Vector Space Models

You shall know a word by the company it keeps.

Vector space models

Why learn vector space models ?
So suppose you have two questions.
The first one is, where are you heading?
And the second one is, where are you from?
These sentences have identical words, except for the last ones. However, they both have a different meaning. On the other hand, say you have two more questions whose words are completely different but both sentences mean the same thing.

Vector space models will help you identify whether the first pair of questions or the second pair are similar in meaning even if they do not share the same words. They can be used to identify similarity for a **question answering, paraphrasing, and summarization**.

The second half of the sentence is dependent on the first half. With vector space models, you will be able to capture this and many other types **of relationships among different sets of words**. Vector space models are used in **information extraction to answer questions in the style of who, what, where, how, and etc., in machine translation and in chatbots programming**.

When using vector space models, the way that representations are made is by identifying the context around each word in the text, and this captures the relative meaning. vector space models allow you to represent words and documents as vectors.

Word by Word and Word by Doc

The co-occurrence of two different words is the number of times that they appear in your corpus together within a certain word distance k

Word by Word Design

Number of times they *occur together within a certain distance k*

I like simple data

I prefer simple raw data

k=2

	simple	raw	like	I
data	2	1	1	0

Word by Document Design

Number of times a word *occurs within a certain category*

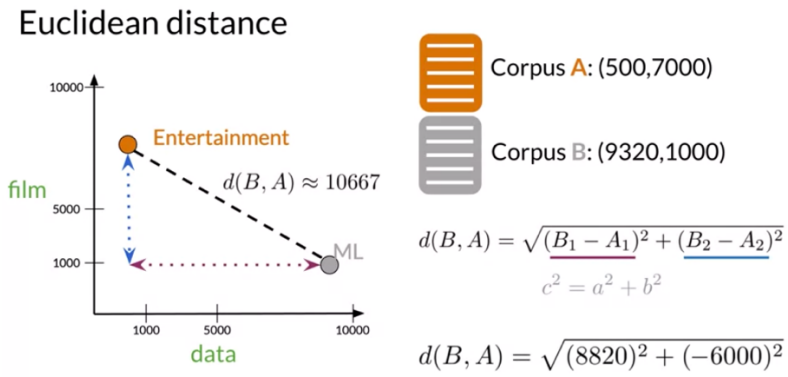
	Entertainment	Economy	Machine Learning
data	500	6620	9320
film	7000	4000	1000

Similarity using cosine distance or Euclidean distance.

Euclidean distance

Euclidean distance is basically the length of the straight line that's connects two vectors.

And that to get the Euclidean distance, you have to calculate the norm of the difference between the vectors that you are comparing. By using this metric, you can get a sense of how similar two documents or words are



Euclidean distance for n-dimensional vectors

		$\vec{w}$	$\vec{v}$
	data	boba	ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$= \sqrt{(1-0)^2 + (6-4)^2 + (8-6)^2}$$
$$= \sqrt{1+4+4} = \sqrt{9} = 3$$

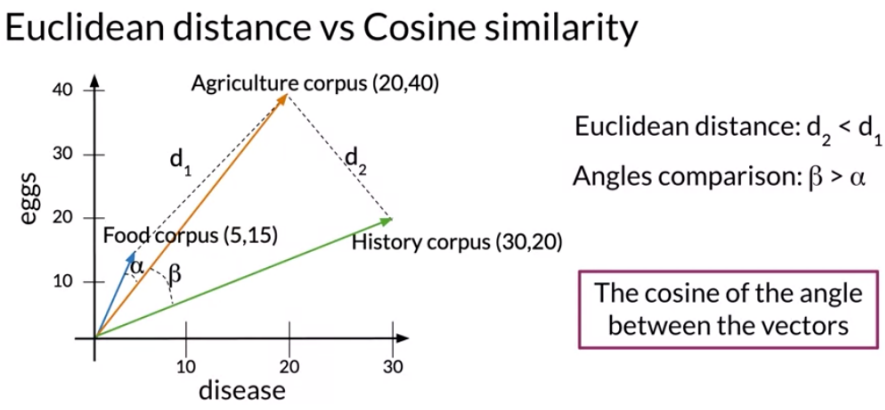
$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2} \longrightarrow \text{Norm of } (\vec{v} - \vec{w})$$

Euclidean distance in Python

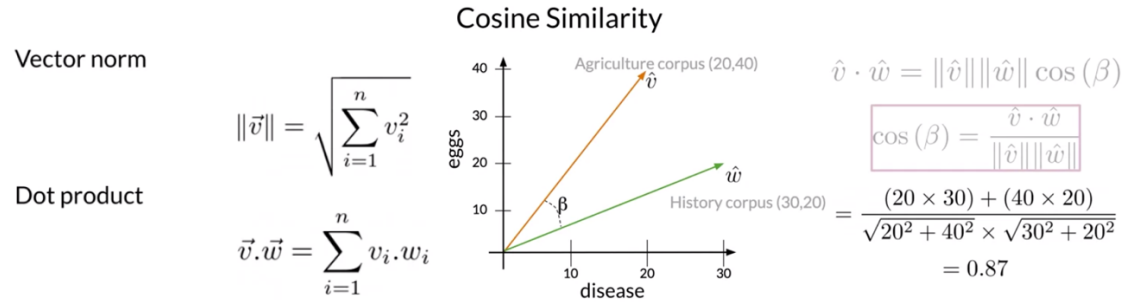
```
# Create numpy vectors v and w
v = np.array([1, 6, 8])
w = np.array([0, 4, 6])

# Calculate the Euclidean distance d
d = np.linalg.norm(v-w)
# Print the result
print("The Euclidean distance between v and w is: ", d)
```

```
The Euclidean distance between v and w is: 3
```

Cosine Similarity Intuition

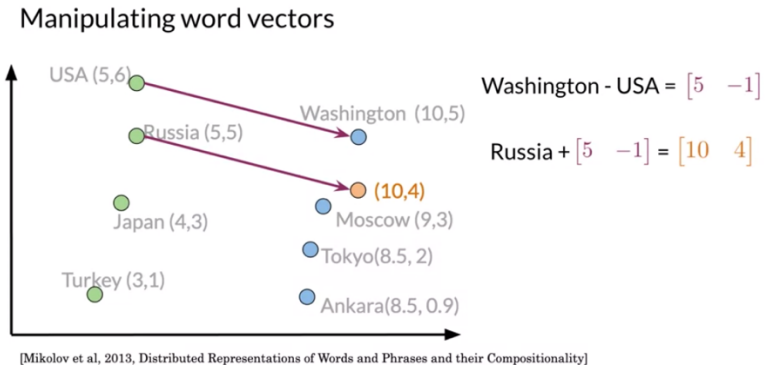


Cosine Similarity



Cos(0) = 1, cos(90) = 0
Np.dot(a,b) / (np.linalg.norm(a) * np.linalg.norm(b))

Manipulating Words in Vector Spaces

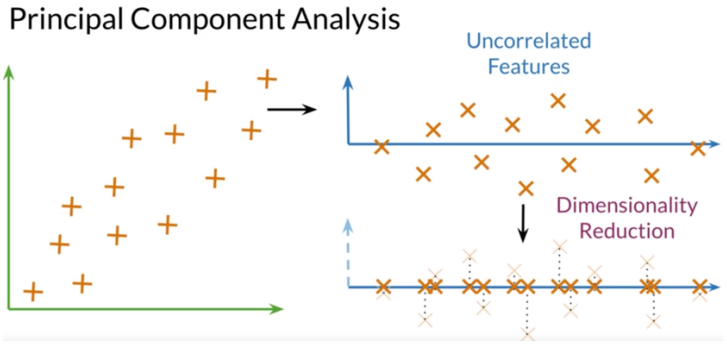


Find similar vectors using cosine distance or Euclidean distance. Catch here is we need a vector space where the representations capture the relative meaning of words.

Vectors of words that occur in similar places in the sentences will be encoded in a similar way.

Visualization and PCA

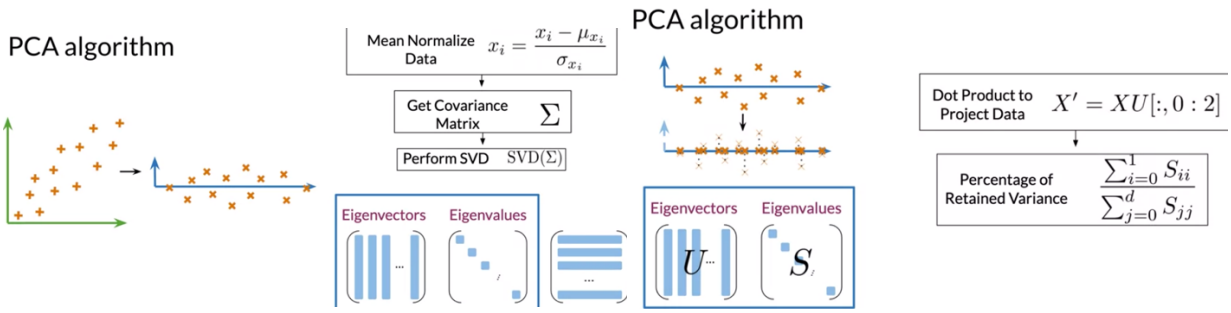
Visualization to see word relationships in vector space.
Original space → Uncorrelated Features → Dimension reduction



PCA algorithm

how to get uncorrelated features for your data
how to reduce the dimensions of your word representations while trying to keep as much information as possible from your original embedding.

the eigenvectors of the covariance matrix from your data, they give directions of uncorrelated features.
And the eigenvalues are the variance of your data sets in each of those new features.
The first step is to get a set of uncorrelated features.
For this step, you will mean normalize your data, then get the covariance matrix, and finally, perform a singular value decomposition to get a set of three matrices.



eigenvectors from the covariance matrix of your normalized data give the directions of uncorrelated features.
The eigenvalues associated with those eigenvectors tell you the variance of your data on those features.
The dot products between your word embeddings and the matrix of eigenvectors will project your data onto a new vector space of the dimension that you choose.

```
def compute_pca(X, n_components=2):
    """
    Input:
        X: of dimension (m,n) where each row corresponds to a word vector
        n_components: Number of components you want to keep.
    Output:
        X_reduced: data transformed in 2 dims/columns + regenerated original
data
    """

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
    # mean center the data
    X_demeaned = X - np.mean(X, axis=0).reshape((1,-1))

    # calculate the covariance matrix
    covariance_matrix = np.cov(X, rowvar=False)

    # calculate eigenvectors & eigenvalues of the covariance matrix
    eigen_vals, eigen_vecs = np.linalg.eigh(covariance_matrix, UPLO='L')

    # sort eigenvalue in increasing order (get the indices from the sort)
    idx_sorted = np.argsort(eigen_vals)

    # reverse the order so that it's from highest to lowest.
    idx_sorted_decreasing = idx_sorted[::-1]

    # sort the eigen values by idx_sorted_decreasing
    eigen_vals_sorted = eigen_vals[idx_sorted_decreasing]
```

```
    # sort eigenvectors using the idx_sorted_decreasing indices
    eigen_vecs_sorted = eigen_vecs[:,idx_sorted_decreasing]

    # select the first n eigenvectors (n is desired dimension
    # of rescaled data array, or dims_rescaled_data)
    eigen_vecs_subset = eigen_vecs_sorted[:,0:n_components]

    # transform the data by multiplying the transpose of the eigenvectors
    # with the transpose of the de-meaned data
    # Then take the transpose of that product.
    X_reduced = np.dot(X_demeaned, eigen_vecs_subset)

    ### END CODE HERE ###

    return X_reduced
```

```
def cosine_similarity(A, B):
    '''
    Input:
        A: a numpy array which corresponds to a word vector
        B: A numpy array which corresponds to a word vector
    Output:
        cos: numerical number representing the cosine similarity between A
and B.
    '''

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    dot = np.dot(A,B)
    norma = np.linalg.norm(A)
    normb = np.linalg.norm(B)
    cos = dot / (norma * normb)

    ### END CODE HERE ###
    return cos
```
