1. Aim: Build Term-Document incidence matrix and process boolean queries.

Program:

```
# Importing Modules
import numpy as np
import re
# Construction of Matrix
def termIncidenceMatrix(l,n):
    docs = []
    terms = set()
    for i in l:
        file = open(i,'r').read().lower()
        content = re.sub("[^a-z0-9]"," ",file).split()
        terms.update(content)
        docs.append(set(content))
    mat = np.zeros((len(terms),n),dtype=bool)
    terms = sorted(terms)
    for i in range(len(terms)):
        for j in range(n):
            if terms[i] in docs[j]:
                mat[i][j] = 1
            else:
                mat[i][j] = 0
    return mat,terms
# Postfix Building
def precedence(op):
    if op == 'NOT':
        return 3
    elif op == 'AND':
        return 2
    elif op == 'OR':
        return 1
    else:
        return 0
def postfix(l):
    pf = []
    stack = []
    opList = ['AND','OR','NOT']
    for i in l:
        if i in opList:
            if stack == []:
                stack.append(i)
            else:
                while(stack != [] and precedence(i) <= precedence(stack[-1])):
                    pf.append(stack.pop())
                stack.append(i)
        elif i == '(':
            stack.append(i)
        elif i == ')':
            while(stack[-1] != '('):
                pf.append(stack.pop())
            stack.pop()
        else:
```

```
            pf.append(i)
    while(stack != []):
        pf.append(stack.pop())
    return pf
def getIncidences(s,d,n):
    if isinstance(s,str):
        try:
            return d[s]
        except:
            return np.zeros(n,dtype=bool)
    else:
        return s
# Query Evaluation
def queryEval(query,d,n):
    pf = postfix(query.split())
    stack = []
    opList = ['AND','OR','NOT']
    if len(pf) == 1:
        return getIncidences(pf[0],d,n)
    for i in pf:
        if i in opList:
            if i == 'NOT':
                op = stack.pop()
                r = ~getIncidences(op,d,n)
            elif i == 'AND':
                op1 = getIncidences(stack.pop(),d,n)
                op2 = getIncidences(stack.pop(),d,n)
                r = op1&op2
            else:
                op1 = getIncidences(stack.pop(),d,n)
                op2 = getIncidences(stack.pop(),d,n)
                r = op1|op2
            stack.append(r)
        else:
            stack.append(i)
    return stack.pop()
n = int(input("Enter the no. of documents : "))
l = []
for i in range(n):
    l.append(input("Enter the doc - "+str(i+1)+" name : "))
l = np.array(l)
TIM,terms = termIncidenceMatrix(l,n)
print("the generated matrix is : \n",TIM.astype(int))
d = {}
for i in range(len(terms)):
    d[terms[i]] = TIM[i]
rules = """
The term insurance matrix have been created.
Rules for entering the query:
 1.There should be a single space between operand and operator
 2.There should be space before And after '(' , ')'
 3.Boolean operators should be in capital and words in small letters.
```

```
Enter the query:
"""
query = input(rules)
resultDoc = queryEval(query,d,n).astype(bool)
if(len(l[resultDoc]) == 0):
   print("No matching documents")
else:
   print(l[resultDoc])
```

Output:

```
Enter the no. of documents : 4
Enter the doc - 1 name : doc1.txt
Enter the doc - 2 name : doc2.txt
Enter the doc - 3 name : doc3.txt
Enter the doc - 4 name : doc4.txt
the generated matrix is :
 [[0 1 0 1]
 [1 1 1 1]
 [1 0 0 0]
 [0 1 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 0 1 0]
 [1 0 0 1]
 [1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 1 0]
 [1 1 1 0]
 [1 1 1 0]
 [0 0 1 0]
 [1 0 0 1]
 [0 1 0 0]
 [0 0 0 1]
 [1 0 0 0]
...
 [0 1 0 0]
 [0 1 1 1]
 [0 0 1 0]
 [0 0 0 1]]

The term insurance matrix have been created.
Rules for entering the query:
 1.There should be a single space between operand and operator
 2.There should be space before And after '(' , ')'
 3.Boolean operators should be in capital and words in small letters.
Enter the query:
words AND pen OR NOT sheet
['doc2.txt' 'doc3.txt']
```

2. Aim: Build inverted index and process boolean queries

Program:

```
# Importing Modules
import re
# Postfix Building
def precedence(op):
    if op == 'NOT':
        return 3
    elif op == 'AND':
        return 2
    elif op == 'OR':
        return 1
    else:
        return 0

def postfix(l):
    pf = []
    stack = []
    opList = ['AND','OR','NOT']
    for i in l:
        if i in opList:
            if stack == []:
                stack.append(i)
            else:
                while(stack != [] and precedence(i) <= precedence(stack[-1])):
                    pf.append(stack.pop())
                stack.append(i)
        elif i == '(':
            stack.append(i)
        elif i == ')':
            while(stack[-1] != '('):
                pf.append(stack.pop())
            stack.pop()
        else:
            pf.append(i)
    while(stack != []):
        pf.append(stack.pop())
    return pf
def getPostings(s,d):
    if isinstance(s,str):
        try:
            return d[s]
        except:
            return set()
    else:
        return s
# Query Evaluation
def queryEval(query,d,l):
```

```
    pf = postfix(query.split())
    stack = []
    opList = ['AND','OR','NOT']
    if len(pf) == 1:
        return getPostings(pf[0],d)
    else:
        for i in pf:
            if i in opList:
                if i == 'NOT':
                    op = stack.pop()
                    r = set(l).difference(getPostings(op,d))
                elif i == 'AND':
                    op1 = getPostings(stack.pop(),d)
                    op2 = getPostings(stack.pop(),d)
                    r = op1.intersection(op2)
                else:
                    op1 = getPostings(stack.pop(),d)
                    op2 = getPostings(stack.pop(),d)
                    r = op1.union(op2)
                stack.append(r)
            else:
                stack.append(i)
        return set() if len(stack) == 0 else stack.pop()
# Index construction
docs = []
terms = set()
l = input("Enter the documents : ").split()
# l = ["doc1.txt","doc2.txt","doc3.txt","doc4.txt"]

for i in l:
    file = open(i,'r').read().lower()
    content = re.sub("[^a-z0-9]"," ",file).split()
    terms.update(content)
    docs.append(set(content))

d = {}
for i in terms:
    post_list = []
    for j in range(len(docs)):
        if i in docs[j]:
            post_list.append(l[j])
    d[i] = set(post_list)
print(d)

# query reading
rules = """
The inverted index have been created.
Rules for entering the query:
 1.There should be a single space between operand and operator
```

```
 2.There should be space before And after '(' , ')'
 3.Boolean operators should be in capital and words in small letters.
Enter the query:
"""
query = input(rules)
# resultDoc = queryEval("drug AND op",d,l)
resultDoc = queryEval(query,d,l)
if(len(resultDoc) == 0):
   print("No relevant documents")
else:
   print(resultDoc)
```

Output:

```
Enter the documents : doc1.txt doc2.txt doc3.txt doc4.txt

{'ink': {'doc4.txt', 'doc1.txt'}, 'things': {'doc1.txt'}, 'ideas': {'doc3.txt'}, 'the': {'doc3.txt'}, 'and':
{'doc4.txt', 'doc1.txt', 'doc2.txt', 'doc3.txt'}, 'make': {'doc1.txt'}, 'on': {'doc3.txt'}, 'a': {'doc4.txt',
'doc2.txt'}, 'so': {'doc2.txt'}, 'with': {'doc4.txt', 'doc2.txt', 'doc3.txt'}, 'words': {'doc3.txt'}, 'beautiful':
{'doc1.txt'}, 'such': {'doc1.txt'}, 'tell': {'doc4.txt'}, 'flow': {'doc3.txt'}, 'world': {'doc4.txt'}, 'sweet':
{'doc2.txt'}, 'shape': {'doc3.txt'}, 'story': {'doc4.txt'}, 'together': {'doc2.txt'}, 'create': {'doc4.txt'}, 'meet':
{'doc2.txt'}, 'paper': {'doc1.txt', 'doc2.txt', 'doc3.txt'}, 'bond': {'doc2.txt'}, 'out': {'doc3.txt'}, 'sheet':
{'doc4.txt', 'doc1.txt'}, 'to': {'doc4.txt'}, 'pen': {'doc1.txt', 'doc2.txt', 'doc3.txt'}, 'take': {'doc3.txt'}}


The inverted index have been created.
Rules for entering the query:
 1.There should be a single space between operand and operator
 2.There should be space before And after '(' , ')'
 3.Boolean operators should be in capital and words in small letters.
Enter the query:
ink AND sheet OR sweet
{'doc4.txt', 'doc1.txt', 'doc2.txt'}
```

3. Aim: Build positional index and process phrase queries

Program:

```
# Importing Modules
import re
# index construction
s=input("Enter file names:").split() #file names(s)
# s=['doc1.txt','doc2.txt' ,'doc3.txt', 'doc4.txt']
x=[]#file content
for i in s:
    file = re.sub("[^a-z0-9]"," ",open(i,'r').read().lower()).split()
    x.append(file)
y=sorted(set(sum(x,[])))#terms
d={}#parent dictionary
for i in y:
    m={} # temporary list
    for j in range(len(s)):
        l=[] #list to hold document id
        if i in x[j]:
            l.append(j+1)
            ind=[] #indexes of the term that appeared in document
            for k in range(len(x[j])): #k is the required index
                if i==x[j][k]:
                    ind.append(k)
            m[l[0]]=ind
    d[i]=m #placing the list as the value for the term in  the dictionary
print(d)
# query evaluation
def QueryEval(query,d,rel_doc):
    if len(rel_doc) == 1:
        return rel_doc[0]
    comm_doc = rel_doc[0]
    for i in rel_doc:
        comm_doc = comm_doc.intersection(i)
    result = set()
    for i in comm_doc:
        z =[]  # storing of temporary result
        for j in range(len(query)-1):
            if(z == []):
                l1 = d[query[j]][i]
            else:
                l1 = z
            l2 = d[query[j+1]][i]
            x = 0;y = 0
            len1 = len(l1)-1;len2 = len(l2)-1
            while(x<=len1 and y<=len2):
                if l1[x]+1 == l2[y]:
                    z.append(l2[y])
                    x+=1
```

```
                y+=1
            elif l1[x]+1 < l2[y]:
                x+=1
            else:
                y+=1
        if(len(z) == len(query)-1):
            result.add(i)
            break
    return result
# query reading
query = input("Enter the phrase : ").lower().split()
rel_doc = []
for i in query:
    if i in d.keys():
        rel_doc.append(set(d[i].keys()))
    else:
        print("No relevant documents")
        break
else:
    resultDoc = QueryEval(query,d,rel_doc)
    if(len(resultDoc) == 0):
        print("No relevant documents")
    else:
        print("relevant document ids : ",resultDoc)
```

Output:

Enter file names:doc1.txt doc2.txt doc3.txt doc4.txt

{'a': {2: [6], 4: [0, 5]}, 'and': {1: [1, 4], 2: [2], 3: [6], 4: [4, 11]}, 'beautiful': {1: [8]}, 'bond': {2: [7]}, 'create': {4: [8]}, 'flow': {3: [1]}, 'ideas': {3: [7]}, 'ink': {1: [3], 4: [10]}, 'make': {1: [6]}, 'meet': {2: [4]}, 'on': {3: [3]}, 'out': {3: [2]}, 'paper': {1: [2], 2: [3], 3: [5]}, 'pen': {1: [0], 2: [1], 3: [12]}, 'shape': {3: [9]}, 'sheet': {1: [5], 4: [12]}, 'so': {2: [8]}, 'story': {4: [1]}, 'such': {1: [7]}, 'sweet': {2: [9]}, 'take': {3: [8]}, 'tell': {4: [3]}, 'the': {3: [4, 11]}, 'things': {1: [9]}, 'to': {4: [2, 7]}, 'together': {2: [0]}, 'with': {2: [5], 3: [10], 4: [9]}, 'words': {3: [0]}, 'world': {4: [6]}}

Enter the phrase : Ideas take shape
relevant document ids : {3}

4. Aim: Build bi-gram index and process wildcard queries

Program:

```
# Importing modules
import re
# Query evaluation
def getPostings(s,d):
   if isinstance(s,str):
      try:
         return d[s]
      except:
         return set()
   else:
      return s
def bqueryEval(phrase,d):
   phrase = [" ".join(phrase[i:i+2]) for i in range(0,len(phrase)-1,1)]
   phrase += ['AND']*(len(phrase)-1)
   pf = phrase
   stack = []
   if len(pf) == 1:
      return getPostings(pf[0],d)
   else:
      for i in pf:
         if i == 'AND':
               op1 = getPostings(stack.pop(),d)
               op2 = getPostings(stack.pop(),d)
               r = op1.intersection(op2)
               stack.append(r)
         else:
            stack.append(i)
      return set() if len(stack) == 0 else stack.pop()
# Index Construction
docs = []
terms = set()
l = input("Enter the documents : ").split()
# l = ["doc1.txt","doc2.txt","doc3.txt","doc4.txt"]

for i in l:
   file = open(i,'r').read().lower()
   content = re.sub("[^a-z0-9]"," ",file).split()
   biwords = [" ".join(content[i:i+2]) for i in range(0,len(content)-1,1)]
   terms.update(biwords)
   docs.append(set(biwords))

d = {}
for i in terms:
   post_list = []
   for j in range(len(docs)):
      if i in docs[j]:
```

```
            post_list.append(j+1)
    d[i] = set(post_list)

print(d)
# query reading

phrase = input("Enter the phrase : ").lower()
r = bqueryEval(phrase.split(),d)
if len(r) == 0:
    print("no relevant documents")
else:
    print("relevant document ids are : ",r)
```

Output:

Enter the documents : doc1.txt doc2.txt doc3.txt doc4.txt

{'and paper': {1, 2}, 'create with': {4}, 'with a': {2}, 'pen and': {1, 2}, 'with the': {3}, 'sheet make': {1}, 'meet with': {2}, 'world to': {4}, 'paper and': {3}, 'and sheet': {1, 4}, 'shape with': {3}, 'out on': {3}, 'with ink': {4}, 'flow out': {3}, 'the pen': {3}, 'to tell': {4}, 'paper ink': {1}, 'and ideas': {3}, 'such beautiful': {1}, 'on the': {3}, 'make such': {1}, 'beautiful things': {1}, 'a story': {4}, 'story to': {4}, 'ideas take': {3}, 'take shape': {3}, 'a world': {4}, 'paper meet': {2}, 'a bond': {2}, 'so sweet': {2}, 'tell and': {4}, 'and a': {4}, 'words flow': {3}, 'to create': {4}, 'the paper': {3}, 'ink and': {1, 4}, 'bond so': {2}, 'together pen': {2}}

Enter the phrase : Words flow out on the paper
relevant document ids are : {3}

5. Aim: Implement skip pointers

Program:

```
import math
def SkipIntersect(p1,p2):
   ans = []
   i = 0;j = 0
   l1 = len(p1);l2 = len(p2)
   skip1 = math.floor(math.sqrt(len(p1)))
   skip2 = math.floor(math.sqrt(len(p2)))
   while(i<l1 and j<l2):
      if p1[i] == p2[j]:
         ans.append(p1[i])
         i+=1
         j+=1
      elif p1[i] < p2[j]:
         while(i%skip1 == 0 and i+skip1 <= l1-1 and p1[i+skip1] <= p2[j]):
               i+=skip1
         else:
            i+=1
      else:
         while(j%skip2 == 0 and j+skip2 <= l2-1 and p2[j+skip2] <= p1[i]):
               j+=skip2
         else:
            j+=1
   return ans
p1 = [1,2,3,4,9,12,18,37,72,93,103,109,135,143,147,150]
p2 = [4,5,12,19,35,70,71,72,104]
print("After intrersection : ",SkipIntersect(p1,p2))
```

Output:

```
After intrersection :  [4, 12, 72]
```

6. Aim: Correct spellings in the query using edit distance

Program:

```
# Correcting spellings in the query using edit distance
import numpy as np
import collections as c
import copy as cp
# s=input("Enter file names:").split()
s=['doc5.txt','doc6.txt' ,'doc7.txt']
d=cp.deepcopy(s)
x=[]
for i in s:
    f=open(i,'r').read().split()
    x.append(f)
y=sorted(set(sum(x,[])))
print(y)
def findDis(a,b):
    alen=len(a)+1
    blen=len(b)+1
    m=[[0 for i in range(alen)] for j in range(blen)]
    for i in range(1,alen):
        m[0][i]=i
    for j in range(1,blen):
        m[j][0]=j
    for j in range(1,alen):
        for i in range(1,blen):
            if a[j-1]==b[i-1]:
                f=0
            else:
                f=1
            one=m[i-1][j-1]+f
            two=m[i-1][j]+1
            three=m[i][j-1]+1
            m[i][j]=min(one,two,three)
    return m
def printvals(a,b):
    c=findDis(a,b)
    l=a.split()
    print("\t\t",end=' ')
    for k in range(len(a)):
        print(a[k],"\t",end='')
    print("\n")
    for i in range(len(c)):
        if i==0:
            print('\t',end='')
        if i>0:
            print(b[i-1],'\t',end='')
        for j in range(len(c[0])):
            print(c[i][j],'\t' ,end='')
```

```
        print("\n")
    print("Edit Distance=",c[-1][-1])
    return
q = input("Enter query:").split(); w=cp.deepcopy(q)
for i in range(len(q)):
    dis=999
    if q[i] not in y:
        for j in y:
            m=findDis(q[i],j); val=m[-1][-1]
            if val < dis :
                w[i]=j; dis=val
print(w)
for i in range(len(q)):
    if w[i]!=q[i]:
        printvals(q[i],w[i])
```

Output:

```
['and', 'for', 'me', 'tea', 'two', 'you']
Enter query:twwo yuu mm
['two', 'you', 'me']
                t       w       w       o

        0       1       2       3       4

t       1       0       1       2       3

w       2       1       0       1       2

o       3       2       1       1       1
Edit Distance= 1
                y       u       u

        0       1       2       3

y       1       0       1       2

o       2       1       1       2

u       3       2       1       1
Edit Distance= 1
                m       m

        0       1       2

m       1       0       1

e       2       1       1
Edit Distance= 1
```

7. Aim: Implement BSBI algorithm

Program:

```
# BSBI implementation
import nltk
import os
import copy as cp
import collections as c
import numpy as np
import pandas as pd
import pickle
from nltk.corpus import stopwords
sw=set(stopwords.words('english'))
p='Cranfield Data Set'
s=os.listdir(p)
di='opdir'
par='C:/Users/User/Desktop/jup/'
path=os.path.join(par, di)
os.mkdir(path)
a=1
b=1
docid={}
termid={}
d=cp.deepcopy(s)
x=[]
st='pair'
for i in s:
    block=[]
    invind={}
    f=set(open(p+'/'+i,'r').read().split())
    f=f.difference(sw) #all stopwords removed
    f=sorted(list(f))
    if i not in docid.keys():
        docid[i]=a
        a+=1
        for j in f:
            if j not in termid.keys():
                termid[j]=b
                b+=1
            block.append([docid[i],termid[j]])
    for j in range(len(block)):
        if block[j][1] not in invind.keys():
            invind[block[j][1]]=[]
        l=invind[block[j][1]]
        l.append(block[j][0])
        invind[block[j][1]]=sorted(l)
    name='C:/Users/User/Desktop/jup/opdir/'+st+str(s.index(i)+1)+'.pkl'
    with open(name, 'wb') as zx:
        pickle.dump(invind, zx)
```

```
def findkey(d,value):
   for k,v in d.items():
      if value == v:
         return k
n=os.listdir(di)
mainind={}
for i in n:
   with open(par+di+'/'+i, 'rb') as zx:
      block=pickle.load(zx)
      for k,v in block.items():
         key=findkey(termid,k)
         if key not in mainind.keys():
            mainind[key]=v
         else:
            l=mainind[key]+v
            mainind[key]=sorted(l)
print(mainind)
```

Output:

{'agree': [1, 14, 72, 165, 177, 233, 289, 318, 330, 363, 383, 417, 648, 666, 688, 728, 830, 869, 887, 903, 923, 950, 959, 996, 1118, 1185, 1191, 1199, 1204, 1269 ……, 'formulating': [997], '(considered': [998], '180)': [999]}

8. Aim: Implement SPIMI algorithm

Program:

```
# SPIMI implementation
import nltk
import os
import copy as cp
import collections as c
import numpy as np
import pandas as pd
import pickle
from nltk.corpus import stopwords
sw=set(stopwords.words('english'))
p='Cranfield Data Set'
s=os.listdir(p)
di='opdirforSPIMI'
par='C:/Users/exam2/Desktop/IR/'
path=os.path.join(par, di)
os.mkdir(path)
a=1
docid={}
d=cp.deepcopy(s)
x=[]
invind={}
for i in s:
    block=[]
    f=set(open(p+'/'+i,'r').read().split())
    f=f.difference(sw) #all stopwords removed
    f=sorted(list(f))
    if i not in docid.keys():
        docid[i]=a
        a+=1
        for j in f:
            block.append([j,docid[i]])
    for j in range(len(block)):
        if block[j][0] not in invind.keys():
            invind[block[j][0]]=[]
        l=invind[block[j][0]]
        l.append(block[j][1])
        invind[block[j][0]]=l
name='C:/Users/exam2/Desktop/IR/opdirforSPIMI/'+'SPIMIoutput.pkl'
with open(name, 'wb') as zx:
    pickle.dump(invind, zx)
```

Output:

{'agree': [1, 14, 72, 165, 177, 233, 289, 318, 330, 363, 383, 417, 648, 666, 688, 728, 830, 869, 887, 903, 923, 950, 959, 996, 1118, 1185, 1191, 1199, 1204, 1269 ……, 'formulating': [997], '(considered': [998], '180)': [999]}

9. Aim: Implement vector space model with various functions.

Program:

```
# Vector Space Model
import numpy as np
import collections as c
import copy as cp
#s=input("Enter file names:").split()
s=['DocE1.txt','DocE2.txt' ,'DocE3.txt','DocE4.txt']
d=cp.deepcopy(s)
x=[]
for i in s:
    f=open(i,'r').read().split()
    x.append(f)
y=sorted(set(sum(x,[])))
tcm=[]
for i in range(len(s)):
    m=[]
    freq=c.Counter(x[i])
    for j in y:
        if j in x[i]:
            m.append(freq[j])
        else:
            m.append(0)
    tcm.append(m)
tf=[]
for i in tcm:
    temp=[]
    for j in i:
        if j!=0:
            temp.append(round(1+np.log(j),2))
        else:
            temp.append(0)
    tf.append(temp)

idf=[]
for i in y:
    n=0
    for j in range(len(d)):
        if i in x[j]:
            n+=1
    if n!=0:
        idf.append(round(np.log(len(d)/n),2))
    else:
        idf.append(0)

tf_idf=[]
for i in tf:
    temp=[]
```

```
    for j in range(len(i)):
        tfidf=i[j]*idf[j]
        temp.append(tfidf)
    tf_idf.append(temp)
tf_idf
```

Output:

```
[[1.4489999999999998, 0.609, 0.0],
 [0.0, 0.0, 0.0],
 [0.0, 0.4900999999999999, 0.0],
 [1.4489999999999998, 0.29, 0.0]]
```

10. Aim: Implement Naïve Bayes classification algorithm

Program:

```
# Naïve Bayes classification
import numpy as np
import pandas as pd
data=pd.read_csv("golf_df.csv")
col=list(data.columns)
def makecounts(l):
    q=list(set(l))
    c=[]
    for i in q:
        c.append(l.count(i))
    return [q,c]
arr=[]
for i in col:
    l=list(data[i])
    arr.append(makecounts(l))
inp=[]
for i in range(len(col)-1):
    print('Select appropriate option for',col[i])
    for j in range(len(arr[i][0])):
        print(j+1,arr[i][0][j],sep='-->')
    x=int(input("Enter option:"))
    inp.append(arr[i][0][x-1])
print('\n\n')
data.set_index(['Play']).sort_index()
Po=[]
for i in range(len(arr[-1][1])):
    Po.append(arr[-1][1][i]/sum(arr[-1][1]))
fin=[]
for i in range(len(Po)):
    l=[]
    for j in range(len(inp)):
        des=list(data[col[-1]])
        ref=list(data[col[j]])
        c=0
        for k in range(len(des)):
            if ref[k]==inp[j] and des[k]==arr[-1][0][i]:
                c+=1
        l.append(c)
    fin.append(l)
    fin[i]=np.prod(np.array(fin[i])/arr[-1][1][i])*Po[i]
print('For given input\ninput=',inp,
      '\ndecision for the value -',col[-1],
      ' is',arr[-1][0][fin.index(max(fin))])
```

Output:

```
Select appropriate option for Outlook
1-->overcast
2-->rainy
3-->sunny
Enter option:3
Select appropriate option for Temperature
1-->hot
2-->cool
3-->mild
Enter option:1
Select appropriate option for Humidity
1-->normal
2-->high
Enter option:2
Select appropriate option for Windy
1-->False
2-->True
Enter option:1


For given input
input= ['sunny', 'hot', 'high', False]
decision for the value - Play  is no
```

11. Aim: Implement KNN classification algorithm

Program:

```
# K Nearest Neighbour
import nltk
import os
import copy as cp
import collections as c
import numpy as np
import pandas as pd
import pickle
from nltk.corpus import stopwords
sw=set(stopwords.words('english'))
p='Cranfield Data Set'
s=os.listdir(p)
def distance(a,b):
    a=np.array(a)
    b=np.array(b)
    return np.sqrt(np.dot(a-b,a-b))
d=cp.deepcopy(s)
x=[]
for i in s:
    f=set(open(p+'/'+i,'r').read().split())
    f=f.difference(sw) #all stopwords removed
    f=sorted(list(f))
    x.append(f)
y=sorted(set(sum(x,[])))
tcm=[]
for i in range(len(s)):
    m=[]
    freq=c.Counter(x[i])
    for j in y:
        if j in x[i]:
            m.append(freq[j])
        else:
            m.append(0)
    tcm.append(m)
tf=[]
for i in tcm:
    temp=[]
    for j in i:
        if j!=0:
            temp.append(round(1+np.log(j),2))
        else:
            temp.append(0)
    tf.append(temp)
idf=[]
for i in y:
    n=0
```

```
    for j in range(len(d)):
        if i in x[j]:
            n+=1
    if n!=0:
        idf.append(round(np.log(len(d)/n),2))
    else:
        idf.append(0)
tf_idf=[]
for i in tf:
    temp=[]
    for j in range(len(i)):
        tfidf=i[j]*idf[j]
        temp.append(tfidf)
    tf_idf.append(temp)
query_doc=input("Enter the document path:")
freq=c.Counter(query_doc)
inp=[]
for i in y:
    if i in query_doc:
        inp.append(freq[i])
    else:
        inp.append(0)
vals=[]
for i in range(len(s)):
    vals.append(distance(tf_idf[i],inp))
sort_vals=sorted(vals,reverse=True)
ranked_order=[]
for i in sort_vals:
    ind=vals.index(i)
    vals.pop(ind)
    ranked_order.append(s[ind])
    s.pop(ind)
h=int(input("Enter number of classes to be classified into to feed data:"))
vote=[0]*h
feed_data=np.array_split(d,h)
k=int(input("Enter K :"))
for i in range(h):
    for j in range(k):
        if ranked_order[j] in feed_data[i]:
            vote[i]+=1
print("The new data belongs to ",vote.index(max(vote))+1)
```

Output:

```
Enter the document path:Cranfield Data Set/cranfield0002.txt
Enter number of classes to be classified into to feed data:700
Enter K :10
The new data belongs to  42
```

12. Aim: Implement K-Means algorithm

Program:

```
# K-Means
import numpy as np
import pandas as pd

data=pd.read_csv("Mall_Customers.csv")
data=data.drop(['CustomerID'],axis=1)
col=data.columns

def hashdata(df,data):
    for i in range(len(col)):
        if type(list(data[data.columns[i]])[0])==type('str'):
            nvalues=data[data.columns[i]].nunique()
            values=list(data[data.columns[i]].unique())
            l=range(nvalues)
            for j in range(nvalues):
                try:
                    df[data.columns[i]]=df[data.columns[i]].str.replace(values[j],str(l[j]))
                    df[data.columns[i]] = df[data.columns[i]].astype(int)
                except:
                    continue
    return df

df=data.copy()
df=hashdata(df,data)
k=int(input("Enter number of clusters you would like to divide data into:"))
clus=[]
cent=[]
for i in range(k):
    x=pd.Series(np.ravel(df.values[i:i+1]))
    clus.append(x)
    cent.append([x])

def return_cluster_index(k,clus,x):
    dis=[]
    for i in range(k):
        dis.append(sum((clus[i]-x)**2)**0.5)
    ind=dis.index(min(dis))
    return ind
def clusterise(x,clus,k,cent):
    ind=return_cluster_index(k,clus,x)
    clus[ind]=(clus[ind]*len(cent[ind])+x)/(len(cent[ind])+1)
    cent[ind].append(x)
    return clus,cent

for i in range(data[col[0]].count()):
    if(i<k):
```

```
        continue
    x=pd.Series(np.ravel(df.values[i:i+1]))
    clus,cent=clusterise(x,clus,k,cent)
print('Input format is as follows')
for i in df.columns:
    print(i,end=' ')
inp=input("\nEnter input:").split()
dic={}
inpdf=pd.DataFrame(inp)
inpdf2=inpdf.transpose()
for i in inpdf2.columns:
    dic[i]=col[i]
inpdf2.rename(columns = dic, inplace = True)
inpdf2=hashdata(inpdf2,data)
inp=pd.Series(np.ravel(inpdf2.values[:1].astype(int)))
ind=return_cluster_index(k,clus,inp)
print("The input belongs to the cluster",ind+1)
data
```

Output:

```
Enter number of clusters you would like to divide data into:2
Input format is as follows
Gender Age Annual Income (k$) Spending Score (1-100)
Enter input:Female 29 61 73
The input belongs to the cluster 2
```

13. Aim: Dynamic Indexing

Program:

```
#  Dynamic Indexing
import heapq
def GETNEXTTOKEN():
   global token_stream
   if len(token_stream) > 0:
      next_token = token_stream.pop(0)  # Get the next token from the stream
      return next_token
   else:
      return None  # Return None if the token stream is empty
def LMERGEADDTOKEN(ind, Z, token, n=10):
   Z[0].add(token)
   if len(Z[0]) == n:
      i = 0
      while True:
         if i<len(ind):
            if ind[i] != -1:
               Z.append(Z[i].union(ind[i]))
               ind[i] = -1
            else:
               ind[i] = Z[i]
         else:
            ind.append(Z[i])
            break
      Z = [set()]

def LOGARITHMICMERGE(docs):
   Z  = [set()]
   ind = []
   while True:
      token = GETNEXTTOKEN()
      if token is None:
         break
      post_list = []
      for j in range(len(docs)):
         if token in docs[j]:
            post_list.append(j+1)
      token = (token,tuple(post_list))
      LMERGEADDTOKEN(ind, Z, token)
   return(heapq.merge(*ind))
import re

docs = []
terms = set()
l = ['documents/'+str(i)+'.txt' for i in range(1,4)]

for i in l:
```

```
    file = open(i,'r').read().lower()
    content = re.sub(r"[^a-zA-Z0-9]"," ",file).split()
    terms.update(content)
    docs.append(set(content))

token_stream = list(terms)
print("constructed index : ")
for i in LOGARITHMICMERGE(docs):
    print(i)
```

Output:

```
constructed index :
('angel', (2,))
('his', (2,))
('told', (1,))
('oath', (3,))
('o', (1,))
('best', (1,))
('thine', (3,))
('know', (1, 2))
('vow', (3,))
('soon', (2,))
('fiend', (2,))
('vapour', (3,))
('broke', (3,))
('doth', (3,))
('be', (1, 2))
('vows', (3,))
('but', (1, 2, 3))
('both', (2,))
('cures', (3,))
('smiling', (1,))
('bad', (2,))
('that', (1, 2, 3))
('colour', (2,))
('fire', (2,))
...
('truth', (1, 2))
('me', (1, 2, 3))
('forgeries', (1,))
('her', (1, 2))
```