**Методические указания к лабораторной работе №2**

В данной работе мы продолжаем работать с библиотекой scikit-learn (<http://scikit-learn.org>), и хотим выяснить ее возможности при работе с текстовыми документами.

Ниже приведены новые модули, которые будут использованы в данной работе.

fetch\_20newsgroups - <http://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html> - загружает датасет «20 news groups», состоящий приблизительно из 18000 сообщений на английском языке по 20 тематикам, разбитым на обучающую и тестовую выборки.

Векторизаторы текста:

CountVectorizer - <http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>

TfidfTransformer - <http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer>

Pipeline - <http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html> - конвейерный классификатор

MultinominalNB - <http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html> - Полиномиальный (Мультиномиальный) Наивный Байесовский метод – разновидность Наивного Байесовского метода, которая хорошо работает с текстами, длины которых сильно варьируются[[1]](#footnote-1).

**Загрузка выборки**

Данные в датасете «20 news groups» имеют следующий вид:

From: af774@cleveland.Freenet.Edu (Chad Cipiti)

Subject: Good shareware paint and/or animation software for SGI?

Organization: Case Western Reserve University, Cleveland, OH (USA)

Lines: 15

Reply-To: af774@cleveland.Freenet.Edu (Chad Cipiti)

NNTP-Posting-Host: hela.ins.cwru.edu

Does anyone know of any good shareware animation or paint software for an SGI machine? I've exhausted everyplace on the net I can find and still don't hava a nice piece of software.

Thanks alot!

Chad

--

Knock, knock. Chad Cipiti

Who's there? af774@cleveland.freenet.edu

 cipiti@bobcat.ent.ohiou.edu

It might be Heisenberg. chad@voxel.zool.ohiou.edu

Сообщение состоит из заголовка (header), основной части, подписи или сноски (footer), а также может содержать в себе цитирование предыдущего сообщения (quotes).

Модуль fetch\_20newsgroups позволяет выбирать интересующие тематики и удалять ненужные части сообщений. Для того чтобы выбрать сообщения по интересующим тематикам, необходимо передать список тематик в параметр categories. Для того чтобы удалить ненужные части сообщений, нужно передать их в параметр remove. Кроме того, важным параметром fetch\_20newsgroups является subset - тип выборки – обучающая или тестовая.

Выберем сообщения по тематикам «Атеизм» и «Компьютерная графика», а также укажем, что нас не интересуют заголовки, цитаты и подписи:

|  |
| --- |
| categories = ['alt.atheism', 'comp.graphics'] remove = ('headers', 'footers', 'quotes')twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True, random\_state=42, categories = categories, remove = remove )twenty\_test = fetch\_20newsgroups(subset='test', shuffle=True, random\_state=42, categories = categories, remove = remove ) |

Возвращаемый набор данных — это scikit-learn совокупность: одномерный контейнер с полями, которые могут интерпретироваться как признаки объекта (object attributes). Например, target\_names содержит список названий запрошенных категорий, target - тематику сообщения, а data – непосредственно текст сообщения:

|  |
| --- |
| print (twenty\_train.data[2]) |

Does anyone know of any good shareware animation or paint software for an SGI machine? I've exhausted everyplace on the net I can find and still don't hava a nice piece of software.

Thanks alot!

Chad

**Векторизация**

Чтобы использовать машинное обучение на текстовых документах, первым делом, нужно перевести текстовое содержимое в числовой вектор признаков.

Предобработка текста, токенизация и отбрасывание стоп-слов включены в состав модуля CountVectorizer, который позволяет создать словарь характерных признаков и перевести документы в векторы признаков:

|  |
| --- |
| vect = CountVectorizer(max\_features = 10000, stop\_words = 'english') |

Создадим объект-векторизатор vect со следующими параметрами:

max\_features = 10000 - количество наиболее частотных терминов, из которых будет состоять словарь

stop\_words = 'english' – на данный момент модулем поддерживается отсечение английских стоп-слов. Кроме того, здесь можно указать список стоп-слов вручную. Если параметр не указывать, будут использованы все термины словаря.

Также, в работе может потребоваться настройка следующих параметров:

max\_df -float в диапазоне [0.0, 1.0] или int, по умолчанию = 1.0. При построении словаря игнорирует термины, частота которых в документе строго превышает заданный порог (стоп-слова для конкретного корпуса). Если float, параметр обозначает долю документов, если целое число – то абсолютное значение.

min\_df - float в диапазоне [0.0, 1.0] или int, по умолчанию = 1.0. При построении словаря игнорируйте термины, частота которых в документе строго ниже заданного порога. В литературе это значение также называется порогом. Если float, параметр обозначает долю документов, если целое число – то абсолютное значение.

После того как объект-векторизатор создан, необходимо создать словарь характерных признаков с помощью метода fit() и перевести документы в векторы признаков c помощью метода transform(), подав на него обучающую выборку:

|  |
| --- |
| vect.fit(twenty\_train.data)train\_data = vect.transform(twenty\_train.data)test\_data = vect.transform(twenty\_test.data) |

Также, можно отметить что эти два действия могут быть объединены одним методом fit\_transform(). Однако, в этом случае нужно учесть, что для перевода тестовой выборки в вектор признаков, по-прежнему нужно использовать метод transform().

|  |
| --- |
| train\_data = vect.fit\_transform(twenty\_train.data)test\_data = vect.transform(twenty\_test.data) |

Если для тестовых данных также воспользоваться методом fit\_transform(), это приведет к перестроению словаря признаков и неправильным результатам классификации.

Следующий блок кода позволит вывести первые 10 терминов, упорядоченных по частоте встречаемости:

|  |
| --- |
| x = list(zip(vect.get\_feature\_names(), np.ravel(train\_data.sum(axis=0))))def SortbyTF(inputStr): return inputStr[1]x.sort(key=SortbyTF, reverse = True)print (x[:10])) |

**Стемминг**

Существует целый ряд алгоритмов стемминга. В работе предлагается использовать алгоритм Портера, реализация которого приведена в библиотеке nltk <https://www.nltk.org/>

Для проведения стемминга нужно создать объект PorterStemmer().Стеммер работает таким образом: у созданного объекта PorterStemmer есть метод stem, производящий стемминга. Таким образом, необходимо каждую из частей выборки (обучающую и тестовую) разбить на отдельные документы, затем, проходя в цикле по каждому слову в документе, произвести стемминг и объединить эти слова в новый документ.

|  |
| --- |
| from nltk.stem import \*from nltk import word\_tokenizeporter\_stemmer = PorterStemmer()stem\_train = []for text in twenty\_train.data: nltk\_tokens = word\_tokenize(text) line = '' for word in nltk\_tokens: line += ' ' + porter\_stemmer.stem(word) stem\_train.append(line)print (stem\_train[0]) |

**TF- и TF-IDF взвешивание**

CountVectorizer позволяет лишь определять частоту встречаемости термина во всей выборке, но такой подход к выявлению информативных терминов не всегда дает качественный результат. На практике используют более продвинутые способы, наибольшее распространение из которых получили TF- и TF-IDF взвешивания.

Воспользуемся методом fit() класса TfidfTransformer(), который переводит матрицу частот встречаемости в TF- и TF-IDF веса.

|  |
| --- |
| tfidf = TfidfTransformer(use\_idf = True).fit(train\_data)train\_data\_tfidf = tfidf.transform(train\_data) |

Отметим, что в метод fit() нужно передавать не исходные текстовые данные, а вектор слов и их частот, полученный с помощью метода transform() класса CountVectorizer.

Для того, чтобы получить tf-idf значения, необходимо установить параметр use\_idf = True, в противном случае на выходе мы получим значения tf

**Классификация**

После того как мы провели векторизацию текста, обучение модели и классификация для текстовых данных выглядит абсолютно идентично классификации объектов в первой лабораторной работе.

Задача обучения модели заключается не только в выборе подходящих данных обучающей выборки, способных качественно охарактеризовать объекты, но и в настройке многочисленных параметров метода классификации, предварительной обработке данных и т.д.

Рассмотрим, какие возможности предлагаются в библиотеке scikit-learn для автоматизации и упрощения данной задачи.

**Pipeline**

Чтобы с цепочкой vectorizer => transformer => classifier было проще работать, в scikit-learn есть класс Pipeline (конвейер), который функционирует как составной (конвейерный) классификатор.

|  |
| --- |
| from sklearn.pipeline import Pipeline |

Промежуточными шагами конвейера должны быть преобразования, то есть должны выполняться методы fit() и transform(), а последний шаг – только fit().

При этом, pipeline позволяет устанавливать различные параметры на каждом своем шаге. Таким образом, проделанные нами действия по векторизации данных, взвешиванию с помощью TF-IDF и классификации методом К-БС с использованием pipeline будут выглядеть следующим образом:

|  |
| --- |
| text\_clf = Pipeline([('vect', CountVectorizer(max\_features= 1000, stop\_words = 'english')), ('tfidf', TfidfTransformer(use\_idf = True)), ('clf', KNeighborsClassifier (n\_neighbors=1)),])  |

Названия vect, tfidf и clf выбраны нами произвольно. Мы рассмотрим их использование в следующей лабораторной работе. Теперь обучим модель с помощью всего 1 команды:

|  |
| --- |
| text\_clf = text\_clf.fit(twenty\_train.data, twenty\_train.target) |

И проведем классификацию на тестовой выборке:

|  |
| --- |
| prediction = text\_clf.predict(twenty\_test.data) |

1. <https://logic.pdmi.ras.ru/~sergey/teaching/mlstc12/sem01-naivebayes.pdf> [↑](#footnote-ref-1)