recommendations



Generating Recommendations

- Content-based recommendations
  - Recommend similar items based on item attributes
- Collaborative filtering
  - Recommend items based on the behaviour of similar users

Shopping Baskets

- Given transactions in the form of shopping baskets belonging to different individuals
- Looking for recommendations for further items for each person

TransactionID	CustomerID	Items
1	1	Bread, Oatmeal, Potatoes
2	2	Apples, Bread, Sugar
3	3	Apples, Bread, Potatoes, Sugar
4	2	Bread, Potatoes, Sugar
5	4	Bread, Oatmeal, Potatoes, Sugar

Modelling of Utility

- We are looking for the utility of an item for a customer

Modelling of Utility

- We are looking for the utility of an item for a customer
  - Set of customers / customer IDs C

Modelling of Utility

- We are looking for the utility of an item for a customer
  - Set of customers / customer IDs C
  - Set of available items I

Modelling of Utility

- We are looking for the utility of an item for a customer
  - Set of customers / customer IDs C
  - Set of available items I
  - Utility function $u\colon C \times I \to R$

Modelling of Utility

- We are looking for the utility of an item for a customer
  - Set of customers / customer IDs C
  - Set of available items I
  - Utility function $u\colon C \times I \to R$
  - R is the set of reviews (stars, interval $[0, 1]$, etc.)

Modelling of Utility

- We are looking for the utility of an item for a customer
  - Set of customers / customer IDs C
  - Set of available items I
  - Utility function $u\colon C \times I \to R$
  - R is the set of reviews (stars, interval $[0, 1]$, etc.)
- Assumption: Recommendation of an item with high utility is our goal

Maximising the Utility

»Determine for each customer $c \in C$ the item i'_c from the set of items I, which maximises the utility for the customer c.«

Maximising the Utility

»Determine for each customer $c \in C$ the item i'_c from the set of items I, which maximises the utility for the customer c.«

$$\forall c \in C\colon\ i'_c = \arg\max_{i \in I} u(c, i)$$

Maximising the Utility

»Determine for each customer $c \in C$ the item i'_c from the set of items I, which maximises the utility for the customer c.«

$$\forall c \in C\colon\ i'_c = \arg\max_{i \in I} u(c, i)$$

- Now, we need to determine $u(c, i)$
- Assumption: The item with the highest utility is a good recommendation

Content-Based Utility

i	Item	Weight	Durability	Price
A	Apples	6	4	1
B	Bread	2	1	8
O	Oatmeal	4	4	6
P	Potatoes	9	4	1
S	Sugar	1	10	9

Content-Based Utility

i	Item	Weight	Durability	Price
A	Apples	6	4	1
B	Bread	2	1	8
O	Oatmeal	4	4	6
P	Potatoes	9	4	1
S	Sugar	1	10	9

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}, content(B) = \begin{pmatrix} 2 \\ 1 \\ 8 \end{pmatrix}$$

$$content(O) = \begin{pmatrix} 4 \\ 4 \\ 6 \end{pmatrix}, content(P) = \begin{pmatrix} 9 \\ 4 \\ 1 \end{pmatrix}$$

$$content(S) = \begin{pmatrix} 1 \\ 10 \\ 9 \end{pmatrix}$$

Content-Based Utility

i	Item	Weight	Durability	Price
A	Apples	6	4	1
B	Bread	2	1	8
O	Oatmeal	4	4	6
P	Potatoes	9	4	1
S	Sugar	1	10	9

- Attributes for items are known
- Items are described as vectors

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}, content(B) = \begin{pmatrix} 2 \\ 1 \\ 8 \end{pmatrix}$$

$$content(O) = \begin{pmatrix} 4 \\ 4 \\ 6 \end{pmatrix}, content(P) = \begin{pmatrix} 9 \\ 4 \\ 1 \end{pmatrix}$$

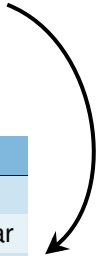
$$content(S) = \begin{pmatrix} 1 \\ 10 \\ 9 \end{pmatrix}$$

Customer Profiles

CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, Bread, Sugar
3	Apples, Bread, Potatoes, Sugar
2	Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar

Customer Profiles

CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, Bread, Sugar
3	Apples, Bread, Potatoes, Sugar
2	Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar



CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, Bread, Bread, Potatoes, Sugar, Sugar
3	Apples, Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar

Customer Profiles

CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, 2x Bread, Potatoes, 2x Sugar
3	Apples, Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar

$$profile(c) = \frac{\sum_{i \in items(c)} content(i)}{|items(c)|}$$

Customer Profiles

CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, 2x Bread, Potatoes, 2x Sugar
3	Apples, Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar

$$profile(c) = \frac{\sum_{i \in items(c)} content(i)}{|items(c)|}$$

$$profile(c_1) = \frac{1}{3}\Big(content(B) + content(O) + content(P)\Big)$$
$$= \frac{1}{3}\left(\begin{pmatrix}2\\1\\8\end{pmatrix} + \begin{pmatrix}4\\4\\6\end{pmatrix} + \begin{pmatrix}9\\4\\1\end{pmatrix}\right) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

Customer Profiles

CustomerID	Items
1	Bread, Oatmeal, Potatoes
2	Apples, 2x Bread, Potatoes, 2x Sugar
3	Apples, Bread, Potatoes, Sugar
4	Bread, Oatmeal, Potatoes, Sugar

$$profile(c) = \frac{\sum_{i \in items(c)} content(i)}{|items(c)|}$$

$$profile(c_1) = \frac{1}{3}\Big(content(B) + content(O) + content(P)\Big)$$

$$= \frac{1}{3}\left(\begin{pmatrix}2\\1\\8\end{pmatrix} + \begin{pmatrix}4\\4\\6\end{pmatrix} + \begin{pmatrix}9\\4\\1\end{pmatrix}\right) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

A profile is a vector of attributes for each customer

Utility Function

- $u(c, i) = \cos(\mathit{profile}(c), \mathit{content}(i))$ (cosine similarity)

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|_2 \|\vec{w}\|_2} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} w_i^2}}$$

Utility Function

- $u(c, i) = \cos(\mathit{profile}(c), \mathit{content}(i))$ (cosine similarity)

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|_2 \|\vec{w}\|_2} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} w_i^2}}$$

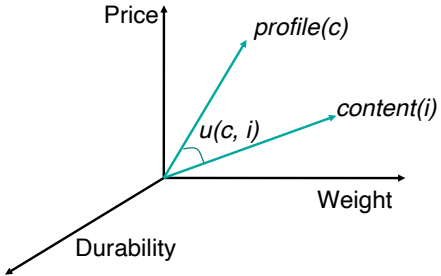
- Items are described as vectors
- Profiles as vectors based on shopping baskets
- Determine angles between items and profiles
  - Utility estimation
- Recommendation if a threshold is reached

Utility Function

- $u(c, i) = \cos(\mathit{profile}(c), \mathit{content}(i))$ (cosine similarity)

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|_2 \|\vec{w}\|_2} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} w_i^2}}$$

- Items are described as vectors
- Profiles as vectors based on shopping baskets
- Determine angles between items and profiles
  - Utility estimation
- Recommendation if a threshold is reached



Example

- New item tomatoes (T) available
- Compute the utility for customer c_1

$$profile(c_1) = \begin{pmatrix} 5 \\ 3 \\ 5 \end{pmatrix}$$

$$content(T) = \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}$$

Example

- New item tomatoes (T) available
- Compute the utility for customer c_1

$$profile(c_1) = \begin{pmatrix} 5 \\ 3 \\ 5 \end{pmatrix}$$

$$content(T) = \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}$$

$$\begin{aligned} u(c_1, T) &= \cos(profile(c_1), content(T)) \\ &= \cos\left(\begin{pmatrix} 5 \\ 3 \\ 5 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}\right) \\ &= \frac{5 \cdot 5 + 3 \cdot 2 + 5 \cdot 1}{\sqrt{5^2 + 3^2 + 5^2}\sqrt{5^2 + 2^2 + 1^2}} \\ &= \frac{36}{7.68 \cdot 5.48} = 0.86 \end{aligned}$$

Implementation in Python I

```
ATTRIBUTES = [
  "weight",
  "durability",
  "price"
]
ARTICLES = {
  "A" : {
    "name" : "Apples",
    "weight" : 6,
    "durability" : 4,
    "price" : 1
  },
  "B" : {
    "name" : "Bread",
    "weight" : 2,
    "durability" : 1,
    "price" : 8
  },
  "O" : {
    "name" : "Oatmeal",
    "weight" : 4,
    "durability" : 4,
    "price" : 6
  },
  "P" : {
    "name" : "Potatoes",
    "weight" : 9,
    "durability" : 4,
    "price" : 1
  },
  "S" : {
    "name" : "Sugar",
    "weight" : 1,
    "durability" : 10,
    "price" : 9
  }
}
TRANSACTIONS = [
  (1, ["B", "O", "P"]),
  (2, ["A", "B", "S"]),
  (3, ["A", "B", "P", "S"]),
  (2, ["B", "P", "S"]),
  (4, ["B", "O", "P", "S"])
]
```

Implementation in Python I

```python
ATTRIBUTES = [
  "weight",
  "durability",
  "price"
]
ARTICLES = {
  "A" : {
    "name" : "Apples",
    "weight" : 6,
    "durability" : 4,
    "price" : 1
  },
  "B" : {
    "name" : "Bread",
    "weight" : 2,
    "durability" : 1,
    "price" : 8
  },
  "O" : {
    "name" : "Oatmeal",
    "weight" : 4,
    "durability" : 4,
    "price" : 6
  },
  "P" : {
    "name" : "Potatoes",
    "weight" : 9,
    "durability" : 4,
    "price" : 1
  },
  "S" : {
    "name" : "Sugar",
    "weight" : 1,
    "durability" : 10,
    "price" : 9
  }
}
TRANSACTIONS = [
  (1, ["B", "O", "P"]),
  (2, ["A", "B", "S"]),
  (3, ["A", "B", "P", "S"]),
  (2, ["B", "P", "S"]),
  (4, ["B", "O", "P", "S"])
]
```

In reality, read from a database (or file)

Implementation in Python II

```python
def content(s:str) -> np.ndarray:
  return np.array([ARTICLES[s][attr] for attr in ATTRIBUTES])

print(content("A"))
```

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}$$

Implementation in Python II

```python
def content(s:str) -> np.ndarray:
  return np.array([ARTICLES[s][attr] for attr in ATTRIBUTES])

print(content("A"))
```

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}$$

```
$> python3 script.py

[6 4 1]
```

Implementation in Python III

```
def profile(c:int) -> np.ndarray:
  v, n = np.zeros(len(ATTRIBUTES)), 0
  for user, articles in TRANSACTIONS:
    if c == user:
      n += len(articles)
      for article in articles:
        v += content(article)

  return v/n

print(profile(1))
```

$$profile(c_1) = \frac{1}{3}\Big(content(B) + content(O) + content(P)\Big) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

Implementation in Python III

```python
def profile(c:int) -> np.ndarray:
  v, n = np.zeros(len(ATTRIBUTES)), 0
  for user, articles in TRANSACTIONS:
    if c == user:
      n += len(articles)
      for article in articles:
        v += content(article)

  return v/n

print(profile(1))
```

```
$> python3 script.py

[5.  3.  5.]
```

$$profile(c_1) = \frac{1}{3}\Big(content(B) + content(O) + content(P)\Big) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

Implementation in Python IV

```
contents = np.vstack((content("A"), content("B"), content("O"),
                      content("P"), content("S")))
profiles = np.vstack((profile(1), profile(2), profile(3), profile(4)))
print(contents, profiles)

from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(profiles, contents))
```

Implementation in Python IV

```
contents = np.vstack((content("A"), content("B"), content("O"),
                      content("P"), content("S")))
profiles = np.vstack((profile(1), profile(2), profile(3), profile(4)))
print(contents, profiles)

from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(profiles, contents))
```

```
$> python3 script.py

[[6 4 1]
[2 1 8]
[4 4 6]
[9 4 1]
[1 10 9]]
[[5.  3.  5.  ]
[3.5 5.  6.  ]
[4.5 4.75 4.75]
[4.  4.75 6.  ]]
```

```
[[0.84049264 0.83066386 0.97883892 0.81536609 0.77201953]
[0.75432084 0.843962 0.99183578 0.67865803 0.93104132]
[0.86216872 0.77051298 0.98238335 0.80270181 0.8686357 ]
[0.77946733 0.84695728 0.99711135 0.71360283 0.90564196]]
```

Implementation in Python IV

```
contents = np.vstack((content("A"), content("B"), content("O"),
                      content("P"), content("S")))
profiles = np.vstack((profile(1), profile(2), profile(3), profile(4)))
print(contents, profiles)

from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(profiles, contents))
```

```
$> python3 script.py

[[6 4 1]
[2 1 8]
[4 4 6]
[9 4 1]
[1 10 9]]
[[5.  3.  5.  ]
[3.5 5.  6.  ]
[4.5 4.75 4.75]
[4.  4.75 6.  ]]
```

```
[[0.84049264 0.83066386 0.97883892 0.81536609 0.77201953]
[0.75432084 0.843962 0.99183578 0.67865803 0.93104132]
[0.86216872 0.77051298 0.98238335 0.80270181 0.8686357 ]
[0.77946733 0.84695728 0.99711135 0.71360283 0.90564196]]
```

- Attributes of all items (rows: items, columns: attributes)
- Profiles of all customers (rows: customers, columns: attributes)
- Utility for each item and customer (rows: customers, columns: items)

Implementation in Python V

```python
def new_article(profiles, article_content):
  utilities = cosine_similarity(profiles, article_content)
  print(utilities)

  best = np.argmax(utilities)
  print("Recommend", article_content, "to user", best+1)

  return best

new_article(profiles, [[5, 2, 1]])
```

Implementation in Python V

```python
def new_article(profiles, article_content):
  utilities = cosine_similarity(profiles, article_content)
  print(utilities)

  best = np.argmax(utilities)
  print("Recommend", article_content, "to user", best+1)

  return best

new_article(profiles, [[5, 2, 1]])
```

```
$> python3 script.py

[[0.85568884]
[0.71462855]
[0.82983331]
[0.75059816]]
Recommend [[5, 2, 1]] to user 1
```

Implementation in Python – Remarks

- The input and processing of data (transactions, articles) is not efficient
  - A database (or Pandas) should be used instead
  - The matrices `profiles` and `contents` could be stored on the hard drive
    - Storing them as NumPy arrays enables efficient handling with `sklearn`

Natural Language Processing (Recap)

- *Bag-of-Words* model
- A document is a bit vector (or frequency vector) of its words
- Texts are represented as vectors (which we can compare, etc.)

Word	Sentence 0	Sentence 1	Sentence 2
access	0	0	1
also	0	1	0
ask	0	1	0
creat	0	0	1
enter	1	0	0
git	0	0	1
github	0	0	1
grant	0	0	1
hidden	0	1	1
link	1	0	0
moodl	1	0	0
public	0	1	0
rememb	0	0	1
repositori	1	1	2
right	0	0	1
user	0	0	1
werkzeugewissarbeiten	0	0	1
whether	0	1	0

Natural Language Processing (Recap)

- *Bag-of-Words* model
- A document is a bit vector (or frequency vector) of its words
- Texts are represented as vectors (which we can compare, etc.)

For cosine similarity and k-Means, we need vectors as inputs!

Word	Sentence 0	Sentence 1	Sentence 2
access	0	0	1
also	0	1	0
ask	0	1	0
creat	0	0	1
enter	1	0	0
git	0	0	1
github	0	0	1
grant	0	0	1
hidden	0	1	1
link	1	0	0
moodl	1	0	0
public	0	1	0
rememb	0	0	1
repositori	1	1	2
right	0	0	1
user	0	0	1
werkzeugewissarbeiten	0	0	1
whether	0	1	0

Unfortunately, it's not that easy...

- Term frequency in a document ignored
- Rarity of a term across all documents ignored
- Length of documents ignored

Unfortunately, it's not that easy...

- Term frequency in a document ignored
- Rarity of a term across all documents ignored
- Length of documents ignored

We are now no longer looking at individual sentences, but at corpora consisting of documents (which in turn consist of sentences and words).

Term Frequency (TF)

- Long documents achieve high word counts
- Normalisation by document length

$$tf_{t,d} = \frac{t_d}{|d|},$$

where

$$\begin{aligned} t &= \text{Term (word)} \\ d &= \text{Document} \\ t_d &= \text{Number of occurrences of term } t \text{ in document } d \\ |d| &= \text{Number of words (length) of document } d \end{aligned}$$

Term Frequency (TF)

- Long documents achieve high word counts
- Normalisation by document length

Does this already suffice?

E.g., term »hello« vs. term »law«

$$tf_{t,d} = \frac{t_d}{|d|},$$

where

t = Term (word)
d = Document
t_d = Number of occurrences of term t in document d
$|d|$ = Number of words (length) of document d

Inverse Document Frequency (IDF)

- Measure of information
- Rarity of a term in the corpus
  - »Hello« frequently occurs and conveys little meaning
  - »Law« rarely occurs as a legal term and thus says a lot

$$idf_t = \log\left(\frac{N}{df_t}\right),$$

where

t = Term (word)
N = Number of documents in the corpus
df_t = Number of documents containing term t

TF.IDF

$$tf.idf_{t,d} = tf_{t,d} \cdot idf_t = \frac{t_d}{|d|} \cdot \log\left(\frac{N}{df_t}\right)$$

- Defined for each term and each document
- Weighting the relevance of each word in the corpus
- Weighting used as weight for word vectors

Implementation Using SKLearn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
  'Hello Alice, I have news for you.',
  'Thank you, I also have news for you.',
  'Hello, I have a letter!',
  'I have news for you.',
]
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

print(vectorizer.get_feature_names_out())
print(vectorizer.transform(corpus).toarray())
```

Implementation Using SKLearn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
  'Hello Alice, I have news for you.',
  'Thank you, I also have news for you.',
  'Hello, I have a letter!',
  'I have news for you.',
]
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

print(vectorizer.get_feature_names_out())
print(vectorizer.transform(corpus).toarray())
```

```
$> python3 script.py

['alice' 'also' 'for' 'have' 'hello' 'letter' 'news' 'thank' 'you']
...
```

Implementation Using SKLearn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
  'Hello Alice, I have news for you.',
  'Thank you, I also have news for you.',
  'Hello, I have a letter!',
  'I have news for you.',
]
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

print(vectorizer.get_feature_names_out())
print(vectorizer.transform(corpus).toarray())
```

```
...
[[0.56648884 0.  0.36158249 0.29561738 0.44662631 0.  0.36158249 0.  0.36158249]
[0.  0.46044436 0.29389567 0.24027897 0.  0.  0.29389567 0.46044436 0.58779134]
[0.  0.  0.  0.37919167 0.5728925 0.72664149 0.  0.  0.  ]
[0.  0.  0.52210862 0.42685801 0.  0.  0.52210862 0.  0.52210862]]
```

Implementation Using SKLearn

```
$> python3 script.py

['alice' 'also' 'for' 'have' 'hello' 'letter' 'news' 'thank' 'you']
[[0.56648884 0.  0.36158249 0.29561738 0.44662631 0.  0.36158249 0.  0.36158249]
[0.  0.46044436 0.29389567 0.24027897 0.  0.  0.29389567 0.46044436 0.58779134]
[0.  0.  0.  0.37919167 0.5728925 0.72664149 0.  0.  0.  ]
[0.  0.  0.52210862 0.42685801 0.  0.  0.52210862 0.  0.52210862]]
```

- »Alice« and »letter« are weighted higher than »have« (which occurs in all documents)
- More details are given in the documentation:
  - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Clustering of Documents

- Example corpus: 20 Newsgroups Dataset
  - Collection of approximately 20 000 newsgroup documents
  - 20 different newsgroups (categories)

Clustering of Documents

```python
import numpy as np
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

# train model
train = load_files(
  "20newsbydate/20news-bydate-train",
  categories=["comp.graphics", "rec.autos"],
  encoding="latin1"
)
X_train, y_train = train.data, train.target

num_labels = np.unique(y_train).shape[0]

print(y_train)
print(X_train[1:2])
```

Clustering of Documents

```python
import numpy as np
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

# train model
train = load_files(
  "20newsbydate/20news-bydate-train",
  categories=["comp.graphics", "rec.autos"],
  encoding="latin1"
)
X_train, y_train = train.data, train.target

num_labels = np.unique(y_train).shape[0]

print(y_train)
print(X_train[1:2])
```

```
$> python3 script.py

[0 1 1 ...  1 0 1]
["From:  keys@starchild.ncsl.nist.gov
(Lawrence B. Keys)\nSubject :  Re:
Alarm systems ...]
```

Training of the Model

(1) Pre-processing (stop words, etc.)

Training of the Model

(1) Pre-processing (stop words, etc.)
(2) Convert corpus to TF.IDF (matrix: number of documents $\times$ words)

Training of the Model

(1) Pre-processing (stop words, etc.)
(2) Convert corpus to TF.IDF (matrix: number of documents $\times$ words)
(3) Compute k-Means clustering

Training of the Model

(1) Pre-processing (stop words, etc.)
(2) Convert corpus to TF.IDF (matrix: number of documents $\times$ words)
(3) Compute k-Means clustering

```
19 tfidf = TfidfVectorizer(stop_words="english")
20 tfidf.fit(X_train)
21
22 kmeans = KMeans(n_clusters=num_labels, n_init="auto", random_state=42)
23 kmeans.fit(tfidf.transform(X_train))
```

Training of the Model

(1) Pre-processing (stop words, etc.)
(2) Convert corpus to TF.IDF (matrix: number of documents $\times$ words)
(3) Compute k-Means clustering

```
19 tfidf = TfidfVectorizer(stop_words="english")
20 tfidf.fit(X_train)
21
22 kmeans = KMeans(n_clusters=num_labels, n_init="auto", random_state=42)
23 kmeans.fit(tfidf.transform(X_train))
```

Pre-processing via NLTK usually obtains better results

First convert the texts to the TF.IDF matrix, then compute the clustering

Evaluation of the Model

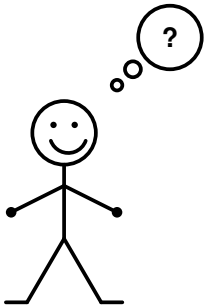
```
25 # test model
26 from sklearn.metrics import accuracy_score
27
28 test = load_files(
29   "20newsbydate/20news-bydate-test",
30   categories=["comp.graphics", "rec.autos"],
31   encoding="latin1"
32 )
33 X_test, y_test = test.data, test.target
34
35 y_prediction = kmeans.predict(tfidf.transform(X_test))
36
37 print(np.unique(y_test == y_prediction, return_counts=True))
38
39 print(accuracy_score(y_test, y_prediction))
```

Evaluation of the Model

```
25 # test model
26 from sklearn.metrics import accuracy_score
27
28 test = load_files(
29   "20newsbydate/20news-bydate-test",
30   categories=["comp.graphics", "rec.autos"],
31   encoding="latin1"
32 )
33 X_test, y_test = test.data, test.target
34
35 y_prediction = kmeans.predict(tfidf.transform(X_test))
36
37 print(np.unique(y_test == y_prediction, return_counts=True))
38
39 print(accuracy_score(y_test, y_prediction))
```

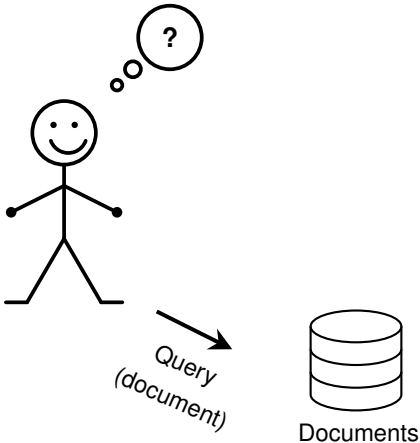
```
$> python3 script.py
(array([False, True]), array([253, 532]))
0.6777070063694267
```

Recommendation of Documents

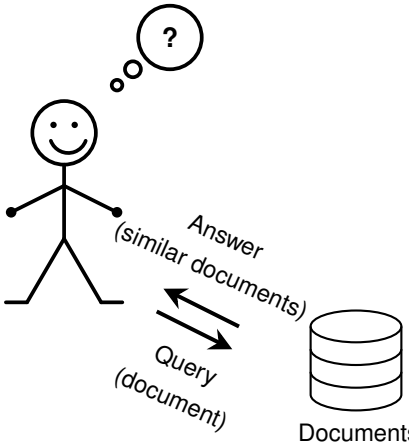


Documents

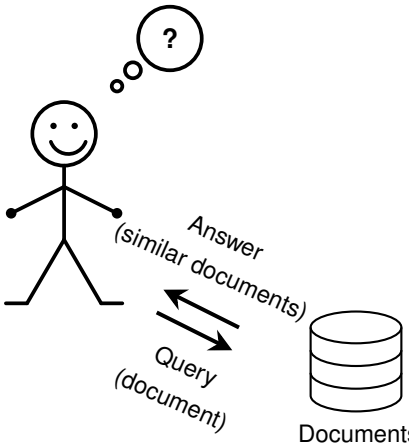
Recommendation of Documents



Recommendation of Documents

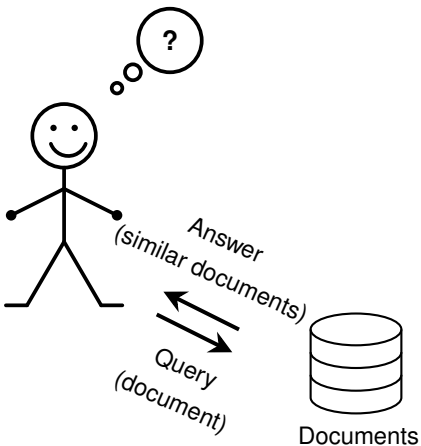


Recommendation of Documents



- Similar to a search engine
  - »Document retrieval«

Recommendation of Documents



- Similar to a search engine
  - »Document retrieval«
- Answering of queries
  - Query: E.g., a document
  - Answer: Similar documents from the corpus

Recommendation of Documents

Code from before (for completeness)

```python
import numpy as np
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import TfidfVectorizer

# data
train = load_files(
  "20newsbydate/20news-bydate-train",
  categories=["comp.graphics", "rec.autos"],
  encoding="latin1"
)
X_train, y_train = train.data, train.target

test = load_files(
  "20newsbydate/20news-bydate-test",
  categories=["comp.graphics", "rec.autos"],
  encoding="latin1"
)
X_test, y_test = test.data, test.target

num_labels = np.unique(y_train).shape[0]

# model
tfidf = TfidfVectorizer(stop_words="english")
X_train_tfidf = tfidf.fit_transform(X_train)
```

Computing Similar Documents

```
# predict (fetch similar documents)
from sklearn.metrics.pairwise import cosine_similarity
from collections import Counter

def fetch_similar_documents(doc:str, top_n:int=10):
  sim = cosine_similarity(tfidf.transform([doc]), X_train_tfidf)
  return (-sim).argsort(axis=None)[:top_n]

for i in [2, 10, 20, 30, 50, 100, 400]:
  best = fetch_similar_documents(X_test[i])
  y_counts = Counter([y_train[b] for b in best])
  print("Document {: >3} (class {}) fetched {: >2} times same class:
      {}".format(
    i, y_test[i], y_counts[y_test[i]], best
  ))
```

Computing Similar Documents

```python
# predict (fetch similar documents)
from sklearn.metrics.pairwise import cosine_similarity
from collections import Counter

def fetch_similar_documents(doc:str, top_n:int=10):
  sim = cosine_similarity(tfidf.transform([doc]), X_train_tfidf)
  return (-sim).argsort(axis=None)[:top_n]

for i in [2, 10, 20, 30, 50, 100, 400]:
  best = fetch_similar_documents(X_test[i])
```

```
$> python3 script.py

Document   2 (class 1) fetched  4 times same class:[ 399 830 25 184 1014 816 214 248 870 251]
Document  10 (class 0) fetched 10 times same class:[ 718 1015 913 1040 154 1131 881 608 653 812]
Document  20 (class 1) fetched 10 times same class:[ 741 1045 907 507 371 1154 1126 573 657 679]
Document  30 (class 1) fetched 10 times same class:[ 865 814 451 611 1106 721 303 1126 993 264]
Document  50 (class 0) fetched  8 times same class:[116 524 396 505 367 73 661 828 480 140]
Document 100 (class 0) fetched  8 times same class:[1053 283 1038 304 684 1077 400 557 924 167]
Document 400 (class 0) fetched  7 times same class:[ 316 499 1057 960 1032 526 900 427 1028 919]
```

Computing Similar Documents

Apply TF.IDF and then use cosine similarity to return the 10 most similar documents (indices)

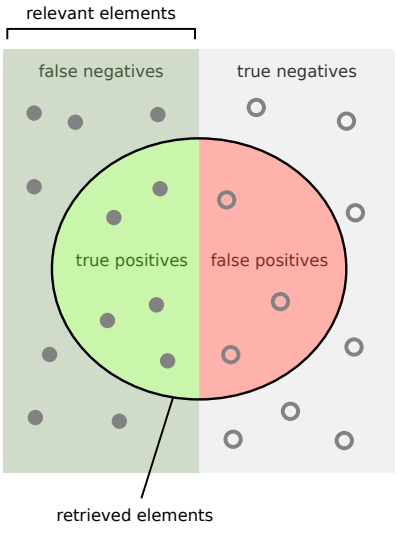
```python
# predict (fetch similar documents)
from sklearn.metrics.pairwise import cosine_similarity
from collections import Counter

def fetch_similar_documents(doc:str, top_n:int=10):
  sim = cosine_similarity(tfidf.transform([doc]), X_train_tfidf)
  return (-sim).argsort(axis=None)[:top_n]

for i in [2, 10, 20, 30, 50, 100, 400]:
  best = fetch_similar_documents(X_test[i])
```

```
$> python3 script.py

Document   2 (class 1) fetched  4 times same class:[ 399 830 25 184 1014 816 214 248 870 251]
Document  10 (class 0) fetched 10 times same class:[ 718 1015 913 1040 154 1131 881 608 653 812]
Document  20 (class 1) fetched 10 times same class:[ 741 1045 907 507 371 1154 1126 573 657 679]
Document  30 (class 1) fetched 10 times same class:[ 865 814 451 611 1106 721 303 1126 993 264]
Document  50 (class 0) fetched  8 times same class:[116 524 396 505 367 73 661 828 480 140]
Document 100 (class 0) fetched  8 times same class:[1053 283 1038 304 684 1077 400 557 924 167]
Document 400 (class 0) fetched  7 times same class:[ 316 499 1057 960 1032 526 900 427 1028 919]
```

Metrics

		Prediction: Positive	Prediction: Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Metrics



https://commons.wikimedia.org/wiki/File:Precisionrecall.svg »Precision and recall« by Walber is licensed under CC BY-SA 4.0

Metrics

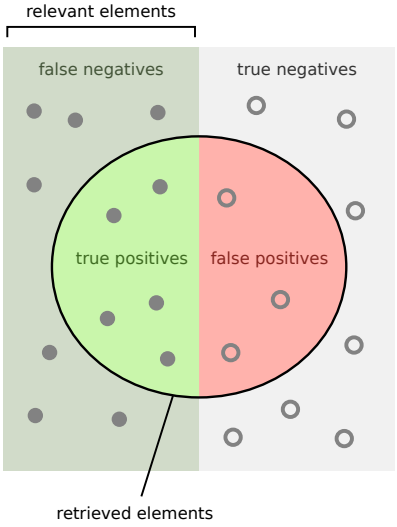
How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

```
sklearn.metrics.precision_score()
sklearn.metrics.recall_score()
sklearn.metrics.precision_recall_fscore_support()
```



Summary

- Terminology
  - Artificial intelligence, machine learning, data science
  - Agents
- Clustering
- Recommendations
- Natural language processing
  - TF.IDF
  - Clustering of documents
  - Recommendation of documents