

Практическая работа № 4

Настройка параметров SVM и его использование для решения задач бинарной классификации

В ходе выполнения примеров программного кода вам потребуется библиотека `mglearn`. Установите ее с помощью команды `pip install mglearn`.

В этой работе Вы узнаете, как использовать функции `sklearn`, реализующие линейный и нелинейный SVM и посмотрите, как работа этого классификатора зависит от его параметров и насколько важна предобработка данных.

Далее, во второй части работы, Вы построите классификатор спама на основе SVM.

1. Параметры SVM и предварительная нормализация данных.

В этой части работы Вам предстоит решить четыре простых задачи.

1) Линейный SVM для линейно разделимых классов.

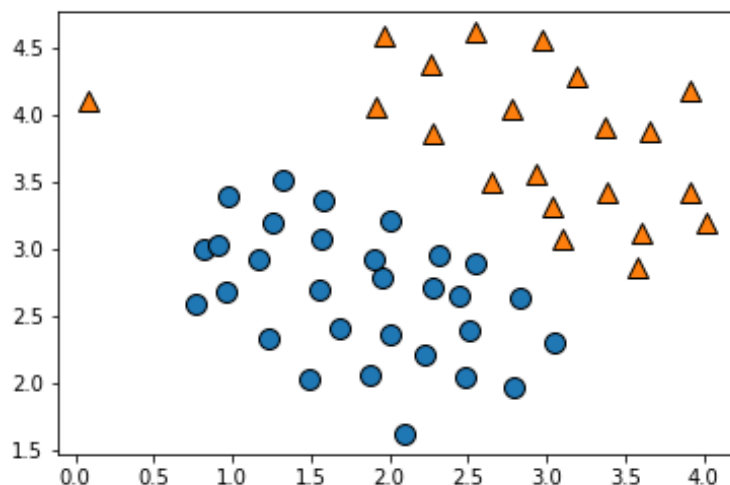
Исходные данные - в файле `ex4data1.mat`

Для загрузки файлов с расширением `mat` воспользуемся функцией `loadmat` из пакета `Scipy`.

```
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn import svm
import mglearn
```

```
# 1. Линейный SVM
data = loadmat('ex4data1.mat')
X = data["X"]
y = data["y"].ravel()
```

Имеем следующие данные.



Одинокий треугольник слева представляет собой, по-видимому, выброс и дополнительную сложность для задачи классификации.

Посмотрим, как с этим справится линейный SVM.

```
svclassifier = svm.LinearSVC(C=1, loss='hinge', max_iter=10000)
```

Обучим классификатор с такими параметрами.

```
svclassifier.fit(X,y)
```

Можно посмотреть на точность работы данного классификатора.

```
svclassifier.score(X,y)
```

Построим границу решений и посмотрим, как проходит эта прямая по отношению к данным.

```
# Строим линию, разделяющую два класса
```

```
mglearn.plots.plot_2d_separator(svclassifier, X, eps=.5)
```

```
# Строим исходный набор данных
```

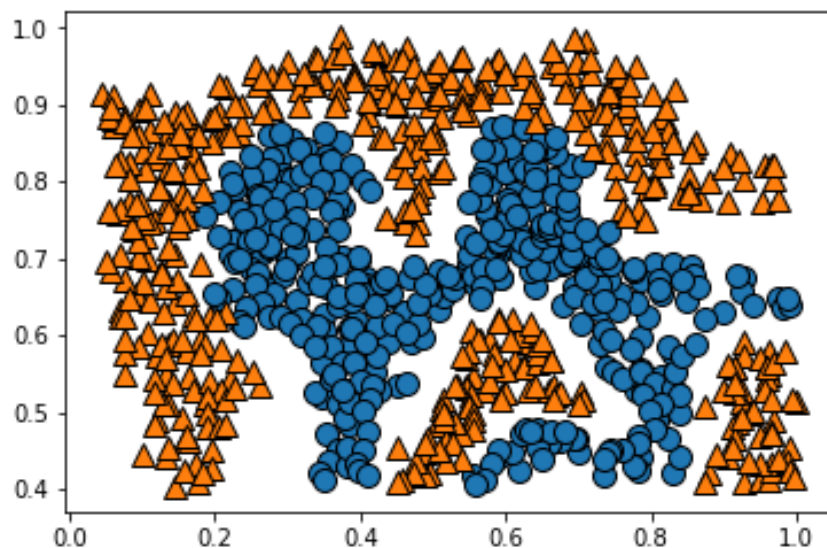
```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
```

Посмотрите на качество решения Вашей задачи. Поменяйте значение параметра C с 1 на, например, 2000. Посмотрите, что поменялось и почему? Объясните результат.

2) Нелинейный SVM для линейно неразделимых классов

Исходные данные - в файле ex4data2.mat

В этом случае набор данных выглядит следующим образом и требует нелинейного SVM.



Воспользуемся соответствующими функциями sklearn.

```
svmclassifier = svm.SVC(kernel='rbf', C=100, gamma=10)
```

```
svmclassifier.fit(X,y)
```

Здесь мы берем $C=100$ и RBF ядро с параметром $\gamma = 10$: $e^{-\gamma\|x-l\|^2}$.

Посмотрим на результат классификации.

Строим кривую, разделяющую два класса

```
mglearn.plots.plot_2d_separator(svmclassifier, X, eps=.05)
```

Строим исходный набор данных

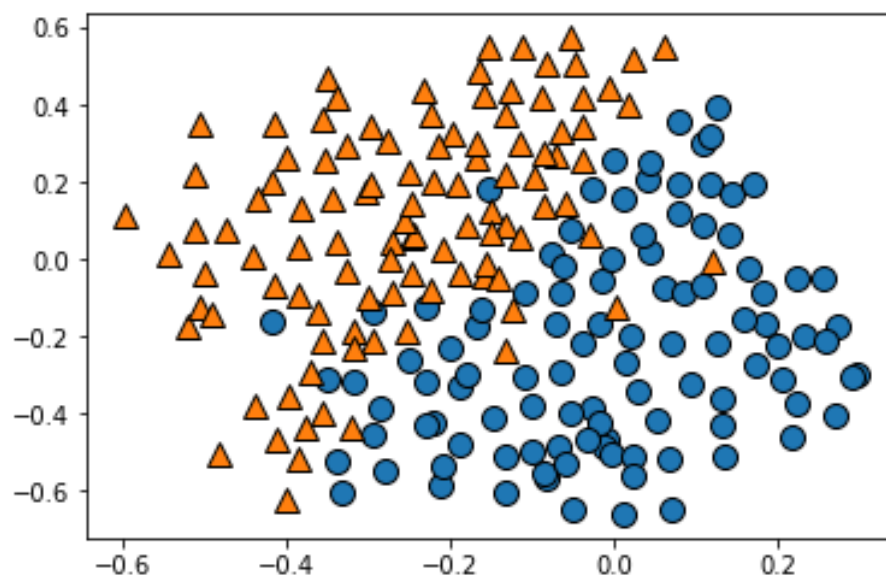
```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
```

Оцените точность работы классификатора. Попробуйте ее увеличить. За счет чего это можно сделать?

3) Подбор параметров SVM для решения задачи нелинейной разделимости

Исходные данные - в файле ex4data3.mat

Эта задача похожа на предыдущую, у нас снова линейно неразделимые два класса,



но помимо обучающего набора данных, у нас еще есть и набор данных для проверки.

Загрузим данные:

```
data = loadmat('ex4data3.mat')
```

```
X = data["X"]
```

```
y = data["y"].ravel()
```

```
Xval = data["Xval"]
```

```
yval = data["yval"].ravel()
```

Для разделения классов будем использовать нелинейный SVM я ядром RBF.

Подберите оптимальные параметры C и γ нелинейного классификатора так, чтобы точность его работы на *проверочном* наборе данных была как можно больше.

Мы собираемся брать разные значения C и γ , обучать каждый раз классификатор на тренировочных данных и следить за точностью его работы на проверочных данных, которые **не** участвуют в процессе обучения.

Наиболее часто используемый метод – это решетчатый поиск (grid search), который по сути является попыткой перебрать все возможные комбинации интересующих параметров. Давайте возьмем, например, значения

```
C_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100]
```

```
gamma_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100]
```

Вы можете взять и свои похожие значения или расширить эти наборы.

Реализуйте простой решетчатый поиск с помощью вложенных циклов `for` по двум параметрам, обучая и оценивая классификатор для каждой их комбинации.

Для наилучшей пары значений постройте границу решений.

Постройте ее также и для некоторых других значений параметров с тем, чтобы оценить влияние C и γ на качество классификации.

С этой точки зрения полезно построить несколько рисунков с постоянным значением одного из параметров и переменным значением другого. Сделайте вывод, как каждый из параметров влияет на качество классификации.

4) Влияние предварительной нормализации данных на работу SVM

Загрузим готовый набор данных Breast Cancer, содержащий 569 данных о злокачественных и доброкачественных опухолях. Каждая точка набора характеризуется 30 признаками.

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

Мы собираемся решить задачу бинарной классификации и посмотреть, как сильно точность решения зависит от предварительной нормализации данных. Предварительно давайте часть данных оставим для проверки. Используем функцию `train_test_split` для разделения набора данных на обучающий и проверочный. По умолчанию данные будут разделены в соотношении 80%+20%.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target)
```

Обучите SVM классификатор на тренировочном наборе данных. Можно вызвать стандартный вариант SVM, пока никак не подбирая параметры метода

```
svc = svm.SVC()
```

```
svc.fit(X_train, y_train)
```

Оцените точность работы классификатора на тренировочном и проверочном наборе данных. Будут ли они близки?

```
print("Правильность на обучающем наборе: {:.2f}".format(svc.score(X_train, y_train)))
```

```
print("Правильность на тестовом наборе: {:.2f}".format(svc.score(X_test, y_test)))
```

Как видите, точность на проверочном наборе серьезно ниже. Это во многом обусловлено сильно различающимися по величине признаками объектов.

Давайте построим минимальное и максимальное значения каждого признака с тем, чтобы оценить, насколько одинаков диапазон их изменений.

```
plt.plot(X_train.min(axis=0), 'o', label="min")
```

```
plt.plot(X_train.max(axis=0), '^', label="max")
```

```
plt.legend(loc=4)
```

```
plt.xlabel("Индекс признака")
```

```
plt.ylabel("Величина признака")
```

```
plt.yscale("log")
```

Как видно, признаки отличаются очень сильно.

Приведем их к одному масштабу, например, так, чтобы значения всех признаков принадлежали диапазону [0,1].

Это можно сделать с помощью функции `MinMaxScaler`, но лучше реализуйте это сами.

Обратите внимание, что проверочных данных у нас как бы нет, максимальное и минимальное значения каждого признака мы вычисляем, основываясь только на тренировочном наборе данных. Далее ТО ЖЕ САМОЕ преобразование мы проделываем с точками проверочного набора - т.е. вычитаем максимальное значение признака и делим на разность между максимальным и минимальным значением, которые были получены для тренировочного набора!

Вновь обучите SVM и оцените точность его работы на обоих наборах данных. Сделайте вывод о том, как на нее повлияла предварительная нормализация данных.

Попробуйте теперь дополнительной настройкой параметров SVM еще больше улучшить точность, полученную для проверочного набора данных.

2. Решите задачу классификации спам/не спам

Многие почтовые службы сегодня предоставляют спам-фильтры, которые позволяют с высокой точностью классифицировать электронные письма как спам и не спам. В этой части работы Вы будете использовать SVM для создания спам-фильтра.

Ваш классификатор SVM будет определять, является ли данное письмо x спамом $y = 1$ или не спамом $y = 0$. Ясно, что каждое письмо должно быть преобразовано в вектор признаков $x \in R^n$. Давайте разберем, как это делается.

Посмотрим на одно из писем.

```
f = open('emailSample1.txt', 'r').read()
```

```
f
```

```
> Anyone knows how much it costs to host a web portal ?\n>\nWell, it depends on how many visitors you're expecting.\nThis can be anywhere from less than 10 bucks a month to a couple of $100. \nYou should checkout http://www.rackspace.com/ or perhaps Amazon EC2 \nif youre running something big..\n\nTo unsubscribe yourself from this mailing list, send an email to:\ngroupname-unsubscribe@egroups.com
```

Образец электронного письма содержит URL, адрес электронной почты (в конце), цифры и суммы в долларах. Многие электронные письма будут содержать объекты аналогичного типа (цифры, другие URL-адреса или другие адреса электронной почты), которые будут отличаться почти в каждом электронном письме. Поэтому один из методов, часто используемых при обработке электронных писем, заключается в «нормализации» этих значений, чтобы все URL обрабатывались одинаково, все числа обрабатывались одинаково и т. д. Например, мы могли бы заменить каждый URL в письме уникальной строкой «httpaddr», чтобы указать, что URL присутствовал.

Это позволяет классификатору спама принимать решение о классификации на основе наличия какого-либо URL-адреса, а не наличия определенного URL-адреса. Как правило, это повышает производительность классификатора спама, поскольку спамеры часто меняют URL-адреса, и, таким образом, вероятность повторного просмотра любого конкретного URL-адреса в новом спам-письме очень мала.

В функции **processEmail** были реализованы следующие этапы предварительной обработки и нормализации электронной почты:

- **Строчный регистр (Lower-casing):** вся электронная почта преобразуется в нижний регистр, использование заглавных букв игнорируется (например, IndIcaTE обрабатывается так же, как и Indicate).
- **Удаление HTML (Stripping HTML):** все HTML удаляются из электронных писем.
- **Нормализация URL (Normalizing URLs):** все URL заменяются текстом «httpaddr».
- **Нормализация адресов электронной почты (Normalizing Email Addresses):** все адреса электронной почты заменяются текстом «emailaddr»..
- **Нормализация чисел (Normalizing Numbers):** все числа заменяются текстом «число».

- **Нормализация доллара (Normalizing Dollars):** все знаки доллара (\$) заменяются текстом «доллар».
- **Стемминг слов (Word Stemming):** слова сводятся к их основной форме. Например, “discount”, “discounts”, “discounted” и “discounting” заменяются на “discount”. Иногда стемминг фактически удаляет символы с конца, так что “include”, “includes”, “included” и “including” все заменяются на “includ”.
- **Удаление не-слов (Removal of non-words):** не-слова и знаки препинания удаляются. Все пробелы (табуляции, новые строки, пробелы) обрезаются до одного пробела.

Результат обработки письма выглядит так. С ним легче дальше работать.

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

Словарь слов (Мешок слов, Vocabulary List, Bag of Words)

После предварительной обработки писем, у нас есть список слов для каждого письма. Следующий шаг - выбрать, какие слова будут использоваться в классификаторе, а какие не будут включены.

Для простоты были выбраны только наиболее часто встречающиеся слова в качестве набора рассматриваемых слов (словарь слов). Использование слов, которые редко встречаются в обучающем наборе (только в нескольких электронных письмах) может привести к переобучению нашей модели.

Список слов находится в файле vocab.txt. Этот словарь был создан путем выбора всех слов, которые встречаются не менее 100 раз среди всех спам-писем. Объем нашего словаря 1899 слов. Однако на практике часто используются словари, содержащие 10000 и даже 50000 слов.

Каждому слову письма (предварительно обработанного) мы можем теперь поставить в соответствие индекс этого слова из нашего словаря. Например, слову «anyone» преобразованному до «anyon» соответствует число 86 в списке словаря.

Часть кода в **processEmail** выполняет это отображение. В коде заданная строка, представляющая собой одно слово из обработанного электронного письма, ищется в списке словаря vocabList. Если слово есть в словаре, индекс слова добавляется в переменную word_indices. Если же слово отсутствует в словаре, то его можно пропустить.

Наше письмо теперь выглядит так:

[86, 916, 794, 1077, 883, 370, 1699, 790, 1822, 1831, 883, 431, 1171, 794, 1002, 1895, 592, 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, 992, 961, 1477, 71, 530, 1699, 531]

Вектор признаков для каждого письма

Теперь мы можем поставить каждому объекту (письму) x его вектор признаков $x \in R^n$. Здесь $n = 1899$ - объем нашего словаря. Координата $x_i = 1$, если i -ое слово (словаря) есть в письме (возможно, оно встретится в письме несколько раз) и $x_i = 0$ в противном случае.

Ясно, что такой 1899-мерный вектор будет в основном заполнен нулями.

Функция **emailFeatures** генерирует вектор признаков для каждого письма. Для нашего примера это будет вектор с 43 ненулевыми элементами.

Обучение SVM классификатора

Теперь давайте загрузим предварительно обработанный набор тренировочных данных, который будет использоваться для обучения классификатора SVM.

spamTrain.mat содержит 4000 примеров спама и не спама, а spamTest.mat содержит 1000 тестовых примеров. Каждое исходное письмо было обработано как описано выше и преобразовано в вектор признаков.

Загрузите данные и обучите SVM классификатор на тренировочном наборе данных. Получите точность около 99,8% на тренировочном и 98,9% на тестовом наборах данных.

Теперь когда у Вас есть обученный классификатор, Вы можете его использовать для классификации Ваших писем. Проверьте работу Вашего классификатора на письмах emailSample1.txt, emailSample2.txt, spamSample1.txt, spamSample2.txt.