**Федеральное государственное образовательное**

**бюджетное учреждение**

**высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ»**

**(Финансовый университет)**

**Факультет**

**информационных технологий и анализа больших данных**

**Направление «Прикладная математика и информатика»**

**Домашнее задание № 8**

**«Методы стохастической оптимизации»**

Студенты группы ПМ19-3

Караваев Артем Евгеньевич

Лазаренко Владлена Владимировна

Минаков Артем Дмитриевич

Пластун Екатерина Сергеевна

Голомысов Даниил Владиславович

Руководитель:

Аксенов Дмитрий Андреевич

**Москва 2022**

Оглавление.

1. Математическая модель (постановка задачи)
2. Алгоритмы
   1. Алгоритм 1
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
   2. Алгоритм 2
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
   3. Алгоритм 3
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
3. Варианты использования системы
   1. ВИ 1
   2. ВИ 2
4. Архитектура решения
   1. Функции считывания информации
   2. Функции обработки информации
   3. Функции вывода информации
5. Тестирование
6. Выводы и заключение
7. **Постановка задачи (физическая модель)**

К нам пришел заказчик с новым проектом по нахождению оптимального способа реализации автомобилией. Фирма реализует автомобили двумя способами: через оптовую и розничную торговлю. При реализации х автомобилей в розницу расходы на реализацию составляют у. е., а при продаже у автомобилей оптом ‒ у. е. Найти оптимальный способ реализации автомобилей, минимизирующий суммарные расходы, если общее число предназначенных для продажи автомобилей составляет 200 шт.

1. **Математическая модель и алгоритмы**

Функция **L(x,y) =** ‒ суммарные расходы при реализации. По условию требуется найти минимум функции L.

Так как для продажи предназначено 200 автомобилей, то х и у связаны между собой условием связи: х + у = 200, х ≥ 0, у ≥ 0. Таким образом, получили задачу на условный экстремум с «простым» условием связи. Для решения такого типа задач, как известно из курса математического анализа, нужно из условия связи выразить одну переменную через другую, например, у через х: у = 200 − х и подставить полученное выражение в функцию L(х, у).

Тогда последняя превратится в функцию одной переменной.

**L(x) =**

* 1. **Алгоритм 1**

**Функция, решающая задачу поиска экстремума функции методом стохастического градиентного спуска (SGD).**

**Стохастический градиентный спуск** (англ. stochastic gradient descent) − оптимизационный алгоритм, отличающийся от обычного [градиентного спуска](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA) тем, что градиент оптимизируемой функции считается на каждом шаге не как сумма градиентов от каждого элемента выборки, а как градиент от одного, случайно выбранного элемента.

Единственное формальное отличие от обычного градиентного спуска — использование вместо градиента функции  такой, что  ( — математическое ожидание по случайной величине ), таким образом стохастический градиентный спуск имеет вид



* + 1. **Описание входных данных**

**Обязательные параметры:**

Обязательные параметры:

а) Функция в явном виде;

б) Ограничения типа равенства или неравенства в явном виде;

*\*количество ограничений и состав переменных в них может варьироваться в зависимости от выбора пользователя*

**Необязательные параметры:**

*\* определяются самостоятельно*

* + 1. **Описание алгоритма решения**

1. Задаем начальную точку х0. Выбираем рандомно (должна лежать в пределах eps – критерий останова, в предыдущих тз был по умолчанию ***0,00001, а maxk = 500***), ну или можно просто задать х0 = (0;0).
2. Находим градиент функции.

,  — математическое ожидание по случайной величине 

1. Далее будет вычисляться значение градиента функции в рандомной точке. Формула: .
2. Пока > eps или j (колво итераций) < maxk, то:

6.1. Вычисляем новые координаты точки:

, здесь alpha = const, подается на вход.

6.2. Вычисляем значение градиента в точке (x(k+1)).

6.3. Вычисляем значение функции в точке x(k+1).

* + 1. **Описание выходных данных**

а) координаты точки экстремума, значение функции в найденной координате экстремума.

* 1. **Алгоритм 2**

**Функция, реализующая модель классификации на два класса методом опорных векторов SVM с применением алгоритма градиентного спуска для минимизации функции ошибок (PEGASOS algorithm).**

Стохастический градиентный спуск — это метод оптимизации для задач безусловной оптимизации. В отличие от (пакетного) градиентного спуска, SGD аппроксимирует истинный градиент E(w,b) рассматривая один обучающий пример за раз.

* + 1. **Описание входных данных**

**Обязательные параметры:**

а) массив обучающей выборки (, , );

б) массив предсказываемой переменной ();

**Необязательные параметры:**

в) вид регуляризации (по умолчанию None, регрессия выполняется без нее);

г) построение графика классификации (по умолчанию False), определяет, будет ли построен график.

* + 1. **Описание алгоритма решения**

1. Класс [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html" \l "sklearn.linear_model.SGDClassifier" \t "_blank) реализует процедуру обучения SGD первого порядка. Алгоритм повторяет обучающие примеры и для каждого примера обновляет параметры модели в соответствии с правилом обновления, заданным , где η- это скорость обучения, которая контролирует размер шага в пространстве параметров. Перехватbобновляется аналогично, но без регуляризации (и с дополнительным распадом для разреженных матриц)
2. Скорость обучения ηможет быть как постоянным, так и постепенно затухающим. Для классификации расписание скорости обучения по умолчанию ( learning\_rate='optimal') задается следующим образом: , где t шаг по времени (всего шагов по времени **n\_samples \* n\_iter**), t0 определяется на основе эвристики, предложенной Леоном Ботту, так что ожидаемые начальные обновления сопоставимы с ожидаемым размером весов (это предполагает, что норма обучающих выборок составляет приблизительно.
3. Для регрессии график скорости обучения по умолчанию — обратное масштабирование ( learning\_rate='invscaling'), заданное как, где eta0 а также powert- гиперпараметры, выбираемые пользователем с помощью **eta0** и **power\_t**, соответственно.
4. Для постоянной скорости обучения используйте **learning\_rate='constant'** и используйте, **eta0** чтобы указать скорость обучения.
5. Для адаптивного уменьшения скорости обучения используйте **learning\_rate='adaptive'** и используйте **eta0**, чтобы указать начальную скорость обучения. Когда критерий остановки достигнут, скорость обучения делится на 5, и алгоритм не останавливается. Алгоритм останавливается, когда скорость обучения опускается ниже 1e-6.
6. Параметры модели могут быть доступны через coef\_ и intercept\_.

**2.2.3 Описание выходных данных**

а) массив предсказанных классов (в формате [,]);

б) массив коэффициентов регрессии ();

г) график классификации, если стоит соответствующий параметр.

1. **Варианты использования системы**

Предполагается, что пользователь будет предлагать программе на вход функцию для оптимизации, а также ограничения типа равенств или неравенств. Функция сама будет производить необходимые вычисления и возвращать координаты точки Х и значение У в ней.

В соответствии с нужным алгоритмом оптимизации пользователь будет выбирать нужную ему функцию среди двух, так как они будут иметь говорящие названия.

* 1. **ВИ Алгоритм 1**
     1. **Ввод данных.**

1. В первую очередь зададим ф-цию для оптимизации, создав функцию, для этого предпримем следующие шаги:

* Создаем ф-цию с помощью def(), в скобках указываем переменные x,y.
* Следующей строкой (с отступом!) прописываем Return - желаемую для оптимизации ф-цию, как показано ниже на примере:

****

2. Далее, присваиваем функции Z = f1(x,y):



3. С помощью встроенной ф-ции np.linspace задаем желаемую размерность графика.

Изображение выглядит как текст

Автоматически созданное описание

4. Полученные Z, x, y, а также желаемую точность - max\_epochs указываем в скобках у созданной функции def stochastic\_gradient\_descent().

****

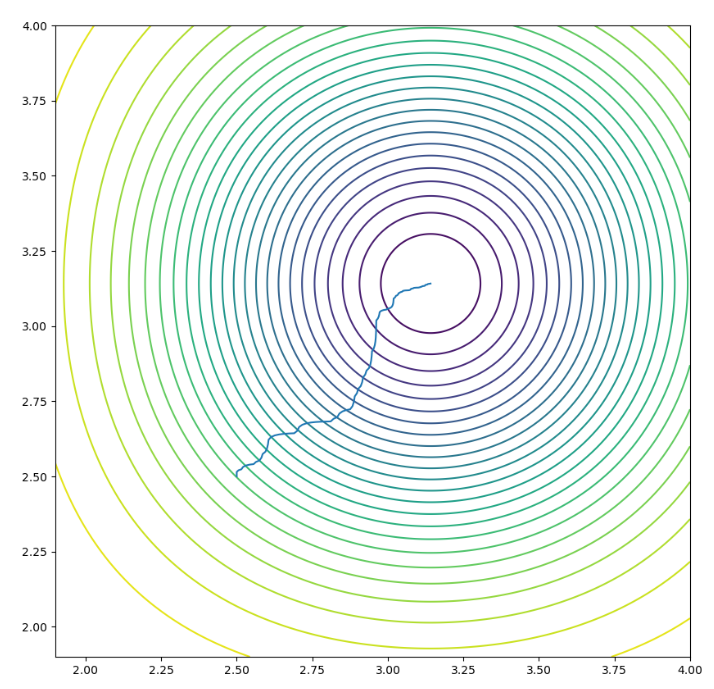
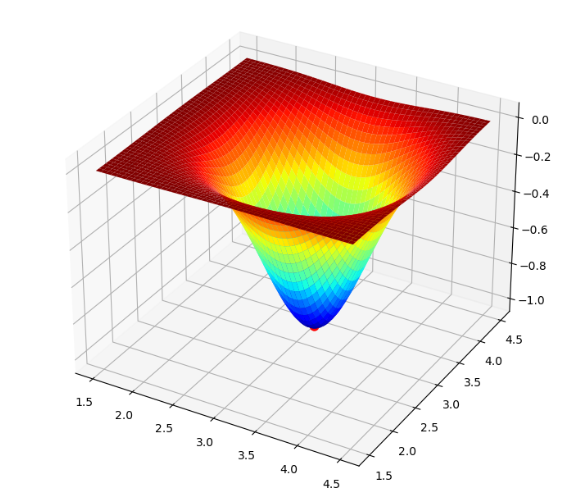
* + 1. **Вывод полученных результатов.**

На выход вы получите следующие результаты:

* **Координаты итоговой точки**

****

* **Графики**

**** ****

* 1. **ВИ Алгоритм 2**
     1. **Ввод данных.**

1. В первую очередь следует задать Х,У с помощью функции make\_blobs. Ниже пример данных размерностью 50 на 2, то есть n\_samples = 50, n\_features =2.

****

1. Далее следует разделить данные на тестовую и обучающую выборки в соотношении 20%.

Изображение выглядит как текст, внутренний

Автоматически созданное описание

1. Финальный шаг – вызов созданной функции SVM(). В скобках поочередно указываем созданные выше переменные x\_train, y\_train, x\_test, y\_test. Для построения графика укажите graph=True, в противном случае - graph=False.



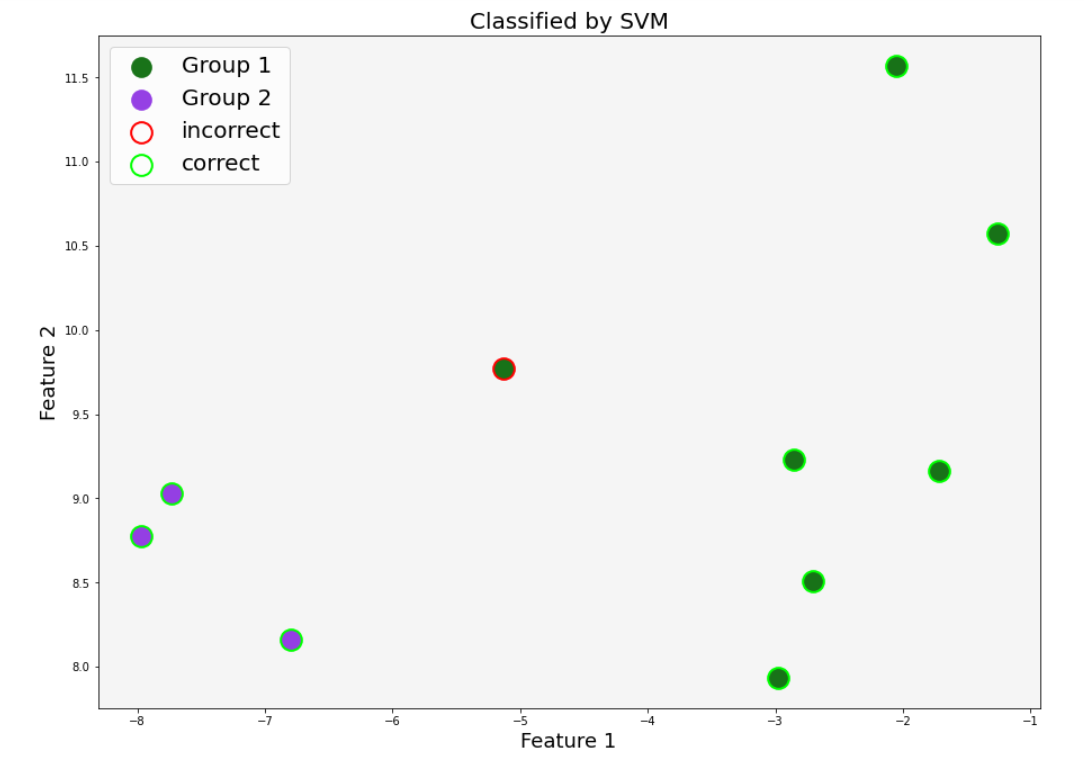
* + 1. **Вывод полученных результатов.**

На выход вы получите следующие результаты:

* **Коэффициенты регрессии**

****

* **График полученной классификации (если** graph=True)

****

* **Предсказанные и реальные значения**

**Изображение выглядит как текст

Автоматически созданное описание**

1. **Архитектура решения**
   1. **Функция, решающая задачу поиска экстремума функции методом стохастического градиентного спуска (SGD).**
      1. Функция для нахождения градиента функции

def SGD\_grad(X, Y):

"""

Функция для нахождения градиента функции

Parameters

----------

X: numpy.ndarray

Массив значений переменной x

Y: numpy.ndarray

Массив значений переменной y

Returns

-----------

np.array

Массив значений градиента

"""

* + 1. Функция для нахождения стохастического градиента функции

def stochastic\_gradient\_descent(max\_epochs, xy\_start, obj\_func, grad\_func):

"""

Функция для нахождения градиента функции

Parameters

----------

max\_epochs: int

Максимальное количество итераций для запуска

xy\_start: np.array

Начальная точка, с которой начинается градиентный спуск

obj\_func: function

Ссылка на функцию, которая вычисляет целевую функцию

grad\_func: function

Ссылка на функцию, вычисляющую градиент функции

Returns

-----------

xy\_history

Все точки в пространстве, посещенные градиентным спуском, в которых оценивалась целевая функция

f\_history

Соответствующее значение целевой функции, вычисленное в каждой точке

"""

* 1. **Функция, реализующая модель классификации на два класса методом опорных векторов SVM с применением алгоритма градиентного спуска для минимизации функции ошибок (PEGASOS algorithm).**

def SVM(X\_train : list, y\_train : list, x\_test : list, y\_test: list, regularization = None , graph = False):

'''

PEGASOS algorithm

Returns an array of predicted classes, an array of regression coefficients and a classification graph.

Parameters

----------

X\_train : list

Train data

y\_train : list

Classes for train data

x\_test : tuple

Test data

regularization : str

Type of regularization.

graph : bool, default=False

A parameter that determines whether a graph is needed

'''

1. **Тестирование**

Таблица 1. Результаты тестирования алгоритма 1 на заданных данных.

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Алгоритм 1** | **Скорость выполнения** |
| -np.cos(x)\*np.cos(y)\*np.exp(-((x-np.pi)\*\*2+(y-np.pi)\*\*2) | [3.14027939, 3.14143054] |  |
| (x+2\*y-7)\*\*2+(2\*x+y-5)\*\*2 | [1.00032212, 2.99965379] |  |
| 2\*x\*\*2+2\*y\*\*2+2\*x\*y+20\*x+10\*y+10 | [-4.99999858e+00, -1.44832908e-06] |  |

Таблица 2. Результаты тестирования алгоритма 2 на заданных данных.

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Алгоритм 2** | **Скорость выполнения** |
| Данные размерностью 50 на 2 | Коэффициенты регрессии = [-6.86181545 -2.61064481 -4.75211008]  Разделение на 2 группы, одна точка неверно  предсказана | 7.8559782505035 seconds |
| Данные размерностью 80 на 4 | Коэффициенты регрессии = [-5.68624091e-02 -2.64548985e-02 2.19041707e-01 -1.36832345e-01  -1.39982628e-18]  Разделение на 2 группы | 7.8559782505035 seconds |
| Данные размерностью 50 на 6 | Коэффициенты регрессии = [ 0.02744429 -0.03628793 0.04077988 -0.04866457 -0.00714394 -0.10635275  0.00509305]  Разделение на 2 группы | 112.01089429855seconds |

1. **Заключение**

**Решение методом стохастического градиентного спуска:**

Время: 0.7411296367645264, Решение: [-5.95763562e+01, -3.40832328e-12].

1. **Таблица 3. Сравнение алгоритмов.**

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Стохастического градиентного спуска** | **SGM** |
| Количество возможных параметров | Не ограничено | Не ограничено |
| Требования к памяти | Нет | Нет |
| Время выполнения | 0.7411 | 0.0418810 |
| Удобство использования | **+** | **+** |