In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
from nltk.tokenize import sent_tokenize, word_tokenize
from gensim.summarization.bm25 import get_bm25_weights
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
import sklearn
import math
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min,jaccard_similarity_score
from sklearn.metrics.pairwise import cosine_similarity as cosu, euclidean_distances
```

In [2]:

```
f=open("a.txt", "r")
```

In [3]:

```
data=(f.read())
print(data)
```

```
A decision tree is a decision support tool that uses a tree-like graph
or model of decisions and their possible consequences, including chanc
e event outcomes, resource costs, and utility. It is one way to displa
y an algorithm that only contains conditional control statements.
A decision tree is a flowchart-like structure in which each internal n
ode represents a “test” on an attribute (e.g. whether a coin flip come
s up heads or tails), each branch represents the outcome of the test,
and each leaf node represents a class label (decision taken after comp
uting all attributes). The paths from root to leaf represent classific
ation rules.
Tree based learning algorithms are considered to be one of the best an
d mostly used supervised learning methods. Tree based methods empower
predictive models with high accuracy, stability and ease of interpreta
tion. Unlike linear models, they map non-linear relationships quite we
ll. They are adaptable at solving any kind of problem at hand (classif
ication or regression). Decision Tree algorithms are referred to as CA
RT (Classification and Regression Trees).
“The possible solutions to a given problem emerge as the leaves of a t
ree, each node representing a point of deliberation and decision.”
- Niklaus Wirth (1934 – ), Programming language designer
Methods like decision trees, random forest, gradient boosting are bein
g popularly used in all kinds of data science problems. Common terms u
sed with Decision trees:
Root Node: It represents entire population or sample and this further
gets divided into two or more homogeneous sets.
Splitting: It is a process of dividing a node into two or more sub-nod
es.
Decision Node: When a sub-node splits into further sub-nodes, then it
is called decision node.
Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal nod
e.
Pruning: When we remove sub-nodes of a decision node, this process is
called pruning. You can say opposite process of splitting.
Branch / Sub-Tree: A sub section of entire tree is called branch or su
b-tree.
Parent and Child Node: A node, which is divided into sub-nodes is call
ed parent node of sub-nodes whereas sub-nodes are the child of parent
node.
Applications for Decision Tree - Decision trees have a natural “if … t
hen … else …” construction that makes it fit easily into a programmati
c structure. They also are well suited to categorization problems wher
e attributes or features are systematically checked to determine a fin
al category. For example, a decision tree could be used effectively to
determine the species of an animal.
As a result, the decision making tree is one of the more popular class
ification algorithms being used in Data Mining and Machine Learning. E
xample applications include:
· Evaluation of brand expansion opportunities for a business using his
torical sales data
· Determination of likely buyers of a product using demographic data t
o enable targeting of limited advertisement budget
· Prediction of likelihood of default for applicant borrowers using pr
edictive models generated from historical data
· Help with prioritization of emergency room patient treatment using a
predictive model based on factors such as age, blood pressure, gender,
location and severity of pain, and other measurements
```

· Decision trees are commonly used in operations research, specificall
y in decision analysis, to help identify a strategy most likely to rea
ch a goal.
Because of their simplicity, tree diagrams have been used in a broad r
ange of industries and disciplines including civil planning, energy, f
inancial, engineering, healthcare, pharmaceutical, education, law, and
business.
How does Decision Tree works ?
Decision tree is a type of supervised learning algorithm (having a pre
-defined target variable) that is mostly used in classification proble
ms. It works for both categorical and continuous input and output vari
ables. In this technique, we split the population or sample into two o
r more homogeneous sets (or sub-populations) based on most significant
splitter / differentiator in input variables.
Example:-
Let’s say we have a sample of 30 students with three variables Gender
(Boy/ Girl), Class (IX/ X) and Height (5 to 6 ft). 15 out of these 30
play cricket in leisure time. Now, we want to create a model to predic
t who will play cricket during leisure period? In this problem, we nee
d to segregate students who play cricket in their leisure time based o
n highly significant input variable among all three.
This is where decision tree helps, it will segregate the students base
d on all values of three variable and identify the variable, which cre
ates the best homogeneous sets of students (which are heterogeneous to
each other). In the snapshot below, you can see that variable Gender i
s able to identify best homogeneous sets compared to the other two var
iables.
Decision tree identifies the most significant variable and its value t
hat gives best homogeneous sets of population. To identify the variabl
e and the split, decision tree uses various algorithms.
Types of Decision Trees
Types of decision tree is based on the type of target variable we hav
e. It can be of two types:
Categorical Variable Decision Tree: Decision Tree which has categorica
l target variable then it called as categorical variable decision tre
e. E.g.:- In above scenario of student problem, where the target varia
ble was “Student will play cricket or not” i.e. YES or NO.
Continuous Variable Decision Tree: Decision Tree has continuous target
variable then it is called as Continuous Variable Decision Tree.
E.g.:- Let’s say we have a problem to predict whether a customer will
pay his renewal premium with an insurance company (yes/ no). Here we k
now that income of customer is a significant variable but insurance co
mpany does not have income details for all customers. Now, as we know
this is an important variable, then we can build a decision tree to pr
edict customer income based on occupation, product and various other v
ariables. In this case, we are predicting values for continuous variab
le.
The decision tree algorithm tries to solve the problem, by using tree
representation. Each internal node of the tree corresponds to an attri
bute, and each leaf node corresponds to a class label.
Place the best attribute of the dataset at the root of the tree.
Split the training set into subsets. Subsets should be made in such a
way that each subset contains data with the same value for an attribut
e.
Repeat step 1 and step 2 on each subset until you find leaf nodes in a
ll the branches of the tree.
In decision trees, for predicting a class label for a record we start
from the root of the tree. We compare the values of the root attribute
with record’s attribute. On the basis of comparison, we follow the bra
nch corresponding to that value and jump to the next node.
We continue comparing our record’s attribute values with other interna

l nodes of the tree until we reach a leaf node with predicted class va
lue. The modeled decision tree can be used to predict the target class
or the value.
Assumptions while creating Decision Tree
Some of the assumptions we make while using Decision tree:
At the beginning, the whole training set is considered as the root.
Feature values are preferred to be categorical. If the values are cont
inuous then they are discretized prior to building the model.
Records are distributed recursively on the basis of attribute values.
Order to placing attributes as root or internal node of the tree is do
ne by using some statistical approach.
Advantages of Decision Tree:
Easy to Understand: Decision tree output is very easy to understand ev
en for people from non-analytical background. It does not require any
statistical knowledge to read and interpret them. Its graphical repres
entation is very intuitive and users can easily relate their hypothesi
s.
Useful in Data exploration: Decision tree is one of the fastest way to
identify most significant variables and relation between two or more v
ariables. With the help of decision trees, we can create new variables
/ features that has better power to predict target variable. It can al
so be used in data exploration stage. For e.g., we are working on a pr
oblem where we have information available in hundreds of variables, th
ere decision tree will help to identify most significant variable.
Decision trees implicitly perform variable screening or feature select
ion.
Decision trees require relatively little effort from users for data pr
eparation.
Less data cleaning required: It requires less data cleaning compared t
o some other modeling techniques. It is not influenced by outliers and
missing values to a fair degree.
Data type is not a constraint: It can handle both numerical and catego
rical variables. Can also handle multi-output problems.
Non-Parametric Method: Decision tree is considered to be a non-paramet
ric method. This means that decision trees have no assumptions about t
he space distribution and the classifier structure.
Non-linear relationships between parameters do not affect tree perform
ance.
The number of hyper-parameters to be tuned is almost null.
Disadvantages of Decision Tree:
Over fitting: Decision-tree learners can create over-complex trees tha
t do not generalize the data well. This is called overfitting. Over fi
tting is one of the most practical difficulty for decision tree model
s. This problem gets solved by setting constraints on model parameters
and pruning.
Not fit for continuous variables: While working with continuous numeri
cal variables, decision tree loses information, when it categorizes va
riables in different categories.
Decision trees can be unstable because small variations in the data mi
ght result in a completely different tree being generated. This is cal
led variance, which needs to be lowered by methods like bagging and bo
osting.
Greedy algorithms cannot guarantee to return the globally optimal deci
sion tree. This can be mitigated by training multiple trees, where the
features and samples are randomly sampled with replacement.
Decision tree learners create biased trees if some classes dominate. I
t is therefore recommended to balance the data set prior to fitting wi
th the decision tree.
Information gain in a decision tree with categorical variables gives a
biased response for attributes with greater no. of categories.
Generally, it gives low prediction accuracy for a dataset as compared

to other machine learning algorithms.
Calculations can become complex when there are many class label.
Regression Trees vs Classification Trees
The terminal nodes (or leaves) lies at the bottom of the decision tre
e. This means that decision trees are typically drawn upside down such
that leaves are the bottom & roots are the tops.
Both the trees work almost similar to each other. The primary differen
ces and similarities between Classification and Regression Trees are:
Regression trees are used when dependent variable is continuous. Class
ification Trees are used when dependent variable is categorical.
In case of Regression Tree, the value obtained by terminal nodes in th
e training data is the mean response of observation falling in that re
gion. Thus, if an unseen data observation falls in that region, we'll
make its prediction with mean value.
In case of Classification Tree, the value (class) obtained by terminal
node in the training data is the mode of observations falling in that
region. Thus, if an unseen data observation falls in that region, we'l
l make its prediction with mode value.
Both the trees divide the predictor space (independent variables) into
distinct and non-overlapping regions.
Both the trees follow a top-down greedy approach known as recursive bi
nary splitting. We call it as 'top-down' because it begins from the to
p of tree when all the observations are available in a single region a
nd successively splits the predictor space into two new branches down
the tree. It is known as 'greedy' because, the algorithm cares (looks
for best variable available) about only the current split, and not abo
ut future splits which will lead to a better tree.
This splitting process is continued until a user defined stopping crit
eria is reached. For e.g.: we can tell the algorithm to stop once the
number of observations per node becomes less than 50.
In both the cases, the splitting process results in fully grown trees
until the stopping criteria is reached. But, the fully grown tree is l
ikely to over fit data, leading to poor accuracy on unseen data. This
bring 'pruning'. Pruning is one of the technique used tackle overfitti
ng.

In [4]:

```
data=[data]
```

In [5]:

```
sentences=[]
for s in data:
    sentences.append(sent_tokenize(s))
sentences = [[y] for x in sentences for y in x] # flatten list
```

In [6]:

```
print(len(sentences))
```

```
116
```

In [7]:

```
sentences[0]
```

Out[7]:

```
['A decision tree is a decision support tool that uses a tree-like gra
ph or model of decisions and their possible consequences, including ch
ance event outcomes, resource costs, and utility.']
```

In [8]:

```
word_sentence=[]
for i in sentences:
    for j in i:
        word_sentence.append(word_tokenize(j))
word_sentence
```

Out[8]:

```
[['A',
  'decision',
  'tree',
  'is',
  'a',
  'decision',
  'support',
  'tool',
  'that',
  'uses',
  'a',
  'tree-like',
  'graph',
  'or',
  'model',
  'of',
  'decisions',
  'and',
```

In [9]:

```
#removal of stop words
stop_words = set(stopwords.words('english'))
line=[]
filtered_sentence=[]

for i in word_sentence:
    for j in i:
        if not j in stop_words:
            line.append(j)
    filtered_sentence.append(line)
    line=[]
```

In [10]:

```
filtered_sentence[0]
```

Out[10]:

```
['A',
 'decision',
 'tree',
 'decision',
 'support',
 'tool',
 'uses',
 'tree-like',
 'graph',
 'model',
 'decisions',
 'possible',
 'consequences',
 ',',
 'including',
 'chance',
 'event',
 'outcomes',
 ',',
 'resource',
 'costs',
 ',',
 'utility',
 '.']
```

In [11]:

```
#stemming the words
from nltk.stem import PorterStemmer
ps = PorterStemmer()
stemmed_sentences=[]
line=[]
for i in filtered_sentence:
    for j in i:
       line.append(ps.stem(j))
    stemmed_sentences.append(line)
    line=[]
```

In [12]:

```
# remove punctuations, numbers and special characters
alphabetic_sentences=[]
line=[]
for i in stemmed_sentences:
    line.append(pd.Series(i).str.replace("[^a-zA-Z]", " "))


    alphabetic_sentences.append(line)
    line=[]
```

In [13]:

```
line = []

for i in alphabetic_sentences:

    for j in i:
        l=[]
        for x in j.values:
            if not x ==' ':
                l.append(x.lower())
        line.append(l)
line
```

Out[13]:

```
[['a',
  'decis',
  'tree',
  'decis',
  'support',
  'tool',
  'use',
  'tree lik',
  'graph',
  'model',
  'decis',
  'possibl',
  'consequ',
  'includ',
  'chanc',
  'event',
  'outcom',
  'resourc',
```

In [14]:

```
words=[]
words=[y for x in line for y in x ]

#Taking all distinct words in an array
Distinct_Words=[y for y in set(words)]
```

In [15]:

```
len(Distinct_Words)
```

Out[15]:

```
460
```

In [16]:

```

len(words)
```

Out[16]:

```
1239
```

# Data Visulaizaton

In [17]:

```
for i in range(len(Distinct_Words)):
    print (i,Distinct_Words[i])
```

```
437 girl
438 variou
439 prior
440 drawn
441 snapshot
442 research
443 stabil
444 on
445 poor
446 decision
447 popularli
448 evalu
449 thu
450 jump
451 singl
452 requir
453 ft
454 niklau
455 simplic
```

## Count representation

In [18]:

```
word_Count_matrix=np.empty(shape=(len(line),len(Distinct_Words)))
word_Count_matrix.fill(0)

for i in range(len(line)):
    for j in range(len(line[i])):
        for x in range(len(Distinct_Words)):
            if (Distinct_Words[x]==line[i][j]):
                word_Count_matrix[i][x] = word_Count_matrix[i][x] +1
                continue

word_Count_matrix
```

Out[18]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

## BM-25

In [19]:

```
BM_25_weight=np.empty(shape=(len(line),len(Distinct_Words)))
BM_25_weight.fill(0)


avg_length=np.mean(word_Count_matrix.sum(axis=1))

length_size=np.array(1.5*word_Count_matrix.sum(axis=1)/avg_length)

isf=np.array(np.log(len(line)/np.count_nonzero(word_Count_matrix,axis=0)))

denominator=np.empty(shape=(len(line),len(Distinct_Words)))
for i in range(len(line)):
    denominator[i,:]=(word_Count_matrix[i,:]+length_size[i]+0.5)

BM_25_weight=(word_Count_matrix*isf)/(denominator)
BM_25_weight
```

Out[19]:

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       ...,
       [0.929864  , 1.05715939, 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ]])
```

# Tf-idf

In [20]:

```
tfidf_weight=np.empty(shape=(len(line),len(Distinct_Words)))
tfidf_weight.fill(0)

tfidf_weight = word_Count_matrix*isf
tfidf_weight
```

Out[20]:

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       ...,
       [2.96183072, 3.36729583, 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ]])
```

## Binary

In [21]:

```
word_binary_matrix=np.empty(shape=(len(line),len(Distinct_Words)))
word_binary_matrix.fill(0)

for i in range(len(line)):
    for j in range(len(line[i])):
        for x in range(len(Distinct_Words)):
            if (Distinct_Words[x]==line[i][j]):
                word_binary_matrix[i][x] = 1
                continue
word_binary_matrix
```

Out[21]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

## Euclidean with BM25

In [22]:

```
euclidean_bm25 =euclidean_distances(BM_25_weight)
euclidean_bm25 = (1-(euclidean_bm25/np.max(euclidean_bm25)))
print(euclidean_bm25)
```

```
[[1.         0.09698036 0.22188399 ... 0.14395208 0.19859701 0.1664926
7]
 [0.09698036 1.         0.15848007 ... 0.11287815 0.16630175 0.1642462
9]
 [0.22188399 0.15848007 1.         ... 0.20945314 0.26859172 0.2235459
8]
 ...
 [0.14395208 0.11287815 0.20945314 ... 1.         0.21655412 0.1743425
6]
 [0.19859701 0.16630175 0.26859172 ... 0.21655412 1.         0.3145342
8]
 [0.16649267 0.16424629 0.22354598 ... 0.17434256 0.31453428 1.
]]
```

## Cosine with BM25

In [23]:

```
cosine_bm25=np.empty(shape=(len(line),len(line)))

for i in range(len(line)):
    for j in range(len(line)):
        dot_product = np.dot(BM_25_weight[i], BM_25_weight[j])
        norm_i = np.linalg.norm(BM_25_weight[i])
        norm_j = np.linalg.norm(BM_25_weight[j])
        cosine_bm25[i, j] = dot_product/(norm_i*norm_j)
cosine_bm25
```

Out[23]:

```
array([[1.        , 0.        , 0.07716111, ..., 0.00185534, 0.
,
        0.02179388],
       [0.        , 1.        , 0.        , ..., 0.        , 0.
,
        0.08534227],
       [0.07716111, 0.        , 1.        , ..., 0.00309712, 0.
,
        0.        ],
       ...,
       [0.00185534, 0.        , 0.00309712, ..., 1.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.
,
        0.20468351],
       [0.02179388, 0.08534227, 0.        , ..., 0.        , 0.2046835
1,
        1.        ]])
```

## Euclidean with tfidf

In [24]:

```
euclidean_tfidf =euclidean_distances(tfidf_weight)
euclidean_tfidf = (1-(euclidean_tfidf/np.max(euclidean_tfidf)))
print(euclidean_tfidf)
```

```
[[1.         0.50875844 0.54453759 ... 0.49966453 0.56724457 0.5494181
4]
 [0.50875844 1.         0.63325454 ... 0.59779666 0.68650326 0.6713430
3]
 [0.54453759 0.63325454 1.         ... 0.62116027 0.71634312 0.6841110
7]
 ...
 [0.49966453 0.59779666 0.62116027 ... 1.         0.67178282 0.6435577
]
 [0.56724457 0.68650326 0.71634312 ... 0.67178282 1.         0.7737939
6]
 [0.54941814 0.67134303 0.68411107 ... 0.6435577  0.77379396 1.
]]
```

## Cosine with tfidf

In [25]:

```
cosine_tfidf=np.empty(shape=(len(line),len(line)))

for i in range(len(line)):
    for j in range(len(line)):
        dot_product = np.dot(tfidf_weight[i], tfidf_weight[j])
        norm_i = np.linalg.norm(tfidf_weight[i])
        norm_j = np.linalg.norm(tfidf_weight[j])
        cosine_tfidf[i, j] = dot_product/(norm_i*norm_j)
cosine_tfidf
```

Out[25]:

```
array([[1.        , 0.        , 0.08177021, ..., 0.00180392, 0.
,
        0.02162267],
       [0.        , 1.        , 0.        , ..., 0.        , 0.
,
        0.08534227],
       [0.08177021, 0.        , 1.        , ..., 0.00303513, 0.
,
        0.        ],
       ...,
       [0.00180392, 0.        , 0.00303513, ..., 1.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.
,
        0.20468351],
       [0.02162267, 0.08534227, 0.        , ..., 0.        , 0.2046835
1,
        1.        ]])
```

## Euclidean with count

In [26]:

```
euclidean_count =euclidean_distances(word_Count_matrix)
euclidean_count = (1-(euclidean_count/np.max(euclidean_count)))
print(euclidean_count)
```

```
[[1.         0.52480904 0.58263499 ... 0.50486235 0.56745315 0.5600586
5]
 [0.52480904 1.         0.6407894  ... 0.61478954 0.72175666 0.7103951
5]
 [0.58263499 0.6407894  1.         ... 0.61478954 0.69946285 0.6688241
9]
 ...
 [0.50486235 0.61478954 0.61478954 ... 1.         0.66882419 0.6407894
]
 [0.56745315 0.72175666 0.69946285 ... 0.66882419 1.         0.7874881
4]
 [0.56005865 0.71039515 0.66882419 ... 0.6407894  0.78748814 1.
]]
```

## Cosine with count

In [27]:

```
cosine_count=np.empty(shape=(len(line),len(line)))

for i in range(len(line)):
    for j in range(len(line)):
        dot_product = np.dot(word_Count_matrix[i], word_Count_matrix[j])
        norm_i = np.linalg.norm(word_Count_matrix[i])
        norm_j = np.linalg.norm(word_Count_matrix[j])
        cosine_count[i, j] = dot_product/(norm_i*norm_j)
cosine_count
```

Out[27]:

```
array([[1.        , 0.        , 0.2956562 , ..., 0.05241424, 0.
,
        0.08006408],
       [0.        , 1.        , 0.        , ..., 0.        , 0.
,
        0.13608276],
       [0.2956562 , 0.        , 1.        , ..., 0.0805823 , 0.
,
        0.        ],
       ...,
       [0.05241424, 0.        , 0.0805823 , ..., 1.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.
,
        0.23570226],
       [0.08006408, 0.13608276, 0.        , ..., 0.        , 0.2357022
6,
        1.        ]])
```

## Euclidean with binary

In [28]:

```
euclidean_binary =euclidean_distances(word_binary_matrix)
euclidean_binary = (1-(euclidean_binary/np.max(euclidean_binary)))
print(euclidean_binary)
```

```
[[1.         0.39595955 0.44249591 ... 0.39595955 0.4672864  0.4547502
4]
 [0.39595955 1.         0.48012476 ... 0.48012476 0.59730637 0.5808631
8]
 [0.44249591 0.48012476 1.         ... 0.48012476 0.56504116 0.5206987
1]
 ...
 [0.39595955 0.48012476 0.48012476 ... 1.         0.56504116 0.5206987
1]
 [0.4672864  0.59730637 0.56504116 ... 0.56504116 1.         0.6924376
6]
 [0.45475024 0.58086318 0.52069871 ... 0.52069871 0.69243766 1.
]]
```

## Cosine with binary

In [29]:

```
cosine_binary=np.empty(shape=(len(line),len(line)))

for i in range(len(line)):
    for j in range(len(line)):
        dot_product = np.dot(word_binary_matrix[i], word_binary_matrix[j])
        norm_i = np.linalg.norm(word_binary_matrix[i])
        norm_j = np.linalg.norm(word_binary_matrix[j])
        cosine_binary[i, j] = dot_product/(norm_i*norm_j)
cosine_binary
```

Out[29]:

```
array([[1.        , 0.        , 0.21320072, ..., 0.07106691, 0.
,
        0.09622504],
       [0.        , 1.        , 0.        , ..., 0.        , 0.
,
        0.13608276],
       [0.21320072, 0.        , 1.        , ..., 0.09090909, 0.
,
        0.        ],
       ...,
       [0.07106691, 0.        , 0.09090909, ..., 1.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.
,
        0.23570226],
       [0.09622504, 0.13608276, 0.        , ..., 0.        , 0.2357022
6,
        1.        ]])
```

## Jaccard with binary

In [30]:

```
def pairwise_jaccard(X):
    """Computes the Jaccard distance between the rows of `X`.
    """
    X = X.astype(bool).astype(int)

    intrsct = X.dot(X.T)
    row_sums = intrsct.diagonal()
    unions = row_sums[:,None] + row_sums - intrsct
    dist = intrsct / unions
    return dist
```

In [31]:

```
jaccard_binary = pairwise_jaccard(word_binary_matrix)
print(jaccard_binary)
```

```
[[1.         0.         0.11538462 ... 0.03571429 0.         0.0434782
6]
 [0.         1.         0.         ... 0.         0.         0.0714285
7]
 [0.11538462 0.         1.         ... 0.04761905 0.         0.
]
 ...
 [0.03571429 0.         0.04761905 ... 1.         0.         0.
]
 [0.         0.         0.         ... 0.         1.         0.125
]
 [0.04347826 0.07142857 0.         ... 0.         0.125      1.
]]
```

## summary function

In [32]:

```
def summary_func(similarity_matrix):
    summary=""
    n_clusters = int(np.ceil(len(similarity_matrix)**0.5))
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    kmeans = kmeans.fit(similarity_matrix)
    avg = []
    closest = []
    for j in range(n_clusters):
        idx = np.where(kmeans.labels_ == j)[0]
        avg.append(np.mean(idx))
    closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_,similarity_m
    ordering = sorted(range(n_clusters), key=lambda k: avg[k])
    summary = ' '.join([sentences[closest[idx]][0] for idx in ordering])
    return summary
```

## euclidean BM25 summary

In [33]:

```
euclidean_BM25_summary=summary_func(euclidean_bm25)
print(euclidean_BM25_summary)
```

```
15 out of these 30 play cricket in leisure time. Advantages of Decisio
n Tree:
Easy to Understand: Decision tree output is very easy to understand ev
en for people from non-analytical background. To identify the variable
and the split, decision tree uses various algorithms. Repeat step 1 an
d step 2 on each subset until you find leaf nodes in all the branches
of the tree. :- Let’s say we have a problem to predict whether a custo
mer will pay his renewal premium with an insurance company (yes/ no).
Decision Tree algorithms are referred to as CART (Classification and R
egression Trees). It can be of two types:
Categorical Variable Decision Tree: Decision Tree which has categorica
l target variable then it called as categorical variable decision tre
e. Less data cleaning required: It requires less data cleaning compare
d to some other modeling techniques. Over fitting is one of the most p
ractical difficulty for decision tree models. This means that decision
trees have no assumptions about the space distribution and the classif
ier structure. Thus, if an unseen data observation falls in that regio
n, we’ll make its prediction with mean value.
```

## cosine BM25 summary

In [34]:

```
cosine_BM25_summary=summary_func(cosine_bm25)
print(cosine_BM25_summary)
```

```
In this problem, we need to segregate students who play cricket in the
ir leisure time based on highly significant input variable among all t
hree. Splitting: It is a process of dividing a node into two or more s
ub-nodes. E.g. We compare the values of the root attribute with recor
d’s attribute. Decision tree is a type of supervised learning algorith
m (having a pre-defined target variable) that is mostly used in classi
fication problems. A decision tree is a decision support tool that use
s a tree-like graph or model of decisions and their possible consequen
ces, including chance event outcomes, resource costs, and utility. For
e.g., we are working on a problem where we have information available
in hundreds of variables, there decision tree will help to identify mo
st significant variable. If the values are continuous then they are di
scretized prior to building the model. This means that decision trees
are typically drawn upside down such that leaves are the bottom & root
s are the tops. It is known as ‘greedy’ because, the algorithm cares
(looks for best variable available) about only the current split, and
not about future splits which will lead to a better tree. In case of C
lassification Tree, the value (class) obtained by terminal node in the
training data is the mode of observations falling in that region.
```

## euclidean tfidf summary

In [35]:

```
euclidean_tfidf_summary=summary_func(euclidean_tfidf)
print(euclidean_tfidf_summary)
```

```
“The possible solutions to a given problem emerge as the leaves of a t
ree, each node representing a point of deliberation and decision.”
- Niklaus Wirth (1934 – ), Programming language designer
Methods like decision trees, random forest, gradient boosting are bein
g popularly used in all kinds of data science problems. Parent and Chi
ld Node: A node, which is divided into sub-nodes is called parent node
of sub-nodes whereas sub-nodes are the child of parent node. Example a
pplications include:
· Evaluation of brand expansion opportunities for a business using his
torical sales data
· Determination of likely buyers of a product using demographic data t
o enable targeting of limited advertisement budget
· Prediction of likelihood of default for applicant borrowers using pr
edictive models generated from historical data
· Help with prioritization of emergency room patient treatment using a
predictive model based on factors such as age, blood pressure, gender,
location and severity of pain, and other measurements
· Decision trees are commonly used in operations research, specificall
y in decision analysis, to help identify a strategy most likely to rea
ch a goal. Because of their simplicity, tree diagrams have been used i
n a broad range of industries and disciplines including civil plannin
g, energy, financial, engineering, healthcare, pharmaceutical, educati
on, law, and business. whether a coin flip comes up heads or tails), e
ach branch represents the outcome of the test, and each leaf node repr
esents a class label (decision taken after computing all attributes).
Each internal node of the tree corresponds to an attribute, and each l
eaf node corresponds to a class label. Now, as we know this is an impo
rtant variable, then we can build a decision tree to predict customer
income based on occupation, product and various other variables. It ca
n also be used in data exploration stage. This is called variance, whi
ch needs to be lowered by methods like bagging and boosting. Decision
trees implicitly perform variable screening or feature selection. For
e.g.
```

## cosine tfidf summary

In [36]:

```
cosine_tfidf_summary=summary_func(cosine_tfidf)
print(cosine_tfidf_summary)
```

```
Decision Node: When a sub-node splits into further sub-nodes, then it
is called decision node. Decision tree identifies the most significant
variable and its value that gives best homogeneous sets of population.
“The possible solutions to a given problem emerge as the leaves of a t
ree, each node representing a point of deliberation and decision.”
- Niklaus Wirth (1934 – ), Programming language designer
Methods like decision trees, random forest, gradient boosting are bein
g popularly used in all kinds of data science problems. Types of Decis
ion Trees
Types of decision tree is based on the type of target variable we hav
e. E.g. Not fit for continuous variables: While working with continuou
s numerical variables, decision tree loses information, when it catego
rizes variables in different categories. We compare the values of the
root attribute with record’s attribute. The primary differences and si
milarities between Classification and Regression Trees are:
Regression trees are used when dependent variable is continuous. Pruni
ng is one of the technique used tackle overfitting. Decision trees req
uire relatively little effort from users for data preparation. Thus, i
f an unseen data observation falls in that region, we’ll make its pred
iction with mode value.
```

## euclidean count summary

In [37]:

```
euclidean_count_summary=summary_func(euclidean_count)
print(euclidean_count_summary)
```

```
Parent and Child Node: A node, which is divided into sub-nodes is call
ed parent node of sub-nodes whereas sub-nodes are the child of parent
node. Example applications include:
· Evaluation of brand expansion opportunities for a business using his
torical sales data
· Determination of likely buyers of a product using demographic data t
o enable targeting of limited advertisement budget
· Prediction of likelihood of default for applicant borrowers using pr
edictive models generated from historical data
· Help with prioritization of emergency room patient treatment using a
predictive model based on factors such as age, blood pressure, gender,
location and severity of pain, and other measurements
· Decision trees are commonly used in operations research, specificall
y in decision analysis, to help identify a strategy most likely to rea
ch a goal. Each internal node of the tree corresponds to an attribute,
and each leaf node corresponds to a class label. It can be of two type
s:
Categorical Variable Decision Tree: Decision Tree which has categorica
l target variable then it called as categorical variable decision tre
e. :- Let’s say we have a problem to predict whether a customer will p
ay his renewal premium with an insurance company (yes/ no). Advantages
of Decision Tree:
Easy to Understand: Decision tree output is very easy to understand ev
en for people from non-analytical background. It does not require any
statistical knowledge to read and interpret them. Decision tree learne
rs create biased trees if some classes dominate. Unlike linear models,
they map non-linear relationships quite well. YES or NO. Applications
for Decision Tree - Decision trees have a natural “if … then … else …”
construction that makes it fit easily into a programmatic structure.
```

## cosine count summary

In [38]:

```
cosine_count_summary=summary_func(cosine_count)
print(cosine_count_summary)
```

```
Parent and Child Node: A node, which is divided into sub-nodes is call
ed parent node of sub-nodes whereas sub-nodes are the child of parent
node. In this problem, we need to segregate students who play cricket
in their leisure time based on highly significant input variable among
all three. A decision tree is a decision support tool that uses a tree
-like graph or model of decisions and their possible consequences, inc
luding chance event outcomes, resource costs, and utility. Records are
distributed recursively on the basis of attribute values. With the hel
p of decision trees, we can create new variables / features that has b
etter power to predict target variable. Decision tree learners create
biased trees if some classes dominate. YES or NO. E.g. Both the trees
follow a top-down greedy approach known as recursive binary splitting.
It can also be used in data exploration stage. In case of Regression T
ree, the value obtained by terminal nodes in the training data is the
mean response of observation falling in that region.
```

# euclidean binary summary

In [39]:

```
euclidean_binary_summary=summary_func(euclidean_binary)
print(euclidean_binary_summary)
```

```
A decision tree is a decision support tool that uses a tree-like graph
or model of decisions and their possible consequences, including chanc
e event outcomes, resource costs, and utility. “The possible solutions
to a given problem emerge as the leaves of a tree, each node represent
ing a point of deliberation and decision.”
- Niklaus Wirth (1934 – ), Programming language designer
Methods like decision trees, random forest, gradient boosting are bein
g popularly used in all kinds of data science problems. Example applic
ations include:
· Evaluation of brand expansion opportunities for a business using his
torical sales data
· Determination of likely buyers of a product using demographic data t
o enable targeting of limited advertisement budget
· Prediction of likelihood of default for applicant borrowers using pr
edictive models generated from historical data
· Help with prioritization of emergency room patient treatment using a
predictive model based on factors such as age, blood pressure, gender,
location and severity of pain, and other measurements
· Decision trees are commonly used in operations research, specificall
y in decision analysis, to help identify a strategy most likely to rea
ch a goal. Decision tree is a type of supervised learning algorithm (h
aving a pre-defined target variable) that is mostly used in classifica
tion problems. In this technique, we split the population or sample in
to two or more homogeneous sets (or sub-populations) based on most sig
nificant splitter / differentiator in input variables. Can also handle
multi-output problems. Greedy algorithms cannot guarantee to return th
e globally optimal decision tree. Its graphical representation is very
intuitive and users can easily relate their hypothesis. In case of Cla
ssification Tree, the value (class) obtained by terminal node in the t
raining data is the mode of observations falling in that region. YES o
r NO. Advantages of Decision Tree:
Easy to Understand: Decision tree output is very easy to understand ev
en for people from non-analytical background.
```

# cosine binary summary

In [40]:

```
cosine_binary_summary=summary_func(cosine_binary)
print(cosine_binary_summary)
```

```
Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal nod
e. :- In above scenario of student problem, where the target variable
was “Student will play cricket or not” i.e. Decision tree is a type of
supervised learning algorithm (having a pre-defined target variable) t
hat is mostly used in classification problems. Its graphical represent
ation is very intuitive and users can easily relate their hypothesis.
Each internal node of the tree corresponds to an attribute, and each l
eaf node corresponds to a class label. Decision trees implicitly perfo
rm variable screening or feature selection. Decision tree learners cre
ate biased trees if some classes dominate. E.g. Records are distribute
d recursively on the basis of attribute values. Both the trees follow
a top-down greedy approach known as recursive binary splitting. This b
ring ‘pruning’.
```

## jaccard binary summary

In [41]:

```
jaccard_binary_summary=summary_func(jaccard_binary)
print(jaccard_binary_summary)
```

```
Each internal node of the tree corresponds to an attribute, and each l
eaf node corresponds to a class label. 15 out of these 30 play cricket
in leisure time. Splitting: It is a process of dividing a node into tw
o or more sub-nodes. Decision Tree algorithms are referred to as CART
(Classification and Regression Trees). Records are distributed recursi
vely on the basis of attribute values. Data type is not a constraint:
It can handle both numerical and categorical variables. Types of Decis
ion Trees
Types of decision tree is based on the type of target variable we hav
e. This problem gets solved by setting constraints on model parameters
and pruning. E.g. Applications for Decision Tree - Decision trees have
a natural “if … then … else …” construction that makes it fit easily i
nto a programmatic structure. Thus, if an unseen data observation fall
s in that region, we’ll make its prediction with mean value.
```

In [ ]:

In [ ]:

In [ ]: