# 1. Постановка задачи бинарной классификации

Имеется размеченная тренировочная последовательность , где – -мерный вектор, а – метка этого вектора.

Ставится следующая задача, где

* – перпендикуляр к разделяющей гиперплоскости (вектор весовых коэффициентов)
* – некоторое число (смещение)
* – параметр регуляризации, контролирующий ширину полосы
* – ослабляющие переменные

Решая эту задачу, находят оптимальную разделяющую гиперплоскость с максимальной шириной полосы .

# 2. Идея SOR алгоритма построения SVM

Рассмотрим линейный SV-классификатор. Добавим к целевой функции исходной задачи оптимизации переменную :

## 2.1. Получение двойственной задачи

Сведём задачу (2) к эквивалентной ей двойственной задаче. Для этого составим функцию Лагранжа. Введём множители Лагранжа , и , .

Теперь найдём частные производные (3) по :

Подставляем найденные значения и в (3) и получаем двойственную задачу:

В этот момент можно перейти к нелинейному классификатору, так как процедура будет аналогичной, лишь с той разницей все скалярные произведения все скалярные произведения линейного классификатора вычисляются с использованием ядерных функций .

* В случае линейного SV-классификатора:
* В случае нелинейного SV-классификатора будем использовать RBF-ядро (Гауссовское ядро):

Тогда задача принимает вид

Введём обозначения:

Тогда двойственная задача в введённых обозначениях выглядит следующим образом:

Пусть получено решение . Тогда . Решение выглядит следующим образом:

Разделяющая гиперплоскость .

## 2.2. Переход к СЛАУ

Идея алгоритма SOR состоит в том, что естественная модификация целевой функции задачи оптимизации ведёт к исчезновению ограничения типа равенств, что позволяет вместо минимизации квадратичной функции решать эквивалентную её СЛАУ.

Обозначим . Решение задачи (7) эквивалентно решению СЛАУ

# 3. Основные этапы алгоритма SOR

Примерные этапы алгоритма:

* Выбирают .
* Выбирают любой начальный вектор , длиной (обычно берут нулевой вектор).
* вычисляют с помощью следующим образом, пока (обычно ):

Имеем формулу, где -тая компонента вычисляется следующим образом:

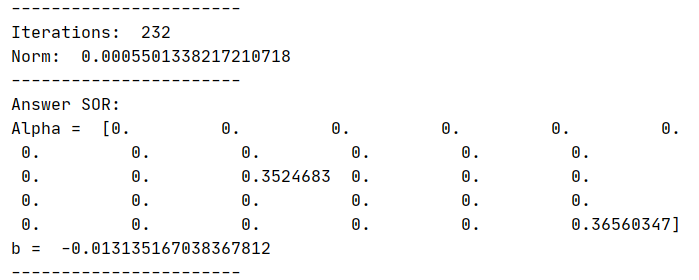
Причём необходимо учитывать на каждом шаге ограничения типа неравенств задачи (7) следующим образом:

# 4. Результаты работы реализации алгоритма на модельных данных

## 4.1. Линейный SV-классификатор

Возьмём линейно разделимые данные и посмотрим, как работает линейный SV-классификатор.

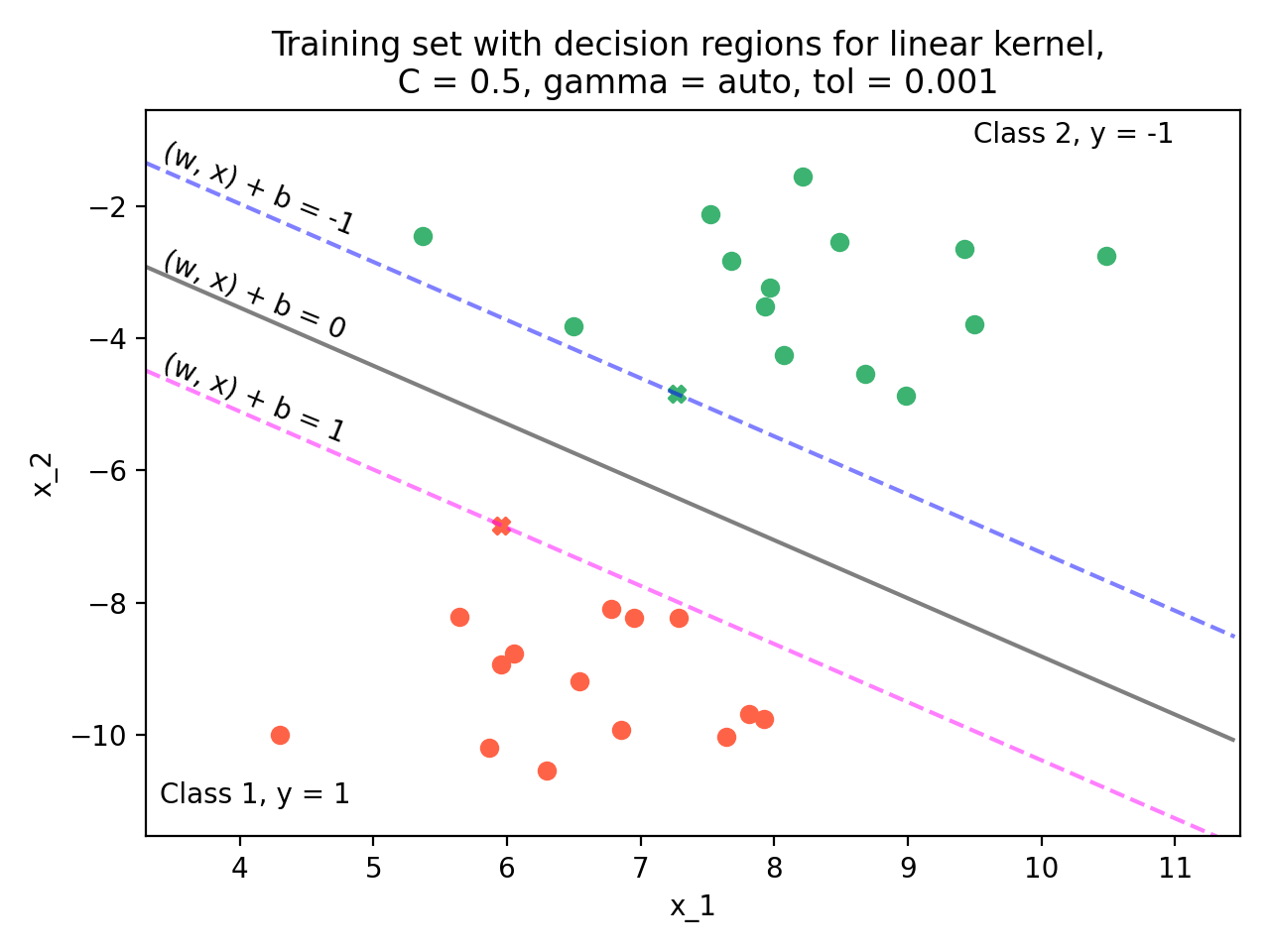
Генерируем множество из элементов. Запускаем алгоритм SOR. Он находит и за итерации при и заданной толерантности :

Варьируя параметром можно уменьшить количество итераций. Мною найдено , что даёт итераций.

Для наших данных получаем следующий результат:

|  |  |  |
| --- | --- | --- |
|  |  | *Support or Outside* |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Support |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Outside |
|  |  | Support |

На картинке ниже красные и зелёные элементы – сгенерированное множество, два класса. Серая линия – разделяющая гиперплоскость, цветные прерывистые линии – поддерживающие гиперплоскости.



## 4.2. Нелинейный SV-классификатор

Теперь рассмотрим линейно неразделимые данные. Самый распространённый пример – это когда объекты одного класса сосредоточены в круге, а объекты другого – вне этого круга.

Сгенерируем искусственный набор таких данных. При настройке и по грубой решётке были найдены и . В итоге получаем 10 опорных векторов. На рисунке ниже красные и зелёные элементы – сгенерированное множество, два класса. Крестиком отмечены опорные вектора. Серая линия – разделяющая гиперплоскость, цветные прерывистые линии – поддерживающие гиперплоскости.

# Изображение выглядит как карта Автоматически созданное описание5. Результаты работы реализации алгоритма на реальных данных

Посмотрим, как реализованный алгоритм SOR работает на реальных данных. Возьмём набор данных «German credit dataset». Этот набор данных представлен в двух датасетах, один из которых – полностью числовой, что подходит для нашего алгоритма. Датасет (размером 1000 элементов) представляет собой набор из 24 числовых атрибутов, которые характеризуют риски при предоставлении кредита, например, текущее состояние счёта, срок кредита в месяцах, кредитная история, цель, сумма кредита, пол и семейное положение, возраст, работа и т.д. В аннотации к датасету утверждается, что «лучше классифицировать клиента как плохого, когда он хороший, чем классифицировать его как хороший, когда он плохой».

1. Делим датасет из 1000 элементов на два множества:
   1. Тренировочное – 800 элементов.
   2. Тестовое – 200 элементов.
2. Работаем с тренировочным множеством. Используем процедуру кросс-валидации:
   1. Делим тренировочное множество на 5 частей, получаем по 160 элементов в каждой части.
   2. На каждом шаге будем обучать модель на 4 частях множества, а 1 будет служить для валидации. На валидационной части получаем ошибку .
   3. Средняя ошибка .
   4. Повторяем шаги 2.a – 2.с, используя различные значения гиперпараметров.
3. Выбираем лучшую модель из шага 2 (с наименьшей средней ошибкой ).
4. Обучаем выбранную модель на данных из тестового множества.
5. Оцениваем качество построенной машины на данных из тестового множества.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тренировочное множество | | | | | Тестовое множество | |
| 160 эл | 160 эл | 160 эл | 160 эл | 160 эл | 200 эл | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Был произведён поиск параметров по грубой сетке. На таблице ниже представлена средняя ошибка в зависимости от параметров и .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 10 | 100 | 1000 |
| 0.01 | 0.2725 | 0.2875 | - | - |
| 0.1 | 0.29625 | 0.305 | 0.30875 | 0.3025 |
| 0.5 | 0.29375 | 0.29375 | 0.31125 | 0.29125 |
| 1 | 0.29375 | 0.29375 | 0.3 | 0.29375 |

Лучший результат был получен при и .