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 $A_{m \times n} = [\mathbf{d}_1 | \mathbf{d}_2 | \ldots | \mathbf{d}_n]$ 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

$$IDF(w) = \log \frac{N}{n_w},$$

```
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

$$\cos\theta_j = \frac{\mathbf{q}^T\mathbf{d}_j}{\|\mathbf{q}\|\|\mathbf{d}_j\|} = \frac{\mathbf{q}^T\mathbf{A}\mathbf{e}_j}{\|\mathbf{q}\|\|\mathbf{A}\mathbf{e}_j\|}$$

```
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

```
/home/pawel/Desktop/Git/Mownit/venv5/bin/python /home/pawel/Desktop/Git/Mownit/lab4_SVD2/lab4.py
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.4472135954999578
texts/text138.txt similiarity: 0.18342535215472283
```

8.Usunięcie szumu

$$\mathbf{A} \simeq \mathbf{A}_k = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^T = [\mathbf{u}_1|\dots|\mathbf{u}_k] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_k^T \end{bmatrix} = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

Funkcja zwracająca A_k

```
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:

```
 k = 4
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text641.txt similiarity: 0.04268050688485881
texts/text746.txt similiarity: 0.04254389972180626
 k = 8
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text472.txt similiarity: 0.07218190801114147
texts/text624.txt similiarity: 0.07121057753935853
 k = 16
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text446.txt similiarity: 0.07747413153456259
texts/text245.txt similiarity: 0.07634682477562593
 k = 32
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.2521029351966064
texts/text138.txt similiarity: 0.21424169879942392
 k = 64
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.3021386979205278
texts/text138.txt similiarity: 0.22266389284008153
 k = 128
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.3446394932153628
texts/text138.txt similiarity: 0.2246891350331475
 k = 256
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.3958510591073781
texts/text138.txt similiarity: 0.22108973595279663
```

```
 k = 512
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.4350456416339556
texts/text138.txt similiarity: 0.20190843208597947
```

Dla porównania wynik przed usunięciem szumu

```
/home/pawel/Desktop/Git/Mownit/venv5/bin/python /home/pawel/Desktop/Git/Mownit/lab4_SVD2/lab4.py
>>Secondary particles military alliances fostered many social historians has shifted
the most similar texts:
texts/text500.txt similiarity: 0.4472135954999578
texts/text138.txt similiarity: 0.18342535215472283
```

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 znaleźć szybciej podobieństwo między zdaniem z tekstu, a samym tekstem.