Классификация электронных писем с помощью SVM

Вторая часть лабораторной работы посвящена классификации электронных писем: спам/не спам. Для того, чтобы использовать метод опорных векторов (SVM) для классификации электронных писем, необходимо выполнить подготовку данных. Подготовка данных состоит из двух этапов:

1. Предобработка писем
2. Формирование матрицы признаков **X**

Рассмотрим эти процедуры более подробно.

Этап 1: Предобработка писем

На этом этапе выполняются следующие действия:

- Текст письма приводится к нижнему регистру
- Из тела письма удаляются HTML-теги
- Все ссылки заменяются на строку **httpaddr**
- Все адреса e-mail заменяются на строку **emailaddr**
- Все числа заменяются на строку **number**
- Все значки доллара **$** заменяются на строку **dollar**
- У слов отбрасываются окончания (**stemming** - поиск слова во всех его морфологических формах)
- Заменяются на одиночный пробел «не слова»: табы, множественные пробелы, переводы строк.

Например, пусть дано электронное письмо:

```
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors youre expecting. This can be
anywhere from less than 10 bucks a month to a couple of $100. You
should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if
youre running something big..
To unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

После процедуры предобработки будет сформирован такой текст:

```
anyon know how much it cost to host a web portal well it depend on how
mani visitor your expect thi can be anywher from less than number buck
a month to a coupl of dollarnumb you should checkout httpaddr or perhap
amazon ecnumb if your run someth big to unsubscrib yourself from thi
mail list send an email to emailaddr
```

Существует специальный словарь, в котором часто употребляемые слова ассоциированы с некоторым числовым индексом. Пример такого словаря:

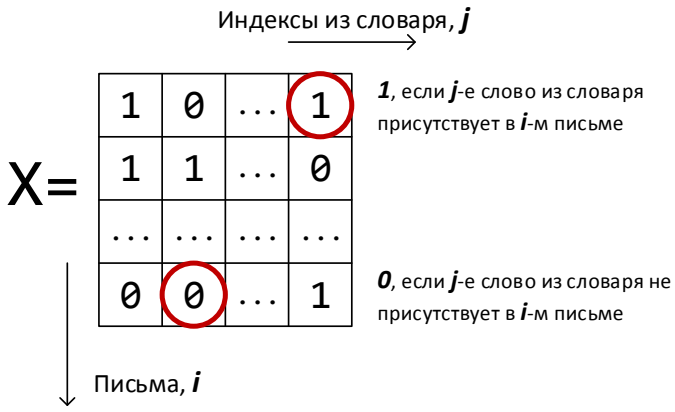
```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```

Словарь для данной лабораторной работы находится в файле **dictionary.txt**. Вот как выглядит наше электронное письмо после замены слов индексами из словаря:

```
86 916 794 1077 883 370 1699 790 1822 1831 883 431 1171 794 1002 1893 1364 592 1676 238
162 89 688 945 1663 1120 1062 1699 375 1162 479 1893 1510 799 1182 1237 810 1895 1440 1547
181 1699 1758 1896 688 1676 992 961 1477 71 530 1699 531
```

Этап 2: Формирование матрицы признаков X

Для работы алгоритма **SVM** необходимо сформировать матрицу признаков **X**. Каждая строка этой матрицы соответствует отдельному электронному письму. Каждый столбец соответствует слову (индексу) из словаря. Если **j**-е слово из словаря присутствует в **i**-м письме, в соответствующей ячейке матрицы будет единица, если отсутствует – то ноль. Рисунок ниже поясняет структуру матрицы признаков **X**.



Задание 1. Подключить необходимые библиотеки

Создайте текстовый файл, в котором будете записывать код вашего скрипта. Файл должен располагаться в том же каталоге, что и прилагаемые к лабораторной работе файлы. В начале файла подключите необходимые библиотеки. Код для подключения приведён ниже.

```
import numpy as np
import scipy.io
from sklearn import svm
from collections import OrderedDict

from process_email import process_email
from process_email import email_features
from process_email import get_dictionary
```

Здесь в 3 строке подключается модуль **svm** из библиотеки **sklearn**. **Sklearn** – мощная Python-библиотека для машинного обучения с большим количеством алгоритмов предобработки данных, снижения размерности, решения задач классификации, кластеризации, регрессии и т.д.

В последних 3 строках подключаются необходимые функции из файла **process_email.py**, которые вам предстоит дополнить в следующих заданиях.

Задание 1. Загрузить и вывести на экран текст электронного письма из файла email.txt

На первом этапе необходимо загрузить электронное письмо из текстового файла **email.txt**. Текст письма должен быть сохранён в строке. Загрузку письма можно выполнить с помощью следующего кода:

```
with open('email.txt', 'r') as file:
  email = file.read().replace('\n', '')
```

В результате текст электронного письма будет сохранён в строковой переменной **email**. Отобразите его на экране с помощью функции **print**. Если вы всё сделали правильно, на экране должен отобразиться следующий текст:

```
> Anyone knows how much it costs to host a web portal ?>Well, it depends on how
many visitors you're expecting.This can be anywhere from less than 10 bucks a mo
nth to a couple of $100. You should checkout http://www.rackspace.com/ or perhap
s Amazon EC2 if youre running something big..To unsubscribe yourself from this m
ailing list, send an email to:groupname-unsubscribe@egroups.com
```

Задание 2. Дополнить код функции предобработки письма process_email

Перейдите в файл **process_email.py**, найдите функцию **process_email**. В начале функции идёт код предобработки текста письма. Здесь выполняется приведение к нижнему регистру, удаление html-тегов и непечатаемых символов. Ссылки, электронные адреса, числа и значки $ заменяются на соответствующие слова. Эти операции реализованы на регулярных выражениях. Данный код менять не нужно.

Найдите ниже в коде функции комментарий, начинающийся со слова **TODO**. Следуя пояснениям в комментарии дополните код так, чтобы заполнить массив **word_indices** индексами слов из словаря **vocabList**.

После того, как в код функции внесены необходимые изменения, перейдите в файл вашего скрипта и вызовите функцию **process_email**. Выведите результат её работы на экран. Если всё выполнено верно, вы должны увидеть на экране следующий текст:

```
[85, 915, 793, 1076, 882, 369, 1698, 789, 1821, 1830, 882, 430, 1170, 793, 1001,
 1894, 237, 161, 88, 687, 944, 1662, 1119, 1061, 1698, 374, 1161, 476, 1119, 189
2, 1509, 798, 1181, 1236, 809, 1894, 1439, 1546, 1757, 1895, 687, 1675, 991, 960
, 1476, 70, 529, 530]
```

Задание 3. Выполнить преобразование текста письма в вектор признаков

Снова перейдите в файл **process_email.py** и найдите функцию **email_features**. Эта функция преобразует массив индексов, полученный на предыдущем этапе, в вектор признаков. Вектор признаков представляет собой массив из **1899** элементов, по числу слов в словаре. Если **i**-е слово из словаря присутствует в электронном письме, то в векторе признаков на **i**-й позиции будет **1**. В противном случае – **0**.

Найдите в **email_features** комментарий **TODO** и дополните код в соответствии с комментариями. Перейдите в код вашего скрипта и вызовите функцию **email_features**, передав в неё список индексов, который вы получили в задании 2. Выведите на экран размер вектора признаков и количество ненулевых элементов в нём. Для определения количества ненулевых элементов можете использовать код

```
sum(features > 0)
```

где **features** – вектор признаков, который вернула функция **email_features**.

Если задание выполнено правильно, должен получиться примерно такой вывод:

```
Длина вектора признаков: 1899
Количество ненулевых элементов: 42
```

Задание 4. Загрузить обучающую выборку и обучить классификатор, оценить его точность

Используйте приведённый ниже код для создания и обучения классификатора:

```
data = scipy.io.loadmat('train.mat')
```

```
X = data['X']
y = data['y'].flatten()

print('Тренировка SVM-классификатора с линейным ядром...')
clf = svm.SVC(C=0.1, kernel='linear', tol=1e-3)
model = clf.fit(X, y)
p = model.predict(X)
```

Оцените точность классификатора. Для этого посчитайте отношение количества совпавших примеров к общему числу примеров. Значения классов из обучающей выборки хранятся в массиве **y**. Значения классов, предсказанных классификатором, хранятся в массиве **p**. Число совпавших примеров можно вычислить как **sum(p == y)**. Подумайте, как можно оценить точность с помощью вызова одной функции **np.mean**. Выведите точность классификации на обучающей выборке в процентах. Если вы всё сделали правильно, должен получиться следующий вывод на экран:

```
Точность на обучающей выборке:  99.825
```

Задание 5. Оцените точность классификатора на тестовой выборке

Загрузите данные из файла **test.mat**, как было показано в предыдущем задании. Используйте ключи **'Xtest'** и **'ytset'** для извлечения матрицы признаков и вектора ответов. Используя модель, обученную на предыдущем этапе, сделайте предсказания для тестовой выборки, а затем оцените точность модели. Если вы всё сделали верно, должен получиться следующий ответ:

```
Точность на тестовой выборке:  98.9
```

Как видим, классификатор почти безошибочно делит письма на спам и не спам для обучающей и тестовой выборки.

Задание 6. Определить, какие слова чаще встречаются в письмах со спамом

Добавьте приведённый ниже код в свой скрипт, запустите и проанализируйте результат.

```
t = sorted(list(enumerate(model.coef_[0])),key=lambda e: e[1], reverse=True)
d = OrderedDict(t)
idx = list(d.keys())
weight = list(d.values())
dictionary = get_dictionary()

print('Топ-15 слов в письмах со спамом: ')
for i in range(15):
  print(' %-15s (%f)' %(dictionary[idx[i]], weight[i]))
```

Задание 7 [опциональное]. Проверить письма из своего почтового ящика

Выберите из своего почтового ящика хорошие письма и письма, содержащие спам, сохраните их в виде текстовых файлов. Выполните предобработку писем с помощью функций **process_email** и **email_features**, а затем проверьте их с помощью модели, созданной в задании 4. Письмо является спамом, если классификатор выдал для него прогноз 1.