LABORATORY 3

DANIL VYSOTIN, OLEG CHISTOV AND VIONE KOROBOFF

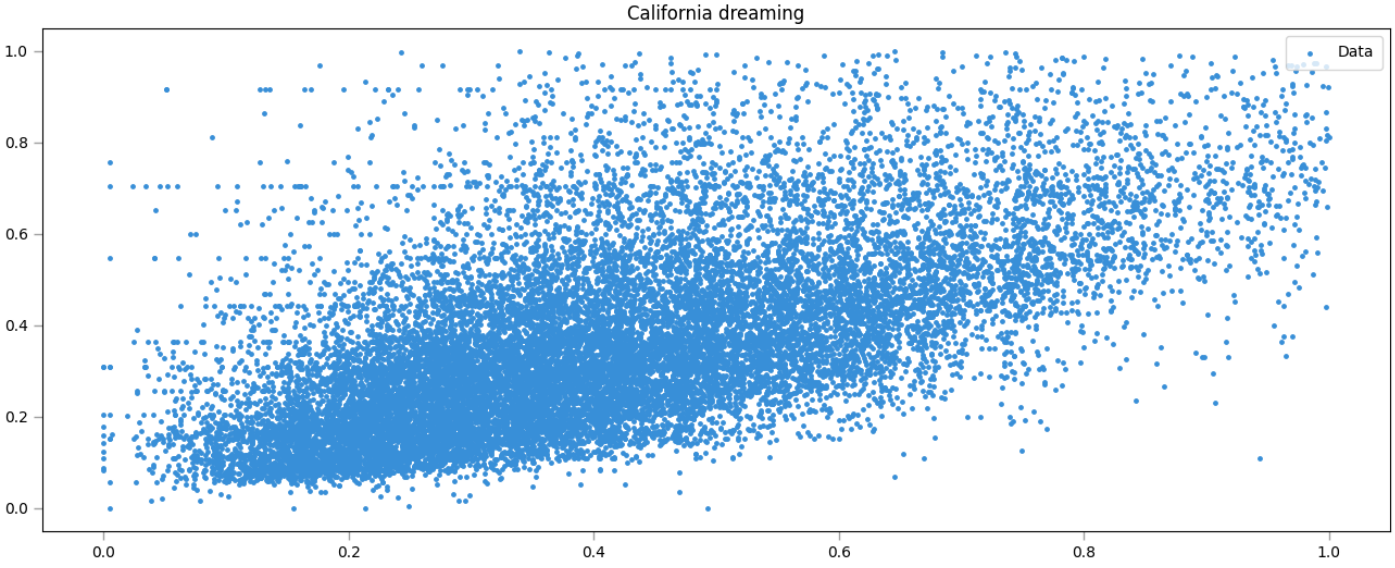
Аннотация. В работе был реализован и исследован *SGD* с его вариациями. Так же было проведено сравнение с методами из библиотеки *Keras*

1. Датасеты на которых проводилось исследование

В работе исследования проводились на классическом датасете *California housing* и некоторых кастомных.

Базовая часть лаборатории была исследована на немного модифицированном датасете *California housing*, а в бонусной части была реализована интерактивная среда для создания кастомных датасетов.

2. California Housing



2.1. **Сравнение эффективности реализация SGD с разным размером батча.** В каждом из следующих замеров использовался *learning rate = 0.01* $\epsilon = 10^{-6}$

Размер Батча	Время Работы	Погрешность значений	Использованной RAM mb	Количество Операций	Количество Итераций
1	0.57978	0.0244537	0.78125	13788	1531
100	1.25613	0.02432986	0.59375	30528	3391
500	1.90107	0.02386615	0.64062	47475	5274
1000	0.29086	0.02560209	0.0	7209	800
5000	0.71891	0.02447795	0.15625	17910	1989
19364	2.63361	0.02507691	0.0625	64881	7208

2.2. **Сравнение эффективности реализация SGD с разными функциями изменения шага.**

2.2.1. *Экспоненциальная функция изменения learning rate.* Изначально *learning rate = 0.5, learning rate decay = 0.95*

Размер Батча	Время Работы	Погрешность значений	Использованной RAM mb	Количество Операций	Количество Итераций
1	0.0713	0.02384018	0.23438	1739	173
100	0.03333	0.02545891	0.04688	849	84
500	0.07917	0.02681229	0.04688	2059	205
1000	0.05925	0.02453922	0.01562	1529	152
5000	0.07904	0.02548277	0.01562	2009	200
19364	0.07194	0.02451831	0.0	1899	189

Note: Использование экспоненциальной функции с правильно подобранными коефицентами заметно сократило количество итераций, не потеряв в точности.

2.2.2. *Ступенчатая функция изменения learning rate.* Изначально *learning rate = 0.5*, его ступенчатые изменения описаны ниже

```
lr_decay_ladder = {
    5: 0.1,
    50: 0.01,
    500: 0.0001,
    1000: 0.000001
}
```

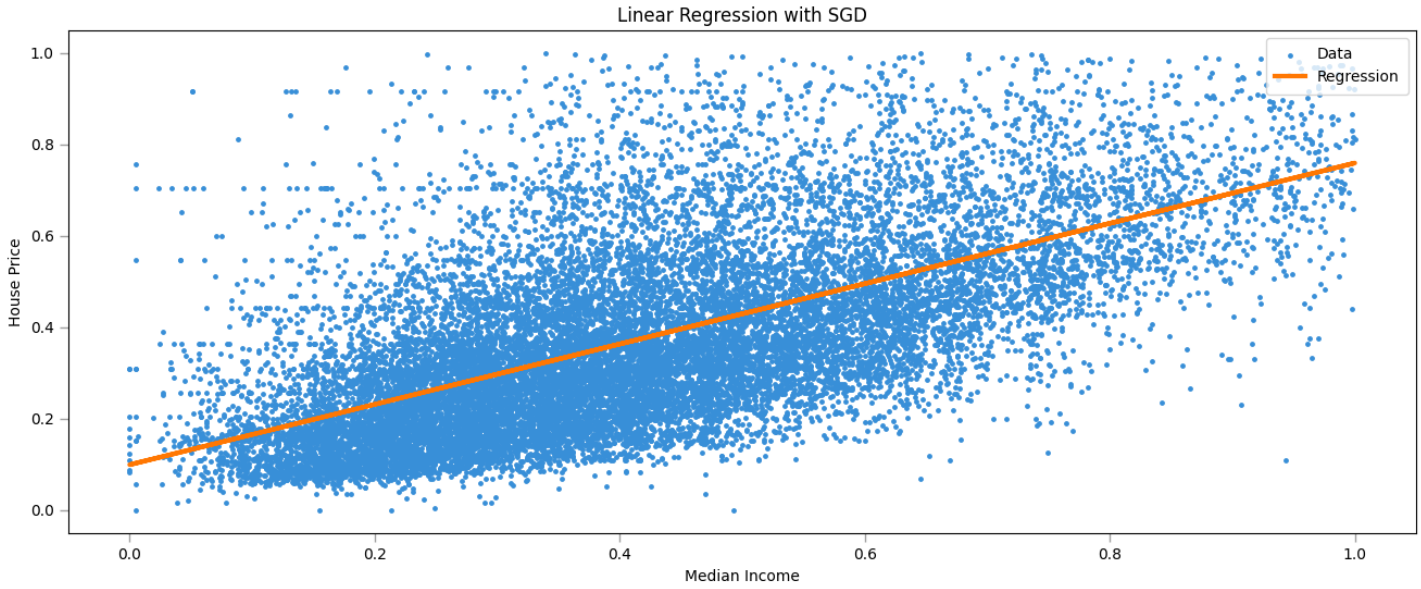
Размер Батча	Время Работы	Погрешность значений	Использованной RAM mb	Количество Операций	Количество Итераций
1	0.03541	0.02554098	0.3125	783	86
100	0.04296	0.02570353	0.21875	972	107
500	0.02857	0.02431373	0.03125	675	74
1000	0.03739	0.0262097	0.01562	882	97
5000	0.19457	0.02436997	0.125	4518	501
19364	0.17966	0.02597009	0.04688	4518	501

Note: Использование ступенчатой функции изменения *learning rate* с некоторых случаях сократило количество итераций, в других же увеличило его. Это произошло из-за того, что для всех размеров батчей использовалась одна и та же функция.
Стоит отметить, что потенциально ступенчатая функция может быть более эффективна нежели экспоненциальная. Так как она позволяет более гибко настраивать изменения *learning rate*

2.3. **Исследование методов из *Keras*.**

2.3.1. *Графики некоторых библиотечных методов.*

1. Решение линейной регрессии при $batch = 50$ используя keras.SGD



2. Решение линейной регрессии при $batch = 50$ используя keras.AdaGrad

2.3.2. *Сравнение производительности библиотечных методов.*

	Размер Батча	Время работы	Погрешность значений	Затраченная RAM mb
SGD	5	8.59497	0.0240505	17.20312
Nesterov	5	8.84991	0.02384317	5.98438
Momentum	5	8.87986	0.0245467	3.8125
AdaGrad	5	8.9582	0.02555547	3.85938
RMSProp	5	8.951	0.02388078	3.32812
Adam	5	9.26263	0.02424253	5.64062

	Размер Батча	Время работы	Погрешность значений	Затраченная RAM mb
SGD	1000	0.18875	0.04335436	3.34375
Nesterov	1000	0.2429	0.02443141	4.04688
Momentum	1000	0.30614	0.02831224	0.76562
AdaGrad	1000	0.24091	0.29233559	0.90625
RMSProp	1000	0.24339	0.02385386	1.21875
Adam	1000	0.27558	0.04796432	2.14062

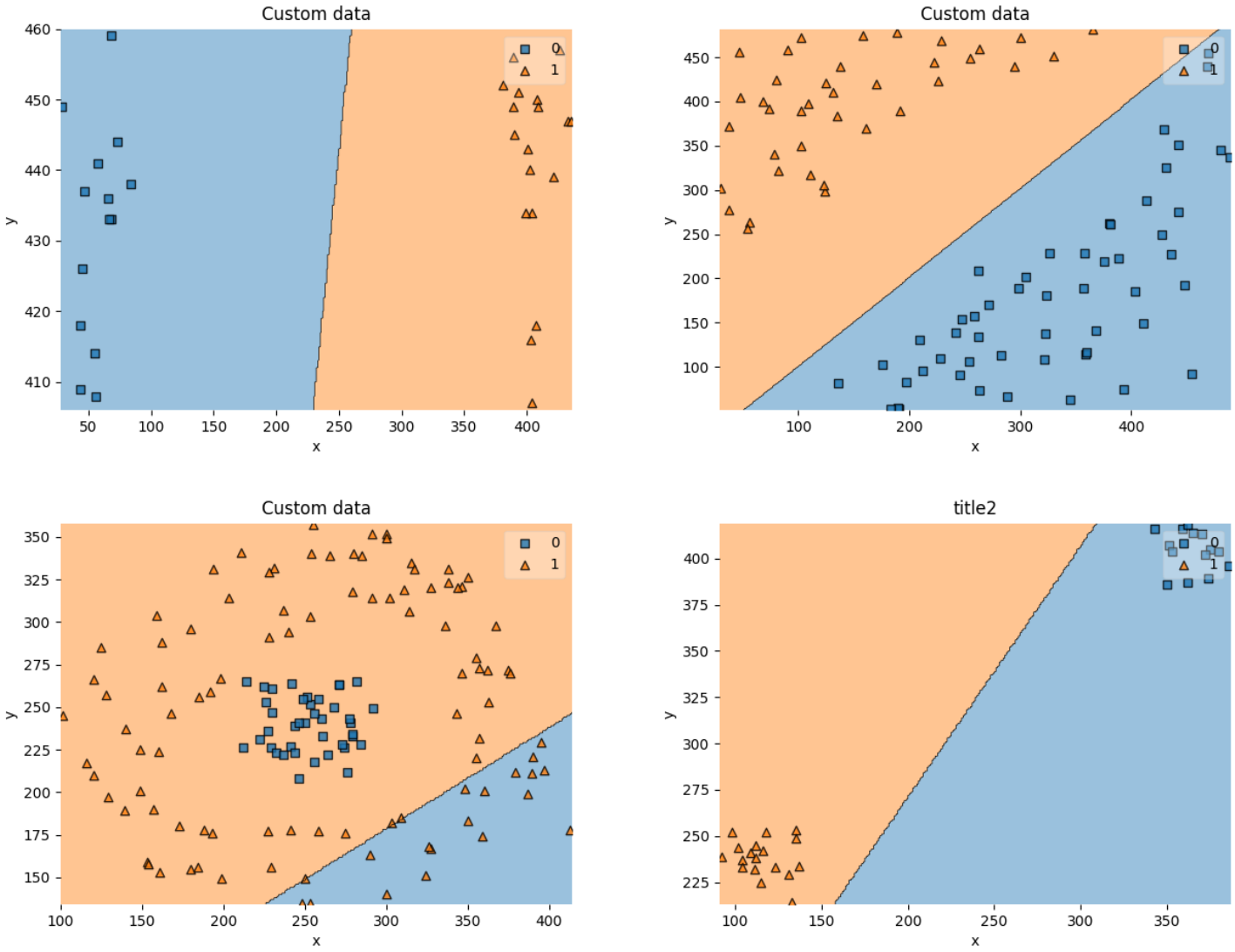
	Размер Батча	Время работы	Погрешность значений	Затраченная RAM mb
SGD	19364	0.12953	0.06858356	1.29688
Nesterov	19364	0.18536	0.03721288	2.67188
Momentum	19364	0.1666	0.14132479	1.3125
AdaGrad	19364	0.16845	0.28934432	2.8125
RMSProp	19364	0.17556	0.02875194	3.32812
Adam	19364	0.19802	0.02550725	3.15625

3. Дополнительное задание

В дополнительном задании был рассмотрен и реализован метод опорных векторов ***(linear support vector machine)***. Его эффективность была проверена на нескольких классических датасетах.

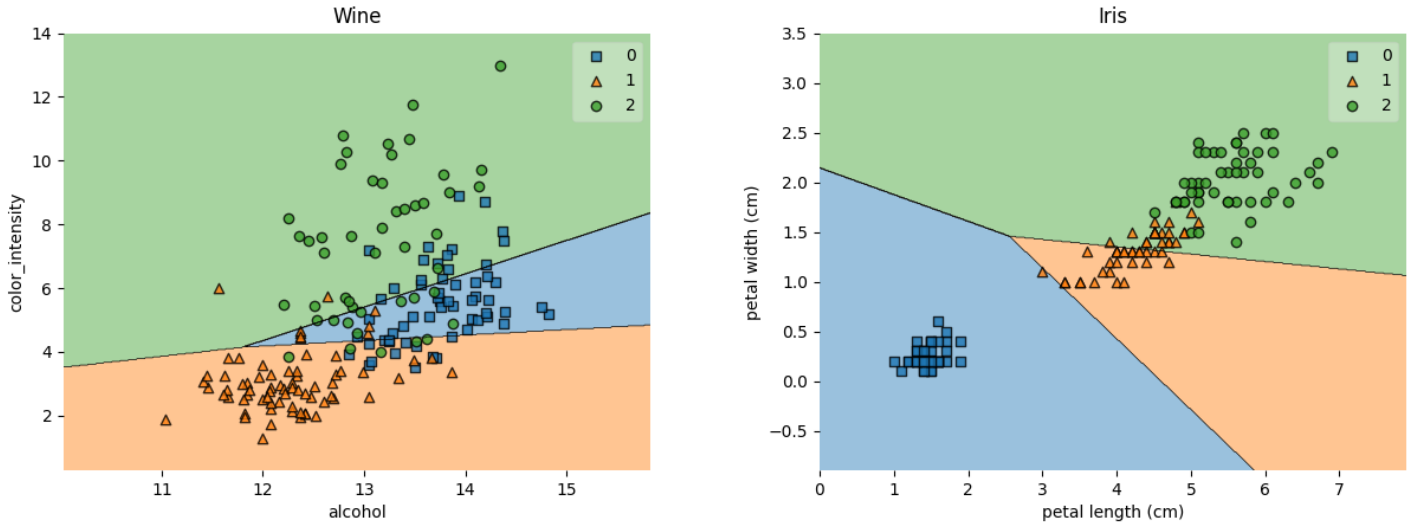
Кроме того, ради интереса и более точных тестов была реализована интерактивная среда, где можно самому создавать нужные датасеты, рисуя точки на плоскости.

3.1. **Решения задачи на кастомных данных, созданных с помощью интерактивной среды (по простому рисовалки).**



На третьем примере мы видим поведение программы на данных, которые нельзя класифицировать линейно.

3.2. **Решения задачи на классических датасетах: Вино и Ирисы.**



На примере этих датасетов видно, что несмотря на общий успех классификации зачастую при большом пересечении категорий, точность может страдать.

Ссылка на Github: Click here