Все манипуляции с матрицами мы будем осуществлять с помощью numpy и scipy приводить слова к изначальной форме будем с помощью nltk. Устанавливаем…  
pip install numpy  
pip install nltk  
pip install scipy  
Если при попытке установить scipy возникнут какие то проблемы (будет требовать установить BLASS) то возможно поможет.  
apt-get install gfortran libopenblas-dev liblapack-dev  
  
  
Инициализация класса.

class LSI(object):

def \_\_init\_\_(self, stopwords, ignorechars, docs):

*# все слова которые встречаются в документах, и содержит номера документов в которых встречается каждое слово*

self.wdict = {}

*# dictionary - Ключевые слова в матрице слева содержит коды слов*

self.dictionary = []

*# слова которые исключаем из анализа типа и, в, на*

self.stopwords = stopwords

if type(ignorechars) == unicode: ignorechars = ignorechars.encode('utf-8')

self.ignorechars = ignorechars

*# инициализируем сами документы*

for doc in docs: self.add\_doc(doc)

Подготовка слов, пополняем словарь, если слово есть в словаре возвращаем его номер, предварительно очищаем от лишних символов и приводим в начальную форму

def dic(self, word, add = False):

if type(word) == unicode: word = word.encode('utf-8')

*# чистим от лишних символом*

word = word.lower().translate(None, self.ignorechars)

word = word.decode('utf-8')

*# приводим к начальной форме*

word = stemmer.stem(word)

*# если слово есть в словаре возвращаем его номер*

if word in self.dictionary: return self.dictionary.index(word)

else:

*# если нет и стоит флаг автоматически добавлять то пополняем словари возвращвем код слова*

if add:

*#self.ready = False*

self.dictionary.append(word)

return len(self.dictionary) - 1

else: return None

Построение исходной матрицы

def build(self):

*# убираем одиночные слова*

self.keys = [k for k in self.wdict.keys() if len(self.wdict[k]) > 0]

self.keys.sort()

*# создаём пустую матрицу*

self.A = zeros([len(self.keys), len(self.docs)])

*# наполняем эту матрицу*

for i, k in enumerate(self.keys):

for d in self.wdict[k]:

self.A[i,d] += 1

Построение остальных матриц

def calc(self):

""" Вычисление U, S Vt - матриц """

self.U, self.S, self.Vt = svd(self.A)

Нормализация веса или важности слов в матрице. Вычисляем важность термина в зависимости от его встречаемости. Например, слово «и» встречается достаточно часто, поэтому это слово будет иметь низкую значимость, а, скажем, слово «США» встречается значительно режем и, соответственно, будет иметь большую значимость. Стандартные обороты речи отсеиваются, а редкие термины остаются.

def TFIDF(self):

*# всего кол-во слов на документ*

wordsPerDoc = sum(self.A, axis=0)

*# сколько документов приходится на слово*

docsPerWord = sum(asarray(self.A > 0, 'i'), axis=1)

rows, cols = self.A.shape

for i in range(rows):

for j in range(cols):

self.A[i,j] = (self.A[i,j] / wordsPerDoc[j]) \* log(float(cols) / docsPerWord[i])

Сравнение документов на оси координат и поиск по ним.

def find(self, word):

self.prepare()

idx = self.dic(word)

if not idx:

print 'слово невстречается'

return []

if not idx in self.keys:

print 'слово отброшено как не имеющее значения которое через stopwords'

return []

idx = self.keys.index(idx)

print 'word --- ', word, '=', self.dictionary[self.keys[idx]], '.\n'

*# получаем координаты слова*

wx, wy = (-1 \* self.U[:, 1:3])[idx]

print 'word {}\t{:0.2f}\t{:0.2f}\t{}\n'.format(idx, wx, wy, word)

arts = []

xx, yy = -1 \* self.Vt[1:3, :]

for k, v in enumerate(self.docs):

*# получаем координаты документа*

ax, ay = xx[k], yy[k]

*#вычисляем расстояние между словом и документом*

dx, dy = float(wx - ax), float(wy - ay)

arts.append((k, v, ax, ay, sqrt(dx \* dx + dy \* dy)))

*# возвращаем отсортированный по расстоянию список*

return sorted(arts, key = lambda a: a[4])

ДЛЯ ЗАПУСКА С ФАЙЛОМ НУЖНО ПЕРЕБИТЬ в Вызов.py

docs = []

with open("имя.расш") as f:

for line in f:

data.append([float(x) for x in line.split()])

Если данные табллиа

with open("data.txt") as f:

    data = [map(float, row) for row in csv.reader(f, delimiter='\t')]