Упражнения по Python и Text-Mining

самостоятельное прорешивание.
ответы сразу после упражнения

Задание 1. Токенизация

1) инициализируем переменную ```text``` с текстом для токенизации
2) импортируем специальную функцию для токенизации из библиотеки ```nltk```
3) запишем результат токенизации в переменную ```words```
4) ```words``` - это массив, выведем 3 слово этого массива (2 номер, так как нумерация с нуля)

видим что это не слово, такова специфика работы ```word_tokenize```

что пунктуацию считает за отдельные слова, а попробуем теперь иным способом

5) давайте воспользуемся ```RegexpTokenizer```, для начала импортировав его из того же модуля
6) объявим экземпляр класса ```RegexpTokenizer```, с регулярным выражением "только слова" - это '\w+'
7) токенизируем и запишем результат в переменную ```words2```
8) выведем 3 элемент массива words2

Решение 1

1) text = 'To be, or not to be, that is the question'
2) from nltk.tokenize import word_tokenize
3) words = word_tokenize(text)
4) print words[2]
5) from nltk.tokenize import RegexpTokenizer
6) tokenizer = RegexpTokenizer(r'\w+')
7) words2 = tokenizer.tokenize(text)
8) print words2[2]

Задание 2. Самописный подсчет встречаемости слов во всех текстах из массиве текстов

0) импортируем RegexpTokenizer из прошлого примера и объявим такой же как в прошлом примере токенайзер

1) Объявим массив текстов
2) Объявим dict куда будем записывать результат по встречаемости (ключ - слово, значение встречаемость)
3) Напишем “шапку” цикла for , который ходит по всем текстам из массива текстов
4) Напишем “шапку” вложенного цикла по всем токенезированным словам
5) Увеличим счетчик встреченного слова, или инициализируем его как 0, если встретили первый раз
6) Выведем результат как есть

Решение 2

```
0)  from nltk.tokenize import RegexpTokenizer
    tokenizer = RegexpTokenizer(r'\w+')
1)  texts = ['to be, or not to be, that is the question', 'devoutly to be wished. to die, to sleep']
2)  word_to_number = {}
3)  for text in texts:
4)      for word in tokenizer.tokenize(text):
5)          word_to_number[word] = word_to_number.get(word, 0) + 1
6)  print word_to_number
```

Задание 3. По всем текстам сразу - не очень интересно. Посчитаем количество слов в по каждому тексту в отдельности. На этот раз воспользуемся библиотекой sklearn

1) Объявим массив текстов
2) Импортируем из библиотеки sklearn класс CountVectorizer
3) Объявим экземпляр класса CountVectorizer
4) Передав тексты в функцию объявленного экземпляра получим матрицу встречаймости каждого слова
5) Полученная матрица хранится как разреженная матрица, переведем ее в нормальный вид, записав в новую переменную
6) Встречаемость получили. Теперь чтобы красиво вывести таблицу, подготовим шапку таблицы. В words запишем массив слов, в специально порядке. (count_vect.vocabulary_ - это dict где слово и порядковый номер. мы превращаем этот dict в массив пар (tuple-ов) и сортируем по второму значению - это порядковый номер, а потом создаем лист только из слов - первых значений, порядковый номер не берем) это самая сложная функция - можно проконсультироваться со мной на занятии или в ФБ
7) Импортировать модуль pandas как pd
8) Отрисуем таблицу как pandas dataframe и передадим туда полученную шапку и саму матрицу matrix_counts

---

Решение 3

1) texts = ['to be, or not to be, that is the question', 'devoutly to be wished. to die, to sleep']
2) from sklearn.feature_extraction.text import CountVectorizer
3) count_vect = CountVectorizer()
4) temp_matrix = count_vect.fit_transform(texts)
5) matrix_counts = temp_matrix.toarray()
6) words = [x[0] for x in sorted(count_vect.vocabulary_.items(), key=lambda x: x[1])]
7) import pandas as pd
   pd.DataFrame(matrix_counts, columns=words)

Задание 4. Создадим похожу таблицу но только со значениями TF-IDF

1) Импортируем из библиотеки sklearn класс TfidfVectorizer
2) Инициализируем массив текстов
3) Объявим экземпляр класса TfidfVectorizer
4) Передав тексты в функцию объявленного экземпляра получим матрицу tf-idf каждого слова
5) Полученная матрица хранится как разреженная матрица, переведем ее в нормальный вид, записав в новую переменную
6) Аналогично выше построим шапку
7) Импортировать модуль pandas как pd
8) Отрисуем таблицу как pandas dataframe и передадим туда полученную шапку и саму матрицу matrix_counts

---

Решение 4

```
1) from sklearn.feature_extraction.text import TfidfVectorizer
2) texts = ['to be, or not to be, that is the question', 'devoutly to be wished. to die, to sleep']
3) vectorizer = TfidfVectorizer(norm=None, smooth_idf=False)
4) temp_matrix = vectorizer.fit_transform(texts)
5) matrix_counts = temp_matrix.toarray()
6) words = [x[0] for x in sorted(vectorizer.vocabulary_.items(), key=lambda x: x[1])]
7) import pandas as pd
8) pd.DataFrame(matrix_counts, columns=words)
```