

BAB I PENDAHULUAN

1.1. Latar Belakang

Sistem Temu Kembali Informasi (*information retrieval system*) digunakan untuk menemukan kembali informasi-informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi. Selain itu *information retrieval* juga bisa didefinisikan sebagai ilmu pencarian informasi pada dokumen, bisa berupa pencarian untuk dokumen itu sendiri, untuk menjelaskan dokumen, atau mencari di dalam *database* berupa teks, suara, gambar atau data.

Kebutuhan akan suatu informasi dari berbagai jenis file menuntut seseorang untuk menelusuri seluruh koleksi dokumen yang dimilikinya. Sehingga pengguna mengalami kesulitan untuk memperoleh informasi yang dibutuhkan, karena tidak dapat melihat isi dokumen satu persatu.

Oleh sebab itu, maka perlu adanya penerapan *information retrieval system* yang diharapkan dapat menghasilkan pencarian dokumen yang relevan dan akurat sesuai kategorinya. Sehingga menghemat waktu dan mempercepat kinerja dalam pencarian dokumen sesuai dengan kata kunci yang dimasukkan.

Ada beberapa metode yang biasa diterapkan pada sebuah *information retrieval system* berbasis teks yakni metode *text preprocessing* (*tokenisasi, stopwords, stemming*) berfungsi untuk mencari kata dasar/kata kunci dari satu atau lebih dokumen dengan cara membersihkan sebuah kata dari tanda baca dan imbuhan, kemudian ada metode pembobotan *term* (contohnya bisa memakai metode *boolean, atau TF-IDF*) untuk mengetahui ada atau tidaknya suatu kata pada sebuah dokumen serta menghitung berapa bobot kemunculan pada dokumen, yang terakhir ada metode perankingan/pembobotan dokumen (contohnya metode *Vector Space Model (VSM)* atau *Weighted Tree Similarity (WTS)*).

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah penulis uraikan di atas, maka rumusan masalah yang akan dibahas dalam makalah ini adalah meliputi :

1. Bagaimana merancang sistem temu kembali atau *information retrieval system*.
2. Bagaimana implementasi *text preprocessing* (*filtering, tokenisasi, stopwords, stemming*) pada sebuah sistem temu kembali atau *information retrieval system*.
3. Bagaimana implementasi metode pencarian informasi (Metode *Boolean*) dan pembobotan *term/query* (Metode TF-IDF) pada sebuah sistem temu kembali atau *information retrieval system*.
4. Bagaimana implementasi metode pencarian similaritas (Metode *Vector Space Model (VSM)* dan *Weighted Tree Similarity (WTS)*) pada sebuah sistem temu kembali atau *information retrieval system*.

BAB II LANDASAN TEORI

2.1. Sistem Temu Kembali Informasi

Sistem temu kembali informasi (*information retrieval system*) merupakan sistem yang dapat digunakan untuk menemukan informasi yang relevan dengan kebutuhan dari penggunaanya secara otomatis dari suatu koleksi informasi (Mandala, 2002). Sistem temu kembali informasi pada dasarnya adalah suatu proses untuk mengidentifikasi, kemudian memanggil (*retrieval*) suatu dokumen dari suatu simpanan (*file*), sebagai jawaban atas permintaan informasi (Hasugian, 2003).

Query dalam *information retrieval* merupakan sebuah formula yang digunakan untuk mencari informasi yang dibutuhkan oleh pengguna, dalam bentuk yang paling sederhana. Sebuah *query* merupakan suatu *keywords* (kata kunci) dan dokumen yang mengandung *keywords* merupakan dokumen yang dicari dalam IRS.

Proses yang berlangsung dalam *information retrieval system* terdiri dari 2 bagian utama, yaitu *indexing subsystem*, dan *searching subsystem (matching system)*. Proses *indexing* dilakukan untuk membentuk basis data terhadap koleksi dokumen yang dimasukkan, atau dengan kata lain, *indexing* merupakan proses persiapan yang dilakukan terhadap dokumen sehingga dokumen siap untuk diproses. Proses *indexing* sendiri meliputi 2 proses, yaitu *document indexing* dan *term indexing*. Dari term indexing akan dihasilkan koleksi kata yang akan digunakan untuk meningkatkan performansi pencarian pada tahap selanjutnya.

Adapun tahap-tahap yang terjadi pada proses *indexing*, yaitu (Harjanto, 2012):

1. *Tokenizing* dokumen, yaitu proses mengubah dokumen menjadi kumpulan *term* dengan cara menghapus semua karakter tanda baca yang terdapat pada *token*. Hingga pada akhirnya yang diperoleh hanya kumpulan kata-kata dari suatu teks/dokumen.
2. *Stopword removal* dokumen, yaitu kata-kata yang sering muncul dalam dokumen namun artinya tidak deskriptif dan tidak memiliki keterkaitan dengan tema tertentu. Pada Bahasa Indonesia, *stopword* disebut juga

sebagai kata yang tidak penting, misalnya “di”, “oleh”, “pada”, “sebuah”, “karena” dan lain sebagainya.

3. *Stemming* dokumen, yaitu tahap penghilangan imbuhan sehingga didapatkan kata dasar dari *term-term* dokumen inputan.
4. *Term Weighting*, yaitu proses pembobotan pada setiap *term* (kata) yang ada didalam dokumen.

2.2. Tujuan Sistem Temu Kembali Informasi

Sistem Temu Kembali Informasi bertujuan untuk menjembatani kebutuhan informasi pengguna dengan sumber informasi yang tersedia dalam situasi seperti dikemukakan oleh Belkin (1980) sebagai berikut:

1. Penulis mempresentasikan sekumpulan ide dalam sebuah dokumen menggunakan sekumpulan konsep.
2. Terdapat beberapa pengguna yang memerlukan ide yang dikemukakan oleh penulis tersebut, tapi mereka tidak dapat mengidentifikasi dan menemukannya dengan baik.
3. Sistem temu kembali informasi bertujuan untuk mempertemukan ide yang dikemukakan oleh penulis dalam dokumen dengan kebutuhan informasi pengguna yang dinyatakan dalam bentuk pertanyaan (*query*).

2.3. Boolean Model

Model ini merupakan suatu sistem temu kembali informasi (informasi retrieval) pertama kali yang digunakan untuk mempresentasikan suatu dokumen, diidentikkan dengan himpunan yang berfungsi sebagai kata kunci dalam pencarian informasi, sedangkan Boolean dalam presentasi, secara ekspresi bertindak sebagai query. Ekspresi Boolean terdiri dari berbagai kumpulan kata kunci yang menggunakan berbagai operator seperti OR, NOT, dan AND yang digunakan untuk mendiskripsikan berbagai informasi yang akan dicari dalam suatu dokumen. Model yang digunakan oleh Boolean dalam melakukan penelusuran ini dengan menggunakan teori logika matematika, sehingga juga disebut logika Boolean yang dikembangkan oleh seorang

matematikus Roger Boolean pada tahun 1894, sehingga juga dikenal dengan aljabar Boolean. Teori ini tidak bersifat statis dan selalu dikembangkan sesuai dengan perkembangan teknologi informasi. Dengan perkembangan teori model Boolean tersebut menurut para ahli dikelompokkan menjadi :

- Teknik Boolean sederhana.
- Teknik Boolean berperingkat
- Teknik Extended Boolean

Yang dimana pada saat ini penulis menerapkan Teknik Boolean sederhana, yaitu :

Pada model logika Boolean sederhana ini ketika akan melakukan penelusuran terhadap sejumlah dokumen diantara sejumlah berkas hanya akan didapatkan dua ukuran true,false atau 0,1. Penelusuran model Boolean dasar teori logika aljabar yang digunakan adalah pertambahan (logical sum) dengan simbol (+) yang diungkapkan dengan kata OR, simbol logika pengurangan (logical difference) dengan simbol (-) yang diungkapkan dengan kata NOT dan logika perkalian (logical product) dengan menggunakan kata AND. Ketiga kata logika tersebut dapat digunakan untuk menghubungkan diskriptor yang terdapat dalam suatu dokumen dengan cara sebagai berikut :

- AND (logical product)

Memperbolehkan penelusur untuk menggunakan pernyataan query ke dalam suatu lebih konsep sehingga hasil penelusuran menjadi lebih terbatas. Formula pernyataan sederhana $A \text{ AND } B$. Contoh untuk menelusur marketing and library, kita memformulasikan pernyataan dengan marketing AND library. Dengan query tersebut maka kita akan menemukan dokumen yang mengandung unsur marketing dan perpustakaan saja, dan tidak mendapatkan dokumen yang hanya mengandung unsur marketingatau perpustakaan saja.

- OR (logical sum)

Memperbolehkan untuk secara spesifik menggunakan alternative term (atau konsep) yang mengindikasikan dua konsep sesuai dengan tujuan penelusuran menjadi lebih luas, karena adanya alternative dalam pernyataan query. Formulasi pernyataan sederhana A OR B. Contoh marketing OR library, dengan query tersebut maka kita akan mencari salah satu query antara marketing atau library.

- NOT (logical difference)

Dapat mengecualikan item-item dari seperangkat term penelusur. Pernyataan formulasi sederhana A NOT B, contoh marketing NOT library. Ini artinya kita hanya menginginkan dokumen yang unsur marketing di dalamnya tidak ada unsur perpustakaan. Artinya NOT mengartikan query yang tidak diinginkan.

2.4. TF-IDF

Metode TF-IDF merupakan metode untuk menghitung bobot setiap kata yang paling umum digunakan pada information retrieval. Metode ini juga terkenal efisien, mudah dan memiliki hasil yang akurat. Metode Term Frequency-Inverse Document Frequency (TF-IDF) adalah cara pemberian bobot hubungan suatu kata (term) terhadap dokumen. TF-IDF ini adalah sebuah ukuran statistik yang digunakan untuk mengevaluasi seberapa penting sebuah kata di dalam sebuah dokumen atau dalam sekelompok kata. Untuk dokumen tunggal tiap kalimat dianggap sebagai dokumen. Frekuensi kemunculan kata di dalam dokumen yang diberikan menunjukkan seberapa penting kata itu di dalam dokumen tersebut. Frekuensi dokumen yang mengandung kata tersebut menunjukkan seberapa umum kata tersebut. Bobot kata semakin besar jika sering muncul dalam suatu dokumen dan semakin kecil jika muncul dalam banyak dokumen.

- TF(Term Frequency)

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{if } tf_{t,d} = 0 \end{cases}$$

Jadi jika suatu term terdapat dalam suatu dokumen sebanyak 5 kali maka diperoleh bobot $= 1 + \log(5) = 1.699$. Tetapi jika term tidak terdapat dalam dokumen tersebut, bobotnya adalah nol.

- IDF(Inverse Document Frequency)

Terkadang suatu term muncul di hampir sebagian besar dokumen mengakibatkan proses pencarian term unik terganggu. Idf berfungsi mengurangi bobot suatu term jika kemunculannya banyak tersebar di seluruh koleksi dokumen kita.

$$idf_t = \log_{10} (N/df_t)$$

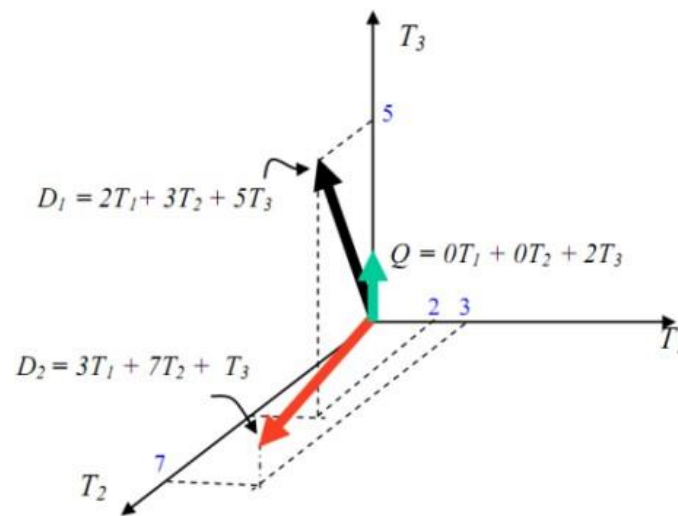
- TF-IDF

Pada algoritma TF-IDF digunakan rumus untuk menghitung bobot (W) yaitu :

$$W = TF * IDF$$

2.5. VSM (Vector Space Model)

Vector Space Model (VSM) merupakan model yang digunakan untuk mengukur kemiripan antara dokumen dan query yang mewakili setiap dokumen dalam sebuah koleksi sebagai sebuah titik dalam ruang. Dalam metode Vector Space Model dihitung weighted dari setiap term yang terdapat dalam semua dokumen dan query dari user. Term adalah kata atau kumpulan kata yang merupakan ekspresi verbal dari suatu pengertian. Penentuan relevansi dokumen dengan query dipandang sebagai pengukuran kesamaan antara vektor dokumen dengan vektor query. Contoh representasi relevansi antara dokumen dan query dapat digambarkan pada Gambar 1. Q merupakan query pembanding, D1 dan D2 adalah dua dokumen yang akan dibandingkan, sedangkan T1, T2 dan T3 adalah tiga term pada dokumen tersebut.



Gambar 1. Representasi dokumen dan *query* pada ruang vektor [8]

Perhitungan kemiripan antara vektor dokumen dan vektor query dilihat dari sudut yang paling kecil. Sudut yang dibentuk oleh dua buah vektor dapat dihitung dengan melakukan inner product. Kemiripan antara vektor dokumen dan vektor query akan dihitung dengan pendekatan cosine similarity. Perumusan cosine similarity adalah sebagai berikut :

$$\begin{aligned} Sim(D, D_i) &= \cos \theta \\ &= \frac{D \cdot D_i}{|D||D_i|} = \frac{\sum W_{q,j} W_{i,j}}{\sqrt{\sum W_{q,j}^2} \sqrt{\sum W_{i,j}^2}} \end{aligned}$$

Dimana:

- D = dokumen acuan
- D_i = dokumen ke-i
- $W_{q,j}$ = bobot *term* j pada dokumen acuan
- $W_{i,j}$ = bobot *term* j pada dokumen i

2.6. Weighted Tree Similarity (WTS)

Weighted Tree Similarity merupakan salah satu algoritma untuk menentukan tingkat kesamaan dua objek yang disajikan dalam bentuk tree. Pada Weighted Tree Similarity dokumen yang sudah melalui proses pre processing direpresentasikan kedalam bentuk tree. Tree dalam Weighted Tree memiliki konsep node atau titik yang berlabel, arc atau cabang berlabel, dan arc berbobot. Proses pembobotan pada tree dilakukan dengan memperhatikan TF(Term Frequency) dalam sebuah dokumen. Perhitungan pembobotan TF dilakukan dengan persamaan berikut :

$$W = TF/TF_{total}$$

Dimana :

TF = Frekuensi suatu kata

TF_{total} = total frekuensi pada seluruh kata

Dalam perhitungan similarity pada Weighted Tree Similarity, bobot pada semua bobot cabang berpengaruh pada kemiripan total. Perhitungan dilakukan dengan menghitung rata-rata persamaan aritmatika. Perumusan perhitungan kemiripan tree terdapat dalam persamaan diatas, yaitu sebagai berikut :

$$Sim_{tot} = \sum(S_i * W_i) \quad (5)$$

Dimana

w_i = bobot cabang *tree* pertama

S_i = kemiripan cabang

Nilai rata-rata bobot hasil perhitungan dikalikan dengan nilai kemiripan cabang yang dihasilkan dari perhitungan similarity. Perhitungan *Weighted Tree Similarity* dimulai dengan memberi nilai 1 jika *leafnode*nya sama dan diberi nilai 0 jika *leafnode*nya berbeda.

Langkah – langkah perhitungan pada Weighted Tree Similarity adalah sebagai berikut :

1. . Lakukan pembobotan TF pada masing – masing Q dan P pada masing masing dokumen, misalkan pada kasus ini seperti gambar berikut :

Parameter Penyakit					Parameter Penyebab				
Term	TF		Pembobotan TF		Term	TF		Pembobotan TF	
	Q	P	Q	P		Q	P	Q	P
gusi	1	0	0,33333	0	gusi	1	1	0,3333	0,3333
darah	1	0	0,33333	0	darah	1	0	0,3333	0
bengkak	1	0	0,33333	0	bengkak	1	0	0,3333	0
ginggivitis	0	1	0	1	karang	0	1	0	0,3333
Total Frekuensi	3	1			plak	0	1	0	0,3333
					Total Frekuensi	3	3		

Parameter Klinis					Parameter Definisi				
Term	TF		Pembobotan TF		Term	TF		Pembobotan TF	
	Q	P	Q	P		Q	P	Q	P
gusi	1	1	0,3333	0,1667	gusi	1	1	0,3333	0,3333
darah	1	1	0,3333	0,1667	darah	1	0	0,3333	0
bengkak	1	1	0,3333	0,1667	bengkak	1	0	0,3333	0
nyeri	0	1	0	0,1667	radang	0	1	0	0,3333
bau	0	1	0	0,1667	ginggivitis	0	1	0	0,3333
mulut	0	1	0	0,1667	Total Frekuensi	3	3		
Total Frekuensi	3	6							

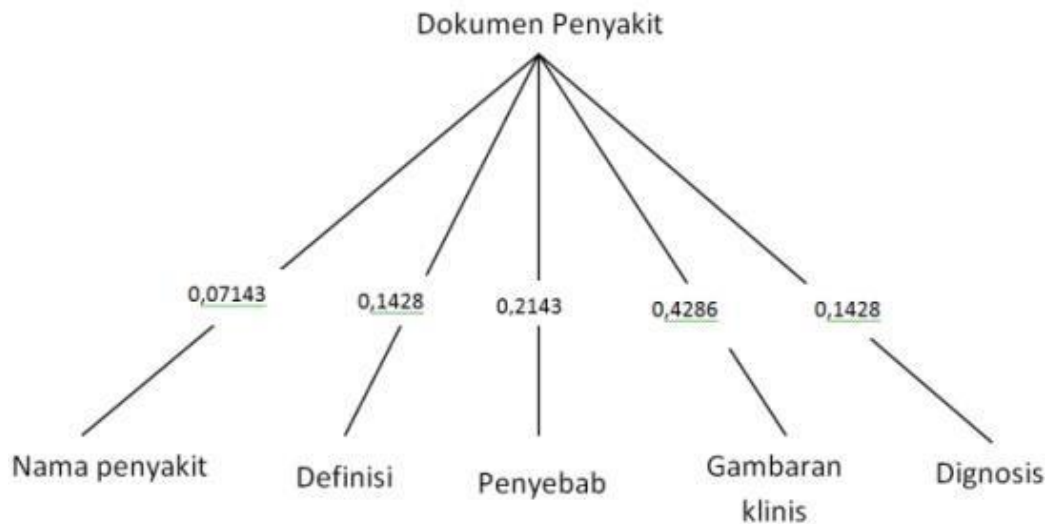
Parameter Diagnosis				
Term	TF		Pembobotan TF	
	Q	P	Q	P
gusi	1	1	0,3333	0,1667
darah	1	1	0,3333	0,1667
bengkak	1	1	0,3333	0,1667
nyeri	0	1	0	0,1667
Total Frekuensi	3	3		

2. Setelah dihitung bobot term pada semua parameter, langkah selanjutnya adalah menghitung nilai similarity pada tiap parameter. Nilai similarity dihitung dengan metode Cosine Similarity berdasarkan bobot term yang sudah dihitung sebelumnya :

Similarity Nama Penyakit (Q,P)	Similarity Penyebab (Q,P)
$= \frac{\sum(Bobot_Q * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$ $= \frac{0}{0,57743 * 1}$ $= 0$	$= \frac{\sum(Bobot_Q * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$ $= \frac{0,01111}{0,57735 * 0,47141}$ $= 0,40825$

Similarity Klinis (Q,P) $= \frac{\sum(Bobot_Q * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$ $= \frac{0,16667}{0,57735 * 0,40285}$ $= 0,70711$	Similarity Definisi (Q,P) $= \frac{\sum(Bobot_Q * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$ $= \frac{0,01111}{0,57735 * 0,47141}$ $= 0,40825$
Similarity Diagnosis (Q,P) $= \frac{\sum(Bobot_Q * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$ $= \frac{0,01111}{0,57735 * 0,47141}$ $= 0,40825$	

3. Kita petakan bobot parameter pada masing masing tree. Bobot parameter didapatkan dari nilai total term frekuensi tiap parameter dibagi total frekuensi term keseluruhan dokumen :



4. Setelah didapatkan nilai bobot pada tiap node parameter, selanjutnya dihitung nilai similarity total. Langkahnya adalah dengan menghitung nilai total similarity pada masing – masing parameter kemudian dikalikan dengan nilai bobot tiap node parameter. Sehingga penghitungan nilai similarity total dengan metode Weighted Tree Similarity didapatkan sebagai berikut :

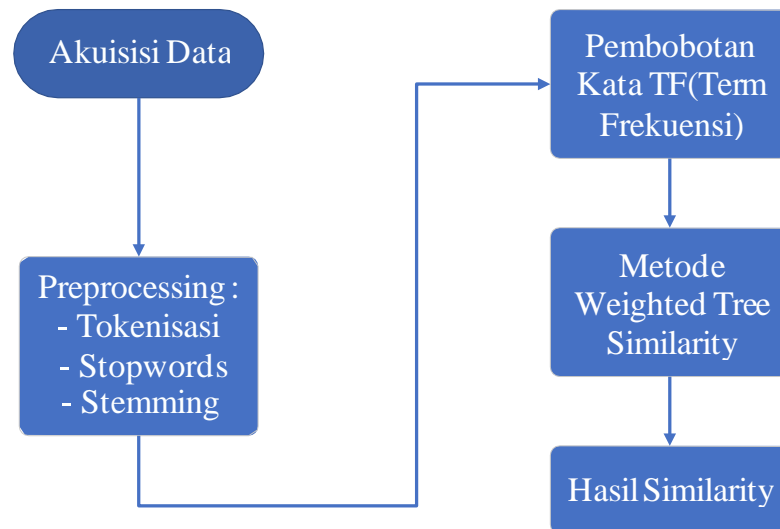
$$Sim_{tot} = \sum(S_i * W_i)$$

Similarity Total

$$\begin{aligned}
 &= (0*0,7143) + (0,40825*0,1428) + \\
 &\quad (0,40825*0,2143) + (0,70711*0,4286) + \\
 &\quad (0,40825*0,1428) \\
 &= \mathbf{0,50715}
 \end{aligned}$$

BAB III PEMBAHASAN SISTEM

3.1. Workflow Sistem



Pada sistem kami terdapat 5 alur kerja (*WorkFlow*) :

1. Akuisisi Data

Pada Proses ini kami mencari file sumber berupa txt, dilanjutkan dengan studi kasus dan pembelajaran pemrograman Python untuk nantinya bisa melakukan implementasi dan uji coba.

2. Preprocessing

Pada proses ini kami melakukan studi dan implementasi tokenisasi, stopwords dan stemming untuk mencari kata kunci/kata dasar dari *term* satu atau lebih dokumen.

3. Pembobotan Kata TF (TermFrequency)

Setelah melewati proses *Preprocessing* dan mendapatkan kata dasar, kami melanjutkan pada proses Pembobotan kata pada tiap dokumen untuk mengetahui seberapa unik atau seberapa penting kata tersebut sesuai *query* yang diminta *user*. Disini kami mengimplementasikan metode pembobotan kata TF-IDF dan VSM.

4. Implementasi Metode Weighted Tree Similarity (WTS)

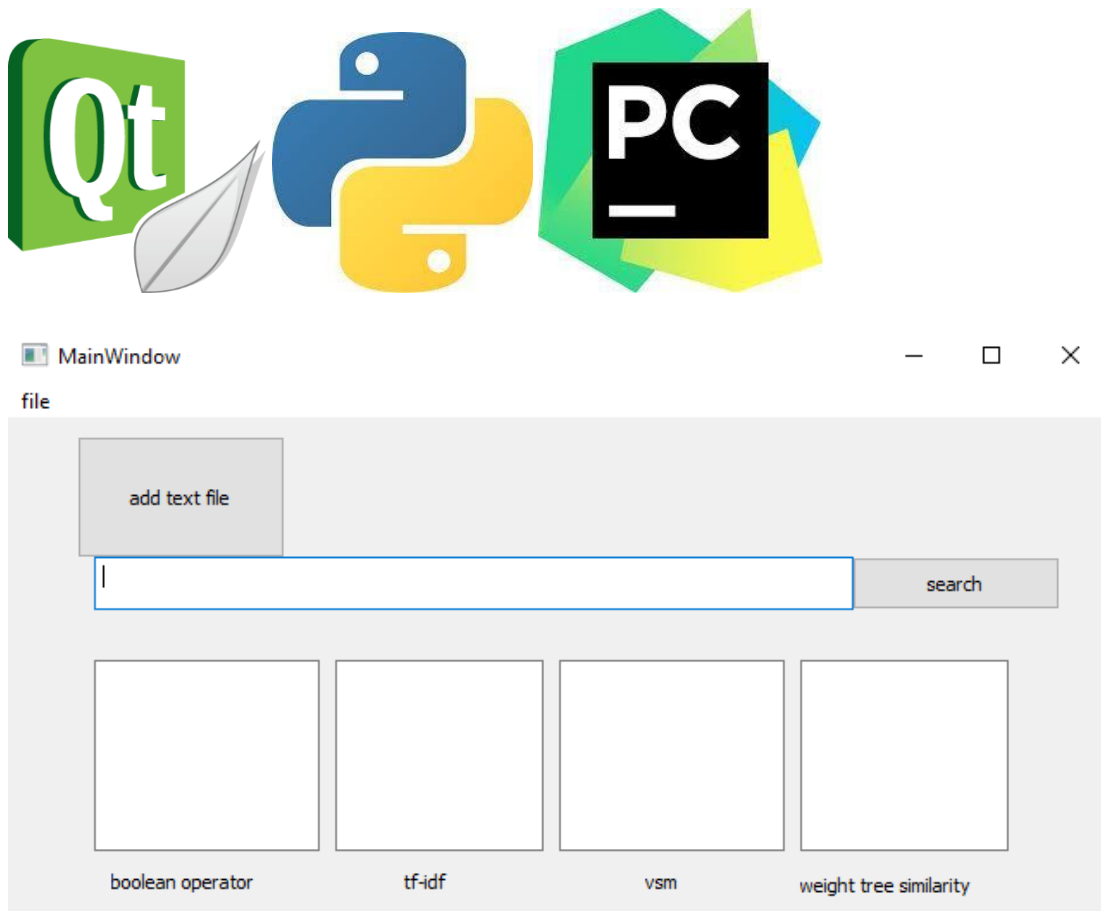
Pada tahap ini kami mencoba mengimplementasi metode pembobotan dan similaritas/kemiripan dokumen dengan metode WTS. Hal ini dilakukan untuk menguji perbandingan dengan metode VSM.

5. Mengamati Hasil Similarity

Pada tahap terakhir kami bisa menyimpulkan diantara kedua metode pembobotan dan similaritas antara WTS dan VSM, manakan diantara keduanya yang merupakan metode terbaik.

3.2. Kebutuhan Sistem dan Tampilan Antarmuka

Implementasi GUI pada sistem kamu menggunakan bantuan aplikasi QtDesigner yang nantinya bisa dihubungkan dengan backend python menggunakan library PyQt5. IDE yang kami gunakan adalah JetBrains Pycharm.



Gambar diatas adalah hasil GUI yang kami desain di dalam aplikasi QtDesigner.

3.3. Penjelasan Koding

3.3.1. Library

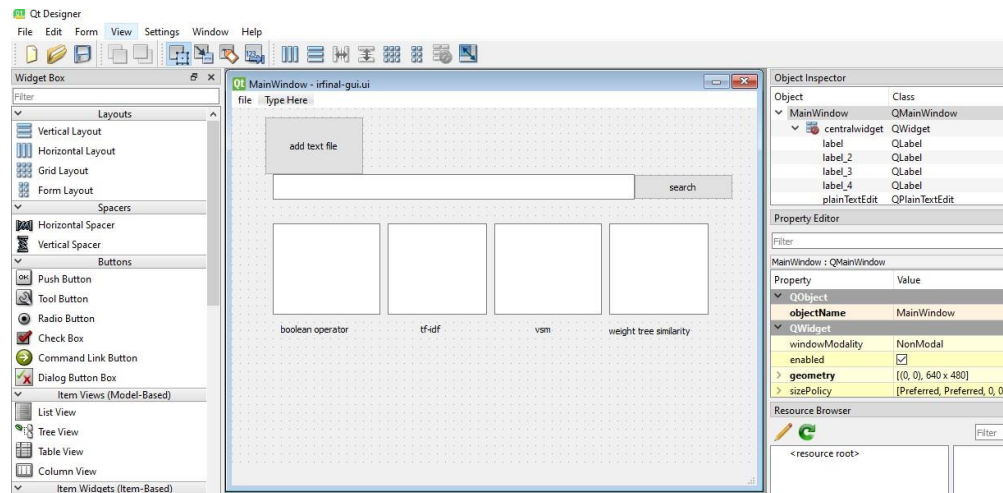
```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog, QApplication, QDialog
import string
import math
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

Penjelasan Kode Program Diatas :

1. Import Modul Library PyQt5

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog, QApplication, Qdialog
```

Penjelasan : PyQt5 adalah modul library yang digunakan untuk memfasilitasi pemanggilan platform GUI perangkat Qt (toolkit untuk pengembangan aplikasi grafis yang bersifat lintas-platform.) untuk bahasa pemrograman Python. Karena GUI yang kami implementasikan menggunakan Aplikasi QtDesigner sebagai berikut :



Sehingga untuk dapat menghubungkan desain Frontend GUI diatas dengan backend pemrograman Python, kami memerlukan Library **PyQt5**. Library ini memiliki properti seperti `QtCore`, `QtGui`, `QtWidgets` yang bisa dipanggil sebagai object class disetiap pemanggilan properti GUI seperti *Button*,

Plaintext, dan *label*. Pada modul properti `QtWidgets` kami juga memanggil properti lain seperti `QFileDialog`, `QApplication`, `QDialog` yang berguna untuk memanggil objek *opendialog* saat *user* melakukan input data.

2. Import Modul Library *String* dan *Math*

```
import string
import math
```

Penjelasan : Import Modul *String* digunakan untuk mendefinisikan string seperti tanda baca koma, titik, tanda tanya/seru atau lebih dikenal (*punctuation*), huruf kecil (*lowercase*), huruf besar (*uppercase*), simbol-simbol *ascii*, dsb. Dan Import Modul *Math* digunakan untuk penggunaan operator aritmetika seperti kali (*x*), bagi (*/*), tambah (*+*), kurang, pangkat (*pow*), dan akar (*sqrt*).

3. Import Modul Library NLTK (*Natural Language Processing*)

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

Penjelasan : Natural Language Toolkit (**NLTK**) adalah modul pada bahasa pemrograman python yang digunakan untuk membangun program analisis teks. Pada proyek ini kami menggunakan properti *corpus* untuk digunakan pada metode *stopwords* dan properti *stem* untuk digunakan pada metode *Stemming*.

3.3.2. Fungsi

3.3.2.1. Fungsi SetupUi

Fungsi ini adalah markup atau setup GUI yang diadaptasi dari library PyQt5.

```
def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(640, 480)
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.plainTextEdit =
    QtWidgets.QPlainTextEdit(self.centralwidget)
    self.plainTextEdit.setGeometry(QtCore.QRect(50, 80, 441, 31))
    self.plainTextEdit.setObjectName("plainTextEdit")
    self.plainTextEdit_2 =
    QtWidgets.QPlainTextEdit(self.centralwidget)
    self.plainTextEdit_2.setGeometry(QtCore.QRect(50, 140, 131,
111))
```



```

        self.plainTextEdit_2.setObjectName("plainTextEdit_2")
        self.plainTextEdit_3 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_3.setGeometry(QtCore.QRect(190, 140, 121,
111))
        self.plainTextEdit_3.setObjectName("plainTextEdit_3")
        self.plainTextEdit_4 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_4.setGeometry(QtCore.QRect(320, 140, 131,
111))
        self.plainTextEdit_4.setObjectName("plainTextEdit_4")
        self.plainTextEdit_5 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_5.setGeometry(QtCore.QRect(460, 140, 121,
111))
        self.plainTextEdit_5.setObjectName("plainTextEdit_5")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(60, 260, 111, 16))
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(230, 260, 60, 16))
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(370, 260, 60, 16))
        self.label_3.setObjectName("label_3")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(460, 260, 131, 20))
        self.label_4.setObjectName("label_4")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(490, 80, 121, 31))
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(40, 10, 121, 71))
        self.pushButton_2.setObjectName("pushButton_2")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 640, 22))
        self.menubar.setObjectName("menubar")
        self.menufile = QtWidgets.QMenu(self.menubar)
        self.menufile.setObjectName("menufile")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.actionnew = QtWidgets.QAction(MainWindow)
        self.actionnew.setObjectName("actionnew")
        self.actionexit = QtWidgets.QAction(MainWindow)
        self.actionexit.setObjectName("actionexit")
        self.menufile.addAction(self.actionnew)
        self.menufile.addSeparator()
        self.menufile.addAction(self.actionexit)
        self.menubar.addAction(self.menufile.menuAction())

listFile = self.pushButton_2.clicked.connect(self.GetFiles)

```

```

self.pushButton.clicked.connect(self.queryClicked)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)

```

3.3.2.2. Fungsi retranslateUi

Fungsi ini digunakan untuk mengubah fungsi *setupUI* sebelumnya (diatas) menjadi sebuah objek di python.

```

def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow",
    "MainWindow"))
    self.label.setText(_translate("MainWindow", "boolean
operator"))
    self.label_2.setText(_translate("MainWindow", "tf-idf"))
    self.label_3.setText(_translate("MainWindow", "vsm"))
    self.label_4.setText(_translate("MainWindow", "weight tree
similarity"))
    self.pushButton.setText(_translate("MainWindow", "search"))
    self.pushButton_2.setText(_translate("MainWindow", "add text
file"))
    self.menufile.setTitle(_translate("MainWindow", "file"))
    self.actionnew.setText(_translate("MainWindow", "new"))
    self.actionexit.setText(_translate("MainWindow", "exit"))

```

3.3.2.3. Fungsi init

Fungsi init digunakan untuk mendefinisikan parameter yang akan digunakan pada fungsi operasi metode boolean, TF-IDF, dan VSM.

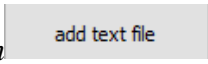
```

def __init__(self, listFile, vocab, hasilTF, documentFreq, tfidf,
finalTfidf, rankTfidf, booleanRes,
activeOperator):
    self.listFile = list()
    self.boolOperator = ['AND', 'OR', 'NOT']
    # vocab = self.checkVocab(vocab, listFile)
    self.activeOperator = activeOperator

```

3.3.2.4. Fungsi GetFiles

Fungsi ini berfungsi untuk mendapatkan informasi input dari *user*. Ketika *user*

mengakses *button* , nama *button* tersebut adalah *pushbutton2* mengacu pada kode program `listFile = self.pushButton_2.clicked.connect(self.GetFiles)` pada fungsi *SetupUi* sebelumnya, maka program akan memanggil Fungsi *GetFiles*.

```
def getFiles(self):
    fname = QFileDialog.getOpenFileNames()
    for r in range(0, len(fname)-1):
        for j in fname[r]:
            myFile = open(j).readlines()
            for m in myFile:
                m.translate(str.maketrans(' ', '',
string.punctuation)).lower()
                tokens = nltk.word_tokenize(m)
                # listFile.append(tokens)

                # filtering stopwords -> mengubah menjadi kata
dasar
                filteredWords = list()
                stop_words = set(stopwords.words('english'))
                for t in tokens:
                    if t not in stop_words:
                        filteredWords.append(t)
                        # stemming -> merubah kata yang tidak
penting
                        ps = PorterStemmer()
                        output = list()
                        for f in filteredWords:
                            output.append(ps.stem(f))
                            # memasukkan data yang bersih ke dalam list
                        self.listFile.append(output)
                # print(self.listFile)
                # mencetak isi tiap tiap dokumen yang sudah bersih:
                c = 1
                for s in self.listFile:
                    print('Dokumen {0} : {1}'.format(c, s))
                    c+=1
```

Penjelasan Kode Program Diatas :

Untuk menampung file yang dipilih *user* sebagai input, maka file akan disimpan/diinisialisasikan sebagai variabel *fname* yang akan diteruskan ke `QFileDialog.getOpenFileNames()`.

```
def getFiles(self):
    fname = QFileDialog.getOpenFileNames()
```

Setelahnya program akan melakukan perulangan untuk membaca/mendeteksi file yang telah dipilih dengan basis *r*.

```
for r in range(0, len(fname)-1):
```

Setelah semua file yang diseleksi *user* terbaca/terdeteksi, program akan melakukan perulangan kembali untuk mendapatkan akses pembacaan isi data dimasing-masing file dengan fungsi *readlines()* yang kemudian disimpan sebagai basis *j* pada variabel *myFile*. Data akan disimpan dalam bentuk list.

```
for j in fname[r]:
    myFile = open(j).readlines()
```

Setelah program mendapatkan data dari masing-masing dokumen, maka isi data akan ditranslasi untuk dihilangkan tanda bacanya dan mengubahnya kedalam huruf kecil (*lowercase*). Data disimpan sebagai variabel *m*.

```
for m in myFile:
    m.translate(str.maketrans(" ", string.punctuation)).lower()
```

Setelah mendapatkan kumpulan string dari isi data tiap dokumen yang sudah dihilangkan tanda bacanya, maka program akan melakukan proses tokenisasi, stopwords dan stemming dengan kode dibawah ini. Setelah program mendapatkan isi data yang sudah bersih (setelah melalui proses *text preprocessing*), maka program akan menampung dari properti *listfile* yang akan ditambahkan dengan variabel output.

```
tokens = nltk.word_tokenize(m)
    # listFile.append(tokens)
    # filtering stopwords -> mengubah menjadi kata dasar
    filteredWords = list()
    stop_words = set(stopwords.words('english'))
```

```

for t in tokens:
    if t not in stop_words:
        filteredWords.append(t)
        # stemming -> merubah kata yang tidak penting
        ps = PorterStemmer()
        output = list()
        for f in filteredWords:
            output.append(ps.stem(f))
            # memasukkan data yang bersih ke dalam list
self.listFile.append(output)

```

Dengan kode program dibawah ini, hasil output akan ditampilkan pada terminal

```

c = 1
for s in self.listFile:
    print('Dokumen {0} : {1}'.format(c, s))
    c+=1

```

Hasil output di terminal akan seperti ini :

```

Dokumen 1 : ['breakthrough', 'drug', 'schizophrenia']
Dokumen 2 : ['new', 'schizophrenia', 'drug']
Dokumen 3 : ['new', 'approach', 'treatment', 'schizophrenia']
Dokumen 4 : ['new', 'hope', 'schizophrenia', 'patient']

```

3.3.2.5. Fungsi queryClicked

Fungsi ini akan dieksekusi jika *user* menekan *button search* seperti dibawah ini :



berdasarkan kode program `self.pushButton.clicked.connect(self.queryClicked)`, *search* adalah bagian dari *pushbutton*, maka program akan memanggil fungsi *queryClicked*.

```

def queryClicked(self):
    userInput = self.plainTextEdit.toPlainText()
    userInput = userInput.split()

```

```

self.boolOps (userInput)
self.tfidfOps (userInput)
self.vsmOps (userInput)
self.wtsOps (userInput)

```

Penjelasan Kode Program Diatas :

Pada fungsi ini program pertama kali akan mengambil nilai dari variabel *userInput* yang nilainya merupakan sekumpulan string yang diinputkan melalui *plaintext*.

```

def queryClicked(self):
    userInput = self.plainTextEdit.toPlainText()

```

Setelah program mendapatkan *userInput*, maka program akan melakukan split terhadap string atau query yang diinputkan pada *PlainText*. Masing-Masing *Query/String* akan diberi tanda koma, seperti contoh :

Misal *User* menginputkan *query* “*drug AND new*”, maka setelah di split akan berubah menjadi “*drug, AND, new*”

```

userInput = self.plainTextEdit.toPlainText()

```

Setelah dilakukan split, maka program akan membawa parameter variabel *userInput* ke seluruh fungsi operasi yang ada seperti *boolean*, *tfIdf*, *VSM* dan *WTS*.

```

self.boolOps(userInput)
self.tfidfOps(userInput)
self.vsmOps(userInput)
self.wtsOps(userInput)

```

3.3.2.6. Fungsi boolOps

Fungsi ini digunakan untuk operasi boolean yang menentukan ada tidaknya suatu *query* pada dokumen.

```
def boolOps(self, userInput):
    for bi in userInput:
        if bi in boolOperator:
            userInput.remove(bi)
            self.activeOperator += bi
    print(userInput)
    print(self.activeOperator)

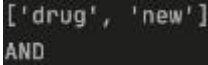
    for bl in userInput:
        for i in self.listFiles:
            if bl not in i:
                booleanRes.append(0)
            else:
                booleanRes.append(1)
    print(booleanRes)
    # membagi list agar tiap2 elemen ada 4
    composite_list = [booleanRes[x:x + 4] for x in range(0,
len(booleanRes), 4)]
    print(composite_list)

    # hitung jika boolean operator adalah AND atau OR
    if self.activeOperator == 'AND':
        flag = '1'
        for c in range(0, len(composite_list)):
            myBinary = ''
            for l in range(0, len(composite_list[c])):
                myBinary = myBinary + str(composite_list[c][l])
            flag = bin(int(myBinary, 2) * int(flag, 2))
            print('hasil boolean : {}'.format(flag))
            self.plainTextEdit_2.insertPlainText('hasil boolean :
{}'.format(flag))
        elif self.activeOperator == 'OR':
            flag = '0'
            for c in range(0, len(composite_list)):
                myBinary = ''
                for l in range(0, len(composite_list[c])):
                    myBinary = myBinary + str(composite_list[c][l])
                flag = bin(int(myBinary, 2) + int(flag, 2))
            print('hasil boolean : {}'.format(flag))
            self.plainTextEdit_2.insertPlainText('hasil boolean :
{}'.format(flag))
```

Penjelasan Kode Program Diatas :

```
(1) def boolOps(self, userInput):
(2)     for bi in userInput:
(3)         if bi in boolOperator:
                userInput.remove(bi)
                self.activeOperator += bi
(4)     print(userInput)
        print(self.activeOperator)
```

1. Pada Fungsi *Boolean Operations* ini kami melewati/*passing* dua parameter untuk diproses, Parameter pertama adalah *Self* konsep OOP seperti *this* pada Java, *Self* adalah object yang memanggil modul GUI. Sedangkan parameter *userInput* membawa hasil query yang telah di split di fungsi sebelumnya.
2. Perulangan *For* yang pertama adalah untuk mengeksekusi pembacaan *query* yang dibawa variabel *userInput*.
3. Kondisi *if* pada perulangan *For* bertujuan untuk menyeleksi operator *boolean* seperti *AND*, *OR* atau *NOT* yang diinputkan bersama *query*. Selanjutnya dengan kode program `userInput.remove(bi)` operator tersebut akan dibuang untuk dipisahkan dengan query. Operator yang dipisahkan akan diubah menjadi *operator active* dengan kode program `self.activeOperator += bi`.
4. Kemudian *userInput* dan *activeOperator* akan ditampilkan pada terminal seperti

berikut ini: 

```
(5) for bl in userInput:
(6)     for i in self.listFiles:
(7)         if bl not in i:
                booleanRes.append(0)
            else:
                booleanRes.append(1)
        print(booleanRes)
```



```
(8) composite_list = [booleanRes[x:x+4]
    for x in range(0, len(booleanRes), 4)]
    print(composite_list)
```

5. Perulangan *for* dengan basis *bl* akan mengeksekusi isi data/*query* yang dibawa variabel *userInput*.
6. Perulangan *for* yang kedua dengan basis *i* akan mengeksekusi isi data/*query* yang dibawa variabel *listfile*. Berikut adalah isi *query* yang dibawa variabel *listFile* yang sudah ditampilkan di terminal :

```
Dokumen 1 : ['breakthrough', 'drug', 'schizophrenia']
Dokumen 2 : ['new', 'schizophrenia', 'drug']
Dokumen 3 : ['new', 'approach', 'treatment', 'schizophrenia']
Dokumen 4 : ['new', 'hope', 'schizophrenia', 'patient']
```

7. Isi data/*query* yang dibawa variabel *inputUser* akan diseleksi dengan kode program kondisi *if*, apakah *query* yang ada pada *userInput* tidak tercantum pada daftar *query listFile*, jika ya yang berarti tidak ada, maka akan ditambah/diberi/*append* nilai 0, jika tidak yang berarti ada, maka akan diberi nilai 1. Hasilnya akan diprint yang dibawa dengan variabel *booleanRes* dan ditampilkan di terminal seperti gambar dibawah :

```
[1, 1, 0, 0, 0, 1, 1, 1]
```

Hal ini menunjukkan ada tidaknya suatu *term/query/kata/istilah* yang diminta *user* terhadap isi dokumen. Jika ada akan bernilai 1, sebaliknya jika *query* tersebut tidak ada pada dokumen yang dipilih, maka akan bernilai 0.

8. Kode program yang kedelapan akan merapikan anggota output list array menjadi 4 set, seperti gambar dibawah :

```
[1, 1, 0, 0, 0, 1, 1, 1]
[[1, 1, 0, 0], [0, 1, 1, 1]]
```

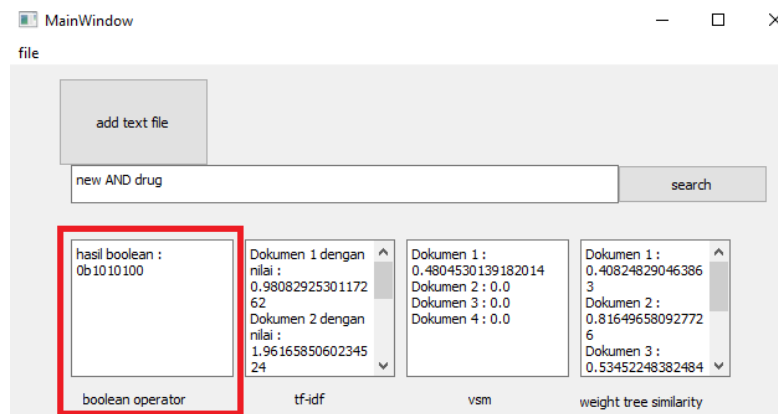
Misalkan dalam kasus ini *user* menginputkan *drug AND new*, maka List pertama *query drug* [1,1,0,0] yang artinya kata “drug” ada di dokumen 1 dan 2, lalu List kedua *query new* [0,1,1,1] yang artinya kata “new” ada di dokumen 2,3, dan 4.

```

(9) if self.activeOperator == 'AND':
    flag = '1'
(10) for c in range(0, len(composite_list)):
    myBinary = ""
    for l in range(0, len(composite_list[c])):
        myBinary = myBinary + str(composite_list[c][l])
(11) flag = bin(int(myBinary, 2) * int(flag, 2))
(12) print('hasil boolean : {}'.format(flag))
self.plainTextEdit_2.insertPlainText('hasil boolean : {}'.format(flag))

```

9. Jika *user* memilih operator *AND*, maka program akan mengeksekusi kondisi *if* yang membawa *activeOperator AND* yang diberi label/*flag* 1.
10. Perulangan *for* pertama dengan basis *c* akan mengeksekusi satu per satu set anggota *boolean* yang dibawa variabel *composite_list*. Lalu Perulangan *for* yang kedua dengan basis *l* akan menggabungkan satu per satu *composite_list* yang sudah dihasilkan perulangan pertama yang ditampung pada variabel *myBinary*.
11. Penampungan di perulangan *for* sebelumnya akan dilanjutkan untuk eksekusi perkalian antar boolean. Perkalian dilakukan karena *user* memilih operator *AND*.
12. Hasil dari perkalian antar boolean yang merepresentasikan perkalian antar binary query, akan ditampilkan pada terminal dan GUI. Seperti gambar dibawah ini : **hasil boolean : 0b1010100** 0b adalah index boolean.



3.3.2.7. Fungsi tfidfOps

Fungsi ini adalah operasi Tf-idf yang berfungsi untuk menentukan seberapa penting pengaruh suatu *term/kata* pada sebuah dokumen, hasil output ditunjukkan dengan desimal frekuensi.

```
def tfidfOps(self, userInput):
    # menghitung TF dari nilai input
    for it in range(0, len(userInput)):
        for lf in range(0, len(self.listFiles)):
            count = 0
            for j in range(0, len(self.listFiles[lf])):
                if (self.listFiles[lf][j] == userInput[it]):
                    if userInput[it] == self.listFiles[lf][j]:
                        count += 1
                else:
                    count += 0
            hasilTF.append(count)

    finalTF = [hasilTF[x:x + len(self.listFiles)] for x in range(0,
len(hasilTF), len(self.listFiles))]
    print('hasil TF : ', finalTF)

    # menghitung DF -> document frequency -> seberapa banyak
    dokumen yang mengandung sebuah kata
    print('document frequency :\n')
    for d in range(0, len(userInput)):
        dfCount = 0
        for e in self.listFiles:
            if userInput[d] in e:
                dfCount = dfCount + 1
            else:
                dfCount = dfCount + 0
        documentFreq.append(dfCount)
    print('Hasil document frequency : {}'.format(documentFreq))
    # merubah IDF
    for df in range(0, len(documentFreq)):
        documentFreq[df] = math.log(4 / documentFreq[df])
    print(documentFreq)

    # menghitung nilai TF-IDF
    for i in range(0, len(documentFreq)):
        for j in range(0, len(finalTF)):
            for k in range(0, len(finalTF[j])):
                tfidf.append(documentFreq[i] * finalTF[j][k])
    finalTfidf = [tfidf[x:x + len(self.listFiles)] for x in range(0,
len(tfidf), len(self.listFiles))]
    print('hasil TF-IDF : \n {}'.format(finalTfidf))

    # ranking tf idf
    rankTfidf = [sum(x) for x in zip(*finalTfidf)]
    c = 0
```

```

for r in rankTfidf:
    c += 1
    print('Dokumen {0} dengan nilai : {1}'.format(c, r))
    self.plainTextEdit_3.insertPlainText('Dokumen {0} dengan
nilai : {1} \n'.format(c, r))

```

Penjelasan Kode Program Diatas :

```

(13) def tfidfOps(self, userInput):
    # menghitung TF dari nilai input

    for it in range(0, len(userInput)):
        for lf in range(0, len(self.listFile)):
            count = 0

            for j in range(0, len(self.listFile[lf])):
                if (self.listFile[lf][j] == userInput[it]):
                    if userInput[it] == self.listFile[lf][j]:
                        count += 1
                    else:
                        count += 0

            hasilTF.append(count)
    finalTF = [hasilTF[x:x + len(self.listFile)] for x in range(0, len(hasilTF), len(self.listFile))]
    print('hasil TF : ', finalTF)

```

13. Kode program diatas untuk menentukan TF (*Term Frequency*) konsepnya hampir sama seperti boolean, yakni melakukan perulangan untuk menentukan ada tidaknya suatu *query/term* yang diminta pengguna muncul pada keseluruhan dokumen yang dipilih. Jika suatu *query* dinyatakan ada pada sebuah dokumen, maka variabel *count* akan diberi nilai 1, sebaliknya jika tidak maka akan tetap diberi nilai 0. Hasil kemunculan item tersebut akan ditampilkan pada terminal seperti gambar dibawah ini :

```

hasil TF : [[1, 1, 0, 0], [0, 1, 1, 1]]

```

```

print('document frequency:\n')
(14) for d in range(0, len(userInput)):
    dfCount = 0
    for e in self.listFiles:
        if userInput[d] in e:
            dfCount = dfCount + 1
        else:
            dfCount = dfCount + 0
    documentFreq.append(dfCount)
print('Hasil document frequency : {}'.format(documentFreq))

```

14. Kode program diatas untuk menentukan frekuensi DF (*Document Frequency*) untuk menentukan berapa kali kemunculan suatu *query/term* yang diminta pengguna muncul pada keseluruhan dokumen yang dipilih. Jika suatu *query* dinyatakan ada pada sebuah dokumen, maka variabel *dfCount* akan ditambah 1, sebaliknya jika tidak maka akan tetap 0. Hasil pada operasi ini akan ditampilkan pada terminal seperti gambar dibawah ini :

```
Hasil document frequency : [2, 3]
```

Misal *user* menginputkan *query* “*drug AND new*”

Gambar diatas menunjukkan bahwa *query* pertama “*drug*” muncul dua kali pada dokumen terpilih, dan *query* kedua “*new*” muncul tiga kali pada dokumen terpilih.

```

(15) for df in range(0, len(documentFreq)):
    documentFreq[df] = math.log(4 / documentFreq[df])
print(documentFreq)

```

15. Selanjutnya Kode Program diatas akan menentukan bobot DF (*Document Frequency*) dengan melakukan perulangan untuk memasukkan masing-masing hasil DF sebelumnya

```
Hasil document frequency : [2, 3]
```

 ke rumus log untuk menentukan bobot DF nya. Hasil operasi ini akan ditampilkan pada terminal :

```
[0.6931471805599453, 0.28768207245178085]
```

```
(16) for i in range(0, len(documentFreq)):
    for j in range(0, len(finalTF)):
        for k in range(0, len(finalTF[j])):
            tfidf.append(documentFreq[i] * finalTF[j][k])
        finalTfidf = [tfidf[x:x + len(self.listFile)] for x in range(0, len(tfidf),
len(self.listFile))]
        print('hasil TF-IDF : \n {}'.format(finalTfidf))
```

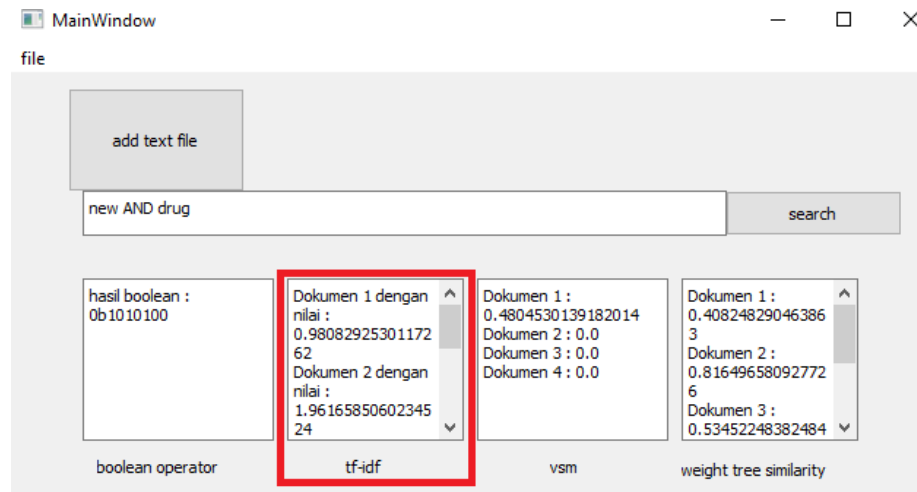
16. Kode Program diatas akan menentukan hasil TF-IDF, caranya mengalikan bobot TF dengan bobot DF, dan hasilnya akan tampil di terminal sebagai berikut :

```
[[0.6931471805599453, 0.6931471805599453, 0.0, 0.0], [0.0, 0.6931471805599453, 0.6931471805599453,
0.0, 0.0], [0.0, 0.28768207245178085, 0.28768207245178085, 0.28768207245178085]]
```

```
(17) rankTfidf = [sum(x) for x in zip(*finalTfidf)]
c = 0
for r in rankTfidf:
    c += 1
    print('Dokumen {0} dengan nilai : {1}'.format(c, r))
    self.plainTextEdit_3.insertPlainText('Dokumen {0} dengan nilai : {1}
\n'.format(c, r))
```

17. Kode program diatas untuk merapikan hasil TF-IDF sebelumnya, agar bisa memberikan informasi perangkian masing-masing nama dokumen dan bobot TF-IDF. Proses ini akan menampilkan output pada terminal dan *plainText* pada GUI seperti gambar dibawah ini :

```
Dokumen 1 dengan nilai : 0.9808292530117262
Dokumen 2 dengan nilai : 1.9616585060234522
Dokumen 3 dengan nilai : 0.9808292530117262
Dokumen 4 dengan nilai : 0.9808292530117262
```



3.3.2.8. Fungsi vsmOps

```
def vsmOps(self, userInput):
    uniqueWords = list()
    # ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach',
    'treatment', 'hope', 'patient']
    q = list()
    listTF = list()
    finalTF = list()
    dfiList = list()
    idfList = list()
    bobotQ = list()
    bobotPerDokumen = list()
    docSimilarity = list()
    finalDocSimilarity = list()
    indexingTerm = list()
    querySimilarity = 0
    totalSimilaritas = list()
    kolomBobotDokumen = list()

    for i in range(0, len(self.listFiles)):
        for w in self.listFiles[i]:
            if w not in uniqueWords:
                uniqueWords.append(w)
    print('Unique words : ', uniqueWords)

    # take user input
    userQuery = userInput
    # ['new', 'drug', 'treatment']

    # membuat list Q (query) untuk tf-idf
    for uni in uniqueWords:
        if uni not in userQuery:
            q.append(0)
        else:
            q.append(1)
```

```

print('kolom Q : ', q)
# [0, 1, 0, 1, 0, 1, 0, 0]

# membuat list-TF
for uni in uniqueWords:
    temp = 0
    for w in self.listFiles:
        if uni in w:
            temp = w.count(uni)
            listTF.append(temp)
        else:
            listTF.append(0)
print('TF : ', listTF)

finalTF = [listTF[x:x + len(self.listFiles)] for x in range(0,
len(listTF), len(self.listFiles))]

print('final tf : ', finalTF)
# [[1, 0, 0, 0], [1, 1, 0, 0], [1, 1, 1, 1], [0, 1, 1, 1], [0,
0, 1, 0], [0, 0, 1, 0], [0, 0, 0, 1], [0, 0, 0, 1]]

# menghitung idf ->bug tidak bisa jika ada angka 2
for st in finalTF:
    dfiList.append(sum(st))
    # if(finalTF[f] > 1):
    #     idfFlag+=1
    #     idfList.append(idfFlag)
    # else:
    #     idfFlag += finalTF[f]
    #     idfList.append(idfFlag)
print('dfi list : ', dfiList)

# menghitung idf
for d in dfiList:
    idfTemp = 0
    idfTemp = len(self.listFiles) / d
    idfTemp = math.log(idfTemp)
    idfList.append(idfTemp)
print('Hasil IDF : ', idfList)
# [1.3862943611198906, 0.6931471805599453, 0.0,
0.28768207245178085, 1.3862943611198906, 1.3862943611198906,
1.3862943611198906, 1.3862943611198906]

# menghitung bobot query
for qi in range(0, len(q)):
    bobotQ.append(q[qi] * idfList[qi])

# menghitung bobot per dokumen:
for to in range(0, len(finalTF)):
    for u in range(0, len(finalTF[to])):
        bobotPerDokumen.append(finalTF[to][u] * bobotQ[to])
print('bobot per dokumen : ', bobotPerDokumen)

# bobot per dokumen (FIX)

```



```

        finalBobotPerDokumen = [bobotPerDokumen[x:x +
len(self.listFiles)] for x in
                                range(0, len(bobotPerDokumen),
len(self.listFiles))]

        print('bobot kata per dokumen (final) : ',
finalBobotPerDokumen)

        kolomBobotDokumen = [sum(x) for x in
zip(*finalBobotPerDokumen)]
        print('=' * 100)
        print('jumlah bobot dokumen per kolom : ', kolomBobotDokumen)
        print('=' * 100)

        # menghitung document docSimilarity

        for a in range(0, len(finalBobotPerDokumen)):
            for b in finalBobotPerDokumen[a]:
                b = math.pow(b, 2)
                docSimilarity.append(b)

        docSimilarity = ([sum(i) for i in zip(*finalBobotPerDokumen)])

        for ds in docSimilarity:
            ds = math.sqrt(ds)
            finalDocSimilarity.append(ds)
        print('Doc similarity : ', finalDocSimilarity)

        # menghitung query similarity:
        tempQ = 0
        for a in range(0, len(bobotQ)):
            tempQ = tempQ + pow(bobotQ[a], 2)
        querySimilarity = math.sqrt(tempQ)
        print('nilai query similarity : ', querySimilarity)

        # menghapus nilai 0 pada query
        for bq in bobotQ:
            if bq == 0:
                bobotQ.remove(bq)
        print('bobot query : ', bobotQ)

        # indexing term
        for bt in range(0, len(bobotQ)):
            indexingTerm.append(bobotQ[bt] * kolomBobotDokumen[bt])
        print('index term : ', indexingTerm)

        # totalSimilaritas
        for t in range(0, len(indexingTerm)):
            totalSimilaritas.append(indexingTerm[t] / querySimilarity *
docSimilarity[t])

        print(totalSimilaritas)
        totNum = 0
        for tot in totalSimilaritas:

```

```

        totNum += 1
        self.plainTextEdit_4.insertPlainText('Dokumen {} :
        {}'.format(totNum, tot))

```

Penjelasan Kode Program Diatas :

```

(18) def vsmOps(self, userInput):
    uniqueWords = list()
    q = list()
    listTF = list()
    finalTF = list()
    dflList = list()
    idfList = list()
    bobotQ = list()
    bobotPerDokumen = list()
    docSimilarity = list()
    finalDocSimilarity = list()
    indexingTerm = list()
    querySimilarity = 0
    totalSimilaritas = list()
    kolomBobotDokumen = list()

```

18. Kode program diatas adalah definisi variabel yang pada awalnya disetting sebagai list kosong, pada VSM kita perlu *uniqueWords* untuk menampung *term/kata* yang muncul hanya sekali pada semua dokumen terpilih, lalu ada variabel *q* yang akan menampung *query* yang diinputkan pengguna, listTF untuk menampung frekuensi *term*, dan variabel lainnya diperlukan untuk memenuhi parameter-parameter metode VSM.

```

(19) for i in range(0, len(self.listFile)):
    for w in self.listFile[i]:
        if w not in uniqueWords:
            uniqueWords.append(w)
    print('Unique words:', uniqueWords)

```

19. Kode program diatas akan membaca kata/*term* yang bisa masuk kategori *uniqueWords* pada tiap dokumen, program akan mengecek tiap dokumen, semisal ada kata/*term uniqueWords* yang lebih dari satu, maka akan ditulis

sekali saja. Hasil dari proses ini akan ditampilkan di terminal seperti gambar berikut :

```
Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach',
```

```
# take user input
(20) userQuery = userInput
    # ['new', 'drug', 'treatment']
    # membuat list Q (query) untuk tf-idf
    for uni in uniqueWords:
        if uni not in userQuery:
            q.append(0)
        else:
            q.append(1)
    print('kolom Q : ', q)
    # [0, 1, 0, 1, 0, 1, 0, 0]
```

20. Kode program diatas akan membuat list *query* untuk *tf-idf* dengan cara mengambil *query* input dari user yang dibawa variabel *userQuery*, akan dicari *query* yang masuk dalam kategori *uniqueWords*, jika *query* itu ada pada daftar *uniqueWords*, maka akan diberi nilai 1, jika tidak maka diberikan nilai 0. Berikut adalah output pada terminal :

```
Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach', 'treatment', 'hope', 'patient']
kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
```

Pada kasus ini kami mencoba mencari *query* “*drug AND new*” sehingga menghasilkan output seperti diatas, kata “*drug*” dan “*new*” akan diberi nilai 1.

```
# menghitung bobot per dokumen:
(21) for to in range(0, len(finalTF)):
        for u in range(0, len(finalTF[to])):
            bobotPerDokumen.append(finalTF[to][u] * bobotQ[to])
    print('bobot per dokumen: ', bobotPerDokumen)

# bobot per dokumen (FIX)
```

```

(22) finalBobotPerDokumen = [bobotPerDokumen[x:x + len(self.listFile)] for x in
                                range(0, len(bobotPerDokumen), len(self.listFile))]

    print('bobot kata per dokumen (final): ', finalBobotPerDokumen)

(23) kolomBobotDokumen = [sum(x) for x in zip(*finalBobotPerDokumen)]

    print('=' * 100)
    print('jumlah bobot dokumen per kolom: ', kolomBobotDokumen)

    print('=' * 100)

```

21. Langsung beralih pada menghitung bobot per dokumen, untuk mendapatkan bobot per masing-masing dokumen, program akan melakukan perulangan untuk mengalikan baris/kolom *finalTF* dengan bobot *query*. Output pada proses ini ditampilkan pada gambar dibawah ini :

```

bobot per dokumen : [0.0, 0.0, 0.0, 0.0, 0.6931471805599453, 0.6931471805599453, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.28768207245178085,
0.28768207245178085, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
bobot kata per dokumen (final) : [[0.0, 0.0, 0.0, 0.0], [0.6931471805599453, 0.6931471805599453, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0,
0.28768207245178085, 0.28768207245178085], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0]]
=====
jumlah bobot dokumen per kolom : [0.6931471805599453, 0.9808292530117262, 0.28768207245178085, 0.28768207245178085]
=====

```

22. Selanjutnya program akan merapikan hasil dari pembobotan dokumen, sehingga hasil output akan menjadi seperti ini :

```

bobot per dokumen : [0.0, 0.0, 0.0, 0.0, 0.6931471805599453, 0.6931471805599453, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.28768207245178085,
0.28768207245178085, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
bobot kata per dokumen (final) : [[0.0, 0.0, 0.0, 0.0], [0.6931471805599453, 0.6931471805599453, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0,
0.28768207245178085, 0.28768207245178085], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0]]
=====
jumlah bobot dokumen per kolom : [0.6931471805599453, 0.9808292530117262, 0.28768207245178085, 0.28768207245178085]
=====

```

23. Kemudian program akan menjumlahkan keseluruhan bobot dokumen dengan perulangan *for* fungsi *sum*. Kode program `print('=' * 100)` Program akan memberikan batas simbol “=” sebanyak 100 kali. Hasil dari proses ini adalah :

```

bobot per dokumen : [0.0, 0.0, 0.0, 0.0, 0.6931471805599453, 0.6931471805599453, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.28768207245178085,
0.28768207245178085, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
bobot kata per dokumen (final) : [[0.0, 0.0, 0.0, 0.0], [0.6931471805599453, 0.6931471805599453, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0,
0.28768207245178085, 0.28768207245178085], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0]]
=====
jumlah bobot dokumen per kolom : [0.6931471805599453, 0.9808292530117262, 0.28768207245178085, 0.28768207245178085]
=====

```

```
# menghitung document docSimilarity
(24) for a in range(0, len(finalBobotPerDokumen)):
    for b in finalBobotPerDokumen[a]:
        b = math.pow(b, 2)
        docSimilarity.append(b)
docSimilarity = ([sum(i) for i in zip(*finalBobotPerDokumen)])
for ds in docSimilarity:
    ds = math.sqrt(ds)
    finalDocSimilarity.append(ds)
print('Doc similarity : ', finalDocSimilarity)
```

24. Selanjutnya program akan melakukan perhitungan Similaritas antar dokumen untuk menilai kemiripan *query* yang diminta *user*. Program akan melakukan perulangan untuk mengambil bobot per dokumen lalu masing-masing bobot akan dipangkatkan. Hasil dari operasi ini adalah sebagai berikut :

```
Doc similarity : [0.8325546111576977, 0.9903682411162659, 0.5363600213026516, 0.5363600213026516]
```

```
# menghitung query similarity:
25 tempQ = 0
for a in range(0, len(bobotQ)):
    tempQ = tempQ + pow(bobotQ[a], 2)
querySimilarity = math.sqrt(tempQ)
print('nilai query similarity : ', querySimilarity)
# menghapus nilai 0 pada query
(26) for bq in bobotQ:
    if bq == 0:
        bobotQ.remove(bq)
print('bobot query : ', bobotQ)
```

25. Selanjutnya Program akan menghitung similaritas *query* bertujuan untuk mengetahui seberapa besar kemiripan suatu *query* pada tiap dokumen, dalam hal ini program akan mengakses kembali pembobotan *query* yang sudah

ditentukan sebelumnya. Misalkan bobot *query* satu adalah 0,002 dan *query* dua 0,0090, maka program akan melakukan operasi $0,002^2 + 0,0090^2$ lalu hasil akhirnya akan diakarkan. Hasil output dari proses ini adalah sebagai berikut :

```
nilai query similarity : 0.7504758415354574
```

26. Setelah mendapatkan hasil pembobotan pada step diatas, maka selanjutnya pada step ini nilai 0 akan dihapus, sehingga hasil output pada terminal adalah sebagai berikut :

```
bobot query : [0.6931471805599453, 0.28768207245178085, 0.0, 0.0]
```

```
# indexing term
(27) for bt in range(0, len(bobotQ)):
    indexingTerm.append(bobotQ[bt] * kolomBobotDokumen[bt])
    print('index term : ', indexingTerm)
# totalSimilaritas
(28) for t in range(0, len(indexingTerm)):
    totalSimilaritas.append(indexingTerm[t] / querySimilarity * docSimilarity[t])
    print(totalSimilaritas)
    totNum = 0
    for tot in totalSimilaritas:
        totNum += 1
    self.plainTextEdit_4.insertPlainText('Dokumen {}: {} \n'.format(totNum, tot))
```

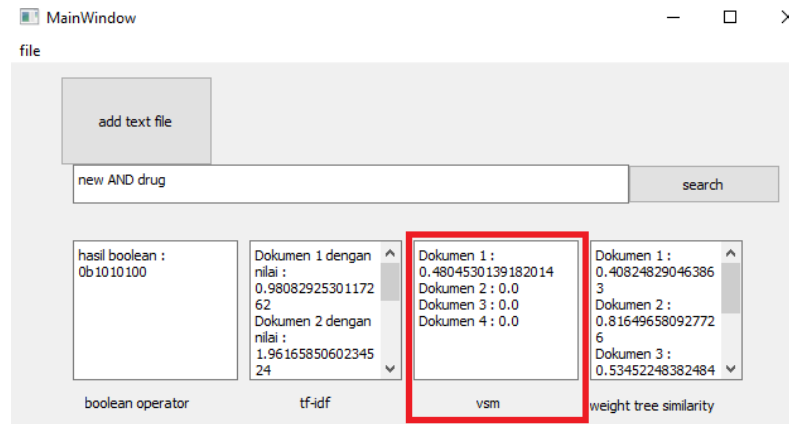
27. Step selanjutnya program akan melakukan *indexing term* untuk mengetahui perangkian *term/query* berdasarkan bobot yang sudah kita dapat pada tiap dokumen dengan cara mengalikan tiap bobot *query* dengan kolom bobot dokumen. Hasil dari proses ini adalah sebagai berikut :

```
index term : [0.4804530139182014, 0.2821669922277455, 0.0, 0.0]
```

28. Step terakhir program akan mencari total similaritas dengan menjumlahkan tiap hasil *indexing term* dibagi dengan *querySimilarity* lalu dikali dengan tiap baris *docSimilarity*. Hasil dari operasi ini adalah sebagai berikut :

```
[0.443751328900299, 0.36877621489462603, 0.0, 0.0]
```

Dan hasil juga ditampilkan pada GUI seperti berikut :



3.3.2.9. Fungsi wtsOps

```
def wtsOps(self, userInput):
    uniqueWords = list()
    # ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach',
    'treatment', 'hope', 'patient']
    q = list()
    t1 = list()
    listTF = list()
    listTF2 = list()
    listF3 = list()
    listTF3 = list()
    listTF4 = list()
    SIM1 = list()
    SMD = list()
    SMD1 = list()
    AKR = list()
    pangkat1 = list()
    pangkat2 = list()
    perkalian2 = list()
    similarity = list()

    # menyaring kata yang unik
    # menentukan jumlah filenya dari 0 sampai max
    for i in range(0, len(self.listFiles)):
        # proses looping per kata
        for w in self.listFiles[i]:
            if w not in uniqueWords:
                uniqueWords.append(w)
    print('Unique words : ', uniqueWords)
    # print(list())

    # take user input
    userQuery = userInput
    # ['new', 'drug', 'treatment']

    # membuat list Q (query) untuk tf-idf
```

```

for uni in uniqueWords:
    if uni not in userQuery:
        q.append(0)
    else:
        q.append(1)
print('kolom Q : ', q)
# [0, 1, 0, 1, 0, 1, 0, 0]

# membuat list-TF
for uni in uniqueWords:
    temp = 0
    for w in self.listFiles:
        if uni in w:
            temp = w.count(uni)
            listTF.append(temp)
        else:
            listTF.append(0)
print('TF : ', listTF)

# TF 2, pembanding
lf = len(self.listFiles)
for b in range(lf, len(listTF)):
    listTF2.append(listTF[b])
# print('TF2: ', listTF2)

finalTF = [listTF[x:x + len(self.listFiles)] for x in range(0,
len(listTF), len(self.listFiles))]
print('final tf : ', finalTF)
# [[1, 0, 0, 0], [1, 1, 0, 0], [1, 1, 1, 1], [0, 1, 1, 1], [0,
0, 1, 0], [0, 0, 1, 0], [0, 0, 0, 1], [0, 0, 0, 1]]

# mencari final tf ke 0
for a in range(0, lf):
    tl.append(listTF[a])
# print('list a : ', tl)
# total frekuensi dokumen
n = 0
for b in range(0, len(q) - 1):
    for c in range(0, lf):
        tl[c] += listTF2[n]
        n += 1
print('total frekuensi dokumen: ', tl)

# total frekuensi Q
n = 0
for d in range(0, len(q)):
    n += q[d]
print('total frekuensi Q: ', n)

# total frekuensi bobot dokumen TF
j = 0
for a in range(0, len(q)):
    for b in range(0, lf):
        listF3.append(listTF[j] / tl[b])

```



```

        j += 1
    print('bobot dokumen F: ', listF3)

    for a in range(0, lf):
        t = 0
        for b in range(0, len(q)):
            listTF3.append(listF3[a + t])
            t += 1f
    print('bobot dokumen TF: ', listTF3)

    # total frekuensi bobot query TF
    for a in range(0, len(q)):
        listTF4.append(q[a] / n)
    print('bobot Q TF: ', listTF4)

    # total frekuensi bobot dokumen TF
    j = 0
    for a in range(0, lf):
        for b in range(0, len(q)):
            SIM1.append(listTF4[b] * listTF3[j])
            j += 1
    print('perkalian1: ', SIM1)

    # sigma total Q & doc
    j = 0
    for a in range(0, lf):
        r = 0
        for d in range(0, len(q)):
            r = r + SIM1[j]
            j += 1
        SMD.append(r)
    print('sigma pembagil: ', SMD)

    sig1 = 0
    for d in range(0, len(q)):
        pangkat1.append(listTF4[d] ** 2)
        sig1 += pangkat1[d]
        akr1 = math.sqrt(sig1)
    print('pangkat Q: ', pangkat1)
    print('sigma Q: ', sig1)
    print('akr Q: ', akr1)

    # sigma total Q & doc
    j = 0
    for a in range(0, lf):
        r = 0
        for d in range(0, len(q)):
            pangkat2.append(listTF3[j] ** 2)
            r += pangkat2[j]
            j += 1
        SMD1.append(r)
        # akr2 = math.sqrt(sig2)
    print('pangkat doc: ', pangkat2)
    print('sigma doc: ', SMD1)

```

```

for a in range(0, lf):
    AKR.append(math.sqrt(SMD1[a]))
    perkalian2.append(akr1 * AKR[a])
print('akr doc: ', AKR)
print('perkalian2 pembagi2 ', perkalian2)

for a in range(0, lf):
    similarity.append(SMD[a] / perkalian2[a])
print('hasil similarity : ', similarity)
totNum = 0
for tot in similarity:
    totNum += 1
    self.plainTextEdit_5.insertPlainText('Dokumen {} :
    {} \n'.format(totNum, tot))

```

Penjelasan Kode Program Diatas :

```

uniqueWords = list()
q = list()
t1 = list()
listTF = list()
listTF2 = list()
listF3 = list()
listTF3 = list()
listTF4 = list()
SIM1 = list()
SMD = list()
SMD1 = list()
AKR = list()
pangkat1 = list()
pangkat2 = list()
perkalian2 = list()
similarity = list()

```

Merupakan variable yang mendeklarasikan pembuatan list.

```

for i in range(0, len(self.listFiles)):
    # proses looping per kata
    for w in self.listFiles[i]:
        if w not in uniqueWords:
            uniqueWords.append(w)
print('Unique words : ', uniqueWords)

```

```

bobot query : [0.6931471805599453, 0.0, 0.0, 0.0]
index term : [0.4804530139182014, 0.0, 0.0, 0.0]
[0.4804530139182014, 0.0, 0.0, 0.0]
Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach', 'treatment', 'hope', 'patient']
kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
TF : [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1]

```

Merupakan proses yang berfungsi untuk mengambil kata yang unik.

```

userQuery = userInput
# ['new', 'drug', 'treatment']

# membuat list Q (query) untuk tf-idf
for uni in uniqueWords:
    if uni not in userQuery:
        q.append(0)
    else:
        q.append(1)
print('kolom Q : ', q)
# [0, 1, 0, 1, 0, 1, 0, 0]

```

```

bobot query : [0.6931471805599453, 0.0, 0.0, 0.0]
index term : [0.4804530139182014, 0.0, 0.0, 0.0]
[0.4804530139182014, 0.0, 0.0, 0.0]
Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach', 'treatment', 'hope', 'patient']
kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
TF : [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1]

```

Membuat list Query yang telah diinputkan user, jika sesuai dengan dokumen maka akan bernilai 1 jika tidak maka akan bernilai 0

```

# membuat list-TF
for uni in uniqueWords:
    temp = 0
    for w in self.listFiles:
        if uni in w:
            temp = w.count(uni)
            listTF.append(temp)
        else:
            listTF.append(0)
print('TF : ', listTF)

```

```

Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach', 'treatment', 'hope', 'patient']
kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
TF : [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1]
TF1 : [1, 0, 0, 0]
total frekuensi dokumen: [3, 3, 5, 4]
total frekuensi Q: 2

```

Mencari Term Frekuensi yang dimiliki oleh dokumen

```

# TF 2, pembandingan dengan TF 1
lf = len(self.listFiles)
for b in range(lf, len(listTF)):
    listTF2.append(listTF[b])
print('TF2: ', listTF2)

```

```

kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
TF : [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1]
TF1 : [1, 0, 0, 0]
total frekuensi dokumen: [3, 3, 5, 4]

```

Pembandingan dengan TF1 dengan tujuan untuk mencari TF baris 2 keatas dalam suatu table.

```

# mencari tf barisan pertama pada tabel sebagai pembanding1
for a in range(0, lf):
    tl.append(listTF[a])
print('TF1 : ',tl)

```

```

Unique words : ['breakthrough', 'drug', 'schizophrenia', 'new', 'approach', 'treatment', 'hope', 'patient']
kolom Q : [0, 1, 0, 1, 0, 0, 0, 0]
TF : [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF1 : [1, 0, 0, 0]
total frekuensi dokumen: [3, 3, 5, 4]
total frekuensi Q: 2

```

Mencara pembanding TF1 untuk dibandingkan dengan TF2

```

# total frekuensi dokumen
n = 0
for b in range(0, len(q) - 1):
    for c in range(0, lf):
        tl[c] += listTF2[n]
        n += 1
print('total frekuensi dokumen: ', tl)

```

```

TF2: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]
TF1 : [1, 0, 0, 0]
total frekuensi dokumen: [3, 3, 5, 4]
total frekuensi Q: 2
bobot dokumen F: [0.3333333333333333, 0.0, 0.0, 0.0, 0.3333333333333333, 0.3333333333333333, 0.0, 0.0, 0.3333333333333333, 0.3333333333333333, 0.2, 0.25, 0.0, 0.3333333333333333,
bobot dokumen TF: [0.3333333333333333, 0.3333333333333333, 0.3333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.3333333333333333, 0.3333333333333333, 0.0, 0.0,
bobot Q TF: [0.0, 0.5, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0]

```

Mencari total frekuensi dokumen dengan cara menjumlahkan setiap frekuensi secara vertical dengan cara menambahkan TF1 dengan TF2 secara berulang.

```

# total frekuensi Q
n = 0
for d in range(0, len(q)):
    n += q[d]
print('total frekuensi Q: ', n)

```



```

for b in range(0, len(q)):
    SIM1.append(listTF4[b] * listTF3[j])
    j += 1
print('perkalian1: ', SIM1)

```

```

bobot dokumen TF: [0.3333333333333333, 0.3333333333333333, 0.3333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333]
bobot Q TF: [0.0, 0.5, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0]
perkalian1: [0.0, 0.16666666666666666, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.16666666666666666, 0.0, 0.16666666666666666, 0.0]
sigma pembagi1: [0.16666666666666666, 0.3333333333333333, 0.2, 0.125]
pangkat Q: [0.0, 0.25, 0.0, 0.25, 0.0, 0.0, 0.0, 0.0]
sigma Q: 0.5
akr Q: 0.7071067811865476

```

$$\begin{aligned}
 & \text{Similarity Nama (Query, Penyakit)} \\
 &= \frac{\sum \text{Bobot}_Q * \text{Bobot}_P}{\sqrt{\sum (\text{Bobot}_Q^2)} * \sqrt{\sum (\text{Bobot}_P^2)}} \\
 &= \frac{0}{0.57743 * 1} \\
 &= 0
 \end{aligned}$$

Menjalankan proses perkalian awal pada cosine similarity.

```

# sigma total Q & doc
j = 0
for a in range(0, lf):
    r = 0
    for d in range(0, len(q)):
        r = r + SIM1[j]
        j += 1
    SMD.append(r)
print('sigma pembagi1: ', SMD)

```

```

bobot dokumen TF: [0.3333333333333333, 0.3333333333333333, 0.3333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333]
bobot Q TF: [0.0, 0.5, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0]
perkalian1: [0.0, 0.16666666666666666, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.16666666666666666, 0.0, 0.16666666666666666, 0.0]
sigma pembagi1: [0.16666666666666666, 0.3333333333333333, 0.2, 0.125]
pangkat Q: [0.0, 0.25, 0.0, 0.25, 0.0, 0.0, 0.0, 0.0]
sigma Q: 0.5
akr Q: 0.7071067811865476

```

$$\begin{aligned}
 &= \frac{\sum (\text{Bobot}_Q * \text{Bobot}_P)}{\sqrt{\sum (\text{Bobot}_Q^2)} * \sqrt{\sum (\text{Bobot}_P^2)}} \\
 &= \frac{0}{0.57743 * 1} \\
 &= 0
 \end{aligned}$$

Menjalankan proses sigma awal pada cosine similarity.

```
sig1 = 0
for d in range(0, len(q)):
    pangkat1.append(listTF4[d] ** 2)
    sig1 += pangkat1[d]
    akr1 = math.sqrt(sig1)
print('pangkat Q: ', pangkat1)
print('sigma Q: ', sig1)
print('akr Q: ', akr1)
```

```
sigma pemuaian: [0.10000000000000000, 0.3333333333333333, 0.2, 0.125]
pangkat Q: [0.0, 0.25, 0.0, 0.25, 0.0, 0.0, 0.0, 0.0]
sigma Q: 0.5
akr Q: 0.7071067811865476
pangkat doc: [0.1111111111111111, 0.1111111111111111, 0.1111111111111111, 0.0, 0.0, 0.0, 0.0, 0.0]
sigma doc: [0.3333333333333333, 0.3333333333333333, 0.28, 0.25]
akr doc: [0.5773502691896257, 0.5773502691896257, 0.5291502622129182, 0.5]
```

$$= \frac{\sum(Bobot_{\tilde{Q}} * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$$

Menjalankan proses pangkat, sigma dan akar pada bobot query

```
# sigma total Q & doc
j = 0
for a in range(0, lf):
    r = 0
    for d in range(0, len(q)):
        pangkat2.append(listTF3[j] ** 2)
        r += pangkat2[j]
        j += 1
    SMD1.append(r)
    # akr2 = math.sqrt(sig2)
print('pangkat doc: ', pangkat2)
print('sigma doc: ', SMD1)
```

```
akr Q: 0.7071067811865476
pangkat doc: [0.1111111111111111, 0.1111111111111111, 0.1111111111111111, 0.0, 0.0, 0.0, 0.0, 0.0]
sigma doc: [0.3333333333333333, 0.3333333333333333, 0.28, 0.25]
akr doc: [0.5773502691896257, 0.5773502691896257, 0.5291502622129182, 0.5]
perkalian2 pembagi2 [0.408248290463863, 0.408248290463863, 0.3741657386773942, 0.3535533905932738]
hasil similarity : [0.408248290463863, 0.816496580927726, 0.5345224838248487, 0.3535533905932738]
```

$$= \frac{\sum(Bobot_{\tilde{Q}} * Bobot_P)}{\sqrt{\sum(Bobot_Q^2)} * \sqrt{\sum(Bobot_P^2)}}$$

Melakukan proses pangkat dan sigma pada bobot dokumen

```
for a in range(0, lf):
    AKR.append(math.sqrt(SMD1[a]))
    perkalian2.append(akr1 * AKR[a])
print('akr doc: ', AKR)
print('perkalian2 pembagi2 ', perkalian2)
```

```
↑
akr Q: 0.7071067811865476
pangkat doc: [0.1111111111111111, 0.1111111111111111, 0.1111111111111111, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
sigma doc: [0.3333333333333333, 0.3333333333333333, 0.28, 0.25]
↓
akr doc: [0.5773502691896257, 0.5773502691896257, 0.5291502622129182, 0.5]
perkalian2 pembagi2 [0.408248290463863, 0.408248290463863, 0.3741657386773942, 0.3535533905932738]
hasil similarity : [0.408248290463863, 0.816496580927726, 0.5345224838248487, 0.3535533905932737]
»
```

$$\text{Similarity Value (Query, Document)} = \frac{\sum(Bobot_q * Bobot_p)}{\sqrt{\sum(Bobot_q^2)} * \sqrt{\sum(Bobot_p^2)}}$$

Melakukan proses akar pada bobot dokumen dan perkalian antara hasil akar bobot query dengan hasil akar bobot dokumen

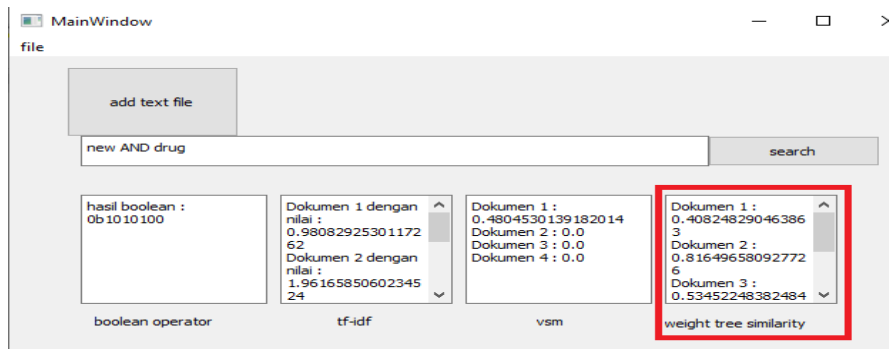
```
for a in range(0, lf):
    similarity.append(SMD[a] / perkalian2[a])
print('hasil similarity : ', similarity)
```

```
↑
pangkat doc: [0.1111111111111111, 0.1111111111111111, 0.1111111111111111, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
sigma doc: [0.3333333333333333, 0.3333333333333333, 0.28, 0.25]
↓
akr doc: [0.5773502691896257, 0.5773502691896257, 0.5291502622129182, 0.5]
perkalian2 pembagi2 [0.408248290463863, 0.408248290463863, 0.3741657386773942, 0.3535533905932738]
hasil similarity : [0.408248290463863, 0.816496580927726, 0.5345224838248487, 0.3535533905932737]
»
```

$$\text{Similarity Value (Query, Document)} = \frac{\sum(Bobot_q * Bobot_p)}{\sqrt{\sum(Bobot_q^2)} * \sqrt{\sum(Bobot_p^2)}}$$

Melakukan proses pembagian akhir pada proses similarity

```
totNum = 0
for tot in similarity:
    totNum += 1
    self.plainTextEdit_5.insertPlainText('Dokumen {} : {} \n'.format(totNum, tot))
```

Menampilkan pada GUI weigh tree similarity.

3.3.2.10. Fungsi Main

Dibawah ini adalah fungsi utama dari sistem kami :

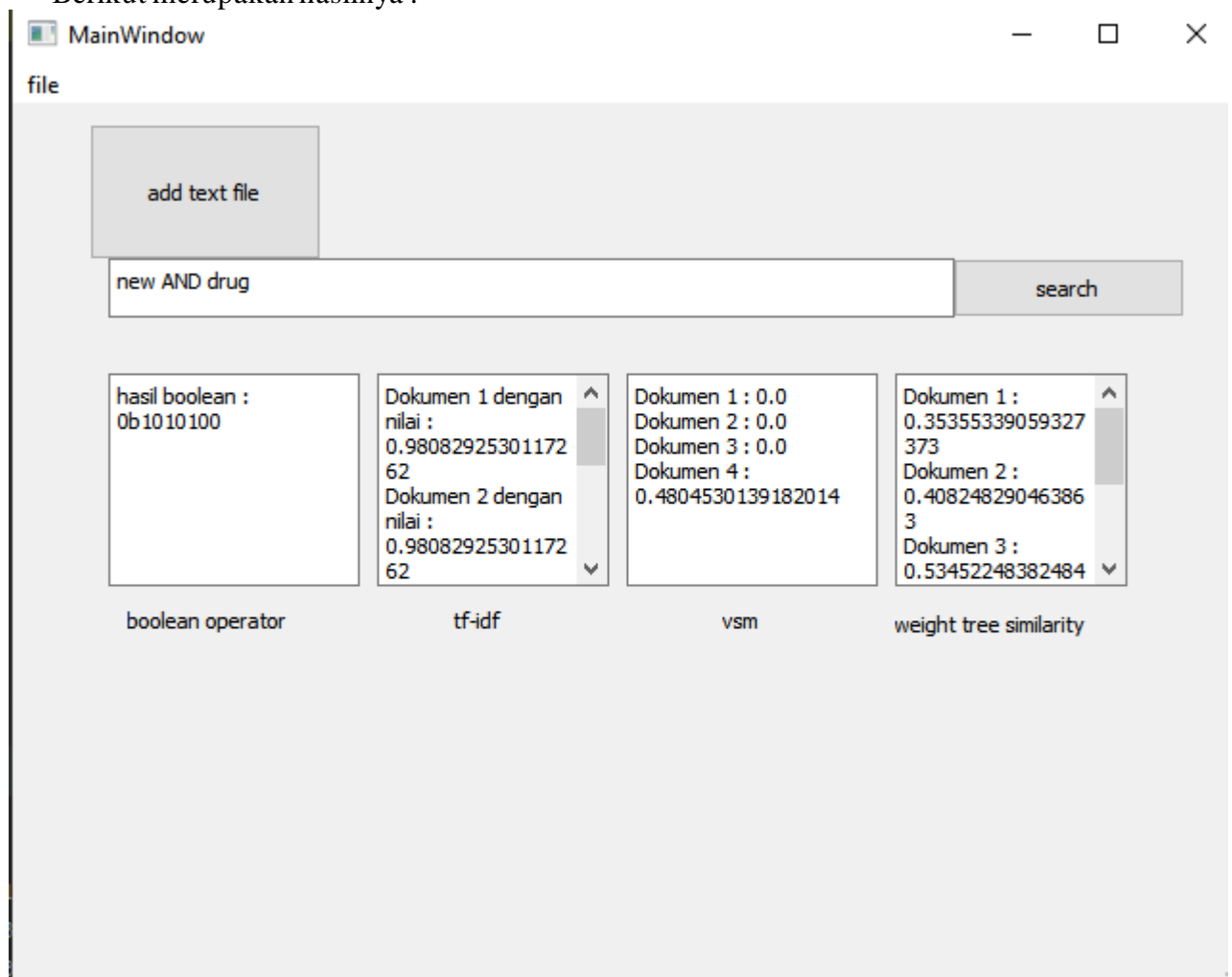
```
if __name__ == "__main__":
    import sys
    #menyimpan file yang dipilih
    listFile = list()
    # vocabulary -> seluruh kata unik yang ada dalam text
    vocab = list()
    # list hasil term frequencies
    hasilTF = list()
    # list document frequency
    documentFreq = list()
    # list TF-IDF
    tfidf = list()
    finalTfidf = list()
    # ranking tfidf
    rankTfidf = list()
    # boolean result
    booleanRes = list()
    # boolOperator
    boolOperator = ['AND', 'OR']
    activeOperator = ''
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui =
    Ui_MainWindow(listFile,vocab,hasilTF,documentFreq,tfidf,finalTfidf,r
    ankTfidf, booleanRes, activeOperator)
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec ())
```

3.3.2.11. Fungsi Boobleshort

Fungsi boobleshort berfungsi untuk mengurutkan dokumen berdasarkan bobot yang dihasilkan, berikut merupakan function boobleshort :

```
def bubbleSort(self, nlist):
    for passnum in range(len(nlist) - 1, 0, -1):
        for i in range(passnum):
            if nlist[i] > nlist[i + 1]:
                temp = nlist[i]
                nlist[i] = nlist[i + 1]
                nlist[i + 1] = temp
```

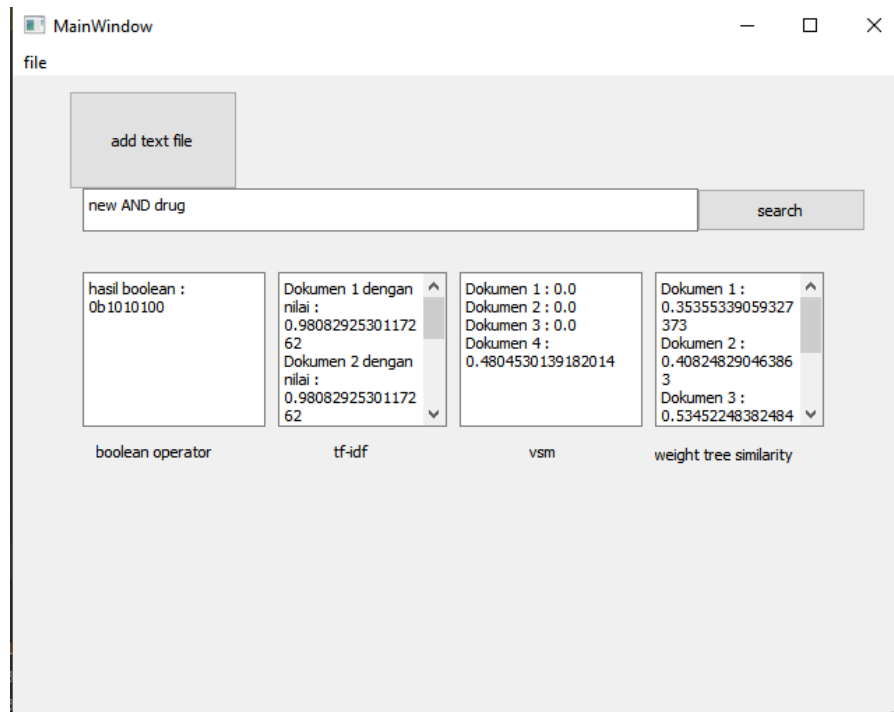
Berikut merupakan hasilnya :



BAB IV KESIMPULAN

Dapat disimpulkan bahwa sistem temu kembali informasi merupakan sebuah sistem yang berguna dalam mengetahui pentingnya sebuah term/kata sesuai dengan query permintaan pengguna. Sistem temu kembali informasi memiliki tujuan akhir, yaitu memberikan kepuasan informasi bagi pengguna sistem. Jadi, temu kembali informasi merujuk pada keseluruhan kegiatan yang meliputi pembuatan wakil informasi (*representation*), penyimpanan (*storage*), pengaturan (*organization*) sampai kepada pengambilan (*access*).

Dan kesimpulan teknis dari implementasi proyek kami menyatakan bahwa metode Wighted Tree Similarity memiliki hasil pembobotan terbaik, setelah diamati dari hasil screenshot berikut :



Berdasarkan screenshot diatas, metode Wighted Tree Similarity mampu menghasilkan pembobotan lebih banyak daripada VSM yang hanya menghasilkan pembobotan pada dokumen 1. Hal ini juga diperkuat pada penelitian yang dilakukan oleh Universitas Sebelas Maret Jurnal Itsmart Vol4. No 2. Desember 2015 Issn : 2301–7201 73 Analisis Perbandingan Metode *Vector Space Model* Dan *Weighted Tree Similarity* Dengan *Cosine Similarity* Pada Kasus Pencarian Informasi Pedoman Pengobatan Dasar Di Puskesmas, menyatakan bahwa metode *Weighted Tree Similarity*

merupakan metode terbaik. Hal ini sudah dibuktikan pada pengujian sistem dan pakar, metode *Weighted Tree Similarity* memberikan nilai *precision* yang lebih baik dibandingkan dengan VSM, meskipun nilai *recall* lebih kecil dibandingkan pada metode VSM. Sedangkan perbedaan nilai *recall* juga tidak terlalu jauh.

DAFTAR PUSTAKA

1. Wicaksono B., Saptono R. dan Sihwi W. (2015). *Analisis Perbandingan Metode Vector Space Model dan Weighted Tree Similarity dengan Cosine Similarity pada kasus Pencarian Informasi Pedoman Pengobatan Dasar di Puskesmas*. Jurnal ITSmart Vol 4. No 2.
2. Zafikri A. (2010). Implementasi Metode *Term Frequency Inverse Document Frequency* (TF-IDF) pada Sistem Temu Kembali Informasi.
3. Amin, F. (2012). Sistem Temu Kembali Informasi dengan Metode *Vector Space Model* . *Jurnal Sistem Informasi Bisnis* 02 , 78-83.
4. Sarno, R., & Rahutomo, F. (2008). Penerapan Algoritma *Weighted Tree Similarity* Untuk Pencarian Semantik. *JUTI*, 35-42.