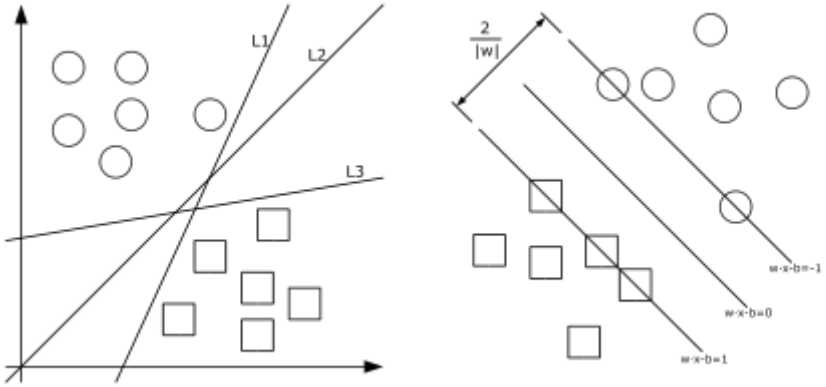
Классификация методом опорных векторов (SVM)

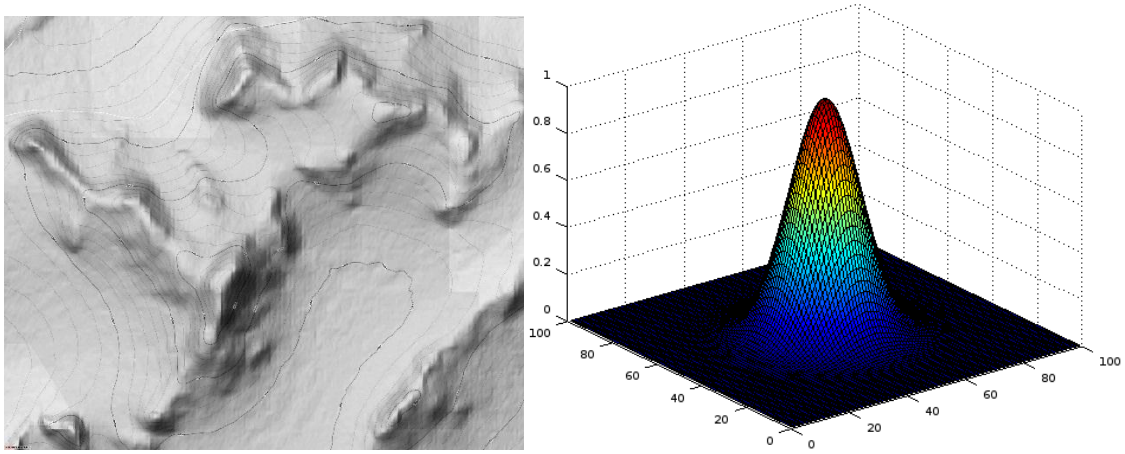
Теоретическая часть

Методопорныхвекторов (SVM, supportvectormachine) – алгоритм классификации данных. Также этот метод называют классификатором с максимальным зазором за его особенность – он стремится провести разделяющую гиперплоскость между классами так, чтобы она максимально далеко отстояла от всех классов. Рисунок ниже поясняет суть оптимальной гиперплоскости:



Среди приведенных прямых только L2 является оптимальной, так как максимально далеко отстоит от обоих классов. Точки, через которые проходят поля разделяющей гиперплоскости (см. второй рисунок), называются опорными векторами. Не вдаваясь в математические подробности алгоритма, следует отметить, что в алгоритме используется параметр **C**. Роль его заключается в следующем. Чем больше значение параметра **C**, тем более точная модель получается на выходе. Но она может быть слишком привязана к обучающей выборке и на реальных данных может не работать (явление переобучения). Если значение параметра **C**не большое, классификатор может допускать много ошибок, то есть может оказаться не точным. На практике проверяют различные значения **C**, по кросс-валидационной (контрольной) выборке оценивают качество обученной модели и выбирают то значение параметра, при котором модель оказалась наиболее точной.

Еще одним важным понятием метода опорных векторов являются функции-ядра. Они используются в том случае, когда не возможно разделить данные с помощью гиперплоскости, и разделяющая граница имеет сложную форму.



Суть ядер поясняют приведенные выше картинки. Данные переводятся в пространство более высокой размерности (на картинке – из плоскости в трехмерное пространство-рельеф). И уже в пространстве более высокой размерности появляется возможность провести разделяющую гиперплоскость. Существует такое понятие, как landmarks (ориентиры). В метафоре с рельефом это горы. Те точки, которые находятся близко к этим ориентирам (горам) будут классифицированы как 1, далеко расположенные точки (на равнине) – как 0. Часто на практике используется гауссово ядро (рисунок выше справа). Две близко расположенные точки обладают высоким

подобием (значение функции ядра для них будет близко к единице), а для далеко расположенных точек подобие будет стремиться к нулю. С помощью набора ориентиров (landmarks) можно описать сколь угодно сложную разделяющую кривую.

В данной лабораторной работе необходимо исследовать параметры **C**и **sigma** (σ), а также линейное и гауссово ядро.Отправной точкой является файл **run.py**. Необходимо создать его и добавить весь необходимый код согласно методическим указаниям, приведенным ниже.

Часть 1. Исследование линейного ядра

В первой части работы необходимо построить границы, разделяющие точки двух классов, с разными значениями параметра **C**. Граница в случае использования линейного ядра представляет собой прямую линию. Если речь идёт об n-мерном пространстве признаков, то граница будет представлять собой не прямую, а гиперплоскость.

Задание 1. Загрузить данные из файла dataset1.mat и отобразить на графике.

В начале вашего скрипта необходимо подключить библиотеки. В данной лабораторной работе будут использоваться библиотеки **scipy.io**, **numpy**и код из файла **svm.py**. Код подключения библиотек следующий:

```
import scipy.io as sio
import numpy as np
import svm
```

Для загрузки данных из файла формата MatLabиспользуйте функцию **loadmat** из модуля **scipy.io**, который подключен под псевдонимом **sio**. Функция возвращает словарь, нас интересуют значения по ключам **'y'**и **'X'**. В первом случае значения представляют собой вектор-столбец размера **51×1**, а во втором – матрицу размера **51×2**. Сохраните значения из словаря в переменные **y**и **X**соответственно. Обратите внимание на то, что значения в векторе-столбце **y**должны быть вещественного типа, поэтому преобразуйте тип, вызвав для **y**метод **astype(np.float64)**.

Напишите код отображения загруженных данных на графике. Используйте для этого функцию **visualize_boundary_linear** из файла **svm.py**. Первыми параметрами передайте переменные **X**и **y**, которые вы только что получили из файла. Третий параметр пока оставьте равным **None**. Передайте так же четвёртый параметр **title**, в котором задайте название графика.

Вот примерный вид графика, который вы должны получить:

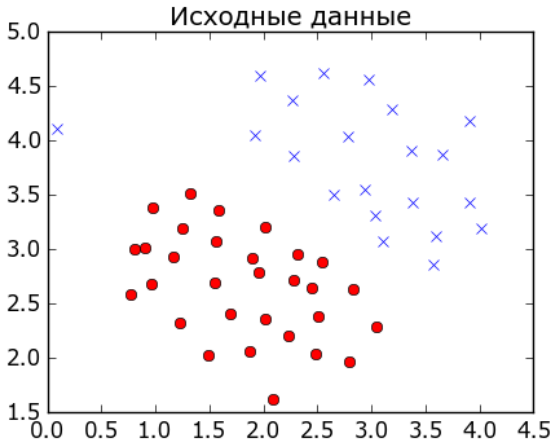


Рисунок 1Исходные данные

Перейдите в файл **svm.py**и разберитесь с тем, как работает функция **visualize_boundary_linear**.

Задание 2. Обучить классификатор на обучающей выборкеи отобразить границу

Для обучения используется функция **svmTrain** из файла **svm.py**. Пример вызова функции:

```
model = svm.svm_train(X, y, C, svm.linear_kernel,0.001,20)
```

Параметры **X**и **y** – обучающая выборка и метки класса, загруженные из файла **dataset1.mat**. **C**–параметр алгоритма. Чем он больше, тем более точную модель выдает классификатор, но она может оказаться сильно подогнанной под обучающую выборку. **svm.linear_kernel** – функция ядра. В данном случае используется линейное ядро.При обучении модели примите параметр **C**равным **1**.

Чтобы вывести график полученной модели, используйте следующий код:

```
svm.visualize_boundary_linear(X, y, model,'<Дайтеназваниеграфику>')
```

Примерный вид графика:

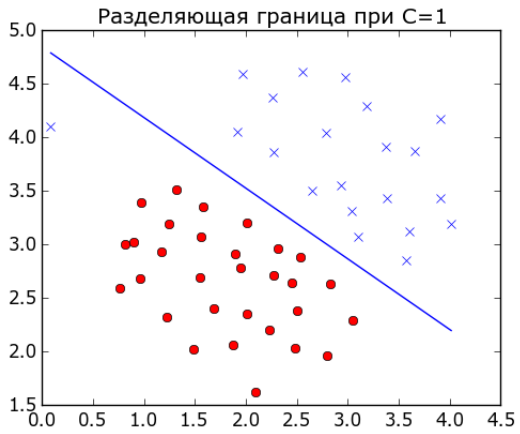


Рисунок 2Разделяющая граница при C=1

Из графика видно, что классификатор допустил одну ошибку.

Задание 3: Выполнить обучение модели с C = 100, привести график

Задание полностью аналогично предыдущим двум пунктам, только в качестве значения параметра **C**необходимо задать 100. Должен получиться такой график:

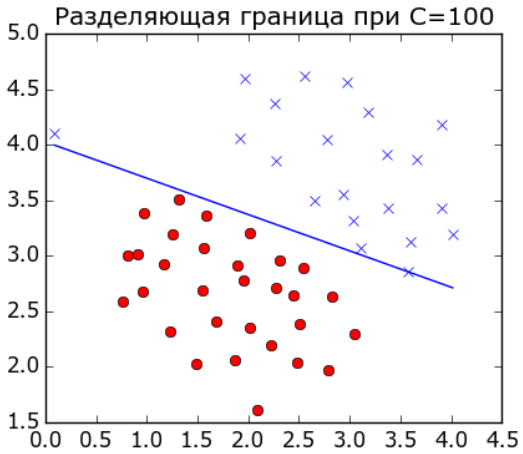


Рисунок 3 Разделяющая граница при C=100

Видно, что теперь классификатор не допустил ни одной ошибки на обучающей выборке, но разделяющая граница проходит не оптимально. Эта проблема называется переобучение (**overfitting**). Параметр **C**как раз используется для решения проблемы переобучения.

Часть 2. Исследование гауссова ядра

Задание 4. Реализовать функцию гауссова ядра, построить контурные графики для sigma = 1 и sigma = 3.

Для этого необходимо в файле **svm.py**добавить функцию со следующей сигнатурой:

```
defgaussian_kernel(x1, x2, sigma=1.0):
return# вычисление значения по формуле (см. ниже)
```

Вместо комментария необходимо вставить код вычисления гауссова ядра по формуле, приведённой ниже:

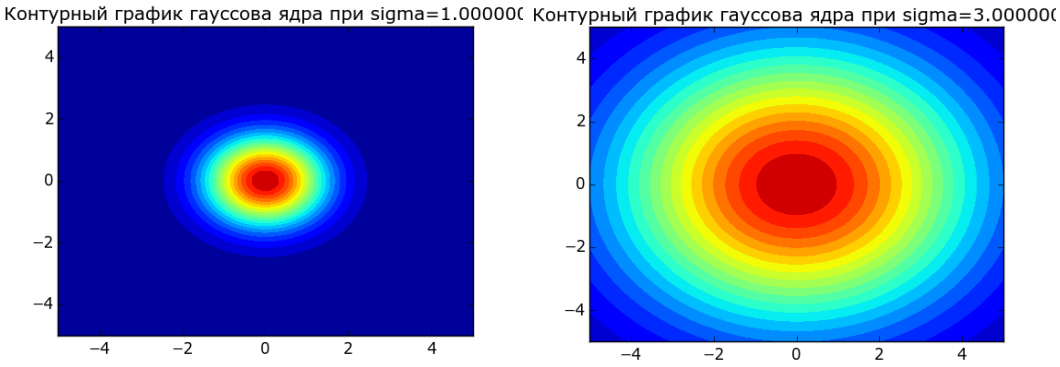
$$K_{gaussian}\left(x^{(i)}, y^{(j)}\right) = exp\left(-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{2\sigma^2}\right) = exp\left(-\frac{\sum_{k=1}^{n}\left(x_k^{(i)} - x_k^{(j)}\right)^2}{2\sigma^2}\right)$$

Для возведения в квадрат используйте функцию **np.power**, для вычисления суммы – **np.sum**, для **exp**–**np.exp**.

Нарисуйте контурные графики гауссова ядра для **sigma=1**и **sigma=3**. Используйте для этого код:

```
svm.contour(1)
svm.contour(3)
```

Если функция вычисления гауссова ядра была реализована верно, должны получиться такие графики:



Задание 5. Загрузить и отобразить данные из файла dataset2.mat

Код загрузки и отображения полностью аналогичен первому заданию. Примерный вид графика:

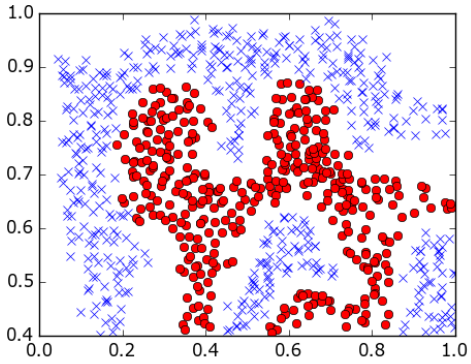


Рисунок 4 Данные из файла dataset2.mat

По графику видно, что разделить данные с помощью прямой линии (линейного ядра) не получится. Поэтому при обучении модели в данном случае будет использоваться гауссово ядро.

Задание 6. Запустить обучение модели, отобразить получившуюся границу

Для обучения модели и отображения границы используйте приведенный ниже код. В данном случае в качестве функции ядра используется гауссово ядро.

```
C =1.0
sigma =0.1
gaussian = svm.partial(svm.gaussian_kernel, sigma=sigma)
gaussian.__name__ = svm.gaussian_kernel.__name__
model = svm.svm_train(X, y, C, gaussian)
svm.visualize_boundary(X, y, model)
```

В данном случае используется функция **partial**. Она возвращает переданную в неё функцию с подставленными параметрами.Суть использования **partial**в том, что список параметров **svm_train**ограничен, и мы не можем передать туда параметры, специфичные для конкретного ядра. Например, для гауссова ядра это параметр **sigma**, а в линейном ядре такого параметра нет. Поэтому создаётся функциональный объект с заранее подставленными специфичными параметрами.

Чтобы понять, как работает **partial**, рассмотрим пример. Объявим функцию **sum_**, которая будет складывать 2 числа:

```
>>> from functools import partial
>>> def sum_(a, b):
      return a + b
```

```
>>> sum_(3, 5)
8
```

Теперь объявим функциональный объект **sum_only_b**, в котором потребуется задавать только параметр **b**, а в параметр **a**будет заранее подставлено значение **10**:

```
>>> sum_only_b = partial(sum_, a=10)
>>>sum_only_b(b=7)
17
```

Если вы выполнили задание правильно, у вас должен получиться такой график:

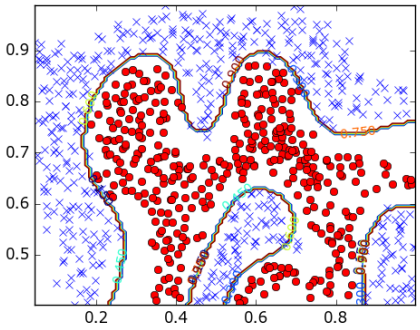


Рисунок 5 Результирующая граница на основе гауссова ядра

Как видим, использование функций-ядер позволяет строить сложные разделяющие границы в тех случаях, когда объекты не могут быть разделены с помощью прямой.

Часть 3. Подбор оптимальных параметров Си sigma

Задание 7. Загрузить данные из файла dataset3.mat и отобразить на экране

В файле присутствуют данные по ключам **X**, **y**, **Xval**, **yval**. **X** – точки обучающей выборки, **y**– классы точек обучающей выборки. **Xval** – точки тестовой выборки, **yval** – классы точек тестовой выборки. Код загрузки данных из файла и отображения их на графике аналогичен предыдущей части. Отобразите два графика: для обучающей и тестовой выборки. Если вы всё сделали правильно, должны получиться такие графики:

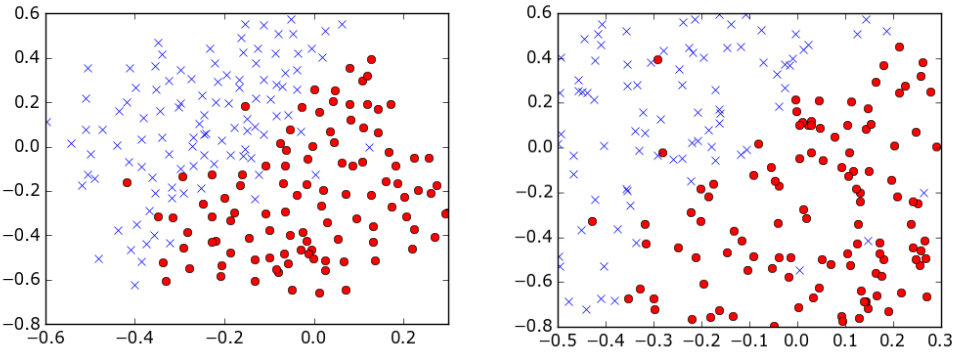


Рисунок 6 Обучающая и тестовая выборки

Задание 8. Выполнить обучение модели с неоптимальными параметрами, вывести график

Возьмите параметры **C = 1**и **sigma = 0.5**, выполните обучение модели на обучающей выборке, выбрав в качестве ядра гауссово ядро. Отобразите получившуюся модель на графике. Должен получиться такой график:

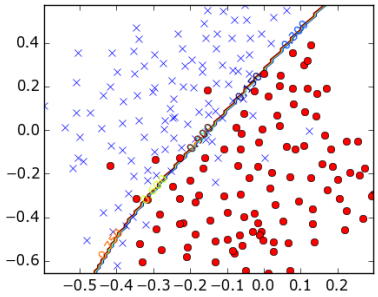


Рисунок 7 Модель при неоптимальных параметрах

Приведённые параметры являются неоптимальными. Это видно из графика. Разделяющая граница проходит почти прямо. Полученная модель является недообученной (**underfitting**). Необходимо подобрать оптимальные параметры.

Задание 9. Вычисление оптимальных значенийCи sigma

Для поиска оптимальных значений параметров **C**и **sigma**нужно рассмотреть несколько сочетаний этих параметров и выяснить, при каком сочетании ошибка классификации будет наименьшей. В вашем скрипте должно быть 2 цикла следующего вида:

```
forCin[0.01,0.03,0.1,0.3,1,3,10,30]:
for sigma in[0.01,0.03,0.1,0.3,1,3,10,30]:
# получение модели с гауссовым ядром, обученной с параметрами C и sigma
# получение результата предсказания для тестовой выборки
# вычисление ошибки предсказания
# если удалось уменьшить ошибку
# запоминаем ошибку, C, sigma и модель
```

Код создания модели с текущими параметрами **C**и **sigma** аналогичен коду, который вы писали выше. Обучать модель нужно на обучающей выборке (используются переменные **X**и **y**).

Далее необходимо получить результаты предсказания для тестовой выборки. Для этого используется функция **svm_predict** из файла **svm.py**. В функцию передаётся обученная ранее модель и тестовая выборка. Ниже приведён пример вызова функции **svm_predict**:

```
ypred = svm.svm_predict(model, Xval)
```

Чтобы вычислить ошибку предсказания, нужно сравнить список классов, которые выдала обученная модель на тестовой выборке, с фактическими классами из тестовой выборки. Эти классы хранятся в переменной **yval**. Есть разные подходы к вычислению уровня ошибки. Здесь мы будем использовать простейший подход – процент неверно предсказанных значений, или отношение количества неверно классифицированных примеров к общему количеству примеров. Например, если классификатор для тестовой выборки выдал массив значений **[1, 0, 0, 1, 0]**, а фактические классы **[1, 1, 0, 1, 0]**, то ошибка будет 20% или 1/5, так как из 5 значений неверно был классифицирован один пример. Ниже приведён пример вычисления ошибки предсказания. Добавьте его внутрь цикла.

```
error = np.mean(ypred != yval.ravel())
```

Функция **ravel**в данном случае нужна для того, чтобы преобразовать матрицу **yval**, состоящую из единственного столбца, к обычному массиву.

Чтобы найти оптимальные значения параметров, нужно выбрать те **C**и **sigma**, при которых ошибка предсказания минимальна. Для этого нужно на каждой итерации внутреннего цикла сравнивать текущее значение ошибки с

наименьшим. Если текущая ошибка оказалась меньше, нужно сохранить в некоторой переменной текущую модель, параметры **C**и **sigma**, а также обновить минимальное значение ошибки.

После того, как оптимальные параметры будут найдены, необходимо вывести графики наилучшей модели для обучающей и тестовой выборки. Если всё сделано правильно, должны получиться графики как на рисунке ниже.

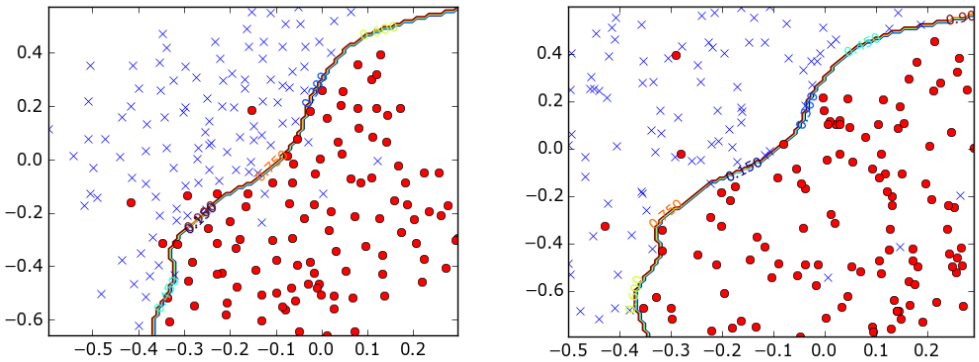


Рисунок 8 Модель с оптимальными параметрами

Видно, что разделяющая граница сложнее, чем в случае с неоптимальными параметрами. Данная модель является оптимально обученной, и хорошо работает как на обучающей, так и на тестовой выборке.

Выведите на экран полученные оптимальные значения **C**и **sigma**.