1.Duży (> 1000 elementów) zbiór dokumentów tekstowych w języku angielskim

def generete\_texts():  
 for i in range(1001):  
 gen = MarkovTextGenerator()  
 file\_name = 'text' + str(i) + '.txt'  
 path = 'texts/' + file\_name  
 string\_to\_write = gen.gen\_text(100)  
 file = open(path, 'w')  
 file.write(string\_to\_write)  
 file.close()  
 # os.remove(path)

2.Słownik słów kluczowych (termów)- vocab w funkcji generate\_bow()

3. Wektor cech bag-of-words bag\_vector[i] w funkcji generate\_bow()

4. Macierz wektorów  cech term-by-document matrix – transpose\_list w funkcji generate\_bow()

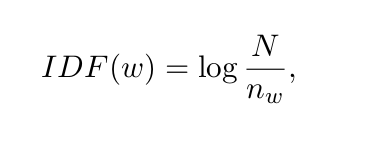
def word\_extraction(sentence):  
 ignore = ['a', "the", "is"]  
 words = re.sub("[^\w]", " ", sentence).split()  
 cleaned\_text = [w.lower() for w in words if w not in ignore]  
 cleaned\_text2 = [w.replace(",", "") for w in cleaned\_text]  
 cleaned\_text3 = [w.replace(".", "") for w in cleaned\_text2]  
 return cleaned\_text3

def tokenize(sentences):  
 words = []  
 for sentence in sentences:  
 w = word\_extraction(sentence)  
 words.extend(w)  
 words = sorted(list(set(words)))  
 return words

def texts\_to\_array():  
 allsentences = []  
 for i in range(1001):  
 file\_name = 'text' + str(i) + '.txt'  
 path = 'texts/' + file\_name  
 sentence = Path(path).read\_text()  
 allsentences.append(sentence)  
 return allsentences

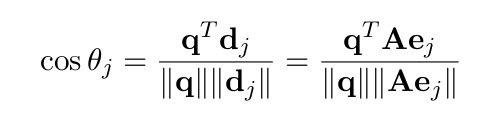
def generate\_bow():  
 allsentences = texts\_to\_array()  
 global vocab  
 vocab = tokenize(allsentences)  
 # print("Word List for Document \n{0} \n".format(vocab))  
 bag\_arr = []  
 LEN\_VOC = len(vocab)  
 for sentence in allsentences:  
 words = word\_extraction(sentence)  
 bag\_vector = [0]\*LEN\_VOC  
 for w in words:  
 for i,word in enumerate(vocab):  
 if word == w:  
 bag\_vector[i] += 1  
  
 bag\_arr.append(bag\_vector)  
  
  
 #print("{0}\n{1}\n".format(words,numpy.array(bag\_vector)))  
 numpy\_array = numpy.array(bag\_arr)  
 transpose = numpy\_array.T  
 global matrix\_before\_trans  
 matrix\_before\_trans = numpy\_array.tolist()  
 transpose\_list = transpose.tolist()  
 return transpose\_list, vocab

5.Przetworzony wstępnie otrzymany zbiór(document\_matrix) danych mnożąc elementy bag-of-words przez inverse document frequency



def inverse\_document\_frequency():  
 global document\_matrix  
 document\_matrix, voc = generate\_bow()  
 LEN\_VOC = len(vocab)  
 counter\_words = [0] \* (LEN\_VOC)  
 for i in range(LEN\_VOC):  
 for j in range(1001):  
 if document\_matrix[i][j] != 0:  
 counter\_words[i] += 1  
 if counter\_words[i] != 0:  
 counter\_words[i] = math.log(1001/counter\_words[i])  
  
 for i in range(1001):  
 for j in range(LEN\_VOC):  
 document\_matrix[j][i] \*= counter\_words[j]  
  
 return document\_matrix

6. Cos i k najbardziej podobnych dokumentów



def cos(vector1, vector2):  
 cos = 0  
 for i in range(len(vector1)):  
 cos += (vector1[i]\*vector2[i])  
 len1 = 0  
 len2 = 0  
 for i in range(len(vector2)):  
 len1 += vector1[i]\*\*2  
 len2 += vector2[i]\*\*2  
 len1 = numpy.sqrt(len1)  
 len2 = numpy.sqrt(len2)  
  
 cos = cos/(len1\*len2)  
 return cos

def k\_most\_similar\_texts(k):  
 compare\_document = []  
 print(matrix\_before\_trans)  
 input\_bag\_of\_words = input\_to\_bag\_of\_words()  
 for i in range(1001):  
 similiar = cos(matrix\_before\_trans[i], input\_bag\_of\_words)  
 t = (similiar, i)  
 compare\_document.append(t)  
 compare\_document.sort(key = lambda x: x[0], reverse = True)  
  
 print("the most similar texts:")  
  
 for i in range(k):  
 file\_name = 'text' + str(compare\_document[i][1]) + '.txt'  
 path = 'texts/' + file\_name  
 print("{0} similiarity: {1}".format(path, compare\_document[i][0]))

7. Normalizacja wektora

def normalize(v):  
 norm=numpy.linalg.norm(v, ord=1)  
 if norm==0:  
 norm=numpy.finfo(v.dtype).eps  
 return v/norm

Normalizacja wektorów w macierzy

def normalize\_matrix():  
 global normalized\_matrix  
 normalized\_matrix = []  
 for i in range(1001):  
 normalized\_matrix.append(normalize(matrix\_before\_trans[i]))

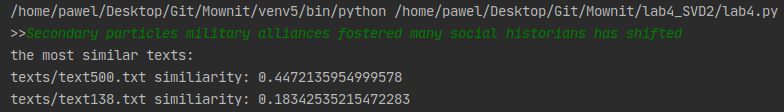
Funkcja pozwalająca wprowadzić tekst ze standardowego wejścia i przekształca go w bag\_of\_words

def input\_to\_bag\_of\_words():  
 input\_string= input(">>")  
 LEN\_VOC = len(vocab)  
 words = word\_extraction(input\_string)  
 bag\_vector = [0] \* LEN\_VOC  
  
 for w in words:  
 for i, word in enumerate(vocab):  
 if word == w:  
 bag\_vector[i] += 1  
  
 return bag\_vector

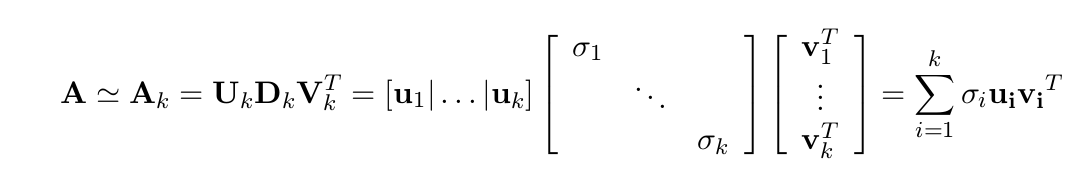
k najbardziej podobnych tekstów do tekstu wpisanego w konsoli

def k\_most\_similar\_texts\_normalize(k):  
 compare\_document = []  
  
 input\_bag\_of\_words = input\_to\_bag\_of\_words()  
 normalized\_input\_bag = normalize(input\_bag\_of\_words)  
 normalize\_matrix()  
 for i in range(1001):  
 similiar = cos(normalized\_matrix[i], normalized\_input\_bag)  
 t = (similiar, i)  
 compare\_document.append(t)  
 compare\_document.sort(key = lambda x: x[0], reverse = True)  
  
 print("the most similar texts:")  
  
 for i in range(k):  
 file\_name = 'text' + str(compare\_document[i][1]) + '.txt'  
 path = 'texts/' + file\_name  
 print("{0} similiarity: {1}".format(path, compare\_document[i][0]))

Wynik wywołania tej funkcji dla wprowadzonego fragmentu tekstu z pliku text500.txt



8.Usunięcie szumu



Funkcja zwracająca Ak

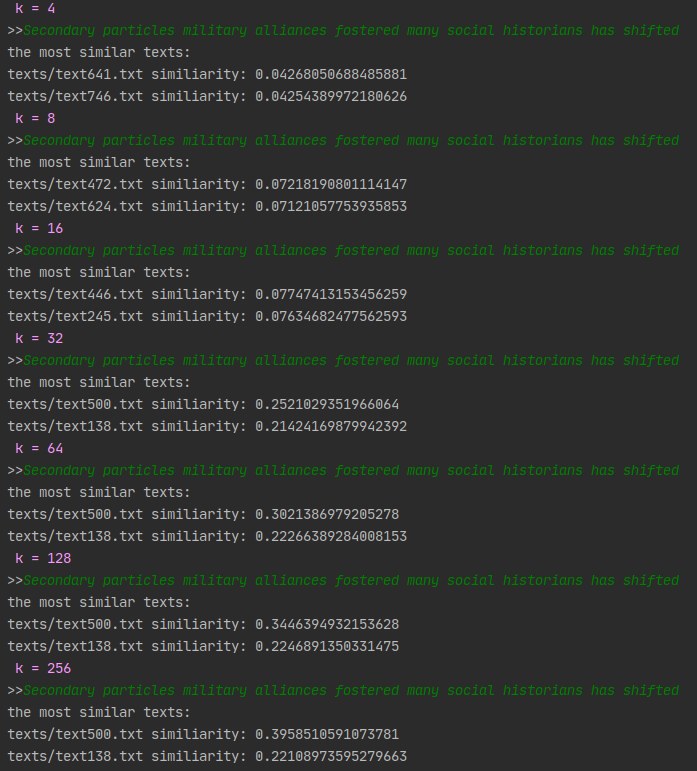
def svd\_matrix(k\_svd):  
 U, S, VT = sparse.linalg.svds(normalized\_matrix, k\_svd)  
 svd\_matrix = U @ numpy.diag(S) @ VT  
 return svd\_matrix

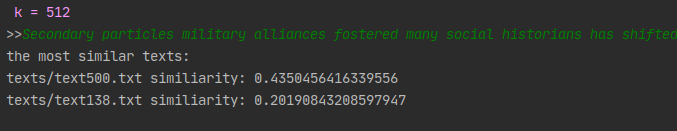
def k\_most\_similar\_texts\_noise\_removal(k, k\_svd):  
 compare\_document = []  
  
 input\_bag\_of\_words = input\_to\_bag\_of\_words()  
 normalized\_input\_bag = normalize(input\_bag\_of\_words)  
 normalize\_matrix()  
 svd = svd\_matrix(k\_svd)  
 for i in range(1001):  
 similiar = cos(svd[i], normalized\_input\_bag)  
 t = (similiar, i)  
 compare\_document.append(t)  
 compare\_document.sort(key=lambda x: x[0], reverse=True)  
  
 print("the most similar texts:")  
  
 for i in range(k):  
 file\_name = 'text' + str(compare\_document[i][1]) + '.txt'  
 path = 'texts/' + file\_name  
 print("{0} similiarity: {1}".format(path, compare\_document[i][0]))

9.Wykonano test, gdzie ponownie wprowadzono fragment pliku text500.txt  
a)bez IDF

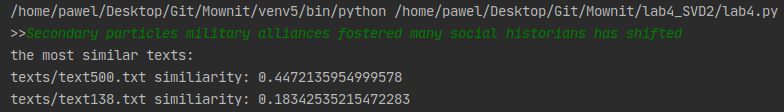
def k\_most\_similar\_texts\_noise\_removal\_test(k):  
 for pow in [2,3,4,5,6,7,8,9]:  
 print(f"{bcolors.HEADER} k = {2\*\*pow} {bcolors.ENDC}")  
 k\_most\_similar\_texts\_noise\_removal(k, 2\*\*pow)

Wynik testu:





Dla porównania wynik przed usunięciem szumu



**Wnioski**

Przy wprowadzeniu 10 słów z pliku text500.txt algorytm znalazł z nim największe podobieństwo przy już k=32. Im większe k tym lepiej był rozpoznawany plik źródłowy. Znaczy skok podobieństwa nastąpił między k=16, a k=32. Myślę, że dla najoptymalniejsze k jest między 50, a 100, bo wtedy jest dość szybki wzrost podobieństwa między zdaniem z tekstu, a samym tekstem.

b) z IDF

IDF faworyzują słowa, które występują w niewielu dokumentach. Słowa występujące rzadziej (np. te które są związane bardziej z jakąś dziedziną, są bardziej fachowe) bardziej upodobniają dwa dokumenty niż słowa pospolite. W powyższym teście można zauważyć, że IDF pomogło znależć szybciej podobieństwo między zdaniem z tekstu, a samym tekstem.