Введение в анализ текстовой информации с помощью Python и методов машинного обучения

* [Data Mining](https://habrahabr.ru/hub/data_mining/),
* [Python](https://habrahabr.ru/hub/python/)

Введение

Сегодня я продолжу рассказ о применении методов анализа данных и машинного обучения на практических примерах. В прошлой [статье](http://habrahabr.ru/post/204500/) мы с вами разбирались с задачей кредитного скоринга. Ниже я попытаюсь продемонстрировать решение другой задачи с того же [турнира](https://www.tcsbank.ru/tournament/), а именно «Задачи о паспортах» (Задание №2).  
При решении будут показаны основы анализа текстовой информации, а также ее кодирование для построения модели с помощью Python и модулей для анализа данных ([pandas](http://pandas.pydata.org/pandas-docs/stable/), [scikit-learn](http://scikit-learn.org/), [pymorphy](https://pymorphy2.readthedocs.org/en/latest/index.html)).

Постановка задачи

При работе с большим объёмом данных важно поддерживать их чистоту. А при заполнении заявки на банковский продукт необходимо указывать полные паспортные данные, в том числе и поле «кем выдан паспорт», число различных вариантов написаний одного и того же отделения потенциальными клиентами может достигать нескольких сотен. Важно понимать, не ошибся ли клиент, заполняя другие поля: «код подразделения», «серию/номер паспорта». Для этого необходимо сверять «код подразделения» и «кем выдан паспорт».  
Задача заключается в том, чтобы проставить коды подразделений для записей из [тестовой выборки](https://static.tcsbank.ru/documents/olymp/passport_test_set.csv), основываясь на [обучающей выборке](https://static.tcsbank.ru/documents/olymp/passport_training_set.csv).

Предварительная обработка данных

Загрузим данные и посмотрим, что мы имеем:

from pandas import read\_csv

import pymorphy2

from sklearn.feature\_extraction.text import HashingVectorizer

from sklearn.cross\_validation import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, roc\_auc\_score

from sklearn.decomposition import PCA

train = read\_csv('https://static.tcsbank.ru/documents/olymp/passport\_training\_set.csv',';', index\_col='id' ,encoding='cp1251')

train.head(5)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **passport\_div\_code** | **passport\_issuer\_name** | **passport\_issue\_month/year** |
| **id** |  |  |  |
| **1** | 422008 | БЕЛОВСКИМ УВД КЕМЕРОВСКОЙ ОБЛАСТИ | 11M2001 |
| **2** | 500112 | ТП №2 В ГОР. ОРЕХОВО-ЗУЕВО ОУФМС РОССИИ ПО МО ... | 03M2009 |
| **3** | 642001 | ВОЛЖСКИМ РОВД ГОР.САРАТОВА | 04M2002 |
| **4** | 162004 | УВД МОСКОВСКОГО РАЙОНА Г.КАЗАНЬ | 12M2002 |
| **5** | 80001 | ОТДЕЛОМ ОФМС РОССИИ ПО РЕСП КАЛМЫКИЯ В Г ЭЛИСТА | 08M2009 |

Теперь можно посмотреть как пользователи записывают поле «кем выдан паспорт» на примере какого-либо подразделения:

example\_code = train.passport\_div\_code[train.passport\_div\_code.duplicated()].values[0]

for i in train.passport\_issuer\_name[train.passport\_div\_code == example\_code].drop\_duplicates():

print i

ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РЕСПУБЛИКЕ КАРЕЛИЯ В МЕДВЕЖ. Р-Е  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО Р. КАРЕЛИЯ В МЕДВЕЖЬЕГОРСКОМ РАЙОНЕ  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РЕСП КАРЕЛИЯ В МЕДВЕЖЬЕГОРСКОМ Р-НЕ  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РЕСПУБЛИКЕ КАРЕЛИЯ В МЕДВЕЖЬЕГОРСКОМ РАЙОНЕ  
ОУФМС РОССИИ ПО РЕСПУБЛИКЕ КАРЕЛИЯ В МЕДВЕЖЬЕГОРСКОМ РАЙОНЕ  
УФМС РОССИИ ПО РК В МЕДВЕЖЬЕГОРСКОМ РАЙОНЕ  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РЕСПУБЛИКЕ КАРЕЛИЯ МЕДВЕЖЬЕГОРСКОМ Р-ОНЕ  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РК В МЕДВЕЖЬЕГОРСКОМ РАЙОНЕ  
ОТДЕЛЕНИЕМ УФМС РОССИИ ПО РЕСПУБЛИКЕ КОРЕЛИЯ В МЕДВЕЖИГОРСКОМ РАЙОНЕ  
УФМС РОССИИ ПО Р. КАРЕЛИЯ МЕДВЕЖЬЕГОРСКОГО Р-НА  
ОТДЕЛОМ УФМС РОССИИ ПО РЕСПУБЛИКЕ КАРЕЛИЯ В МЕДВЕЖЬЕГОРСКОМ  
УФМС РЕСПУБЛИКИ КАРЕЛИИ МЕДВЕЖЬЕГОРСКОГО Р-ОН  
МЕДВЕЖЬЕГОРСКИМ ОВД  
  
Как можно заметить нужно на поле действительно заполняется криво. Но для нормально кодирования мы должны привести это поле к более-менее нормальному (однозначному) виду.  
Для начала я бы предложил привести все записи к одному регистру, например, чтобы все буквы стали строчными. Это легко сделать с помощью атрибута str, столбца DataFrame'a. Этот атрибут позволяет работать со столбцом как с строкой, а также выполнять различного рода поиск и замену по регулярным выражениям:

train.passport\_issuer\_name = train.passport\_issuer\_name.str.lower()

train[train.passport\_div\_code == example\_code].head(5)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **passport\_div\_code** | **passport\_issuer\_name** | **passport\_issue\_month/year** |
| **id** |  |  |  |
| **19** | 100010 | отделением уфмс россии по республике карелия в... | 04M2008 |
| **22** | 100010 | отделением уфмс россии по р. карелия в медвежь... | 10M2009 |
| **5642** | 100010 | отделением уфмс россии по респ карелия в медве... | 08M2008 |
| **6668** | 100010 | отделением уфмс россии по республике карелия в... | 08M2011 |
| **8732** | 100010 | отделением уфмс россии по республике карелия в... | 08M2012 |

C регистром определились. Далее надо по возможности избавиться от популярных сокращений, например район, город и т.д. Сделаем это с помощью регулярных выражений. **Pandas** предоставляет удобное использование регулярных выражений применительно к каждому столбцу. Это выглядит так:

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u'р-(а|й|о|н|е)\*',u'район')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' г( |\.|(ор(\.| )))', u' город ')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' р(\.|есп )', u' республика ')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' адм([а-я]\*)(\.)?', u' административный ')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' окр(\.| |уга( )?)', u' округ ')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' ао ', u' административный округ ')

Теперь избавимся от всех лишних символов, кроме русских букв, дефисов и пробелов. Это связано с тем, что паспорт о одинаковым подразделением может выдаваться отделами с разными номерами, и это ухудшит дальнейшую кодировку:

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' - ?', u'-')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u'[^а-я -]','')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u'- ',' ')

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u' \*',' ')

На следующем шаге, надо расшифровать аббревиатуры, типа УВД, УФНС, ЦАО, ВАО и т.д., т.к. этих их в принципе не много, но на качестве дальнейшего кодирования это скажется положительно. Например если у нас будет две записи «УВД» и «управление внутренних дел», то закодированы они будут по разному, т. к. для компьютера это разные значения.  
Итак перейдем к расшифровке. И, для начала, заведем словарь сокращений, с помощью которого мы и сделаем расшифровку:

sokr = {u'нао': u'ненецкий автономный округ',

u'хмао': u'ханты-мансийский автономный округ',

u'чао': u'чукотский автономный округ',

u'янао': u'ямало-ненецкий автономный округ',

u'вао': u'восточный административный округ',

u'цао': u'центральный административный округ',

u'зао': u'западный административный округ',

u'cао': u'северный административный округ',

u'юао': u'южный административный округ',

u'юзао': u'юго-западный округ',

u'ювао': u'юго-восточный округ',

u'свао': u'северо-восточный округ',

u'сзао': u'северо-западный округ',

u'оуфмс': u'отдел управление федеральной миграционной службы',

u'офмс': u'отдел федеральной миграционной службы',

u'уфмс': u'управление федеральной миграционной службы',

u'увд': u'управление внутренних дел',

u'ровд': u'районный отдел внутренних дел',

u'говд': u'городской отдел внутренних дел',

u'рувд': u'районное управление внутренних дел',

u'овд': u'отдел внутренних дел',

u'оувд': u'отдел управления внутренних дел',

u'мро': u'межрайонный отдел',

u'пс': u'паспортный стол',

u'тп': u'территориальный пункт'}

Теперь, собственно произведем расшифровку абривеатур и отформатируем полученные записи:

for i in sokr.iterkeys():

train.passport\_issuer\_name = train.passport\_issuer\_name.str.replace(u'( %s )|(^%s)|(%s$)' % (i,i,i), u' %s ' % (sokr[i]))

*#удалим лишние пробелы в конце и начале строки*

train.passport\_issuer\_name = train.passport\_issuer\_name.str.lstrip()

train.passport\_issuer\_name = train.passport\_issuer\_name.str.rstrip()

Предварительный этап обработки поля «кем выдан паспорт» на этом закончим. И перейдем к полю, в котором находится дата выдачи.  
Как можно заметить данные в нем хранятся в виде: *месяц****M****год*.  
Соответственно можно просто убрать букву «M» и привести поле к числовому типу. Но если хорошо подумать, то это поле можно удалить, т.к. на один месяц в году может приходиться несколько подразделений выдававших паспорт, и соответственно это может испортить нашу модель. Исходя из этого удалим его из выборки:

train = train.drop(['passport\_issue\_month/year'], axis=1)

Теперь мы можем перейти к анализу данных.

Анализ данных

Итак, данные для построения модели у нас есть, но они находятся в текстовом виде. Для построения модели хорошо бы было их закодировать в числовом виде.  
Авторы пакета [scikit-learn](http://scikit-learn.org/) заботливо о нас позаботились и добавили [несколько способов](http://scikit-learn.org/stable/modules/feature_extraction.html#feature-extraction) для извлечения и кодирования текстовых данных. Из них мне больше всего нравятся два:

1. [FeatureHasher](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html#sklearn.feature_extraction.FeatureHasher)
2. [CountVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer)
3. [HashingVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html#sklearn.feature_extraction.text.HashingVectorizer)

**FeatureHasher** преобразовывает строку в числовой массив заданной длинной с помощью хэш-функции (32-разрядная версия [Murmurhash3](http://ru.wikipedia.org/wiki/Murmur2))  
**CountVectorizer** преобразовывает входной текст в матрицу, значениями которой, являются количества вхождения данного ключа(слова) в текст. В отличие от FeatureHasher имеет больше настраиваемых параметров(например можно задать [токенизатор](http://lingvocourse.ru/wiki/index.php/%D0%A2%D0%BE%D0%BA%D0%B5%D0%BD%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80)), но работает медленнее.  
Для более точного понимания работы CountVectorizer приведем простой пример. Допустим есть таблица с текстовыми значениями:

|  |
| --- |
| **Значение** |
| раз два три |
| три четыре два два |
| раз раз раз четыре |

Для начала CountVectorizer собирает уникальные ключи из всех записей, в нашем примере это будет:  
  
[раз, два, три, четыре]  
  
Длина списка из уникальных ключей и будет длиной нашего закодированного текста (в нашем случае это 4). А номера элементов будут соответствовать, количеству раз встречи данного ключа с данным номером в строке:  
  
раз два три --> [1,1,1,0]  
три четыре два два --> [0,2,1,1]  
  
Соответственно после кодировки, применения данного метода мы получим:

|  |
| --- |
| **Значение** |
| 1,1,1,0 |
| 0,2,1,1 |
| 3,0,0,1 |

**HashingVectorizer** является смесью двух выше описанных методов. В нем можно и регулировать размер закодированной строки (как в *FeatureHasher*) и настраивать токенизатор (как в *CountVectorizer*). К тому же его производительность ближе к FeatureHasher.  
Итак, вернемся к анализу. Если мы посмотрим по внимательнее на наш набор данных то можно заметить, что есть похожие строки но записанные по разному например: "*… республик****а****карелия...*" и "*… по республик****е****карелия...*".  
Соответственно, если мы попробуем применить один из методов кодирования сейчас мы получим очень похожие значения. Такие случаем можно минимизировать если все слова в записи мы приведем к [нормальной форме](http://pymorphy2.readthedocs.org/en/0.1/glossary.html#term-1).  
Для этой задачи хорошо подходит [pymorphy](http://habrahabr.ru/post/176575/) или [nltk](http://nltk.org/). Я буду использовать первый, т.к. он изначально создавался для работы с русским языком. Итак, функция которая будет отвечать за нормализацию и очиску строки выглядит так:

def f\_tokenizer(s):

morph = pymorphy2.MorphAnalyzer()

if type(s) == unicode:

t = s.split(' ')

else:

t = s

f = []

for j in t:

m = morph.parse(j.replace('.',''))

if len(m) <> 0:

wrd = m[0]

if wrd.tag.POS not in ('NUMR','PREP','CONJ','PRCL','INTJ'):

f.append(wrd.normal\_form)

return f

Функция делает следующее:

* Сначала она преобразовывает строку в список
* Затем для всех слов производит разбор
* Если слово является числительным, предикативном, предлогом, союзом, частицей или междометием не включаем его в конечный набор
* Если слово не попало в предыдущий список, берем его нормальную форму и добавляем в финальный набор

Теперь, когда есть функция для нормализации можно приступить к кодированию с помощью метода *CountVectorizer*. Он выбран потому, что ему можно передать нашу функцию, как токенизатор и он составит список ключей по значениям полученным в результате работы нашей функции:

coder = HashingVectorizer(tokenizer=f\_tokenizer, n\_features=256)

Как можно заметить при создании метода кроме токенизатора мы задаем еще один параметр *n\_features*. Через данный параметр задается длина закодированной строки (в нашем случае строка кодируется при помощи 256 столбцов). Кроме того, у *HashingVectorizer* есть еще одно преимущество перед *CountVectorizer*, но сразу может выполнять нормализацию значений, что хорошо для таких алгоритмов, как SVM.  
Теперь применим наш кодировщик к обучающему набору:

TrainNotDuble = train.drop\_duplicates()

trn = coder.fit\_transform(TrainNotDuble.passport\_issuer\_name.tolist()).toarray()

Построение модели

Для начала нам надо задать значения для столбца, в котором будут содержаться метки классов:

target = TrainNotDuble.passport\_div\_code.values

Задача, которую мы решаем сегодня, принадлежит к классу задач классификации со множеством классов. Для решения данной задачи лучше всего подошел алгоритм [RandomForest](http://ru.wikipedia.org/wiki/Random_forest). Остальные алгоритмы показали очень плохие результаты (менее 50%) поэтому я решил не занимать место в статье. При желании любой интересующийся может проверить данные результаты.  
Для оценки качества классификации будем использовать количество документов по которым принято правильное решение, т. е.  
https://habrastorage.org/getpro/habr/post_images/090/795/d9a/090795d9a1358f2ab41e31d0b4d38539.png  
, где *P* — количество документов по которым классификатор принял правильное решение, а *N* – размер обучающей выборки.  
В пакете scikit-learn для этого есть функция: **[accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html" \l "sklearn.metrics.accuracy_score)**  
Перед началом построения собственно модели, давайте сократим размерность с помощью «метода главных компонент», т.к. 256 столбцов для обучения довольно много:

pca = PCA(n\_components = 15)

trn = pca.fit\_transform(trn)

Модель будет выглядеть так:

model = RandomForestClassifier(n\_estimators = 100, criterion='entropy')

TRNtrain, TRNtest, TARtrain, TARtest = train\_test\_split(trn, target, test\_size=0.4)

model.fit(TRNtrain, TARtrain)

print 'accuracy\_score: ', accuracy\_score(TARtest, model.predict(TRNtest))

accuracy\_score: 0.6523456

Заключение

В качестве вывода нужно отметить, что полученная точность в 65% близка к угадыванию. Чтобы улучшить нужно при первичной обработке обработать грамматические ошибки и различного рода описки. Данное действие также скажется положительно и на словаре при кодировании поля, т. е. его размер уменьшиться и соответственно уменьшиться длина строки после ее кодировки.  
Кроме того этап обучения тестовой выборки опущен специально, т. к. в нем нет ничего особенного, кроме его приведения к нужному виду (это можно легко сделать взяв за основу преобразования обучающей выборки)  
В статье я попытался показать минимальный список этапов по обработке текстовой информации для подачи ее алгоритмам машинного обучения. Возможно делающим первые шаги в анализе данных данная информация будет полезной.