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

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №2**

по дисциплине: «Методы машинного обучения»

на тему: «Обучение линейного алгоритма бинарной классификации образов с

помощью градиентного алгоритма»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Забавин А.С..

Проверил: преподаватель кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

**Цель работы**: научиться реализовывать алгоритм градиентного спуска для задачи обучения линейной модели бинарной классификации образов.

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

В файле iris\_data.py даны обучающие выборки (по вариантам) для обучения линейного алгоритма бинарной классификации образов:

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

Модель линейного алгоритма должна иметь вид:

,

где - весовые коэффициенты модели (определяют ориентацию разделяющей линии); - образ обучающей выборки;

– знаковая функция

То есть, метки классов принимают значения

Ваша задача **выполнить обучение линейной модели** (найти значения вектора весовых коэффициентов ) с помощью градиентного алгоритма (программы, написанной на языке Python), который должен минимизировать величину эмпирического риска:

где - нотация Айверсона (квадратные скобки возвращают 1, если условие в скобках истинно, и 0 - в противном случае). То есть, эмпирический риск показывает число неверных классификаций.

Так как градиентный алгоритм может минимизировать только гладкие, дифференцируемые функции, то величину следует сверху ограничить именно таким функционалом:

где - выбранная функция потерь (здесь – отступ).

Функция потерь (также, как и набор обучающих данных) определяется вариантом. Для 5 варианта:

|  |  |
| --- | --- |
| Функция потерь для реализации градиентного алгоритма | Производная функции потерь |
| логарифмическая |  |

В качестве начальных значений весовых коэффициентов можно взять следующие:

Шаг в градиентном алгоритме для коэффициента целесообразно выбрать побольше, а для коэффициентов - поменьше.

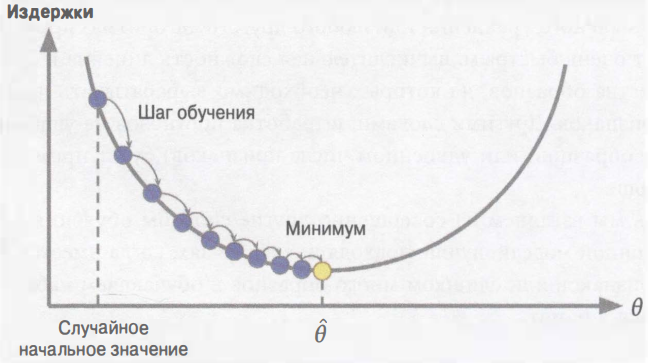
**Построить график** множества точек обучающей выборки (каждый класс точек должен быть представлен разными маркерами и цветами) и полученной разделяющей линии.

# Краткая теория

Градиентный спуск представляет собой самый общий алгоритм оптимизации, способный находить оптимальные решения широкого диапазона задач. Основная идея градиентного спуска заключается в том, чтобы итеративно подстраивать параметры для сведения к минимуму функции издержек.

Градиентный спуск выражается в измерении локального градиента функции ошибок применительно к вектору параметров и движении в направлении убывающего градиента. Как только градиент становится нулевым, был достигнут минимум!

Выражаясь более конкретно, все начинается с наполнения вектора случайными значениями (т.е. случайная инициализация). Затем происходит постепенное его уточнение по одному маленькому шагу за раз и на каждом шаге снижается функция издержек (например, MSE) до тех пор, пока алгоритм не сойдется в минимуме:



Важным параметром в градиентном спуске является размер шагов, определяемый гиперпараметром скорости обучения. Если скорость обучения слишком мала, тогда алгоритму для сведения придется пройти через множество итераций, что потребует длительного времени:

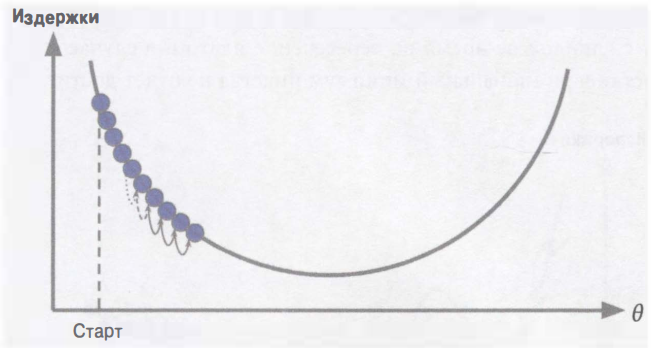


Рис. . Скорость обучения слишком мала

С другой стороны, если скорость обучения слишком высока, тогда можно перескочить «долину» и оказаться на другой стороне, возможно даже выше, чем ранее. Это способно сделать алгоритм расходящимся, что приведет к выдаче постоянно увеличивающихся значений и неудаче в поиске хорошего решения:

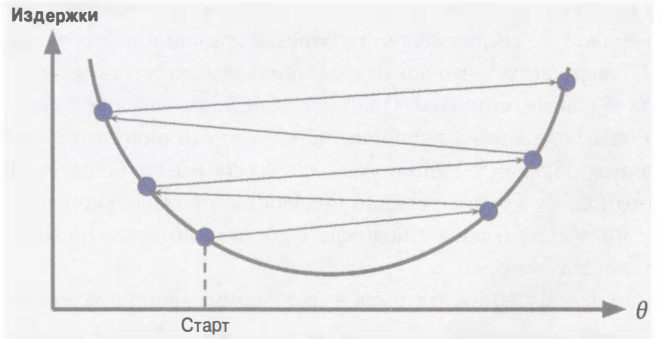
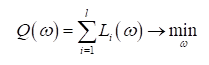
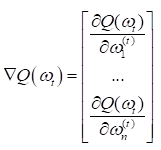


Рис. . Скорость обучения слишком высока

Как известно, эмпирический риск:



В этом случае мы имеем многомерную функцию и на каждой итерации нам нужно вычислять частные производные по каждому из значений:



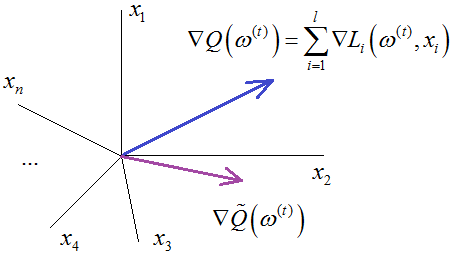
Получим вектор из частных производных, которым, затем, уточним значение вектора коэффициентов на новой итерации:



Если расписать эту формулу еще подробнее, то вместо градиента функционала у нас получится сумма градиентов функции потерь по обучающей выборке:



Получается, чтобы сделать один шаг классического градиентного спуска, нужно вычислить производные по функциям потерь для всех объектов обучающей выборки и сложить их, что ресурсоемко. Поэтому при практической реализации исходный градиент (всю сумму целиком) заменяют псевдоградиентом (субградиентом), который вычисляется значительно проще. И единственное требование к псевдоградиенту – его направление должно в среднем, на каждом шаге, образовывать острый угол с истинным градиентом.



Здесь через  обозначен псевдоградиент.

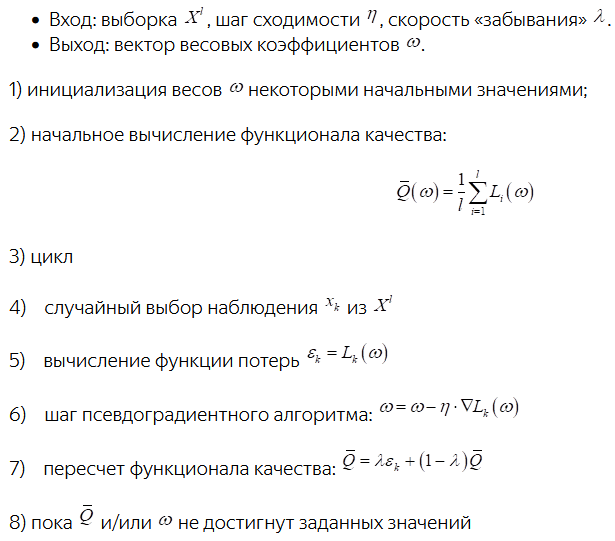
Возникает вопрос, что нам взять в качестве псевдоградиента, то есть, как сократить объем вычислений для истинного градиента и при этом сохранить сходимость алгоритма к точке локального минимума? Самое простое и радикальное, что можно сделать, на каждом шаге из всей суммы брать только одно наблюдение (одно слагаемое) и по нему вычислять текущее приближение:



где  - псевдоградиент; k – случайный индекс вектора из обучающей выборки.

Будет ли в этом случае наш псевдоградиент образовывать в среднем острый угол с истинным градиентом? Другими словами, будем ли мы в среднем двигаться в направлении локального минимума функционала качества? Да, можно показать, что по закону больших чисел, при большом числе шагов (приближений) этот псевдоградиент сходится к истинному градиенту. А, значит, рано или поздно мы достигнем с его помощью точки минимума.

Эта идея положена в основу метода стохастического градиентного спуска (Stochastic Gradient Descent – SGD). Иногда его еще называют методом Роббинса-Монро. Алгоритм можно записать в виде следующего псевдокода:



# Результаты работы

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

import numpy as np

import matplotlib.pyplot as plt

from itertools import chain

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

# 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]

# логарифмическая функция потерь

def **loss**(w, x, y):

*"""*

**:param** *w: вектор коэффициентов модели (весовых коэффициентов)*

**:param** *x: признаки х1 х2 (вектор признаков)*

**:param** *y: класс образа*

*"""*

M = np.dot(w, x) \* y

return np.log2((1 + np.exp(-M)))

# производная логарифмической функции потерь по вектору w

def **df**(w, x, y):

*"""*

**:param** *w: вектор коэффициентов модели (весовых коэффициентов)*

**:param** *x: признаки х1 х2 (вектор признаков)*

**:param** *y: класс образа*

*"""*

M = np.dot(w, x) \* y

return (-np.exp(-M) \* x \* y) / ((1 + np.exp(-M)) \* np.log(2))

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

# Очистка данных ===============

# Удалим дубликаты

data\_x\_plus\_y = zip(data\_x, data\_y)

data\_x\_plus\_y = np.array([[\*xy[0], xy[1]] for xy in data\_x\_plus\_y])

data\_x\_plus\_y = np.unique(data\_x\_plus\_y, axis=0)

data\_x = [list(xy)[:-1] for xy in data\_x\_plus\_y]

data\_y = [list(xy)[-1] for xy in data\_x\_plus\_y]

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

# Добавим к обучающей выборке добавим 1

# (w0 / линейное смещение w0\*x3/ (свободный коэффициент) / который не учавствует функции суммы градиента (скалярном произведении <w,x>),

# мы уточняем w отталкиваясь от w0

x\_train = [list(x) + [1] for x in data\_x]

x\_train = np.array(x\_train)

y\_train = np.array(data\_y)

n\_train = len(x\_train) # размер обучающей выборки

# Начальные коэффициенты алгоритма SGD ===============

w = [0.0, 0.0, 0.0] # начальные весовые коэффициенты

nt = 0.0045 # (эта) - шаг сходимости SGD (шаг обучения)

lm = 0.01 # (лямбда) - скорость "забывания" для Q (функционал качества, среднее ошибок)

N = 500 # число итераций SGD

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

# Оптимизация параметров. В исходной задаче было задано lm = 0.01 и N = 500 на 10 примеров,

N = N \* ((len(x\_train) // 10)) # Прямо пропорционально увеличим количество итераций

lm = lm \* ((len(x\_train) // 10) // 2) # Прямо увеличим скорость забывание в два раза за каждые 30 доп примеров

Q = np.mean([loss(w, x, y) for x, y in zip(x\_train, y\_train)]) # показатель качества при начальном w и для всей обучающенй выборки

Q\_plot = [Q]

for i in range(N):

k = np.random.randint(0, n\_train - 1) # случайный индекс

ek = loss(w, x\_train[k], y\_train[k]) # вычисление потерь для выбранного вектора

nt\_ = nt \* np.exp((-i / N)) # \* (1 - i / N) или \* np.exp((-i / N)) Улучшение сходимости алгоритма

w = w - nt\_ \* df(w, x\_train[k], y\_train[k]) # корректировка весов по SGD

Q = lm \* ek + (1 - lm) \* Q # пересчет показателя качества

Q\_plot.append(Q)

print(*f'Выбранные парамтеры для SGD алгоритма: nt={nt}, lm={lm}, N={N}'*)

print(*f'Весовые коэффициенты: {w}'*)

print(*f'Показатели качества (последние 15-ть): {Q\_plot[-15:]}'*)

def **classify**(x):

*"""*

*Классификатор, -1, +1*

**:param** *x:*

*"""*

asig = np.sign(x[0] \* w[0] + x[1] \* w[1] + w[2])

return asig

line\_x = [min(x\_train[:, 0]), max(x\_train[:, 0])] # формирование графика разделяющей линии # [:, 0] - питон 3 магия \_\_getitem\_\_ которую юзает numpy, эта запись значит "взять срез, тоесть копировать" и взять 0-й столбец из матрицы (первый)

line\_y = [-x \* w[0] / w[1] - w[2] / w[1] for x in line\_x]

x\_0 = x\_train[y\_train == 1] # формирование точек для 1-го

x\_1 = x\_train[y\_train == -1] # и 2-го классов

line\_sign = *'-'* if (-1 \* (w[2] / w[1])) < 0 else *'+'*

abs\_w2 = abs(w[2] / w[1])

plt.suptitle(*f'Метод SGD (градиентный спуск)\n (вариант №5)\n число итераций {N}, λ={lm}, η={nt} (ул. сход. exp)'*, fontsize=12)

plt.scatter(x\_0[:, 0], x\_0[:, 1], color=*'red'*, label=*f"C1=-1"*)

plt.scatter(x\_1[:, 0], x\_1[:, 1], color=*'blue'*, label=*f"C2=+1"*)

plt.plot(line\_x, line\_y, color=*'green'*, label=*f'Разделяющая линия, x2 = {0-w[0]/w[1]:.3f}\*x1 {line\_sign} {abs\_w2:.3f}'*)

plt.xlim([0, 45])

plt.ylim([0, 75])

plt.ylabel(*"x2"*)

plt.xlabel(*"x1"*)

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

plt.legend()

plt.grid(True)

plt.show()

Листинг 1. Код программы main.py

Результат работы программы приведен на Рисунке 5.

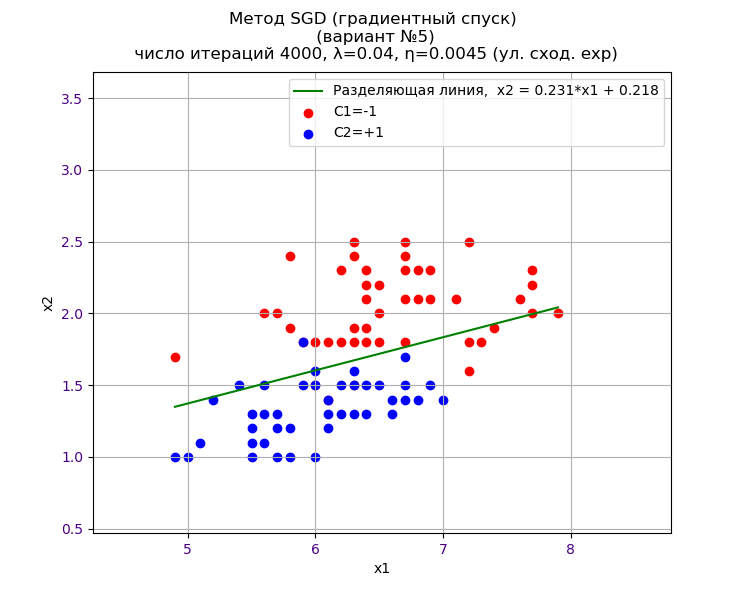


Рисунок 3. Результаты работы программы

Выбранные параметры для SGD алгоритма: nt=0.0045, lm=0.04, N=4000

Весовые коэффициенты: [-0.39549143 1.71249831 -0.37279951]

Показатели качества (последние 15-ть): [0.7142614770398351, 0.7256035995574865, 0.7332613925156632, 0.7433962197670051, 0.7296058499255498, 0.726543705408755, 0.7154093498018795, 0.709579549963005, 0.7224810545136885, 0.7192985884667091, 0.7424114117897676, 0.7442919566163192, 0.7446892156843706, 0.738537356693192, 0.7322125869636381]

# Вывод

В ходе лабораторной работы я выполнил обучение линейной модели (нашел значения вектора весовых коэффициентов ) с помощью градиентного алгоритма, построил соответствующие графики.