

**Applied Natural Language Processing**  
**Prof. Ramaseshan Ramachandran**  
**Visiting Professor at Chennai Mathematical Institute**  
**Indian Institute Technology, Madras**

**Lecture - 12**  
**Document Similarity-demo, Inverted Index, Exercise**

(Refer Slide Time: 00:15)

**DOCUMENT-TERM MATRIX**

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12
t1	0.1	0.0	0.4	0.1	0.2	0.0	0.1	0.9	0.9	0.3	0.0	0.8
t2	0.1	0.0	0.4	0.1	0.2	0.0	0.1	0.9	0.9	0.3	0.0	0.8
t3	0.0	0.9	0.0	0.2	0.3	0.1	0.7	0.0	0.2	0.7	0.5	0.5
t4	0.0	0.9	0.3	0.9	0.5	0.1	0.9	0.3	0.8	0.4	0.1	0.4
t5	0.4	0.0	0.3	0.2	0.5	0.9	0.3	0.7	0.4	0.6	0.0	0.3
t6	0.6	0.0	0.4	0.7	0.3	0.3	0.9	0.1	0.9	0.0	0.0	0.3
t7	0.0	0.8	0.5	0.6	0.6	0.6	0.0	0.1	0.4	0.9	0.3	0.1
t8	0.4	0.0	0.6	0.5	0.5	0.1	0.7	0.1	0.5	0.3	0.8	0.1
t9	0.3	0.0	0.7	0.9	0.8	0.7	0.7	0.8	0.6	0.6	0.8	0.0
t10	0.0	0.5	0.5	0.0	0.2	0.0	0.0	0.1	0.3	0.4	0.5	0.3

The columns of the matrix represent the document as vectors. A document vector is represented by the terms present in the document

So, now take a bigger example where we have a, we have about 10 terms and we have about 12 documents. And then every element in this matrix represents our tf-idf. what we are going to do is, we going to be looking at this and then try to find out which document is closer to the other one.

So, this is a very important exercise in information retrieval at least for document sets which are not very huge. we can actually, use this if the document is very small for you. Again, if you look at this I keep repeating this, this one is your term vectors for document one; that means, the document one contains the words  $t_1$  to  $t_{10}$ , some others are missing, these terms are missing, we have 0 values here.

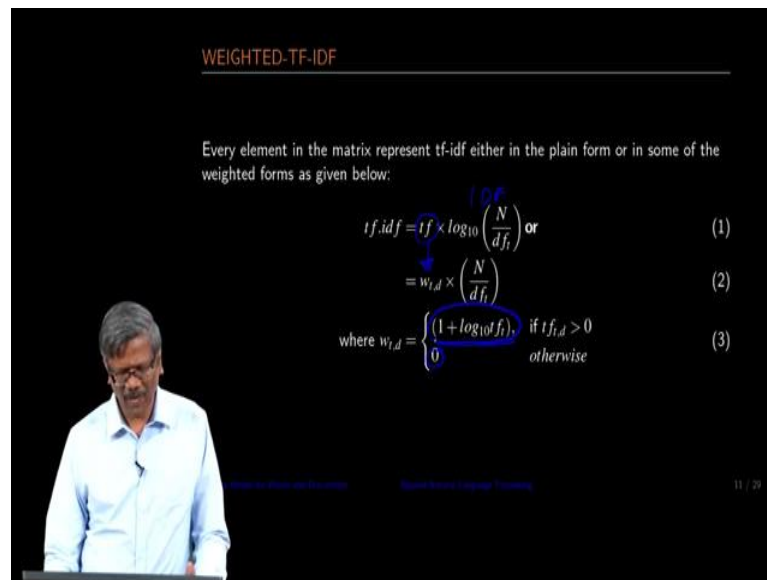
And then if you look at this one, term vector across the corpus. this is very important to understand this structure and most of the term will be representing our term document in this fashion.

(Refer Slide Time: 01:40)

WEIGHTED-TF-IDF

Every element in the matrix represent tf-idf either in the plain form or in some of the weighted forms as given below:

$$tf\text{-idf} = tf \times \log_{10} \left( \frac{N}{df_i} \right) \text{ or} \quad (1)$$
$$= w_{i,d} \times \left( \frac{N}{df_i} \right) \quad (2)$$

where  $w_{i,d} = \begin{cases} (1 + \log_{10}(f_i)), & \text{if } f_{i,d} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$ 

So, how do I represent them actually, I am not sure whether I mentioned this in the previous lecture, the tf-idf which I have spoken about could also be represented using other weighted to-IDF; one mechanism is to consider tf and then the IDF here ok.

And then this would be replaced, not exactly the same this would be replaced as this ok. If you replace the tf with respect to 1 plus log logarithm of term frequency and if term frequency is greater than 0 and use this value and if it is 0, it is going to be 0. this is again a very important representation of our terms in this tf rather a document term matrix.

So, in our case, we have just used random generation of these numbers and I represent them as tf-IDF, the plane of-IDF.

(Refer Slide Time: 03:01)

QUERY MODELING

Each query is modeled as a vector using the same attribute space of the documents.

$$q = [q_1, q_2, q_3, \dots, q_n] \quad (4)$$

The relevancy ranking of a document depends on the distance of the document with respect to the query. The proximity of the query with every document is computed using distance measures.

12 / 29

So, now what is required? we have a set of documents, we have collected the term frequencies with respect to weighted term frequency using tf-IDF or the log of those which I explained earlier. Now I want to find out, using a query how that particular query is very close to some other documents and then rank them accordingly. When I represent a query; the query is going to be again a composition of terms right. we can consider that it has a document very similar to this. I can write a query here and then my query can contain some values related to  $t_1$  and then some values related to  $t_3$  some value related to  $t_7$  and so on, ok.

So, this is how I construct this, and these terms if it is found we already have created a corpus, we already have found some frequency related value for these queries and so on this is that what I am replacing with. When somebody gives a query, find me a document with terms 1 3 and 7. I go to the dictionary and then find out what values of what are the values for these three terms and then replace them, and then I construct them as a separate vector ok, like this.

So, I am now constricting the query, which is again very close to, is a closed representation of the document; now it is possible for me to go and then find out, a how these documents are related to this query and then probably rank them according to the highest one which is closer to q would be the first and the lowest one would be the last in

that ranking order ok. We going to be using some proximate is called mechanism, to find out how a query is related to this document.

(Refer Slide Time: 05:16)

**DOCUMENT SIMILARITY**

Earlier, using the binary incidence matrix, a query returned a set of documents whether the query keywords were found in documents or absent. It did not give any ranking for the retrieved documents. A similarity measure is a real-valued function that quantifies the similarity between two objects [1]. Some of the methods are given below.

Euclidean Distance -  $d(\vec{d}_1, \vec{d}_2) = \sqrt{d_1^2 - d_2^2}$  (5)

Cosine Distance -  $\cos(\vec{d}_1, \vec{d}_2) = \frac{\vec{d}_1 \cdot \vec{d}_2}{\|\vec{d}_1\| \|\vec{d}_2\|} = \frac{\vec{d}_1 \cdot \vec{d}_2}{\|\vec{d}_1\| \|\vec{d}_2\|} = \frac{\vec{d}_1 \cap \vec{d}_2}{\sqrt{\|\vec{d}_1\| \times \|\vec{d}_2\|}}$  (6)

Cluster similarity -  $\mathcal{L}(\vec{d}_1, \vec{d}_2) = \frac{\vec{d}_1 \cdot \vec{d}_2}{\|\vec{d}_1\|_1}$  (7)

Jaccard Similarity -  $\mathcal{J}(\vec{d}_1, \vec{d}_2) = \frac{\vec{d}_1 \cap \vec{d}_2}{\vec{d}_1 \cup \vec{d}_2}$  (8)

13 / 20

It is usually done using certain formula let us look at some of them. I am sure, in the earlier case of binary incidence matrix, the query returned a set of documents for that particular set of keywords right, but it never gave you the rank. We definitely require the rank, because we want to look at the document with the highest relevance first rather than something with the lowest relevance. In the case of the binary incident matrix it is not possible for you to get that rank in that order. you have to assume, that all documents have the same ranks, but in reality we would like to have a rank for all, of those documents.

So, we are going to be using some of those measures like Euclidean distance, Cosine distance, Cluster similarity, and Jaccard similarity, I would be using cosine similarity for the sake of showing some demo and so on. why is it not a good idea to use a  $d_1$  and  $d_2$  ? look at this ok. if you compute the distance right, a sum would be long like this in the vector space. Let us assume that, this is your query and then some vector would be very long, some would be very short like this. it is not going to really give you good measure if the length is too long. it is not advisable to really go for distance measure for finding the similarity of documents.

(Refer Slide Time: 07:08)

**WHICH MEASURE?**

Euclidean measure does not work well for unequal sized vectors as the vectors are not normalized. We often use a normalized correlation coefficient, Cosine distance for the

Cosine Distance =  $1 - \cos(\vec{d}_1, \vec{d}_2)$

Similarity measure.

The slide features a 3D coordinate system with axes  $d_1$ ,  $d_2$ , and  $d_3$ . A query vector  $q_1$  is shown. Handwritten notes include:

- A table for 'plane':

rank	1
$d_1$	2
$d_2$	3
- A table for 'car':

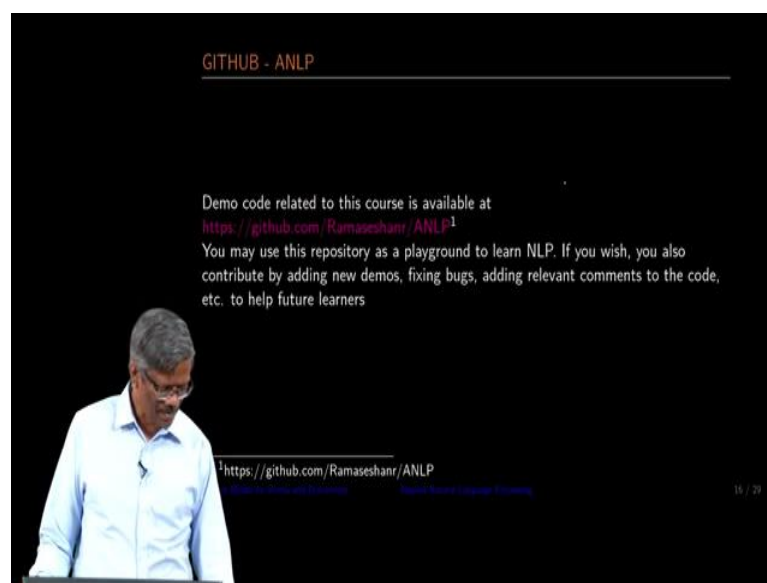
rank	1
$d_1$	2
$d_2$	3
- Handwritten cosine calculations:
$$\cos \theta_1 = \frac{q_1 \cdot d_1}{|q_1| |d_1|} = \frac{1 \cdot 1}{\sqrt{1} \cdot 1} = 1$$
$$\cos \theta_2 = \frac{q_1 \cdot d_2}{|q_1| |d_2|} = \frac{1 \cdot 2}{\sqrt{1} \cdot 2} = 1$$
$$\cos \theta_3 = \frac{q_1 \cdot d_3}{|q_1| |d_3|} = \frac{1 \cdot 3}{\sqrt{1} \cdot 3} = 1$$

So, which measure, as I mentioned earlier that I am going to be using Cosine distance as a similarity measure because it is a normalized correlation coefficient. Why is it normalized? Because you are actually dividing the vector by the length of the vector; you get a unit vector there. It is normalized for all the vectors that you are considering; for example,  $d_1$  is normalized to the unit vector and then this also is normalized.

So, for the sake of representation I am going to be taking two words car and plane, right. We have a query here, based on the term frequency and the length. We have got a query in this direction, of length here. And then we have other documents representing car and plane in various directions ok. We have  $d_3$  here,  $d_1$  here and  $d_2$  here;  $d_2$  is very close to the plane because it may contain a lot of terms related to the plane than a car. It is closely aligned to the axis plane and  $d_3$  is very closely aligned to the car for the same reason.

So, are the values, I am going to be finding these values. the rank is accorded, based on how close this particular is, the document is, to  $q_1$ . in this case we know visually we can find out that  $d_3$  is close is; that means, we can rank it accordingly,  $d_3$  if you look at the rank,  $d_3$  will come first and then  $d_1$  and then  $d_2$  right. This is how you find the similarity measure between documents and the query, that you have just posted, and then using this you can now rank the documents accordingly and then say ok.  $D_3$  is very close let us take a look at the document  $d_3$  first before, looking at the other ones if I do not get enough information in  $d_3$  ok.

(Refer Slide Time: 10:50)

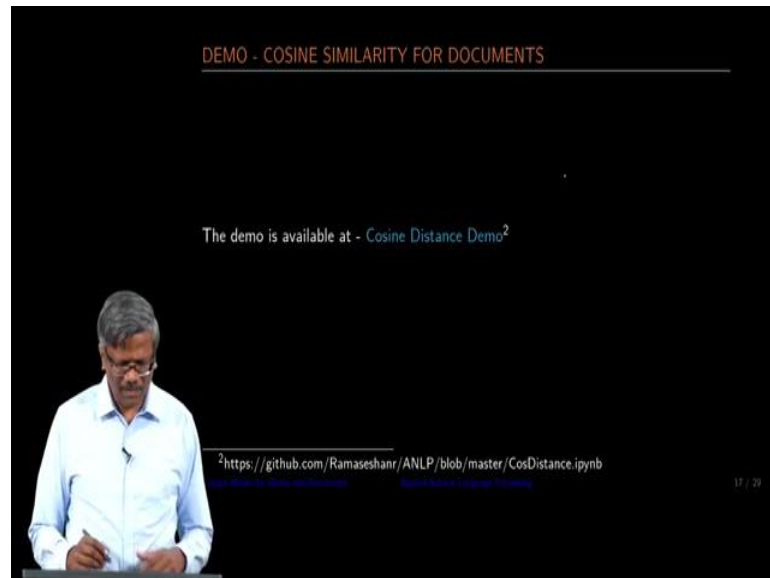


As I mentioned earlier Cosine distance is preferred and it is very easy to compute the score, using them ok. Let us look at a demo before, the demo I just want to mention the few things. What I am doing right now, is I am posting all the demos in one space in GitHub under Ramaseshanr ANLP.

Whatever I am showing you like a demo those, programs are available in this library this is open-sourced, you can take a look at it, you can clone it, you can make changes to that. And, if you suggest some new ideas into that, or you are want to add more programs to this library your more than welcome to do, if there are any errors found you can also mention or send a note saying that there are some errors and you would like to correct them.

I can give you access. that you can do all the, changes in the code and then again submit the code back to the open-source. This is something that I would like you to go and then look at it and then add value to that if you want to or you can use this as a playground. that we can play with the demos and understand the call, underlying concepts in all these demos.

(Refer Slide Time: 12:29)



So now, let us go to the demo code.

(Refer Slide Time: 12:38)

```
# MIT License
# Copyright (c) 2019.
# Project Name: ANLP
# File name: cos4.py
# Last modification date: 2/28/19 2:31 PM
# Author: ramaseshan
# Email:ramaseshan@yahoo.com
# /CosineDistance.py
#

from numpy import *
from numpy import dot
from numpy.linalg import norm
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

def cosine_distance(a,b):
    return dot(a,b)/(norm(a)*norm(b))

def find_end_points(point, angle, length):
    """
    #Source - https://stackoverflow.com/questions/2841704/plotting-a-line-from-a-coordinate-with-end-angle
    #
    point - Tuple (x, y)
    angle - Angle you want your end point at in degrees.
    length - Length of the line you want to plot.

    Will plot the line on a 10 x 10 plot.
    """
    # unpack the first point
    x, y = point
```

Cosine similarity is used instead of Cosine distance.  
The Github contains the modified version of this program

So, instead of clicking and then going to the GitHub and then finally, going to the collab dot research dot com, I have taken you directly to space. when you go to that GitHub space, there will be a button that will allow you to come to this collab dot research dot google dot com ok. it will open up that particular file. that you can run the application, directly from this. I am going to run this, first ok.

(Refer Slide Time: 13:27)

The screenshot shows a Google Colab notebook titled 'CosDistance.ipynb'. The code defines a function to calculate cosine similarity between two documents based on their word counts. It then applies this function to a set of documents (D0-D10) and prints the resulting similarity matrix and document ranks for a specific query (D0).

```

ax.text(x[1],y[1],D0 + str(ref_doc) + "-" + D0 + str(i) + "-" + str(df.iloc[ref_doc][i]) + "\u0000")
max_val,patch=plt.imshow(D0+D1:D0, .25+1/12.0, .25+1/12.0, theta=0, theta2=0, cmap=plt.cm.hot, label = str(df.iloc[0][i]) + "\u0000")
fig.savefig('CosineDistance.pdf')
fig.show()
query = D0 + str(ref_doc)
rank_order = df.sort_values(query)
print("\nDocument Rank for the query ", query)
print(rank_order[query])

```

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
D0	0.0	4.0	90.0	45.7	50.6	64.8	41.9	64.6	74.6	72.1	56.9
D1	4.0	0.0	90.0	46.9	52.6	66.0	42.3	67.3	75.3	73.7	59.0
D2	90.0	90.0	0.0	56.5	66.1	71.8	59.5	81.4	57.6	41.7	61.7
D3	45.7	46.9	56.5	0.0	39.5	46.6	28.5	50.5	53.9	45.2	49.7
D4	50.6	52.6	66.1	39.5	0.0	29.5	48.9	53.8	60.8	31.9	36.1
D5	64.8	66.0	71.8	46.6	29.5	0.0	58.1	54.3	66.9	40.5	61.2
D6	41.9	42.3	59.5	28.5	48.9	58.1	0.0	63.0	56.4	53.5	59.5
D7	64.6	67.3	81.4	58.5	53.8	54.3	63.0	0.0	54.3	51.1	69.1
D8	74.6	75.3	57.6	53.9	60.8	66.9	56.4	54.3	0.0	50.3	69.2
D9	72.1	73.7	41.7	45.2	31.9	40.5	53.5	51.1	50.3	0.0	44.5
D10	56.9	59.0	61.7	49.7	36.1	61.2	50.5	69.1	69.2	44.5	0.0

Document Rank for the query D0

```

D0 0.0
D1 4.0
D6 41.9
D3 45.7
D4 50.6
D10 56.9

```

So, what I have, I am just, I will first show the output then I will probably take you to the code and explain what is going on there ok.

(Refer Slide Time: 13:47)

The screenshot shows a Google Colab notebook with code that calculates a cosine similarity matrix for a set of documents. It uses NumPy for matrix operations and Pandas for data handling. The output is a DataFrame where each row and column represents a document (D0-D10) and the cells contain the cosine similarity values between them.

```

def cosine_similarity(x,y):
    max = plt.subplot(111)
    return (x,y)
    max.plot(x, y)

@returns fig
column_header = ['D0','D1','D2','D3','D4','D5','D6','D7','D8','D9','D10']
row_header = ['D0','D1','D2','D3','D4','D5','D6','D7','D8','D9','D10']
doc_term = array([
    [0.1, 0.1, 0.0, 0.1, 0.2, 0.0, 0.1, 0.9, 0.9, 0.3, 0.0, 0.0],
    [0.1, 0.1, 0.0, 0.1, 0.0, 0.0, 0.1, 0.9, 0.9, 0.3, 0.0, 0.0],
    [0.0, 0.0, 0.9, 0.0, 0.3, 0.1, 0.7, 0.0, 0.1, 0.7, 0.5, 0.5],
    [0.0, 0.0, 0.9, 0.1, 0.0, 0.1, 0.9, 0.3, 0.0, 0.4, 0.1, 0.4],
    [0.0, 0.0, 0.0, 0.0, 0.5, 0.9, 0.3, 0.7, 0.4, 0.6, 0.0, 0.3],
    [0.5, 0.6, 0.0, 0.7, 0.3, 0.3, 0.9, 0.1, 0.0, 0.0, 0.0, 0.3],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.2, 0.4, 0.0, 0.3, 0.1],
    [0.35, 0.4, 0.0, 0.5, 0.5, 0.1, 0.7, 0.1, 0.5, 0.3, 0.0, 0.1],
    [0.3, 0.3, 0.0, 0.2, 0.0, 0.7, 0.7, 0.0, 0.0, 0.6, 0.0, 0.0],
    [0.0, 0.0, 0.5, 0.0, 0.2, 0.0, 0.0, 0.1, 0.0, 0.4, 0.5, 0.3]
])

cos_list = []
pd_cols = []

for i in range(0,11):
    for j in range(0,11):
        cos_value = cosine_similarity(transpose(doc_term[i, :]), doc_term[:, j])
        cos_list.append(around(math.degrees(math.acos(min(max(cos_value,-1.0),1.0))), decimals=4) ))
pd_cols.append(cos_list)
cos_list = []
df = pd.DataFrame(pd_cols, columns=column_header, index=row_header)
print(df)

fig = plt.figure()

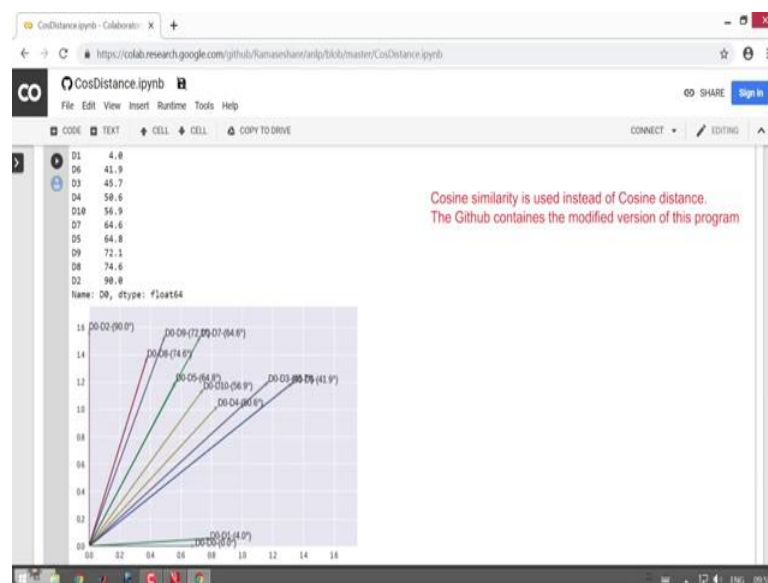
```



So, in this case what I have done is, I have used the same matrix that I have shown you ok, the document term matrix, this is what I am using it as an input. And then when I run this, what I get is this, here you have a matrix containing all the documents along the x and y-axis; I have taken every document in that space and then try to compute the, distance from that document to another document ok.

So, I have not used any query at this point in time. we will come to that little later. if you consider document one, it goes and then finds out the angle between  $D_0$  and  $D_1$   $D_2$  and  $D_{10}$ . And then it will considered  $D_0$  as the base document and then try to compute all the angles between  $D_0$  and  $D_1$ ,  $D_0$  and  $D_2$ ,  $D_0$  and  $D_6$  and so , on. in this fashion if you look at the top site, you will have all the angles computed for all the documents ok. I am going to be now considering only one document as the reference document.

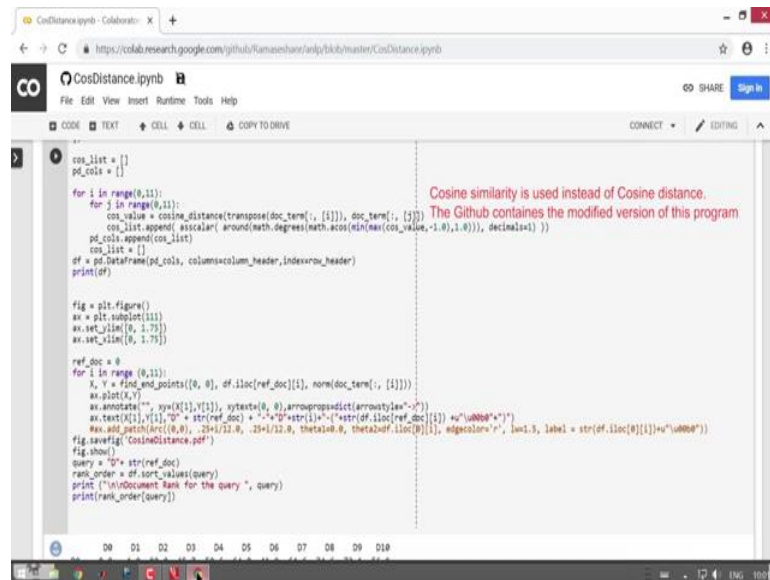
(Refer Slide Time: 15:06)



And then try to see how closely those documents are related to that particular reference document. in this case I am considering  $D_0$  as my reference document here. And then I computed the ranks based on the distance,  $D_1$  is very close,  $D_6$  is the next one and then  $D_2$  is orthogonal to  $D_0$ ; that means, none of the terms are close to  $D_0$  that is why it is not showing any relevant there. the most relevant documents for me is  $D_1$  here. if we go and then look at pictorially, you will see that ok.  $D_1$  against  $D_0$  because it is the same document dot product of the same will result in one of the angles is 0 there.

And then the next one which is closer to  $D_0$  is  $D_1$ , it is separated by an angle 4 degrees, and then you have all the others and then if you look at  $D_0$  and  $D_2$  it is separated by 90 degrees. Let me take you to this space, where the code is available as an I showed earlier right;  $D_0$  to  $D_{10}$  matrix here is computed by this.

(Refer Slide Time: 16:25)



```
cos_list = []
pd_cols = []

for i in range(0,11):
    for j in range(0,11):
        cos_value = cosine_distance(transpose(doc_term[:, i]), doc_term[:, j])
        cos_list.append( str(float( round(math.degrees(math.acos(cos_value/(cos_value*-1.0)),1.0)), decimal=1) ) )
    pd_cols.append(cos_list)
cos_list = []

df = pd.DataFrame(pd_cols, columns=column_header, index=row_header)
print(df)

fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0, 1.75)
ax.set_ylim(0, 1.75)

ref_doc = 0
for i in range(0,11):
    X, Y = find_end_points((0, 0), df.iloc[ref_doc][i], norm(doc_term[:, i]))
    ax.plot(X,Y)
    ax.annotate("", xy=(X[1],Y[1]), xytext=(0, 0), arrowprops=dict(arrowstyle=">"))
    ax.text(X[1],Y[1], "D"+str(i)+" "+str(ref_doc)+" "+str(df.iloc[ref_doc][i])+" "+u"0000")
    #ax.add_patch(Arc((0,0), 23+1/12.0, 23+1/12.0, theta1=0.0, theta2=0.0, df.iloc[0][i], edgecolor="r", lw=1.5, label = str(df.iloc[0][i])+"u0000"))
fig.show()
fig.savefig("CosineDistance.pdf")
query = "0"+str(ref_doc)
rank_order = df.sort_values(query)
print ("rank_order for the query ", query)
print(rank_order[query])
```

Cosine similarity is used instead of Cosine distance. The Github contains the modified version of this program

So, first I compute all the angles, and then for the plotting sake and ranking sake I am just using reference, document ok. I am just using a reference document here as 0 and then trying to find out what is the distance measure or the similarity measure in the case. Here, there are some elements that you will find for plotting. for plotting what you require is, you need to find the end distance, so that we can draw that vector.

So, initial value 0 and then the end value is calculated by the vector length and then we plot that x and y, and then I have made the annotation. that it is easy for you to understand that and then at the end you have the rank-ordered in the sorted fashion. once the values are computed for the reference document, you sort the, document with respect to the reference document. A use case for printing the rank is, the ascending order, alright. we have seen how these documents could be compared when the tf-IDF values are available.

So, now, let us try to construct a query and then see how the similarities are measured and then how the ranks are computed ok. in this case what I am going to do is, earlier case you saw the document 0 as the reference and then we computed the rank. And now

if we are going to be considering, document 2 as our query, as I mentioned earlier right. the query can be constructed with respect to the term, that is present in the queries and it can be construed as a document as well.

So, now, let us consider  $D_2$  as our query, I am going to be changing my reference, we already have completed all the angles, I am only going to change the reference document as my query and make it as 2. And I am going to be running this program, to find out how this distance varies with respect to query  $D_2$ , that I have chosen ok. if you look at the query,  $D_2$  query on itself is going to give you 0 as I mentioned earlier, because it is the dot product, of the same will give you the value of 0, so you have it there.

And, then you have  $D_2$  at an angle  $D_2$  to  $D_9$  is 41, and then  $D_1$  and  $D_0$  are orthogonal to document 2 ok. if you look at the rank here, at the query is closer to  $D_9$  and  $D_8$  and  $D_6$  you know if you look at the other angles they are 61 66 71 and so on; that means, the document  $D_9$  and  $D_8$  are closer to  $D_2$  than any other document this place. in this way you can actually play with these numbers, like the reference document or you can construct a new query and then add as part of the matrix that you have here, in this and then try this.

So, make sure that you use the right range I have hot coded the ranges in this program. you may want to avoid that hot coding and want to give some variables there, so that you can find the actual sizes of the matrixes and then do the computation accordingly ok. For the explanation's sake, I just put together some simple programs for you. you can go ahead and then change this in the, Github that I have provided for you. this is the example of how you can construct the similarity of document ok. Let us go back to the presentation again.

(Refer Slide Time: 21:18)

**INVERTED INDEX**

- ▶ D1 - A ball is thrown from a bridge horizontally at a speed of 8m/s. Just before it hits the ground it is moving 50m/s vertically. How long was the ball in the air?
- ▶ D3- A ball is kicked at an angle of  $35^\circ$  with the ground. (a) What should be the initial velocity of the ball so that it hits a target that is 30 meters away at a height of 1.8 meters?
- ▶ ...

Term	TF	Postings
ball	4	2,1,8,9
bridge	2	1
air	1	1
...	...	...

18 / 29

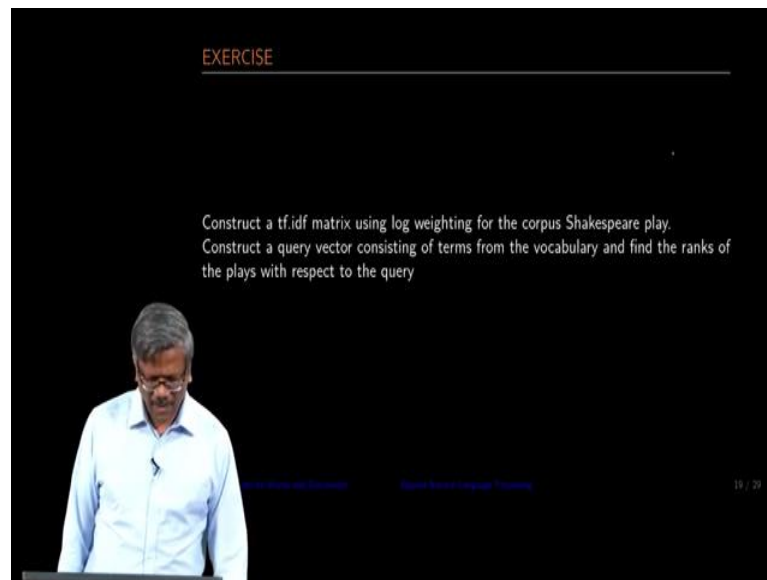
So, we have just finished our cosine distance demo. And then one more thing we also want to look at, when we try doing the similarity for huge document asset. we can create one inverted index for all, the document that we have; for example, for the given two documents that I have, you can construct a term frequency and a posting.

For every term you have a column ok. this is a column for you and then the term frequencies for the, word ball across the document space, in this case this is going to be 4. And then the postings; the postings are nothing, but the documents where this particular word is present. if I have the word ball present in document 1 2 8 and 9, I just mentioned that.

So, if you want to do the comparison or the similarity of the documents with respect to the query where you have the word ball is mention, you do not have to go through the entire document space, you can look at these postings and then say document 2 has this. let us go on and then look at document 2 and then find out how close the document with respect to this a term that I have any query. in the same way, you can go and look at it.

So, this way you can reduce the computational time, you know instead of going through the entire document set, you can only go to a small subset where that particular term is present in those documents ok, alright ok.

(Refer Slide Time: 23:22)



So, this is an exercise for you, the exercise is you have to construct a tf-idf matrix, using the log weighting that I had shown earlier for the corpus Shakespeare's play. Then, you need to construct a query vector containing some terms, see, for example, you pick a few terms, at random in Shakespeare's play and then construct that as a vector. And then use the Cosine similarity, that I had shown in the demo and find out how these plays are staged with respect to the rank.

So, this is one exercise for you and the program is already available. What you have to change is, you only have to change the document term matrix that I have given with Shakespeare's play. This would be very useful, if you implement this small exercise.