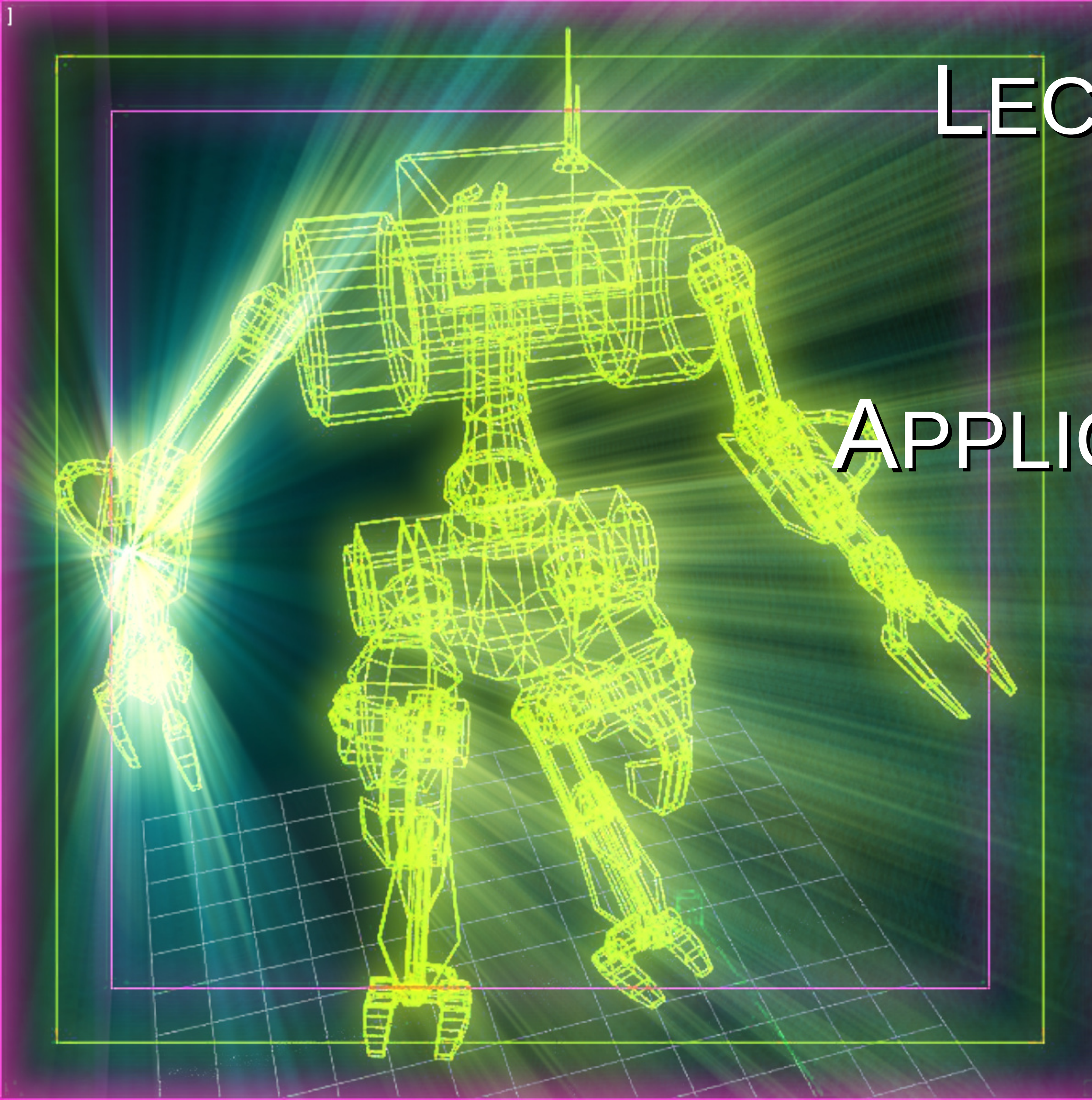


Lecture 4 part 1: First Applications

Document representation

- **Representation 1: binary vector.** For each term v in a vocabulary of V terms, check whether the word occurs in document i

$$\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,V}]$$
$$= [1('aardvark' \in document\ i), \ldots, 1('zebra' \in document\ i)]$$

- **Representation 2: count vector.** For each term v in a vocabulary of V terms, count how many times the word occurs in document i

$$\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,V}]$$
$$= [count('aardvark' \in document\ i), \ldots, count('zebra' \in document\ i)]$$

Weighting and normalization

- In document vectors, features with the biggest values tend to affect distances and cosine similarities the most.
- Longer documents will have vectors with greater counts.
- Considering only the counts of a term (word) in a document itself does not take into account the distribution of the term in a collection
  - If the word is common in all documents, it is not surprising that it occurs a lot in the current document
- To improve the performance of document representation in different tasks, it is typical to adjust document vectors by
  - **weighting** of the features: multiplying feature values by feature importances.
  - **normalization of the features**: trying to make all features have a similar range/amount of variation over a collection
  - **normalization** of the document vectors

Weighting and normalization

- **Feature normalization 1: per-feature z-score.** The z-score is compares a feature value to its distribution in a collection: it is the number of standard deviations the value is higher/lower than the mean value.

$$z_{iv} = \frac{x_{iv} - \mu_v}{\sigma_v}$$

where $\mu_v = (1/T)\sum_{i=1}^{T} x_{iv}$ is the mean of feature v

and $\sigma_v = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(x_{iv} - \mu_v)^2}$ is the standard deviation of feature v in the collection

Weighting and normalization

- **Document normalization 1: unit vector norm.** This normalization makes the document vector have length (norm) 1.

$$\mathbf{x}_i' = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|} = \left[\frac{x_{i1}}{\sqrt{\sum_{v=1}^{V} x_{iv}^2}}, \ldots, \frac{x_{iV}}{\sqrt{\sum_{v=1}^{V} x_{iv}^2}} \right]$$

- **Document normalization 2: proportional counts.** If the feature values are counts of terms, this normalization changes them to proportions of all terms in the document.

$$\mathbf{x}_i' = \frac{\mathbf{x}_i}{\sum_{v=1}^{V} x_{iv}} = \left[\frac{x_{i1}}{\sum_{v=1}^{V} x_{iv}}, \ldots, \frac{x_{iV}}{\sum_{v=1}^{V} x_{iv}} \right]$$

TF-IDF

- Considering only the counts of a term (word) in a document itself does not take into account the distribution of the term in a collection
  - If the word is common in all documents, it is not surprising that it occurs a lot in the current document
- **Term frequency - inverse document frequency** (**TF-IDF**) is a weighting method popular in natural language processing and information retrieval
  - TF-IDF is not just term weighting, but a more general transformation of features
  - Idea: upweight terms that appear a lot in the current document, but downweight terms that appear in a large part of the document collection

TF-IDF

- TF-IDF formula:

$$\boldsymbol{x}_i' = \left[tf(1,i) \cdot idf(1), \ldots, tf(V,i) \cdot idf(V) \right]$$

for each feature (term) v, compute the product of

  - a term frequency value $tf(v,i)$ of the term in document i
  - an inverse document frequency value $idf(v)$ of the term in the collection

- There are multiple variant equations for the tf and idf values
- Term frequency variants:

1. Boolean: $tf(v,i) = \delta(v \in document\ i)$ one if v is in i, zero otherwise
2. Raw count: $tf(v,i) = count(v \in document\ i)$
3. Length-normalized frequency: $tf(v,i) = \dfrac{count(v \in document\ i)}{number\ of\ terms \in i}$
4. Logarithm of the count: $tf(v,i) = \log(1 + count(v \in document\ i))$
5. Count relative to most frequent term:

$$tf(v,i) = \alpha + (1-\alpha) \frac{count(v \in document\ i)}{max_{u=1}^{V} count(u \in document\ i)}$$

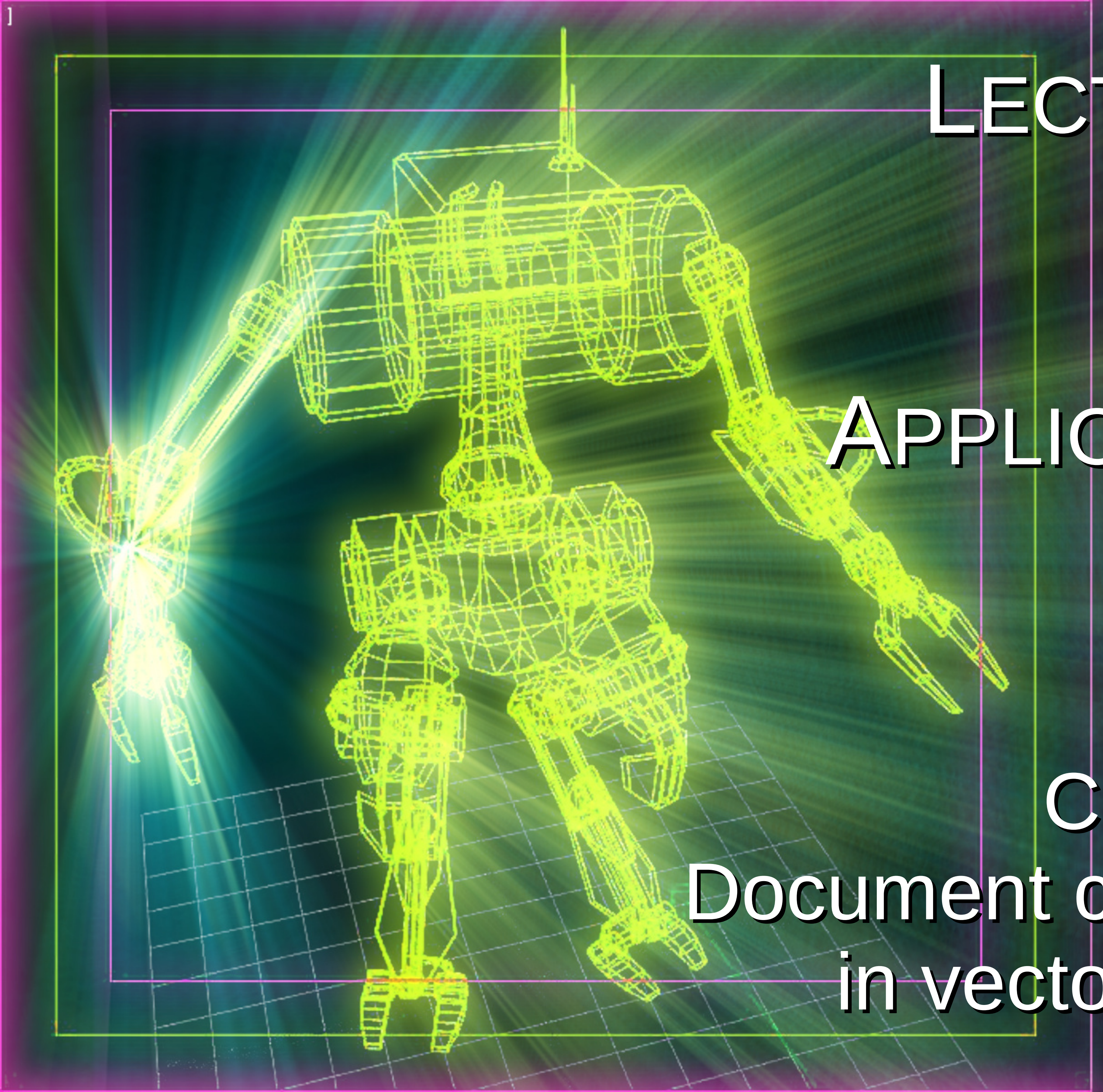
TF-IDF

- Inverse document frequency variants:
  1. Unary/boolean: $idf(v)=1$ for all v in the collection
  2. Logarithmic inverse document frequency: $$idf(v)=\log\left(\frac{T}{\sum_{i=1}^{T}\delta(v\in document\ i)}\right)$$
  3. Smoothed version to avoid zero and infinite values: $$idf(v)=1+\log\left(\frac{T}{1+\sum_{i=1}^{T}\delta(v\in document\ i)}\right)$$
  4. Version proportional to most common term: $$idf(v)=\log\left(\frac{max_{u=1}^{V}\sum_{i=1}^{T}\delta(u\in document\ i)}{1+\sum_{i=1}^{T}\delta(v\in document\ i)}\right)$$
  5. Version proportional to documents without the term $$idf(v)=\log\left(\frac{T-\sum_{i=1}^{T}\delta(v\in document\ i)}{\sum_{i=1}^{T}\delta(v\in document\ i)}\right)$$

TF-IDF

- In Python:

```
#%% Create TF-IDF vectors
n_docs=len(mycrawled_prunedtexts)
n_vocab=len(remainingvocabulary)
# Matrix of term frequencies
tfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Row vector of document frequencies
dfvector=scipy.sparse.lil_matrix((1,n_vocab))
# Loop over documents
for k in range(n_docs):
    # Row vector of which words occurred in this document
    temp_dfvector=scipy.sparse.lil_matrix((1,n_vocab))
    # Loop over words
    for l in range(len(mycrawled_prunedtexts[k])):
        # Add current word to term-frequency count and document-count
        currentword=myindices_in_prunedvocabulary[k][l]
        tfmatrix[k,currentword]=tfmatrix[k,currentword]+1
        temp_dfvector[0,currentword]=1
    # Add which words occurred in this document to overall document counts
    dfvector=dfvector+temp_dfvector
# Use the count statistics to compute the tf-idf matrix
tfidfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Let's use raw term count, and smoothed logarithmic idf
idfvector=numpy.squeeze(numpy.array(dfvector.todense()))
idfvector=1+numpy.log(((idfvector+1)**-1)*n_docs)
for k in range(n_docs):
    # Find nonzero term frequencies
    tempindices=numpy.nonzero(tfmatrix[k,:])[1]
    tfterm=numpy.squeeze(numpy.array(tfmatrix[k,tempindices].todense()))
    # Combine the tf and idf terms
    tfidfmatrix[k,tempindices]=tfterm*idfvector[tempindices]
```



Lecture 4 part 1: First Applications

Chapter 9: Document clustering in vector spaces

Document clustering

- The TF-IDF representation can be used for various modeling and machine learning tasks.
- Document clustering aims to find subgroups of documents that are semantically similar in content
- A simple statistical solution is to learn a **mixture of Gaussians model** for the set of TF-IDF vectors: each TF-IDF vector **x** is assumed to be generated by one of several high-dimensional multivariate normal distributions
- Gaussians are not a good model for term counts (because counts cannot be less than zero) but can be suitable for some TF-IDF representations
  - We will see mixture models based on other language models later
  - There exist tests for Gaussianity: one could test, after clustering, whether data of each cluster is Gaussian

Document clustering

- Probability density in a mixture of K Gaussians:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)\, p(\mathbf{x}|k)$$

$$= \sum_{k=1}^{K} \beta_k \cdot \frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}_k|}} \cdot \exp\left(-(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}-\boldsymbol{\mu}_k)\right)$$

- The parameters are the prior mixture component probabilities $\beta_k = p(k)$, component mean vectors $\boldsymbol{\mu}_k$ and component covariance matrices $\boldsymbol{\Sigma}_k$
- Covariance matrices are sometimes restricted to be diagonal or even to spherical Gaussians (same value in all diagonal entries)
- A maximum-likelihood solution (parameters maximizing probability of the observed vectors **x**) can be found by the **expectation-maximization (EM) algorithm**

Document clustering

- The EM algorithm repeats the following updates.
- **E (expectation) step:** using current values of mixture component parameters, compute component membership probabilities: probability each data point was generated by a particular mixture component

$$\kappa_{ik} = p(k|\boldsymbol{x}_i) = \frac{p(k)\,p(\boldsymbol{x}_i|k)}{\sum_{k'=1}^{K} p(k')\,p(\boldsymbol{x}_i|k')}$$

$$= \frac{\frac{\beta_k}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^T \Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)/2\right)}{\sum_{k'=1}^{K} \frac{\beta_{k'}}{(2\pi)^{d/2}|\Sigma_{k'}|^{1/2}} \exp\left(-(\boldsymbol{x}_i-\boldsymbol{\mu}_{k'})^T \Sigma_{k'}^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_{k'})/2\right)}$$

Document clustering

- The EM algorithm repeats the following updates.
- **M (maximization) step:** using current values of component membership probabilities, compute updated component parameters that maximize the expected data log-likelihood

$$\beta_k = p(k) \triangleq \frac{1}{N}\sum_{i=1}^{N}\kappa_{ik} \qquad \mu_k = E[\mathbf{x}_i|k] \triangleq \frac{\sum_{i=1}^{N}\kappa_{ik}\mathbf{x}_i}{\sum_{i=1}^{N}\kappa_{ik}}$$

$$\Sigma_k = cov[\mathbf{x}_i|k] \triangleq \frac{\sum_{i=1}^{N}\kappa_{ik}(\mathbf{x}_i-\mu_k)(\mathbf{x}_i-\mu_k)^T}{\sum_{i=1}^{N}\kappa_{ik}}$$

- Repeat until the change in the parameters is small enough

Document clustering

- Let's try this for 20 Newsgroups documents!
- Sometimes to get good results, data has to be **cleaned in a domain-specific way**. For 20 Newsgroups this means excluding **header lines of emails**:

```
# Exclude header lines from each message
excludedlinemarkers=['Xref:','Path:','From:','Newsgroups:','Subject:','Summary:', \
    'Keywords:','Message-ID:','Date:','Expires:','Followup-To:','Distribution:', \
    'Organization:','Approved:','Supersedes:','Lines:','NNTP-Posting-Host:', \
    'References:','Sender:','In-Reply-To:','Article-I.D.:','Reply-To:', \
    'Nntp-Posting-Host:']
for k in range(len(mycrawled_texts)):
    print(k)
    templines=mycrawled_texts[k].splitlines()
    remaininglines=[]
    for l in range(len(templines)):
        line_should_be_excluded=0
        for m in range(len(excludedlinemarkers)):
            if len(templines[l])>=len(excludedlinemarkers[m]):
                if excludedlinemarkers[m]==\
                    templines[l][0:len(excludedlinemarkers[m])]:
                    line_should_be_excluded=1
                    break
        if line_should_be_excluded==0:
            remaininglines.append(templines[l])
    mycrawled_texts[k]='\n'.join(remaininglines)
```

- We will run this after crawling texts
- Then repeat the rest of the processing (lemmatization, vocabulary-finding, vocabulary pruning) as in Lecture 2 - in the pruning we will prune all words with less than 4 occurrences.
- Then perform TF-IDF counting as at the end of lecture 3

Document clustering

- Total vocabulary size after cleanup and pruning: 27872, top most frequent words after the cleanup:

['repair' 'pair' 'usage' 'mouth' 'tube' 'campaign' 'threaten' 'balance' 'tree' 'matt' 'turbo' 'math' 'violent' 'rape' 'previously' 'unlike' 'perspective' 'trick' 'edward' 'irrelevant' 'supposedly' 'survivor' 'angel' 'wayne' 'stock' 'sad' 'plot' 'neutral' 'salt' 'gather' 'accuracy''grace' 'ontario' 'eliminate' 'flow' 'ignorance' 'voltage' 'quadra' 'france' 'char' 'minnesota' 'tor' 'lc' 'ab' 'exit' 'replacement' 'presence' 'publication' 'module' 'carefully' 'confirm' 'phenomenon' 'unable' 'pd' 'luke' 'intent' 'toolkit' 'un' 'writing' 'hill''philosophy' 'subjective' 'nick' 'declare' 'motorola' 'buffalo' 'palestine' 'impression' 'index' 'winner' 'surrender' 'ax' 'istanbul' 'reverse' 'crowd' 'earlier' 'converter' 'minister' 'que' 'daughter' 'automatically' 'execute' 'july' 'expansion' 'importance' 'london' 'serdar' 'jerusalem' 'unlikely' 'loop' 'affair' 'lady' 'senator' 'sweden' 'lebanon' 'sector' 'category' 'hypothesis' 'primarily' 'struggle' 'plate' 'related' 'wild' 'expand' 'desktop' 'percentage' 'icon' 'restriction' 'restrict' 'hopefully' 'fellow' 'core' 'influence' 'pope' 'surround' 'regulation' 'slaughter' 'puck' 'roy' 'hawk' 'ai' 'friday' 'southern' 'psalm' 'complaint' 'sgi' 'successful' 'buck' 'helpful' 'accuse' 'negative' 'glad' 'tx' 'eastern' 'offense' 'plenty' 'plastic' 'acquire' 'kuwait' 'atom' 'bond' 'defensive' 'polygon' 'prohibit' 'immoral' 'messiah' 'inform' 'transmit' 'dispute' 'shout' 'randy' 'trace' 'announcement' 'economy' 'priest' 'progress' 'physician' 'england' 'election' 'silly' 'atmosphere' 'film' 'steven' 'empire' 'improvement''newsletter' 'award' 'argic' 'ms' 'effectively' 'derive' 'manufacture' 'ot' 'diamond' 'camp' 'partner' 'portable' 'broadcast' 'contest' 'union' 'feed' 'museum' 'february' 'equally' 'theist' 'chief' 'compress' 'orthodox' 'pound' 'promote' 'sony' 'correction' 'pad' 'venus' 'cipher' 'univ' 'fourth' 'sox' 'japan' 'karl' 'online' 'pack' 'divine' 'amateur' 'lucky' 'storage' 'french' 'politics' 'clh' 'extreme' 'marry' 'header' 'december' 'dean' 'florida' 'practical' 'era' 'metal' 'whenever' 'jake' 'pointer' 'combination' 'global' 'cat' 'gon' 'enable' 'favorite' 'blind' 'august' 'flat' 'min' 'ethnic' 'inning' 'firm' 'chi' 'capture' 'offensive' 'bitnet' 'numerous' 'lewis' 'mainly' 'chemistry' 'virginia' 'smart' 'intel' 'maynard' 'ear' 'baltimore' 'rsa' 'significantly' 'widely' 'dry' 'height' 'weekend' 'decade' 'comic' 'strategy' 'rational' 'elsewhere' 'eisa' 'crazy' 'conservative' 'saturn' 'constant' 'ottawa' 'sp' 'prime' 'lebanese' 'consist' 'justification' 'eight' 'unto' 'gift' 'observer' 'believer' 'lift' 'opposite' 'tiff' 'candida' 'quantum' 'bosnian' 'commandment' 'permission' 'patent' 'grab' 'winnipeg' 'brief' 'parameter' 'critical' 'spencer' 'shark' 'subscribe' 'socket' 'animation' 'depth' 'dear' 'clinical' 'propaganda' 'canon' 'aim' 'resurrection' 'dare' 'saint' 'destruction' 'skill' 'mm' 'senior' 'dennis' 'sheet' 'ar' 'introduction' 'gene' 'reno' 'skin' 'cloud' 'premise' 'sternlight''spiritual' 'quit' 'sake' 'combine' 'mix' 'encounter' 'houston' 'selection' 'vs' 'carl' 'bruin' 'assembly' 'dozen' 'fat' 'empty' 'guard' 'bitmap' 'thanx' 'ml' 'nyi' 'gaza' 'similarly' 'cub' 'taste' 'universal' 'hotel' 'republican' 'superior' 'cylinder' 'rush' 'russell' 'mt' 'substance' 'wonderful' 'ridiculous' 'array' 'useless' 'ideal' 'inc' 'murray' 'terrorism' 'essential' 'divide' 'goalie' 'grenade' 'calgary' 'rumor' 'teacher' 'wise' 'signature' 'democracy' 'hebrew' 'sabre' 'dynamic' 'linux' 'dale' 'acid' 'penguin' 'minimum' 'presumably' 'gk' 'xview' 'torture' 'saturday' 'husband' 'mv' 'cursor' 'ac' 'fpu' 'francis' 'app' 'zionist' 'extent' 'furthermore' 'sam' 'quebec' 'ignorant' 'investigate' 'accelerator' 'huh' 'tb' 'centris' 'miracle' 'knife' 'sacrifice' 'supreme' 'idle' 'aaron' 'translate' 'employ' 'moslem' 'ra' 'crap' 'clue' 'appearance' 'apparent' 'irvine' 'forum' 'employee' 'occupation' 'centre' 'svga' 'warranty' 'duo' 'xlib' 'silver' 'annoy' 'wilson' 'hr' 'ought' 'terrible' 'idiot' 'immediate' 'famous' 'identical' 'virtually' 'suddenly' 'log' 'elect' 'anderson' 'rev' 'resolve' 'humanity' 'ta' 'lawrence' 'mp' 'ql' 'outlet' 'captain' 'mi' 'classic' 'printf' 'square' 'traditional' 'flee' 'co' 'eg' 'tonight' 'honor' 'nra' 'jonathan' 'forth' 'punish' 'infant' 'settle' 'demo' 'verify' 'advanced' 'sink' 'stl' 'cultural' 'convex' 'salvation' 'yeast' 'gang' 'disappear' 'islander' 'hiv' 'quiet' 'nonsense' 'jersey' 'developer' 'worker' 'dont' 'mailing' 'doc' 'davidian' 'sunos' 'contradiction' 'highway' 'vol''stereo' 'tactic' 'accomplish' 'contribution' 'pa' 'leg' 'january' 'shaft' 'austin' 'catalog' 'spare']

Document clustering

- **Using the scikit-learn library** works for small data: let's try this for 20 Newsgroups documents, using only the top-500 overall highest-valued TF-IDF features.

```
# Reduce the data to 500 highest-total TF-IDF features
dimensiontotals=numpy.squeeze(numpy.array( \
    numpy.sum(tfidfmatrix,axis=0)))
highesttotals=numpy.argsort(-1*dimensiontotals)
Xsmall=tfidfmatrix[:,highesttotals[0:500]]
Xsmall=Xsmall.todense()
# Normalize the documents to unit vector norm
tempnorms=numpy.squeeze(numpy.array(numpy.sum(numpy.multiply(Xsmall,Xsm
all),axis=1)))
# If any documents have zero norm, avoid dividing them by zero
tempnorms[tempnorms==0]=1
Xsmall=scipy.sparse.diags(tempnorms**-0.5).dot(Xsmall)

import sklearn
import sklearn.mixture
# Create the mixture model object, and
# choose the number of components and EM iterations
mixturemodel=sklearn.mixture.GaussianMixture(n_components=20, \
    covariance_type='diag',max_iter=100,init_params='random')

fittedmixture=mixturemodel.fit(Xsmall)
sklearn_mixturemodel_means=fittedmixture.means_
sklearn_mixturemodel_weights=fittedmixture.weights_
sklearn_mixturemodel_covariances=fittedmixture.covariances_
```

Document clustering

- **Top words per cluster:**

```
# Find top 20 words with highest mean feature value for each cluster
for k in range(n_components):
    print(k)
    highest_dimensionweight_indices=numpy.argsort( \
        -numpy.squeeze(sklearn_mixturemodel_means[k,:]),axis=0)
    highest_dimensionweight_indices=highesttotals[highest_dimensionweight_indices]
    print(' '.join(remainingvocabulary[highest_dimensionweight_indices[1:20]]))
```

- **Results:**

0: senator dare winnipeg ottawa union shark french premise fallacy winner balance intent ar quebec expansion threaten skill messiah daughter
1: wild wayne dry plastic quantum skin weekend buck grenade sp lc clh silver pound shark ac encounter loop cartridge
2: centris ontario quadra svga diamond turbo halat exit accelerator sony vesa warranty repair socket dale portable film capture japan
3: outlet polygon canon chemistry announcement wiring vlb ot gravity chi conservative shark pope tor practical cipher signature luke catalog
4: lebanon divine moslem acquire blind serdar argic gaza southern numerous occupation theist pope lady neutral passenger tartar effectively empire
5: maynard goalie francis theist acid puck skill observer storage flow sink sabre dispute stock mainly crowd min km mt
6: gld winnipeg bruin winner calgary baltimore penguin dare app quebec islander inning era comic min roy outlet dean florida
7: storage transmit captain unlikely unlike sam penguin initiative acquire believer era pointer colormap pair hr cursor islander priest strategy
8: ridiculous hopefully subjective atom sink stock dean ignorance tiff enable equally partner halat plenty perspective signature theist reverse importance
9: carl desktop sweden wilson wonderful carefully nick cat sector polygon complaint gene fpu physician pd crazy cub minnesota lewis
10: lebanese occupation serdar iraq argic kuwait jake henrik terrorism istanbul moslem un palestine slaughter empire politics torture france zionist
11: clinical presence pad resurrection grenade apostle opposite fallacy progress km clh introduction immoral chemistry employ earlier mouth nra philosophy
12: spencer pointer sternlight steven amateur sad icon helpful survivor motorola supposedly grab broadcast glad dear impression edward replacement univ
13: atmosphere mv tor gene restriction contest crowd baku min que chi stl friday permission goalie lift camp transmit tb
14: array essential sacrifice guard rev wonderful knife grenade sgi affair xlib carl england significantly lift mix reno complaint stereo
15: hebrew accuse translate gift grace gant therapy marry unto salvation divine believer defensive resurrection spiritual offensive thou torture constant
16: duo gang accelerator intel universal karl vesa turbo quadra senior developer significantly transmit height plenty centris portable irvine desktop
17: pair animation randy matt sox catalog hill hawk apps quit irvine subscribe gon inc ms rumor houston trace baltimore
18: saturn knife ra encounter hr ethnic dynamic hill mouth module jerusalem violent accuse passenger ax negative buck ms japan
19: yeast introduction telescope venus interactive hess demo sony tiff compress toolkit converter clinical sgi global online hotel houston july

Document clustering

- Unfortunately, this scikit-learn function does not work with sparse matrices, so cannot be used for very large or high-dimensional data: even the TF-IDF data matrix itself may not fit in memory as a non-sparse matrix
  - Solution: use another library, or code your own (next slides!)
- Other problems with coding implementations of mixture modeling:
  - The E-step requires inversion of covariance matrices, which is very slow for full covariance matrices in high dimensionalities. Solution: restrict e.g. to diagonal matrices
  - In high dimensionalities data may be very far from mixture component means, causing numeric underflow errors in computation. Solution: cancel out minimum-distance term in numerator & denominator of e-step
  - Sometimes even that is enough, more overflow/underflow errors remain. Solution: perform computations using logarithms of the quantities.
  - For-loops over large data sizes, dimensions and components are extremely slow in Python. Solution: perform as many computations as possible using vector-matrix algebra, since those Python functions are internally implemented in a faster way.

Document clustering

- In Python, using the **self-made code: first initialize the data**

```
#%% Use the TF-IDF matrix as data to be clustered
X=tfidfmatrix
# Normalize the documents to unit vector norm
tempnorms=numpy.squeeze(numpy.array(numpy.sum(X.multiply(X),axis=1)))
# If any documents have zero norm, avoid dividing them by zero
tempnorms[tempnorms==0]=1
X=scipy.sparse.diags(tempnorms**-0.5).dot(X)

n_data=numpy.shape(X)[0]
n_dimensions=numpy.shape(X)[1]
```

Document clustering

- In Python, using the **self-made code: initialize the model parameters**

```
#%% Initialize the Gaussian mixture model

# Function to initialize the Gaussian mixture model, create component parameters
def initialize_mixturemodel(X,n_components):
    # Create lists of sparse matrices to hold the parameters
    n_dimensions=numpy.shape(X)[1]
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    mixturemodel_weights=numpy.zeros((n_components))
    mixturemodel_covariances=[]
    mixturemodel_inversecovariances=[]
    for k in range(n_components):
        tempcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_covariances.append(tempcovariance)
        tempinvcovariance=scipy.sparse.lil_matrix((n_dimensions,n_dimensions))
        mixturemodel_inversecovariances.append(tempinvcovariance)

    # Initialize the parameters
    for k in range(n_components):
        mixturemodel_weights[k]=1/n_components
        # Pick a random data point as the initial mean
        tempindex=scipy.stats.randint.rvs(low=0,high=n_components)
        mixturemodel_means[k]=X[tempindex,:].toarray()
        # Initialize the covariance matrix to be spherical
        for l in range(n_dimensions):
            mixturemodel_covariances[k][l,l]=1
            mixturemodel_inversecovariances[k][l,l]=1
    return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
           mixturemodel_inversecovariances)
```

Document clustering

- In Python, using the **self-made code: define a function that performs the E-step**

```
def run_estep(X,mixturemodel_means,mixturemodel_covariances, \
              mixturemodel_inversecovariances,mixturemodel_weights):
    # For each component, compute terms that do not involve data
    meanterms=numpy.zeros((n_components))
    logdeterminants=numpy.zeros((n_components))
    logconstantterms=numpy.zeros((n_components))
    for k in range(n_components):
        # Compute mu_k*inv(Sigma_k)*mu_k
        meanterms[k]=(mixturemodel_means[k,:]* \
            mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]
        # Compute determinant of Sigma_k. For a diagonal matrix
        # this is just the product of the main diagonal
        logdeterminants[k]=numpy.sum(numpy.log(mixturemodel_covariances[k].diagonal(0)))
        # Compute constant term beta_k * 1/(|Sigma_k|^1/2)
        # Omit the (2pi)^d/2 as it cancels out
        logconstantterms[k]=numpy.log(mixturemodel_weights[k]) - 0.5*logdeterminants[k]

    print('E-step part2 ')
    # Compute terms that involve distances of data from components
    xnorms=numpy.zeros((n_data,n_components))
    xtimesmu=numpy.zeros((n_data,n_components))
    for k in range(n_components):
        print(k)
        xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
        xtimesmu[:,k]=numpy.squeeze((X*mixturemodel_inversecovariances[k]* \
            mixturemodel_means[k,:].T).toarray())
    xdists=xnorms+numpy.matlib.repmat(meanterms,n_data,1)-2*xtimesmu
    # Substract maximal term before exponent (cancels out) to maintain computational precision
    numeratorterms=logconstantterms-xdists/2
    numeratorterms-=numpy.matlib.repmat(numpy.max(numeratorterms,axis=1),n_components,1).T
    numeratorterms=numpy.exp(numeratorterms)
    mixturemodel_componentmemberships=numeratorterms/numpy.matlib.repmat( \
        numpy.sum(numeratorterms,axis=1),n_components,1).T
    return(mixturemodel_componentmemberships)
```

Document clustering

- In Python, using the **self-made code: define functions that perform the M-step**

```
def run_mstep_sumweights(mixturemodel_componentmemberships):
    # Compute total weight per component
    mixturemodel_weights=numpy.sum(mixturemodel_componentmemberships,axis=0)
    return(mixturemodel_weights)

def run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights):
    # Update component means
    mixturemodel_means=scipy.sparse.lil_matrix((n_components,n_dimensions))
    for k in range(n_components):
        mixturemodel_means[k,:]=\
            numpy.sum(scipy.sparse.diags(mixturemodel_componentmemberships[:,k]).dot(X),axis=0)
        mixturemodel_means[k,:]/=mixturemodel_weights[k]
    return(mixturemodel_means)

def run_mstep_covariances(X,mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel_means):
    # Update diagonal component covariance matrices
    n_dimensions=numpy.shape(X)[1]
    n_components=numpy.shape(mixturemodel_componentmemberships)[1]
    tempcovariances=numpy.zeros((n_components,n_dimensions))
    mixturemodel_covariances=[]
    mixturemodel_inversecovariances=[]
    for k in range(n_components):
        tempcovariances[k,:]= \
            numpy.sum(scipy.sparse.diags(mixturemodel_componentmemberships[:,k]).dot(X.multiply(X)),axis=0) \
            -mixturemodel_means[k,:].multiply(mixturemodel_means[k,:])*mixturemodel_weights[k]
        tempcovariances[k,:]/=mixturemodel_weights[k]
        # Convert to sparse matrices
        tempepsilon=1e-10
        # Add a small regularization term
        temp_covariance=scipy.sparse.diags(tempcovariances[k,:]+tempepsilon)
        temp_inversecovariance=scipy.sparse.diags((tempcovariances[k,:]+tempepsilon)**-1)
        mixturemodel_covariances.append(temp_covariance)
        mixturemodel_inversecovariances.append(temp_inversecovariance)
    return(mixturemodel_covariances,mixturemodel_inversecovariances)

def run_mstep_normalizeweights(mixturemodel_weights):
    # Update mixture-component prior probabilities
    mixturemodel_weights/=sum(mixturemodel_weights)
    return(mixturemodel_weights)
```

Document clustering

- In Python, using the **self-made code: run the resulting algorithm**

```
#%% Perform the EM algorithm iterations
def perform_emalgorithm(X,n_components,n_emiterations):
    mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
        mixturemodel_inversecovariances=initialize_mixturemodel(X,n_components)

    for t in range(n_emiterations):
        # ====== E-step: Compute the component membership
        # probabilities of each data point ======
        print('E-step ' + str(t))

mixturemodel_componentmemberships=run_estep(X,mixturemodel_means,mixturemodel_covariances,\
            mixturemodel_inversecovariances,mixturemodel_weights)
        # ====== M-step: update component parameters======
        print('M-step ' + str(t))
        print('M-step part1 ' + str(t))
        mixturemodel_weights=run_mstep_sumweights(mixturemodel_componentmemberships)
        print('M-step part2 ' + str(t))

mixturemodel_means=run_mstep_means(X,mixturemodel_componentmemberships,mixturemodel_weights)
        print('M-step part3 ' + str(t))
        mixturemodel_covariances,mixturemodel_inversecovariances=run_mstep_covariances(X,\
            mixturemodel_componentmemberships,mixturemodel_weights,mixturemodel_means)
        print('M-step part4 ' + str(t))
        mixturemodel_weights=run_mstep_normalizeweights(mixturemodel_weights)
    return(mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
        mixturemodel_inversecovariances)
# Try out the functions we just defined on the data
n_components=20
n_emiterations=100
mixturemodel_weights,mixturemodel_means,mixturemodel_covariances,\
        mixturemodel_inversecovariances = perform_emalgorithm(X,n_components,n_emiterations)
```

Document clustering

- **Inspect results: find top words per cluster**

```
# Find top 20 words for each cluster
for k in range(n_components):
    print(k)
    highest_dimensionweight_indices=\
        numpy.argsort(-numpy.squeeze(\
        mixturemodel_means[k,:].toarray()),axis=0)

    print(''.join(remainingvocabulary[\
        highest_dimensionweight_indices[1:20]]))
```

Document clustering

- **Let's try this for 20 Newsgroups documents!**
- **Mixture modeling result** after 20 iterations of EM: top words per cluster (highest TF-IDF feature value in mean vectors):

0: speculation premise sternlight pink mechalas kaldis omniscient deletion mouth fantasy outcome creationism roby pet logically holland wayne credibility password

1: speculation premise sternlight pink mechalas kaldis omniscient deletion mouth fantasy outcome creationism roby pet logically holland wayne credibility password

2:nope pose ultra ns eisa aew spencer pirate cub lopez denver revolver polygon chuq gilmour watt adb idle yankee

3: jaeger gregg annoy psychology bcci explicitly phenomenon namely rushdie sarcasm khomeini atom criticism mohammad oracle vlb eisa odds benedikt

4: rawlins disorder wpr liquid macroevolution bitzm josephus contradiction dna locally attend concrete mcole imaginative creationism tiff thruster complexity mixture

5: believer behaviour arrogant campaign spiritual promote priest marry justification exists laissez bosnian core agnostic mob dogma reform meme gene

6: racism zionism odd racist psychological jake burden zionist andi polygon usage rauser propaganda beyer depression fdisk askew centris reno

7: diamond motto pointer bmug thanx christmas xpert cub leftover truelove originate download stealth cosmo mattingly coin powerbook subscribe trident

8: philip fish dick london tx paperback symbol lynn santa der gainey isbn austin sabre chemistry captain turner bruin atom

9: alomar sony goalie puck baerga rbi converter offense shark xpert finland fuhr sgi lankford lc sweden defensive clone runner

10: borrow gld applicable merit bontchev objectivity rescorla ekr apps jagr subjectively bmp mustang guyd des iici francis halat animation

11: borrow gld applicable merit bontchev objectivity rescorla ekr apps jagr subjectively bmp mustang guyd des iici francis halat animation

12: alomar sony goalie puck baerga rbi converter offense shark xpert finland fuhr sgi lankford lc sweden defensive clone runner

13: objectively killing arbitrary specie saudi halat cruel schneider unto arabia subjective happiness confusion hernlem golden instinctive odwyer conlon evelyn

14: prison irrelevant minimum bobbe beauchaine punish segment sole murderer closely hamid chair pathetic sympathy sink premium matt gang margoli

15: receiver disciple mangoe resurrection jam wingate sword charley horizon clh apostle psalm hebrew mess portable saturn manuscript grace messiah

16: prison irrelevant minimum bobbe beauchaine punish segment sole murderer closely hamid chair pathetic sympathy sink premium matt gang margoli

17: believer behaviour arrogant campaign spiritual promote priest marry justification exists laissez bosnian core agnostic mob dogma reform meme gene

18: funding terrorism apparent river width mill lebanese partner scout lebanon billboard promiscuous parker baalke horizontal dramatically donate monty vertical

19: funding terrorism apparent river width mill lebanese partner scout lebanon billboard promiscuous parker baalke horizontal dramatically donate monty vertica

- Some mixture components above are identical - they have "collapsed into one another". This is typical in mixture modeling. There are strategies to break apert too similar mixture components, or components whose prior probability goes down near zero.

Document clustering

- We can inspect a cluster also by getting the **top documents** for it.
  - Possibility 1: Documents i with highest membership in cluster k: highest $p(k|\boldsymbol{x}_i)$. More probable in this cluster than anywhere else, but could still be outlier documents that do not belong well to any cluster
  - Possibility 2: Documents with highest observation probability in cluster k: highest $p(\boldsymbol{x}_i|k)$. "most typical" examples of the cluster, but might also be fairly typical in other close-by clusters

Document clustering

- Code for the two options:

```
# Version 1 - Get component membership probabilities for each document d
# Find the document d with highest-probability p(k|d) to be from cluster k
for k in range(n_components):
    tempprobs=numpy.array(numpy.squeeze(mixturemodel_componentmemberships[:,k]))
    highest_componentprob_indices=numpy.argsort(-tempprobs,axis=0)
    print(k)
    print(highest_componentprob_indices[0:10])
    print(' '.join(mycrawled_nltktexts[highest_componentprob_indices[0]]))

# Version 2 - Get documents closest to component mean, i.e. highest p(d|k).
# ---The computation of distances here is the same as done in the E-step of EM---
# For each component, compute terms that do not involve data
meanterms=numpy.zeros((n_components))
logdeterminants=numpy.zeros((n_components))
logconstantterms=numpy.zeros((n_components))
for k in range(n_components):
    # Compute mu_k*inv(Sigma_k)*mu_k
    meanterms[k]=(mixturemodel_means[k,:]* \
        mixturemodel_inversecovariances[k]*mixturemodel_means[k,:].T)[0,0]
# Compute terms that involve distances of data from components
xnorms=numpy.zeros((n_data,n_components))
xtimesmu=numpy.zeros((n_data,n_components))
for k in range(n_components):
    xnorms[:,k]=(X*mixturemodel_inversecovariances[k]*X.T).diagonal(0)
    xtimesmu[:,k]=numpy.squeeze((X*mixturemodel_inversecovariances[k]* \
        mixturemodel_means[k,:].T).toarray())
xdists=xnorms+numpy.matlib.repmat(meanterms,n_data,1)-2*xtimesmu

for k in range(n_components):
    tempdists=numpy.array(numpy.squeeze(xdists[:,k]))
    highest_componentprob_indices=numpy.argsort(tempdists,axis=0)
    print(k)
    print(highest_componentprob_indices[0:10])
    print(' '.join(mycrawled_nltktexts[highest_componentprob_indices[0]]))
```

Document clustering

- **Example for Gaussian mixture component 0 in 20 Newsgroups:**

  0: speculation premise sternlight pink mechalas kaldis omniscient deletion mouth fantasy outcome creationism roby pet logically holland wayne credibility password

- Document with highest p(k|**x**):
  "... Morals are , in essence , personal opinions . Usually # > # (ideally) well-founded , motivated such , but nonetheless personal . The # > # fact that a real large lot of people agree on some moral question , # > # sometimes even for the same reason , does not make morals objective ; it # > # makes humans somewhat alike in their opinions on that moral question , # > # which can be good for the evolution of a social species . # > # > And if a `` real large lot '' (nice phrase) of people agree that there is a # > football on a desk , I 'm supposed to see a logical difference between the two ? # > Perhaps you can explain the difference to me , since you seem to see it # > so clearly . # > # (rest deleted) # # That 's a fallacy , and it is not the first time it is pointed out . It 's not a fallacy - note the IF . IF a supermajority of disinterested people agree on a fundamantal value (we 're not doing ethics YET ..."
- Documents like these might be outliers of the cluster
- Document with highest p(**x**|k):

  "This is just a test to see if this works ."
- Documents like these might be near the center of the cluster since it is composed of "generic" words that may be common enough in every cluster.

Document clustering

- Some other tasks that can be done based on a vector-space model:
  - Visualization of a document collection
  - Outlier detection (e.g. spam, trolling, malformed messages)
  - Information retrieval
- More on all of those on later lectures