Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

**Отчет по лабораторной работе №1**

**по курсу «Тестирование и отладка ПО»**

Студент: Тагер А.Д.

Группа: ИУ7-73

Преподаватель: Рогозин О.В.

Москва, 2016

1. **Цель лабораторных работ**

Написать спецификацию какого-либо программного продукта и протестировать его с помощью модульных, интеграционных, функциональных и системных тестов.

1. **Спецификация программного продукта**

**Описание задачи:** Дана выборка, состоящая из значений инвестиций компаний в рекламу по ТВ, в газетах и по радио, а также выручка компании. Необходимо построить регрессор для прогнозирования выручки компании, в зависимости от ее инвестиций.

**Описание реализации**: Была разработана программа, которая находит веса признаков для линейного регрессора. Данные считываются из файла, результат выводится в файл.

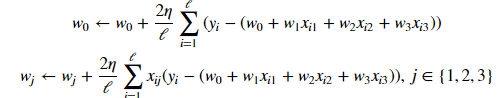
Язык реализации: Python 3.5

**Описание алгоритма:** Линейная регрессия – используемая в статистике регрессионная модель одной переменной y от нескольких других переменных x с линеной функцией зависимости. Определим модель зависимости как y_i= w_1 + w_2x_i + \varepsilon_i.. Искомый вектор весов можно найти двумя способами:

1. Нахождение вектора оптимальных весов w может быть сделано аналитически. Мы хотим найти такой вектор весов w, чтобы вектор y, соответствующий целевому признаку, получался умножением матрицы X (состоящей из всех признаков объектов обучающей выборки, кроме целевого) на вектор весов w. То есть, чтобы выполнялось матричное уравнение y = X \* w. Согласно методу наименьших квадратов, искомый вектор параметров есть решение нормального уравнения .

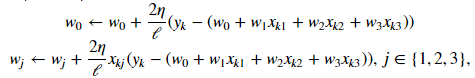
Матрица - псевдообратная для матрицы X. Но все же на практике для больших матриц X быстрее работает градиентный спуск, особенно его стохастическая версия

2 Параметры w0,w1,w2,w3, по которым минимизируется среднеквадратичная ошибка, можно находить численно с помощью градиентного спуска. Градиентный шаг для весов будет выглядеть следующим образом.



Здесь η - параметр, шаг градиентного спуска.

У градиентного спуска, описанного выше, есть один недостаток. На больших выборках вычисление градиента по всем имеющимся данным на каждом шаге может быть вычислительно сложно. В стохастическом варианте градиентного спуска поправки для весов вычисляются только с учетом одного случайно взятого объекта обучающей выборки:



где k - случайный индекс, k∈{1,…,ℓ}.

1. **Описание абстрактных классов и методов:**

* Class **Matrix** – класс для нахождения весов аналитически.
* Class **SGD** – класс для нахождения весов с помощью стохастического градиентного спуска.
* Class **Vector** – класс для работы с вектором, в данной задаче – с вектором значений целевой переменной.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Класс **Matrix**:

Метод **shape(Marr):** возвращает размер матрицы

Входные данные: Матрица

Выходные данные: массив, состоящий из размеров матрицы

Метод **T(Marr):** транспонирует матрицу:

Входные данные: Матрица

Выходные данные: транспонированная матрица

Метод **scales(Marr):** масштабирует матрицу

Входные данные: Матрица

Выходные данные: из каждого элемента каждого стоца матрицы вычитается его среднее значение и делится на стандартное отклонение значений этого столбца.

Метод **add\_ones\_col\_left(Marr):** Добавляет единичный столбец слева к матрице

Входные данные: Матрица

Выходные данные: матрица с единичным стобцом слева

Метод **inv(Marr):** находит обратную матрицу

Входные данные: Матрица

Выходные данные: обратная матрица

Метод **matmult**(a, b)**:** перемножает матрицы

Входные данные: 2 матрицы

Выходные данные: Произведение матриц

Метод **pseudo\_inverse**(Marr)**:** Возвращает псевдо-обратную матрицу

Входные данные: Матрица

Выходные данные: псевдо-обратная матрица

Метод **transform (self):** масштабирует матрицу и приписывает к ней единичный вектор слева.

Входные данные: нет

Выходные данные: трансформмированная матрица

Метод **weights(self, y):** находит веса для последующего предсказания

Входные данные: вектор y, соответствующий целевому признаку

Выходные данные: веса для линейной регрессии

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Класс **Vector**:

Метод **length(self):** возвращает длину вектора

Входные данные: нет

Выходные данные: значение длинны вектора

Метод **n(self, n):** Возвращает n-й элемент вектора

Входные данные: **n** - номер элемента

Выходные данные: n-й элемент

Метод **to\_array(self):** Возвращает вектор

Входные данные: нет

Выходные данные: вектор

Метод **mserror(self, y\_pred):** вычисляет значение средней квадратической ошибки

Входные данные: y\_pred - массив

Выходные данные: Значение средней квадратической ошибки

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Класс **SGD:**

Метод **euclidean(self, w\_new):** возвращает Евклидово расстояние между векторами

Входные данные: новый вектор весов

Выходные данные: значение Евклидово расстояние между новым и текущим векторами весов

Метод **def stochastic\_gradient\_descent (self, eta=1e-2, max\_iter=1e6, min\_weight\_dist=1e-8, seed=42, verbose=False):** возвращает результирующий вектор весов и вектор ошибок на каждой итерации

Входные данные: значение градиентного шага, максимальное число итереций, превысив которое прекратится вычисление весов, минимальное значение Евклидова расстояния между весами, получив которое прекращается спуск, значение для генерации псевдо случайных чисел.

Выходные данные: результирующий вектор весов и вектор ошибок на каждой итерации

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Спецификация модульных тестов**

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **matmult(self, b, a=None)**

Кассы эквивалентности для входных данных:

1. Матрицы, где в число столбцов в первой совпадает с числом строк во второй
2. Матрицы, где в число столбцов в первой не совпадает с числом строк во второй
3. Матица размера 1\*1
4. Пустые матрицы

Анализ:

Тест **test\_1:**

Вход: [[1,0,1], [0,1,5]] и [[1,2], [1,4], [0,4]]

Ожидаемый результат: [[1, 6], [1, 24]]

Тест **test\_2:**

Вход: матрицы [[1,0,1], [0,1,5]] и [[1,4], [0,4]]

Ожидаемый результат: **Error**

Тест **test\_3:**

Вход: [[1]] и [[3]]

Ожидаемый результат: **[[3]]**

Тест **test\_4:**

Вход: матрицы без элементов: [[]] и [[]]

Ожидаемый результат: **[[]]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **inv(self):**

Кассы эквивалентности для входных данных:

1. Пустая матица
2. Матрица с одним элементов
3. Прямоугольная матрица
4. Квадратная матрица с несколькими элементами
5. Матрица, с определителем равным 0

Анализ:

Тест **test\_5:**

Вход: **[[]]**

Ожидаемый результат: **[[]]**

Тест **test\_6:**

Вход: **[[2]]**

Ожидаемый результат: **[[0.5]]**

Тест **test\_7:**

Вход: **[[2, 1]]**

Ожидаемый результат: **Error**

Тест **test\_8:**

Вход: [[2, 1], [2, 3]]

Ожидаемый результат: **[[0.75, -0.25], [-0.5, 0.5]]**

Тест **test\_9:**

Вход: [[4, 2], [2, 1]]

Ожидаемый результат: **Error**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **\_T(self):**

Кассы эквивалентности для входных данных:

1. Матрица без элементов
2. Матрица с одним элементом
3. Прямоугольная матрица
4. Квадратная матрица

Анализ:

Тест **test\_10:**

Вход: **[[]]**

Ожидаемый результат: [[]]

Тест **test\_11:**

Вход: **[[5]]**

Ожидаемый результат: [[5]]

Тест **test\_12:**

Вход: [[5, 1, 4], [8, 0, -1]]

Ожидаемый результат: [[5, 8], [1, 0], [4, -1]]

Тест **test\_13:**

Вход: [[5, 1], [8, 0]]

Ожидаемый результат: [[5, 8], [1, 0]]

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **\_scales(Marr):**

Кассы эквивалентности для входных данных:

1. Матрица без элементов
2. Матрица с одним элементом
3. Матрица, в которой стандартное отклонение элементов столбца != 0
4. Матрица, в которой стандартное отклонение элементов столбца == 0

Анализ:

Тест **test\_14:**

Вход: **[[]]**

Ожидаемый результат: [[]]

Тест **test\_15**

Вход: **[[2]]**

Ожидаемый результат: **[[1, 2]]**

Тест **test\_16:**

Входы: [[2, 1], [3,5]]

Ожидаемый результат: [[1, -1.0, -1.0], [1, 1.0, 1.0]]

Тест **test\_17:**

Входы: [[2, 1], [5, 1]]

Ожидаемый результат: [[1, -1.0, 1], [1, 1.0, 1]]

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **\_add\_one\_col\_left(Marr):**

Кассы эквивалентности для входных данных:

1. Матрица без элементов
2. Матрица с одной строкой
3. Матрица, с несколькими строками

Анализ:

Тест **test\_18:**

Вход: **[[]]**

Ожидаемый результат: [[]]

Тест **test\_19:**

Вход: **[[2]]**

Ожидаемый результат: **[[1, 2]]**

Тест **test\_20:**

Входы: [[2, 1], [3,5]]

Ожидаемый результат: [[1, 2, 1], [1, 3, 5]]

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Метод для тестирования: def **mserror(self, y\_pred):**

Кассы эквивалентности для входных данных:

1. Пустые вектора
2. Вектора разной длины
3. Вектора одинаковой длины с разными значениями
4. Вектора одинаковой длины с одинаковыми значениями

Анализ:

Тест **test\_1:**

Вход: **[], []**

Ожидаемый результат: **Error**

Тест **test\_2:**

Вход: **[1, 2, 3], [3, 1]**

Ожидаемый результат: **Error**

Тест **test\_3:**

Вход: **[1, 2, 3], [3, 1, 4]**

Ожидаемый результат: **2**

Тест **test\_4:**

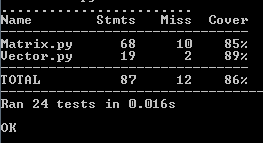
Вход: [[1, 2, 3], [1, 2, 3]]

Ожидаемый результат: **0**

1. **Покрытие кода тестами**

Для исследования объема покрытого тестами кода, использовалась сстандартная библиотека языка python – nose, из-за ее простоты, встроенного подсчета различных метрик, в частности процента покрытия кода, и встроенности в язык python.

В результате были получены следующие результаты:

****

Видно, что покрытие тестами методов класса Matrix составляет 85%. Это объяснимо, если учесть, что не были протестированы методы pseudo\_inverse, transform и weight, так как эти методы целиком состоят из вызовов уже протестированных методов.

Покрытие же тестами методов класса Vector составляет 89%, что так же объяснимо, так как не были протестированы методы length, n и to\_array из-за их простоты и использования в них стандартных функций языка python.