**Министерство науки и высшего образования РФ**Государственное образовательное учреждение высшего профессионального   
образования **«Тихоокеанский Государственный университет»**

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №5**

по дисциплине: «Методы машинного обучения»

на тему: «Реализация алгоритма метода опорных векторов для задачи бинарной

классификации»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Забавин А.С.

Проверил: преподаватель кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

**Цель работы**: реализовать метод опорных векторов (SVM) для задачи бинарной классификации.

**Задания на лабораторную работу** (5 вариант):

1. Необходимо построить (реализовать на языке Python с применением пакета Scikit-Learn) линейный вариант метода опорных векторов, для следующих данных обучающей выборки (для своего варианта):

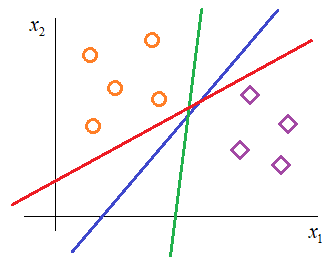
http://tk.ulstu.ru/files/iris data.py

1. Для данной обучающей выборки подсчитать число и процент неверных классификаций.
2. Отобразить обучающую выборку в виде графика точек на плоскости (объекты разных классов должны быть иметь разные маркеры и цвет), а также полученную (в результате обучения) разделяющую линию.

# Краткая теория

Метод опорных векторов (Sиpport Vector Machiпe - SVМ) — это очень мощная и универсальная модель машинного обучения, способная выполнять линейную или нелинейную классификацию, регрессию и даже выявление выбросов. Методы SVM особенно хорошо подходят для классификации сложных, но небольших или средних наборов данных.

Пусть образы обучающей выборки распределяются, следующим образом:



Тогда можно построить множество различных разделяющих линий (в общем случае, гиперплоскостей) так, что каждая из них будет корректно отделять один класс от другого. И здесь возникает вопрос, какое разделение лучше? В машинном обучении мы исходим из того, что модель, обученная на некоторой выборке, должна хорошо работать с другими произвольными наборами из того же распределения. То есть, модель должна иметь хорошие обобщающие способности (не быть слишком переобученной).

Идея проведения разделяющей гиперплоскости, которая бы ориентировалась только на распределение обучающей выборки и по возможности не делала бы дополнительных предположений о распределении образов в классах, положена в основу метода опорных векторов.

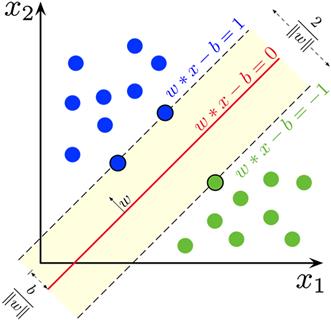
Чтобы описать эту идею на уровне математики, мы вначале должны задать модель классификатора, которая, фактически, определяет уравнение гиперплоскости в признаковом пространстве. Простейшая линейная модель:



Я ее записал здесь тремя способами, но все они означают линейную комбинацию вектора параметров ω с образом x плюс смещение -b. На выходе модель выдает значения:

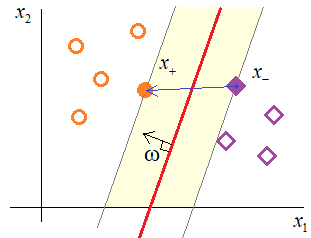


Далее, мы предположим, что наша обучающая выборка состоит из линейно разделимых образов (затем, мы этот случай обобщим на линейно неразделимый). Тогда ширина полосы будет определяться расположением граничных векторов x в признаковом пространстве:



Если взять любые два образа разных классов, расположенные ближе всего к разделяющей границе (то есть, лежащие на границе полосы), то ширину полосы можно вычислить как проекцию вектора на вектор :





То есть, получаем ширину, умноженную на длину вектора коэффициентов ω. Поэтому, окончательно, имеем:



И эту величину нужно максимизировать. Но для удобства в SVM в знаменателе записывают не длину вектора ω, а квадрат его нормы:



получаем выражение для максимизации ширины:



Принципиально такая замена картины не меняет, но зато мы точно не будем сталкиваться с вычислениями квадратных корней при решении оптимизационной задачи.

Наконец, сделаем еще одно, последнее упрощение. Как вы помните из предыдущих занятий, в задачах бинарной классификации вводится понятие отступа (margin):



Эта величина характеризует расстояние от разделяющей гиперплоскости до выбранного образа. Причем:



Но, так как мы рассматриваем случай линейно-разделимых образов, то заведомо существуют такие значения ω и b, что:



Так вот, нам ничто не мешает умножить эту величину на некоторое число:



В результате, получим:



Сути оптимизационной задачи это не изменит. Но зато мы теперь можем подобрать такое значение α и изменить параметры:



чтобы для любых образов , лежащих на границе полосы, величина:



Строго математически эту нормировку можно записать, следующим образом:



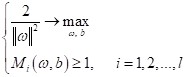
Очевидно, что для линейно-разделимого случая мы всегда можем это сделать. Но тогда автоматически получается, что:



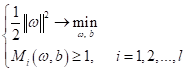
и ширина полосы будет определяться выражением:



В результате для линейно-разделимых образов мы получаем следующую оптимизационную задачу:



Но ее привычно сводят к задаче минимизации:



То есть, нам нужно найти такие ω и b, чтобы минимизировать квадратичную норму весов и вместе с тем обеспечить все отступы больше единицы, кроме тех образов, что лежат непосредственно на границах полосы (там отступ должен быть равен единице).

Это называется задачей квадратичного программирования, когда мы минимизируем квадраты весовых коэффициентов при линейных ограничениях неравенствах.

# Результаты работы

Работа была выполнена на языке программирования Python 3. Код программы представлена на Листинге 1.

import numpy as np # 1.23.4

import matplotlib.pyplot as plt # 3.6.1

from sklearn import svm

if \_\_name\_\_ == *"\_\_main\_\_"*:

debug = True

#===============================================================================

# Обучающая выборка

#===============================================================================

# вариант 5

data\_x = [(5.8, 1.2), (5.6, 1.5), (6.5, 1.5), (6.1, 1.3), (6.4, 1.3), (7.7, 2.0), (6.0, 1.8), (5.6, 1.3), (6.0, 1.6), (5.8, 1.9), (5.7, 2.0), (6.3, 1.5), (6.2, 1.8), (7.7, 2.3), (5.8, 1.2), (6.3, 1.8), (6.0, 1.0), (6.2, 1.3), (5.7, 1.3), (6.3, 1.9), (6.7, 2.5), (5.5, 1.2), (4.9, 1.0), (6.1, 1.4), (6.0, 1.6), (7.2, 2.5), (7.3, 1.8), (6.6, 1.4), (5.6, 2.0), (5.5, 1.0), (6.4, 2.2), (5.6, 1.3), (6.6, 1.3), (6.9, 2.1), (6.8, 2.1), (5.7, 1.3), (7.0, 1.4), (6.1, 1.4), (6.1, 1.8), (6.7, 1.7), (6.0, 1.5), (6.5, 1.8), (6.4, 1.5), (6.9, 1.5), (5.6, 1.3), (6.7, 1.4), (5.8, 1.9), (6.3, 1.3), (6.7, 2.1), (6.2, 2.3), (6.3, 2.4), (6.7, 1.8), (6.4, 2.3), (6.2, 1.5), (6.1, 1.4), (7.1, 2.1), (5.7, 1.0), (6.8, 1.4), (6.8, 2.3), (5.1, 1.1), (4.9, 1.7), (5.9, 1.8), (7.4, 1.9), (6.5, 2.0), (6.7, 1.5), (6.5, 2.0), (5.8, 1.0), (6.4, 2.1), (7.6, 2.1), (5.8, 2.4), (7.7, 2.2), (6.3, 1.5), (5.0, 1.0), (6.3, 1.6), (7.7, 2.3), (6.4, 1.9), (6.5, 2.2), (5.7, 1.2), (6.9, 2.3), (5.7, 1.3), (6.1, 1.2), (5.4, 1.5), (5.2, 1.4), (6.7, 2.3), (7.9, 2.0), (5.6, 1.1), (7.2, 1.8), (5.5, 1.3), (7.2, 1.6), (6.3, 2.5), (6.3, 1.8), (6.7, 2.4), (5.0, 1.0), (6.4, 1.8), (6.9, 2.3), (5.5, 1.3), (5.5, 1.1), (5.9, 1.5), (6.0, 1.5), (5.9, 1.8)]

data\_y = [-1, -1, -1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, 1, -1, -1, -1, -1, 1]

#===============================================================================

clean\_data = list(set(zip(data\_x, data\_y))) # чистим данные от дублей

classes = {

*'green'*: {

*'data'*: np.array([list(data[0]) + [1] for data in clean\_data if data[1] == -1]),

*'label'*: *'Образы 1 класса'*,

*'alpha'*: 1,

},

*'blue'*: {

*'data'*: np.array([list(data[0]) + [1] for data in clean\_data if data[1] == 1]),

*'label'*: *'Образы 2 класса'*,

*'alpha'*: 1,

}

}

x\_train = np.concatenate((classes[*'green'*][*'data'*], classes[*'blue'*][*'data'*]))

y\_train = np.concatenate(

(

np.full(classes[*'green'*][*'data'*].shape[0], -1),

np.full(classes[*'blue'*][*'data'*].shape[0], 1)

)

)

# regularization parameter

clf = svm.SVC(kernel=*'linear'*, gamma=0.7, C=1.0) # SVM с линейным ядром

clf.fit(x\_train, y\_train) # нахождение вектора w по обучающей выборке

v = clf.support\_vectors\_ # выделение опорных векторов

w = clf.coef\_[0]

# Перехват (также известный как смещение)

# добавлен в функцию принятия решения. (тета 0)

w0 = clf.intercept\_

# координаты разделяющей линии по осям

dividing\_line\_xx = [np.min(x\_train[:, 0]), np.max(x\_train[:, 0])] # относительно оси 0 (х)

dividing\_line\_yy = np.dot((-1. / w[1]), (np.dot(w[0], dividing\_line\_xx) + w0))

# print(f"Разделяющая линия = {w}", f"Опорные вектора = {v}", sep='\n')

y\_pr = clf.predict(x\_train) # проверка на обучающей выборке

# нули - без ошибок; иначе - ошибка

number\_of\_errors = x\_train.shape[0] - np.count\_nonzero((np.array(y\_train) - np.array(y\_pr)) == 0) # .count(0)

error\_rate = 100 \* number\_of\_errors / x\_train.shape[0]

# Построение графиков ----------------------------------------------------------

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 8))

fig.suptitle(*f'SVM\n(вариант №5, размер выборки: {len(clean\_data)})'*, fontsize=16)

for class\_i in classes:

ax.scatter(

classes[class\_i][*'data'*][:, :1],

classes[class\_i][*'data'*][:, 1:2],

color=class\_i, label=classes[class\_i][*'label'*],

alpha=classes[class\_i][*'alpha'*],

)

ax.scatter(

v[:, 0], v[:, 1],

s=150, edgecolor=None,

alpha=0.5, color=*'red'*,

linewidths=1, marker=*'+'*,

label=*'Точки опорного вектора'*

)

ax.plot(dividing\_line\_xx, dividing\_line\_yy, color=*'orange'*)

ax.tick\_params(labelcolor=*'indigo'*)

ax.legend()

ax.set\_title(

*'Распределение классификатора: '* +

*f"{number\_of\_errors} ошибок = "* +

*f"{error\_rate} %"*,

color=*'black'*

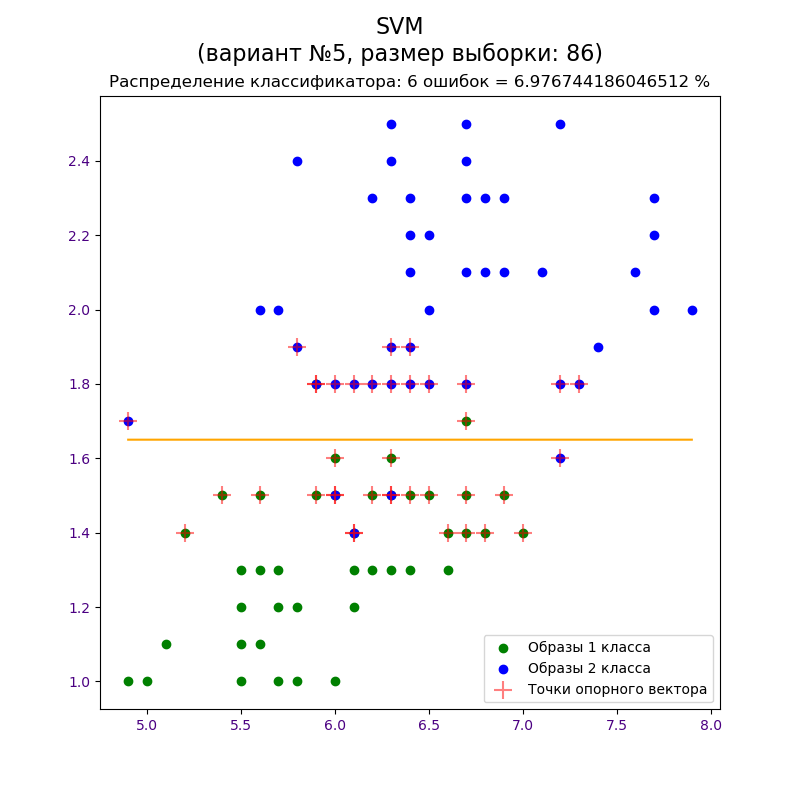
)

plt.show()

# ------------------------------------------------------------------------------

Листинг 1. Код программы main.py

В результате работы программы был получен следующий график :

Рисунок 4. Результаты работы программы

# Вывод

В ходе лабораторной работы был построен линейный вариант метода опорных векторов, определено число и процент ошибок, построен соответствующий график.