Специализация "Машинное обучение и анализ данных"

Автор материала: программист-исследователь Mail.ru Group, старший преподаватель Факультета Компьютерных Наук ВШЭ Юрий Кашницкий

ДЕМОНСТРАЦИОННЫЙ ПРОЕКТ

Идентификация пользователей по посещенным веб-страницам

Часть 1.

Цели и задачи проекта, описание исходных и обработанных данных и признаков.

1.1. Цели и задачи проекта

В этом проекте решается задача идентификации пользователя по его поведению в сети Интернет. Это сложная и интересная задача на стыке анализа данных и поведенческой психологии. В качестве примера, компания Яндекс решает задачу идентификации взломщика почтового ящика по его поведению. Если в двух словах, взломщик будет себя вести не так, как владелец ящика: он может не удалять сообщения сразу по прочтении, как это делал хозяин, он будет по-другому ставить флажки сообщениям и даже по-своему двигать мышкой. Тогда такого злоумышленника можно идентифицировать и "выкинуть" из почтового ящика, предложив хозяину войти по SMS-коду. Этот пилотный проект описан в статье на Хабрахабре. Похожие вещи делаются, например, в Google Analytics и описываются в научных статьях, найти можно многое по фразам "Traversal Pattern Mining" и "Sequential Pattern Mining".

Мы будем решать похожую задачу: по последовательности из нескольких веб-сайтов, посещенных подряд одним и тем же человеком, мы будем идентифицировать этого человека. **Идея такая**: пользователи Интернета по-разному переходят по ссылкам, и это может помогать

их идентифицировать (кто-то сначала в почту, потом про футбол почитать, затем новости, контакт, потом наконец – работать, кто-то – сразу работать).

1.2. Описание исходных данных

Будем использовать данные из статьи "A Tool for Classification of Sequential Data". И хотя мы не можем рекомендовать эту статью (описанные методы далеки от state-of-the-art, лучше обращаться к книге "Frequent Pattern Mining" и последним статьям с ICDM), но данные там собраны аккуратно и представляют интерес.

Имеются данные с прокси-серверов Университета Блеза Паскаля, они имеют очень простой вид. Для каждого пользователя заведен csv-файл с названием **user****.csv** (где вместо звездочек – 4 цифры, соответствующие ID пользователя), а в нем посещения сайтов записаны в следующем формате:

timestamp (время), посещенный веб-сайт

Скачать исходные данные можно по ссылке в статье, там же описание. На первом этапе мы будем разрабатывать и тестировать необходимые программы для идентификации пользователе. Поэтому нам удобнее будет использовать данные не по всем 3000 пользователям, а по 10 и 150. Ссылка на архив *capstone_user_identification* (~7 Mb, в развернутом виде ~ 60 Mb).

Многие вычисления достаточно ресурсозатратны и требуют много времени, поэтому мы будем использовать параллельно 2 выборки: по 10 пользователям и по 150. Для 10 пользователей будем писать и отлаживать код, для 150 – будет рабочая версия.

Создадим три каталога:

- В каталоге 10users лежат 10 csv-файлов с названием вида "user[USER_ID].csv", где [USER_ID] – ID пользователя;
- Аналогично для каталога 150users – там 150 файлов;
- В каталоге 3users – игрушечный пример из 3 файлов, это для отладки кода предобработки, который мы напишем в ходе проекта.

На финальной стадии проекта мы продемонстрируем разработанные возможности идентификации в ходе публичного соревнования Kaggle Inclass, которое организовано специально для подобных предсказательных систем.

1.3. Подготовка данных к анализу и построению моделей

Первая часть проекта посвящена подготовке данных для дальнейшего описательного анализа и построения прогнозных моделей. Будет написан код для предобработки данных (исходно посещенные веб-сайты указаны для каждого пользователя в отдельном файле) и формирования единой обучающей выборки. Также в этой части мы применим разреженный формат данных (матрицы `Scipy.sparse`), который хорошо подходит для данной задачи:

- Подготовка обучающей выборки
- Работа с разреженным форматом данных

Используемые инструменты (со ссылками):

- Циклы, функции, генераторы, list comprehension

- Чтение данных из файлов
- Запись файлов, изменение файлов
- Pandas.DataFrame
- Pandas. Индексация и селекция
- **Кроме того, будут использоваться библиотеки Python `glob`, `pickle` и класс `csr_matrix` из `Scipy.sparse`.**

Приведем список версий основных используемых в проекте библиотек: NumPy, SciPy, Pandas, Matplotlib, Statsmodels и Scikit-learn. Для этого воспользуемся расширением watermark.

In [2]:

```
# pip install watermark
%load_ext watermark
```

In [3]:

```
%watermark -v -m -p numpy,scipy,pandas,matplotlib,statsmodels,sklearn -g
```

```
CPython 3.6.3
IPython 6.1.0

numpy 1.13.3
scipy 0.19.1
pandas 0.20.3
matplotlib 2.1.0
statsmodels 0.8.0
sklearn 0.19.1

compiler   : MSC v.1900 64 bit (AMD64)
system     : Windows
release    : 10
machine    : AMD64
processor  : Intel64 Family 6 Model 142 Stepping 10, GenuineIntel
CPU cores  : 8
interpreter: 64bit
Git hash   : e890eadd0386859d086a09bfc6fd92576557b899
```

In [1]:

```
from glob import glob
import os
import pickle
#pip install tqdm
from tqdm import tqdm_notebook
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
```

Посмотрим на один из файлов с данными о посещенных пользователем (номер 31) веб-страницах.

In [2]:

```
# Путь к данным
PATH_TO_DATA = ('../data' )
```

In [3]:

```
user31_data = pd.read_csv(os.path.join(PATH_TO_DATA,
                                       '10users/user0031.csv'))
```

In [4]:

```
user31_data.head()
```

Out[4]:

	timestamp	site
0	2013-11-15 08:12:07	fpdownload2.macromedia.com
1	2013-11-15 08:12:17	laposte.net
2	2013-11-15 08:12:17	www.laposte.net
3	2013-11-15 08:12:17	www.google.com
4	2013-11-15 08:12:18	www.laposte.net

Поставим задачу классификации: идентифицировать пользователя по сессии из 10 подряд посещенных сайтов. Объектом в этой задаче будет сессия из 10 сайтов, последовательно посещенных одним и тем же пользователем, признаками – индексы этих 10 сайтов (чуть позже здесь появится "мешок" сайтов, подход Bag of Words). Целевым классом будет id пользователя.

Пример для иллюстрации

Пусть пользователя всего 2, длина сессии – 2 сайта.

user0001.csv

timestamp	site
00:00:01	vk.com
00:00:11	google.com
00:00:16	vk.com
00:00:20	yandex.ru

user0002.csv

timestamp	site
00:00:02	yandex.ru
00:00:14	google.com

00:00:17	facebook.com
00:00:25	yandex.ru

Идем по 1 файлу, нумеруем сайты подряд: vk.com – 1, google.com – 2 и т.д. Далее по второму файлу.

Отображение сайтов в их индесы должно получиться таким:

site	site_id
vk.com	1
google.com	2
yandex.ru	3
facebook.com	4

Тогда обучающая выборка будет такой (целевой признак – user_id):

session_id	**site1**	**site2**	**user_id**
1	1	2	1
2	1	3	1
3	3	2	2
4	4	3	2

Здесь 1 объект – это сессия из 2 посещенных сайтов 1-ым пользователем (target=1). Это сайты vk.com и google.com (номер 1 и 2). И так далее, всего 4 сессии. Пока сессии у нас не пересекаются по сайтам, то есть посещение каждого отдельного сайта относится только к одной сессии.

1.4. Подготовка обучающей выборки

Реализуем функцию *prepare_train_set*, которая принимает на вход путь к каталогу с csv-файлами *path_to_csv_files* и параметр *session_length* – длину сессии, а возвращает 2 объекта:

- DataFrame, в котором строки соответствуют уникальным сессиям из *session_length* сайтов, *session_length* столбцов – индексам этих *session_length* сайтов и последний столбец – ID пользователя
- частотный словарь сайтов вида {'site_string': [site_id, site_freq]}, например для недавнего игрушечного примера это будет {'vk.com': (1, 2), 'google.com': (2, 2), 'yandex.ru': (3, 3), 'facebook.com': (4, 1)}

Детали:

- Для обхода файлов в каталоге будем использовать `glob`. Список файлов можно также отсортировать лексикографически. Для удобства применим `tqdm_notebook` для отслеживания числа выполненных итераций цикла
- Создадим частотный словарь уникальных сайтов (вида {'site_string': (site_id, site_freq)}) и заполним его по ходу чтения файлов (начнем с 1)
- По возможности меньшие индексы будем давать более часто попадающимся сайтам (приниципп наименьшего описания)
- Пока для упрощения не будем делать entity recognition, то есть будем считать *google.com*, *http://www.google.com* и *www.google.com* разными сайтами (подключить entity recognition можно будет в дальнейшем)
- Скорее всего в файле число записей не кратно числу *session_length*. Тогда последняя сессия будет короче. Остаток заполним нулями. То есть если в файле 24 записи и сессии длины 10, то 3 сессия будет состоять из 4 сайтов, и ей мы сопоставим вектор [*site1_id*, *site2_id*, *site3_id*, *site4_id*, 0, 0, 0, 0, 0, 0, *user_id*]
- В итоге некоторые сессии могут повторяться – оставим как есть, без удаления дубликатов. Если в двух сессиях все сайты одинаковы, но сессии принадлежат разным пользователям, то тоже оставим как есть, это естественная неопределенность в данных
- Не будем оставлять в частотном словаре сайт 0 (уже в конце, когда функция возвращает этот словарь).

In [5]:

```
def prepare_train_set(path_to_csv_files, session_length=10):
    '''
    Функция принимает на вход путь к каталогу с csv-файлами path_to_csv_fil
es и параметр session_length - длину сессии,
    а возвращает 2 объекта:
        - DataFrame, в котором строки соответствуют уникальным сессиям из s
ession_length сайтов, session_length столбцов -
        индексам этих session_length сайтов и последний столбец - ID пользо
вателя;
        - частотный словарь сайтов вида {'site_string': [site_id, site_freq
]}, например для недавнего игрушечного примера это
        будет {'vk.com': (1, 2), 'google.com': (2, 2), 'yandex.ru': (3, 3),
'facebook.com': (4, 1)}
    '''
    # инициализация
    freq_vocabulary = {} # частотный словарь

    # заготовка для массива уникальных сессий
    array_toy = np.zeros(session_length + 1).reshape(1, session_length + 1)

    # подготовим пути для обхода и обойдем:
    for path in tqdm_notebook(glob(os.path.join(path_to_csv_files, 'user*.c
sv'))):

        user_id = int(path[-8:-4]) # id пользователя

        # считаем файл
        data_readed = pd.read_csv(path, header=0, index_col=0)
        data = data_readed['site'].values.copy() # и скопируем его в numpy-
массив

        # формируем частотный словарь и сразу перекодируем сайты в их site_
id
```

```
        data_site_ids = []
        for site_string in data:
            site_id = freq_vocabulary.setdefault(site_string, [len(freq_voca
bulary) + 1, 0])[0]
            freq_vocabulary[site_string][1] += 1
            data_site_ids.append(site_id)

        # переведем в numpy-массив, чтобы применить resize
        data_site_ids = np.array(data_site_ids)

        # формируем массив уникальных сессий через resize

        row = int(len(data_site_ids) / session_length) + 1 if len(data_site_
ids) % session_length != 0 \
                                                           else int(len
ta_site_ids) / session_length)
        data_site_ids.resize((row, session_length))



        # добавим колонку с user_id
        col_user_id = np.zeros([len(data_site_ids), 1]) + user_id # forming
columns with user_id
        data_site_ids = np.hstack((data_site_ids, col_user_id))

        # накапливаем сессии
        array_toy = np.vstack((array_toy, data_site_ids))

    # готовим датафрейм сессий на выдачу
    data_toy = pd.DataFrame(array_toy[1:]) # убираем 1 нулевую строку

    data_toy.columns = ['site{}'.format(i) for i in range(1, session_length
+ 1)] + ['user_id']

    return data_toy.applymap(int), freq_vocabulary
```

Применим полученную функцию к игрушечному примеру, чтобы убедиться, что все работает как надо.

In [6]:

```
!cat $PATH_TO_DATA/3users/user0001.csv
```

```
timestamp,site
2013-11-15 09:28:17,vk.com
2013-11-15 09:33:04,oracle.com
2013-11-15 09:52:48,oracle.com
2013-11-15 11:37:26,geo.mozilla.org
2013-11-15 11:40:32,oracle.com
2013-11-15 11:40:34,google.com
2013-11-15 11:40:35,accounts.google.com
2013-11-15 11:40:37,mail.google.com
2013-11-15 11:40:40,apis.google.com
2013-11-15 11:41:35,plus.google.com
2013-11-15 12:40:35,vk.com
2013-11-15 12:40:37,google.com
2013-11-15 12:40:40,google.com
2013-11-15 12:41:35,google.com
```

In [7]:

```
!cat $PATH_TO_DATA/3users/user0002.csv
```

```
timestamp,site
2013-11-15 09:28:17,vk.com
2013-11-15 09:33:04,oracle.com
2013-11-15 09:52:48,football.kulichki.ru
2013-11-15 11:37:26,football.kulichki.ru
2013-11-15 11:40:32,oracle.com
```

In [8]:

```
!cat $PATH_TO_DATA/3users/user0003.csv
```

```
timestamp,site
2013-11-15 09:28:17,meduza.io
2013-11-15 09:33:04,google.com
2013-11-15 09:52:48,oracle.com
2013-11-15 11:37:26,google.com
2013-11-15 11:40:32,oracle.com
2013-11-15 11:40:34,google.com
2013-11-15 11:40:35,google.com
2013-11-15 11:40:37,mail.google.com
2013-11-15 11:40:40,yandex.ru
2013-11-15 11:41:35,meduza.io
2013-11-15 12:28:17,meduza.io
2013-11-15 12:33:04,google.com
2013-11-15 12:52:48,oracle.com
```

In [9]:

```
train_data_toy, site_freq_3users = prepare_train_set(os.path.join(PATH_TO_D
ATA, '3users'),
                                                     session_length=10)
```

In [10]:

```
train_data_toy
```

Out[10]:

	site1	site2	site3	site4	site5	site6	site7	site8	site9	site10	user_id
0	1	2	2	3	2	4	5	6	7	8	1
1	1	4	4	4	0	0	0	0	0	0	1
2	1	2	9	9	2	0	0	0	0	0	2
3	10	4	2	4	2	4	4	6	11	10	3
4	10	4	2	0	0	0	0	0	0	0	3

In [11]:

```
site_freq_3users
```

Out[11]:

```
{'accounts.google.com': [5, 1],
 'apis.google.com': [7, 1],
 'football.kulichki.ru': [9, 2],
 'geo.mozilla.org': [3, 1],
 'google.com': [4, 9],
 'mail.google.com': [6, 2],
 'meduza.io': [10, 3],
 'oracle.com': [2, 8],
 'plus.google.com': [8, 1],
 'vk.com': [1, 3],
 'yandex.ru': [11, 1]}
```

1. Применим полученную функцию к данным по 10 пользователям, посмотрите, сколько уникальных сессий из 10 сайтов получится.

In [12]:

```
train_data_10users, site_freq_10users = prepare_train_set(os.path.join(PATH
_TO_DATA, '10users'),
                                                          session_length=10
```

In [13]:

```
len(train_data_10users)
```

Out[13]:

```
14061
```

2. Посмотрим, сколько всего уникальных сайтов в выборке из 10 пользователей:

In [14]:

```
len(site_freq_10users)
```

Out[14]:

```
4913
```

3. Применим полученную функцию к данным по 150 пользователям и посмотрим, сколько уникальных сессий из 10 сайтов получится.

In [15]:

```
%%time
train_data_150users, site_freq_150users =
prepare_train_set(os.path.join(PATH_TO_DATA, '150users'),

session_length=10)
```

```
Wall time: 9.88 s
```

4. Посмотрим, сколько всего уникальных сайтов в выборке из 150 пользователей:

In [16]:

```
len(site_freq_150users)
```

Out[16]:

```
27797
```

5. Выведим топ-10 самых популярных сайтов среди посещенных 150 пользователями.

In [17]:

```
top10_popular = list(map(lambda x: x[0], sorted(site_freq_150users.items(),
key=lambda x: x[1][1], reverse=True)))[:10]
```

In [18]:

```
top10_popular
```

Out[18]:

```
['www.google.fr',
 'www.google.com',
 'www.facebook.com',
 'apis.google.com',
 's.youtube.com',
 'clients1.google.com',
 'mail.google.com',
 'plus.google.com',
 'safebrowsing-cache.google.com',
 'www.youtube.com']
```

Для дальнейшего анализа запишем полученные объекты DataFrame в csv-файлы.

In [32]:

```
train_data_10users.to_csv(os.path.join(PATH_TO_DATA,
                                       'train_data_10users.csv'),
                          index_label='session_id', float_format='%d')
train_data_150users.to_csv(os.path.join(PATH_TO_DATA,
                                        'train_data_150users.csv'),
                           index_label='session_id', float_format='%d')
```

1.5. Работа с разреженным форматом данных

Если так подумать, то полученные признаки *site1*, ..., *site10* смысла не имеют как признаки в задаче классификации. А вот если воспользоваться идеей мешка слов из анализа текстов – это другое дело. Создадим новые матрицы, в которых строкам будут соответствовать сессии из 10 сайтов, а столбцам – индексы сайтов. На пересечении строки i и столбца j будет стоять число n_{ij} – сколько раз сайт j встретился в сессии номер i. Делать это будем с помощью разреженных матриц Scipy – csr_matrix. Создадим такие матрицы для наших данных. Сначала проверем на игрушечном примере, затем применим для 10 и 150 пользователей.

Обратим внимание, что в коротких сессиях, меньше 10 сайтов, у нас остались нули, так что первый признак (сколько раз попался 0) по смыслу отличен от остальных (сколько раз попался сайт с индексом i). Поэтому первый столбец разреженной матрицы удалим.

In [34]:

```
X_toy, y_toy = train_data_toy.iloc[:, :-1].values, train_data_toy.iloc[:, -1
].values
```

In [35]:

```
X_toy
```

Out[35]:

```
array([[ 1,  2,  2,  3,  2,  4,  5,  6,  7,  8],
       [ 1,  4,  4,  4,  0,  0,  0,  0,  0,  0],
       [ 1,  2,  9,  9,  2,  0,  0,  0,  0,  0],
       [10,  4,  2,  4,  2,  4,  4,  6, 11, 10],
       [10,  4,  2,  0,  0,  0,  0,  0,  0,  0]], dtype=int64)
```

In [60]:

```
X_sparse_toy = csr_matrix(((np.ones(X_toy.size),
                            X_toy.ravel(),
                            np.arange(X_toy.shape[0] + 1) * X_toy.shape[1]))
dtype=int)[:, 1:]
```

Размерность разреженной матрицы должна получиться равной 11, поскольку в игрушечном примере 3 пользователя посетили 11 уникальных сайтов.

In [61]:

```
X_sparse_toy.todense()
```

Out[61]:

```
matrix([[1, 3, 1, 1, 1, 1, 1, 1, 0, 0, 0],
        [1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0],
        [1, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0],
        [0, 2, 0, 4, 0, 1, 0, 0, 0, 2, 1],
        [0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0]], dtype=int32)
```

In [62]:

```
X_10users, y_10users = train_data_10users.iloc[:, :-1].values, \
                       train_data_10users.iloc[:, -1].values
X_150users, y_150users = train_data_150users.iloc[:, :-1].values, \
                         train_data_150users.iloc[:, -1].values
```

In [63]:

```
X_sparse_10users = csr_matrix(((np.ones(X_10users.size),
                            X_10users.ravel(),
                            np.arange(X_10users.shape[0] + 1) * X_10users.sh
pe[1])), dtype=int)[:, 1:]
X_sparse_150users = csr_matrix(((np.ones(X_150users.size),
                            X_150users.ravel(),
                            np.arange(X_150users.shape[0] + 1) * X_150users.
hape[1])), dtype=int)[:, 1:]
```

Сохраним эти разреженные матрицы с помощью pickle (сериализация в Python), также сохраним вектора *y_10users, y_150users* – целевые значения (id пользователя) в

выборках из 10 и 150 пользователей. На этих данных мы будем проверять первые модели классификации. Сохраним также и частотные словари сайтов для 3, 10 и 150 пользователей.

In [64]:

```
with open(os.path.join(PATH_TO_DATA, 'X_sparse_10users.pkl'), 'wb') as X10_
pkl:
    pickle.dump(X_sparse_10users, X10_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'y_10users.pkl'), 'wb') as y10_pkl:
    pickle.dump(y_10users, y10_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'X_sparse_150users.pkl'), 'wb') as X15
0_pkl:
    pickle.dump(X_sparse_150users, X150_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'y_150users.pkl'), 'wb') as y150_pkl:
    pickle.dump(y_150users, y150_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'site_freq_3users.pkl'), 'wb') as site
_freq_3users_pkl:
    pickle.dump(site_freq_3users, site_freq_3users_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'site_freq_10users.pkl'), 'wb') as sit
e_freq_10users_pkl:
    pickle.dump(site_freq_10users, site_freq_10users_pkl, protocol=2)
with open(os.path.join(PATH_TO_DATA, 'site_freq_150users.pkl'), 'wb') as si
te_freq_150users_pkl:
    pickle.dump(site_freq_150users, site_freq_150users_pkl, protocol=2)
```

Чисто для подстраховки проверим, что число столбцов в разреженных матрицах `X_sparse_10users` и `X_sparse_150users` равно ранее посчитанным числам уникальных сайтов для 10 и 150 пользователей соответственно.

In [65]:

```
assert X_sparse_10users.shape[1] == len(site_freq_10users)
```

In [66]:

```
assert X_sparse_150users.shape[1] == len(site_freq_150users)
```

Что еще можно сделать

- можно обработать исходные данные по 3000 пользователей; обучать на такой выборке модели лучше при наличии доступа к хорошим мощностям (можно арендовать инстанс Amazon EC2, как именно, описано тут). Хотя далее в проекте мы будем использовать алгоритмы, способные обучаться на больших выборках при малых вычислительных потребностях;
- Помимо явного создания разреженного формата можно еще составить выборки с помощью `CountVectorizer`, `TfidfVectorizer` и т.п. Поскольку данные по сути могут быть описаны как последовательности, то можно вычислять n-граммы сайтов. Все это мы будем применять в ходе проекта.

Во 2 части мы еще немного поготовим данные и потестируем первые гипотезы, связанные с нашими наблюдениями.