Unit 12
Clustering, K-Means, NMF, and EM

Prof. Phil Schniter

THE OHIO STATE UNIVERSITY

ECE 4300: Introduction to Machine Learning, Sp20

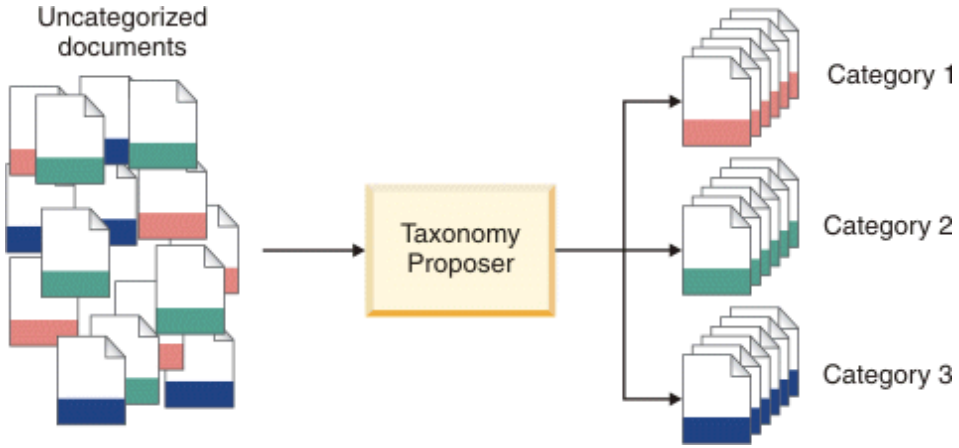
Learning objectives

- Understand the k-means clustering objective and Lloyd's algorithm
- Understand term-document matrices and TF-IDF scores for text-mining
- Understand NMF, its relation to PCA, and application to clustering
- Understand GMMs and their application to clustering
- Understand the EM algorithm and its application to GMM parameter fitting

Outline

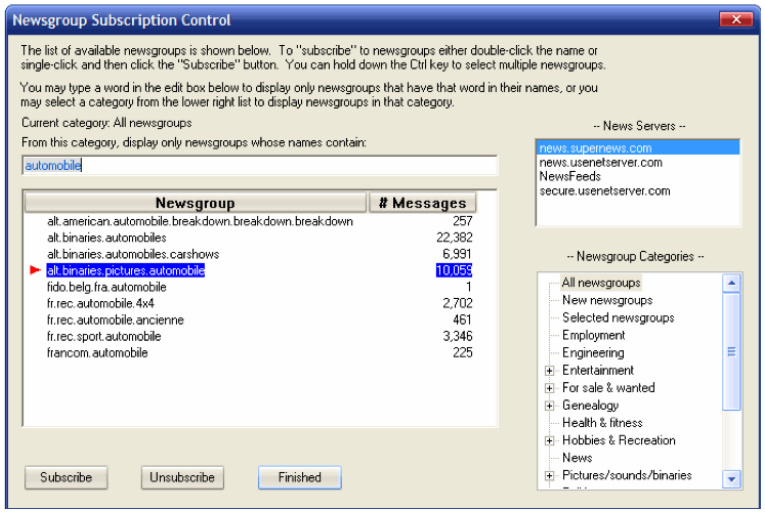
Document clustering

- Say you have a huge corpus of documents
- How do you organize them?
- Idea: Documents can be clustered
  - Grouped into similar categories
  - An example of "text mining"
- How exactly do we do this?



Example: UseNet newsgroup articles

- UseNet was an online discussion forum for various topics
  - Started on university networks in late 1970s
  - Migrated to internet, peaked in 1990s
- Useful for studying document clustering
  - UseNet documents are simple
  - There exists "ground truth": documents have categories



20Newsgroups dataset

- The 20Newsgroups dataset contains . . .
  - data from 20 UseNet categories
  - $\approx$ 1000 documents/category

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

- The 20Newsgroups dataset is available via sklearn
  - We will only load 4 categories, resulting in 3387 samples total

```
remove = ('headers', 'footers')
dataset = fetch_20newsgroups(subset='all', categories=categories,
                             remove = remove, shuffle=True, random_state=42)
print(dataset.target_names)
```

```
['alt.atheism', 'comp.graphics', 'sci.space', 'talk.religion.misc']
```

20Newsgroups dataset (cont.)

- Data is loaded into . . .
  - `dataset.data`: an array of UseNet posts (represented as strings)
  - `dataset.labels`: an array of class labels (i.e., post categories)
  - `dataset.target_names`: the title of each category
- An example UseNet post:

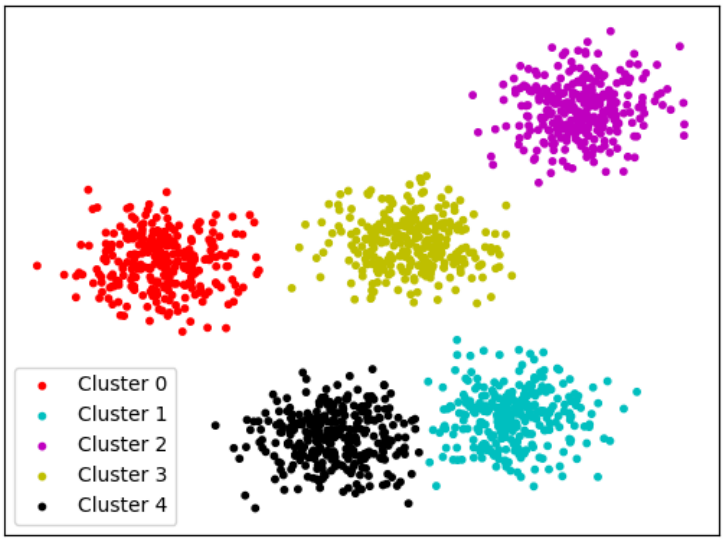
```
doc_ind = 10  # Index of an example document
data_ex = dataset.data[doc_ind]
cat_ex  = dataset.target_names[labels[doc_ind]]
print('Post from {0:s}:'.format(cat_ex))
print()
print(data_ex)
```

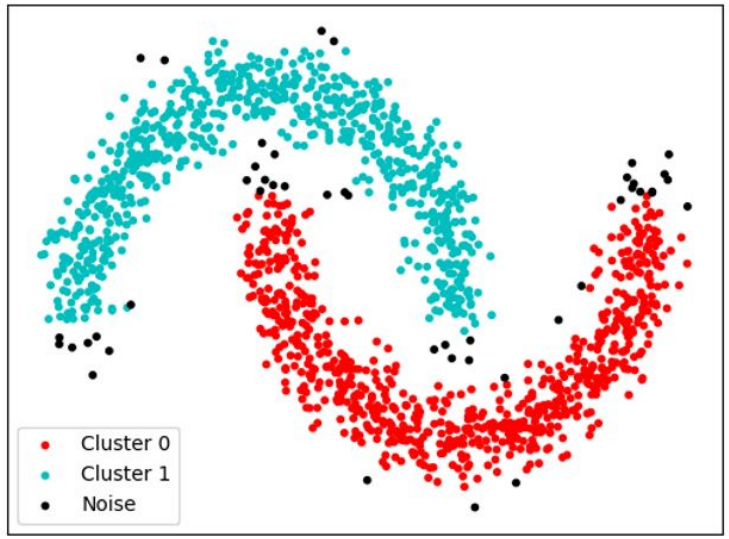
```
Post from comp.graphics:


Hallo POV-Renderers !
I've got a BocaX3 Card. Now I try to get POV displaying True Colors
while rendering. I've tried most of the options and UNIVESA-Driver
but what happens isn't correct.
Can anybody help me ?
```

Outline

The clustering problem

- Say we have a dataset $\{\boldsymbol{x}_i\}_{i=1}^n$ with samples $\boldsymbol{x}_i \in \mathbb{R}^d$
- Goal: Partition the dataset into K groups of "similar" samples
  - Called "clustering"
  - Usually, "similar" means close together, but there are many possible definitions
  - Like PCA, this is an unsupervised learning task; there are no labels





K-means clustering

- Idea: Design K centroids $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K]$ to minimize the RSS cost

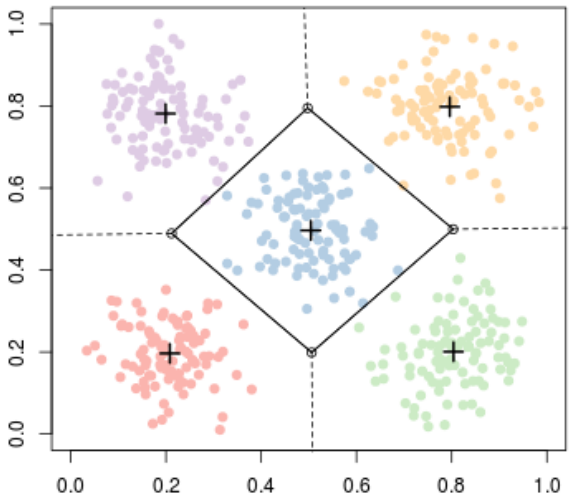
$$J(\boldsymbol{C}) \triangleq \sum_{i=1}^{n} \min_{k=1\ldots K} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|^2$$

  - This cost is the sum-squared distance from each sample to the *closest* centroid

- Define the kth cluster as the samples in $\{\boldsymbol{x}_i\}$ closest to $\boldsymbol{c}_k$
  - The cluster label given to $\boldsymbol{x}_i$ is

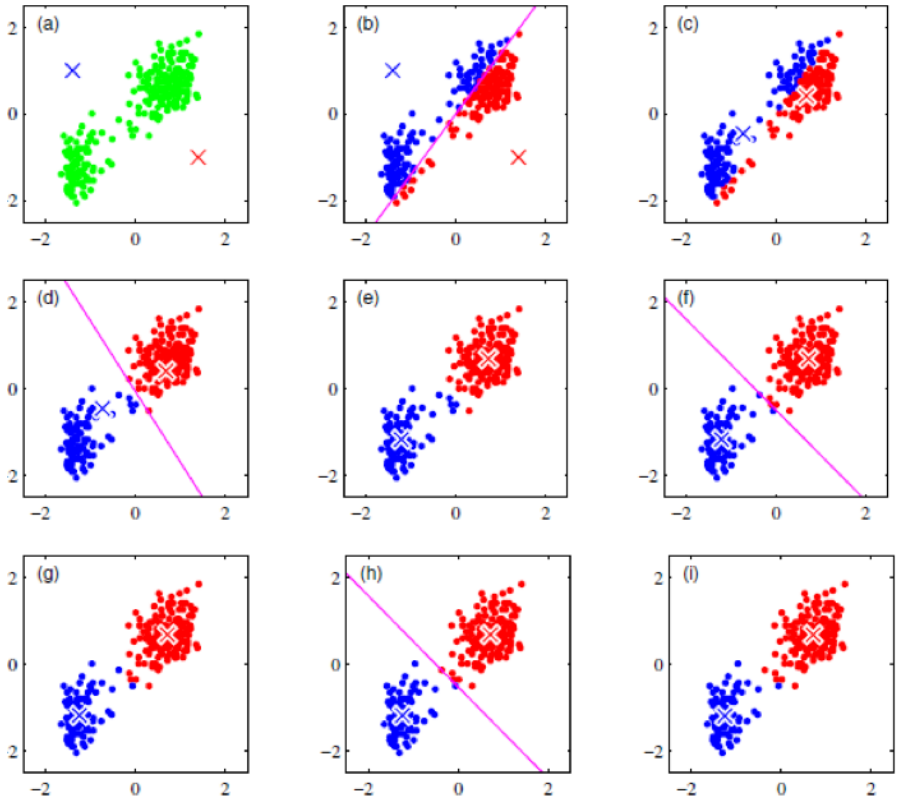
$$\widehat{k}(\boldsymbol{x}_i) \triangleq \arg \min_{k=1\ldots K} \|\boldsymbol{x}_i - \boldsymbol{c}_k\| \in \{1, \ldots, K\}$$

  - The decision regions are Voronoi cells:



Lloyd's algorithm

- Lloyd's algorithm is often used to (approximately) solve the k-means problem
- The algorithm iterates two steps (given some centroid initialization)
  1. For each k: compute sample mean $\boldsymbol{\mu}_k$ of data $\{\boldsymbol{x}_i\}$ within kth Voronoi cell
  2. For each k: Set new centroid at sample mean, i.e., $\boldsymbol{c}_k \leftarrow \boldsymbol{\mu}_k$
- Problem: Can get stuck in local minima
- Remedy: Careful initialization
  - known as k-means++
  - used by default in sklearn.cluster.KMeans



Bishop, *Pattern Recognition and Machine Learning*, 2006

Outline

Bag-of-words model

- To apply k-means to document classification, we must represent the documents *numerically* (i.e., construct features)
- A simple approach is the bag-of-words model:
  - List all words (or "terms")
  - Represent each document as an array of word counts
  - But omit common words (called "stopwords")
- Issues:
  - Some documents are much longer than others
  - Some terms are much more common than others

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List

for
is
of
the
to

TF-IDF features

- So instead use . . .
  - $\mathrm{TF}[i,j] = \dfrac{\#\text{ occurrences of term } j \text{ in doc } i}{\text{total } \#\text{ of terms in doc } i}$, "term frequency"
    - invariant to document size
  - $\mathrm{IDF}[j] = \log\left(\dfrac{\text{total } \#\text{ of docs in corpus}}{\#\text{ docs with term } j \text{ in corpus}}\right)$, "inverse document frequency"
    - penalizes common terms
- The TF-IDF features $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^{\mathsf{T}}$ are usually constructed as

$$X[i,j] = \mathrm{TF}[i,j]\,\mathrm{IDF}[j]$$

  - Used by 83% of text recommender systems today!
  - Note: in much of the literature, $\boldsymbol{X}$ above is transposed
    - We write it as above to be consistent with our earlier lectures
- $\boldsymbol{X}$ is often called a "term-document" matrix

The term-document matrix

- The term-document matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ has many uses
  - Recall $X[i,j]$ is the TF-IDF score of document i and word j
- In document retrieval, the goal is to find which documents are most associated with a given word or words
  - Used for internet search (e.g., Google, Bing)
  - Simple technique: Given word j, extract jth column of $\boldsymbol{X}$, i.e., $\boldsymbol{t}_j \triangleq \boldsymbol{X}\boldsymbol{\delta}_j$
  - The most relevant documents have the highest scores in the "term vector" $\boldsymbol{t}_j$
- In semantic analysis, one wants to find relationships among terms
  - Simple technique: Given words $\{j, j'\}$, compute $\boldsymbol{t}_j^{\mathsf{T}}\boldsymbol{t}_{j'}$ to measure their similarity

Term-document structure

- Challenge: The term-document matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ can be huge!
  - $n = \#$ documents in corpus
  - $d = \#$ terms in vocabulary (38777 even in our toy example!)
- Fortunately, the rows of $\boldsymbol{X}$ are sparse
  - Most documents use only a small fraction of the vocabulary
  - The sparsity of $\boldsymbol{X}$ helps to reduce the memory footprint
- But sometimes $\boldsymbol{X}$ also exhibits low-rank structure that can be used for ...
  - further reducing the memory footprint
  - reducing effects of randomness in the dataset
  - clustering the documents
  - We will discuss this further in the context of LSA and NMF ...

Computing TF-IDF in Python

- To compute TF-IDF features $\boldsymbol{X}$, we use sklearn's TfidfVectorizer method:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english') # remove English stopwords

X = vectorizer.fit_transform(dataset.data)
print("n_samples: %d, n_features: %d" % X.shape)
```

```
n_samples: 3387, n_features: 38777
```

- To do document clustering, we run k-means on the term-document matrix $\boldsymbol{X}$

Example TF-IDF scores

- For an example document, the terms with the highest TF-IDF scores are shown on the right:

```
doc_ind = 10  # Index of an example document
xi = X[doc_ind,:].todense()
term_ind = xi.argsort()[:, ::-1]
xi_sort = xi[0,term_ind]
terms = vectorizer.get_feature_names()

for i in range(30):
    term = terms[term_ind[0,i]]
    tfidf = xi[0,term_ind[0,i]]
    print('{0:20s} {1:f} '.format(term, tfidf))
```

pov	0.417453
hallo	0.314297
bocax3	0.314297
renderers	0.280154
univesa	0.273361
ve	0.216374
displaying	0.215961
rendering	0.206597
options	0.205576
driver	0.199119
happens	0.187562
colors	0.184011
card	0.177312
tried	0.163320
anybody	0.157071
correct	0.155991
isn	0.132214
got	0.127099
try	0.126131
help	0.125774
true	0.124041
determining	0.000000
determinism	0.000000
determininant	0.000000
deterministic	0.000000
determnined	0.000000
determines	0.000000
deterrant	0.000000
determined	0.000000
detest	0.000000

- Note that this is the same `comp.graphics` post that we saw earlier, on page 7

Running k-means

- We can implement k-means using sklearn.cluster.KMeans:

```
np.random.seed(1) # to make repeatable

from sklearn.cluster import KMeans, MiniBatchKMeans
km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1,
                verbose=True)
km.fit(X)
```

```
Initialization complete
Iteration  0, inertia 6387.081
Iteration  1, inertia 3298.568
Iteration  2, inertia 3286.525
Iteration  3, inertia 3281.397
Iteration  4, inertia 3277.493
Iteration  5, inertia 3276.394
Iteration  6, inertia 3276.072
Iteration  7, inertia 3275.957
Iteration  8, inertia 3275.905
Iteration  9, inertia 3275.871
Iteration 10, inertia 3275.855
Iteration 11, inertia 3275.843
Iteration 12, inertia 3275.833
Iteration 13, inertia 3275.831
Converged at iteration 13: center shift 0.000000e+00 within tolerance 9.816505e-09

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=100,
    n_clusters=4, n_init=1, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=True)
```

Result of k-means clustering

- Recall that, in this example, k-means computes $K = 4$ centroids
  - Each centroid is a 38777-length vector of TF-IDF scores
  - Each score corresponds to a particular word
- Let's print the words with the 10 top TF-IDF scores in each centroid:

```
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
for i in range(true_k):
    print("Cluster %d:" % i, end='')
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind], end='')
    print()
```

```
Cluster 0: god jesus people objective sandvik don believe moral say morality
Cluster 1: edu writes article com just think people don like know
Cluster 2: graphics thanks image file files program format know images looking
Cluster 3: space nasa shuttle launch orbit moon edu writes gov just
```

  - We can see some correspondence with the true categories:

    `['alt.atheism', 'comp.graphics', 'sci.space', 'talk.religion.misc']`

  - But it's not perfect, and the category order is not preserved

Confusion matrix

- To better understand the relationship between the k-means clusters and the newsgroups, we can compute a (normalized) confusion matrix $\boldsymbol{C}$:

$$C[l,k] \triangleq \text{the fraction of cluster } k \text{ that came from newsgroup } l$$

```
from sklearn.metrics import confusion_matrix

labelkm = km.labels_
C = confusion_matrix(labels,labelkm)
Csum = np.sum(C,axis=0)
Cnorm = C / Csum[None,:]
with np.printoptions(precision=3, suppress=True):
    print(Cnorm)
```

```
[[0.548 0.342 0.004 0.002]
 [0.    0.165 0.938 0.008]
 [0.002 0.233 0.052 0.989]
 [0.45  0.26  0.006 0.002]]
```

- The confusion matrix shows that . . .
  - cluster $k = 2$ is dominated by posts from `comp.graphics`
  - cluster $k = 3$ is dominated by posts from `sci.space`
  - cluster $k = 0$ is split between `alt.atheism` and `talk.religion.misc`
  - cluster $k = 1$ is very mixed

An example clustering "error"

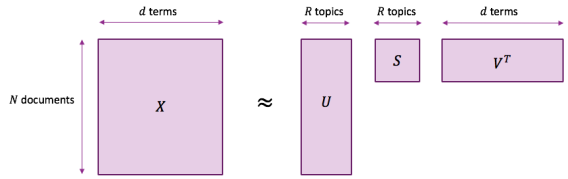
- Recall that cluster $k = 2$ was dominated by `comp.graphics`
- But a small number of `alt.atheism` posts were included
- One such post is on the right
- Not surprising this post was clustered with `comp.graphics`, since it repeatedly uses "color" and "red"

```
Actual newsgroup: alt.atheism
Cluster index: 0

sandvik@newton.apple.com (Kent Sandvik) writes:

>>To borrow from philosophy, you don't truly understand the color red
>>until you have seen it.
>Not true, even if you have experienced the color red you still might
>have a different interpretation of it.

But, you wouldn't know what red *was*, and you certainly couldn't judge
it subjectively.  And, objectivity is not applicable, since you are wanting
to discuss the merits of red.
```

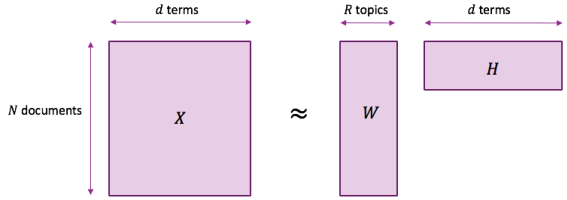
Outline

Topic modeling via low-rank factorization

- By factorizing the $n \times d$ term-document matrix $\boldsymbol{X}$ into a product of $n \times R$ and $R \times d$ matrices with $R \ll r = \text{rank}(\boldsymbol{X})$, we implicitly build an R-topic model.
- There are two common ways to do this:
  1. PCA: $\boldsymbol{X} \approx \boldsymbol{U}_R(\boldsymbol{S}_R\boldsymbol{V}_R^{\mathsf{T}})$ with $\boldsymbol{U}_R \in \mathbb{R}^{n \times R}$ and $(\boldsymbol{S}_R\boldsymbol{V}_R^{\mathsf{T}}) \in \mathbb{R}^{R \times d}$



  2. NMF: $\boldsymbol{X} \approx \boldsymbol{W}_R\boldsymbol{H}_R$ with $\boldsymbol{W}_R \in \mathbb{R}^{n \times R}$ and $\boldsymbol{H}_R \in \mathbb{R}^{R \times d}$



- PCA minimizes the RSS $\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2$. To do so, it builds a "dictionary" $\boldsymbol{U}_R$ with orthonormal columns.
- NMF adds the requirement that $\boldsymbol{W}_R$ and $\boldsymbol{H}_R$ are non-negative.

Latent semantic analysis (PCA revisited!)

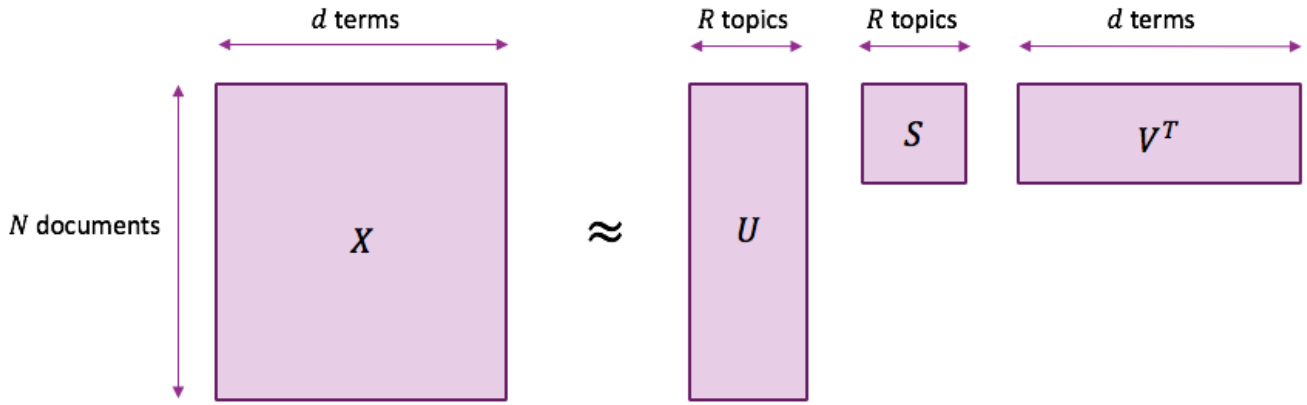
- Recall that the economy SVD decomposes $\boldsymbol{X}$ as follows:

$$\boldsymbol{X} = \boldsymbol{U}_r \boldsymbol{S}_r \boldsymbol{V}_r^{\mathsf{T}} = \sum_{k=1}^{r} \boldsymbol{u}_k s_k \boldsymbol{v}_k^{\mathsf{T}} \quad \text{where} \quad r = \operatorname{rank}(\boldsymbol{X}) \leq \min\{n, d\}$$

- Recall that PCA gives a low-rank approximation of $\boldsymbol{X}$, i.e.,

$$\boldsymbol{X} \approx \boldsymbol{U}_R \, \boldsymbol{S}_R \, \boldsymbol{V}_R^{\mathsf{T}} = \sum_{k=1}^{R} \boldsymbol{u}_k s_k \boldsymbol{v}_k^{\mathsf{T}} \quad \text{for some} \quad R \ll r$$

  - Called latent semantic analysis (LSA) when $\boldsymbol{X}$ is a term-document matrix
  - R acts as the number of "topics"



Latent semantic analysis as topic modeling

- Performing LSA on the term-document matrix $\boldsymbol{X}$ gives

$$\boldsymbol{X} \approx \boldsymbol{U}_R \, \boldsymbol{S}_R \, \boldsymbol{V}_R^{\mathsf{T}} = \sum_{k=1}^{R} \boldsymbol{u}_k s_k \boldsymbol{v}_k^{\mathsf{T}} \quad \text{for some} \quad R \ll \min\{n, d\}$$

  - $\{\boldsymbol{u}_k\}_{k=1}^R$ are the R principal document vectors in the corpus
  - $\{\boldsymbol{v}_k\}_{k=1}^R$ are the R principal term vectors in the corpus
  - $\{s_k\}_{k=1}^R$ are the weights of the R topics in the corpus

- Interpretation:
  - The corpus can be approximated using R topics
  - The kth topic contributes $\boldsymbol{u}_k s_k \boldsymbol{v}_k^{\mathsf{T}}$
  - The principal components are orthogonal to one another

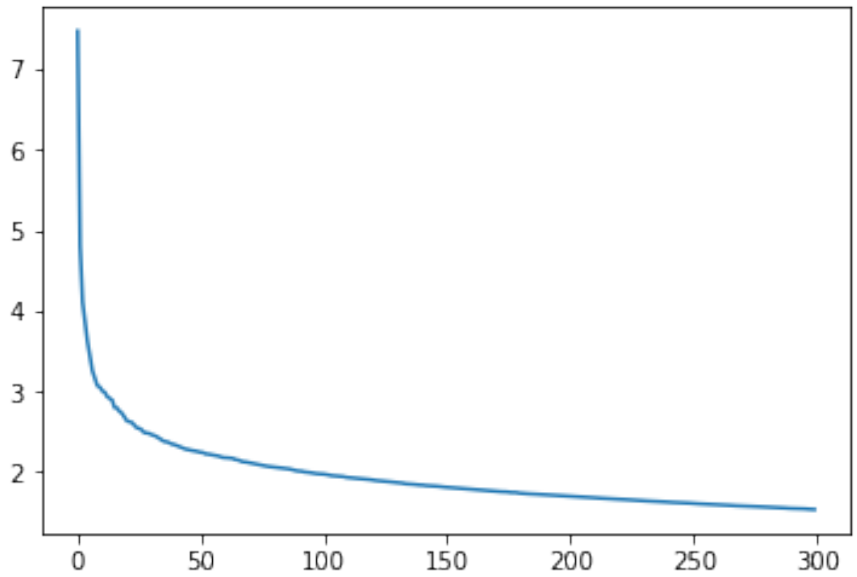
- Challenge:
  - The values in $\boldsymbol{U}_R$ and $\boldsymbol{V}_R$ can be negative
  - How do we interpret the negative values?

LSA on 20Newsgroups

- We now try LSA on the 20Newsgroups data
- The singular values don't quickly decay to zero, meaning that this particular dataset isn't well approximated by a few topics
- For each topic k, the terms $\{j\}$ associated with the largest $|v_{kj}|$ are shown below, but they don't match our 4 newsgroups

```
import scipy.sparse.linalg
U,S,Vt = scipy.sparse.linalg.svds(X,k=300)
```

```
plt.plot(S[::-1])
```



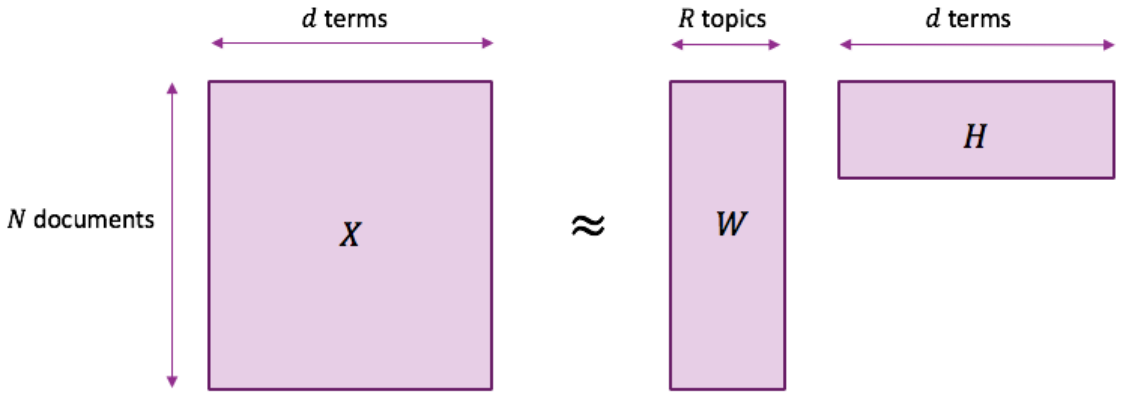

```
PC 0: 3do rh craig post site number vesa terrorist days society
PC 1: allah send anti test ye 3do faq actually dos posted
PC 2: phigs program new psilink p00261 zip universe thing lot send
PC 3: free polygon rle rh phigs read set siggraph convert different
```

Nonnegative matrix factorization

- An alternative low-rank approx is nonnegative matrix factorization (NMF):

$$\boldsymbol{X} \approx \boldsymbol{W}_R \boldsymbol{H}_R \quad \text{for} \quad (\boldsymbol{W}_R, \boldsymbol{H}_R) = \underset{\boldsymbol{W} \in \mathbb{R}_+^{n \times R},\, \boldsymbol{H} \in \mathbb{R}_+^{R \times d}}{\arg\min} \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F^2$$

- For a given number of topics R ...
  - the approximation error is larger than with SVD, but
  - the coefs in $\boldsymbol{W}_R$ and $\boldsymbol{H}_R$ are *interpretable*: larger means more relevant



NMF on 20Newsgroups

- For each topic k, the terms $\{j\}$ yielding the 10 largest h_{kj} coefficients look well clustered!

```
NMF 0: god jesus people believe bible christian don religion edu com
NMF 1: graphics image thanks file files format program gif images ftp
NMF 2: space nasa edu shuttle orbit launch moon writes earth article
NMF 3: objective morality moral edu livesey frank writes keith cobb values
```

- If we cluster documents via $\widehat{k}(\boldsymbol{x}_i) = \arg\max_{k=1,\ldots,R} w_{ik}$, confusion matrix looks good!
  - cluster $k = 1$ dominated by `comp.graphics`
  - cluster $k = 2$ dominated by `sci.space`
  - cluster $k = 0, 3$ mix `alt.altheism` and `talk.religion.misc`

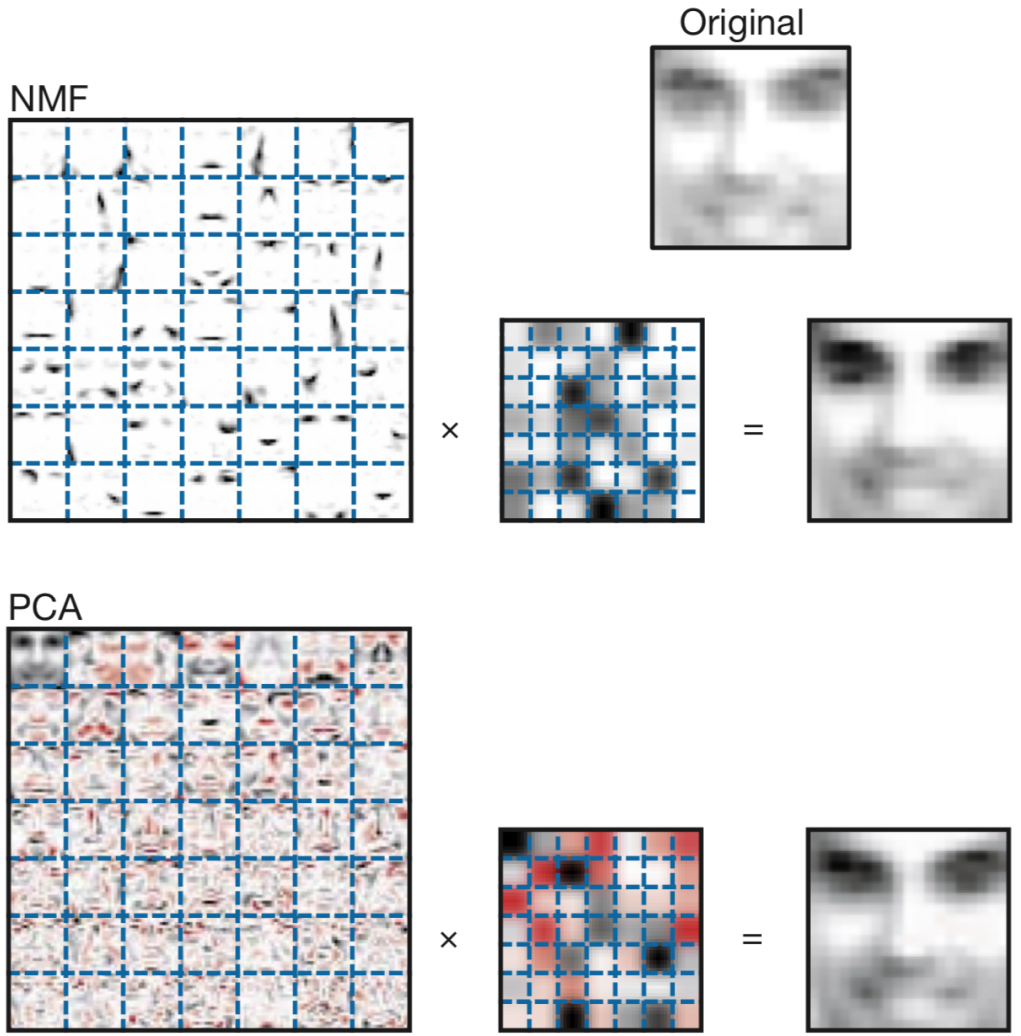
```
labelnmf = np.argmax(W,axis=1)
Cnmf = confusion_matrix(labels,labelnmf)
Csum = np.sum(Cnmf,axis=0)
Cnormnmf = Cnmf / Csum[None,:]
with np.printoptions(precision=3, suppress=True):
    print(Cnormnmf)
```

```
[[0.508 0.012 0.043 0.653]
 [0.014 0.926 0.047 0.035]
 [0.012 0.047 0.85  0.04 ]
 [0.467 0.016 0.059 0.273]]
```

- Original categories: `['alt.atheism', 'comp.graphics', 'sci.space', 'talk.religion.misc']`

NMF versus LSA/PCA

- NMF approximates $\boldsymbol{X}$ by a *non-negative* sum of *non-negative* components
  - Yields a "parts based" representation
- LSA/PCA approximates $\boldsymbol{X}$ by a sum of *orthonormal* components
  - Allows subtractive combinations
  - Negative values shown as red on right



Lee and Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, 1999

NMF versus LSA/PCA

$$\boldsymbol{X} \overset{\text{NMF}}{\approx} \boldsymbol{W}_R \boldsymbol{H}_R \qquad \text{versus} \qquad \boldsymbol{X} \overset{\text{PCA}}{\approx} \boldsymbol{U}_R \boldsymbol{S}_R \boldsymbol{V}_R^{\mathsf{T}}$$

- Both are low-rank approximations
- NMF designed for non-negative $\boldsymbol{X}$ while PCA allows any $\boldsymbol{X}$
- NMF gives non-negative $\boldsymbol{W}_R, \boldsymbol{H}_R$ while PCA can have negative $\boldsymbol{U}_R, \boldsymbol{V}_R$
- NMF gives non-orthogonal $\boldsymbol{W}_R, \boldsymbol{H}_R$ while PCA gives orthogonal $\boldsymbol{U}_R, \boldsymbol{V}_R$
- NMF encourages sparse $\boldsymbol{W}_R, \boldsymbol{H}_R$ while PCA gives dense $\boldsymbol{U}_R, \boldsymbol{V}_R$
- NMF is non-unique while PCA is unique
- NMF solvers can get stuck in local minima, while PCA solvers don't

See http://www.cs.rochester.edu/~jliu/CSC-576/NMF-tutorial.pdf

Outline

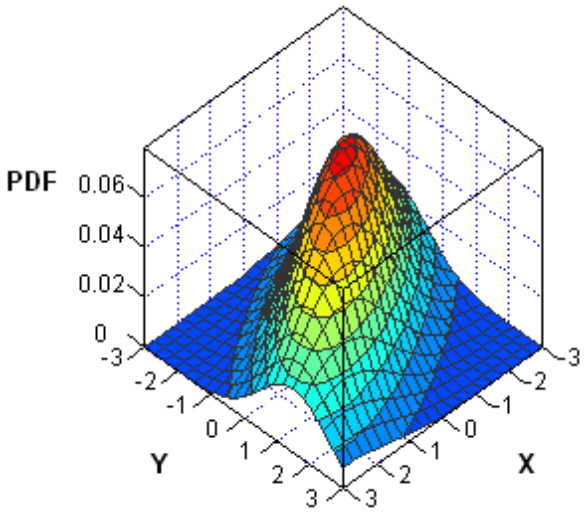
- Motivating Example: Document Clustering
- Clustering and K-Means
- Text Mining with Bag-of-Words, TF-IDF, and K-Means
- Clustering via Low-Rank Models: LSA and NMF
- Gaussian Mixture Models (GMMs)
- Expectation-Maximization (EM) Fitting of GMMs
- Other Clustering Methods

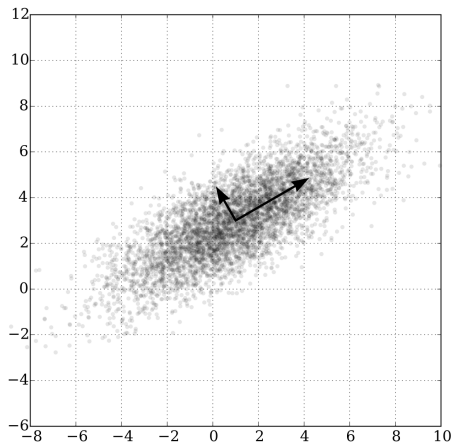
Multivariate Gaussian distribution

- Consider a multivariate Gaussian random vector $\boldsymbol{x}$
- If $\boldsymbol{x}$ has mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{Q}$, then its pdf takes the form

$$p(\boldsymbol{x}) = \underbrace{|2\pi\boldsymbol{Q}|^{-1/2}}_{\text{scale factor}} \underbrace{e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{Q}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}}_{\text{exponentiated quadratic form}} \triangleq \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{Q})$$

- The eigendecomp $\boldsymbol{Q} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}}$ determines the shape of the pdf & scatterplot
  - Contours are ellipsoidal, with principle axis $\boldsymbol{v}_k$ stretched by $\lambda_k^{-1/2}$ for $k = 1 \dots K$





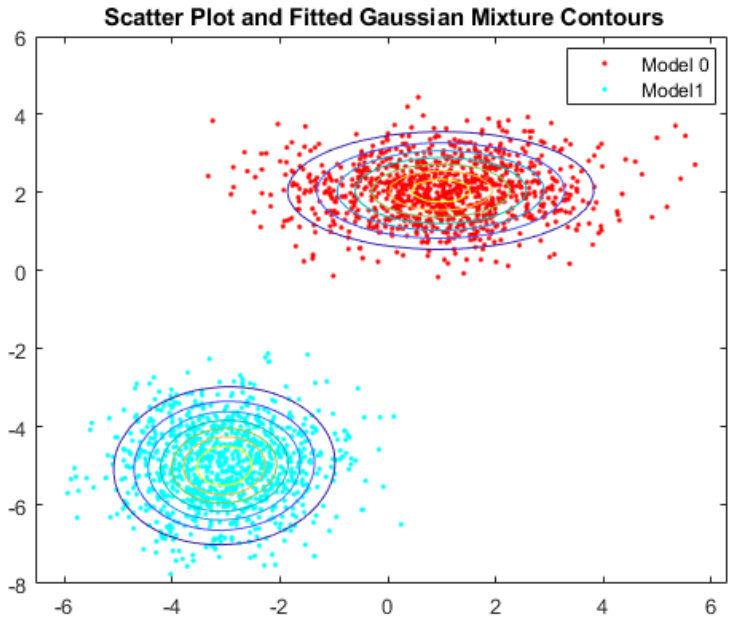
Gaussian mixture models

- A Gaussian mixture model (GMM) is a weighted sum of K Gaussian pdfs:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \gamma_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{Q}_k) \quad \text{with} \quad \gamma_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} \gamma_k = 1$$

  - The kth pdf has mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{Q}_k$
  - The weight γ_k controls the strength of the kth component

- A GMM can be used to cluster!
  - First, fit the GMM parameters $\{\boldsymbol{\mu}_k, \boldsymbol{Q}_k, \gamma_k\}$ to the dataset $\{\boldsymbol{x}_i\}_{i=1}^n$
  - Then, for a test $\boldsymbol{x}$, cluster by determining which GMM component k it "belongs to"
  - This latter step is better understood by writing the GMM in a hierarchical form ...



A hierarchical view of the GMM

- Consider a two-stage approach to generating a random vector $\boldsymbol{x}$:
  1. draw component index $\kappa \in \{1, \ldots, K\}$ according to pmf $[\gamma_1, \ldots, \gamma_K]$
     - Thus $\Pr\{\kappa = k\} = \gamma_k$
  2. then draw $\boldsymbol{x}$ according to pdf $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_\kappa, \boldsymbol{Q}_\kappa)$
     - Thus $p(\boldsymbol{x} \,|\, \kappa = k) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{Q}_k)$
- From the law of total probability,

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} p(\boldsymbol{x}, \kappa = k) = \sum_{k=1}^{K} p(\boldsymbol{x} \,|\, \kappa = k) \Pr\{\kappa = k\} = \sum_{k=1}^{K} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{Q}_k) \gamma_k$$

  - This matches our GMM!
  - So, can think of GMM as a two-level hierarchy: first draw κ, then draw $\boldsymbol{x}|\kappa$
  - Because κ doesn't explicitly appear in $p(\boldsymbol{x})$, it's called a "hidden variable"

Inferring the hidden variable

- Recall the hierarchical form of the GMM:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \underbrace{\Pr\{\kappa = k\}}_{\gamma_k} \underbrace{p(\boldsymbol{x} \,|\, \kappa = k)}_{\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{Q}_k)}$$

- Assigning $\boldsymbol{x}$ to a cluster is equivalent to inferring membership κ from $\boldsymbol{x}$:
  - Given $\boldsymbol{x}$, the probability that membership κ equals some value k is

$$\Pr\{\kappa = k \,|\, \boldsymbol{x}\} = \frac{p(\boldsymbol{x} \,|\, \kappa = k) \Pr\{\kappa = k\}}{p(\boldsymbol{x})} \quad \text{via Bayes rule}$$

$$= \frac{\gamma_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{Q}_k)}{\sum_{k'} \gamma_{k'} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{k'}, \boldsymbol{Q}_{k'})} \quad \text{"soft" assignment}$$

  - A "hard" assignment of $\boldsymbol{x}$ would then be

$$\widehat{k}(\boldsymbol{x}) = \arg \max_{k=1\ldots K} \Pr\{\kappa = k \,|\, \boldsymbol{x}\}$$

Summary of GMM-based clustering

- GMM-based clustering of $\{\boldsymbol{x}_i\}_{i=1}^n$ is done in two steps:
  - Fit the GMM parameters $\boldsymbol{\theta} \triangleq \{\gamma_k, \boldsymbol{\mu}_k, \boldsymbol{Q}_k\}_{k=1}^K$ to the data $\{\boldsymbol{x}_i\}_{i=1}^n$
  - For each i, compute soft assignment $\Pr\{\kappa_i = k \,|\, \boldsymbol{x}_i\} = \dfrac{\gamma_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{Q}_k)}{\sum_{k'} \gamma_{k'} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{k'}, \boldsymbol{Q}_{k'})}$
  - For each i, compute hard assignment $\widehat{k}(\boldsymbol{x}_i) = \arg\max_{k=1\ldots K} \Pr\{\kappa_i = k \,|\, \boldsymbol{x}_i\}$
- To fit the GMM parameters, ideally use maximum-likelihood (ML) estimation:

$$\widehat{\boldsymbol{\theta}}_{\text{ml}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X} \,|\, \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \big\{ \ln p(\boldsymbol{X} \,|\, \boldsymbol{\theta}) \big\}$$

$$\ln p(\boldsymbol{X} \,|\, \boldsymbol{\theta}) = \sum_{i=1}^n \ln \left(\sum_{k=1}^K \gamma_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{Q}_k) \right) \quad \text{assuming i.i.d. } \{\boldsymbol{x}_i\}$$

  - But non-convex, so often difficult to solve!
  - Usually we compute an approximate solution using the EM algorithm ...

Outline

Expectation maximization (EM)

- The EM algorithm is an iterative approximation to (generic) ML estimation:
  - guaranteed to converge to a local maximum of likelihood function
  - but need a good initialization to avoid bad local maxima (if they exist)
- Main idea: Identify some hidden variables $\boldsymbol{Z}$ that simplify the estimation. Then alternate between estimating $\boldsymbol{Z}$ and maximizing $\boldsymbol{\theta}$:

$$\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}\big\{\ln p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})\big\} \qquad \text{“E step”}$$

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \qquad \text{“M step”}$$

- We say “estimate $\boldsymbol{Z}$” because its posterior $p(\boldsymbol{Z}\,|\,\boldsymbol{X},\boldsymbol{\theta}^{(t)})$ is used in the E-step:

$$\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \int p(\boldsymbol{Z}\,|\,\boldsymbol{X},\boldsymbol{\theta}^{(t)}) \ln p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{Z}$$

  - Note: if $p(\boldsymbol{Z}\,|\,\boldsymbol{X},\boldsymbol{\theta}^{(t)}) = \delta(\boldsymbol{Z} - \widehat{\boldsymbol{Z}}^{(t)})$, then $\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \ln p(\boldsymbol{X},\widehat{\boldsymbol{Z}}^{(t)};\boldsymbol{\theta})$
  - More generally, $\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ averages $\ln p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})$ over all *probable* values of $\boldsymbol{Z}$

EM for GMM fitting: E-step

- As hidden variables, we choose $\kappa_i \in \{1, \ldots, K\}$ for each $\boldsymbol{x}_i$, i.e., $\boldsymbol{Z} = [\kappa_1 \ldots \kappa_n]$
- At iteration t, the E-step computes

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &= \int p(\boldsymbol{\kappa} \,|\, \boldsymbol{X}, \boldsymbol{\theta}^{(t)}) \ln p(\boldsymbol{X}, \boldsymbol{\kappa}; \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\kappa} \\
&= \sum_i \int p(\kappa_i \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}^{(t)}) \, \ln p(\boldsymbol{x}_i, \kappa_i; \boldsymbol{\theta}) \, \mathrm{d}\kappa_i \\
&= \sum_i \sum_k \underbrace{\Pr\{\kappa_i = k \,|\, \boldsymbol{x}_i, \boldsymbol{\theta}^{(t)}\}}_{\triangleq \gamma_{ki}^{(t)}} \ln \underbrace{p(\boldsymbol{x}_i, \kappa_i = k; \boldsymbol{\theta})}_{p(\boldsymbol{x}_i|\kappa_i=k;\boldsymbol{\theta}) \Pr\{\kappa_i=k;\boldsymbol{\theta}\}}
\end{aligned}$$

where

$$\gamma_{ki}^{(t)} = \frac{\gamma_k^{(t)} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{Q}_k^{(t)})}{\sum_{k'} \gamma_{k'}^{(t)} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{k'}^{(t)}, \boldsymbol{Q}_{k'}^{(t)})} \quad \text{can be computed } \forall k, i$$

$$p(\boldsymbol{x}_i|\kappa_i = k; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{Q}_k)$$

$$\Pr\{\kappa_i = k; \boldsymbol{\theta}\} = \gamma_k$$

EM for GMM fitting: M-step

- At iteration t, the M-step computes

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \\ &= \arg\max_{\boldsymbol{\theta}} \left\{ \sum_{i,k} \gamma_{ki}^{(t)} \ln \gamma_k + \sum_{i,k} \gamma_{ki}^{(t)} \ln \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{Q}_k) \right\}\end{aligned}$$

- To maximize over γ_k s.t. $\sum_k \gamma_k = 1$, we zero the gradient of the Lagrangian:

$$0 = \frac{\partial}{\partial \gamma_k} \left\{ \ln \gamma_k \sum_i \gamma_{ki}^{(t)} + \lambda \Big(1 - \sum_{k'} \gamma_{k'} \Big) \right\} = \frac{1}{\gamma_k} \sum_i \gamma_{ki}^{(t)} - \lambda$$

$$\Rightarrow 0 = \sum_i \gamma_{ki}^{(t)} - \lambda \gamma_k \tag{1}$$

$$\Rightarrow 0 = \sum_i \sum_k \gamma_{ki}^{(t)} - \lambda \sum_k \gamma_k = \sum_i 1 - \lambda = n - \lambda$$

$$\Rightarrow \lambda = n, \quad \text{and so now we know the value of the Lagrange multiplier } \lambda.$$

EM for GMM fitting: M-step (cont.)

- Plugging λ back into (1) we find

$$\gamma_k = \frac{1}{\lambda}\sum_i \gamma_{ki}^{(t)} = \frac{1}{n}\sum_i \gamma_{ki}^{(t)}, \quad \text{which we use for } \gamma_k^{(t+1)}$$

- To maximize over $\boldsymbol{\mu}_k$, we zero the gradient of the cost w.r.t. $\boldsymbol{\mu}_k$:

$$\begin{aligned}
\mathbf{0} &= \nabla_{\boldsymbol{\mu}_k}\left\{\sum_i \gamma_{ik}^{(t)} \ln \mathcal{N}(\boldsymbol{x}_i;\boldsymbol{\mu}_k,\boldsymbol{Q}_k)\right\} \\
&= \sum_i \gamma_{ik}^{(t)} \nabla_{\boldsymbol{\mu}_k}\left\{-\frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^{\mathsf{T}}\boldsymbol{Q}_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)+\text{const}\right\} \\
&= \sum_i \gamma_{ik}^{(t)} \boldsymbol{Q}_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k) = \boldsymbol{Q}_k^{-1}\left(\sum_i \gamma_{ik}^{(t)}\boldsymbol{x}_i - \boldsymbol{\mu}_k\sum_i \gamma_{ik}^{(t)}\right) \\
\Rightarrow \boldsymbol{\mu}_k &= \frac{\sum_i \gamma_{ki}^{(t)}\boldsymbol{x}_i}{\sum_i \gamma_{ki}^{(t)}}, \quad \text{which we use for } \boldsymbol{\mu}_k^{(t+1)}
\end{aligned}$$

- Finally, $\boldsymbol{Q}_k$ can be optimized in a similar manner, giving $\boldsymbol{Q}_k^{(t+1)}$

EM for GMM fitting: Summary

- Initialize $\gamma_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{Q}_k^{(0)}$ and repeat for $t = 0, 1, 2 \ldots$

$$\forall k, i: \quad \gamma_{ki}^{(t)} = \frac{\gamma_k^{(t)} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{Q}_k^{(t)})}{\sum_{k'} \gamma_{k'}^{(t)} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{k'}^{(t)}, \boldsymbol{Q}_{k'}^{(t)})}$$

$$\forall k: \quad \gamma_k^{(t+1)} = \tfrac{1}{n} \textstyle\sum_{i=1}^n \gamma_{ki}^{(t)}$$

$$\forall k: \quad \boldsymbol{\mu}_k^{(t+1)} = \tfrac{1}{n} \textstyle\sum_{i=1}^n \frac{\gamma_{ki}^{(t)}}{\gamma_k^{(t+1)}} \boldsymbol{x}_i$$

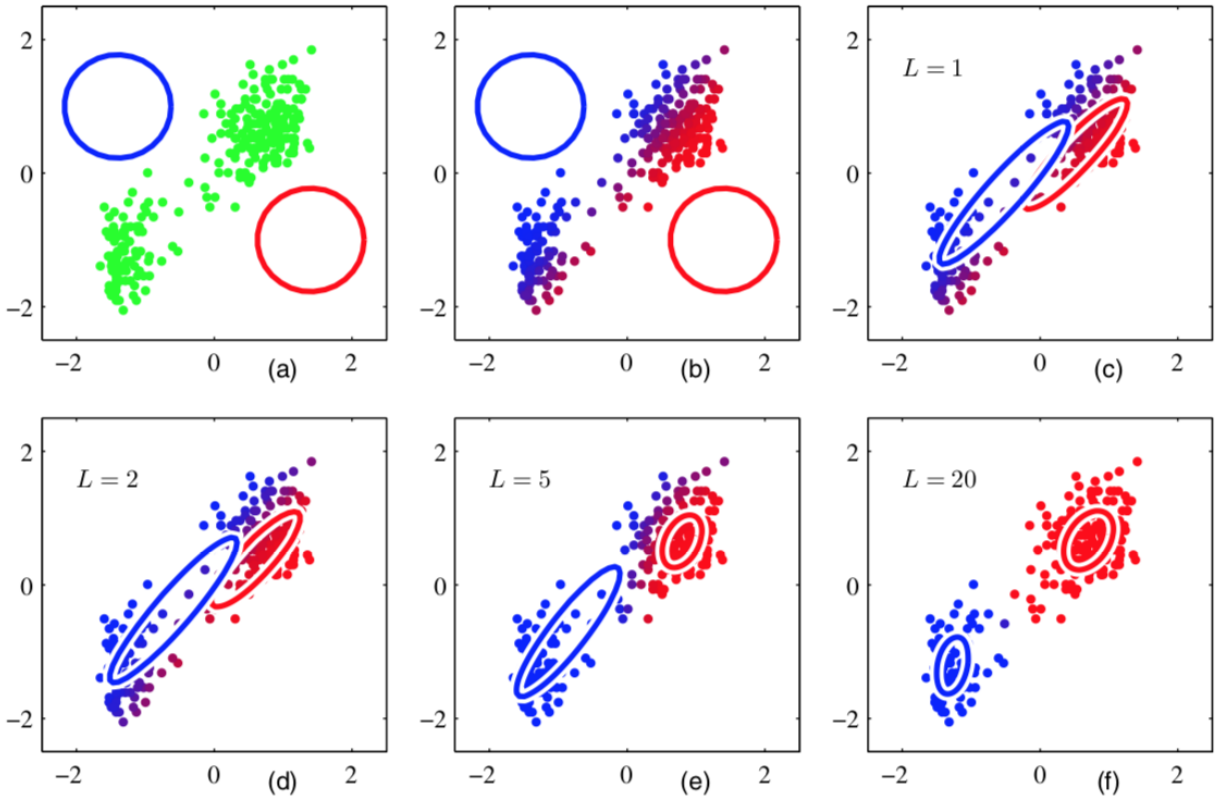
$$\forall k: \quad \boldsymbol{Q}_k^{(t+1)} = \tfrac{1}{n} \textstyle\sum_{i=1}^n \frac{\gamma_{ki}^{(t)}}{\gamma_k^{(t+1)}} (\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(t+1)})^{\mathsf{T}}$$

- For the initialization, it is common to use k-means:
  - set $\gamma_k^{(0)}$ to the fraction of samples that k-means assigns to k
  - set $\boldsymbol{\mu}_k^{(0)}, \boldsymbol{Q}_k^{(0)}$ to mean & covariance of samples that k-means assigned to k
- With large dimension d, one often restricts $\boldsymbol{Q}_k$ to be diagonal

EM for GMM fitting: Example

Example of EM algorithm fitting a Gaussian mixture with $K = 2$ components

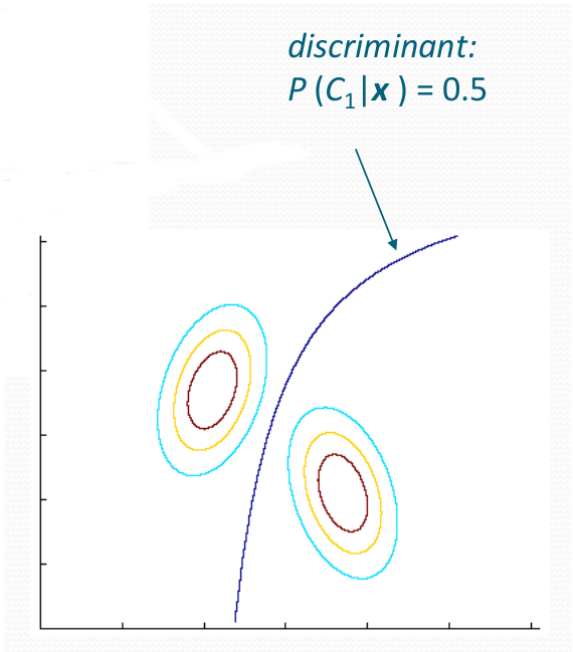
- L denotes iteration
- Soft assignments are visible as purple colors (hard would be pure red or blue)



Bishop, *Pattern Recognition and Machine Learning*, 2006

EM-GMM versus k-means

- EM computes soft assignments $\Pr\{\kappa_i = k\}$, while k-means computes hard assignments $\widehat{k}(\boldsymbol{x}_i)$
- EM estimates covariances $\boldsymbol{Q}_k$, while k-means implicitly uses $\boldsymbol{Q}_k = \epsilon \boldsymbol{I}\ \forall k$
  - EM uses piecewise curved decision boundaries
  - k-means uses piecewise linear boundaries
- EM estimates weights γ_k, while k-means implicitly uses $\gamma_k = \frac{1}{K}\ \forall k$
- EM is more computationally expensive than k-means



- Both can get stuck in local minima; both benefit from a good initialization
- Both implemented in sklearn, e.g.,

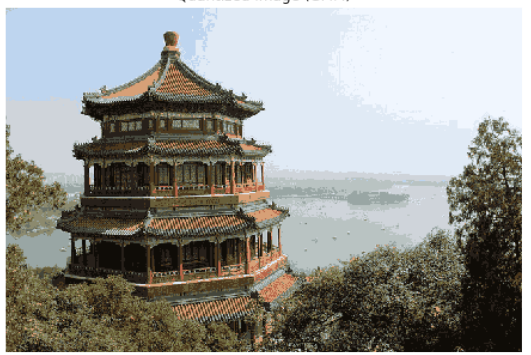
```
GMM = GaussianMixture(n_components=n_colors, covariance_type = 'diag',
                      max_iter=100, random_state=0, init_params = 'kmeans')
GMM.fit(image_array_sample)
GMM_labels = GMM.predict(image_array)
```

Demo: Color quantization via clustering

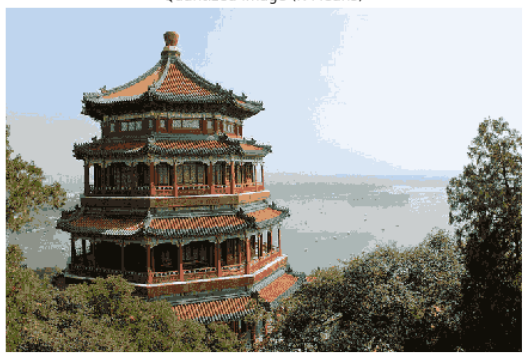
- Standard "true color" image
  - $256 = 2^8$ possibilities each for R, G, B
  - 24 bits per pixel
  - $2^{24} = 16\,777\,216$ unique possibilities in color palette
- Goal: Reduce color palette down to 16 possibilities
  - 4 bits per pixel
  - Question: What are the $2^4 = 16$ best colors?
- Idea: Treat as a clustering problem:
  - $d = 3$ features, $K = 16$ centroids
  - learn centroids from $n = 1000$ random pixels
  - then quantize all $273\,280$ pixels to centroids
- We compare EM-GMM and k-means on the right
  - EM-GMM better preserves fine details

Original image (96,615 colors)

Quantized image (GMM)



Quantized image (K-Means)



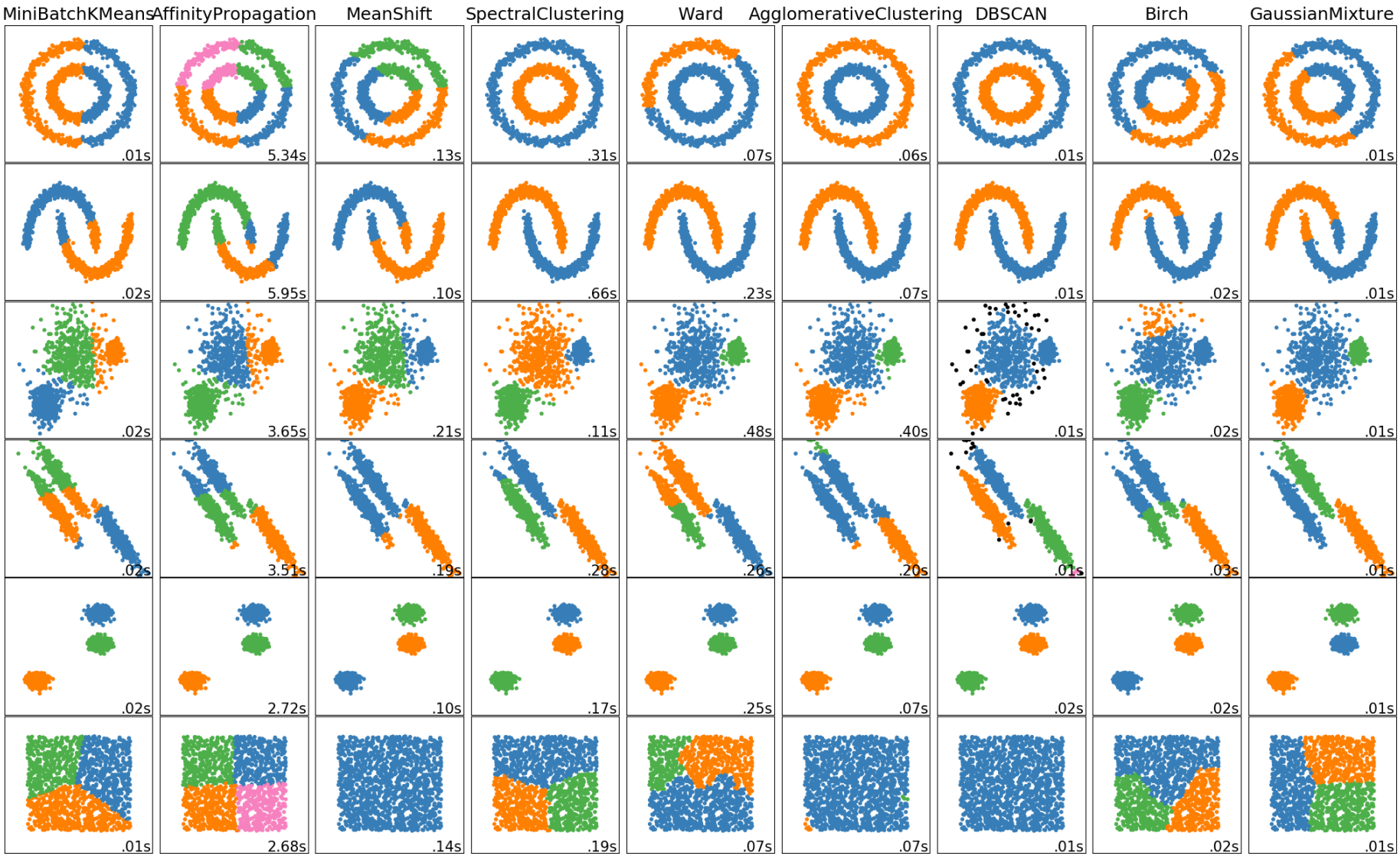
Outline

- Motivating Example: Document Clustering
- Clustering and K-Means
- Text Mining with Bag-of-Words, TF-IDF, and K-Means
- Clustering via Low-Rank Models: LSA and NMF
- Gaussian Mixture Models (GMMs)
- Expectation-Maximization (EM) Fitting of GMMs
- Other Clustering Methods

Model-order selection

- Both k-means and EM-GMM require the number of components K as input
- Various other clustering methods learn K automatically, e.g.,
  - Dirichlet process mixture model
  - mean-shift
  - affinity propagation
  - many others . . . see examples on next page
- Another important question is: What features should we use for clustering?
  - Can imagine using non-linear feature transformations, e.g., a kernel
  - Leads to the spectral clustering method . . . see example on next page

Other clustering methods in `sklearn.cluster`



Learning objectives

- Understand the k-means clustering objective and Lloyd's algorithm
- Understand term-document matrices and TF-IDF scores for text-mining
- Understand NMF, its relation to PCA, and application to clustering
- Understand GMMs and their application to clustering
- Understand the EM algorithm and its application to GMM parameter fitting