Carnegie Mellon University
Heinz College

# 94-775 Unstructured Data Analytics

## Lecture 9: Clustering (cont'd); topic modeling

Slides by George H. Chen

# Last Time: Clustering on Text Demo

- We were clustering on the 20 Newsgroups dataset (preprocessed by lemmatizing every token)
- We filtered out some documents that are likely not in English
- We filtered out vocab words that showed up in too many or too few documents
  - Resulting 2D table of feature vectors: `filtered_tf`
  - Resulting 1D table of vocabulary words: `filtered_vocab`
- After filtering, text documents still varied wildly in length
  - Convert each row of `filtered_tf` into a probability vector
    - Resulting 2D table of probability vectors: `prob_vectors`

# Clustering on Text Demo

Raw text doc. → 11707-dim prob vector → 100-dim PCA space → 2-dim t-SNE space

Distance between points in prob vector space is not helpful

Distance between points in PCA space is more helpful

We show clustering results in 100-dim PCA space

We also show clustering results in 2-dim t-SNE space

# Today

- Intuition on ellipse/ellipsoid shapes for GMMs and why they matter
- Finally answer the question of when we should expect k-means to work well (2 example scenarios)
- TF-IDF representation
- Topic models

Note: I've decided to delay when we cover how to automatically choose the number of clusters/topics

(I've reordered the topics covered also to try to more quickly finish covering what you need to do your HW2)

Reminder: Your 40-minute Quiz 2 this Friday covers weeks 3 & 4 + today's lecture

# (Flashback) Learning a GMM

Step 0: Guess $k$

Step 1: Guess cluster probabilities, means, and covariances

(often done using $k$-means)

**Repeat until convergence:**

Step 2: Compute probability of each point being in each of the $k$ clusters

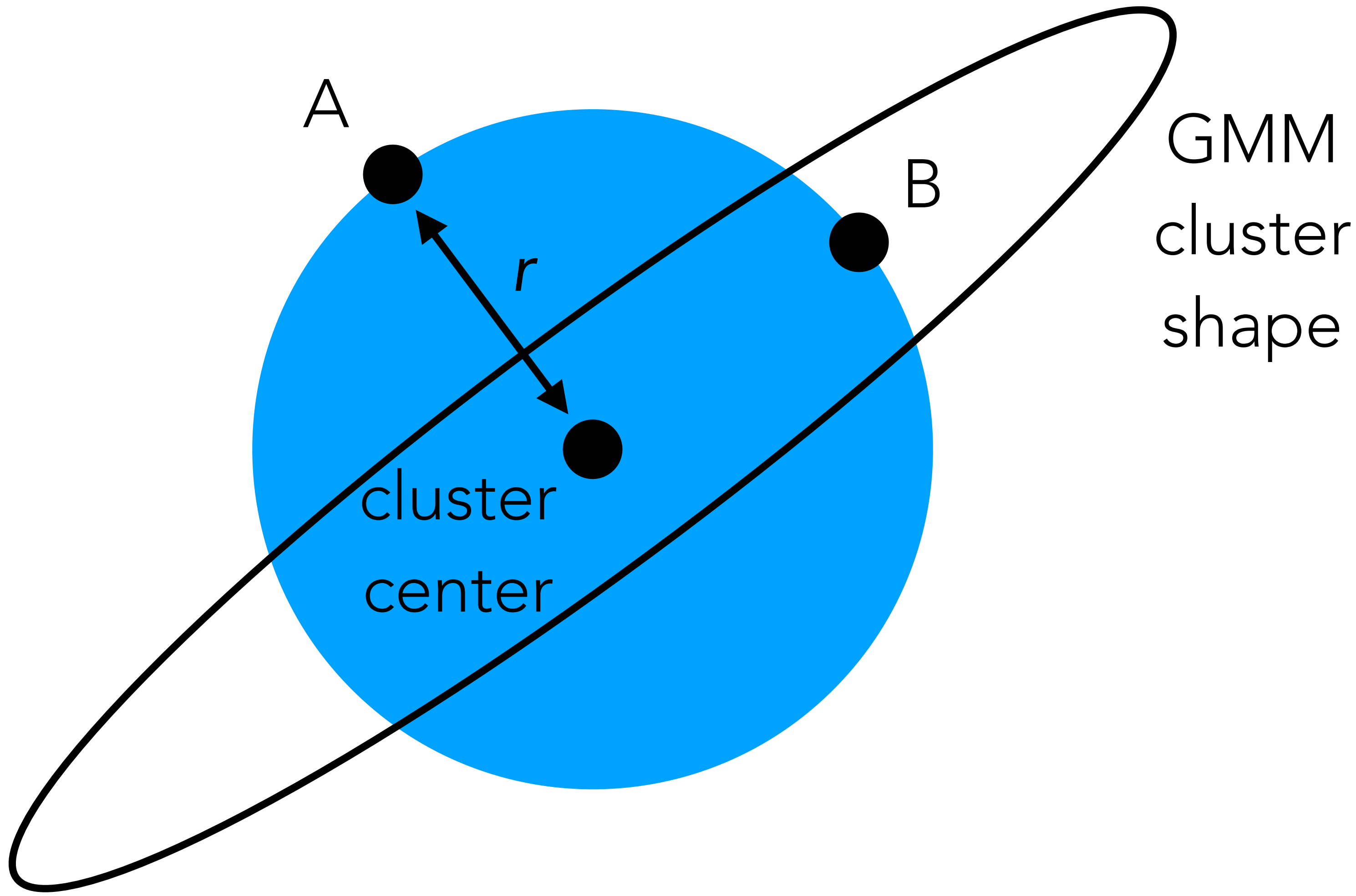
Step 3: Update cluster probabilities, means, and covariances accounting for probabilities of each point belonging to each of the clusters

This algorithm is called the **Expectation-Maximization** (EM) algorithm for GMMs (and approximately does maximum likelihood)

(Note: EM by itself is a general algorithm not just for GMMs)

# (Rough Intuition) How Shape is Encoded by a GMM

For this ellipse-shaped Gaussian, point B is considered more similar to the cluster center than point A




*k*-means would think that point A and point B are equally similar to the cluster center (since both points are distance *r* away from the center)

# Relating *k*-means to GMMs

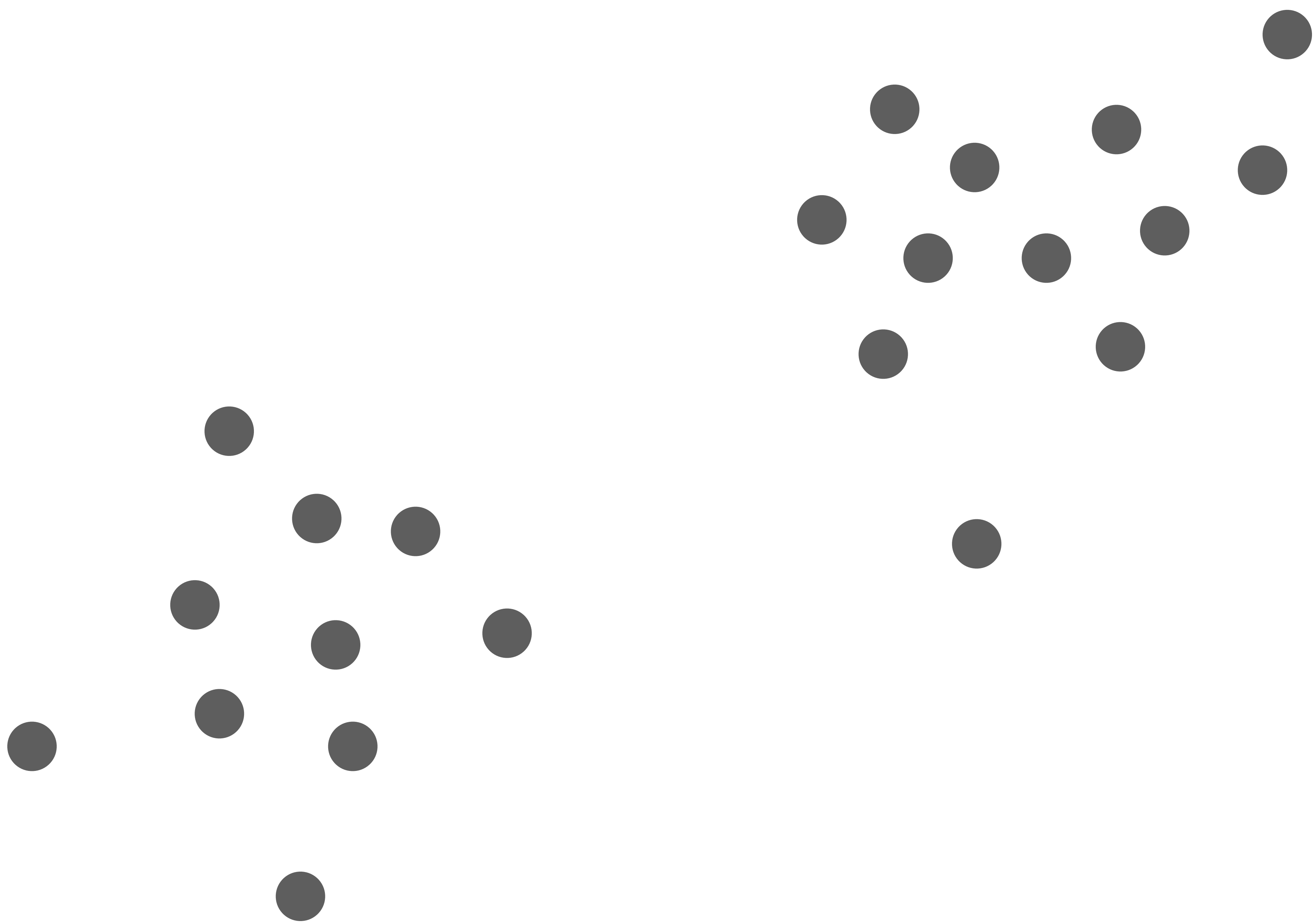
If the ellipses are all circles and have the same "skinniness"
(e.g., in the 1D case it means they all have same variance):

- *k*-means approximates the EM algorithm for GMMs
(as there is no need to keep track of cluster shape)

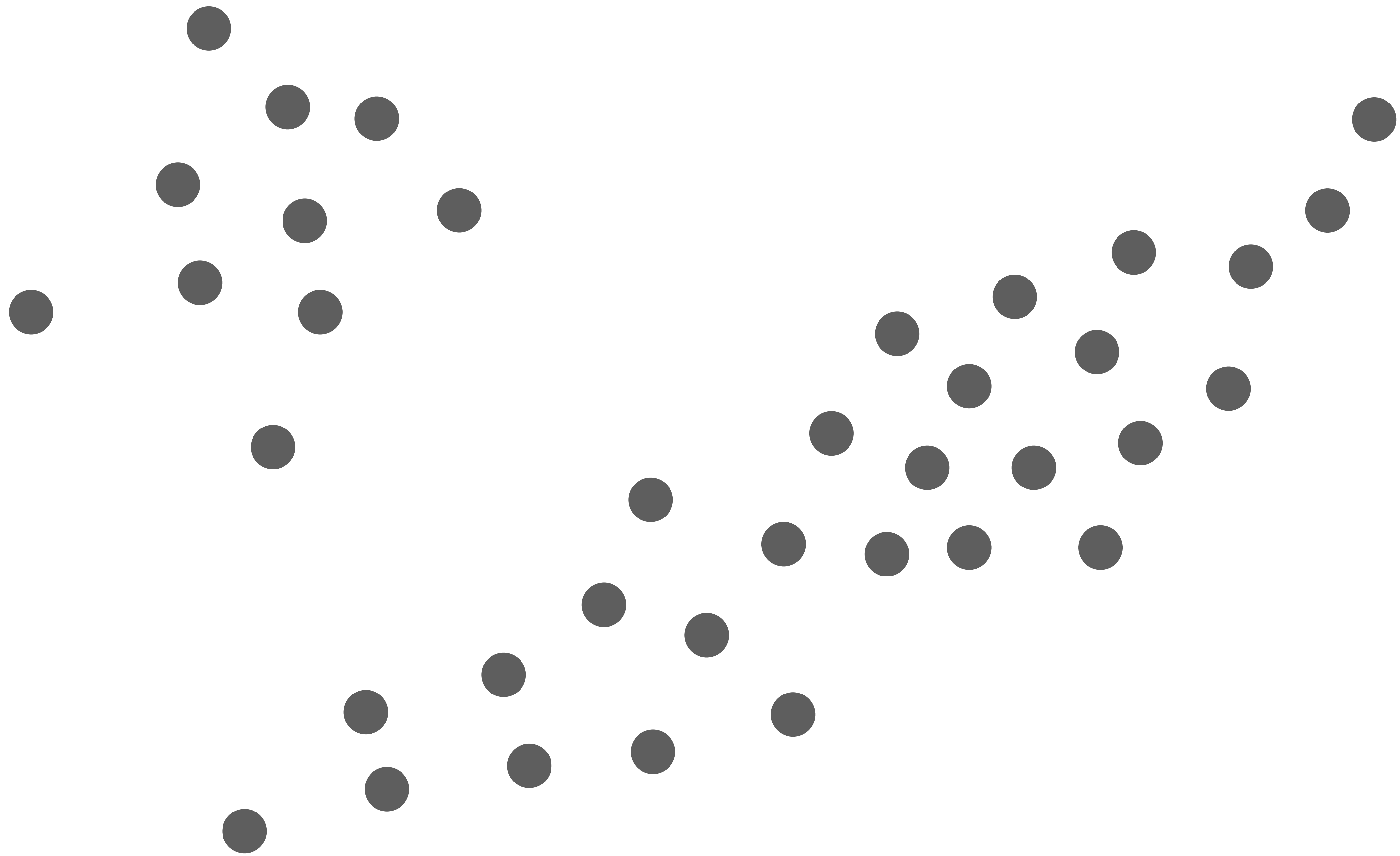
- *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment

*Interpretation: When the data appear as if they're from a GMM with true clusters that "look like circles of equal size", then k-means should work well*

# *k*-means should do well on this



# But not on this



# Relating *k*-means to GMMs

If the ellipses are all circles and have the same "skinniness"
(e.g., in the 1D case it means they all have same variance):

- *k*-means approximates the EM algorithm for GMMs
(as there is no need to keep track of cluster shape)

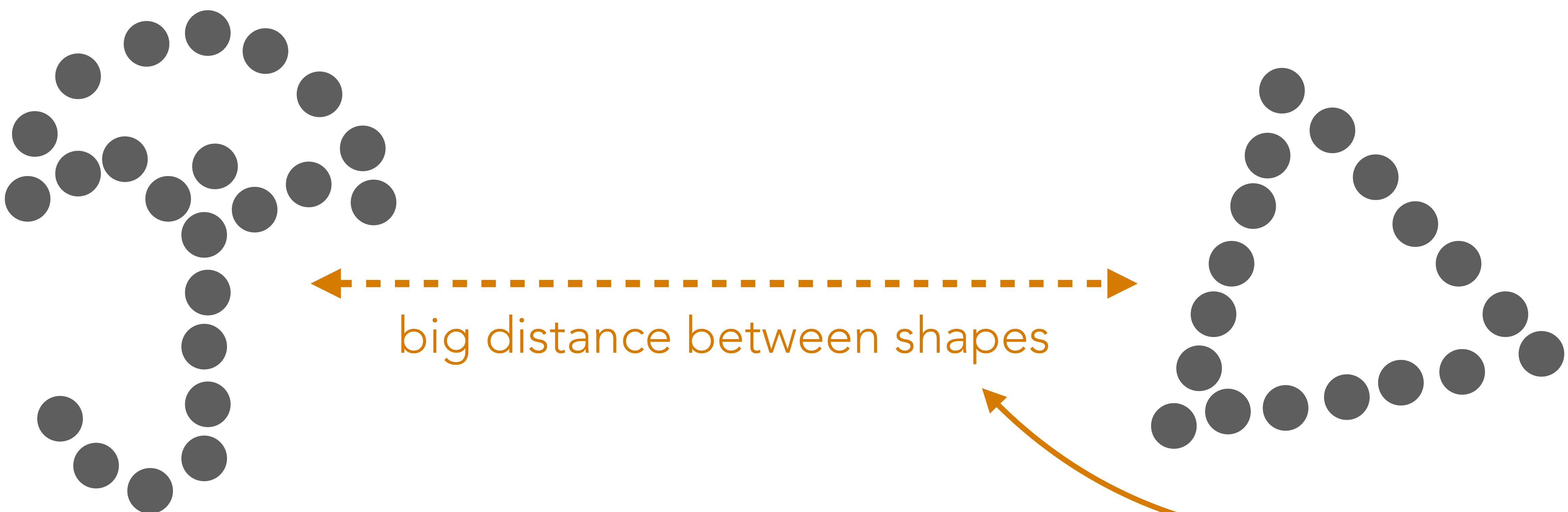
- *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment

*Interpretation: When the data appear as if they're from a GMM with true clusters that "look like circles of equal size", then k-means should work well*

This is *not* the only scenario in which *k*-means should work well

Even if data aren’t generated from a GMM, $k$-means and GMMs can still cluster correctly

This dataset obviously doesn't appear to be generated by a GMM




*k*-means with $k$ = 2, and 2-component GMM will both work well in identifying the two shapes as separate clusters

Key idea: the clusters are very ***well-separated***
(so that *many* clustering algorithms will work well in this case!)

# An Alternative Feature Vector Representation for Text: TF-IDF

Intuition: words that appear in more documents are likely less useful (same intuition as stop words!) — let's *downweight* these words!

$$
\textbf{Document}\quad
\begin{array}{c|cccc}
 & \multicolumn{4}{c}{\textbf{Word}} \\
 & 1 & 2 & \dots & d \\
\hline
1 & & & & \\
2 & & & & \\
\vdots & & & & \\
n & & & & 
\end{array}
$$

Term frequency (TF)

*i*-th row, *j*-th column: $\boxed{\text{\# times doc word } j \text{ appears in doc } i}$

multiply TF by $\log \dfrac{1}{\mathbb{P}(\text{document mentions word } j)}$

$$= \log \boxed{\frac{n}{\#\ \text{documents that mention word } j}}$$

Hack (additive smoothing): can add 1 to numerator & 1 to denominator

# An Alternative Feature Vector Representation for Text: TF-IDF

Intuition: words that appear in more documents are likely less useful (same intuition as stop words!) — let's *downweight* these words!

There are many TF-IDF variants! (Lots of hacks!)

**Word**

1 2 … *d*

**Document**

1
2
⋮
*n*

Term frequency (TF)

*i*-th row, *j*-th column: # times doc word *j* appears in doc *i*

$$\times \left[ 1 + \log \frac{n+1}{\#\ \text{documents that mention word}\ j + 1} \right]$$

Default TF-IDF weighting in sklearn

sklearn's default behavior further normalizes each row to have Euclidean norm 1

# An Alternative Feature Vector Representation for Text: TF-IDF

Demo

# Is clustering structure enough?

# (Flashback) GMM with *k* Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$

Gaussian covariance = $\Sigma_1$

$\cdots$

<u>Cluster *k*</u>

Probability of generating a point from cluster *k* = $\pi_k$

Gaussian mean = $\mu_k$

Gaussian covariance = $\Sigma_k$

*d*-dim.

*d*-by-*d* matrices

How to generate points from this GMM:

1. Flip biased coin (side 1 has probability $\pi_1$, …, side *k* has probability $\pi_k$)

   Let *Z* be the side that we got (it is some value 1, …, *k*)

2. Sample 1 point from the Gaussian from cluster *Z*

Each data point has a single true cluster assignment *Z* & is generated from the Gaussian for cluster *Z*

## In reality, a data point could have "mixed" membership and belong to multiple "clusters"

For example, for news articles, possible topics could be *sports*, *medicine*, *movies*, or *finance*

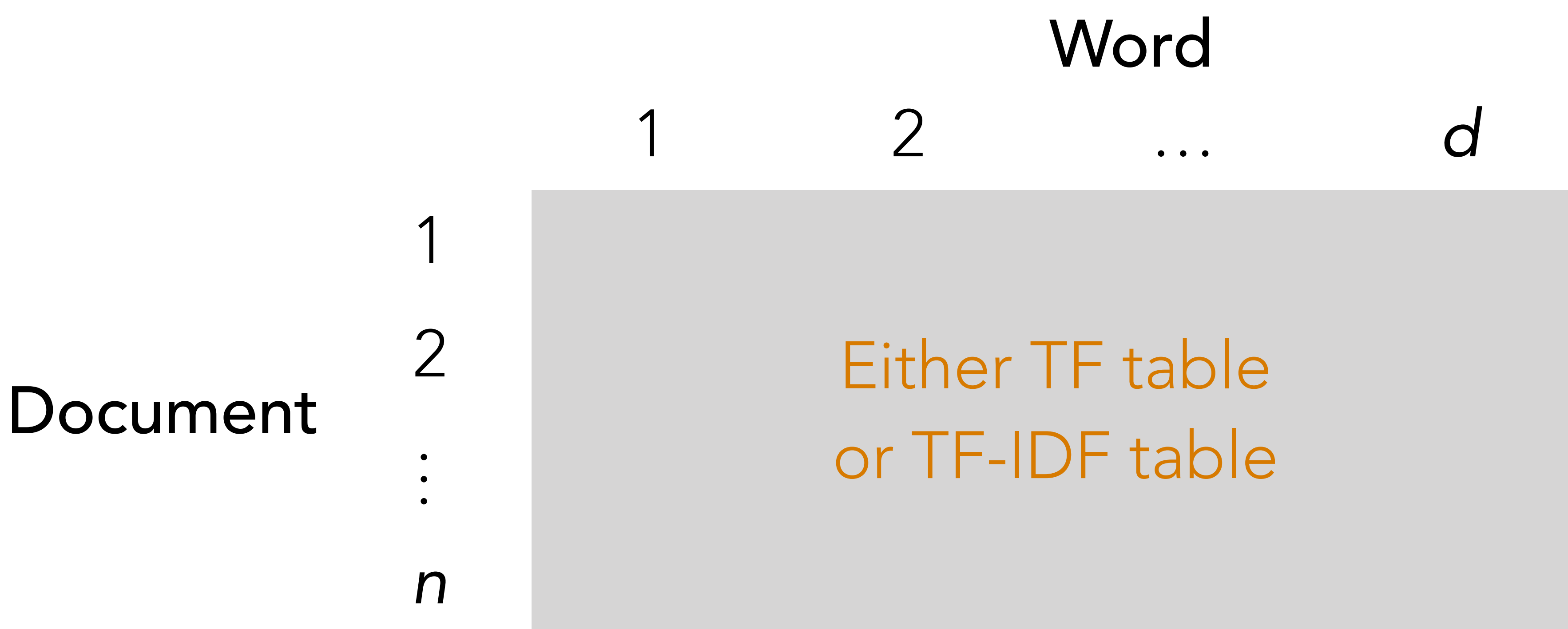
A news article could be about *sports* and also about *finance*

How do we model this?

# Topic Modeling:
# Latent Dirichlet Allocation (LDA)

- A generative model

- Input: "document-word" matrix, and pre-specified # topics *k*




- Output: what the *k* topics are (details on this shortly)

# LDA Generative Model Example

| | | Topic | |
|---|---|---|---|
| | | weather | food |
| Document | Alice's text | 0.1 | 0.9 |
| | Bob's text | 0.5 | 0.5 |

| | | Word | | | |
|---|---|---|---|---|---|
| | | cold | hot | apple | pie |
| Topic | weather | 0.3 | 0.7 | 0.0 | 0.0 |
| | food | 0.1 | 0.3 | 0.5 | 0.1 |

Each word in Alice's text is generated by:

1. Flip 2-sided coin for Alice
2. If weather: flip 4-sided coin for weather
   If food: flip 4-sided coin for food

# LDA Generative Model Example

| | | **Topic** | |
|---|---|---|---|
| | | weather | food |
| **Document** | Alice's text | 0.1 | 0.9 |
| | Bob's text | 0.5 | 0.5 |

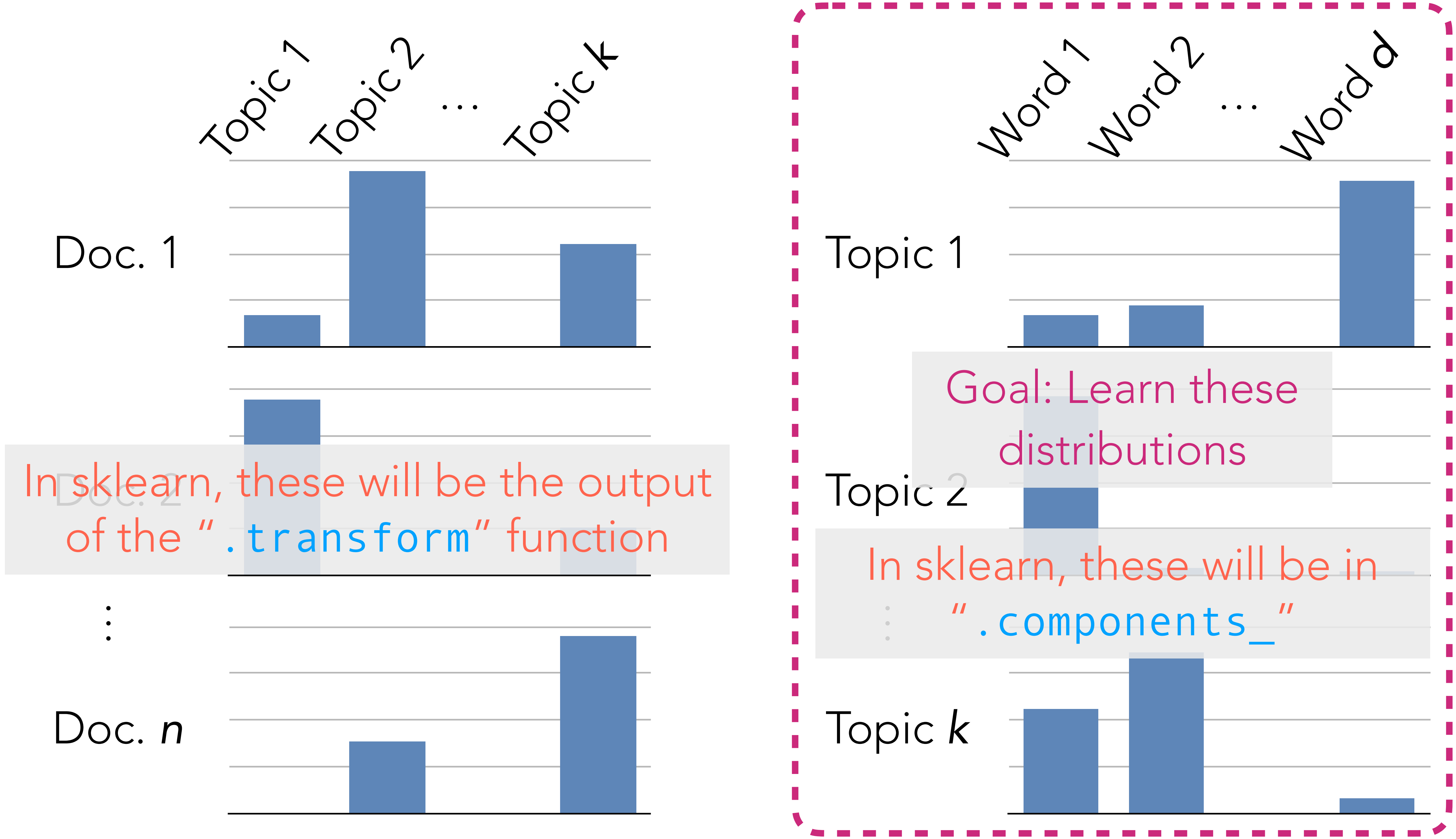
| | | **Word** | | | |
|---|---|---|---|---|---|
| | | cold | hot | apple | pie |
| **Topic** | weather | 0.3 | 0.7 | 0.0 | 0.0 |
| | food | 0.1 | 0.3 | 0.5 | 0.1 |

Each word in Bob's text is generated by:

1. Flip 2-sided coin for Bob
2. If weather: flip 4-sided coin for weather
   If food: flip 4-sided coin for food

"Learning the topics" means figuring out these 4-sided coin probabilities

# LDA Generative Model




LDA models each word in document *i* to be generated as:

1. Randomly choose a topic *Z* (use topic distribution for doc *i*)
2. Randomly choose a word (use word distribution for topic *Z*)

# Topic Modeling: Latent Dirichlet Allocation (LDA)

- A generative model

- Input: “document-word” matrix, and pre-specified # topics *k*


- Output: the *k* topics’ distributions over words

# LDA

Demo