Nama : Mokh Rif’an Ardiansyah
NIM : 16/394089/PA/17180

result :

gejala => hasil diagnosis
Mata lengket, mata berair, pandangan sedikit kabur,kelopak mata membengkak => Conjunctivitis
Gusi bengkak, gusi kemerahan, gusi berdarah => Karies dentis
Batuk lebih dari tiga minggu, sesak napas atau nyeri dada, dahak berdarah, keringat malam, nafsu makan berkurang, berat badan menurun => Tuberkulosis
Demam, menggigil, suhu tubuh meningkat, batuk berdahak kadang disertai darah, sesak nafas, nyeri dada => Pneumonia
Nyeri kolik daerah pinggang,malaise, mual, kencing bisa berdarah => Pelvicinflammatory disease
ruam yang gatal terdiri dari macula-makula kecil, lesu dan demam lebih dari tujuh hari, pembentukan parut pada ekstremitas => Rubela campak jerman
Demam, muntah, diare cair,ampas sedikit seperti biji Lombok, feses asam => Kolera

link source code : https://github.com/rifansyah/pattern_recognition/blob/master/Tugas%203/Tf-idf.py

source code :

```python
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
import math

class TFIDF(object):
    def __init__(self, n_data_train, n_data_test):
        self.words_data = []
        self.tf_map = []
        self.diagnosis_list = []
        self.idf_dict = {}
        self.tf_idf_map = []
        self.count_dict = {}
        self.tf_idf_vector_map = []
        self.n_data_train = n_data_train
        self.n_data_test = n_data_test
        self.symptoms_list = []

        self.factory = StemmerFactory()
        self.stemmer = self.factory.create_stemmer()

    def stemming(self, line):
        line = self.stemmer.stem(line)
        return line

    # def getDiagnosisLabe(self, ):

    # separate diagnosis and symptoms
    # collect the symptoms
    def generate_words_data(self, file):

        for line in file.readlines():
            arr_temp = line.split(';')
            self.symptoms_list.append(arr_temp[0])
            arr_temp[0] = self.stemming(arr_temp[0])
            temp = []
            for word in arr_temp[0].split(" "):
                temp.append(word.replace(",", ""))
```

```
                                        .replace(".", "")
                                        .replace("(", "")
                                        .replace(")", ""))
            self.words_data.append(temp)
            self.diagnosis_list.append(arr_temp[1].replace(",", "")
                                        .replace(".", "")
                                        .replace("(", "")
                                        .replace(")", "")
                                        .replace("\n",""))


    def computeTfValue(self, document_words_data):
        tfDictTemp = {}

        for word in document_words_data:
            if word in tfDictTemp:
                tfDictTemp[word] += 1
            else:
                tfDictTemp[word] = 1

        for word in tfDictTemp:
            tfDictTemp[word] = tfDictTemp[word] / len(tfDictTemp)

        return tfDictTemp

    def computeCountDict(self):
        dict_temp = {}

        for document in self.tf_map:
            for word in document:
                if word in dict_temp:
                    dict_temp[word] += 1
                else:
                    dict_temp[word] = 1
        return dict_temp

    def computeIDFDict(self):
        idf_dict_temp = {}

        for word in self.count_dict:
            idf_dict_temp[word] = math.log(len(self.words_data) / self.count_dict[word])
        return idf_dict_temp

    def computeTFIDFDict(self, documentTFDict):

        tfidf_dict_temp = {}

        for word in documentTFDict:
            tfidf_dict_temp[word] = documentTFDict[word] * self.idf_dict[word]
        return tfidf_dict_temp

    def computeTFIDFVector(self, document_tfidf_dict, word_dict):

        tfidf_vector = [0.0] * len(word_dict)

        for i, word in enumerate(word_dict):
            if word in document_tfidf_dict:
                tfidf_vector[i] = document_tfidf_dict[word]
        return tfidf_vector
    def dotProduct(self, vector_x, vector_y):
        dot = 0.0
        for e_x, e_y in zip(vector_x, vector_y):
            dot += e_x * e_y
        return dot

    def getMagnitude(self, vector):
        mag = 0.0
        for index in vector:
            mag += math.pow(index, 2)
        return math.sqrt(mag)

    def similiarity(self, vector_1, vector_2):
        return self.dotProduct(vector_1, vector_2) / (self.getMagnitude(vector_1) * 
self.getMagnitude(vector_2))


    def train(self, file):
        self.generate_words_data(file)

        document_row = len(self.words_data)

        for i in range(document_row):
            self.tf_map.append(self.computeTfValue(self.words_data[i]))

        self.count_dict = self.computeCountDict()

        self.idf_dict = self.computeIDFDict()

        self.tf_idf_map = [self.computeTFIDFDict(document) for document in self.tf_map]

        word_dict = sorted(self.count_dict.keys())

        self.tf_idf_vector_map = [self.computeTFIDFVector(document_tf_idf_dict, word_dict) 
for document_tf_idf_dict in self.tf_idf_map]

    def showResult(self):
        print("gejala => hasil diagnosis")
        for i in range(self.n_data_train, self.n_data_train + self.n_data_test):
            best_diagnosis_possibilities = ""
            temp = 0
            best_index = -1
            for j in range(self.n_data_train):
                if temp < self.similiarity(self.tf_idf_vector_map[i], 
self.tf_idf_vector_map[j]):
                    temp = max(temp, self.similiarity(self.tf_idf_vector_map[i], 
self.tf_idf_vector_map[j]))
                    best_index = j
            best_diagnosis_possibilities = self.diagnosis_list[best_index]

            print(self.symptoms_list[i] + " => " + best_diagnosis_possibilities)




def main():
    file = open('diagnosis.txt', 'r')
    n_data_train = 93
    n_data_test = 7

    tfidf = TFIDF(n_data_train, n_data_test)
    tfidf.train(file)
    tfidf.showResult()

if __name__ == '__main__':
    main()
```