# Vector Space Models

Vector spaces are fundamental in many applications in NLP. If you were to represent a word, document, tweet, or any form of text, you will probably be encoding it as a vector. These vectors are important in tasks like information extraction, machine translation, and chatbots. Vector spaces could also be used to help you identify relationships between words as follows:
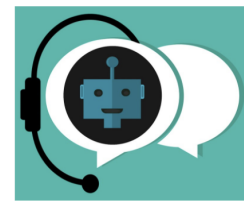
- You eat _cereal_ from a _bowl_

- You _buy_ something and someone else _sells_ it
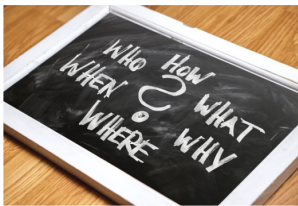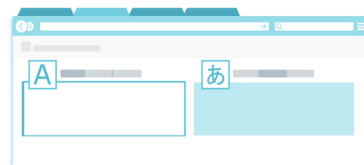
| Information Extraction | Machine Translation | Chatbots |

The famous quote by Firth says, **"You shall know a word by the company it keeps".** When learning these vectors, you usually make use of the neighboring words to extract meaning and information about the center word. If you were to cluster these vectors together, as you will see later in this specialization, you will see that adjectives, nouns, verbs, etc. tend to be near one another. Another cool fact is that synonyms and antonyms are also very close to one another. This is because you can easily interchange them in a sentence and they tend to have similar neighboring words!

Word by Word and Word by Doc.

Word by Word Design

We will start by exploring the word by word design. Assume that you are trying to come up with a vector that will represent a certain word. One possible design would be to create a matrix where each row and column corresponds to a word in your vocabulary. Then you can iterate over a document and see the number of times each word shows up next to each other word. You can keep track of the number in the matrix. In the video I spoke about a parameter K. You can think of K as the bandwidth that decides whether two words are next to each other or not.

I like simple data

I prefer simple raw data

| k=2				
	simple	raw	like	I
data	2	1	1	0

n

In the example above, you can see how we are keeping track of the number of times words occur together within a certain distance k. At the end, you can represent the word data, as a vector $v=[2,1,1,0]$.

Word by Document Design

You can now apply the same concept and map words to documents. The rows could correspond to words and the columns to documents. The numbers in the matrix correspond to the number of times each word showed up in the document.

Corpus

	Entertainment	Economy	Machine Learning
data	500	6620	9320
film	7000	4000	1000

You can represent the entertainment category, as a vector $v=[500,7000]$. You can then also compare categories as follows by doing a simple plot.
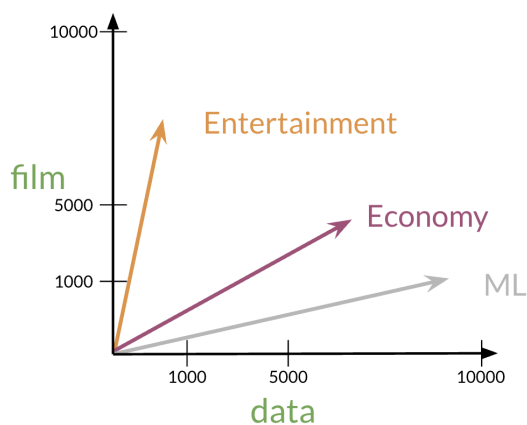
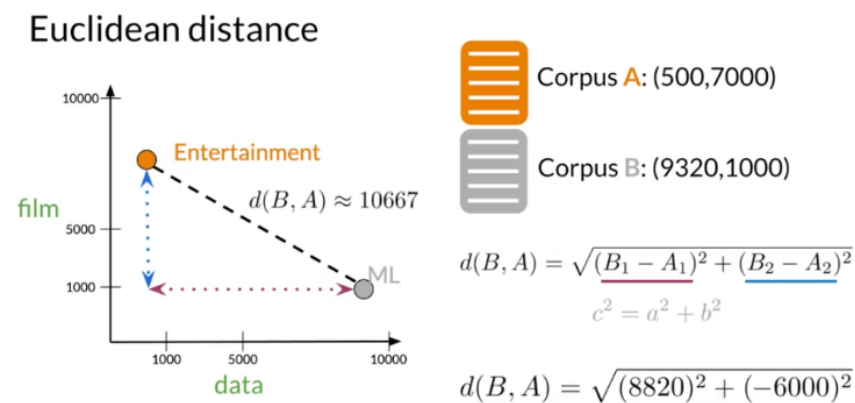|	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Measures of "similarity:"
Angle
Distance

Later this week, you will see how you can use the angle between two vectors to measure similarity.

Euclidian Distance

Let us assume that you want to compute the distance between two points: A, B. To do so, you can use the euclidean distance defined as

$$d(B, A) = \sqrt{((B_1 - A_1)^2 + (B_2 - A_2)^2)}$$

Euclidean distance

Corpus A: (500,7000)

Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

$$c^2 = a^2 + b^2$$

$$d(B, A) = \sqrt{(8820)^2 + (-6000)^2}$$

You can generalize finding the distance between the two points (A, B) to the distance between an n dimensional vector as follows:

$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^{n}(v_i - w_i)^2}$$

Here is an example where I calculate the distance between 2 vectors $(n = 3)$.

	data	boba	ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$= \sqrt{(1-0)^2 + (6-4)^2 + (8-6)^2}$$

$$= \sqrt{1 + 4 + 4} = \sqrt{9} = 3$$

Cosine Similarity: Intuition

One of the issues with euclidean distance is that it is not always accurate and sometimes we are not looking for that type of similarity metric. For example, when comparing large documents to smaller ones with euclidean distance one could get an inaccurate result. Look at the diagram below:

Euclidean distance: $d_2 < d_1$

Angles comparison: $\beta > \alpha$

The cosine of the angle between the vectors
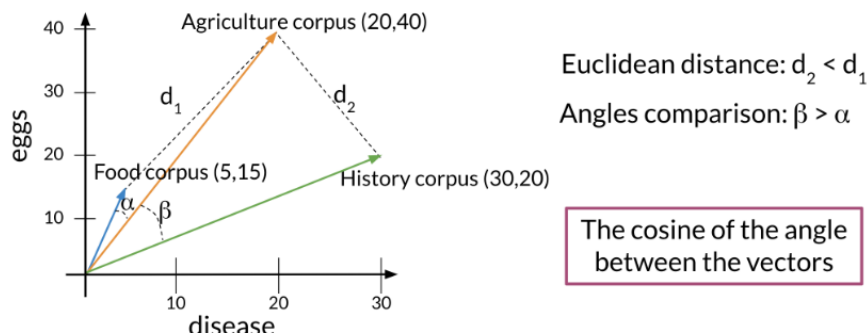
Normally the **food** corpus and the **agriculture** corpus are more similar because they have the same proportion of words. However the food corpus is much smaller than the agriculture corpus. To further clarify, although the history corpus and the agriculture corpus are different, they have a smaller euclidean distance. Hence $d_2 < d_1$.

To solve this problem, we look at the cosine between the vectors. This allows us to compare B and α.
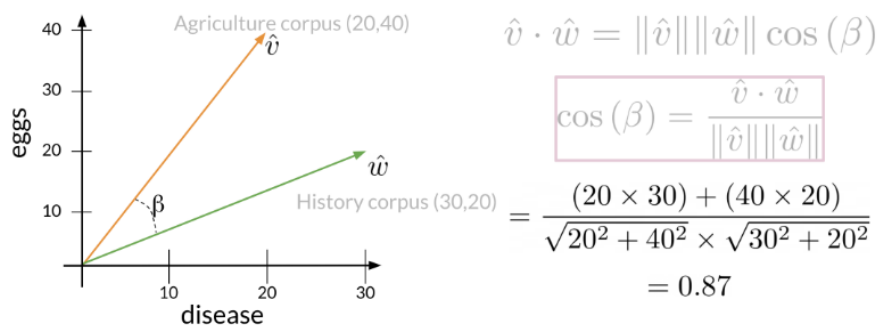
Cosine Similarity

Before getting into the cosine similarity function remember that the **norm** of a vector is defined as:

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} |v_i|^2}$$
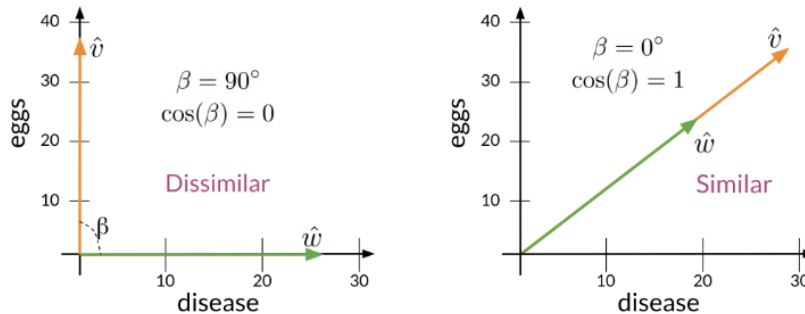
The **dot product** is then defined as:

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i \cdot w_i$$

$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

$$= \frac{(20 \times 30) + (40 \times 20)}{\sqrt{20^2 + 40^2} \times \sqrt{30^2 + 20^2}}$$

$$= 0.87$$

The following cosine similarity equation makes sense:

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

If $\hat{v}$ and $\hat{w}$ are the same then you get the numerator to be equal to the denominator. Hence $\beta = 0$. On the other hand, the dot product of two orthogonal (perpendicular) vectors is 0. That takes place when $\beta = 90$.
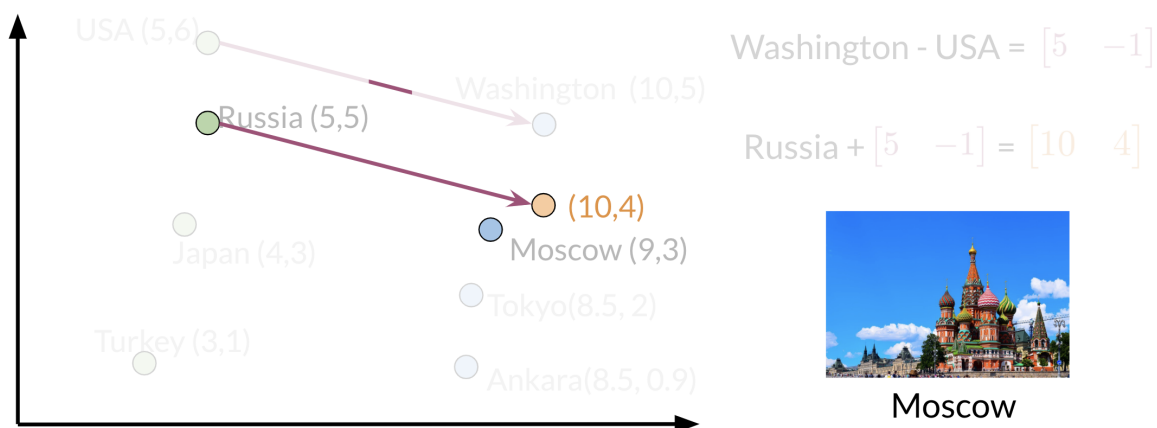

Manipulating Words in Vector Spaces

You can use word vectors to actually extract patterns and identify certain structures in your text. For example:

USA → Washington DC
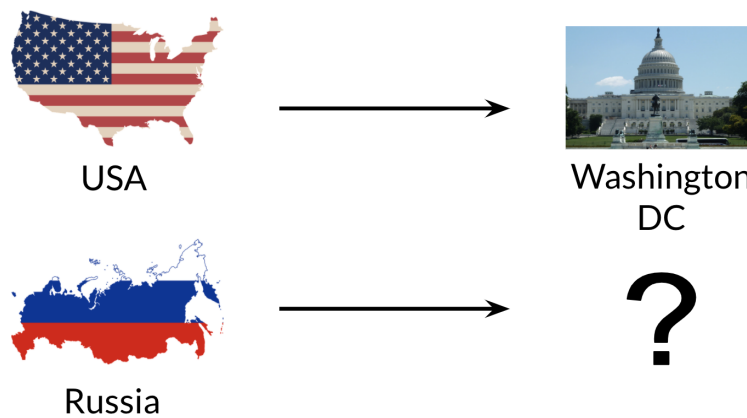
Russia → ?

You can use the word vector for Russia, USA, and DC to actually compute a **vector** that would be very similar to that of Moscow. You can then use cosine similarity of the **vector** with all the other word vectors you have and you can see that the vector of Moscow is the closest. Isn't that cool?

Washington - USA = $\begin{bmatrix} 5 & -1 \end{bmatrix}$

Russia + $\begin{bmatrix} 5 & -1 \end{bmatrix}$ = $\begin{bmatrix} 10 & 4 \end{bmatrix}$

Moscow

Note that the distance (and direction) between a country and its capital is relatively the same. Hence manipulating word vectors allows you identify patterns in the text.

Visualization and PCA

Principal component analysis is an unsupervised learning algorithm which can be used to reduce the dimension of your data. As a result, it allows you to visualize your data. It tries to combine variances across features. Here is a concrete example of PCA:

Visualization of word vectors

$d > 2$

oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

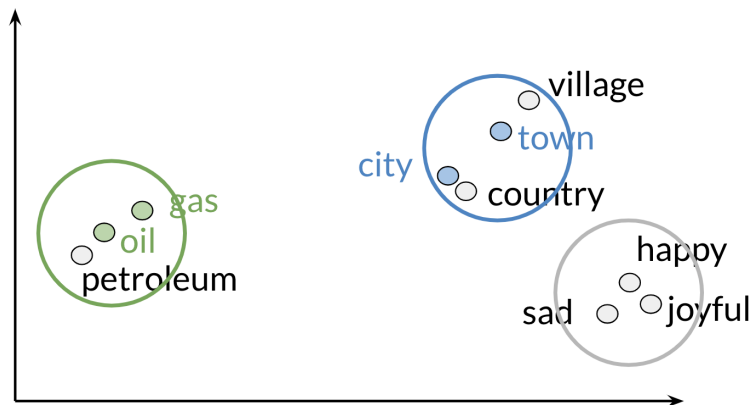How can you visualize if your representation captures these relationships?

oil & gas

town & city

Note that when doing PCA on this data, you will see that oil & gas are close to one another and town & city are also close to one another. To plot the data you can use PCA to go from $d>2$ dimensions to $d=2$.

Those are the results of plotting a couple of vectors in two dimensions. Note that words with similar part of speech (POS) tags are next to one another. This is because many of the training algorithms learn words by identifying the neighboring words. Thus, words with similar POS tags tend to be found in similar locations. An interesting insight is that synonyms and antonyms tend to be found next to each other in the plot. Why is that the case?

PCA algorithm

PCA is commonly used to reduce the dimension of your data. Intuitively the model collapses the data across principal components. You can think of the first principal component (in a 2D dataset) as the line where there is the most amount of variance. You can then collapse the data points on that line. Hence you went from 2D to 1D. You can generalize this intuition to several dimensions.

Principal Component Analysis
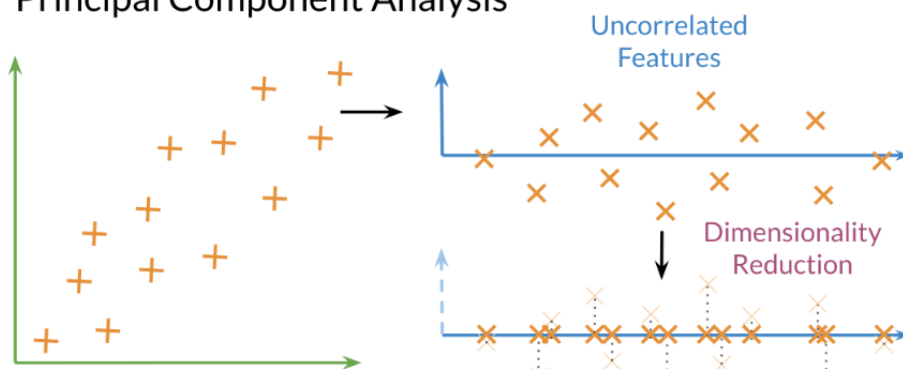
Eigenvector: the resulting vectors, also known as the uncorrelated features of your data

Eigenvalue: the amount of information retained by each new feature. You can think of it as the variance in the eigenvector.

Also each **eigenvalue** has a corresponding eigenvector. The eigenvalue tells you how much variance there is in the eigenvector. Here are the steps required to compute PCA:
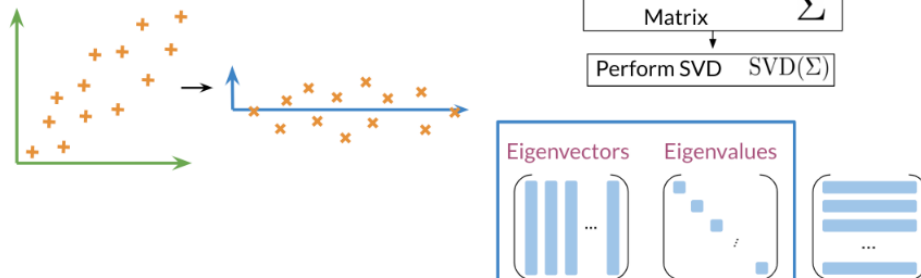
Steps to Compute PCA:

- Mean normalize your data
- Compute the covariance matrix
- Compute SVD on your covariance matrix. This returns $[USV] = svd(\Sigma)$. The three matrices U, S, V are drawn above. U is labelled with eigenvectors, and S is labelled with eigenvalues.
- You can then use the first n columns of vector U, to get your new data by multiplying $XU[:, 0 : n]$.