DeepSeek

Краткое описание (как ты любишь):

Этот код анализирует текст, находит похожие предложения и пытается их классифицировать. Как если бы ты сравнивал речи Наполеона и Ленина на предмет упоминания кошек! 😸

Разберём по частям:

1. Функция `get_class(sent)` - наш "детектор кошек"

```
def get_class(sent):
    if 'cat' in sent or 'cats' in sent:
        if 'unix' in sent or 'command' in sent or 'utility' in sent:
            return "Unix утилита cat"
        elif 'os' in sent or 'apple' in sent or 'mac' in sent or 'os x' in sent:
            return "Версии OS Apple"
        else:
            return "Упоминание кошек"
    return "Не определен"
```

Это как историк определяет, о чём документ:

- Видит слово "кот" → начинает расследование 🕵️‍♂️
- Если рядом слова про Unix → это команда `cat` (а не животное)
- Если рядом Apple/Mac → это версия ОС (Cat-alina, Tiger и т.д.)
- Иначе → обычные котики (ура!)

Пример из жизни: Это как если итальянец скажет "gatto" — нужно понять, говорит он о пицце "Quattro gatti" или о компьютерной программе!

2. Чтение файла

```
with open('sentences.txt', 'r') as file:
    data_list = file.readlines()
```

Открываем файл со предложениями, как архив с историческими документами.

3. Создание списка уникальных слов

```
words = []
for sentence in data_list:
    sentence_words = re.findall(r"\b[a-zA-Z]+\b", sentence.lower())
    for word in sentence_words:
        if word not in words:
            words.append(word)
```

Здесь мы:

1. Разбиваем предложения на слова (как историк разбирает летопись на термины)

2. Приводим к нижнему регистру (чтобы "Кот" и "кот" были одним словом)

3. Собираем уникальные слова — наш "словарь древнего языка"

4. Создание словаря {слово: индекс}

```
word_dict = {word: idx for idx, word in enumerate(words)}
```

Это как если бы мы пронумеровали все слова в словаре Даля. Слово "революция" → строка 42, "монархия" → строка 13 и т.д.

5. Создание матрицы встречаемости

```
matrix = np.zeros((n, d), dtype=int)
for i, sentence in enumerate(data_list):
    sentence_words = re.findall(r'\b[a-zA-Z]+\b', sentence.lower())
    for word in sentence_words:
        j = word_dict[word]
        matrix[i, j] += 1
```

Здесь мы создаём "таблицу цитирования":

- Строки — предложения
- Столбцы — слова
- Число в ячейке — сколько раз слово встретилось в предложении

Аналогия: Как если бы мы отмечали, в каких исторических документах встречается слово "король" (1 раз у Шекспира, 0 раз у Маркса, 3 раза у Дюма).

6. Вычисление косинусных расстояний

```
distances = []
for i in range(n):
    for j in range(i + 1, n):
        dist = distance.cosine(matrix[i], matrix[j])
        distances.append((i, j, dist))
```

Это самая "научная" часть! Мы сравниваем, насколько предложения похожи по набору слов.

Объяснение на примере: Допустим, у нас есть два предложения:

1. "Кот ест рыбу" (вектор: [1, 1, 1, 0, 0])
2. "Кот спит на диване" (вектор: [1, 0, 0, 1, 1])

Расстояние покажет, насколько они похожи. Чем меньше число — тем более "родственные" предложения, как французский и итальянский языки!

7. Вывод топ-5 похожих пар

```
top_5 = sorted(distances, key=get_distance)[:5]
for pair in top_5:
    i, j, dist = pair
```

Здесь мы:

1. Сортируем пары по "близости" (как историк ранжирует исторические периоды по схожести)
2. Берём 5 самых "похожих" пар
3. Проверяем, совпадают ли их классификации (из функции `get_class`)

Итог (как в историческом докладе):

Этот код — как умный архивариус, который:

1. Читает все документы (предложения)
2. Составляет каталог терминов (уникальные слова)
3. Анализирует, какие документы похожи (косинусное расстояние)
4. И пытается их классифицировать (наша функция про кошек)

Шутка напоследок: Если бы этот код анализировал русскую историю, он бы путал "КПСС" и "КПЗ" (кошачий приют для зверей) — оба содержат "КП"! 😄

Теперь ты, как будущий историк, можешь объяснить этот код даже своей бабушке! Ну или как минимум своему коту, если он интересуется Python. 🐍