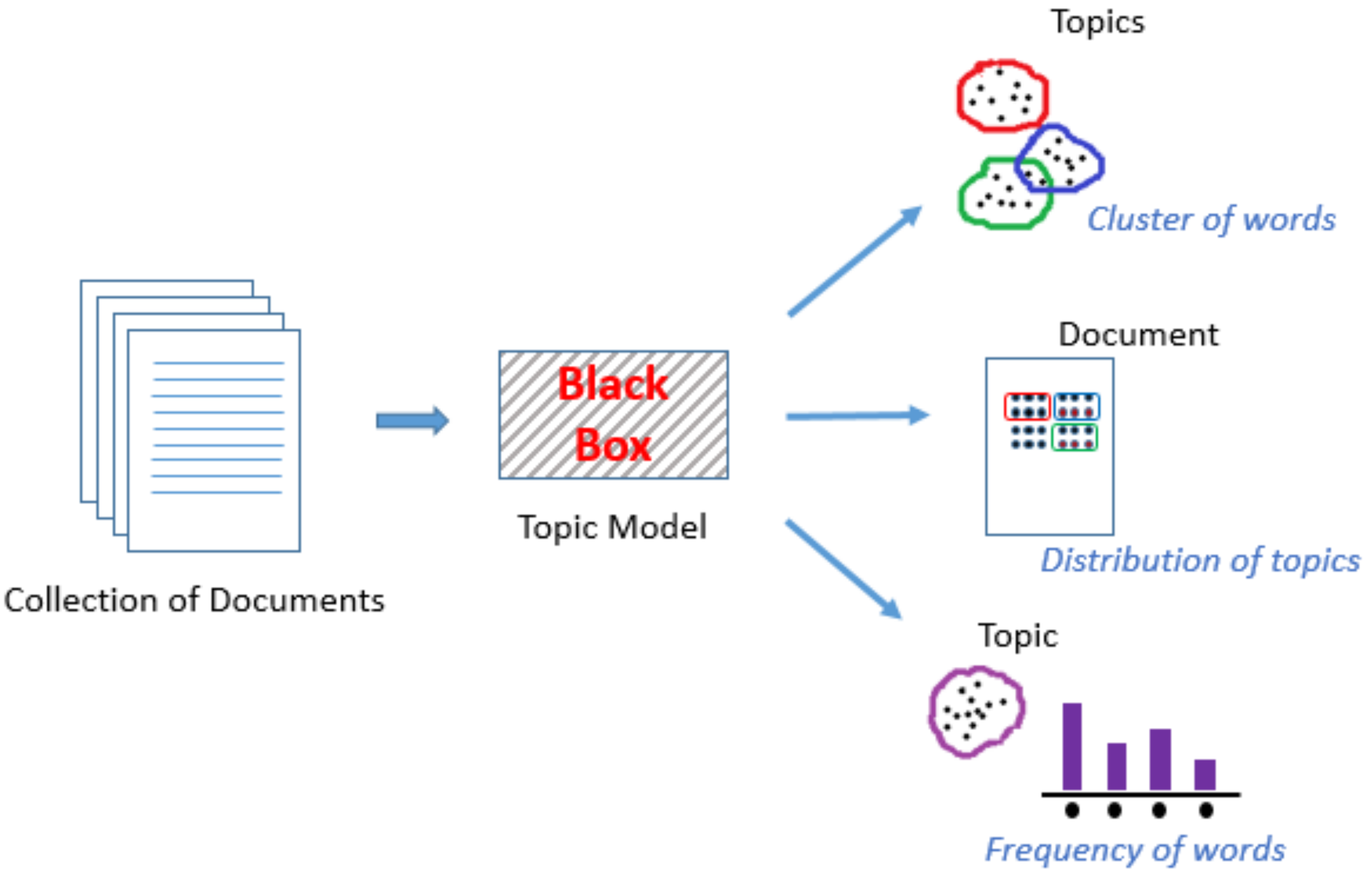
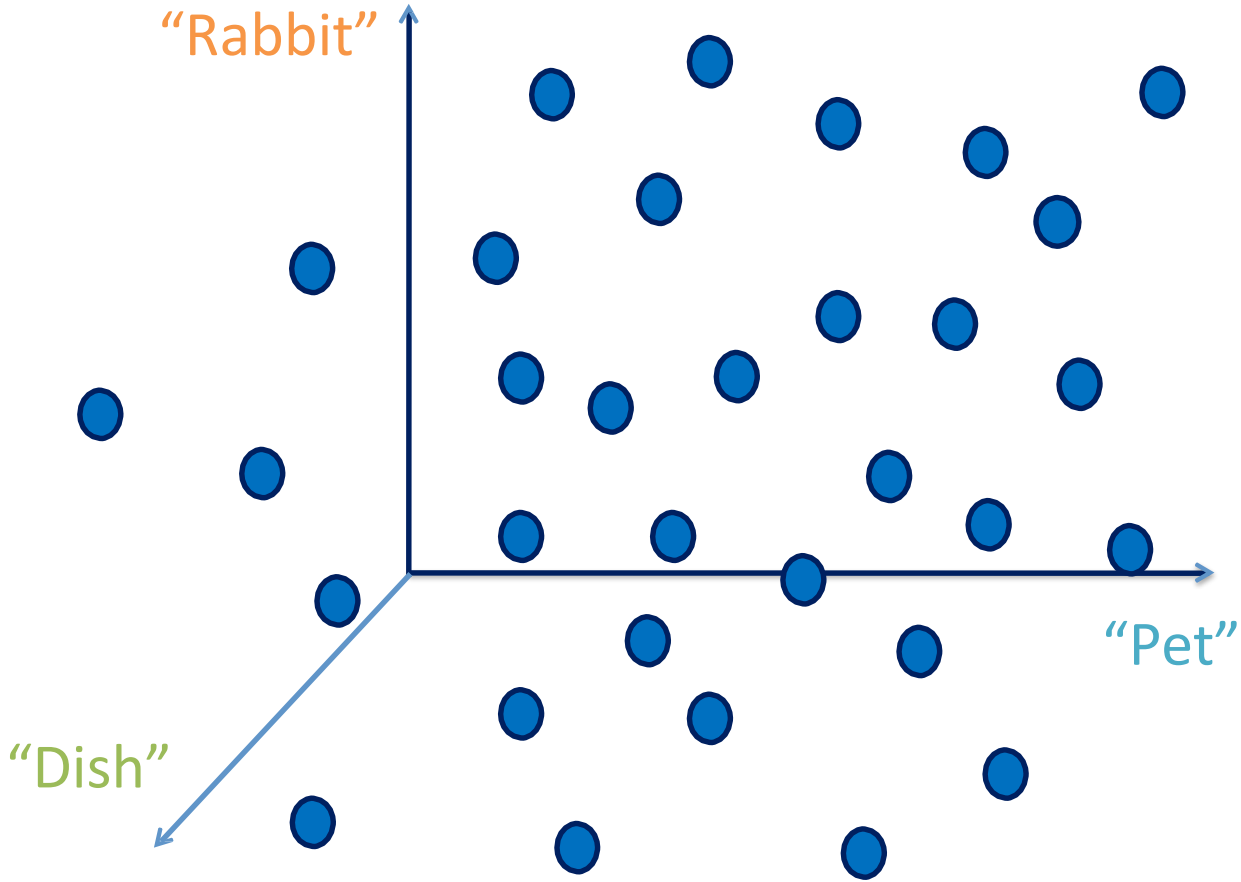
Matrix Decomposition Techniques

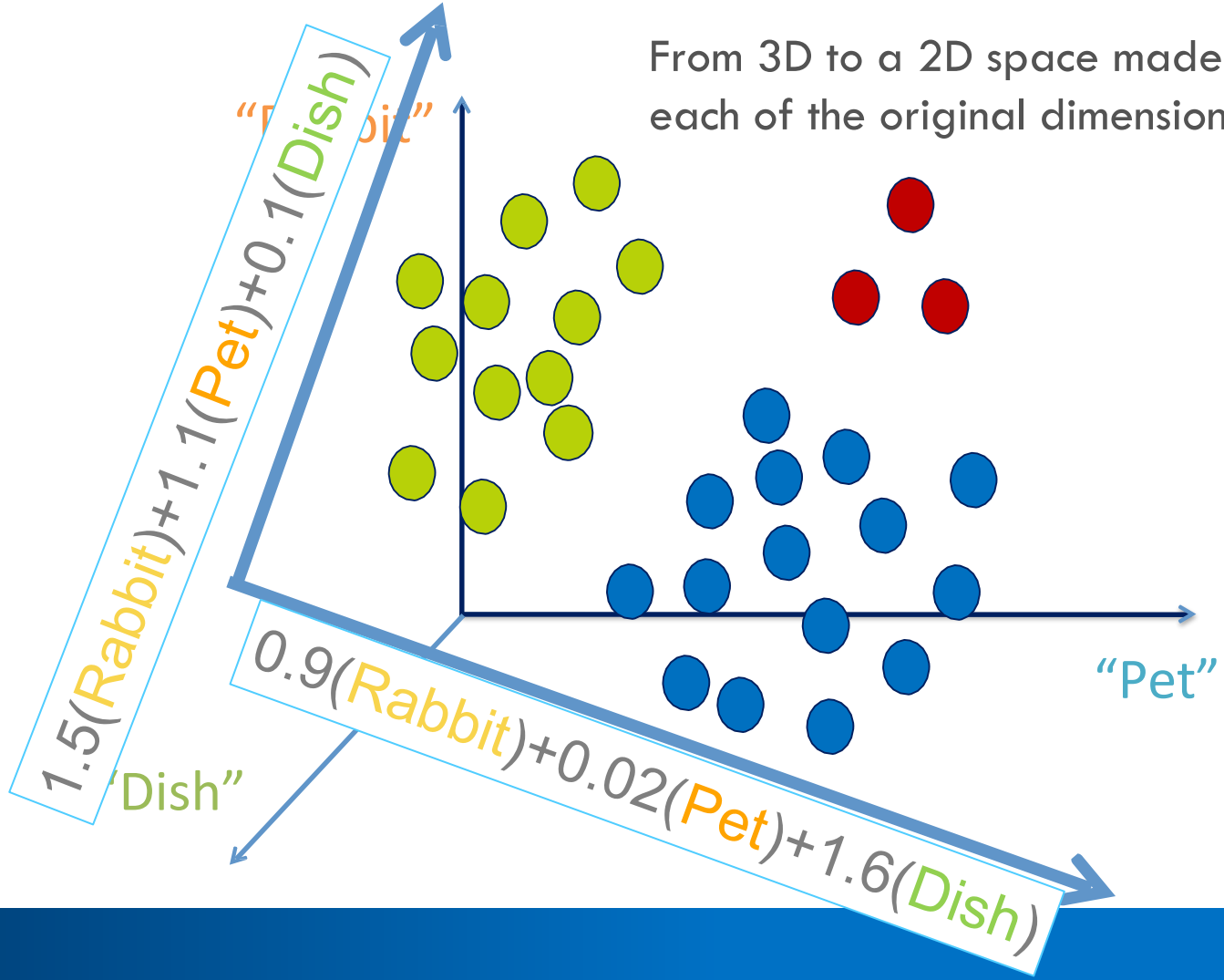

Topics
Cluster of words
Document
Distribution of topics
Topic
Frequency of words
Black
Box
Topic Model
Collection of Documents

Matrix Decomposition

- Last week we examined the idea of latent spaces and how we could use Latent Dirichlet Allocation to create a ”topic space.”
- LDA is not the only method to create latent spaces, so today we’ll investigate some more “mathematically rigorous” ways to accomplish the same task.
- Let’s start by reviewing latent spaces…


“Rabbit”
“Pet”
“Dish”

From 3D to a 2D space made of a bit of each of the original dimensions.

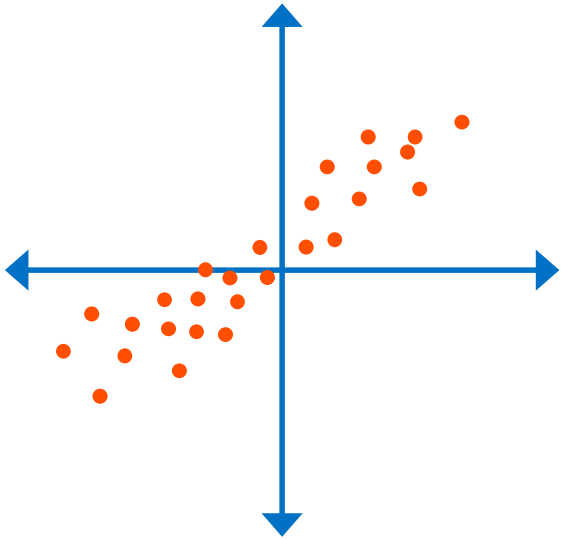


Latent Spaces – A Reminder

- Latent spaces are a "hidden" set of axes that we use to look at the information in a different way.
- For NLP, we typically want to go from a "word space," which may be 1000s of dimensions, to a smaller dimensionality "combinations of words space." We called this a "topic space" last week.
- We used LDA to decide on how the topics were defined last week – it's not our only option.
- Clustering can work MUCH better in these lower dimensional spaces due to the curse of dimensionality.
- Let's look at some other ways to get to latent spaces.

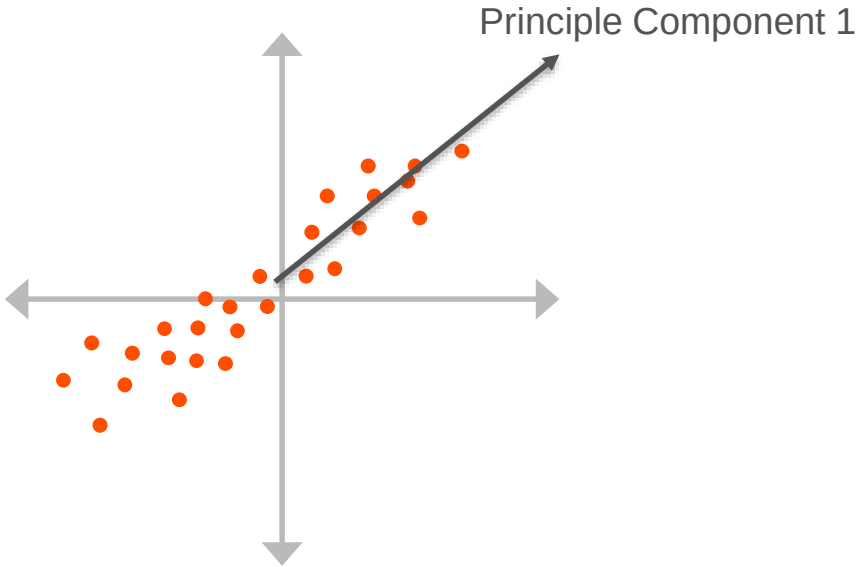
Dimensionality Reduction without NLP

- Dimensionality reduction is an “old” technique in terms of mathematical problems. The most famous version is Principal Component Analysis (PCA).



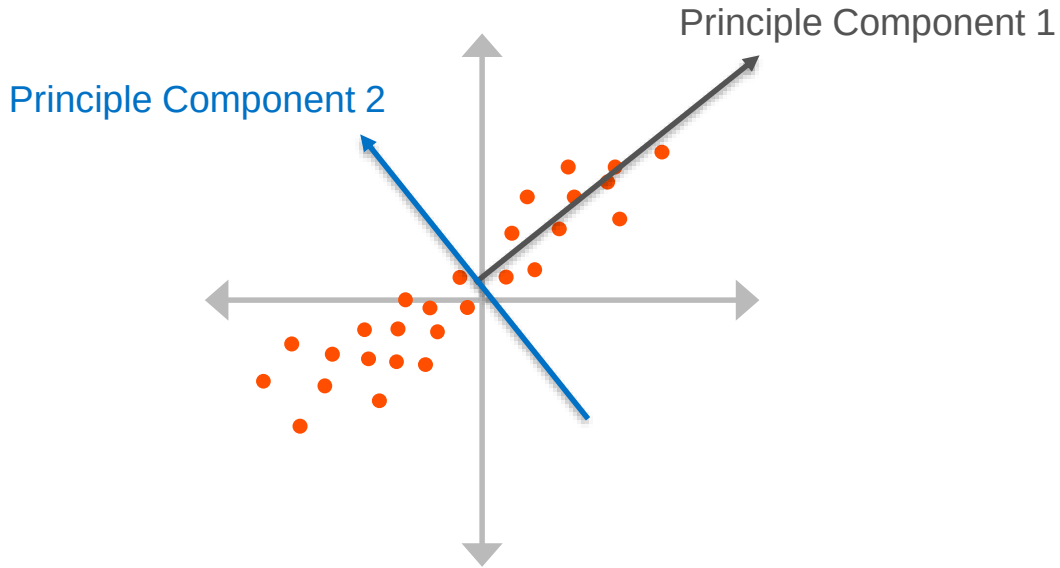
Dimensionality Reduction without NLP

- A principle component analysis shifts the axes around to find a set of axes that capture most of the variance in the data. In this example, with just this one axes, we're getting the majority of how the data changes.



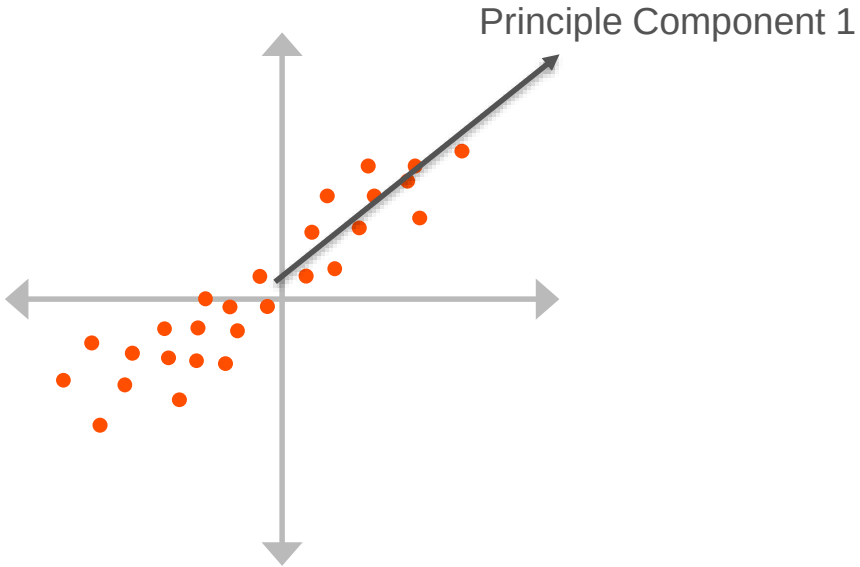
Dimensionality Reduction without NLP

- If we add back the same number of components we capture the data fully. However, we don't HAVE to add that last component back. We could just convert to a lower dimensionality representation of the data.



Dimensionality Reduction without NLP

- If we stick with one dimension in this example, we've captured MOST of what makes each data point special, and we're only storing half as much data (PC1 vs [x,y]).



Simple PCA Example in Python

Input:

```
from sklearn.datasets import load_iris

iris_data = iris.data
iris_data = pd.DataFrame(iris.data)
iris_data.columns = [‘Sepal_Length', ‘Sepal_Width', ‘Petal_Length', ‘Petal_Width']
print(iris_data.head()) # print function requires Python 3
```

Output:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Simple PCA Example in Python

Input:

```
from sklearn.decomposition import PCA

pca_1 = PCA(n_components=1)
iris_1_component = pca_1.fit_transform(iris_data)
print(pca_1.components_[0]) # print function only works in Python 3
```

Output:

```
array([ 0.36158968, -0.08226889,  0.85657211,  0.35884393])
```

First principal component is:
0.36 * Sepal_length + (-0.08 * Sepal_width) + 0.86 * Petal_length + 0.36 * Petal_width

Simple PCA Example in Python

Input:

```
from sklearn.decomposition import PCA

iris_1_component_df = pd.DataFrame(iris_reduced,
                                   columns=[“new_component”])

print(iris_1_component_df.head()) # print function only works in Python 3
```

Output:

	new_component
0	-2.684207
1	-2.715391
2	-2.889820
3	-2.746437
4	-2.728593

Values of observations on this new component

Principle Component Analysis: The Math

- In PCA, we start by finding the covariance matrix. Then we can find eigenvalues and eigenvectors on that matrix.
- Let's imagine that we have some covariance matrix we're calling A. We can imagine there is some linear transformation X that can be equivalent to applying a scaler to A.

$$\mathbf{XA} = \lambda\mathbf{A}$$

$$\mathbf{X}(\lambda\mathit{I} - \mathbf{A}) = 0$$

$$det(\lambda\mathit{I} - \mathbf{A}) = 0$$

- This last line is a set of equations that we can solve for lambda's – the eigenvalues. This also gives us the eigenvectors, with equation 1.

Principle Component Analysis

- The eigenvectors will define the new axis set, since the eigen-decomposition is asking “what axes can we find that make the covariance matrix as small (and diagonal) as possible.”
- The new space is a latent space. Great!
- So, can we just use this with text?

Principle Component Analysis

- Not really.
- One way to calculate principal components uses the covariance matrix of the data.
- A covariance matrix requires that the variance of each data point be truly meaningful. For example, does the appearance of x1 precipitate a change in x2? For something like CountVectorizer, we're approximating those relationships in a multinomial way, but we're not really setup to truly understand the covariance.

Enter: Singular Value Decomposition (SVD)

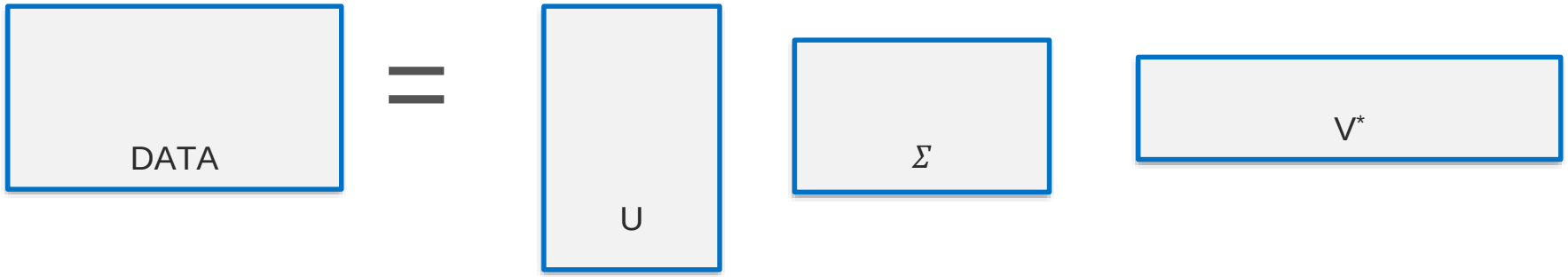
- Singular value decomposition is another method of finding the principle components, but without relying on the covariance matrix. We can interact with the data directly to get out resulting latent space. We also don't need a square matrix like in PCA.
- SVD results in taking one matrix: our data, and converting it to a product of three matrices.

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^*$$

- This separation is called a "matrix factorization." We're taking one matrix and breaking it into sub-matrices or "factors."

Enter: Singular Value Decomposition (SVD)

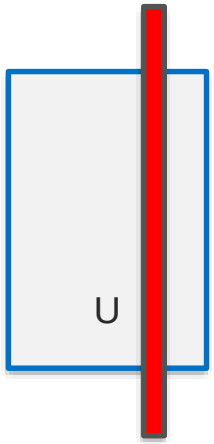
$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^*$$



Singular Value Decomposition (SVD)

Why does converting to 3 matrices help?

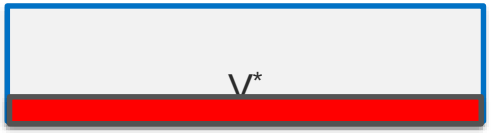
- The columns of U are the eigenvectors of MM*, meaning those eigenvectors are informed about the data. They know something about our data set.



Singular Value Decomposition (SVD)

Why does converting to 3 matrices help?

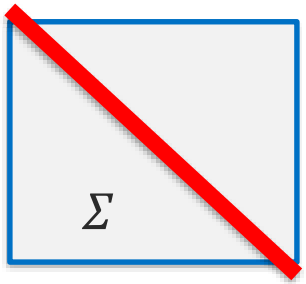
- The rows of V are the eigenvectors of M*M, so they also know something about our data.



Singular Value Decomposition (SVD)

Why does converting to 3 matrices help?

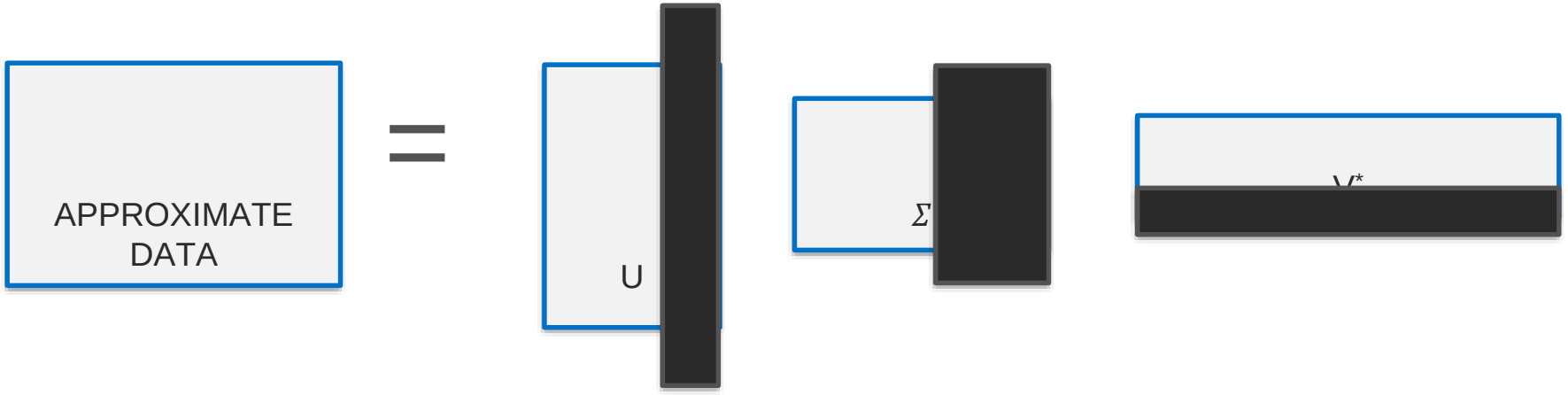
- Σ is made up of the square roots of the eigenvalues from both MM* and M*M, placed along the diagonal. These are known as the "Singular Values."



Singular Value Decomposition (SVD)

- If all the vectors and singular values are aligned properly, these three matrices multiply together to give us exactly our data back.
- However, what if we chopped off the vectors and singular values that are small, and only contribute a little bit to giving us our data back? We'd then be approximating our data and getting close to the right answer… but we'd have fewer dimensions!
- The remaining eigenvectors can help us determine our latent space axes as well.

Truncated SVD



SVD in Python

Input:

```
import numpy as np

U, sigma, V = np.linalg.svd(iris_data)

iris_reconst = np.matrix(U)[:, :4] * np.diag(sigma) * np.matrix(V)

print(np.array(iris_reconst[:5, :])) # first 5 rows of reconstructed data
print(np.array(iris_data)[:5,]) # first 5 rows of original data
```

Output:

Identical

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

SVD in Python (reducing dimensions)

Input:

```
import numpy as np

U, sigma, V = np.linalg.svd(iris_data)

iris_reconst = np.matrix(U)[:, :2] * np.diag(sigma[:2]) * np.matrix(V[:2])

print(np.array(iris_reconst[:5, :])) # first 5 rows of reconstructed data
print(np.array(iris_data)[:5,]) # first 5 rows of original data
```

Output:

Close, with only one dimension

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.7,  3.2,  1.5,  0.3],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.3],
       [ 5.1,  3.5,  1.4,  0.2]])
```

Latent Semantic Analysis (LSA)

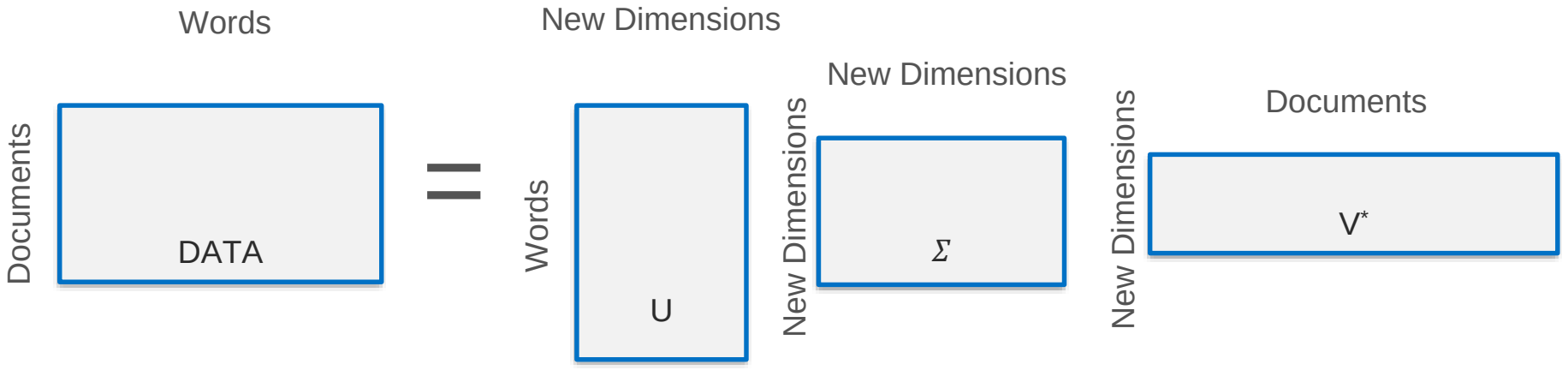
- If we apply SVD, and keep track of the how the words (features) are combined in the latent space, it's called Latent Semantic Analysis (LSA).
- Essentially, we do SVD with reducing the dimensionality ("Truncated SVD"). Then we keep track that the new 1st dimension is made up of mostly old dimensions that were called "bat, glove, base, and outfield."
- That sounds an awful lot like topic modeling.

Topic Modeling

- Documents are not required to be one topic or another. They will be an overlap of many topics.
- For each document, we will have a ‘percentage breakdown’ of how much of each topic is involved.
- The topics will not be defined by the machine, we must interpret the word groupings.

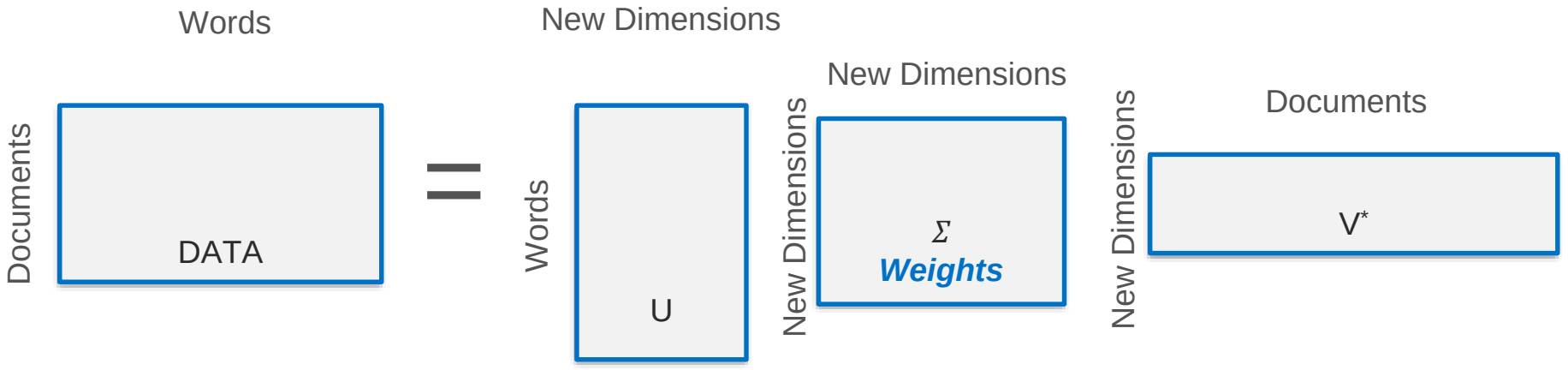
	Pet Rabbits	Eating Rabbit
“I love my pet rabbit.”	87%	13%
“That’s my pet rabbit’s favorite dish.”	42%	58%
“She cooks the best rabbit dish.”	14%	84%

Why does LSA work?



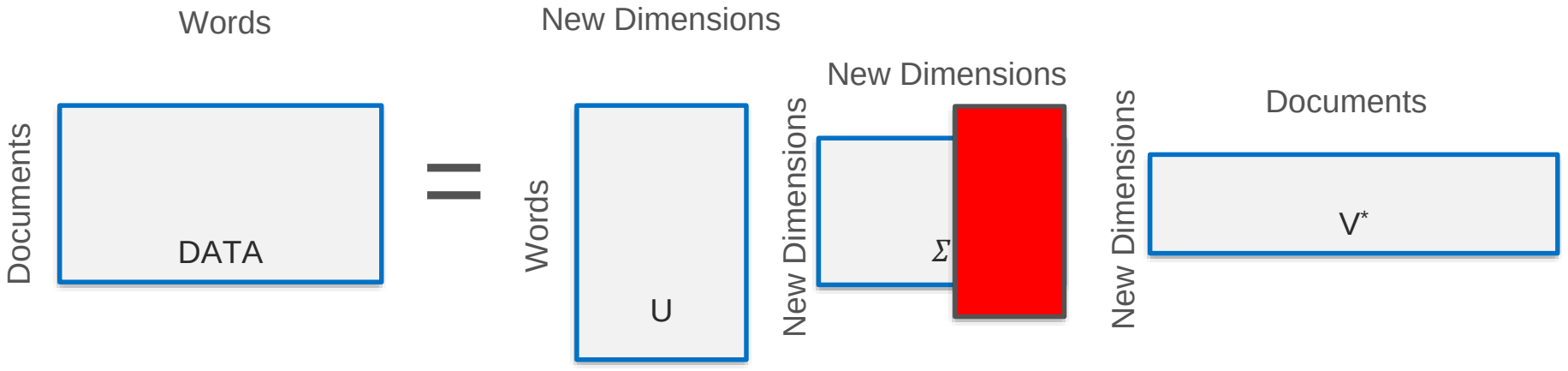
We end up creating a system where the words and documents are both understood in our latent space individually. We can then recombine them to get our original data back.

Why does LSA work?



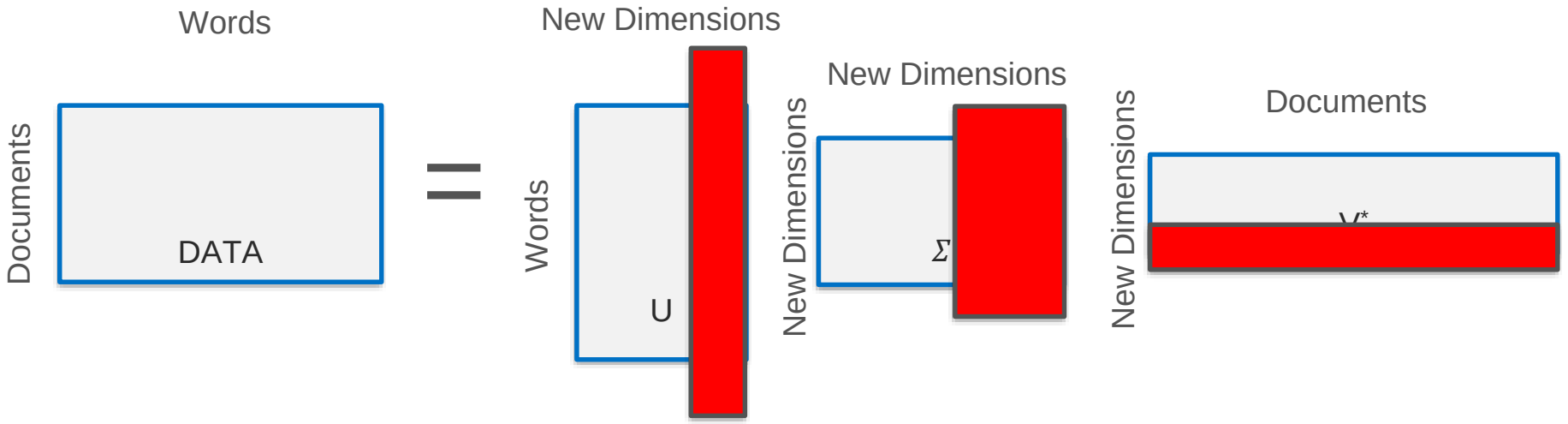
The Σ matrix acts as a set of weights to determine how the words and documents combine together in the latent space to correctly represent the real data.

Why does LSA work?



If we truncate our analysis to a certain number of topics, we're essentially saying let's only take the most USEFUL hidden dimensions. That's akin to setting everything in the red region of Σ to 0. That means the word*document combinations no longer contribute.

Why does LSA work?



If these dimensions don't contribute, we're now limited to a smaller number of "topics" to represent our data. Which means we won't be able to reproduce the data exactly, but as long as we don't cut too many topics, we should still get a good representation of the data.

Latent Semantic Analysis (LSA)

So an LSA pipeline for topic modeling would look like:

1. Preprocess the text
2. Vectorize the text (TF-IDF, Binomial, Multinomial…)
3. Reduce the dimensionality with SVD, keeping track of the feature composition of the new latent space
4. Investigate the output ”topics”

LSA with Python - Dataset Creation

Input:

```
# Dataset creation same as in prior weeks

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups

ng_train = fetch_20newsgroups(subset='train',
                              remove=('headers', 'footers', 'quotes'))

tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2),
                                   stop_words='english',
                                   token_pattern="\\b[a-z][a-z]+\\b",
                                   lowercase=True,
                                   max_df = 0.6)

tfidf_data = tfidf_vectorizer.fit_transform(ng_train.data)
```

LSA with Python - Modeling

Input:

```
from sklearn.decomposition import TruncatedSVD

lsa_tfidf = TruncatedSVD(n_components=10)

lsa_tfidf_data = lsa_tfidf.fit_transform(tfidf_data)

show_topics(lsa_tfidf,tfidf.get_feature_names()) # a function I wrote
```

Output:

```
Topic 0: graphics, image, files, file, thanks, program, format, ftp, windows, gif

Topic 1: god, atheism, people, atheists, say, exist, belief, religion, believe,
bible

Topic 2: just, aspects graphics, aspects, group, graphics, groups, split, posts
week, week group, think
```

Non-Negative Matrix Factorization (NMF)

- SVD isn't the only way to do matrix factorization. That means LSA isn't the only way to do topic modeling with matrix factorization.
- Non-negative Matrix Factorization is another mathematical technique to decompose a matrix into sub-matrices.
- NMF assumes there are no negative numbers in your original matrix.

Non-Negative Matrix Factorization (NMF)

- In NMF, we break the data matrix M down into two component matrices.
- W is the features matrix. It will be of the shape (number of words, number of topics). So a system with 5000 words and 10 topics will have W be (5000,10).
- H is the “coefficients matrix.” It’s of the shape (number of topics, number of documents).
- The combination of these two tells us how each word contributes to each document, via some assumed middle dimension of size 10 (in this example). That hidden dimension is our topic space!

Non-Negative Matrix Factorization (NMF)

- Since NMF assumes no negative values, so we have to be careful about any specialized document-vectorization we do. For instance, depending on how you normalize TF-IDF, it may be possible to end up with a negative value.
- NMF and LSA are very similar in technique, but can end up with different results due to different underlying assumptions.
- Always try both, to see which is working better for your dataset!

NMF with Python

Input:

```
from sklearn.decomposition import NMF
nmf_tfidf = NMF(n_components=10)
nmf_tfidf_data = lsa_tfidf.fit_transform(tfidf_data)
show_topics(nmf_tfidf,tfidf.get_feature_names()) # a function I wrote
```

Output:

```
Topic 0: jpeg, image, gif, file, color, images, format, quality, version, files
Topic 1: edu, graphics, pub, mail, ray, send, ftp, com, objects, server
Topic 2: jesus, matthew, prophecy, people, said, messiah, isaiah, psalm, david, king
```

Some rules of thumb, that you will break:

- In practice, LSA/NMF tends to outperform LDA on small documents. Things like tweets, forum posts, etc. That's not always true, but worth keeping in mind.
- In contrast, doing LSA/NMF on HUGE documents can lead to needing 100s of topics to make sense of things. LDA usually does a better job on big documents with many topics.
- LSA and NMF are similar techniques and one doesn't systematically outperform the other.
- LSA and NMF both perform differently with TF-IDF vs Count Vectorizer vs Binomial Vectorizers. Try them all.

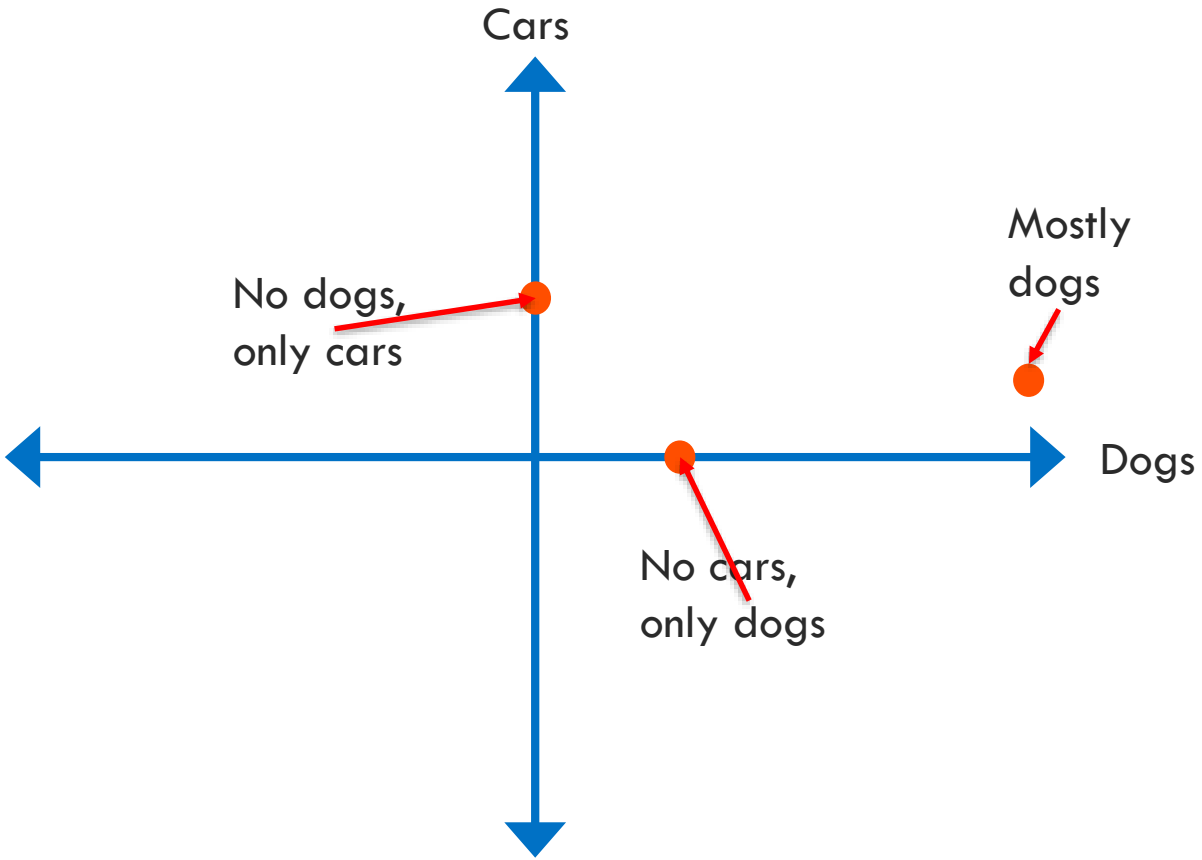
Some rules of thumb, that you will break:

- If you're getting topics that make sense, and your code isn't buggy, your process is good. There's a lot of art in NLP, and especially in topic modeling.

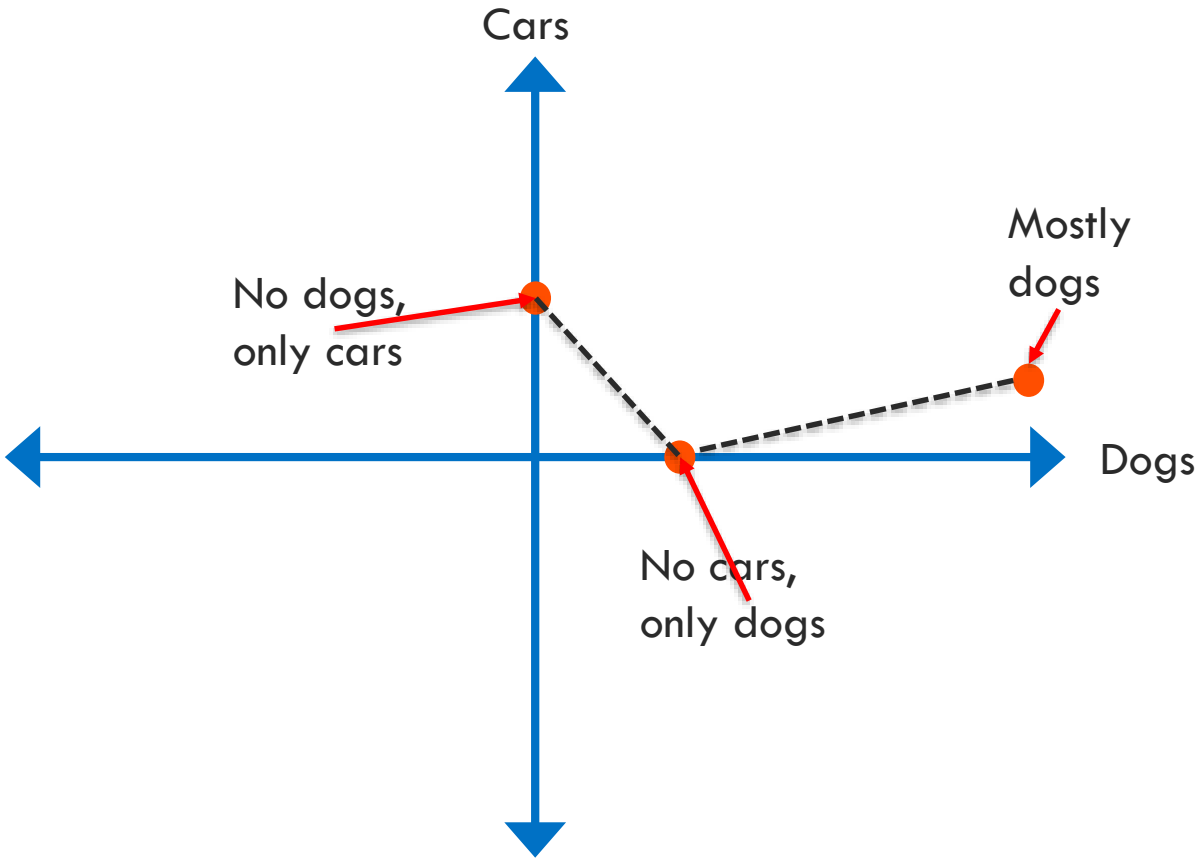
Similarity

- Two similar documents should be near each other in the latent space, since they should have overlapping topics.
- We can't rely on Euclidean distance in this space, because we want to count documents as similar whether they have 900 mentions of a certain topic, or only 600 mentions of a certain topic.
- Cosine distance measures this similarity well.

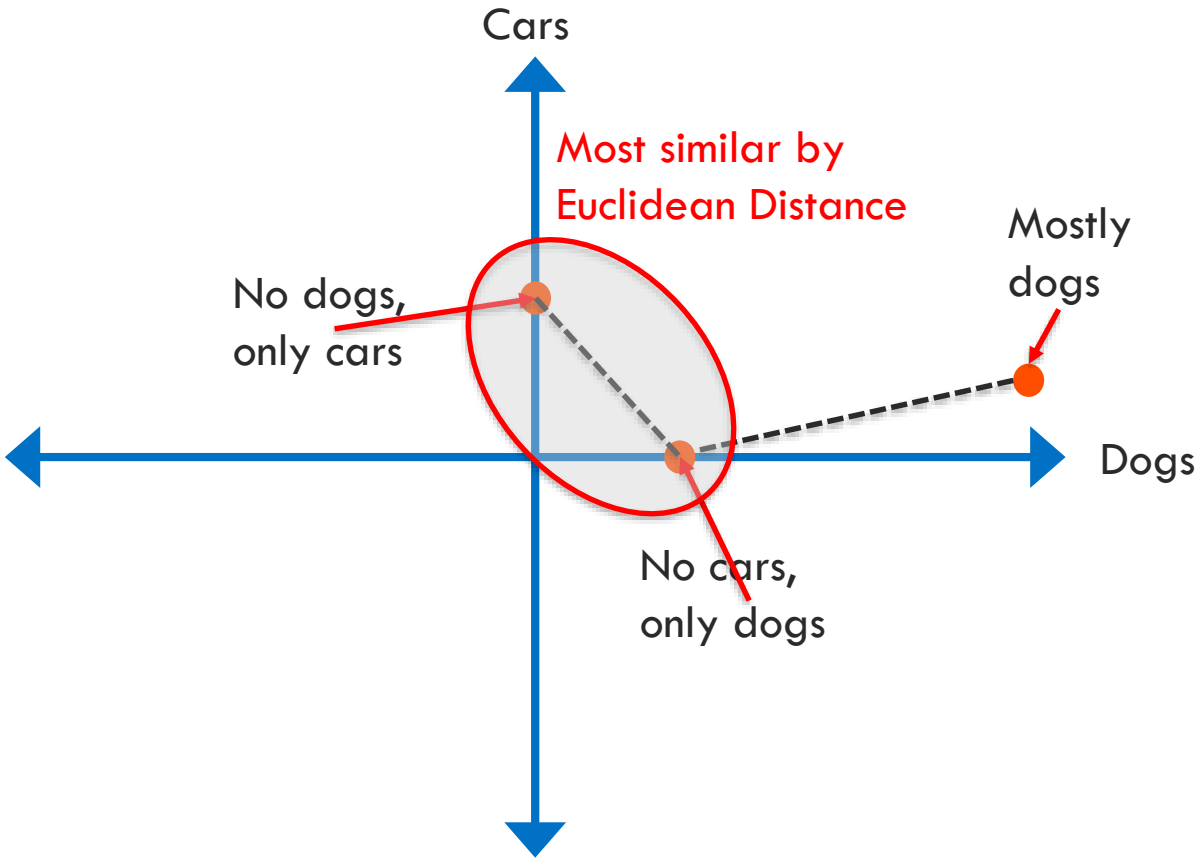
Similarity



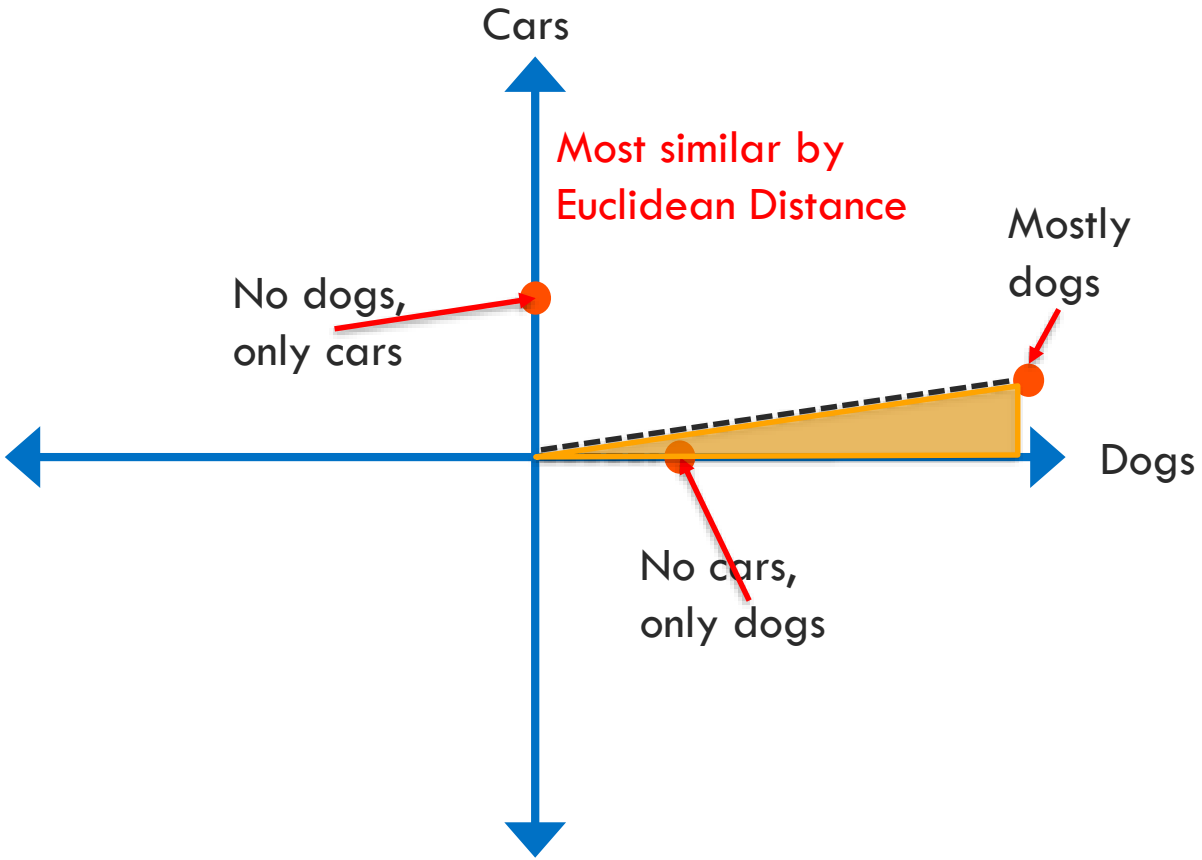
Similarity



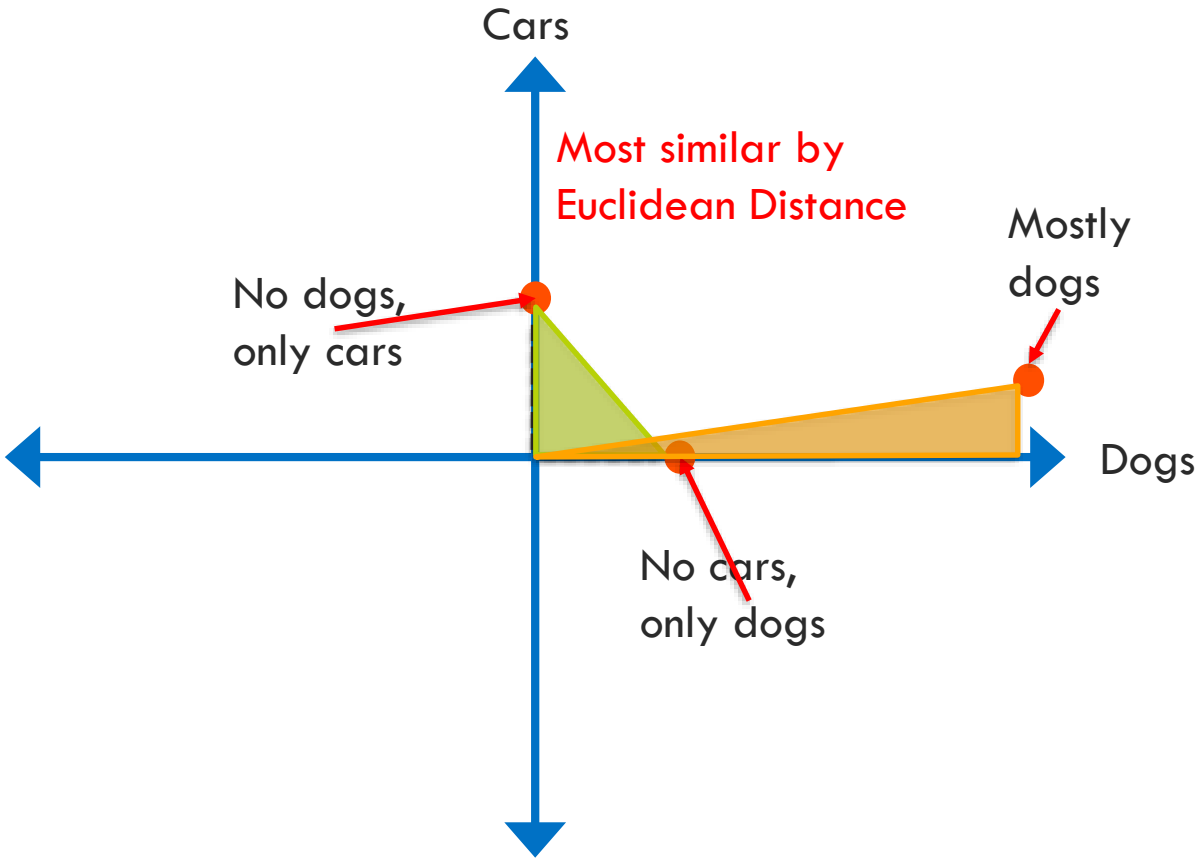
Similarity


Cars
Most similar by
Euclidean Distance
Mostly
dogs
No dogs,
only cars
Dogs
No cars,
only dogs

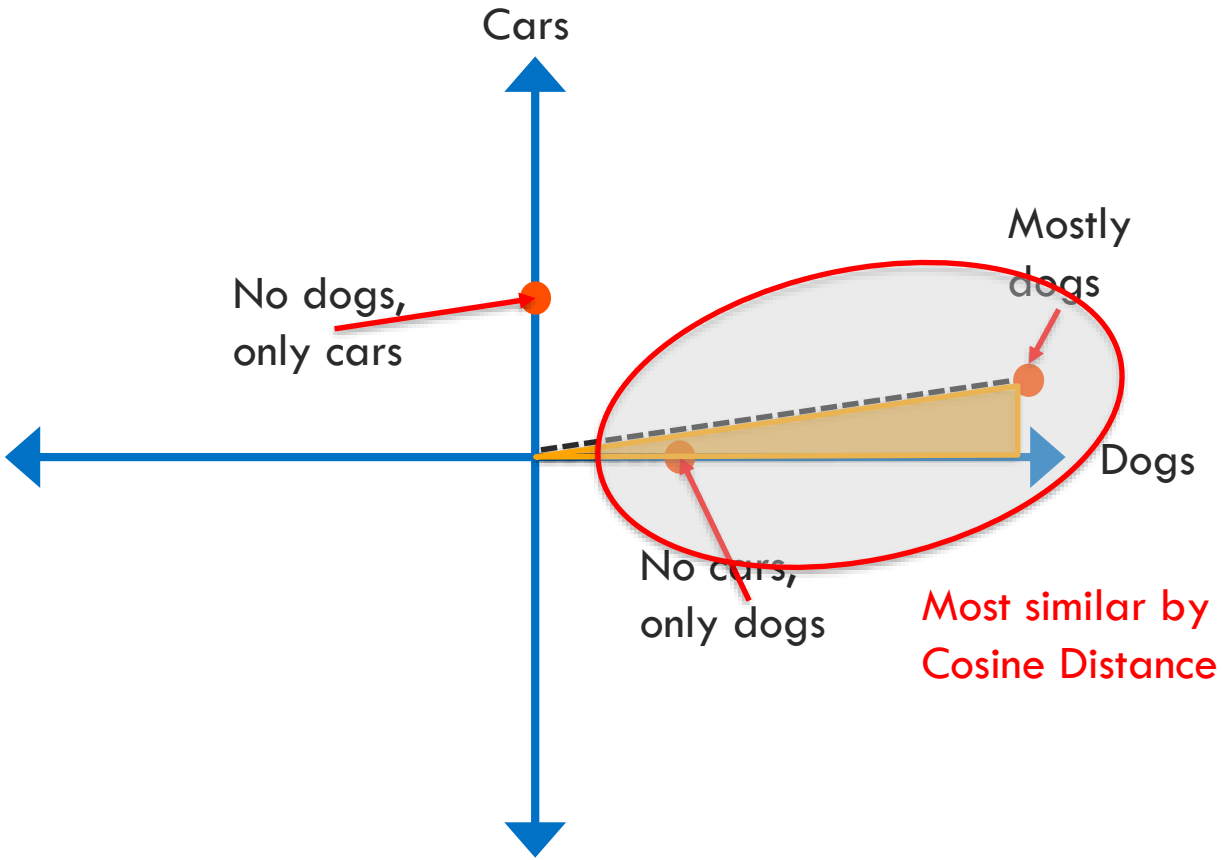
Similarity


Cars
Most similar by
Euclidean Distance
Mostly
dogs
No dogs,
only cars
Dogs
No cars,
only dogs

Similarity


Cars
Most similar by
Euclidean Distance
Mostly
dogs
No dogs,
only cars
Dogs
No cars,
only dogs

Similarity


Cars
No dogs,
only cars
Mostly
dogs
Dogs
No cars,
only dogs
Most similar by
Cosine Distance

Similarity

- By using cosine distance in the topic space, the length of the documents doesn't matter – either relative or absolute.
- Comparisons in the latent space can allow us to make recommendations: "If you like this, you'll probably like this!"
- If we know each documents breakdown in the latent space, we can find documents about the same topics.

Clustering

- In our latent spaces, if we've done our dimensionality reduction, we should get much more meaningful clusters.
- LSA/NMF allow us to use clustering, just like LDA does.
- We also know that distances are more meaningful. So a two documents that are very similar will have very similar place in the topic space. Example: two articles about baseball should be very high on the "bat, glove, outfield" topic and very low on the "gif, jpg, png" topic.
- We can use this to document recommendation too!

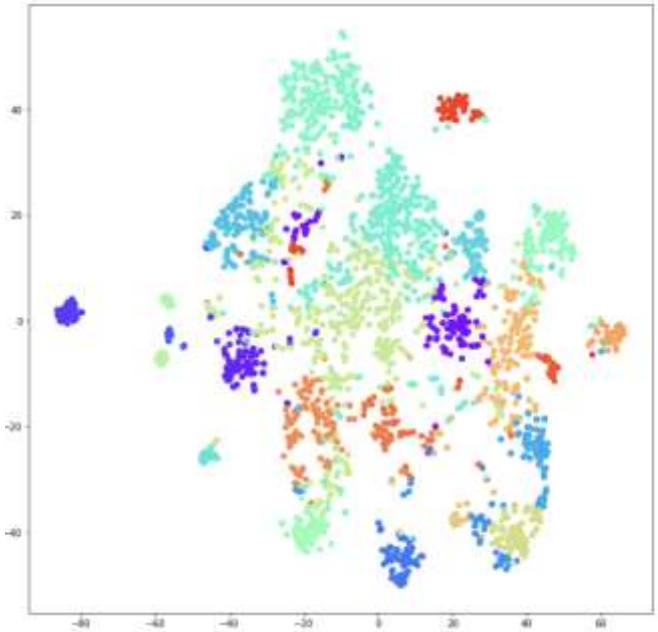
Clustering with Python

Input:

```
from sklearn.cluster import Means

clus = KMeans(n_clusters=25,random_state=42)

# Note: similarity in KMeans determined by Euclidean distance

labels = clus.fit_predict(lsa_tfidf_data)
```

Clustering with Python

- 2D visualization of resulting 25 clusters:



Topic Modeling Summary – A Reminder

- Choose some algorithm to figure out how words are related
  (LDA, LSA, NMF)
- Create a latent space that makes use of those in-document correlations to transition from a “word space” to a latent “topic space”
- Each axis in the latent space represents a topic
- We, as the humans, must understand the meaning of the topics. The machine doesn’t understand the meanings, just the correlations.

Software

Appendix

Input:

```
from gensim import corpora, models, utils

lsi = models.LsiModel(corpus, id2word=id2word, num_topics=200)

lsi.print_topics(num_topics=2, num_words=5)
```

Output:

```
topic #0(3.341): 0.644*"system" + 0.404*"user" + 0.301*"eps" + 0.265*"time" +
0.265*"response"

topic #1(2.542): 0.623*"graph" + 0.490*"trees" + 0.451*"minors" + 0.274*"survey" + -
0.167*"system"
```

LSA with Python (GenSim)

Input:

```
from gensim import corpora, models, utils

lsi = models.LsiModel(corpus, id2word=id2word, num_topics=200)

lsi.print_topics(num_topics=2, num_words=5)
```

Output:

```
topic #0(3.341): 0.644*"system" + 0.404*"user" + 0.301*"eps" + 0.265*"time" +
0.265*"response"

topic #1(2.542): 0.623*"graph" + 0.490*"trees" + 0.451*"minors" + 0.274*"survey" + -
0.167*"system"
```