

## Содержание

<b>I Модуль 1</b>	<b>13</b>
<b>1 Лекция 1. Линейный методы регрессии</b>	<b>13</b>
1.1 Напоминание о том, что такое линейная регрессия . . . . .	13
1.2 Общий вид линейной регрессии . . . . .	13
1.3 Как мы обучаем модель . . . . .	13
1.4 Проблемы возникающие при обучении на данных . . . . .	13
1.5 One-hot encoding - решение проблемы категориальных признаков . . . . .	13
1.5.1 Проблемы ОНЕ при обучении . . . . .	13
1.6 Решение проблемы немонотонных признаков . . . . .	14
1.6.1 Как разбивать эти переменные? . . . . .	14
1.7 Метрики . . . . .	14
1.7.1 Среднеквадратичное отклонение (MSE) . . . . .	14
1.7.2 RMSE . . . . .	14
1.7.3 R <sup>2</sup> . . . . .	15
1.7.4 MAE . . . . .	15
1.7.5 MAPE . . . . .	15
1.7.6 SMAPE . . . . .	15
1.7.7 MSLE . . . . .	16
1.7.8 Квантильная регрессия . . . . .	16
1.8 Небольшое дополнение по линейной регрессии в skelarn . . . . .	16
<b>2 Лекция 2 (Анастасия)</b>	<b>16</b>
2.0.1 Дополнительные ссылки для проверки . . . . .	16
2.1 Работа с кодом . . . . .	16
2.1.1 Скачивание и обработка данных . . . . .	16
2.1.2 Обучение модели и оценка качества . . . . .	17
2.2 Классическая предобработка данных для линейной регрессии . . . . .	17
2.3 Счётчики . . . . .	18
<b>3 Лекция 3. Логистическая регрессия. Лекция с ФЭН 2021</b>	<b>19</b>
3.1 Модель бинарной классификации . . . . .	19
3.1.1 . . . . .	19
3.1.2 Функции потерь . . . . .	19
3.2 Функция потерь . . . . .	20
3.3 Предсказание для логистической регрессии . . . . .	20
3.4 Функция потерь . . . . .	20
3.5 Вероятностная постановка задачи . . . . .	20
3.6 Правдоподобие и Log-Loss . . . . .	21
3.7 Выбор алгоритма предсказания . . . . .	21
3.8 Смысл (w, x) в логистической регрессии . . . . .	21
3.9 Логарифмическая функция потерь . . . . .	21
3.10 Бонус. Алгоритм Персептрона Розенблата . . . . .	21
<b>4 Лекция 3. Логистическая регрессия. Наша лекция</b>	<b>22</b>
4.1 Кодовая реализация . . . . .	22
4.1.1 Загрузка данных . . . . .	22
4.1.2 Функция для предсказания сигмоиды и функции потерь Log-Loss . . . . .	22
4.1.3 Градиент функции потерь . . . . .	22
4.1.4 Функция предсказания . . . . .	23
4.2 Различие гипер-параметров и параметров модели . . . . .	23

<b>5 Лекция 4. SVM, классификаторы, метрики качества классификации. Лекция с ФЭН</b>	<b>24</b>
5.1 Метод опорных векторов . . . . .	24
5.1.1 Линейно разделимая выборка . . . . .	24
5.1.1.1 Постановка задачи нахождения гиперплоскости . . . . .	24
5.2 Линейно неразделимая выборка . . . . .	24
5.2.1 Постановка задачи нахождения гиперплоскости . . . . .	24
5.3 Сравнение с логистической регрессией . . . . .	25
5.4 Метрики качества классификации . . . . .	25
5.4.1 Accuracy . . . . .	25
5.4.1.1 Недостаток . . . . .	25
5.4.2 Матрица ошибок . . . . .	25
5.4.3 Precision (Точность) . . . . .	25
5.4.4 Recall (Полнота) . . . . .	26
5.4.5 F-мера . . . . .	26
5.4.6 Регулирование точности и полноты . . . . .	26
5.4.7 ROC-AUC . . . . .	26
5.4.8 Индекс Джини . . . . .	26
5.4.9 Precision-Recall кривая . . . . .	26
<b>6 Лекция 4. SVM. Наша лекция</b>	<b>27</b>
6.1 Кодирование Категориальных признаков . . . . .	27
6.1.1 One-Hot Encoding . . . . .	27
6.1.1.1 Недостаток . . . . .	27
6.1.2 Счётчики . . . . .	27
6.1.2.1 Недостаток . . . . .	27
6.1.2.2 Сглаживание . . . . .	27
6.1.2.3 Отложенная выборка . . . . .	27
6.1.2.4 Кросс-валидация . . . . .	27
6.1.3 Expanding Mean . . . . .	27
<b>7 Лекция 5. Многоклассовая классификация, отбор признаков и методы снижения размерности. Лекция ФЭН</b>	<b>28</b>
7.1 Подходы многоклассовое классификации . . . . .	28
7.1.1 One-Vs-All . . . . .	28
7.1.2 All-Vs-All . . . . .	28
7.2 Метрики качества . . . . .	28
7.2.1 Micro-average . . . . .	28
7.2.2 Macro-average . . . . .	28
7.3 Многоклассовая логистическая регрессия . . . . .	28
7.3.0.1 Многоклассовый Log-Loss . . . . .	29
7.4 Отбор признаков . . . . .	29
7.4.1 Фильтрационные методы . . . . .	29
7.4.1.1 Преимущества . . . . .	29
7.4.1.2 sklearn . . . . .	29
7.4.1.3 Тест $\chi^2$ . . . . .	29
7.4.1.4 Mutual Information . . . . .	30
7.4.2 Обёрточные методы . . . . .	30
7.4.2.1 Recursive Feature Elimination . . . . .	30
7.5 Встроенные в модель методы . . . . .	30
7.5.1 $L_1$ регуляризация . . . . .	30
7.5.2 $L_0$ регуляризация . . . . .	30
7.5.3 Информационный критерий . . . . .	30
7.5.3.1 Критерий Акаике (AIC) . . . . .	30
7.5.3.2 Критерий Шварца (BIC) . . . . .	31
7.5.3.3 Отбор признаков с помощью информационных критериев . . . . .	31
7.6 Метод главных компонент (PCA) . . . . .	31
7.7 Manifold Embeddings . . . . .	32
7.7.1 MultiDimensional Scaling (MDS) . . . . .	32
7.7.2 ISOMAP . . . . .	32
7.7.3 TSNE . . . . .	32
7.7.3.1 Близость объектов в исходном пространстве . . . . .	32

<b>8 Лекция 6. Нелинейные алгоритмы классификации и регрессии. Лекция ФЭН</b>	<b>34</b>
8.1 Наивный байесовский классификатор . . . . .	34
8.1.1 Теорема Байеса . . . . .	34
8.1.1.1 В случае нескольких признаков . . . . .	34
8.1.1.2 Преимущества . . . . .	34
8.1.1.3 Недостатки . . . . .	34
8.2 Метод ближайших соседей . . . . .	34
8.2.0.1 Классификация нового объекта . . . . .	34
8.3 Решающие деревья . . . . .	34
8.3.1 Что влияет на построение решающего дерева . . . . .	35
8.3.2 Критерий информативности . . . . .	35
8.3.2.1 Функции потерь . . . . .	35
8.3.2.2 H(R) в задачах мягкой классификации . . . . .	35
8.3.2.3 Критерий Джини . . . . .	36
8.3.2.4 Энтропийный критерий . . . . .	36
8.3.3 Критерий останова . . . . .	36
8.3.4 Плюсы решающих деревьев . . . . .	36
8.3.5 Минусы решающих деревьев . . . . .	36
<b>9 Лекция 6. Решающие деревья. Наша Лекция. Елена</b>	<b>37</b>
9.1 ROC-AUC: интуиция . . . . .	37
9.1.1 ROC-AUC алгоритм . . . . .	37
9.2 Решающие деревья . . . . .	38
9.2.1 Задачи . . . . .	38
9.2.1.1 Задача 1 . . . . .	38
9.2.1.2 Решение задачи 1 . . . . .	38
9.2.1.3 Задача 3 . . . . .	38
9.2.1.4 Решение задачи 3 . . . . .	39
9.2.1.5 Задача 4 . . . . .	39
9.2.1.6 Решение задачи 4 . . . . .	39
9.3 Работа с кодом . . . . .	39
9.3.1 Подключение библиотек . . . . .	39
9.3.2 Генерация выборки для задачи регрессии . . . . .	39
9.3.3 Функция для обучения классификатора и построения разделяющей прямой . . . . .	39
9.3.4 Сравнение качества между логистической регрессией и деревом решений . . . . .	40
9.3.5 Генерация выборки логического ИЛИ . . . . .	40
9.3.6 Сравнение качества между логистической регрессией и деревом решений . . . . .	40
9.3.7 Переобучение дерева . . . . .	40
9.3.7.1 Генерация выборки . . . . .	40
9.3.7.2 Перебор гиперпараметров и отображение результата . . . . .	40
9.3.7.3 Дерево с нулевой ошибкой . . . . .	41
9.3.8 Неустойчивость . . . . .	41
9.3.9 Анализ деревьев на основе датасета Бостона . . . . .	41
9.3.9.1 Загрузка датасета . . . . .	41
9.3.9.2 Построение и отображение дерева с предикатами . . . . .	41
9.3.9.3 Получение построенных параметров дерева . . . . .	41
9.3.9.4 Построение зависимости MSE от гиперпараметра max_depth и отбор лучших значений на кросс-валидирующй выборке . . . . .	42
9.3.9.5 Построение зависимости MSE от гиперпараметра max_samples_leaf и отбор лучших значений на кросс-валидирующй выборке . . . . .	42
9.3.9.6 Разделение выборки на обучающую и тестовую . . . . .	42
9.3.9.7 Построение дерева без ограничивающих параметров . . . . .	42
9.3.9.8 Поиск лучших параметров max_depth и max_features . . . . .	43
9.3.9.9 Стрижка дерева . . . . .	43
9.3.9.10 Перебор по гиперпараметру регуляризации . . . . .	43
9.3.9.11 Применение лучшего параметра регуляризации . . . . .	44
9.3.10 Решающее дерево своими руками . . . . .	44
9.4 Калибровка вероятностей . . . . .	44
9.4.1 Программируем это всё дело . . . . .	44
9.4.1.1 Генерация выборки . . . . .	44
9.4.1.2 Обучение моделей . . . . .	44

9.4.1.3	Отобразим наши вероятности . . . . .	45
<b>10</b>	<b>Материалы к лекции 6</b>	<b>46</b>
10.1	Статья про разложение ошибки . . . . .	46
10.2	Уроки 6.4, 6.5 со степика . . . . .	47
<b>11</b>	<b>Лекция 7. Композиция алгоритмов, разложение ошибки и лес</b>	<b>48</b>
11.1	Смещение и разброс . . . . .	48
11.1.1	Формула для разложения ошибки на смещение и разброс . . . . .	49
11.2	Бэггинг . . . . .	49
11.2.1	Бутстрэп . . . . .	49
11.2.2	Описание бэггинга . . . . .	49
11.2.3	Смещение и разброс у бэггинга . . . . .	50
11.2.4	Как сделать некоррелированные алгоритмы . . . . .	50
11.3	Случайный лес (Random Forest) . . . . .	50
11.3.1	Практические рекомендации . . . . .	50
11.3.2	Out-Of-Bag ошибка . . . . .	50
<b>12</b>	<b>Лекция 7. Случайный лес. Наша лекция</b>	<b>51</b>
12.1	Программируем . . . . .	51
12.1.1	Загружаем датасет . . . . .	51
12.1.2	Обучим решающее дерево без ограничений . . . . .	51
12.1.3	Получим примерное смещение и разброс . . . . .	51
12.1.4	Посмотрим на смещение и разброс у бэггинга деревьев . . . . .	51
12.1.5	Посмотрим как это повлияло на MSE . . . . .	51
12.1.6	Построим случайный лес . . . . .	52
12.1.7	Переобучение случайного леса . . . . .	52
12.2	Важность признаков . . . . .	52
12.2.1	Программируем изучение важности признаков . . . . .	52
<b>13</b>	<b>Лекция 8. Бустинг. Лекция с ФЭН</b>	<b>54</b>
13.1	Случайный лес . . . . .	54
13.2	Бустинг . . . . .	54
13.2.1	Частный случай бустинга . . . . .	54
13.2.2	Алгоритм бустинга для MSE . . . . .	54
13.3	Выбор базовых алгоритмов . . . . .	55
13.4	Смещение и разброс бустинга . . . . .	55
13.5	Стохастический градиентный бустинг . . . . .	55
13.6	Имплементации градиентного бустинга . . . . .	55
13.6.1	Xgboost . . . . .	56
13.6.1.1	Особенности xgboost . . . . .	56
13.6.2	CatBoost . . . . .	56
13.6.2.1	Особенности CatBoost . . . . .	56
13.6.3	LightGBM . . . . .	57
<b>14</b>	<b>Лекция 8. Бустинги. Наша лекция</b>	<b>58</b>
14.1	Программируем . . . . .	58
14.1.1	Устанавливаем библиотеки . . . . .	58
14.1.2	Импорт библиотек . . . . .	58
14.1.3	Генерация датасета и функция для визуализации решений дерева . . . . .	58
14.1.4	Отобразим визуализацию дерева для CatBoost и вычислим ROC-AUC . . . . .	59
14.1.5	Обучаем обычный градиентный бустинг и анализируем качество от количества деревьев . . . . .	59
14.1.6	Сделаем то же самое для CatBoost . . . . .	59
14.1.7	Сравнение GradientBoostingClassifier и CatBoost . . . . .	60
14.1.8	Решение XGBoost . . . . .	60
14.2	CatBoost для решения задачи + интерпритация признаков . . . . .	61
14.2.1	Загрузка данных и разбитие для обучения и теста . . . . .	61
14.2.2	Обучим модель . . . . .	61
14.2.3	Построим предсказание . . . . .	61
14.2.4	Переберём параметры . . . . .	61

14.3	Оценка важности признаков . . . . .	62
14.3.1	Вспомогательные функции для оценки . . . . .	62
14.3.2	Визуализация важности признаков . . . . .	62
14.3.3	Визуализация важности признаков при случайной перестановке . . . . .	63
14.4	Блендинг и стекинг . . . . .	63
14.4.1	Стекинг . . . . .	63
14.5	Блендинг . . . . .	63
14.6	Программируем блендинг . . . . .	64
14.6.1	Загружаем данные . . . . .	64
14.6.2	Проверка качества алгоритмов, если просто обучить на train и проверить на test и блендинг на глазок . . . . .	64
14.6.3	Простой блендинг . . . . .	65
14.6.4	Подбор весов . . . . .	65
14.7	Полноценный блендинг . . . . .	65
14.7.1	Загрузка данных . . . . .	65
14.7.2	Использование блендинга . . . . .	65
14.8	Используем стекинг . . . . .	66
<b>15</b>	<b>Материалы к лекции 9. Многоклассовая классификация. Уроки со stepik</b>	<b>67</b>
15.1	Многоклассовая и multi-label классификация . . . . .	67
15.1.1	Сведение к бинарной классификации . . . . .	67
15.1.1.1	One-Versus-Rest . . . . .	67
15.1.1.2	One-Vs-One . . . . .	67
15.2	Метрики качества многоклассовой классификации . . . . .	68
15.2.1	Accuracy . . . . .	68
15.2.2	Recall и f1-score . . . . .	68
15.2.2.1	Макроусреднение (macro-average) . . . . .	69
15.2.2.2	Взвешенное усреднение (weighted-average) . . . . .	69
15.2.2.3	Микроусреднение (micro-average) . . . . .	69
15.2.3	Результат в python . . . . .	69
15.3	Метод ближайших соседей (KNN) . . . . .	70
15.3.1	Алгоритм метода . . . . .	70
15.3.2	Влияние гиперпараметра k . . . . .	70
15.3.3	Выбор метрики . . . . .	71
15.3.3.1	Евклидова метрика . . . . .	71
15.3.3.2	Манхэттенское расстояние . . . . .	71
15.3.3.3	Расстояние Хемминга . . . . .	71
15.3.3.4	Мера Жаккара . . . . .	71
15.3.4	Масштабирование данных для KNN . . . . .	71
15.3.5	Обобщения KNN . . . . .	72
15.3.6	KNN в задачах регрессии . . . . .	73
15.3.7	Преимущества и недостатки KNN . . . . .	74
15.3.8	Реализация в питоне . . . . .	74
15.4	Быстрый поиск соседей . . . . .	74
15.4.1	Хеширование . . . . .	75
15.4.2	Метод случайных проекций . . . . .	75
15.4.3	MiniHashing . . . . .	75
15.4.3.1	Мера Жаккара . . . . .	76
15.4.3.2	Матрица текстов . . . . .	76
15.4.3.3	MinHashing . . . . .	76
15.4.3.4	Locality-sensitive hashing (LSH) . . . . .	78
15.4.3.5	Гиперпараметры LSH . . . . .	79
<b>16</b>	<b>Лекция 9. Быстрый поиск ближайших соседей (LSH + kNN). Наша лекция</b>	<b>80</b>
16.1	Метод случайных проекций . . . . .	80
16.1.1	Загрузка данных . . . . .	80
16.1.2	Кодирование TfIdf . . . . .	80
16.1.3	Найдём похожие слова . . . . .	80
16.1.4	Сгенерируем для LSH случайные плоскости . . . . .	81
16.1.5	Получим наши LSH . . . . .	81
16.1.6	Переводим из массива True/False в десятичный вид . . . . .	81

16.1.7 Соберём всё в одну функцию . . . . .	81
16.1.8 Посмотрим на похожие объекты . . . . .	82
16.1.9 Напишем алгоритм поиска похожих слов, который учитывает количество различающихся позиций . . . . .	82
16.1.10 kNN . . . . .	83
16.1.11 Пример применения . . . . .	83
16.1.12 Проверка на адекватность . . . . .	83
16.2 MiniHashing . . . . .	84
16.2.1 Сделаем шинглы . . . . .	84
16.2.2 Сделаем шинглы ещё одного текста . . . . .	84
16.2.3 Проведём эксперименты minihash помощью библиотеки . . . . .	84
16.2.4 Увеличим кол-во перестановок . . . . .	84
16.2.5 Сделаем LSH по матрице сигнатур . . . . .	84
<b>17 Статья про SHAP</b>	<b>86</b>
17.1 Какую проблему решает SHAP . . . . .	86
17.2 Рассчитываем важность фичей с помощью SHAP . . . . .	86
17.2.1 Вступление из теории игр . . . . .	86
17.2.2 Пример рассчёта значения Шэпли . . . . .	86
17.2.2.1 Ситуация, когда нет значений фичей . . . . .	87
17.2.2.2 Ситуация, когда знаем пол . . . . .	87
17.2.2.3 Суммируем . . . . .	87
17.3 Реальный пример из бизнеса . . . . .	87
<b>18 Статья про LIME</b>	<b>89</b>
18.1 Интерпретация моделей машинного обучения . . . . .	89
18.1.1 Распределение данных . . . . .	89
18.1.2 Сдвиг данных . . . . .	90
18.1.3 Утечка данных . . . . .	91
18.1.4 Локальная интерпретация моделей . . . . .	91
18.2 LIME: Local Interpretable Model-agnostic Explanations . . . . .	92
18.2.1 Локальное упрощенное представление . . . . .	92
18.2.2 Объясняющая модель . . . . .	93
18.2.3 Примеры и обсуждение . . . . .	94
18.2.4 LIME-SP: объединение локальных интерпретаций в глобальную . . . . .	94
18.3 SHAP: Shapley Additive Explanation Values . . . . .	95
18.3.1 Shapley values в теории игр . . . . .	95
18.3.2 Shapley regression values . . . . .	95
18.3.3 SHAP values . . . . .	96
18.3.4 Проблемы и ограничения SHAP values . . . . .	97
18.3.5 Independent SHAP . . . . .	97
18.3.6 Kernel SHAP . . . . .	97
18.3.7 Tree SHAP . . . . .	99
18.4 SHAP на практике . . . . .	99
18.4.1 Waterfall plot . . . . .	99
18.4.2 Summary plot . . . . .	100
18.4.3 Dependence plot . . . . .	100
18.4.4 Диагностика сдвига данных с помощью SHAP loss values . . . . .	101
18.4.5 Supervised-кластеризация данных с помощью SHAP . . . . .	102
18.5 Проблемы SHAP values . . . . .	103
18.5.1 SHAP values в условиях взаимной зависимости признаков . . . . .	103
18.5.2 Соблюдение границ многообразия данных . . . . .	104
18.6 Shapley Flow . . . . .	105
18.6.1 Использование метода Shapley Flow . . . . .	105
18.6.1.1 Шаг 1. . . . .	105
18.6.1.2 Шаг 2. . . . .	105
18.6.1.3 Шаг 3. . . . .	105
18.6.2 Принцип работы метода Shapley Flow . . . . .	105
18.7 Заключение . . . . .	107

<b>19 Лекция 10. Интерпретация моделей: SHAP и LIME</b>	<b>108</b>
19.1 Практика . . . . .	108
19.1.1 Загружаем данные . . . . .	108
19.1.2 Используем модель с уже найденными гиперпараметрами . . . . .	108
19.1.3 Стандартный feature_importances . . . . .	108
19.1.4 Нарисует то же самое в виде гистограммы . . . . .	109
19.1.5 Оставляем только 10 самых важных признаков . . . . .	109
19.1.6 Обучение линейной модели . . . . .	109
19.1.7 Как изменился результат у бустинга . . . . .	109
19.2 Применяем LIME . . . . .	110
19.2.1 Найдём максимально верные и максимально неверные предсказания . . . . .	110
19.2.2 Используем LIME . . . . .	110
19.2.3 Посмотрим на объяснение в том случае, когда модель предсказывает максимально плохо . . . . .	110
19.2.4 Представление в ноутбуке . . . . .	110
19.3 Используем SHAP . . . . .	111
19.3.1 Объясним самое плохое предсказание с помощью SHAP . . . . .	111
19.3.2 Общие результаты . . . . .	111
<b>20 Лекция 11. Кластеризация и визуализация данных. Лекция ФЭН</b>	<b>112</b>
20.1 Постановка задачи . . . . .	112
20.2 Метрики качества . . . . .	112
20.2.1 Внутрекластерное расстояние . . . . .	112
20.2.2 Межкластерное расстояние . . . . .	112
20.2.3 Индекс Данна . . . . .	112
20.3 Как считать расстояние? . . . . .	112
20.3.1 Евклидово расстояние . . . . .	112
20.3.2 Манхэттенское расстояние . . . . .	113
20.4 Алгоритмы кластеризации . . . . .	113
20.4.1 K-means . . . . .	113
20.4.1.1 Алгоритм . . . . .	113
20.4.1.2 Идея метода. . . . .	113
20.4.2 Графовые методы кластеризации . . . . .	113
20.4.3 Иерархическая кластеризация . . . . .	113
20.4.3.1 Алгоритм Ланса-Уильямса . . . . .	113
20.4.4 Density-Based Clustering . . . . .	114
20.4.4.1 Алгоритм DBSCAN . . . . .	114
20.5 Метрики качества . . . . .	114
20.5.1 Rand Index (RI) . . . . .	114
20.5.2 Adjusted Rand Index (ARI) . . . . .	114
20.5.3 Mutual Information (AMI) . . . . .	115
20.5.4 Гомогенность, Полнота, V-мера . . . . .	115
20.5.5 Силуэт (Silhouette) . . . . .	115
20.6 Визуализация . . . . .	116
20.6.1 MultiDimensional Scaling (MDS) . . . . .	116
20.6.2 t-SNE(T-distributed stochastic neighbour embedding) . . . . .	116
20.6.2.1 Обучение t-SNE . . . . .	116
<b>21 Кластеризация. Обучение без учителя. Конспект Соколова</b>	<b>117</b>
21.1 Кластеризация . . . . .	117
21.1.1 Метрики качества кластеризации . . . . .	117
21.1.2 K-Means . . . . .	118
21.1.3 Графовые методы . . . . .	118
21.1.4 Иерархическая кластеризация . . . . .	118
21.2 Визуализация . . . . .	118
21.3 Обучение представлений . . . . .	119

<b>22 Перевод статьи DBSCAN Clustering — Explained</b>	<b>121</b>
22.1 Кластеризация на основе плотности . . . . .	121
22.2 Алгоритм DBSCAN . . . . .	122
22.3 Scikit-learn реализация . . . . .	122
22.3.1 Импортирование библиотек . . . . .	122
22.3.2 Генерация датасета . . . . .	122
22.3.3 Визуализация полученного датасета . . . . .	122
22.3.4 Использование DBSCAN . . . . .	123
22.3.5 Визуализация результатов работы DBSCAN . . . . .	123
22.4 Плюсы и минусы DBSCAN . . . . .	123
<b>23 HDBSCAN - перевод статьи</b>	<b>124</b>
<b>24 Лекция 11. Кластеризация. Наша лекция</b>	<b>125</b>
24.1 Практика . . . . .	125
24.1.1 Генерация датасета . . . . .	125
24.1.2 Применение K-Means . . . . .	125
24.1.3 Визуализация предсказания K-Means . . . . .	125
24.1.4 Применение иерархической кластеризации . . . . .	125
24.1.5 Что будет если подобрать неверное количество кластеров? . . . . .	125
24.1.5.1 K-Means . . . . .	125
24.1.5.2 DBSCAN . . . . .	126
24.2 Пример: кластеризация игроков NBA . . . . .	126
24.2.1 Получаем данные . . . . .	126
24.2.2 Применим K-Means с 5-ю кластерами только к числовым столбцам . . . . .	127
24.2.3 Визуализируем данные с помощью метода главных компонент (PCA) . . . . .	127
24.2.4 Посмотрим, какое смысловое значение несут кластеры . . . . .	127
24.3 Сжатие изображений с помощью K-Means . . . . .	127
24.3.1 Возьмём изображение . . . . .	127
24.3.2 Сжимаем изображение . . . . .	127
24.4 Нахождение тем в текстах . . . . .	128
24.4.1 Загрузка данных . . . . .	128
24.4.2 TF-IDF . . . . .	128
24.4.3 Применим K-Means к получившимся векторам и выведем метрики качества кластеризации . . . . .	128
24.4.4 Выведем слова, соответствующие самым весомым компонентам центров кластеров . . . . .	129
24.5 Кластеризация рукописных цифр . . . . .	129
24.5.1 Загрузка данных . . . . .	129
24.5.2 Обучим K-Means с 10-ю кластерами . . . . .	129
24.5.3 Посмотрим на центры кластеров . . . . .	129
24.5.4 Визуализация с помощью PCA . . . . .	129
24.6 HDBSCAN . . . . .	130
24.6.1 Подключение библиотек . . . . .	130
24.6.2 Генерация данных . . . . .	130
24.6.3 Кластеризуем с помощью hdbscan . . . . .	130
24.6.4 Алгоритм вкратце . . . . .	130
24.6.5 Визуализация мин остава для наших данных . . . . .	130
24.6.6 Визуализация дендрограммы для наших данных . . . . .	130
24.6.7 Схлопываем кластеры . . . . .	131
24.6.8 Найдём итоговые кластеры на наших данных . . . . .	131
24.6.9 Сравнение с DBSCAN . . . . .	131
<b>25 Кластеризация. Продолжение. Конспект Соколова</b>	<b>132</b>
25.1 Графовые методы . . . . .	132
25.1.1 От выборки к графу . . . . .	132
25.1.2 Лапласиан графа . . . . .	132
25.1.3 Теорема Лапласиана Графа . . . . .	133
25.1.4 Гипотеза . . . . .	133
25.1.5 Спектральная кластеризация . . . . .	134
25.2 Оценка качества кластеризации . . . . .	134
25.2.1 Метрика на разметке . . . . .	134

25.2.2 BCubed . . . . .	134
<b>26 Перевод статьи про спектральную кластеризацию . . . . .</b>	<b>135</b>
26.1 Введение . . . . .	135
26.2 Собственные векторы и собственные значения . . . . .	135
26.2.1 Реализация поиска собственных значений в Python . . . . .	135
26.3 Графы . . . . .	135
26.4 Матрица смежности . . . . .	136
26.5 Степень матрицы . . . . .	136
26.6 Лапласиан Графа . . . . .	137
26.7 Собственные значения Лапласиана графа . . . . .	137
26.8 Спектральная кластеризация для произвольных данных . . . . .	139
26.9 Граф ближайших соседей . . . . .	140
26.10 Другие подходы . . . . .	141
26.11 Заключение . . . . .	141
<b>27 Лекция 12. Кластеризация-2. Наша лекция . . . . .</b>	<b>142</b>
27.1 Спектральная кластеризация . . . . .	142
27.1.1 Подключение библиотек . . . . .	142
27.1.2 Генерация данных . . . . .	142
27.1.3 Попробуем кластеризовать с помощью K-Means . . . . .	142
27.1.4 Попробуем кластеризовать с помощью DBSCAN . . . . .	142
27.1.5 Попробуем спектральную кластеризацию . . . . .	143
27.2 Сегментация изображений при помощи спектральной кластеризации . . . . .	143
27.3 Optuna . . . . .	144
27.3.1 Пример использования optuna . . . . .	144
27.3.2 Ключевая идея Optuna . . . . .	144
27.3.3 Что под капотом? . . . . .	144
27.3.4 Sampler: Tree-structured Parzen Estimator (TPE) . . . . .	145
27.3.5 Pruner: Median Pruner . . . . .	147
27.3.6 Successive Halving Algorithm (SHA) of Pruning . . . . .	147
27.3.7 Пример запуска . . . . .	147
27.3.8 Ещё один пример запуска . . . . .	148
27.3.9 Как добавить optuna для своей модели? . . . . .	149
<b>28 Лекция 13. Рекомендательные системы. Наша лекция . . . . .</b>	<b>150</b>
28.1 Что такое рекомендательная система? . . . . .	150
28.2 Основные подходы . . . . .	150
28.3 Collaborative Filtering . . . . .	150
28.3.1 User-based CF . . . . .	151
28.3.1.1 Явная и неявная оценка . . . . .	151
28.3.1.2 Корреляция оценок . . . . .	152
28.3.1.3 Прогноз . . . . .	152
28.3.1.4 Минусы . . . . .	152
28.3.2 Item-based CF . . . . .	152
28.3.2.1 Плюсы Item-base CF . . . . .	152
28.3.3 Обзор Collaborative Filtering . . . . .	153
28.4 Матричные разложения . . . . .	153
28.4.1 Векторы интересов . . . . .	153
28.4.2 Рейтинг . . . . .	153
28.4.3 Модели со скрытыми переменными . . . . .	153
28.4.4 Матричные разложения . . . . .	154
28.5 Хорошие свойства рекомендательной системы . . . . .	154
28.5.1 Diversity(Разнообразие) . . . . .	154
28.5.2 Novelty (новизна) . . . . .	155
28.5.3 Serendipity (способность удивить) . . . . .	155
28.6 Метрики . . . . .	155
28.6.1 Метрики качества ранжирования . . . . .	155
28.7 Практика . . . . .	156
28.7.1 Импорт библиотек . . . . .	156
28.7.2 Загрузка данных . . . . .	156

28.7.3 Предобработка данных . . . . .	156
28.7.4 Построим гистограмму взаимодействия . . . . .	157
28.7.5 Сгладим количество взаимодействий . . . . .	157
28.7.6 Разобъём выборку на train и test опираясь на временной фактор . . . . .	157
28.7.7 Изменим формат данных . . . . .	158
28.7.8 Baseline (модель по популяции) . . . . .	158
28.7.9 Коллаборативная фильтрация . . . . .	159
28.7.9.1 Memory-based . . . . .	159
28.7.10 Модель со скрытыми переменными . . . . .	160
<b>29 Рекомендательные системы. Признаки в рекомендательных системах. Конспект Соколова. Первый конспект</b>	<b>162</b>
29.1 Коллаборативная фильтрация . . . . .	162
29.1.1 Memory-based . . . . .	162
29.1.2 Модели со скрытыми переменными . . . . .	163
29.1.3 Учёт неявной информации . . . . .	164
29.1.4 Факторизационные машины . . . . .	165
29.1.5 FFM . . . . .	165
29.2 Контентные модели . . . . .	166
29.3 Статистические признаки . . . . .	166
<b>30 Рекомендательные системы. Метрики качества рекомендаций. Конспект Соколова. Второй конспект</b>	<b>167</b>
30.1 Качество предсказаний . . . . .	167
30.1.1 Предсказание рейтингов . . . . .	167
30.1.2 Предсказание событий . . . . .	167
30.1.3 Качество ранжирования . . . . .	167
30.1.3.1 Недостатки оценок качества предсказания . . . . .	168
30.2 Покрытие . . . . .	168
30.2.1 Покрытие товаров . . . . .	168
30.2.2 Покрытие пользователей . . . . .	168
30.3 Новизна . . . . .	168
30.4 Прозорливость (serendipity) . . . . .	168
30.5 Разнообразие . . . . .	169
30.6 Архитектура рекомендательных систем . . . . .	169
<b>31 Статья про FM и модификации. Перевод</b>	<b>170</b>
31.1 Введение . . . . .	170
31.2 Рекомендательные системы . . . . .	170
31.3 Матричное разложение . . . . .	170
31.4 Машины факторизации . . . . .	171
31.5 Learning-to-Rank . . . . .	172
31.6 Оценка модели . . . . .	173
31.7 Сравнение с бейзлайном . . . . .	175
31.8 Заключение . . . . .	176
<b>32 Лекция 14. Факторизационные машины. Наша лекция</b>	<b>178</b>
32.1 Контентные модели . . . . .	178
32.2 Факторизационные машины . . . . .	178
32.3 Метрики . . . . .	178
32.3.1 Precision at K . . . . .	178
32.3.2 Average precision at K . . . . .	179
32.3.3 Mean average precision at K . . . . .	179
32.3.4 Cumulative Gain at K . . . . .	179
32.4 Практика . . . . .	180
32.4.1 Загрузка библиотек . . . . .	180
32.4.2 Загрузка и обработка данных . . . . .	180
32.4.3 Контентные модели . . . . .	182
32.4.4 Факторизационная машина . . . . .	184

<b>33 Статья. Дисбаланс Классов</b>	<b>186</b>
33.1 Что понимается под дисбалансом классов? . . . . .	186
33.2 Что делать при дисбалансе классов? . . . . .	186
33.3 Перебалансировка данных/изменение выборки . . . . .	187
33.4 Nearnmiss1/2, Tomek links, Edited nearest neighbors (ENN) . . . . .	188
33.5 SMOTE = Synthetic Minority Oversampling Techniques, ADASYN = Adaptive Synthetic . . . . .	189
33.6 Взвешивание объектов . . . . .	190
33.7 Решающее правило: выбор порога . . . . .	190
33.8 Что использовать на практике . . . . .	191
33.9 Какие ещё методы существуют? . . . . .	192
<b>34 Одноклассовые методы и обнаружение аномалий. Конспект Соколова</b>	<b>193</b>
34.1 Несбалансированная классификация . . . . .	193
34.2 Одноклассовая классификация . . . . .	193
34.2.1 Статистические методы . . . . .	193
34.2.1.1 Непараметрический подход . . . . .	193
34.2.1.2 Параметрический подход . . . . .	194
34.2.2 Метрические методы . . . . .	194
34.2.3 Одноклассовый метод опорных векторов . . . . .	194
34.2.4 Isolation forest . . . . .	195
<b>35 Лекция 15. Дисбаланс классов и поиск аномалий. Наша лекция</b>	<b>197</b>
35.1 Практическая работа с балансировкой данных . . . . .	197
35.1.1 Импортируем библиотеки . . . . .	197
35.1.2 Загружаем датасет . . . . .	197
35.1.3 Предобрабатываем данные . . . . .	197
35.1.4 Разбиение на train и test . . . . .	198
35.1.5 Обучим логрег . . . . .	198
35.1.6 Попробуем обучить бустинг . . . . .	198
35.1.7 Посмотрим сколько у нас объектов каждого класса . . . . .	198
35.1.8 Случайный undersampling . . . . .	198
35.1.9 Случайный oversampling . . . . .	199
35.1.10 undersampling с библиотекой imblearn . . . . .	199
35.1.11 oversampling с библиотекой imblearn . . . . .	199
35.1.12 undersampling Tomek links . . . . .	200
35.1.13 Синтетический oversampling меньшего класса (SMOTE) . . . . .	200
35.1.14 NearMiss . . . . .	200
35.1.15 Penalize algorithm (cost-sensitive training) . . . . .	200
35.1.16 Xgboost с лучшей техникой семплирования . . . . .	201
35.2 Поиск аномалий . . . . .	201
35.2.1 Введение . . . . .	201
35.2.2 Простые методы . . . . .	202
35.2.2.1 Box plot . . . . .	202
35.2.2.2 Z-score . . . . .	204
35.2.3 Elliptic Envelope . . . . .	204
35.2.4 Одноклассовый SVM . . . . .	207
35.2.5 Изолирующий лес (Isolation Forest) . . . . .	208
35.2.6 Разделяющие поверхности для разных алгоритмов . . . . .	210
35.2.6.1 OneClassSVM . . . . .	211
35.2.6.2 Isolation Forest . . . . .	212
35.2.6.3 Elliptic Envelope . . . . .	213
35.2.7 Библиотека PyOD . . . . .	213
35.2.7.1 Импортируем библиотеки . . . . .	213
35.2.7.2 Сгенерируем набор точек для задачи поиска выбросов . . . . .	214
35.2.7.3 Применим KNN для нахождения выбросов . . . . .	214
35.2.7.4 Посмотрим на качество алгоритма . . . . .	215
35.2.7.5 Визуализируем результат . . . . .	215
35.2.7.6 Сравним различные алгоритмы для нахождения выбросов по времени работы и качеству . . . . .	215
35.2.7.7 Попробуем определить долю выбросов в датасете . . . . .	217



# Часть I

## Модуль 1

### 1. Лекция 1. Линейный методы регрессии

Базовые вещи знаем из курса математики - как обучать с помощью градиентного спуска и как решать аналитически.

#### 1.1. Напоминание о том, что такое линейная регрессия

Предположим, что мы хотим предсказать стоимость одома  $y$  по его площади ( $x_1$ ) и количеству комнат ( $x_2$ ).

Линейная модель для предсказания стоимости:  $\alpha(x) = w_0 + w_1x_1 + w_2x_2$ ,

где  $w_0, w_1, w_2$  - параметры модели (веса)

Линейная регрессия означает то, что все веса линейны от признаков (сами признаки могут быть любыми).

#### 1.2. Общий вид линейной регрессии

$$\alpha(x) = w_0 + w_1x_1 + \dots + w_nx_n$$

Сокращённая запись:

$$\alpha(x) = w_0 + \sum_{i=1}^n w_i x_i$$

Запись через скалярное произведение (с добавлением признака  $x_0 = 1$ ):

$$\alpha(x) = (w, x)$$

#### 1.3. Как мы обучаем модель

Мы просто пытаемся минимизировать ошибку предсказаний:

$$Q(\alpha, X) = \frac{1}{l} \sum_{i=1}^l (\alpha(x_i) - y_i)^2 = \frac{1}{l} \sum_{i=1}^l ((w, x_i) - y_i)^2 \rightarrow \min_w$$

где  $l$  - кол-во объектов

#### 1.4. Проблемы возникающие при обучении на данных

Допустим к предсказанию стоимость квартиры мы захотели добавить два признака:  $x_3$  - район, в котором находится квартира и  $x_4$  - удалённость от МКАД.

- Что такое район? Это категориальный признак и непонятно как на нём обучать.
- Также, когда мы добавляем признак, мы предполагаем, что он как-то монотонно влияет на ответ. А вот с удалённостью от МКАД так нельзя сказать. Есть районы, которые лежат вне МКАД, но при этом там квартиры дороже, чем внутри.

#### 1.5. One-hot encoding - решение проблемы категориальных признаков

One-hot encoding (OHE) - мы создаём новые числовые столбцы, каждого из которых является индикатором одного из районов.

##### 1.5.1. Проблемы ОНЕ при обучении

Но при ОНЕ мы можем столкнуться с проблемой, что мы один из столбцов ОНЕ можем выразить через остальные, как  $1$  минус сумма остальных. Столбцы линейно зависимы. Это плохо для линейных моделей:

- При аналитическом решении у нас сломается формула

- При градиентном спуске, если  $x_1, \dots, x_l$  линейно зависимы, то  $\exists v : (v, x) = 0$ , а это означает, что мы можем добавить сколько угодно добавить  $(w + \alpha v, x)$  и предсказание не поменяется, а вектор весов получается неоднозначный и получается много решений у задачи. При минимизации функции потерь могут получиться большие веса, а большие веса - переобучение.

В качестве решения мы можем выкинуть одну из колонок ОНЕ (в sklearn за это отвечает параметр `drop_first`). Мы можем так делать, если знаем, что новый район не добавится. Если районы будут добавляться - мы можем не выкидывать, потому что линейная зависимость пропадает.

## 1.6. Решение проблемы немонотонных признаков

Мы можем разбить признак удалённости от МКАД на отрезки и сделать что-то типа ОНЕ: сделать признаки, что квартира находится в  $[0; 10]$  км от МКАД, в  $[10; 30]$  км от МКАД. Получаются бинарные признаки

### 1.6.1. Как разбивать эти переменные?

Можно разбивать по квантилям - универсальное решение, чтобы не думать.

## 1.7. Метрики

Функцию потерь мы минимизируем. Метрика - другая функция, которую мы считаем, чтобы понять - насколько модель хорошая.

### 1.7.1. Среднеквадратичное отклонение (MSE)

Mean Squared Error - MSE

$$MSE(\alpha, X) = \frac{1}{l} \sum_{i=1}^l (\alpha(x_i) - y_i)^2$$

где  $l$  - количество объектов. Почему чаще всего используют MSE для обучения линейной регрессии? Минимизация этой ошибки - это максимизация правдоподобия, в случае если мы предполагаем, что наши данные имеют нормальное распределение, а это часто так.

Какие плюсы:

- Позволяет сравнивать модели между собой
- Подходит для контроля качества во время обучения

С какими проблемами мы столкнёмся?

- Выбросы
- Плохая интерпретируемость
- Неочевидно, хорошая ошибка или нет, нужно, например, сравнивать со средним значением целевой переменной, чтобы понять, хорошо мы предсказываем или нет. Неограничена сверху.

### 1.7.2. RMSE

Root Mean Squared Error - RMSE

$$RMSE(\alpha, X) = \sqrt{\frac{1}{l} \sum_{i=1}^l (\alpha(x_i) - y_i)^2}$$

Плюсы:

- Все плюсы MSE
- Сохраняет единицы измерения (в отличие от MSE)

Минусы:

- Тяжело понять, насколько хорошо данная модель решает задачу, так как тоже не ограничена сверху, как и MSE

### 1.7.3. $\mathbf{R}^2$

Коэффициент детерминации -  $R^2$

$$R^2(\alpha, X) = 1 - \frac{\sum_{i=1}^l (\alpha(x_i) - y_i)^2}{\sum_{i=1}^l (y_i - \bar{y})^2}$$

где  $\bar{y} = \frac{1}{l} \sum_{i=1}^l y_i$

Коэффициент детерминации - это доля дисперсии целевой переменной, объясняемая моделью.

- Чем ближе  $R^2$  к 1, тем лучше модель объясняет данные
- Чем ближе  $R^2$  к 0, тем ближе модель к константному предсказанию
- Отрицательный  $R^2$  говорит о том, что модель плохо решает задачу, и даже хуже, чем константное предсказание.

### 1.7.4. MAE

Mean Absolute Error - MAE

$$MAE(\alpha, X) = \frac{1}{l} \sum_{i=1}^l |\alpha(x_i) - y_i|$$

### 1.7.5. MAPE

Mean Absolute Percentage Error - MAPE

$$MAPE(\alpha, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - \alpha(x_i)|}{|y_i|}$$

MAPE измеряет относительную ошибку

Плюсы:

- Ограничена:  $0 \leq MAPE \leq 1$
- Хорошо интерпретируема: например, MAPE = 0.16 значит, что ошибка модели в среднем составляет 16% от фактических значений

Минусы:

- По разному относится к недо- и перепрогнозу. Например, если правильный ответ  $y = 10$ , а прогноз  $\alpha(x) = 20$ , то ошибка  $\frac{|10-20|}{10} = 1$ , а если ответ  $y = 30$ , то ошибка  $\frac{|30-20|}{30} = \frac{1}{3} = 0.33$

### 1.7.6. SMAPE

Symmetric Mean Absolute Percentage Error - SMAPE. Симметричный вариант MAPE:

$$SMAPE(\alpha, X) = \frac{1}{l} \sum_{i=1}^l \frac{|y_i - \alpha(x_i)|}{(|y_i| + |\alpha(x_i)|)/2}$$

SMAPE - попытка сделать симметричным прогноз - то есть дать одинаковую ошибку для недо- и перепрогноза

Проверим: пусть правильный ответ  $y = 10$ , а прогноз  $\alpha(x) = 20$ , то ошибка  $= \frac{|10-20|}{|10+20|/2} = \frac{2}{3} = 0.67$ , а если ответ  $y = 30$ , то ошибка  $\frac{|30-20|}{|30+20|/2} = \frac{2}{5} = 0.4$

Сейчас уже в среде прогнозистов сложилось более-менее устойчивое понимание, что SMAPE не является хорошей ошибкой. Тут дело не только в завышении прогнозов, но и в том, что наличие прогноза в знаменателе позволяет манипулировать результатами оценки.

### 1.7.7. MSLE

Mean Squared Logarithmic Error - MSLE. Среднеквадратическая логарифмическая ошибка

$$MSLE(\alpha, X) = \frac{1}{l} \sum_{i=1}^l (\log(\alpha(x_i) + 1) - \log(y + 1))^2$$

Особенности:

- Подходит для задач с неотрицательной целевой переменной ( $y \geq 0$ )
- Штрафует за отклонения в порядке величин
- Штрафует заниженные прогнозы сильнее, чем завышенные

### 1.7.8. Квантильная регрессия

Квантильная функция потерь:

$$Q(\alpha, X^l) = \sum_{i=1}^l \rho_\tau(y_i - \alpha(x_i))$$

где  $\rho_\tau(z) = (\tau - 1)[z < 0]z + \tau[z \geq 0]z = (\tau - \frac{1}{2}) + \frac{1}{2}|z|$

Параметр  $\tau \in [0; 1]$ . Чем больше  $\tau$ , тем больше штрафуем за занижение прогноза.

Теорема: Пусть в каждой точке  $x \in X$  (пространство объектов) задано распределение  $p(y|x)$  на ответах для данного объекта. Тогда оптимизация функции потерь  $\rho_\tau(z)$  даёт алгоритм  $\alpha(x)$ , приближающий  $\tau$ -квантиль распределения ответов в каждой точке  $x \in X$

Иными словами: допустим мы предсказываем стоимость квартиры. Если мы используем MSE, то мы получим  $\bar{y}$ , если по признакам все параметры совпадают у разных объектов, но целевая переменная однаковая. К примеру  $\{10, 20, 90\}$  выдаст  $\{40\}$ . Мы можем попросить завысить прогноз, поставить квантиль.

- Если мы хотим завысить прогноз, то берём  $\tau$  ближе к единице.
- Если занизить - берём  $\tau$  ближе к нулю.

## 1.8. Небольшое дополнение по линейной регрессии в sklearn

В sklearn класс LinearRegression всегда использует MSE. В то время, как в SGDRegressor, эта же самая линейная регрессия, но мы можем подставить нужную функцию потерь.

## 2. Лекция 2 (Анастасия)

### 2.0.1. Дополнительные ссылки для проверки

Будут позже

### 2.1. Работа с кодом

#### 2.1.1. Скачивание и обработка данных

```
import pandas as pd
X_raw = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-database/autos/import...
print(X_raw.head())

print("nan_is_ ", X_raw[25].isna().sum())
X_raw = X_raw[X_raw.notna()]
y = X_raw[25]
X_raw = X_raw.drop(25, axis=1)
print(X_raw.shape, len(y))

from sklearn import impute

% get cat features
cat_feature_mask = (X_raw.dtypes == "object").values
```

```

X_real = X_raw[X_raw.columns[~cat_feature_mask]]
mis_replacer = impute.SimpleImputer(strategy="mean")
X_no_miss_real = pd.DataFrame(data=mis_replacer.fit_transform(X_real), columns=X_real.columns)

X_cat = X_raw[X_raw.columns[cat_feature_mask]].fillna("")
X_cat = X_cat.reset_index(drop=True)

X_no_miss = pc.concat([X_no_miss_real, X_cat], axis=1)

X_dum = pd.get_dummies(X_no_miss, drop_first=True)
print(X_dum.shape)
print(X_dum.head())

from sklearn import preprocessing

normalizer = preprocessing.MinMaxScaler()
X_real_norm_np = normalizer.fit_transform(X_dum)
X = pd.DataFrame(data=X_real_norm_np)

```

### 2.1.2. Обучение модели и оценка качества

```

from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25, random_state=123)

from sklear.linear_model import LinearRegression

model = LinearRegression()
model.fit(Xtrain, ytrain)

pred_mse = model.predict(Xtest)

from sklearn.metrics import r2_score

print(r2_score(y_test, pred_mse))
print(r2_score(ytrain, model.predict(Xtrain)))

from sklearn import SGDRegressor

lr_mse = SGDRegressor(loss='squared_loss')
lr_mse.fit(Xtrain, ytrain)

pred_mse = lr_mse.predict(Xtest)

print(r2_score(ytest, pred_mse))

lr_mae = SGDRegressor(loss='squared_loss')
lr_mae.fit(Xtrain, ytrain)

pred_mae = lr_mae.predict(Xtest)

print(r2_score(ytest, pred_mae))

```

## 2.2. Классическая предобработка данных для линейной регрессии

- pairplot by seaborn

- бинаризировать данные
- Ordinal Encoding
- target encoding - может быть переобучение
- category\_encoders

### 2.3. Счётчики

#### Counts

$$counts(u, K) = \sum_{(x,y) \in X} [f(x) = u]$$

#### Successes

$$successes_k(u, X) = \sum_{(x,y) \in X} [f(x) = u][y = k]$$

**Замена категориального счётчика на вещественный**

$$g_k(x, X) = \frac{successes_k(f(x), X) + c_k}{counts(f(x), X) + \sum_{m=1}^K c_m}$$

### 3. Лекция 3. Логистическая регрессия. Лекция с ФЭН 2021

#### 3.1. Модель бинарной классификации

Мы можем всё ещё использовать нашу линейную регрессию таким образом:

$$\alpha(x, w) = \text{sign}\left(\sum_{j=1}^l w_j x_j\right)$$

где

- Если  $\sum_{j=1}^l w_j x_j > 0$ , то  $\text{sign}(\sum_{j=1}^l w_j x_j) = +1$ , то есть объект отнесён к положительному классу
- Если  $\sum_{j=1}^l w_j x_j < 0$ , то  $\text{sign}(\sum_{j=1}^l w_j x_j) = -1$ , то есть объект отнесён к отрицательному классу
- Значит  $\sum_{j=1}^l w_j x_j = 0$  - уравнение разделяющей границы между классами. Это уравнение плоскости (или прямой в двумерном случае), поэтому классификатор является линейным

В данном случае мы будем минимизировать такой функционал ошибки:

$$Q(\alpha, X) = \frac{1}{n} \sum_{i=1}^n [\alpha(x_i) \neq y_i] \rightarrow \min$$

##### 3.1.1.

Отступ

Давайте обозначим  $M_i = y_i \cdot (w, x_i)$  - отступ на  $i$ -м объекте. В терминах отступа мы можем переопределить нашу задачу:

Какой должен быть знак у отступа ( $\text{sign}(M_i)$ ), если  $\alpha(x_i) = y_i$ ?

На самом деле знак положительный. Нужно рассмотреть  $\{-, +\}^2$  и понятно откуда это получается.

Тогда задача минимизации ошибки будет  $\frac{1}{n} \sum_{i=1}^n [M_i < 0] \rightarrow \min$

Кроме этого величина отступа  $M$  обозначает степень уверенности классификатора в ответе - чем ближе  $M$  к нулю, тем меньше уверенность в ответе.

Сложность с минимизацией этой функции - то что от неё нельзя взять производную - эта функция разрывна. Давайте заменим её на другую функцию потерь, которая будет решать примерно ту же самую задачу.

Пусть у нас была эта функция  $L(\alpha, y) = L(M) = [M < 0]$  - разрывная функция потерь

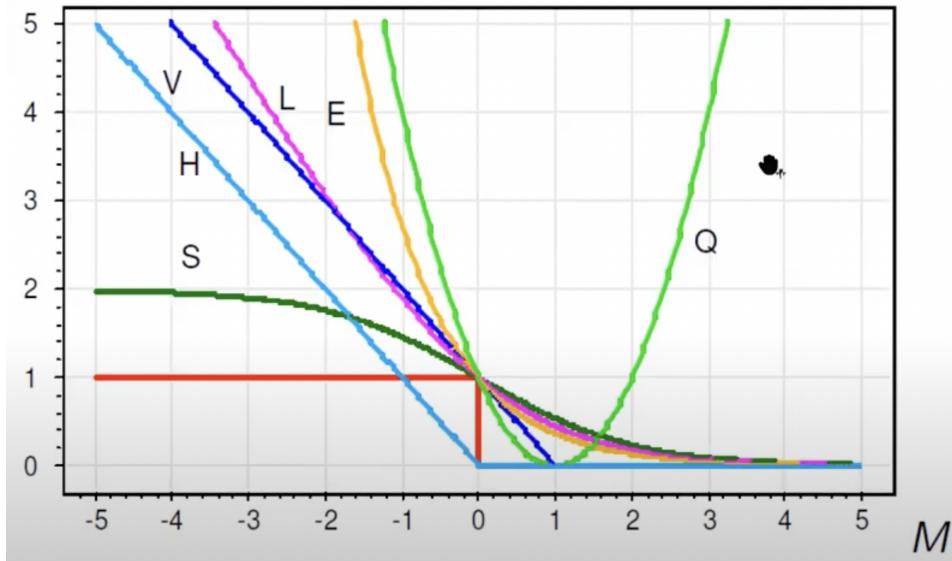
А теперь мы найдём какую-то  $\tilde{L}$ , такую, что  $L(M) \leq \tilde{L}(M)$ , где  $\tilde{L}(M)$  - непрерывная или гладкая функция потерь, тогда

$$Q(\alpha, X) = \frac{1}{n} \sum_{i=1}^n L(y_i \cdot (w, x_i)) \leq \frac{1}{n} \sum_{i=1}^n \tilde{L}(y_i \cdot (w, x_i)) \rightarrow \min$$

##### 3.1.2. Функции потерь

Минимизируя различные функции потерь, получаем разные результаты. Поэтому разные функции потерь определяют различные классификаторы.

- $L(M) = \log(1 + e^{-M})$  - логистическая функция потерь
- $V(M) = (1 - M)_+ = \max(0, 1 - M)$  - кусочно-линейная функция потерь (метод опорных векторов PCA)
- $H(M) = (-M)_+ = \max(0, -M)$  - кусочно-линейная функция потерь (персентрон)
- $E(M) = e^{-M}$  - экспоненциальная функция потерь
- $S(M) = \frac{2}{1+e^{-M}}$  - сигмоидная функция потерь
- $[M < 0]$  - пороговая функция потерь



### 3.2. Функция потерь

Очевидно, что для градиентного спуска теперь мы используем стандартный спуск по формуле

### 3.3. Предсказание для логистической регрессии

Формула предсказания для логистической регрессии выглядит как  $\sigma(w^T x)$ , где  $\sigma(z) = \frac{1}{1+e^{-z}} \in (0; 1)$

$$w^{(k)} = w^{(k-1)} - \eta \cdot \nabla Q(w^{(k-1)})$$

Где мы предсказываем  $y = +1$ , если  $\alpha(x, w) \geq 0.5$  Где  $\alpha(x, w) = \sigma(w^T x) \geq 0.5$ , если  $w^T x \geq 0$

Получаем, что

- $y = +1$  при  $w^T x > 0$
- $y = -1$  при  $w^T x < 0$

То есть,  $w^T x = 0$  - разделяющая гиперплоскость

### 3.4. Функция потерь

Если мы будем использовать MSE, то для совсем неправильного класса (вероятность нужного = 0) у нас будет ошибка  $(1 - 0)^2 = 1$ , это очень мало.

Давайте возьмём логистическую функцию потерь:

$$Q(w) = - \sum_{i=1}^l ([y_i = +1] \cdot \log(\alpha(x_i, w)) + [y_i = -1] \cdot \log(1 - \alpha(x_i, w)))$$

В этом случае:

- если  $\alpha(x, w) = 1$  и  $y = +1$ , то штраф  $L(\alpha, y) = 0$
- если  $\alpha(x, w) \rightarrow 0$  и  $y = +1$ , то штраф  $L(\alpha, y) \rightarrow +\infty$

### 3.5. Вероятностная постановка задачи

У нас могут быть разные объекты с одинаковым описанием (просто самого описания может быть недостаточно): на примере задачи "вернёт ли клиент кредит возраст у людей может быть одинаковый, задача одинаковая, профессия одна и та же, но мы можем не знать о жизненной ситуации людей. Поэтому объекты с одним признаком описаниям могут иметь разные значения целевой переменной.

Цель: построить алгоритм  $b(x)$  в каждой точке  $x$  предсказывающий  $p(y = +1|x)$ .

Пусть объект  $x$  встречается в выборке  $n$  раз с ответами  $\{y_1, \dots, y_n\}$ . Хотим, чтобы алгоритм выдавал вероятность положительного класса:

$$b_*(x) = \arg \min_{b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n L() y_i, b \approx p(y = +1|x)$$

### Сверху бага

По закону больших чисел при  $n \rightarrow \infty$  получаем

$$b_*(x) = \arg \min_{b \in \mathbb{R}} E[L(y, b)x]$$

Отсюда получаем условие на функцию потерь

$$\arg \min E[L(y, b)|x] = p(y = +1|x)$$

LogLoss подходит, а вот MAE не подходит.

## 3.6. Правдоподобие и Log-Loss

Вероятности, которые выдаёт алгоритм  $b(x)$ , должны согласовываться с выборкой

Вероятность того, что в выборке встретится объект  $x$  с классом  $y$ :

$$b(x)^{[y=+1]} \cdot (1 - b(x))^{[y=-1]}$$

Правдоподобие выборки:

$$(b, X) = \prod_{i=1}^l b(x_i)^{[y_i=+1]} \cdot (1 - b(x_i))^{[y_i=-1]}$$

Для нахождения оптимальных параметров алгоритма можно воспользоваться методом максимума правдоподобия (ММП):

$$(b, x) = \prod_{i=1}^l b(x_i)^{[y_i=+1]} \cdot (1 - b(x_i))^{[y_i=-1]} \rightarrow \max_b$$

Прологарифмировав и поставив минус в начале - получим аналогичную задачу, да ещё и ровно Log-Loss:

$$-\sum_{i=1}^l ([y_i = +1] \log b(x_i) + [y_i = -1] \log(1 - b(x_i))) \rightarrow \min_b$$

## 3.7. Выбор алгоритма предсказания

Хотим взять алгоритм  $b(x)$  такой, чтобы он возвращал числа из отрезка  $[0; 1]$

Можно взять  $b(x) = \sigma(w^T x)$ , где  $\sigma$  - любая монотонно неубывающая функция с областью значений  $[0; 1]$

Возьмём сигмоиду  $\sigma(z) = \frac{1}{1+e^{-z}}$

## 3.8. Смысл ( $w$ , $x$ ) в логистической регрессии

Логистическая регрессия в каждой точке  $x$  предсказывает вероятность того, что  $x$  принадлежит к положительному классу  $p(y = +1|x)$ . То есть  $p(y = +1|x) = \frac{1}{1+e^{-w^T x}}$ . Отсюда можно выразить  $(w, x) = w^T x$ :

$$(w, x) = w^T x = \log \left( \frac{p(y = +1|x)}{p(y = -1|x)} \right)$$

И эта величина логарифма отношения называется логарифм отношения шансов (log odds). Из формулы видно: что величина может принимать любое значение.

## 3.9. Логарифмическая функция потерь

Тогда функция потерь может быть записана в куда более красивом виде:

$$L(b, x) = \sum_{i=1}^l \log(1 + e^{-y_i(w, x)})$$

## 3.10. Бонус. Алгоритм Персептрана Розенблата

Персептрон - это простейшая модель классификации, при этом являющаяся предшественником нейронных сетей. Он решает задачу классификации с двумя классами и бинарными признаками (каждый признак равен либо 0, либо 1).

Сам алгоритм выглядит как:

$$\alpha(x, w) = [w_1 x_1 + \dots + w_n x_n > 0] = [(w, x) > 0]$$

## 4. Лекция 3. Логистическая регрессия. Наша лекция

Логистическая регрессия решает задачу классификации с помощью регрессии.

Строится линейная модель. Она умеет выдавать вероятность определения какому либо классу и отлично это делает, если классы хорошо линейно разделяются.

У моделей в питоне есть метод `predict_proba`, который выдаёт вероятность попадания в тот или иной класс.

Как получается эта вероятность? В общем случае - это степень уверенности алгоритма с определённым классом. В случае с Логистической регрессией мы минимизируем `log_loss`, то есть константное предсказание самое лучшее будет  $\log\_loss(a, y) = -[y \log(a) + (1 - y) \log(1 - a)]$

### 4.1. Кодовая реализация

#### 4.1.1. Загрузка данных

```
import random
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
%matplotlib inline

df = pd.read_csv("BloodPressure.csv")

print(df.head())

X = df[['SBP', 'DBP']]
y = df['target'].values
```

#### 4.1.2. Функция для предсказания сигмоиды и функции потерь Log-Loss

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    epsilon = 1e-5
    cost = (1/m)*(((-y).T @ np.log(h + epsilon)) - ((1-y).T @ np.log(1-h + epsilon)))
```

#### 4.1.3. Градиент функции потерь

```
def stochastic_gradient_descent(X, y, params, learning_rate, iterations):
    X = np.hstack((np.ones((X.shape[0], 1)), X))
    params = np.random.rand(X.shape[1])

    cost_track = np.zeros((iterations, 1))

    for i in range(iterations):
        ind = random.sample(range(X.shape[0]), 1)

        params = params - learning_rate * (X[ind] * (sigmoid(X[ind] @ params) - y[ind]))
        params = params.ravel()
        cost_track[i] = compute_cost(X, y, params)

    return cost_track, params
```

#### 4.1.4. Функция предсказания

```
def predict(X, params):  
    x = np.hstack((np.ones((X.shape[0], 1)), X))  
    return np.round(sigmoid(X @ params))
```

### 4.2. Различие гипер-параметров и параметров модели

- Гипер-параметры - это те параметры, которые объявляются ещё до обучения модели (в kNN - k будет гиперпараметром)
- Параметры модели - те параметры, которые получаются в процессе обучения

Тогда мы понимаем, что у kNN нет вообще параметров - только гиперпараметры. Это одна из аргументаций почему kNN слабее обычных моделей - она не умеет подстраиваться под данные.

**Написать конспект со сравнением kNN и LogReg**

## 5. Лекция 4. SVM, классификаторы, метрики качества классификации. Лекция с ФЭН

### 5.1. Метод опорных векторов

#### 5.1.1. Линейно разделимая выборка

Выборка линейно разделима, если существует такой вектор параметров  $w^*$ , что соответствующий классификатор  $\alpha(x)$  не допускает ошибок на этой выборке.

Цель метода опорных векторов (Support Vector Machine): максимизировать ширину разделяющей полосы.

Наш алгоритм предсказания  $\alpha(x) = \text{sign}((w, x))$

Дальше нам нужно отнормировать параметры  $w$  так, чтобы

$$\min_{x \in X} |(w, x)| = 1$$

Тогда мы можем посчитать расстояние от точки  $x_0$  до разделяющей гиперплоскости, задаваемой классификатором

$$\rho(x_0, \alpha) = \frac{|(w, x_0)|}{\|w\|}$$

Минимальное расстояние до разделяющей прямой будет  $\frac{1}{\|w\|}$ . Ширина разделяющей полосы получается  $\frac{2}{\|w\|}$ . Мы хотим найти такой вектор параметров, чтобы максимизировать эту полосу.

Если мы перевернём дробь, то у нас как раз будет задача минимизации  $\frac{\|w\|}{2}$ , и для удобства взятия производных возьмём и возведём в квадрат  $\frac{1}{2}\|w\|^2 \rightarrow \min_w$ :

#### 5.1.1.1. Постановка задачи нахождения гиперплоскости

$$\begin{cases} \frac{1}{2}\|w\|^2 \rightarrow \min_w \\ y_i \cdot (w, x_i) \geq 1, i = 1, \dots, l \end{cases}$$

### 5.2. Линейно неразделимая выборка

Так как выборка линейно не разделима, существует хотя бы один объект  $x \in X$  такой, что  $y_i \cdot (w, x_i) < 1$ .

Для того, чтобы находить эту гиперплоскость в линейно неразделяемой выборке, давайте штрафовать объекты штрафом  $\xi_i \geq 0$  за нахождение внутри разделяющей гиперплоскости:

$$y_i \cdot (w, x_i) \geq 1 - \xi_i, i = 1, \dots, l$$

#### 5.2.1. Постановка задачи нахождения гиперплоскости

Хотим:

- Минимизировать штрафы  $\sum_{i=1}^l \xi_i$
- Максимизировать отступ  $\frac{1}{\|w\|}$

Несмотря на то, что эти задачи противоположные по смыслу, мы можем перевернуть дробь в максимизации и начать её минимизировать

Тогда задача оптимизации будет выглядеть как

$$\begin{cases} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^l \xi_i \rightarrow \min_{w, w_0, \xi_i} \\ y_i \cdot (w, x_i) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Где  $C$  - константа регуляризации:

- Чем больше  $C$  - тем больше обращаем внимание на штрафы

- Чем меньше  $C$  - тем меньше обращаем внимание на штрафы

Задача является выпуклой и имеет единственное решение.

Мы можем переписать второе и третье условие в более красивый вид, потому что  $y_i \cdot (w, x_i) \geq 1 - \xi_i \rightarrow \xi_i \geq 1 - y_i \cdot (w, x_i) = 1 - M_i$ , тогда  $\xi_i = \max(0, 1 - M_i)$ , и тогда задача принимает вид

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^l \max(0, 1 - M_i) \rightarrow \min_w$$

Если мы поделим на  $C$ , то сама задача не изменится.

Тогда на задачу оптимизации SVM можно смотреть, как на оптимизацию функции потерь  $L(M) = \max(0, 1 - M) = (1 - M)_+$  с регуляризацией

$$Q(\alpha, X) = \sum_{i=1}^l (1 - M_i)_+ + \frac{1}{2C} \|w\|^2 \min_w$$

Где  $(1 - M_i)_+ = \max(0, 1 - M_i)$

### 5.3. Сравнение с логистической регрессией

На первый взгляд может показаться, что SVM крайне похож на LogReg, но давайте подумаем об этих пунктах:

- LogReg даёт нам вероятность попадения в класс - уверенность алгоритма в ответе, SVM ничего такого не говорит
- В SVM мы просто хотим сделать как можно большую ширину разделяющей полосы (гиперплоскости)

### 5.4. Метрики качества классификации

#### 5.4.1. Accuracy

Accuracy - доля правильных ответов

$$\text{accuracy}(\alpha, X) = \frac{1}{n} \sum_{i=1}^n [\alpha(x_i) = y_i]$$

**5.4.1.1. Недостаток** В случаях сильно несбалансированной выборки не отражает качество работы алгоритма

#### 5.4.2. Матрица ошибок

		Actual Value	
		positives	negatives
Predicted Value	positives	TP True Positive	FN False Negative
	positives	FP False Positive	TN True Negative

#### 5.4.3. Precision (Точность)

$$\text{Precision}(\alpha, X) = \frac{TP}{TP + FP}$$

Показывает, насколько можно доверять классификатору при  $\alpha(x) = +1$

#### 5.4.4. Recall (Полнота)

$$Recall(\alpha, X) = \frac{TP}{TP + FN}$$

Показывает, как много объектов положительного класса находит классификатор

#### 5.4.5. F-мера

F-мера - это метрика качества, учитывающая и точность, и полноту

$$F(\alpha, X) = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

#### 5.4.6. Регулирование точности и полноты

В моделях бинарной классификации с вероятностями попадения в класс - мы можем сами руками изменять в случае какой вероятности в какой класс попадёт объект. Обычно если  $p(x) > 0.5$ , то мы относим объект к одному классу, а  $p(x) < 0.5$  наоборот. Но мы можем сами исправить и поставить другую константу. Точность и полнота при этом изменятся.

При  $p(x) > 0 \rightarrow +1$ :

- Точность будет  $\frac{TP}{n}$ , где  $n$  - количество классов
- Полнота будет равна единичке, потому что мы все объекты предсказали как  $+1$ , и получается у нас нет ни одного FN.

При увеличении порога  $t$ :

- Точность возрастает
- Полнота уменьшается

Хочется, чтобы нам не нужно было выбирать порог, а потом считать качество

#### 5.4.7. ROC-AUC

Для каждого порога  $t$  вычислим

- False Positive Rate:  $FPR = \frac{FP}{FP+TN}$
- True Positive Rate:  $TPR = \frac{TP}{TP+FN}$

Если мы захотим перебрать все возможные пороги, то нам достаточно перебрать  $n + 1$  порог, где  $n$  - количество объектов в выборке.

Давайте построим такой график, где осями будут FPR и TPR. Получится ломанная.

Идеальный алгоритм нам должен выдать прямой угол в вершине TPR 1.

Чтобы посчитать качество - нужно посчитать качество под кривой. Для идеального классификатора площадь будет 1.

#### 5.4.8. Индекс Джини

$$Gini = 2 \cdot AUC - 1$$

Удвоенная площадь между диагональю и ROC кривой.

#### 5.4.9. Precision-Recall кривая

В случае малого количества объектов положительного класса  $AUC - ROC$  может давать неадекватно хороший результат: оси Recall(x) и Precision(y). И называется это PR-кривой

## 6. Лекция 4. SVM. Наша лекция

### 6.1. Кодирование Категориальных признаков

#### 6.1.1. One-Hot Encoding

Предположим, категориальный признак  $f_j(x)$  принимает  $m$  различных значений:  $C_1, C_2, \dots, C_m$

Заменим категориальный признак на  $m$  бинарных признаков  $b_i(x) = [f_j(x) = C_i]$  (индикатор события)

Естественно, нам нужно не забыть выкинуть одну колонку из данных, чтобы между столбцами не было линейной зависимости.

**6.1.1.1. Недостаток** При большом кол-ве значений ( $m$ ) у нас будет большая размерность - много столбцов.

#### 6.1.2. Счётчики

Счётчики (mean target encoding) - это вероятность получить значение целевой переменной для данного значения категориального признака.

Естественно, нам нужно не забыть выкинуть одну колонку из данных, чтобы между столбцами не было линейной зависимости.

**6.1.2.1. Недостаток** Если есть очень редкие категории - то модель может переобучаться.

На примере кредитного скоринга по городам - если из какого-то города было всего два человека и оба не вернули кредит - мы можем переобучиться и на любого человека из этого города говорить, что они кредиты не вернут.

**6.1.2.2. Сглаживание** Для исправления предыдущего недостатка можно использовать сглаживание - учитывать как в среднем по всем категориям у нас признак соотносится:

$$\frac{\text{mean}(\text{target}) \cdot n_{\text{rows}} + \text{global mean} \cdot \alpha}{n_{\text{rows}} + \alpha}$$

где

- $n_{\text{rows}}$  - количество объектов с категорией, у которых целевая переменная равна  $\text{target}$
- $\alpha$  - гипер-параметр регуляризации

**6.1.2.3. Отложенная выборка** Нам придётся выкинуть часть данных, но для того, чтобы не допустить переобучения: Разделим данные на две части, по одной мы подсчитаем счётчики, а во вторую вставим полученные счётчики и будем обучаться на второй части данных.

Модель не смотрела на целевую переменную второй части данных и не переобучивается на этом.

**6.1.2.4. Кросс-валидация** Похоже на отложенную выборку, но мы сможем закодировать все объекты:

1. Разбиваем данные на  $m$  выборок
2. Для каждой выборки  $i$  для подсчёта счётчика используем все выборки, кроме выборки  $i$

#### 6.1.3. Expanding Mean

Отсортируем в определённом порядке датасет и для подсчёта значения на  $m$ -й строке будем использовать предыдущие с 0 по  $m - 1$  строки.

**Дотехать секцию с кодом**

## 7. Лекция 5. Многоклассовая классификация, отбор признаков и методы снижения размерности. Лекция ФЭН

### 7.1. Подходы многоклассовой классификации

#### 7.1.1. One-Vs-All

Решаем задачу классификации на  $k$  классов.

Обучим  $k$  бинарных классификаторов  $b_1(x), \dots, b_k(x)$ , каждый из которых решает задачу: принадлежит ли объект  $x$  классу  $k_i$  или не принадлежит?

Линейный классификатор будет выглядеть как  $b_k(x) = \text{sign}((w_k, x))$

Понятно как принимать решение, но возможна такая ситуация, когда несколько классификаторов сказали, что объект  $x$  принадлежит их классу.

Как сделать выбор?

Возьмём объект самого уверенного классификатора.

Тогда возьмём самый уверенный классификатор:  $\alpha(x) = \arg \max_{k \in [1, \dots, K]} ((w_k, x))$

Но с этим тоже может быть проблема: предсказания классификаторов могут быть в разных масштабах, поэтому сравнивать их некорректно.

#### 7.1.2. All-Vs-All

Для каждой пары классов  $i$  и  $j$  обучим бинарный классификатор, который предсказывает класс  $i$  или класс  $j$ . Получается  $C_K^2$  классификаторов

Как здесь принимать итоговое решение? Выберем просто самый популярный класс.

Что здесь плохого: много классификаторов

## 7.2. Метрики качества

### 7.2.1. Micro-average

Если у нас  $k$  классификаторов, то для каждого находим  $TP_i, FP_i, FN_i, TN_i$  и итоговые считаем как  $TP = \frac{1}{k} \sum_{i=1}^k TP_i$

Тогда все наши изученные ранее для бинарной классификации метрики мы считаем по средним показателям

### 7.2.2. Macro-average

Макро усреднение отличается только тем, что такие метрики как precision, recall итд мы считаем для каждого классификатора, а потом усредняем их:

$$\text{precision}(\alpha, X) = \frac{1}{k} \sum_{i=1}^k \text{precision}_i(\alpha, X)$$

## 7.3. Многоклассовая логистическая регрессия

Напомним, что бинарная логистическая регрессия предсказывает вероятность класса:

$$(w, x) \rightarrow \alpha(x) = \frac{1}{1 + e^{-(w, x)}} = \frac{e^{(w, x)}}{1 + e^{(w, x)}}$$

Предположим у нас есть  $k$  моделей, каждая из которых даёт оценку принадлежности выбранному классу:  $b_k(x) = (w_k, x)$

Преобразуем вектор предсказаний в вектор вероятностей (softmax-преобразование):

$$\text{softmax}(b_1, \dots, b_k) = \left( \frac{\exp(b_1)}{\sum_{i=1}^k \exp(b_i)}, \frac{\exp(b_2)}{\sum_{i=1}^k \exp(b_i)}, \dots, \frac{\exp(b_k)}{\sum_{i=1}^k \exp(b_i)} \right)$$

Тогда вероятность класса  $k$ :

$$P(y = k|x, w) = \frac{\exp((w_k, x))}{\sum_{i=1}^k \exp((w_i, x))}$$

В иной форме, предсказание для  $j$ -го класса:

$$a_j(x) = P(y = j|x, w) = \frac{\exp(b_j(x))}{\sum_{i=1}^k \exp(b_i(x))}$$

Обучение будет происходить по методу максимального правдоподобия (аналогичной бинарной классификации):

$$\prod_{i=1}^n a_1(x_i)^{[y_i=1]} \cdot a_2(x_i)^{[y_i=2]} \cdots \cdot a_k(x_i)^{[y_i=k]} = \prod_{i=1}^n \prod_{j=1}^k a_j(x_i)^{[y_i=j]}$$

### 7.3.0.1. Многоклассовый Log-Loss

$$-\prod_{i=1}^n \prod_{j=1}^k [y_i = j] \log(P(y = j|x_i, w)) \rightarrow \min_{w_1, \dots, w_k}$$

## 7.4. Отбор признаков

На каком основании можно удалить признаки?

- Маленькая дисперсия - признак почти константный и не информативный
- Для линейных моделей можем посчитать его корреляцию с целевой переменной и выкинуть признаки, которые имеют по модулю маленькую корреляцию.
- Filtration methods - фильтрационные методы
- Wrapping methods - обёрточные методы
- Model selection - встроенные в модель методы отбора признаков

### 7.4.1. Фильтрационные методы

Фильтрационные методы - это отбор признаков по различным статистическим тестам. Идея метода состоит в вычислении влияния каждого признака в отдельности на целевую переменную (с помощью вычисления некоторой статистики)

#### 7.4.1.1. Преимущества Скорость: линейна от количества признаков

#### 7.4.1.2. sklearn

- SelectKBest - оставляет  $k$  признаков с наибольшим значением выбранной статистики
- SelectPercentile - оставляет признаки со значениями выбранной статистики, попавшие в заданный пользователем квантиль

#### 7.4.1.3. Тест $\chi^2$

Тест  $\chi^2$  используется в статистике для проверки независимости двух событий. поскольку  $\chi^2$  проверяет степень независимости между двумя переменными, а мы хотим сохранить только признаки, наиболее зависимые от метки, то будем вычислять  $\chi^2$  между каждым признаком и целевой переменной, сохраняя только признаки с наибольшими значениями

Критерий  $\chi^2$  можно применять только для бинарных или порядковых признаков

Статистика  $\chi^2$  вычисляется по формуле

$$\chi^2(X; Y) = \sum_{i,j} \frac{O_{ij} - E_{ij}}{E_{ij}}$$

где  $O_{ij}$  - наблюдаемая частота,  $E_{ij}$  - ожидаемая частота

Чем больше значение этой статистики - тем больше влияние этого признака на ответ (это эмпирический факт).

**7.4.1.4. Mutual Information** Для векторов  $X$  и  $Y$  статистика вычисляется по формуле

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left( \frac{p(x, y)}{p(x)p(y)} \right)$$

Где  $p(x, y)$  - вероятность того, что данный набор значений появился в нашей выборке (частоты)

Недостатки подхода статистического отбора: может быть так быть, что несколько признаков в целом мало влияют на целевую переменную, а в совокупности они влияют очень сильно.

## 7.4.2. Обёрточные методы

Обёрточные методы используют жадный отбор признаков, то есть последовательно выкидывают наименее подходящие по мнению методов признаки

**7.4.2.1. Recursive Feature Elimination** В sklearn есть обёрточный метод Recursive Feature Elimination (RFE)

Параметры метода:

- Алгоритм, используемый для отбора признаков
- Число признаков, которые мы хотим оставить

Недостаток: долгое время работы.

## 7.5. Встроенные в модель методы

### 7.5.1. $L_1$ регуляризация

Например,  $L_1$  регуляризация умеет отбирать признаки - некоторые веса зануляются, поэтому по сути это и есть отбор

$$Q(w) + \alpha \sum_{i=1}^d |w_i| \rightarrow \min_w$$

### 7.5.2. $L_0$ регуляризация

Это регуляризация, которая делает ограничение на кол-во признаков в модели:

$$Q(w) + \alpha \sum_{i=1}^d [w_i \neq 0] \rightarrow \min_w$$

### 7.5.3. Информационный критерий

Информационный критерий - мера качества модели, учитывающая степень "подгонки" модели под данные с корректировкой (штрафом) на используемое количество параметров

Информационные критерии основаны на компромиссе между точностью и сложностью модели

**7.5.3.1. Критерий Акаике (AIC)** Критерий Акаике (AIC - Akaike Information Criterion)

Мы дополнительно предполагаем, что модель  $\alpha$  - линейна, тогда

$$AIC(\alpha, X) = Q(\alpha, X) + \frac{2\hat{\sigma}^2}{l}n \rightarrow \min$$

где:

- $Q$  - функционал ошибки
- $\hat{\sigma}^2$  - оценка дисперсии ошибки  $D(y_i - \alpha(x_i))$
- $n$  - количество используемых признаков
- $l$  - число объектов

Если  $Q$  - среднеуadraticная ошибка для линейной регрессии и шумы нормально распределены, то

$$AIC = -\ln(\prod) + n$$

### 7.5.3.2. Критерий Шварца (BIC) Bayesian Information Criterion

$$BIC(\alpha X) = \frac{l}{\hat{\sigma}^2}(Q(\alpha, X) + \frac{\hat{\sigma}^2 \ln(l)}{l}n) \rightarrow \min$$

Если  $Q$  - среднеувадратичная ошибка для линейной регрессии и шумы нормально распределены, то

$$BIC = -\ln(\prod) + \frac{n}{2} \ln(l)$$

### 7.5.3.3. Отбор признаков с помощью информационных критериев

Если в модели  $k$  признаков (регрессоров), то существует  $2^k$  всевозможных моделей

В идеале необходимо построить все  $2^k$  моделей, для каждой посчитать значение критерия качества ( $AIC, BIC$ ) и выбрать модель, лучшую по этому критерию

При большом количестве регрессоров используют метод включений-исключений

Дотехать сложную часть с подсчётом

## 7.6. Метод главных компонент (PCA)

Principal Component Analysis - PCA

Предыдущие методы отбирали из исходных признаков некоторое подмножество признаков.

Теперь мы хотим придумать новые признаки, каким-то образом выражаются через старые, причём новых признаков хочется получить меньше, чем старых. В этой лекции рассмотрены только случаи, когда новые признаки линейно выражаются через старые.

Ссылка на МАД 1-го модуля

Лекция 5. Отбор признаков и визуализация. Наша лекция Будем обрабатывать данные про покемонов  
**Вставить работу с кодом**

## 7.7. Manifold Embeddings

Задача визуализации состоит в отображении объектов в 2D или 3D пространство с сохранением отношений между ними. Наша цель - просто посмотреть на данные

### 7.7.1. MultiDimensional Scaling (MDS)

Идея метода - минимизация квадратов отклонений между исходными и новыми попарными расстояниями

$$\sum_{i \neq j}^l (\rho(x_i, x_j) - \rho(z_i, z_j))^2 \rightarrow \min$$

где  $\rho$  - расстояние между объектами (метрика),  $x$  - старые объекты,  $z$  - новые объекты

### 7.7.2. ISOMAP

Для начала скажем, что

- Евклидово расстояние в 3D пространстве - как мы обычно привыкли мерить расстояние между предметами
- Геодезическое расстояние - расстояние между точками через ближайших соседей

Как найти Геодезическое расстояние?

1. Найдём  $n$  ближайших соседей
2. Соединим их и получим граф
3. Вычислим кратчайший путь по рёбрам между двумя точками

Для того, чтобы применить алгоритм ISOMAP, осталось только применить MDS для построения проекции в низкоразмерное пространство.

### 7.7.3. TSNE

t-SNE - t-distributed stochastic neighbor embedding

Идея на поверхности: при проекции нам важно не сохранение расстояний между объектами, а сохранение пропорций:

$$(x_1, x_2) = \alpha \rho(x_1, x_3) \rightarrow \rho(z_1, z_2) = \alpha \rho(z_1, z_3)$$

**7.7.3.1. Близость объектов в исходном пространстве** Если есть объекты  $i$  и  $j$ , то

$$\rho(i, j) = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_j^2)}{\sum_{k \neq j} \exp(-\|x_k - x_j\|^2 / 2\sigma_j^2)}$$

(затем симметризуем  $\rho(i, j)$ )

Объекты из окрестности  $x_j$  приближаются нормальным распределением. Чем кучнее объекты из этой окрестности - тем меньше берётся значение  $\sigma_j^2$ .

Поскольку при уменьшении пространства, точек станет меньше - какие-то станут дальше.

Тогда будем измерять сходство объектов в новом пространстве с помощью распределения Коши, так как оно не так сильно штрафует за увеличение расстояний между объектами:

$$q_{ij} = \frac{(1 + \|z_i - z_j\|^2)^{-1}}{\sum_{k \neq j} (1 + \|z_k - z_j\|^2)^{-1}}$$

Для построения проекций  $z_i$  объектов  $x_i$ , будем минимизировать расстояние между исходным и полученным распределениями (минимизируем дивергенцию Кульбака-Лейблера)

$$KL(p, q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \rightarrow \min_{z_1, \dots, z_l}$$

где  $p$  и  $q$  - распределение в старом и новом пространстве.

**Законспектить код питоновский**

## 8. Лекция 6. Нелинейные алгоритмы классификации и регрессии. Лекция ФЭН

### 8.1. Наивный байесовский классификатор

Наивный байесовский классификатор - это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков

Как пример: фрукт может считаться яблоком, если он:

- Красный
- Круглый
- Его диаметр составляет порядка 8 см

и мы предполагаем, что признаки вносят независимый вклад в вероятность того, что фрукт является яблоком: обычно на основании того, что фрукт круглый и красный - мы уже можем сказать, что это скорее всего яблоко, а в модели наивного байесовского классификатора - не можем.

#### 8.1.1. Теорема Байеса

$$\mathbb{P}(c|x) = \frac{\mathbb{P}(x|c) \cdot \mathbb{P}(c)}{\mathbb{P}(x)}$$

$$8.1.1.1. \text{ В случае нескольких признаков } \mathbb{P}(y|x_1, \dots, x_n) = \frac{\mathbb{P}(y) \cdot \prod_{i=1}^n \mathbb{P}(x_i|y)}{\prod_{i=1}^n \mathbb{P}(x_i)}$$

Вероятности вычисляются с помощью частотных таблиц, как и в одномерном случае.

#### 8.1.1.2. Преимущества

- Классификация быстрая и простая
- В случае, если выполняется предположение о независимости, классификатор показывает очень высокое качество

#### 8.1.1.3. Недостатки

- Если в тестовых данных присутствует категория, не встречавшаяся в данных для обучения, модель присвоит ей нулевую вероятность

## 8.2. Метод ближайших соседей

Идея: схожие объекты находятся близко к друг другу в пространстве признаков.

### 8.2.0.1. Классификация нового объекта

1. Вычисляем расстояние до каждого из объектов обучающей выборки
2. Выбираем  $k$  объектов, расстояние до которых минимально
3. Классифицируем этот объект, как самый часто встречающийся класс среди  $k$  ближайших соседей

И при этом, в качестве расстояния обычно берут Евклидову метрику, поэтому данные нужно масштабировать.

## 8.3. Решающие деревья

Решающее дерево - это бинарное дерево, в котором

- В каждой вершине  $v$  приписана функция (предикат)  $\beta_v : X \rightarrow \{0, 1\}$  для ровно одного признака
- В каждой листовой вершине  $v$  приписан прогноз  $c_v \in Y$  (для классификации - класс или вероятность класса, для регрессии - действительное значение целевой переменной)

### 8.3.1. Что влияет на построение решающего дерева

- Вид предикатов в вершинах
- Функционал качества  $Q(X, j, t)$
- Критерий останова

### 8.3.2. Критерий информативности

В каждой вершине оптимизируется функционал  $Q(X, j, t)$

Пусть  $R$  - множество объектов, попадающих в вершину на данном шаге, а  $R_l$  и  $R_r$  - объекты, попадающие в левую и правую ветки после разбиения

Цель: хотим, чтобы после разбиения объектов на две группы, внутри каждой группы как можно больше объектов было одного класса

$H(R)$  - критерий информативности - это мера неоднородности (разнообразности) целевой переменной. Чем меньше разнообразие целевой переменной внутри группы - тем меньше значение  $H(R)$ . То есть хотим решить задачу

$$H(R_l) \rightarrow \min, H(R_r) \rightarrow \min$$

Но это всё даёт сложность, потому что вместо одной задачи оптимизации - мы получаем две задачи оптимизации. Тогда можно определить функционал как

$$Q(R, j, t) = H(R) - \frac{|R_l|}{|R|} H(R_l) - \frac{|R_r|}{|R|} H(R_r) \rightarrow \max_{j,t}$$

где

- $R$  - объекты попавшие в вершину
- $j$  - номер признака
- $t$  - порог признака

В каждом листе дерево выдаёт константу  $c$  - вещественное число в регрессии? класс или вероятность класса в классификации.

#### 8.3.2.1. Функции потерь

Если в качестве функции потерь мы берём квадратичную ошибку, то

$$H(R) = \min_{c \in \mathbb{R}} \frac{1}{R} \sum_{(x_i, y_i) \in R} (y_i - c)^2$$

Её минимум достигается при

$$C = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} y_i$$

Информативность  $H(R)$  в вершине дерева - это дисперсия целевой переменной (для объектов, попавших в вершину)

#### 8.3.2.2. H(R) в задачах мягкой классификации

Будем в каждой вершине в качестве ответа выдавать не класс, а распределение вероятностей классов:

$$c = (c_1, \dots, c_k), \sum_{i=1}^k c_i = 1$$

Качество распределения можно измерить с помощью критерия Бриера, измеряющего точность вероятностных прогнозов:

$$H(R) = \min_c \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{j=1}^k (c_j - [y_i = j])^2$$

### 8.3.2.3. Критерий Джини

1. Минимальное значение функционала  $H(R)$  достигается на векторе, состоящем из долей классов:  
 $c_* = (p_1, \dots, p_k)$
2. На векторе  $c_*$  функционал (\*) переписывается в виде

$$H(R) = \sum_{k=1}^K p_k(1 - p_k)$$

### 8.3.2.4. Энтропийный критерий

Запишем логарифм правдоподобия:

$$H(R) = \min_c \left( -\frac{1}{|R|} \sum_{(x_t, y_t \in R)} \sum_{k=1}^K [y_i = k] \log(c_k) \right) (*)$$

На векторе  $c_* = (p_1, \dots, p_k)$  функционал (\*) записывается в виде

$$H(R) = -\sum_{k=1}^K p_k \log(p_k) \text{ (энтропия)}$$

### 8.3.3. Критерий останова

- Ограничение максимальной глубины дерева (max\_depth)
- Ограничение минимального числа объектов в листьях (min\_samples\_leaf)
- Ограничение максимального числа листьев в дереве
- Остановка в случае, если все объекты в листе из одного класса
- Требование, чтобы функционал качества при дроблении увеличивался как минимум на  $s\%$

### 8.3.4. Плюсы решающих деревьев

- Чёткие правила классификации (интерпретируемые предикаты, например, "возраст > 25")
- Деревья решений легко визуализируются, то есть хорошо интерпретируются
- Быстро обучаются и выдают прогноз
- Малое число параметров

### 8.3.5. Минусы решающих деревьев

- Очень чувствительны к шумам в данных, модель сильно меняется при небольшом изменении обучающей выборки
- Разделяющая граница имеет свои ограничения (состоит из гиперплоскостей)
- Необходимость борьбы с переобучением (стрижка или какой-либо из критериев останова)
- Проблема поиска оптимального дерева (NP-полнная задача, поэтому на практике используется жадное построение дерева)

## 9. Лекция 6. Решающие деревья. Наша Лекция. Елена

### 9.1. ROC-AUC: интуиция

Пусть есть предсказание модели и правильный класс. Отсортируем по убыванию вероятности:

p	класс	p	класс
0.5	0	0.6	1
0.1	0	0.5	0
0.25	0	0.3	1
0.6	1	0.25	0
0.2	1	0.2	1
0.3	1	0.1	0
0.0	0	0.0	0

*Небольшое дополнение:* порог не обязательно ставить 0.5, можно хоть 0.3

Как у хорошей модели будет выглядеть столбец с классами после сортировки по вероятности? По хорошему должны стоять сначала подряд все единицы, а потом все нули.

Если модель чередует единицы и нолики - это плохая модель. В нашем случае как раз видим такое.

#### 9.1.1. ROC-AUC алгоритм

1. Нарисуем квадрат  $1 \times 1$
  2. Горизонтальную сторону ( $x$ ) разобьём на равные отрезки, число которых равно количеству объектов с классом 0
  3. Вертикальную сторону разобьём на равные отрезки, число которых равно количеству объектов с классом 1
  4. Стартуем из точки  $(0, 0)$
  5. Идём сверху вниз нашей таблицы:
    - если попалась единица - шагаем в клетку  $(n, m + 1)$
    - если попался ноль - шагаем в клетку  $(n + 1, m)$
- ( $+1$  естественно по размеченной сетке, а не в прямом смысле  $+1$ )

В идеальном случае мы должны сначала достигнуть угла  $(1, 0)$  нашего квадрата, а потом уйти в  $(1, 1)$ . Сама кривая называется ROC-кривая, а метрика ROC-AUC считается как площадь под нашей описанной кривой.

У идеальной модели площадь будет равна 1.

У случайной модели (при одинаковом кол-ве классов) будет зигзаг около главной диагонали, и площадь будет примерно 0.5.

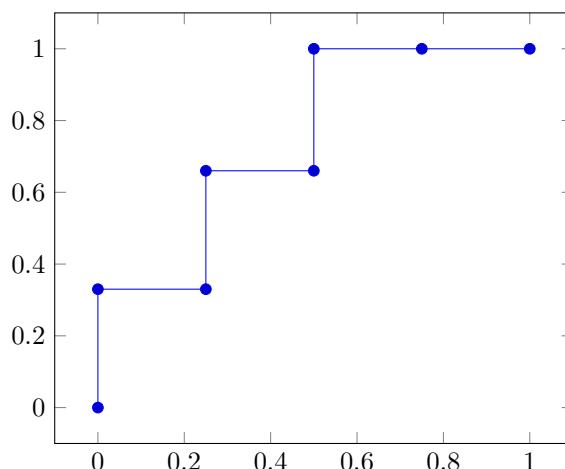


Рис. 1: ROC-AUC нашей задачи

И ROC-AUC метрика у нас будет примерно 0.75.

## 9.2. Решающие деревья

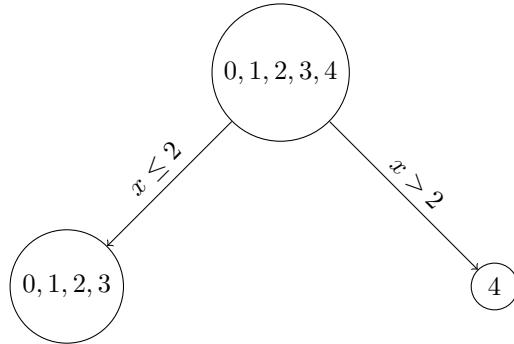
### 9.2.1. Задачи

**9.2.1.1. Задача 1** Постройте регрессионное дерево для прогнозирования  $y$  с помощью  $x$  на обучающей выборке

$x_i$	0	1	2	3
$y_i$	5	6	4	100

Критерий деления узла на два - минимизация RSS (MSE). Дерево строится до трёх терминальных узлов.

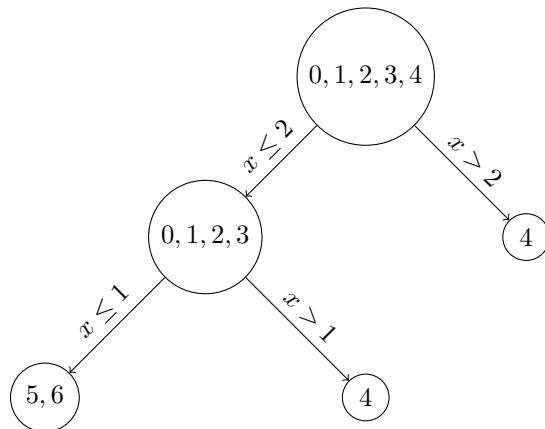
**9.2.1.2. Решение задачи 1** На первом шаге можно взять разделяющим критерием " $x > 2$ ".



Теперь у нас есть вариант сделать разделяющую поверхность  $x > 1$  или  $x > 0$ . Проверим как получится метрика тогда:

- Если мы разбиваем по  $x > 1$ , то в левом поддереве уйдут вершины с  $y_i = \{5, 6\}$ , в правую  $y_i = \{4\}$ . В левом поддереве ответ получится 5.5, а в правом 4. Для левого поддерева MSE получится  $\frac{(5-5.5)^2+(6-5.5)^2}{2} = 0.25$ . Для правого поддерева  $MSE = 0$ .
- Если бьём по  $x > 0$ , то в левом поддереве уйдёт вершина  $y_i = \{5\}$ , а в правом поддереве  $y_i = \{6, 4\}$ . Значит будет предсказано в левом поддереве ответ 5 и в другом 5. Для левого поддерева  $MSE = 0$ , а для правого  $\frac{(6-5)^2+(4-5)^2}{2} = 1$ .

MSE получится лучше, если разделять по " $x > 1$ ":



**9.2.1.3. Задача 3** Дон-Жуан предпочитает брюнеток. Перед Новым Годом он посчитал, что в записной книжке у него 20 блондинок, 40 брюнеток, две рыжих и восемь шатенок. С Нового Года Дон-Жуан решил перенести все сведения в две записные книжки. В одну - брюнеток, во вторую - остальных.

Как изменился индекс Джини и энтропия в результате такого разбиения?

#### 9.2.1.4. Решение задачи 3 До разделения на две книжки:

$$H = - \sum_{i=1}^n p_i \cdot \log(p_i) = -\left(\frac{20}{70} \log\left(\frac{20}{70}\right) + \frac{40}{70} \log\left(\frac{40}{70}\right) + \frac{2}{70} \log\left(\frac{2}{70}\right) + \frac{8}{70} \log\left(\frac{8}{70}\right)\right) = 1.02$$

После разбиения на две книжки:

Книжка с брюнетками:

$$H(L) = -(1 \cdot \log(1)) = 0$$

Книжка с остальными:

$$H(R) = -\left(\frac{20}{30} \log\left(\frac{20}{30}\right) + \frac{2}{30} \log\left(\frac{2}{30}\right) + \frac{8}{30} \log\left(\frac{8}{30}\right)\right) = 0.8$$

#### 9.2.1.5. Задача 4 Привидите набор данных, для которых индекс Джини равен 0, 0.5 и 0.999

#### 9.2.1.6. Решение задачи 4 Индекс Джини: $G = \sum_{i=1}^K p_i(1 - p_i)$

- Для индекса Джини 0 достаточно, чтобы все данные принадлежали одному классу:  $1 \cdot (1 - 1) = 0$
- Для индекса Джини 0.5 достаточно, чтобы было два класса и объектов было поровну:  $0.5 \cdot (1 - 0.5) + 0.5 \cdot (1 - 0.5) = 0.25 + 0.25 = 0.5$
- Для индекса Джини 0.999 достаточно, чтобы было 1000 классов и каждого было по одному объекту:  

$$\overbrace{\frac{1}{1000} \cdot (1 - \frac{1}{1000}) + \dots + \frac{1}{1000} \cdot (1 - \frac{1}{1000})}^{1000 \text{ раз}} = 1000 \cdot \frac{1}{1000} \cdot (1 - \frac{1}{1000}) = 1 - \frac{1}{1000} = 0.999$$

### 9.3. Работа с кодом

#### 9.3.1. Подключение библиотек

```
import matplotlib.pyplot as plt
%matplotlib inline
from mlxtend.plotting import plot_decision_regions
import numpy as np
from sklearn.datasets import load_boston
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
import pandas as pd

plt.rcParams["figure.figsize"] = (11, 6.5)
```

#### 9.3.2. Генерация выборки для задачи регрессии

```
n = 400
np.random.seed(1)
X = np.zeros((n, 2))
X[:, 0] = np.linspace(-5, 5, n)
X[:, 1] = X[:, 0] + 0.5 * np.random.normal(size=n)
y = (X[:, 1] > X[:, 0]).astype(int)

plt.scatter(X[:, 0], X[:, 1], s=100, c=y, cmap='bwr')
plt.show()
```

#### 9.3.3. Функция для обучения классификатора и построения разделяющей прямой

```
def train_model(model=LogisticRegression()):
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```

plot_decision_regions(X_test, y_test, model)
plt.show()

print(f"Accuracy: {accuracy_score(y_pred, y_test):.2f}")

```

### 9.3.4. Сравнение качества между логистической регрессией и деревом решений

```

train_model(LogisticRegression())
train_model(DecisionTreeClassifier(random_state=13))

```

### 9.3.5. Генерация выборки логического ИЛИ

```

X = np.random.randn(n, 2)
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0).astype(int)
plt.scatter(X[:, 0], X[:, 1], s=100, c=y, cmap="bwr")
plt.show()

```

### 9.3.6. Сравнение качества между логистической регрессией и деревом решений

```

train_model(LogisticRegression())
train_model(DecisionTreeClassifier(random_state=13))

```

### 9.3.7. Переобучение дерева

Решающие деревья могут переобучаться под любую выборку, если их не регуляризовать: при большом количестве листьев для каждого объекта может выделиться своя область в признаковом пространстве. Дерево просто выучивает обучающую выборку, но не выделяет закономерности в данных. Давайте убедимся в этом на практике.

```

np.random.seed(13)
n = 100
X = np.random.normal(size=(n, 2))
X[:50, :] += 0.25
X[50:, :] -= 0.25
y = np.array([1] * 50 + [0] * 50)
plt.scatter(X[:, 0], X[:, 1], s=100, c=y, cmap="bwr")
plt.show()

```

```

fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(15, 12))

for i, max_depth in enumerate([3, 5, None]):
    for j, min_samples_leaf in enumerate([1, 5, 15]):
        dt = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_samples_leaf)
        dt.fit(X, y)
        ax[i][j].set_title(f"max_depth={max_depth} | min_samples_leaf={min_samples_leaf}")
        ax[i][j].axis("off")
        plot_decision_regions(X, y, dt, ax=ax[i][j])

plt.show()

```

**9.3.7.3. Дерево с нулевой ошибкой** При этом нужно сказать, что если никак не ограничивать модель, то она может выдавать нулевую ошибку, но при этом сильно переобучиться.

```
model = DecisionTreeClassifier(max_depth=None, min_samples_leaf=1, random_state=13)
model.fit(X, y)

print(f"Accuracy: {accuracy_score(y, dt.predict(X)):.2f}")

plot_decision_regions(X, y, model)
plt.show()
```

### 9.3.8. Неустойчивость

Дерево обычно очень сильно меняется от изменения выборки. Давайте попробуем давать дереву разные 90% от объектов.

```
fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(15, 12))

for i in range(3):
    for j in range(3):
        seed_idx = 3 * i + j
        np.random.seed(seed_idx)
        dt = DecisionTreeClassifier(random_state=13)
        idx_part = np.random.choice(len(X), replace=False, size=int(0.9 * len(X)))
        X_part, y_part = X[idx_part, :], y[idx_part]
        dt.fit(X_part, y_part)
        ax[i][j].set_title(f"sample#{j}.format(seed_idx))")
        ax[i][j].axis("off")
        plot_decision_regions(X_part, y_part, dt, ax=ax[i][j])

plt.show()
```

Мы бы хотели, чтобы от одинакового по размеру датасета и примерно со схожими данными наша модель не менялась, но у деревьев, как мы видим так не происходит.

### 9.3.9. Анализ деревьев на основе датасета Бостона

#### 9.3.9.1. Загрузка датасета .

```
boston = load_boston()
print(boston["DESCR"])

X = pd.DataFrame(data=boston["data"], columns=boston["feature_names"])
y = boston["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=13)

print(f"Shape:{X.shape}")
X.head()
```

#### 9.3.9.2. Построение и отображение дерева с предикатами .

```
tree = DecisionTreeRegressor(max_depth=3, random_state=13)
tree.fit(X_train, y_train)

plot_tree(tree, feature_names=X.columns, filled=True, rounded=True)
plt.show()
```

#### 9.3.9.3. Получение построенных параметров дерева .

```
n_nodes = tree.tree_.node_count
children_left = tree.tree_.children_left
children_right = tree.tree_.children_right
feature = tree.tree_.feature
threshold = tree.tree_.threshold
```

**9.3.9.4. Построение зависимости MSE от гиперпараметра max\_depth и отбор лучших значений на кросс-валидирующей выборке .**

```
from sklearn.model_selection import cross_val_score

max_depth_array = range(2, 20)
mse_array = []

for max_depth in max_depth_array:
    tree = DecisionTreeRegressor(max_depth=max_depth, random_state=13)
#    tree.fit(X_train, y_train)
#    mse_array.append(mean_squared_error(y_test, tree.predict(X_test)))
    mse = -cross_val_score(tree, X, y, cv=3, scoring='neg_mean_squared_error').mean()
    mse_array.append(mse)

plt.plot(max_depth_array, mse_array)
plt.title("Dependence_of_MSE_on_max_depth")
plt.xlabel("max_depth")
plt.ylabel("MSE")
plt.show()

pd.DataFrame({"max_depth": max_depth_array, "MSE": mse_array}).sort_values(by="MSE").reset_index()
```

**9.3.9.5. Построение зависимости MSE от гиперпараметра max\_samples\_leaf и отбор лучших значений на кросс-валидирующей выборке .**

```
min_samples_leaf_array = range(1, 20)
mse_array = []

for min_samples_leaf in min_samples_leaf_array:
    dt = DecisionTreeRegressor(max_depth=4, min_samples_leaf=min_samples_leaf, random_state=13)
    res = -cross_val_score(dt, X, y, cv=3, scoring='neg_mean_squared_error').mean()
    mse_array.append(res)

plt.plot(min_samples_leaf_array, mse_array)
plt.title("Dependence_of_MSE_on_min_samples_leaf")
plt.xlabel("min_samples_leaf")
plt.ylabel("MSE")
plt.show()

pd.DataFrame({"min_samples_leaf": min_samples_leaf_array, "MSE": mse_array}).sort_values(by="MSE").reset_index()
```

**9.3.9.6. Разделение выборки на обучающую и тестовую .**

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)
```

**9.3.9.7. Построение дерева без ограничивающих параметров .**

```
tree = DecisionTreeRegressor()

tree.fit(Xtrain, ytrain)

pred_train = tree.predict(Xtrain)
pred_test = tree.predict(Xtest)

from sklearn.metrics import r2_score

r2_score(ytrain, pred_train), r2_score(ytest, pred_test)
```

### 9.3.9.8. Поиск лучших параметров max\_depth и max\_features .

```
from sklearn.model_selection import GridSearchCV

params = { 'max_depth' : np.arange(2, 12),
           'max_features' : [ "auto", "sqrt", "log2" ] }

gs = GridSearchCV(DecisionTreeRegressor(), params, cv=3, scoring='r2')

gs.fit(X, y)

print(gs.best_estimator_)
```

### 9.3.9.9. Стрижка дерева Мы можем как обычно делать при переобучении: добавлять регуляризатор, и выглядит он как

$$Q_{\text{now}} = Q + \alpha|T|$$

где  $T$  - число вершин в дереве, а  $\alpha$  - наш параметр

```
path = tree.cost_complexity_pruning_path(Xtrain, ytrain)
alphas = path['ccp_alphas']
```

в  $\text{alphas}$  лежат все значения параметра регуляризации, которые нам нужно рассмотреть

### 9.3.9.10. Перебор по гиперпараметру регуляризации .

```
import seaborn as sns
from sklearn.metrics import r2_score

accuracy_train, accuracy_test = [], []
MaxR2 = -1
Alpha = 0

for i in alphas:
    tree = DecisionTreeRegressor(ccp_alpha = i)

    tree.fit(Xtrain, ytrain)
    y_train_pred = tree.predict(Xtrain)
    y_test_pred = tree.predict(Xtest)

    accuracy_train.append(r2_score(ytrain, y_train_pred))
    accuracy_test.append(r2_score(ytest, y_test_pred))

    R2 = r2_score(ytest, y_test_pred)
    if R2 > MaxR2:
        MaxR2 = R2
        Alpha = i

sns.set()
plt.figure(figsize=(14,7))
sns.lineplot(y = accuracy_train, x = alphas, label = "Train_r2")
sns.lineplot(y = accuracy_test, x = alphas, label = "Test_r2")
plt.xticks(ticks = np.arange(0.00, 0.25, 0.01))
plt.show()

print('Best_alpha:', Alpha)
```

### 9.3.9.11. Применение лучшего параметра регуляризации .

```
tree3 = DecisionTreeRegressor(ccp_alpha = Alpha)

tree3.fit(Xtrain, ytrain)

pred_train3 = tree3.predict(Xtrain)
pred_test3 = tree3.predict(Xtest)

r2_score(ytrain, pred_train3), r2_score(ytest, pred_test3)
```

### 9.3.10. Решающее дерево своими руками

## 9.4. Калибровка вероятностей

Почти у всех моделей, которые мы используем, есть две опции:

- predict - возвращает предсказание класса
- predict\_proba - возвращает вероятность класса, насколько классификатор уверен в своём предсказании

Допустим у нас есть 10 одинаковых объекта, но целевая переменная у 6 из них относится к одному классу, а у 4-ёх относится к другому. Тогда наша модель будет уверена на 0.6 в том, что такой же объект будет отнесён к первому классу.

Для большинства моделей эти числа никак не связаны с реальными вероятностями. Но эти результаты можно откалибровать. Чтобы возвращаемое значение predict\_proba правда выдавало что-то похожее на реальные вероятности.

### 9.4.1. Программируем это всё дело

#### 9.4.1.1. Генерация выборки .

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification(
    n_samples=100_000, n_features=20, n_informative=2, n_redundant=10, random_state=42
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.99, random_state=42
)
```

**9.4.1.2. Обучение моделей** Давайте обучим модели и для дерева используем CalibratedClassifierCV, который как раз и калибрует вероятности.

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

from sklearn.calibration import CalibratedClassifierCV, CalibrationDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

lr = LogisticRegression(C=1.0)
dt = DecisionTreeClassifier(max_depth = 6)
dt_isotonic = CalibratedClassifierCV(dt, cv=3, method="isotonic")
dt_sigmoid = CalibratedClassifierCV(dt, cv=3, method="sigmoid")

clf_list = [
    (lr, "Logistic"),
    (dt, "Decision_Tree"),
    (dt_isotonic, "Decision_Tree+Isotonic"),
```

```
] ( dt_sigmoid , "DecisionTree+Sigmoid" ) ,
```

#### 9.4.1.3. Отобразим наши вероятности .

```
fig = plt.figure(figsize=(10, 10))
gs = GridSpec(4, 2)
colors = plt.cm.get_cmap("Dark2")

ax_calibration_curve = fig.add_subplot(gs[:2, :2])
calibration_displays = {}
for i, (clf, name) in enumerate(clf_list):
    clf.fit(X_train, y_train)
    display = CalibrationDisplay.from_estimator(
        clf,
        X_test,
        y_test,
        n_bins=10,
        name=name,
        ax=ax_calibration_curve,
        color=colors(i),
    )
    calibration_displays[name] = display

ax_calibration_curve.grid()
ax_calibration_curve.set_title("Calibration plots (Naive Bayes)")

# Add histogram
grid_positions = [(2, 0), (2, 1), (3, 0), (3, 1)]
for i, (_, name) in enumerate(clf_list):
    row, col = grid_positions[i]
    ax = fig.add_subplot(gs[row, col])

    ax.hist(
        calibration_displays[name].y_prob,
        range=(0, 1),
        bins=10,
        label=name,
        color=colors(i),
    )
    ax.set(title=name, xlabel="Mean predicted probability", ylabel="Count")

plt.tight_layout()
plt.show()
```

При этом на качество и метрики эта калибровка никак не влияет. Это влияет только на вероятности, выдаваемые моделью.

# 10. Материалы к лекции 6

## 10.1. Статья про разложение ошибки

Ссылка на статью - <https://habr.com/ru/company/ods/blog/323890/>

Допустим у нас есть функция предсказания  $f(x)$ . Реальное значение целевой переменной выглядит как  $y = f(x) + \epsilon$ , где  $\epsilon \sim N(0, \sigma^2)$ , а  $y \sim N(f(x), \sigma^2)$ . Пусть также есть точечная оценка для  $f$  -  $\hat{f}$

Тогда ошибка в точке  $x$  раскладывается как

$$Err(x) = \mathbb{E}[(y - \hat{f}(x))^2] = \mathbb{E}(y^2) + \mathbb{E}(\hat{f}(x)^2) - 2\mathbb{E}(y \cdot \hat{f}(x)) =$$

Мы помним, что  $Var(x) = \mathbb{E}(x^2) - \mathbb{E}(x)^2$ , тогда

$$\begin{aligned}\mathbb{E}(y^2) &= Var(y) + \mathbb{E}(y)^2 \\ \mathbb{E}(f(x)^2) &= Var(f(x)) + E(f(x))^2\end{aligned}$$

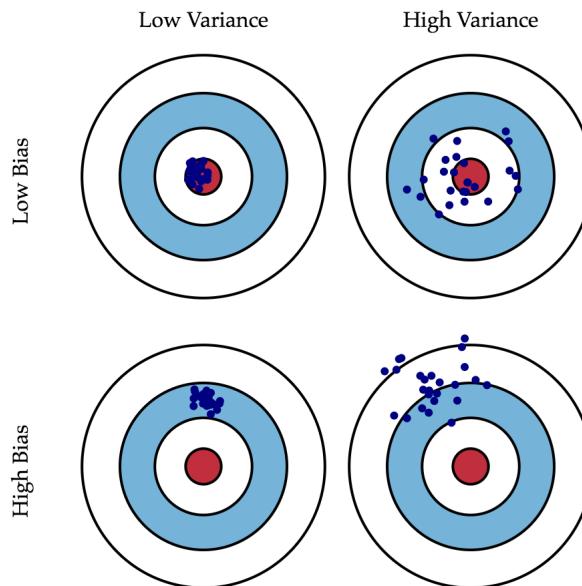
и ещё

$$\mathbb{E}(y \cdot \hat{f}) = \mathbb{E}((f + \epsilon) \cdot \hat{f}) = \mathbb{E}(f\hat{f}) + \mathbb{E}(\epsilon\hat{f}) = f \cdot \mathbb{E}(\hat{f}) + \mathbb{E}(\epsilon) \cdot \mathbb{E}(\hat{f}) = f\mathbb{E}(\hat{f})$$

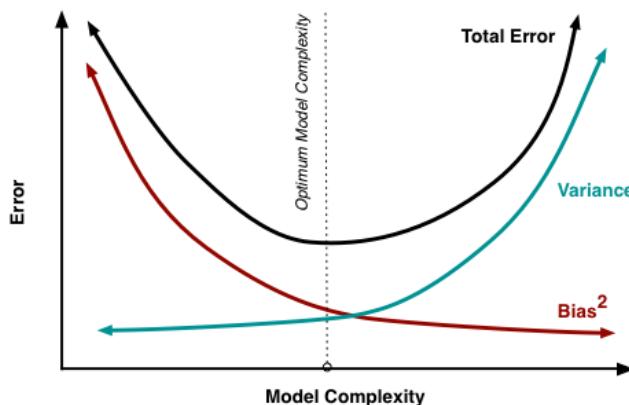
Собрав всё вместе - считаем:

$$Err(x) = \mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + f^2 + Var(\hat{f}) + \mathbb{E}(\hat{f})^2 - 2f\mathbb{E}(\hat{f}) = (f - \mathbb{E}(\hat{f}))^2 + Var(\hat{f}) + \sigma^2 = Bias(\hat{f})^2 + Var(\hat{f}) + \sigma^2$$

С последним слагаемым мы ничего сделать не можем, а вот первые два мы вполне можем попытаться минимизировать. Если делать ничего не будем, то нас ждёт примерно такой результат:



Как правило, при увеличении сложности модели (например, при увеличении количества свободных параметров) увеличивается дисперсия (разброс) оценки, но уменьшается смещение. Из-за того что тренировочный набор данных полностью запоминается вместо обобщения, небольшие изменения приводят к неожиданным результатам (переобучение). Если же модель слабая, то она не в состоянии выучить закономерность, в результате выучивается что-то другое, смещенное относительно правильного решения.



## 10.2. Уроки 6.4, 6.5 со степика

# 11. Лекция 7. Композиция алгоритмов, разложение ошибки и лес

Ссылка на видео - [https://www.youtube.com/watch?v=X4arg\\_OLxUk&list=PLEwK9wdS5g0qi14fXKFnFzruUDg3n16db&index=34](https://www.youtube.com/watch?v=X4arg_OLxUk&list=PLEwK9wdS5g0qi14fXKFnFzruUDg3n16db&index=34)

## 11.1. Смещение и разброс

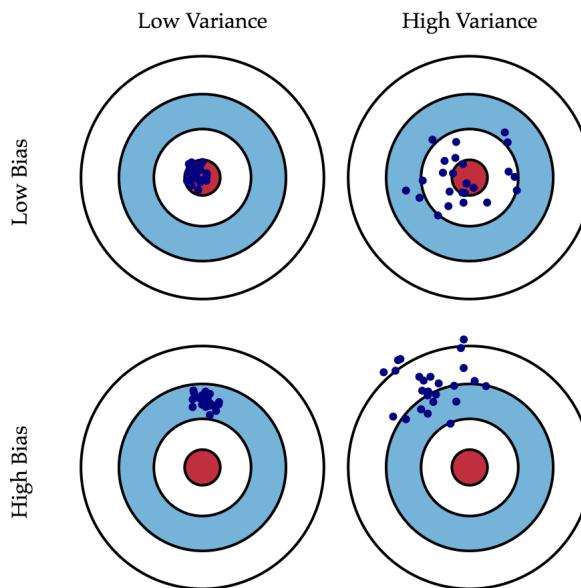
Зачастую для улучшения качества модели необходимо понять из-за чего возникает ошибка в предсказаниях. Ошибку можно представить в виде

$$Err(x) = Bias^2(\alpha(x)) + Var(\alpha(x)) + \sigma^2$$

где

- $Bias(\alpha(x))$  - средняя ошибка по всем возможным наборам данных - смещение. Смещение показывает насколько в среднем модель хорошо предсказывает целевую переменную. Маленькое смещение - хорошее предсказание, большое смещение - плохое предсказание
- $Var(\alpha(x))$  - дисперсия ошибки, то есть как сильно различается ошибка при обучении на различных наборах данных - разброс. Большой разброс означает, что ошибка очень чувствительна к изменению обучающей выборки. Большой разброс - сильно переобученная модель
- $\sigma^2$  - неустранимая ошибка, шум в данных

И их влияние можно увидеть на картинке



Что означают синие точки на этой картинке?

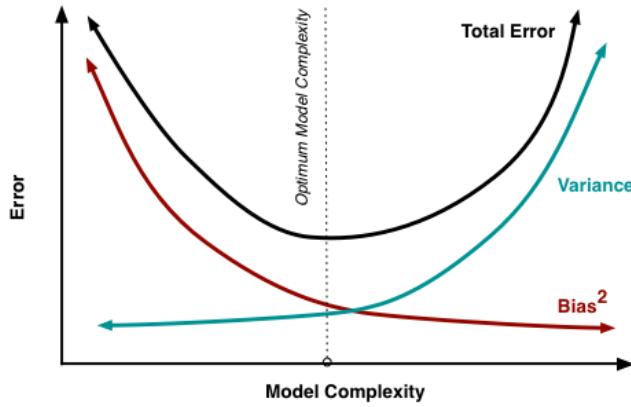
Мы берём модель и одна синяя точка отвечает просто за один набор обучающих данных, а другая за другой. То есть на картинке изображено предсказание одного и того же объекта, но при каждом предсказании обучающая выборка разная для модели, с помощью которой предсказывали.

Что мы понимаем под сложностью модели? Под сложностью модели мы понимаем число её параметров, количество весов, которые мы настраиваем в процессе обучения. Не гипер-параметры. Например у линейной регрессии мало параметров. У метода ближайших соседей вообще нет параметров. А если брать решающие деревья или случайные леса - там много параметров.

Оказывается, модели, у которых мало параметров - могут плохо решить задачу, у них большое смещение. Но из-за того, что параметров мало - они не подогнаны под наши данные. У моделей получается большое смещение, но маленький разброс.

Чем сложнее модель, тем лучше она решает задачу и у неё уменьшается смещение, но увеличивается разброс.

Эти тезисы изображены на рисунке



### 11.1.1. Формула для разложения ошибки на смещение и разброс

Пусть есть какая-то истинная зависимость:  $y = f(x) + \epsilon$ .

Пусть  $\epsilon \sim N(0, \sigma^2)$

Мы строим модель:  $x \rightarrow \alpha(x)$ . Измерять ошибку будем метрикой MSE.

Мы хотим найти  $\mathbb{E}((y - \alpha)^2)$ .

Среднюю ошибку мы вычисляем по всем одинаковым объектам: по всем одинаковым  $x$ .  $x$  бывают очень часто одинаковые: для предсказания стоимости квартиры у нас могут совпасть все данные, которые у нас имеются.

Давайте раскроем нашу ошибку:

$$\mathbb{E}((y - \alpha)^2) = \mathbb{E}(y^2 + \alpha^2 - 2y\alpha) = E(y^2) + \mathbb{E}(\alpha^2) - \mathbb{E}(2y\alpha)$$

Произведём трюк: добавим и вычтем одно и те же слагаемые:

$$E(y^2) - \mathbb{E}(y^2) + \mathbb{E}(y^2) + \mathbb{E}(\alpha^2) - \mathbb{E}(\alpha^2) + \mathbb{E}(\alpha^2) - \mathbb{E}(2y\alpha) = Var(y) + Var(\alpha) + (\mathbb{E}(y)^2 - \mathbb{E}(2y\alpha) + \mathbb{E}(\alpha)^2) = Var(y) + Var(\alpha) + (\mathbb{E}(y) - \mathbb{E}(\alpha))^2$$

При этом у нас  $\mathbb{E}(y) = \mathbb{E}(f(x) + \epsilon) = \mathbb{E}(f(x)) = f(x)$  потому что  $\epsilon$  имеет нулевое матожидание, а  $f(x)$  - константа. Продолжим вычисления:

$$Var(y) + Var(\alpha) + (\mathbb{E}(y) - \mathbb{E}(\alpha))^2 = Var(y) + Var(\alpha) + (f - \mathbb{E}(\alpha))^2$$

При этом  $Var(y)$  - это шум в данных. К примеру, когда некоторые объекты одинаковые, а целевая переменная у них разная.

## 11.2. Бэггинг

Для того, чтобы понять что такое бэггинг, нужно дать другое определение

### 11.2.1. Бутстрэп

Бутстрэп - метод генерации данных.

Мы из нашей исходной выборки  $X$  сгенерируем  $n$  выборок  $X_1, \dots, X_n$ . Каждая из выборок будет сгенерирована взятием объектов с возвращением (то есть объекты могут повторяться внутри новой выборки).

### 11.2.2. Описание бэггинга

Далее для каждой выборки будем обучать алгоритм из одного и того же семейства. Получится  $n$  моделей:  $b_1(x), \dots, b_n(x)$ .

Построим новую функцию регрессии:

$$\alpha(x) = \frac{1}{n} \sum_{j=1}^n b_j(x)$$

Это и называется бэггинг.

В случае классификации это может быть голосование.

Такое усреднение будет снижать переобучение, которое было бы, если бы мы обучали только одну модель (в нашем случае дерево).

### 11.2.3. Смещение и разброс у бэггинга

- Бэггинг не ухудшает смещенность модели: смещение  $\alpha_n(x)$  равно смещению базового алгоритма
- Если базовые алгоритмы некоррелированы, то дисперсия бэггинга  $\alpha_n(x)$  в  $n$  раз меньше дисперсии отдельных базовых алгоритмов.

### 11.2.4. Как сделать некоррелированные алгоритмы

Пусть наши базовые алгоритмы  $\mu_1(x), \dots, \mu_n(x)$ , а наша композиция  $\alpha(x) = \frac{1}{n} \sum_{i=1}^n \mu_i(x)$

Что мы хотим доказать? Что смещение у бэггинга такое же, как у одного дерева, а разброс меньше.

- Смещение:  $\mathbb{E}(\alpha(x) - f(x))^2 = \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n \mu_i(x) - f\right)^2 = \left(\frac{1}{n} \sum_{i=1}^n \mu_i(x) - f\right)^2 = (\mathbb{E}(\mu_i(x)) - f)^2$
- Разброс:  $Var(\alpha) = \mathbb{E}(\alpha - \mathbb{E}(\alpha))^2 = \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^n \mu_i(x) - \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n \mu_i(x)\right)\right]^2 = \frac{1}{n^2} \mathbb{E}\left[\sum_{i=1}^n \mu_i(x) - \mathbb{E}\left(\sum_{i=1}^n \mu_i(x)\right)\right]^2 = \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}[\mu_i(x) - \mathbb{E}\mu_i(x)] + \underbrace{\frac{1}{n^2} \sum_{i_1 \neq i_2} \mathbb{E}[(\mu_{i_1}(x) - \mathbb{E}\mu_{i_1}(x))(\mu_{i_2}(x) - \mathbb{E}\mu_{i_2}(x))]}_{=0 \text{ из-за некоррелированности}} = \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}[\mu_i(x) - \mathbb{E}\mu_i(x)] = \frac{1}{n} Var(\mu(x)).$  Уменьшили разброс.

## 11.3. Случайный лес (Random Forest)

Возьмём в качестве базовых алгоритмов для бэггинга решающие деревья, то есть каждое случайное дерево  $b_i(x)$  построено по своей предвыборке  $X_i$

В каждой вершине дерева будем искать разбиение не по всем признакам, а по подмножеству признаков.

Дерево строится до тех пор, пока в листе не окажется  $n_{min}$  объектов

### 11.3.1. Практические рекомендации

Если  $p$  - количество признаков, то при классификации обычно берут  $m = \lceil \sqrt{p} \rceil$ , а при регрессии  $m = \lceil \frac{p}{3} \rceil$  признаков.

При классификации обычно дерево строится, пока в листе не окажется  $n_{min} = 1$  объект, а при регрессии  $n_{min} = 5$  объектов.

### 11.3.2. Out-Of-Bag ошибка

Для каждого дерева посчитаем  $Err_i$  на тех объектах, на которых оно не обучалось. Итоговая ошибка рассчитывается как

$$Err_{oob} = \frac{1}{B} \sum_{i=1}^B Err_i$$

Это некий аналог cross-валидации.

При большом количестве деревьев, метрика Out-Of-Bag даёт очень хороший результат. По ООВ можно подобрать оптимальное кол-во деревьев.

## 12. Лекция 7. Случайный лес. Наша лекция

Вспомним, что математическое ожидание среднеквадратичной ошибки для различных вариаций тренировочной выборки можно разложить как

$$\mathbb{E}(y - \hat{y})^2 = bias^2 + Var + noise$$

### 12.1. Программируем

#### 12.1.1. Загружаем датасет

```
from sklearn.model_selection import train_test_split
from mlxtend.data import boston_housing_data

X, y = boston_housing_data()
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 123,
                                                    shuffle = True)
```

#### 12.1.2. Обучим решающее дерево без ограничений

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

dt = DecisionTreeRegressor(random_state = 123)
dt.fit(X_train, y_train)
print("MSE_train:", mean_squared_error(y_train, dt.predict(X_train)))
print("MSE_test:", mean_squared_error(y_test, dt.predict(X_test)))
```

#### 12.1.3. Получим примерное смещение и разброс

```
from mlxtend.evaluate import bias_variance_decomp

_, avg_bias, avg_var = bias_variance_decomp(dt, X_train, y_train, X_test, y_test,
                                             loss = 'mse', random_seed = 123)
```

#### 12.1.4. Посмотрим на смещение и разброс у бэггинга деревьев

```
from sklearn.ensemble import BaggingRegressor

base_tree = DecisionTreeRegressor(random_state = 123)

bagging = BaggingRegressor(base_estimator = base_tree, n_estimators = 20, random_state = 123)
_, avg_bias, avg_var = bias_variance_decomp(bagging, X_train, y_train, X_test, y_test,
                                             loss = 'mse', random_seed = 123)
```

#### 12.1.5. Посмотрим как это повлияло на MSE

```
print("MSE_train:", mean_squared_error(y_train, bagging.predict(X_train)))
print("MSE_test:", mean_squared_error(y_test, bagging.predict(X_test)))
```

### 12.1.6. Построим случайный лес

```
from sklearn.ensemble import RandomForestRegressor  
  
rf = RandomForestRegressor(n_estimators = 20, random_state = 123)  
  
, avg_bias, avg_var = bias_variance_decomp(rf, X_train, y_train, X_test, y_test,  
                                             loss = 'mse', random_seed = 123)  
  
print("MSE_train:", mean_squared_error(y_train, rf.predict(X_train)))  
print("MSE_test:", mean_squared_error(y_test, rf.predict(X_test)))
```

### 12.1.7. Переобучение случайного леса

Давайте посмотрим на ошибку случайного леса в зависимости от количества деревьев

```
n_trees = 100  
train_loss = []  
test_loss = []  
  
for i in range(1, n_trees):  
    rf = RandomForestRegressor(n_estimators = i, random_state = 123)  
    rf.fit(X_train, y_train)  
    train_loss.append(mean_squared_error(y_train, rf.predict(X_train)))  
    test_loss.append(mean_squared_error(y_test, rf.predict(X_test)))  
  
plt.figure(figsize = (10, 7))  
plt.grid()  
plt.plot(train_loss, label = 'MSE_train')  
plt.plot(test_loss, label = 'MSE_test')  
plt.ylabel('MSE')  
plt.xlabel('#trees')  
plt.legend()
```

## 12.2. Важность признаков

**Relative importance** - важность признака в относительных величинах. Допустим у нас в корне дерева лежит 1000 объектов. В какой-то вершине мы делим по какому-то признаку 200 объектов и ещё в одной вершине по этому же признаку делим 50 объектов. Тогда у этого признака будет важность  $\frac{200}{1000} + \frac{50}{1000} = 0.2 + 0.05 = 0.25$ .

**Impurity-based importance** - по каждому разбиению по признаку мы ещё можем посчитать  $Q = H(R) - (H(R_l) + H(R_r))$ , а потом просуммируем  $Q_1 + Q_2 + \dots + Q_n$  для одного признака.

Это не всегда хорошо работает, потому что этот подход зависит от количества уникальных значений признаков.

Как получше померить важность признака? Давайте возьмём тот признак, важность которого хотим померить, и перемешаем значения этого признака для всех объектов и обучим модель на таких данных. Если качество модели особо не уменьшилось - значит модель не использовала этот признак.

### 12.2.1. Программируем изучение важности признаков

```
from sklearn.inspection import permutation_importance  
r = permutation_importance(rf, X_test, y_test,  
                           n_repeats=30,  
                           random_state=0)  
  
for i in r.importances_mean.argsort()[:-1]:  
    if r.importances_mean[i] - 2 * r.importances_std[i] > 0:  
        print(f"data.feature_names[{i}]: {r.importances_mean[i]:.3f}"  
              f"\u00b1{r.importances_std[i]:.3f}")
```

```
plt.figure(figsize = (10, 7))
plt.bar(data.feature_names, r.importances_mean)
```

## 13. Лекция 8. Бустинг. Лекция с ФЭН

### 13.1. Случайный лес

Возьмём в качестве базовых алгоритмом для бэггинга решающие деревья. Отличие от бэггинга будет в том, что каждой вершине дерева будем искать разбиение не по всем признакам, а по подмножеству признаков. Это нужно для некоррелированности моделей. Это даст нам уменьшение разброса.

Итоговая композиция имеет вид  $\alpha(x) = \frac{1}{n} \sum_{i=1}^n b_i(x)$ .

### 13.2. Бустинг

В среднем бустинг даёт результаты лучше, чем случайный лес и считается одной из самой лучшей моделью для табличных данных.

Тут деревья не обучаются одновременно, как в случайном лесе. Идея бустинга в том, что каждый алгоритм исправляет ошибку предыдущего.

#### 13.2.1. Частный случай бустинга

Давайте решать задачу минимизации MSE:

$$\frac{1}{l} \sum_{i=1}^l (\alpha(x_i) - y_i)^2 \rightarrow \min_{\alpha}$$

Ищем алгоритм  $\alpha(x)$  в виде суммы  $n$  базовых алгоритмов:

$$\alpha(x) = \sum_{i=1}^n b_i(x)$$

где базовые алгоритмы  $b_i(x)$  принадлежат некоторому семейству  $A$ .

#### 13.2.2. Алгоритм бустинга для MSE

1. Ищем алгоритм  $b_1(x)$ , минимизирующий ошибку:

$$b_1(x) = \arg \min_{b \in A} \frac{1}{l} \sum_{i=1}^l (b(x_i) - y_i)^2$$

Ошибка на объекте  $x$ :  $s = y - b_1(x)$

Следующий алгоритм должен настраиваться на эту ошибку, то есть целевая переменная для следующего алгоритма - это вектор ошибок  $s$  (а не исходный вектор  $y$ ).

2. Ищем алгоритм  $b_2(x)$ , настраивающийся на ошибки  $s$  первого алгоритма:

$$b_2(x) = \arg \min_{b \in A} \frac{1}{l} (b(x_i) - s_i)^2$$

3. Алгоритм  $b_3(x)$  будем выбирать так, чтобы он минимизировал ошибку предыдущей композиции (то есть  $b_1(x) + b_2(x)$ )

4. Ошибка  $s_i^{(n)} = y_i - \sum_{j=1}^n b_j(x_i) = y_i - \alpha_{n-1}(x_i)$ . Ищем алгоритм  $b_n(x)$ :

$$b_n(x) = \arg \min_{b \in A} \frac{1}{l} \sum_{i=1}^l (b(x_i) - s_i^{(n)})^2$$

Такой алгоритм называется "Градиентный бустинг". Ошибка на  $n$ -м шаге - это антиградиент функции потерь по ответу модели, вычисленный в точке ответа уже построенной композиции:

$$s_i^{(n)} = y_i - \alpha_{n-1}(x_i) = -\frac{\partial}{\partial z} \frac{1}{l} (z - y_i)^2 \Big|_{z=\alpha_{n-1}(x)}$$

Пусть  $L(y, z)$  - произвольная дифференцируемая функция потерь. Строим алгоритм  $\alpha_n(x)$  вида

$$\alpha_L(x) = \sum_{i=1}^L \gamma_i b_i(x)$$

где на  $n$ -м шаге

$$b_n(x) = \arg \min_{b \in A} \sum_{i=1}^l (b(x_i) - s_i^{(n)})^2$$

Тогда  $s_i^{(n)} = -\frac{\partial L}{\partial z}$

Коэффициент  $\gamma_n$  должен минимизировать ошибку:

$$\gamma_n = \min_{\gamma_n \in \mathbb{R}} \sum_{i=1}^l L(y_i, \alpha_{n-1}(x_i) + \gamma_n b_n(x_i))$$

### 13.3. Выбор базовых алгоритмов

Что произойдёт с предсказанием бустинга, если базовые алгоритмы слишком простые? Если взять слишком простые базовые алгоритмы, то они не могут найти зависимость в данных и ничего не получится. Бустинг будет что-то типа случайного блуждания.

Что будет, если базовые алгоритмы слишком простые? Если взять слишком сложные модели, по типу решающих деревьев с глубиной 10 это тоже плохо. Алгоритм слишком быстро переобучится. Одно дерево слишком быстро переобучается, а несколько тем более быстрее будут.

Чаще всего в качестве базовых алгоритмов используют решающие деревья.

В таком случае решающие деревья не должны быть очень маленькими, а также очень глубокими: оптимальная глубина 3-6 (зависит от задачи). Оптимальную глубину можно подбирать с помощью GridSearch'a.

### 13.4. Смещение и разброс бустинга

Смещение у бустинга будет меньше, чем у одной модели, потому что мы каждым следующим алгоритмом исправляем ошибки предыдущего.

Разброс может получиться большим, гарантий нет. Мы на каждом алгоритме всё больше и больше подстраиваемся под данные и можем переобучиться.

Для этого ещё нужно смотреть оптимальное количество операций бустинга: если в случайном лесе у нас большое кол-во деревьев не увеличивало ошибку, то здесь спокойно может.

### 13.5. Стохастический градиентный бустинг

Будем обучать базовый алгоритм  $b_n$  не по всей выборке  $X$ , а по случайному взятой подвыборке  $X^k \subset X$ . Благодаря этому:

- Снижается уровень шума в данных
- Вычисления становятся быстрее

Обычно берут  $|X^k| = \frac{1}{2}X$ .

### 13.6. Имплементации градиентного бустинга

- Xgboost
- CatBoost
- LightGBM

### 13.6.1. Xgboost

На каждом шаге градиентного бустинга решается задача

$$\sum_{i=1}^l (b(x_i) - s_i)^2 \rightarrow \min \iff \sum_{i=1}^l \left( -s_i b(x_i) + \frac{1}{2} b^2(x_i) \right)^2 \rightarrow \min_b$$

На каждом шаге xgboost решается задача

$$\sum_{i=1}^l \left( -s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^J b_j^2 \rightarrow \min_b \quad (*)$$

где  $h_i = \frac{\partial^2 L}{\partial z^2} \Big|_{a_{n-1}(x_i)}$ ,  $J$  - количество листов, второе слагаемое - регуляризатор по кол-ву листьев, а третье слагаемое - регуляризатор по значению (норма коэффициентов).

#### 13.6.1.1. Особенности xgboost

- Базовый алгоритм приближает направление, посчитанное с учетом второй производной функции потерь
- Функционал регуляризуется - добавляются штрафы за количество листьев и за норму коэффициентов
- При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига
- критерий останова при обучении дерева также зависит от оптимального сдвига

### 13.6.2. CatBoost

CatBoost - алгоритм, разработанный в Яндексе. Он является оптимизацией xgboost и в отличие от xgboost умеет обрабатывать категориальные признаки.

#### 13.6.2.1. Особенности CatBoost

- Используются симметричные деревья решений. Полные деревья. Если дерево не ограничивать по структуре - оно будет сильно меняться при изменении выборки. Но здесь мы зафиксировали структуру - это всегда будет полное дерево, где на одном уровне дерева будут одни и те же предикаты. Это позволяет дереву быть более устойчивым к переобучению. Выразительная способность у таких деревьев куда хуже - надо брать деревья глубины 12 и больше.
- Для кодирования категориальных признаков используется набор методов (one-hot encoding, счётчики, комбинации признаков и другие). На каждой итерации обучения алгоритмов выбирается наилучшее кодирование.
- Динамический бустинг (ordering boosting). На этапе построения предсказаний в листьях происходит следующие действия:
  - Упорядочиваются объекты в листе
  - Для предсказания  $y_i$  у нас ответ будет  $\frac{1}{i} \sum_{j=0}^i y_j$ .
- Поддержка пропусков в данных
- Обучается быстрее, чем xgboost
- Показывает хороший результат даже без подбора параметров
- Удобные методы: проверка на переобученность, вычисление значений метрик, удобная кросс-валидация и другие

### 13.6.3. LightGBM

- В других реализациях бустинга деревья строятся по уровням (level-wise tree growth). В LightGBM деревья строятся полиствено (leaf-wise tree growth). То есть на каждом шаге будет добавляться лист, который улучшает точность модели.
- LightGBM разбивает значения категориального признака на два подмножества в каждой вершине дерева, находя при этом лучшее разбиение. Если категориальный признак имеет  $k$  различных значений, то возможных разбиений  $2^{k-1} - 1$ . В LightGBM реализован способ поиска оптимального разбиения за  $O(k \log(k))$  операций.
- Ускорение построения деревьев за счёт бинаризации признаков: например из признаков  $(2,3,5,9,11,12,16)$  сделать признаки  $(1,1,1,1,2,2,2)$  - похожие значения собираем в одно.

## 14. Лекция 8. Бустинги. Наша лекция

В классическом бустинге используется квадратичная функция потерь, потому что это удобно: функция выпуклая, с одним минимумом и дифференцируемая.

### 14.1. Программируем

#### 14.1.1. Устанавливаем библиотеки

```
!pip install catboost==1.0.3
!pip install lightgbm==3.2.1
!pip install cmake==3.22.0 # without it xgboost will not import
!pip install xgboost==1.5.0
```

#### 14.1.2. Импорт библиотек

```
import warnings
warnings.filterwarnings('ignore')

import catboost
import lightgbm
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import xgboost

plt.rcParams["figure.figsize"] = (8, 5)
```

#### 14.1.3. Генерация датасета и функция для визуализации решений дерева

```
def plot_surface(X, y, clf):
    h = 0.2
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    #
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())

X, y = make_classification(n_samples=500, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0,
                           n_classes=2, n_clusters_per_class=2,
                           flip_y=0.05, class_sep=0.8, random_state=241)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=241)
```

#### 14.1.4. Отобразим визуализацию дерева для CatBoost и вычислим ROC-AUC

Параметр `n_estimators` у CatBoost - сколько мы хотим деревьев.

```
from catboost import CatBoostClassifier

catboost = CatBoostClassifier(n_estimators=300, logging_level='Silent')
catboost.fit(X_train, y_train)
plot_surface(X_test, y_test, catboost)

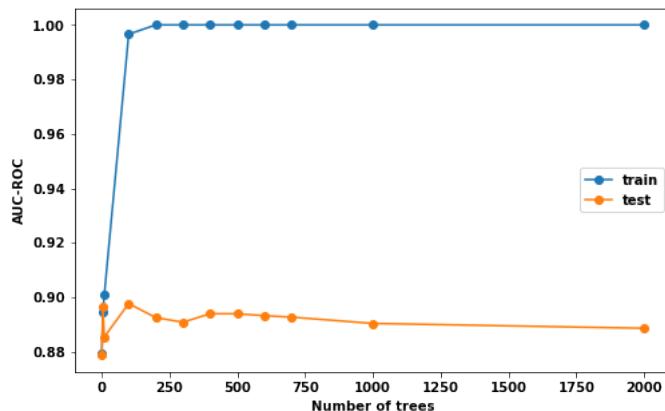
print(roc_auc_score(y_test, catboost.predict_proba(X_test)[:, 1]))
```

#### 14.1.5. Обучаем обычный градиентный бустинг и анализируем качество от количества деревьев

```
from sklearn.ensemble import GradientBoostingClassifier

n_trees = [1, 5, 10, 100, 200, 300, 400, 500, 600, 700, 1000, 2000]
quals_train = []
quals_test = []
for n in n_trees:
    gbc = GradientBoostingClassifier(n_estimators=n)
    gbc.fit(X_train, y_train)
    q_train = roc_auc_score(y_train, gbc.predict_proba(X_train)[:, 1])
    q_test = roc_auc_score(y_test, gbc.predict_proba(X_test)[:, 1])
    quals_train.append(q_train)
    quals_test.append(q_test)

plt.plot(n_trees, quals_train, marker='o', label='train')
plt.plot(n_trees, quals_test, marker='o', label='test')
plt.xlabel('Number of trees')
plt.ylabel('AUC-ROC')
plt.legend()
plt.show()
```



#### 14.1.6. Сделаем то же самое для CatBoost

В алгоритме сделаны улучшения и выбор разных опций для борьбы с переобучением, подсчету среднего таргета на отложенной выборке, подсчету статистик по категориальным фичам, бинаризацией фичей, рандомизации скора сплита, разные типы бутстрэпирования.

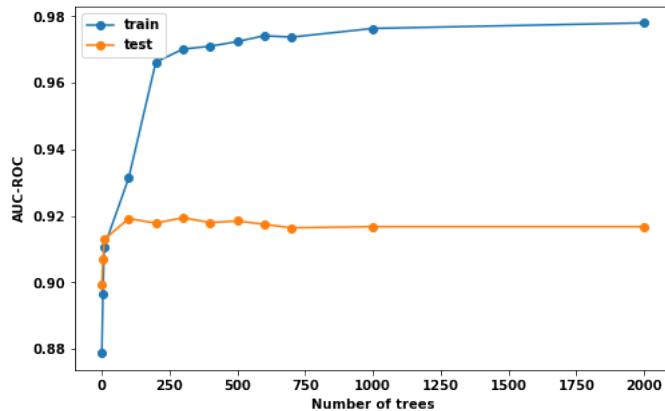
```
n_trees = [1, 5, 10, 100, 200, 300, 400, 500, 600, 700, 1000, 2000]
quals_train = []
quals_test = []
for n in n_trees:
    catboost = CatBoostClassifier(iterations=n, logging_level='Silent')
    catboost.fit(X_train, y_train)
```

```

q_train = roc_auc_score(y_train, catboost.predict_proba(X_train)[:, 1])
q_test = roc_auc_score(y_test, catboost.predict_proba(X_test)[:, 1])
quals_train.append(q_train)
quals_test.append(q_test)

plt.plot(n_trees, quals_train, marker='o', label='train')
plt.plot(n_trees, quals_test, marker='o', label='test')
plt.xlabel('Number_of_trees')
plt.ylabel('AUC-ROC')
plt.legend()
plt.show()

```



#### 14.1.7. Сравнение GradientBoostingClassifier и CatBoost

Разница между графиками

- У CatBoost график выходит на плато и нет переобучения при увеличении числа деревьев.
- Качество у CatBoost на train данных не доходит до единицы - мы не подстраиваемся под данные.
- Метрика ROC-AUC в целом выше.

#### 14.1.8. Решение XGBoost

- Базовый алгоритм приближает направление, посчитанное с учетом второй производной функции потерь
- Функционал регуляризуется – добавляются штрафы за количество листьев и за норму коэффициентов
- При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига
- Критерий останова при обучении дерева также зависит от оптимального сдвига

Ссылка на источник: <https://github.com/esokolov/ml-course-hse/blob/master/2021-fall/lecture-notes/lecture11-ensembles.pdf>

```

n_trees = [1, 5, 10, 100, 200, 300, 400, 500, 600, 700, 1000, 2000]
quals_train = []
quals_test = []
for n in n_trees:
    xgboost = XGBClassifier(n_estimators=n, verbosity=0)
    xgboost.fit(X_train, y_train)
    q_train = roc_auc_score(y_train, xgboost.predict_proba(X_train)[:, 1])
    q_test = roc_auc_score(y_test, xgboost.predict_proba(X_test)[:, 1])
    quals_train.append(q_train)
    quals_test.append(q_test)

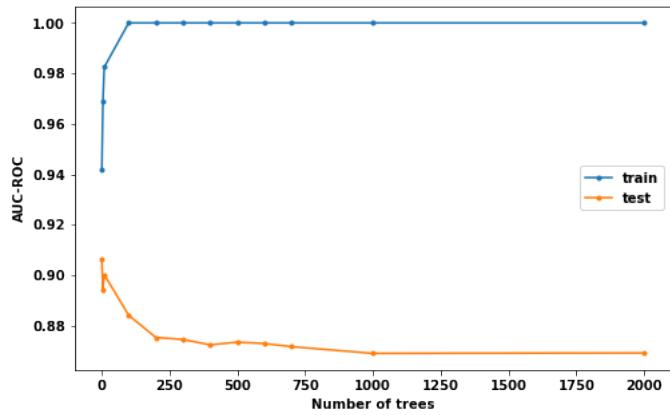
```

```

plt.figure(figsize=(8, 5))
plt.plot(n_trees, quals_train, marker='.', label='train')
plt.plot(n_trees, quals_test, marker='.', label='test')
plt.xlabel('Number of trees')
plt.ylabel('AUC-ROC')
plt.legend()

plt.show()

```



Видно, что на тесте качество от количества деревьев только ухудшается, идёт жёсткое переобучение. Если параметры не настраивать - XGBoost переобучился даже больше, чем обычный бустинг.

## 14.2. CatBoost для решения задачи + интерпритация признаков

### 14.2.1. Загрузка данных и разбитие для обучения и теста

```

data = pd.read_csv("bike_buyers_clean.csv")

X = data.drop(['ID', 'Purchased_Bike'], axis=1)
y = data['Purchased_Bike']

from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.5, random_state=42)

```

### 14.2.2. Обучим модель

```

from catboost import CatBoostClassifier

model1 = CatBoostClassifier(cat_features = [0, 1, 4, 5, 6, 8, 9]), # one_hot_max_size=10
model1.fit(Xtrain, ytrain)

```

### 14.2.3. Построим предсказание

```

from sklearn.metrics import accuracy_score

pred = model1.predict(Xtest)

accuracy_score(ytest, pred)

```

### 14.2.4. Переберём параметры

- max\_depth - глубина деревьев
- learning\_rate - с каким коэффициентом прибавляются модели

- ohe\_hot\_max\_size - если модель решит, что ОНЕ лучше кодировать, модель может сделать очень много столбцов и переобучить модель или сделать дольше обучение. Этим параметром мы говорим, что мы не хотим получать больше сколько-то столбцов.

```
from sklearn.model_selection import GridSearchCV

params = { 'max_depth' : np.arange(10,20,3),
           'learning_rate' : [0.01, 0.05, 0.1],
           'one_hot_max_size' : [10, 50, 100]}

gs = GridSearchCV(CatBoostClassifier(n_estimators=100, cat_features = [0, 1, 4, 5, 6, 8, 9]), params)
gs.fit(Xtrain, ytrain)

print(gs.best_score_, gs.best_params_)
```

Вообще перебирать параметры так долго и в дальнейшем будет лекция по auto-ml, где будет говориться о том, как перебирать лучше.

### 14.3. Оценка важности признаков

Статья - <https://towardsdatascience.com/deep-dive-into-catboost-functionalities-for-model-interpretation>

#### 14.3.1. Вспомогательные функции для оценки

model1 - модель CatBoost

```
from sklearn.metrics import log_loss

def logloss(model, X, y):
    return log_loss(y, model.predict_proba(X)[:,1])

def permutation_importances(model, X, y, metric):
    baseline = metric(model, X, y)
    imp = []
    for col in X.columns:
        save = X[col].copy()
        X[col] = np.random.permutation(X[col])
        m = metric(model, X, y)
        X[col] = save
        imp.append(m-baseline)
    return np.array(imp)
```

#### 14.3.2. Визуализация важности признаков

```
fi_1 = model1.feature_importances_

feature_score = pd.DataFrame(list(zip(Xtest.dtypes.index, fi_1)),
                               columns=['Feature', 'Score'])

feature_score = feature_score.sort_values(by='Score', ascending=False, inplace=False, kind='quicksort')

plt.rcParams["figure.figsize"] = (12,7)
ax = feature_score.plot('Feature', 'Score', kind='bar', color='c')
ax.set_title("Feature_Importance_using_{}".format('Standard_Importances'), fontsize = 14)
ax.set_xlabel("features")
plt.show()
```

### 14.3.3. Визуализация важности признаков при случайной перестановке

```
fi_2 = permutation_importances(model1, Xtest, ytest, logloss)

feature_score = pd.DataFrame(list(zip(Xtest.dtypes.index, fi_2)),
                             columns=['Feature', 'Score'])

feature_score = feature_score.sort_values(by='Score', ascending=False, inplace=False, kind='quicksort')

plt.rcParams["figure.figsize"] = (12,7)
ax = feature_score.plot('Feature', 'Score', kind='bar', color='c')
ax.set_title("Feature_Importance_using_{}".format('Permutation_Importances'), fontsize=14)
ax.set_xlabel("features")
plt.show()
```

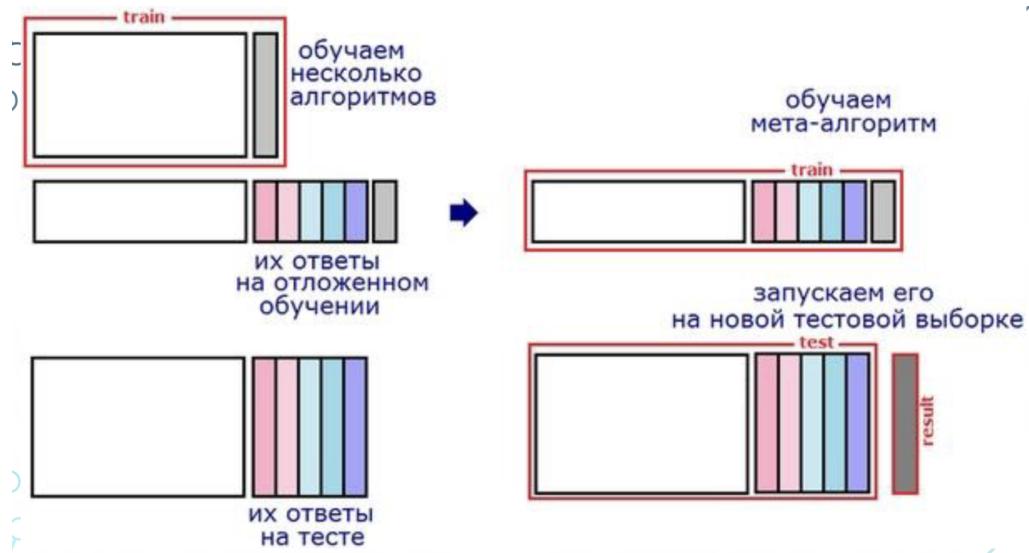
## 14.4. Блендинг и стекинг

### 14.4.1. Стекинг

Идея: обучаем несколько **разных** алгоритмов и передаём их результаты на вход последнему, который принимает итоговые решения. Нужно чтобы все базовые алгоритмы были разными, потому что в другом случае стекинг прироста в качестве не даст.

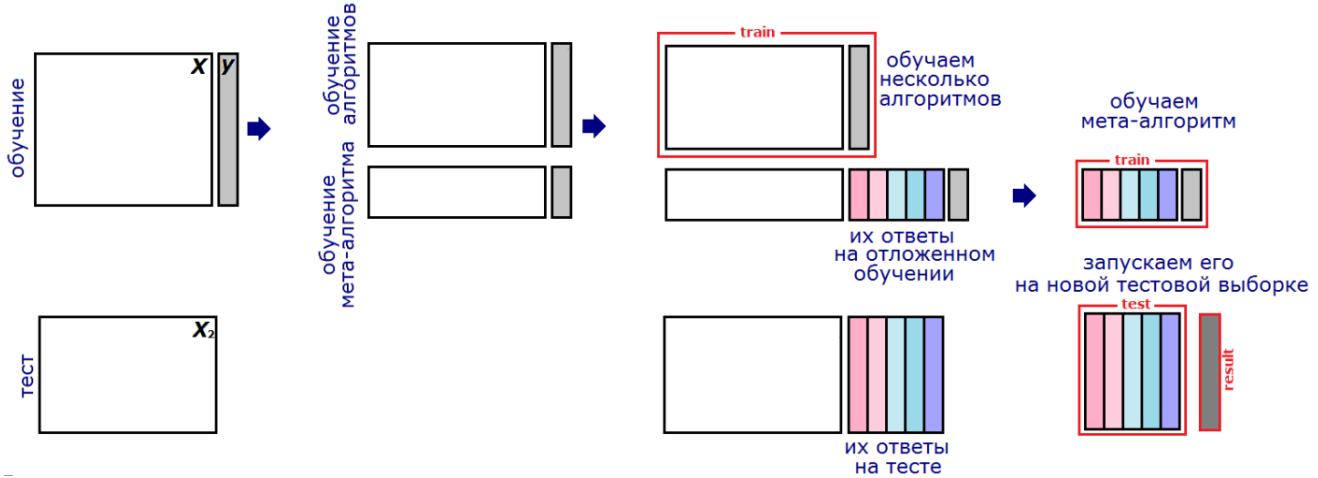
Итоговый алгоритм, который из пресказаний других делает одно, называется мета-алгоритм.

Особенностью стекинга является то, что нужно обучать базовые алгоритмы и мета-алгоритм на разных фолдах.



## 14.5. Блендинг

Блендинг - это частный случай стекинга, в котором мета-алгоритм линеен:



Но при этом, если мы делаем стекинг или блендинг, то у нас страдает интерпретация модели. Если мы преследуем цель просто поднять качество - то вполне рабочий вариант.

## 14.6. Программируем блендинг

### 14.6.1. Загружаем данные

```
from sklearn.datasets import load_boston

data = load_boston()
X_init = pd.DataFrame(data.data, columns=data.feature_names)
y_init = data.target

X, X_test, y, y_test = train_test_split(X_init, y_init, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=42)

assert X_init.shape[0] == X_train.shape[0] + X_val.shape[0] + X_test.shape[0]

def rmse(y_true, y_pred):
    error = (y_true - y_pred) ** 2
    return np.sqrt(np.mean(error))
```

### 14.6.2. Проверка качества алгоритмов, если просто обучить на train и проверить на test и блендинг на глазок

```
from catboost import CatBoostRegressor
from sklearn.linear_model import LinearRegression

cbm = CatBoostRegressor(iterations=100, max_depth=4, learning_rate=0.01, loss_function='RMSE')
cbm.fit(X_train, y_train)
test_pred_cbm = cbm.predict(X_test)

lr = LinearRegression()
lr.fit(X_train, y_train)
test_pred_lr = lr.predict(X_test)

print("Test_RMSE_Linear_Regression=% .3f" % rmse(y_test, test_pred_lr))
print("Test_RMSE_Catboost=% .3f" % rmse(y_test, test_pred_cbm))

pred = 0.7 * test_pred_lr + 0.3 * test_pred_cbm

print("Test_RMSE_mix=% .3f" % rmse(y_test, pred))
```

### 14.6.3. Простой блендинг

Но тут у нас происходит обучение по тесту, это плохо.

### 14.6.4. Подбор весов

Представим новый алгоритм  $\alpha(x)$  как взвешенную сумму из базовых алгоритмов:

$$\alpha(x) = \sum_{i=1}^n w_i \cdot b_i(x)$$

Вот эти самые  $w_i$  нам и нужно подобрать.

Сначала рассмотрим более простой случай подбора весов, методом перебора  $w_1$  и получения  $w_2 = 1 - w_1$  (так как у нас всего два алгоритма).

Будем подбирать веса на валидации, а проверять качество на тесте.

```
def select_weights(y_true, y_pred_1, y_pred_2):
    grid = np.linspace(0, 1, 1000)
    metric = []
    for w_0 in grid:
        w_1 = 1 - w_0
        y_a = w_0 * y_pred_1 + w_1 * y_pred_2
        metric.append([rmse(y_true, y_a), w_0, w_1])
    return metric

val_pred_cbm = cbm.predict(X_val)
val_pred_lr = lr.predict(X_val)

rmse_blending_train, w_0, w_1 = min(select_weights(y_val, val_pred_cbm, val_pred_lr),
                                     key=lambda x: x[0])
rmse_blending_train, w_0, w_1
```

Получилось получше, чем без блендинга, но давайте проведём его полноценно

## 14.7. Полноценный блендинг

### 14.7.1. Загрузка данных

```
data = load_boston()
X_init = pd.DataFrame(data.data, columns=data.feature_names)
y_init = data.target

X, X_test, y, y_test = train_test_split(X_init, y_init, test_size=0.3, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

assert X_init.shape[0] == X_train.shape[0] + X_val.shape[0] + X_test.shape[0]
```

### 14.7.2. Использование блендинга

```
from lightgbm import LGBMRegressor

gb = CatBoostRegressor(iterations=100, max_depth=4, learning_rate=0.01, loss_function='RMSE')
gb.fit(X_train, y_train)

lr = LinearRegression()
lr.fit(X_train, y_train)

meta_train_df = pd.DataFrame()
meta_train_df['gb_preds'] = gb.predict(X_val)
meta_train_df['lr_preds'] = lr.predict(X_val)
```

```

meta_algo = LGBMRegressor()
meta_algo.fit(meta_train_df, y_val)

meta_pred_df = pd.DataFrame()
meta_pred_df[ 'gb_preds' ] = gb.predict(X_test)
meta_pred_df[ 'lr_preds' ] = lr.predict(X_test)
test_preds_meta = meta_algo.predict(meta_pred_df)

rmse(y_test, test_preds_meta)

```

Получается, что при блендинге базовые алгоритмы и мета-алгоритм не используют весь объем выборки обучения, что является недостатком. Для повышения качества нужно усреднять несколько блендигов.

## 14.8. Используем стекинг

Попробуем реализовать стэкинг. Выборку разбивают на два фолда, последовательно перебирая фолды, обучаю базовые алгоритмы на всех фолдах, кроме одного, а на оставшемся получают ответы базовых алгоритмов и используют их как значения соответствующих признаков на этом фолде. Для получения мета-признаков объектов тестовой выборки базовые алгоритмы обучаю на всей обучающей выборке и берут их ответы на тестовой.

В estimators лежат наши базовые модели

```

from sklearn.ensemble import StackingRegressor, RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier

estimators = [ ('rf', RandomForestRegressor(n_estimators=200, random_state=42)),
               ('lr', LinearRegression()),
               ('knn', KNeighborsClassifier(n_neighbors=10))]

reg = StackingRegressor(estimators=estimators,
                        cv=10,
                        final_estimator=CatBoostRegressor(iterations=700, max_depth=5, learning_function='RMSE', logging_level='Silent'))

reg.fit(X_train, y_train).score(X_test, y_test)
reg_preds = reg.predict(X_test)
round(rmse(y_test, reg_preds), 3)

```

Получилось качество, куда лучшее предыдущих наших попыток.

## 15. Материалы к лекции 9. Многоклассовая классификация. Уроки со stepik

### 15.1. Многоклассовая и multi-label классификация

Важное замечание, что многоклассовая классификация на английский переводится как multiclass classification.

Примеров таких задач множество:

- задача определения класса риска клиентов в скоринге
- задача определения объекта на изображении
- задача определения тональности отзыва и многие другие

Отметим, что в английском языке есть ещё один термин для многоклассовой классификации: это multilabel классификация. Multiclass и multilabel классификации - это задачи разных типов.

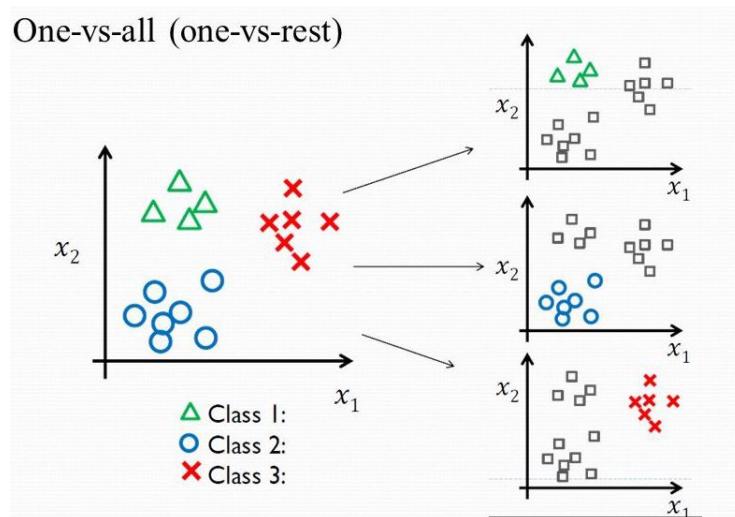
- Multiclass-классификация: задача, в которой целевая переменная - это один из нескольких классов (где классов может быть больше, чем два)
- Multilabel-классификация: задача, в которой для каждого объекта предсказывается несколько меток (например, для фильма одновременно можно предсказать его жанр, год выпуска, язык фильма и так далее). Или же на картинке может быть изображен не один объект, а несколько - и наша задача определить классы всех объектов.

Multilabel-задачу можно свести к серии multiclass-задач.

#### 15.1.1. Сведение к бинарной классификации

Задачи многоклассовой (multiclass) классификации можно свести к серии бинарных задач. Два основных подхода называются one-versus-one (OVO) и one-versus-rest (OVR).

**15.1.1.1. One-Versus-Rest** Этот подход сводит задачу к серии бинарных классификаторов, где  $i$ -й классификатор определяет, относится объект к классу  $i$  или не относится (два варианта, то есть бинарный классификатор).



В этом подходе обучается столько бинарных классификаторов, сколько классов в исходной multiclass-задаче.

**15.1.1.2. One-Vs-One** В этом подходе каждый бинарный классификатор пытается отличить, принадлежит объект классу  $i$  или классу  $j$ . Каждый такой классификатор обучается только на объектах  $i$ -го и  $j$ -го классов.

Сравнительная картина подходов выглядит так:

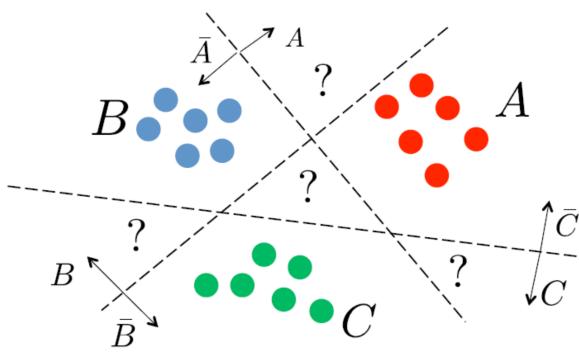


Рис. 2: One-Versus-Rest

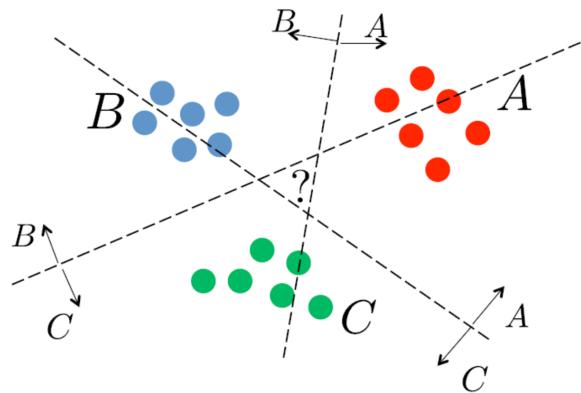


Рис. 3: One-Versus-One

Нельзя в общем случае сказать, какой из подходов лучше. Однако у каждого из подходов есть особенности, связанные со скоростью обучения:

- Если мы выбираем подход one-versus-all, то при большом количестве объектов в данных обучение будет долгим, ведь нужно обучить несколько бинарных классификаторов на большом датасете. В случае one-versus-one каждый бинарный классификатор (с номером  $ij$ ) обучается только на части датасета, состоящей из классов  $i$  и  $j$ .
- Если же классов в задаче много, то подход one-versus-one будет работать долго, так как он сводится к обучению бинарных классификаторов на каждой паре классов, то есть, если в задаче  $N$  классов, то необходимо обучить  $\frac{N \cdot (N - 1)}{2}$  бинарных классификаторов.

## 15.2. Метрики качества многоклассовой классификации

Для бинарной классификации мы использовали такие метрики как accuracy, precision, recall, f1-score, confusion matrix, roc-auc, pr-auc. Многие из них обобщаются на многоклассовый случай. Давайте поговорим как.

### 15.2.1. Accuracy

Метрика accuracy - это доля правильных ответов модели, она без изменений в формуле может применяться для любого количества классов.

### 15.2.2. Recall и f1-score

Теперь матрица ошибок будет не  $2 \times 2$ , а  $N \times N$ , где  $N$  - количество классов. Пример - матрица классификации животных:

		True/Actual		
		Cat (🐱)	Fish (🐠)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐠)	1	2	0
	Hen (🐔)	1	2	6

Для вычисления точности и полноты в этом случае существует несколько подходов:

- Микроусреднение (micro-average)
- Макроусреднение (macro-average)
- Взвешенное усреднение (weighted-average)

**15.2.2.1. Макроусреднение (macro-average)** В этом подходе мы вычисляем значение выбранной метрики для каждой бинарной ситуации (кошка/не кошка, рыба/не рыба, курица/не курица), а затем усредняем полученные числа.

Например, посчитаем точность и полноту для ситуации кошка/не кошка:

		True/Actual		
		Cat (img alt="Cat icon" data-bbox="448 161 478 178")	Fish (img alt="Fish icon" data-bbox="518 161 548 178")	Hen (img alt="Hen icon" data-bbox="648 161 678 178")
Predicted	Cat (img alt="Cat icon" data-bbox="308 188 338 205")	4	6	3
	Fish (img alt="Fish icon" data-bbox="308 218 338 235")	1	2	0
	Hen (img alt="Hen icon" data-bbox="308 248 338 265")	1	2	6

$$\text{Тогда } precision(cat) = \frac{TP}{TP+FP} = \frac{4}{4+6+3} = \frac{4}{13}$$

То есть false positive - это все объекты, которые модель ошибочно назвала кошкой (их  $6 + 3$ ).

$$recall(cat) = \frac{TP}{TP+FN} = \frac{4}{4+1+1} = \frac{4}{6}$$

Здесь false negative - это все кошки, которых модель не нашла (кошки, названные моделью не кошками).

Тогда macro-average считается как

$$precision = \frac{precision(cat) + precision(fish) + precision(hen)}{3}$$

Аналогично вычисляется macro-average recall.

**15.2.2.2. Взвешенное усреднение (weighted-average)** В этом подходе мы усредняем посчитанные для каждого класса метрики с весами, пропорциональными количеству объектов класса.

То есть weighted average

$$precision = \frac{6}{25} \cdot precision(cat) + \frac{10}{25} \cdot precision(fish) + \frac{9}{25} \cdot precision(hen)$$

**15.2.2.3. Микроусреднение (micro-average)** В этом подходе мы вычисляем значения TP, TN, FP, FN по всей матрице ошибок сразу, исходя из их определения. Затем по полученным числам вычисляем выбранные метрики.

TP - это количество верно угаданных объектов положительного класса. В нашем случае  $TP = 4+2+6 = 12$

FP - это суммарное количество false positive-предсказаний. Например, если cat предсказана как fish, то это false positive для fish. Таким образом, FP - это сумма всех неверных предсказаний, то есть  $FP = 6 + 3 + 1 + 0 + 1 + 2 = 13$

Получаем micro-average

$$precision = \frac{TP}{TP+FP} = \frac{12}{12+13} = \frac{12}{25}$$

Но для recall FN - это сумма false negative-предсказаний. Например, если cat предсказана как fish, то это false negative для cat. Таким образом, FN - это опять же сумма всех неверных предсказаний, то есть  $FN = 6 + 3 + 1 + 0 + 1 + 2 = 13$

Получается, что в случае микро-усреднения  $precision = recall$ .

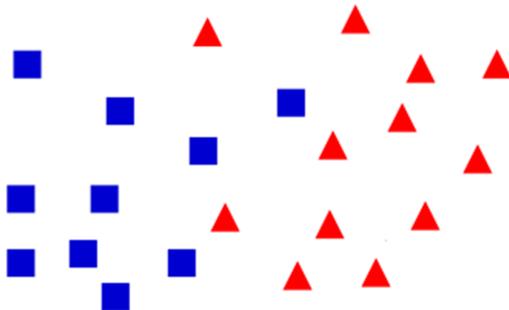
И так как f1-score - это среднее гармоническое точности и полноты, то при микроусреднении  $precision = recall = f1 - score$

### 15.2.3. Результат в python

В python можно получать все эти метрики одной функцией, она называется `sklearn.metrics.classification_report`.

### 15.3. Метод ближайших соседей (KNN)

Идея метода очень простая, она называется **гипотезой компактности**: схожие объекты находятся близко друг к другу в пространстве признаков.



На рисунке изображены объекты двух классов. Видно, что все квадратики находятся относительно близко друг к другу и далеко от треугольников. Треугольники в свою очередь в основном находятся близко друг к другу.

#### 15.3.1. Алгоритм метода

У метода есть гиперпараметр  $k$  - число соседей.

Чтобы определить, к какому классу относится объект, нужно:

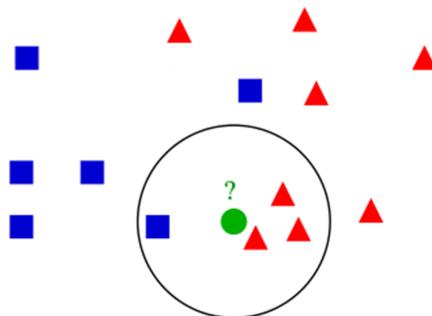
- вычислить расстояние от объекта до каждого объекта выборки
- выбрать  $k$  объектов выборки с наименьшим расстоянием ( $k$  ближайших соседей)
- класс искомого объекта - это наиболее часто встречающийся класс среди  $k$  ближайших соседей

Формально последний пункт записывается так: если  $Y$  - множество всех возможных классов в задаче, а  $y_i$  - класс  $i$ -го объекта из найденных  $k$  ближайших объектов, то предсказание модели на объекте  $q$

$$\alpha(q) = \arg \max_{y \in Y} \sum_{i=1}^k I[y_i = y]$$

Сумма берется по  $k$  ближайшим к объекту  $q$  соседям.

Например, при  $k = 4$  для следующей картинки

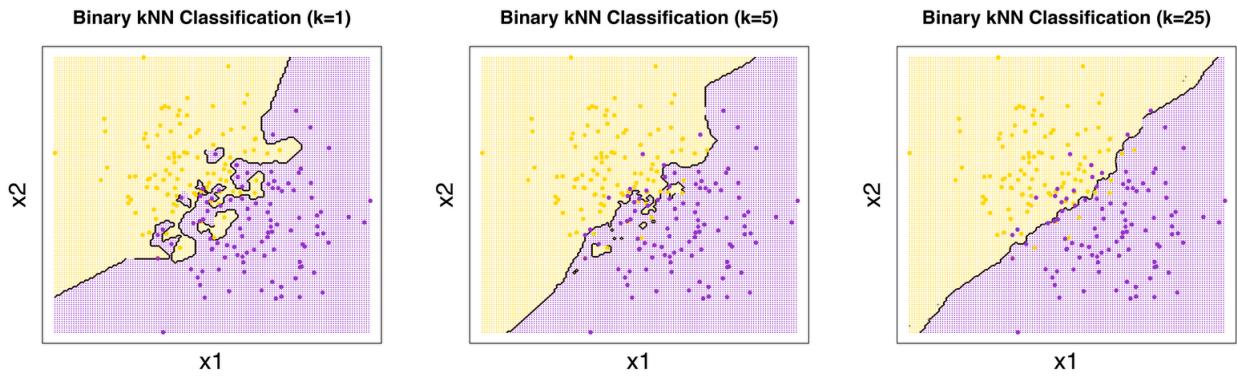


искомый объект будет отнесен к классу треугольников.

#### 15.3.2. Влияние гиперпараметра $k$

Алгоритм определения классов очень простой. На самом деле у метода нет фазы обучения, потому что нет параметров, которые подбираются в процессе обучения по выборке. В этом смысле метод очень простой.

Несмотря на свою простоту, метод легко переобучается. Например, если взять число соседей очень маленьким ( $k = 2$  или  $k = 3$ ), метод будет делать предсказания только по двум или трем самыми близкими точкам, и поэтому сильно подгонится под данные. В этом можно убедиться, посмотрев на разделяющую поверхность метода при разных значениях  $k$ :



Разделяющая поверхность при маленьких значениях  $k$  очень сложная, и это означает, что модель подстраивается под данные, что ведёт к сильному переобучению.

### 15.3.3. Выбор метрики

В методе ближайших соседей мы вычисляем расстояния между объектами. Способов вычислить расстояние очень много, каждый из них задается своей формулой (метрикой). Каждая метрика больше подходит для своего типа задач.

Наиболее используемые метрики:

**15.3.3.1. Евклидова метрика** Евклидова метрика - классический способ измерить расстояние между объектами. Пусть объекты  $a$  и  $b$  имеют координаты  $a = (x_1, y_1)$ ,  $b = (x_2, y_2)$ . Тогда евклидово расстояние между ними

$$\rho(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

**15.3.3.2. Манхэттенское расстояние** Манхэттенское расстояние - другой способ посчитать расстояние между двумя точками:

$$\rho(a, b) = |x_1 - x_2| + |y_1 - y_2|$$

**15.3.3.3. Расстояние Хемминга** Расстояние Хемминга - это число различных позиций в координатах двух векторов

Пусть есть вектор  $A = \{0, 1\}^n$  и  $B = \{0, 1\}^n$ , то расстояние хэмминга считается как

$$\rho(a, b) = \sum_{i=1}^n [a_i \neq b_i]$$

**15.3.3.4. Мера Жаккара** Мера Жаккара - ещё один способ измерить расстояние (часто используется для измерения похожести текстов). Пусть  $a$  и  $b$  некоторые множества, тогда мера Жаккара - это

$$\rho(A, B) = \frac{A \cap B}{A \cup B}$$

то есть отношение числа совпадающих элементов множеств к общему числу элементов.

Существует множество других метрик. Выбор метрики зависит от задачи и свойств объектов в задаче.

### 15.3.4. Масштабирование данных для KNN

При использовании KNN в ситуации, когда объекты описываются вектором из числовых признаков необходимо масштабировать данные. Почему так?

Пусть мы ищем ближайшие объекты к объекту  $a = (40, 1, 100000)$ , где 40 - возраст человека, 1 - пол (1 - мужчина, 0 - женщина), 100000 - месячная зарплата.

Какой объект будет ближайшим к объекту  $a$  по евклидовой метрике?

- $b = (80, 0, 90000)$
- $c = (38, 0, 50000)$
- $d = (25, 1, 10000000)$

Если посчитать расстояния между объектами  $a$  и  $b$ , затем между  $a$  и  $c$  и, наконец, между  $a$  и  $d$ , то наименьшим будет расстояние между  $a$  и  $b$ :

$$\rho(a, b) = \sqrt{(80 - 40)^2 + (0 - 1)^2 + (90000 - 100000)^2} = \sqrt{1600 + 1 + 100000000} \sim 10000$$

Мы видим, что наибольший вклад в ответ вносит зарплата человека, а остальные признаки по большому счету не важны. А это не всегда так, ведь это зависит от задачи и от многих других факторов.

Получилось, что мужчина 40 лет больше всего похож на женщину 80 лет (из предложенных вариантов) просто потому, что у них похожие зарплаты. Так быть не должно!

Поэтому перед применением KNN необходимо привести данные к одному масштабу!

### 15.3.5. Обобщения KNN

У классического KNN есть один большой недостаток - он никак не учитывает расстояния до ближайших объектов. В то же время понятно, что объекты, находящиеся ближе к объекту запроса, должны вносить больший вклад в ответ. Также учёт расстояний будет очень важен, когда мы будем применять KNN для решения задач регрессии.

Как можно учесть расстояния?

Напомним, что классический KNN определяет класс объекта  $q$  как самый популярный класс среди  $k$  его ближайших соседей:

$$\alpha(q) = \arg \max_{y \in Y} \sum_{i=1}^k I[y_i = y]$$

то есть все ближайшие соседи входят в формулу с весом 1. Можно изменить вес в зависимости от объекта.

можно упорядочить соседей по увеличению расстояния от объекта запроса  $qq$  и давать им вес, обратно пропорциональный номеру соседа:  $w_k = \frac{1}{k}$ , то есть

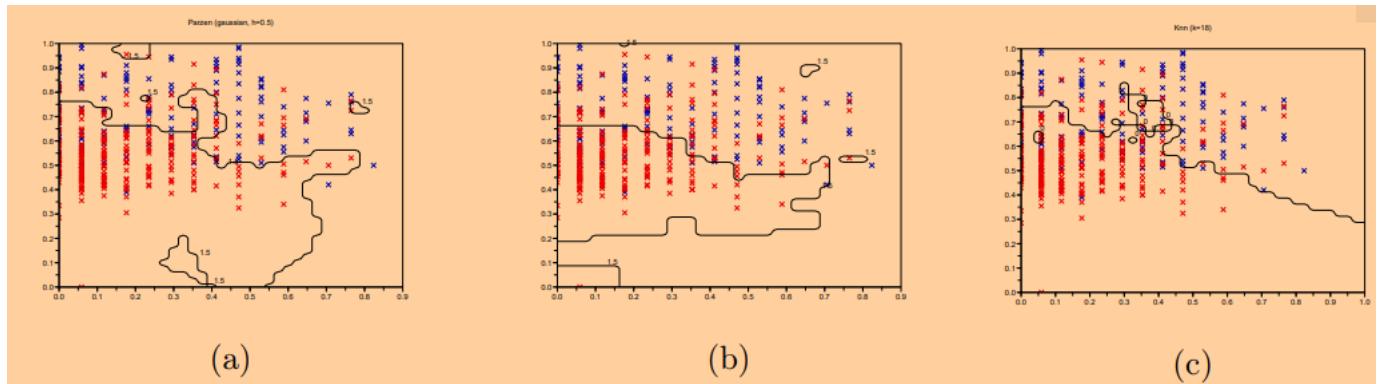
$$\alpha(q) = \arg \max_{y \in Y} \sum_{i=1}^k \frac{I[y_i = y]}{k}$$

Такой подход всё ещё не учитывает реальные расстояния от объекта запроса до ближайших объектов. Способ учесть расстояния называется методом Парзеновского окна:

$$\alpha(q) = \arg \max_{y \in Y} \sum_{i=1}^k K\left(\frac{\rho(q, x_i)}{h}\right) I[y_i = y]$$

где

- $q$  - вектор признаков запроса
- $x_i$  - вектор признаков  $i$ -го ближайшего соседа
- $h$  - ширина окна
- Функция  $K(x)$ , называемая ядром. Существует множество различных ядер, например:
  - $K(x) = \frac{1}{2}I[|x| \leq 1]$  (прямоугольное ядро)
  - $K(x) = (1 - |x|) \cdot I[|X| \leq 1]$  (треугольное ядро)
  - $K(x) = \frac{1}{\sqrt{2\pi}}e^{-2x^2}$  (гауссовское ядро)



На рисунке изображены результаты использования KNN для решения задачи бинарной классификации:

- (a) гауссово ядро,  $h = 0.5$
- (b) кубическое ядро,  $h = 0.2$
- (c) классический KNN с  $k = 18$  соседями

Видно, что использование ядер сильно влияет на результат классификации.

На практике, однако, чаще всего используется прямоугольное ядро для простоты. То есть, например, для окна ширины  $h = 1$  веса объектов просто равны  $w_i = \frac{1}{2}$ , если расстояние между объектами  $q$  и  $x_i$  не превосходит 1, и 0 иначе.

### 15.3.6. KNN в задачах регрессии

С помощью KNN можно решать не только задачи классификации, но и задачи регрессии. Существует как простой, так и более сложные подходы - все они полностью аналогичны подходам в задачах классификации:

Простой вариант:

$$\alpha(q) = \frac{1}{k} \sum_{i=1}^k y_i$$

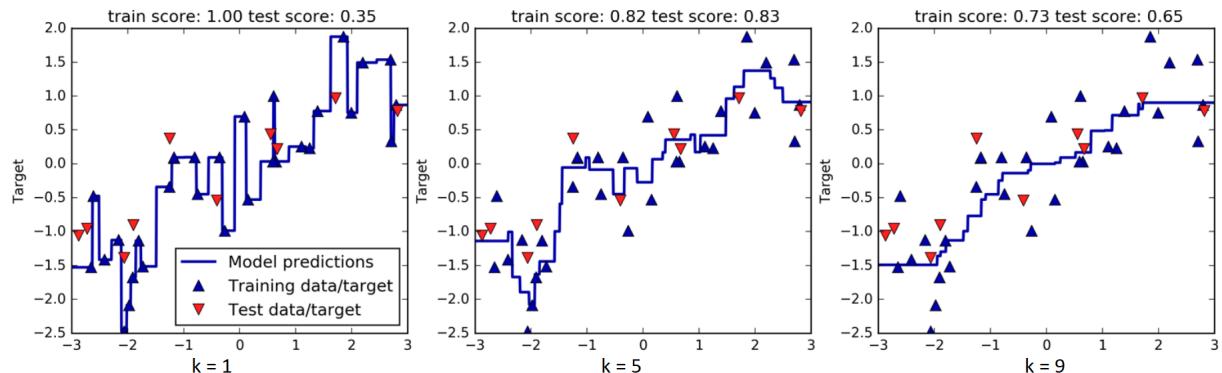
То есть предсказанный ответ это просто среднее арифметическое целевых переменных  $k$  соседей.

Взвешенный вариант (формула Надарай-Батсона):

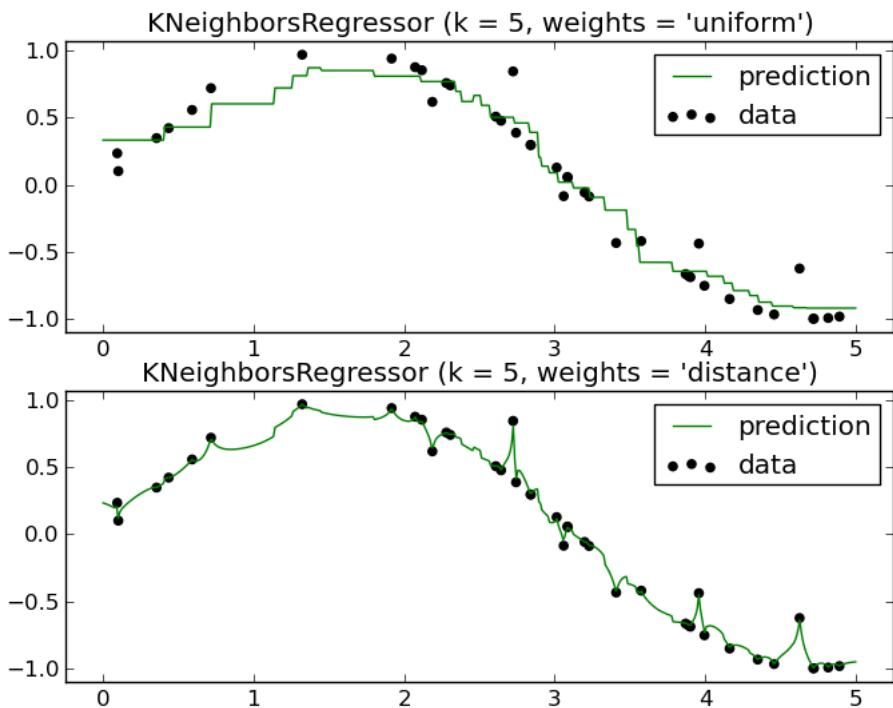
$$\alpha(q) = \frac{\sum_{i=1}^k K\left(\frac{\rho(q, x_i)}{h}\right) \cdot y_i}{\sum_{i=1}^k K\left(\frac{\rho(q, x_i)}{h}\right)}$$

На результат предсказания, как и в задаче классификации, влияет и число соседей, и выбор формулы для предсказания.

Влияние числа соседей изображено на этой картинке:



Пример разных способов учесть расстояния в задаче регрессии изображен здесь:



### 15.3.7. Преимущества и недостатки KNN

Преимущества:

- Простой алгоритм (для объяснения и для интерпретации)
- Метод не делает никаких предположений о данных (об их линейной разделимости, о распределении данных)
- В некоторых задачах достаточно хорошо работает
- Применяется и для классификации, и для регрессии

Недостатки:

- Требует больших ресурсов по памяти, так как хранит всю выборку
- Требует больших ресурсов по времени, так как вычисляет расстояния до всех объектов выборки
- Чувствителен к масштабу данных
- Зависит от выбранной метрики, которая в свою очередь должна отражать реальное сходство объектов. Найти такую метрику не всегда просто или даже невозможно

### 15.3.8. Реализация в питоне

Запустить KNN можно с помощью

```
from sklearn.neighbors import KNeighborsClassifier
```

- Параметр `n_neighbors` - гиперпараметр числа соседей.
- Параметр `weights` принимает способы учесть расстояние до соседей (`uniform`, `distance`).

## 15.4. Быстрый поиск соседей

В алгоритме KNN для определения ближайших соседей необходимо сделать  $O(ld)$  операций, где  $l$  - число объектов,  $d$  - число признаков. Это очень большая величина, поэтому если данных много - алгоритм будет работать очень долго.

В то же время задача поиска близких объектов очень актуальна. В частности, из множества текстов часто требуется находить похожие. Поэтому нужны алгоритмы, которые работают быстрее, чем KNN и могут определить ближайших соседей с высокой точностью.

#### 15.4.1. Хеширование

Идея хеширования состоит в том, чтобы разбить объекты на корзины (buckets) с помощью некоторой функции (хеш-функции). Процедуру разбиения на бакеты необходимо произвести несколько раз.

Тогда интуитивно понятно, что похожие объекты часто будут попадать в одинаковые бакеты (если хеш-функция для задачи выбрана разумно), а непохожие почти всегда будут попадать в разные бакеты.

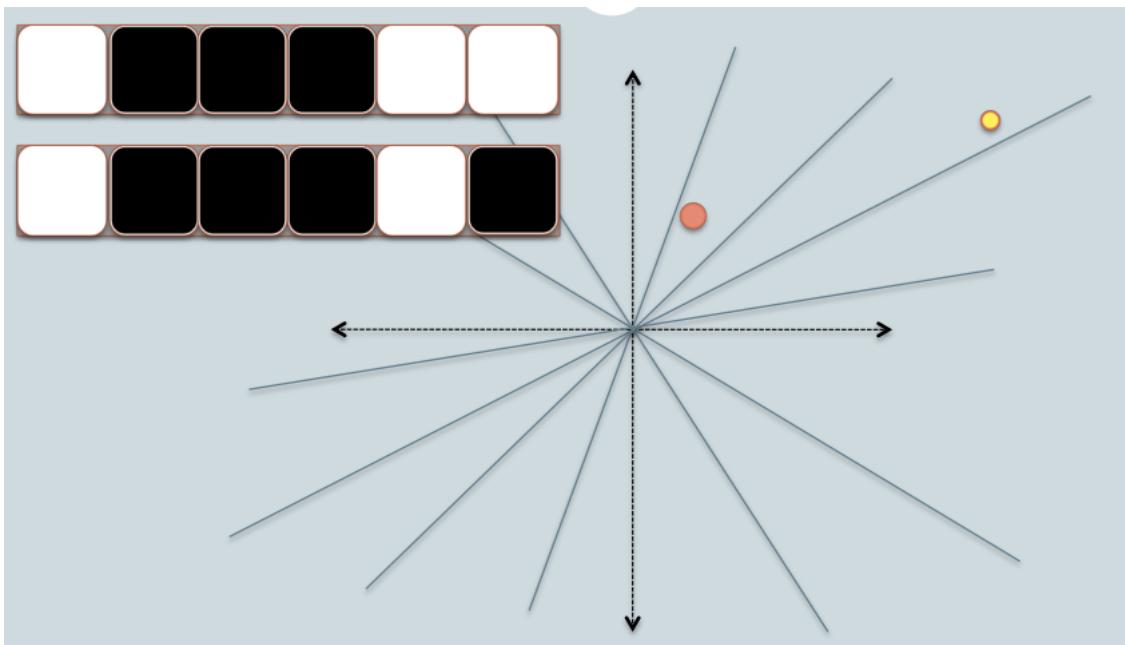
Существует два наиболее используемых подхода к хешированию:

- Метод случайных проекций
- MinHashing

Метод случайных проекций больше используется для табличных задач, где каждый объект - это точка в пространстве признаков. Второй метод больше подходит для поиска похожих текстов.

#### 15.4.2. Метод случайных проекций

Пусть у нас есть произвольные объекты (это точки в  $d$ -мерном пространстве, где  $d$  - число признаков объекта). Как измерить близость этих объектов?



Алгоритм:

1. Проведем несколько случайных гиперплоскостей, проходящих через начало координат (в нашем примере их 6).
2. Поставим 1 для объекта, если он находится выше гиперплоскости, и 0 если ниже.
3. Таким образом, каждый объект закодирован вектором из 0 и 1 длины 6: первая точка = [0, 1, 1, 1, 0, 0], вторая точка = [0, 1, 1, 1, 0, 1]
4. Схожесть объектов можно определять по количеству совпавших в кодировке позиций. Это так, потому что похожие объекты находятся близко друг к другу, поэтому часто будут попадать с одной стороны от гиперплоскости. В данном случае похожесть равна  $\frac{5}{6}$

Такой способ вычислять схожесть связан с расстоянием Хемминга (Hamming distance). Расстояние Хемминга - это число различных позиций в кодировке. В нашем примере расстояние Хемминга равно 1.

#### 15.4.3. MiniHashing

Второй подход более хитрый, чем первый. Для того, чтобы в нем разобраться, необходимо узнать по шагам, что такое:

- Мера Жаккара и матрица текстов
- MinHashing
- Locality-sensitive hashing

**15.4.3.1. Мера Жаккара** Часто стоит задача искать именно похожие тексты. Давайте научимся измерять похожесть текстов.

Часто делают так: разбивают текст на буквенные N-граммы (кусочки из N подряд идущих букв), а затем считают долю совпадающих в двух текстах N-грамм среди общего числа различных N-грамм.

Пример: вычислим похожесть имен Маша и Саша.

Разобъем слова на биграммы ( $N = 2$ )

- Маша → [ма, аш, ша]
- Саша → [са, аш, ша]

Всего различных биграмм 4, а совпадающих 2. Поэтому мера похожести этих слов равна  $\frac{2}{4}$ .

Эта метрика называется мерой Жаккара (другое название - IoU, Intersection over Union).

Общая формула для меры Жаккара для вычисления похожести  $A$  и  $B$ :

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

**15.4.3.2. Матрица текстов** Итак, теперь мы умеем вычислять близость текстов. Надо еще научиться кодировать тексты, используя разбиение на N-граммы.

1. Пронумеруем все N-граммы, встречающиеся в нашем наборе текстов (эти N-граммы иногда называются shingles)
2. Закодируем тексты матрицей:

Documents				
Shingles	0	1	0	1
0	1	0	1	
1	0	1	0	
1	0	0	1	
1	0	1	0	
1	0	1	0	
0	1	0	1	
0	1	0	1	

Здесь по строкам стоят N-граммы, а по столбцам - тексты. Например, если взять второй столбец, то в нём написано, что во втором тексте присутствуют N-граммы 1, 2, 3 и 6, и отсутствуют N-граммы 4, 5 и 7.

Теперь мы умеем кодировать тексты N-граммами и получать из них матрицу. Дальше мы могли бы оценивать схожесть текстов по мере Жаккара, но есть одна довольно **большая проблема**:

В реальных задачах текстов очень много (десятки или сотни тысяч), и различных N-грамм тоже очень много (их могут быть сотни тысяч)! Поэтому полученная матрица будет огромной и при этом разреженной (большинство значений в ней - это нули). Для хранения этой матрицы нужно много ресурсов, кроме того анализ будет занимать большое количество времени.

Поэтому работать с исходной матрицей невозможно, необходимо преобразовать ее так, чтобы по полученной новой матрице меньшего размера аналогичной логикой можно было находить похожие тексты.

**15.4.3.3. MinHashing** Нам необходимо закодировать каждый текст вектором одной длины так, чтобы избежать очень больших размерностей. При этом кодировка должна сохранять свойство похожести текстов. Для этого используется хеширование с MinHash-функцией.

Объясним как работает MinHash. Пронумеруем строки полученной в предыдущем шаге матрицы текстов и перемешаем их случайным образом:

Permutation  $\pi$  Input matrix (Shingles x Documents)

2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

MinHash от каждого предложения - это минимальный из индексов, в котором в соответствующем столбце стоит 1.

Для перестановки, показанной на картинке, MinHash получится следующим:

C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>
----------------	----------------	----------------	----------------

Permutation  $\pi$  Input matrix (Shingles x Documents)

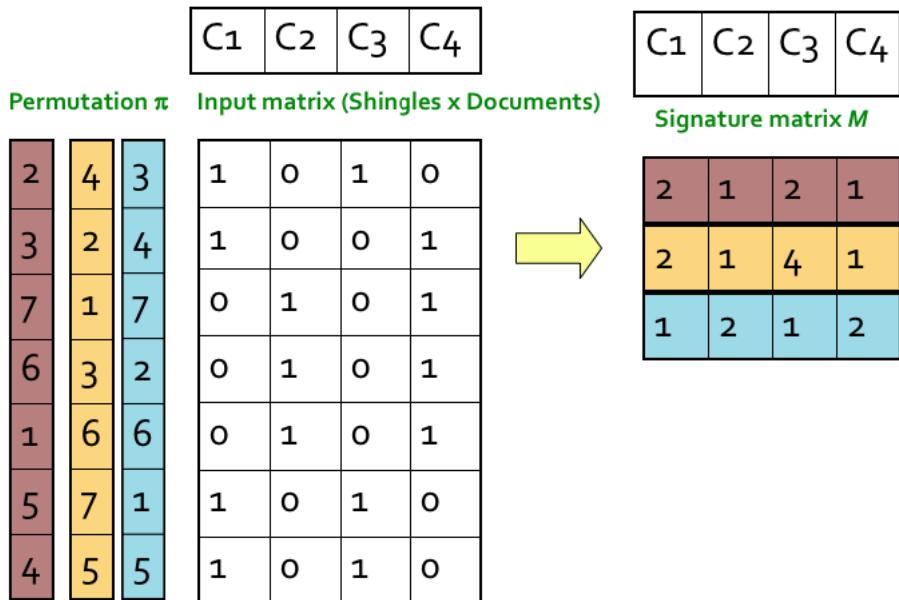
2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

Signature matrix  $M$



2	1	2	1
---	---	---	---

Одной случайной перестановки недостаточно, поэтому эта процедура (случайная перестановка и вычисление MinHash) происходит несколько раз. Например, при трех случайных перестановках может получиться следующее:



Как мы получаем значения матрицы Signature matrix M? По сути, для строки выполнено следующее:

$$M[i] = \min_{j \in \text{Permutation}} C[j][i] == 1$$

Полученная справа матрица называется матрицей сигнатур (а каждый её столбец - сигнатурой соответствующего текста).

Зачем всё это?

Оказывается, что схожесть текстов, вычисленная по матрице сигнатур, приблизительно равна мере Жаккара, посчитанной по исходным N-граммам! (это не очевидный факт, а теорема с непростым доказательством)

То есть, например, схожесть текстов C1 и C3 равна  $\frac{2}{3}$  (так как в столбцах C1 и C3 совпадают два числа из трёх). Тогда мера Жаккара тоже равна примерно  $\frac{2}{3}$ .

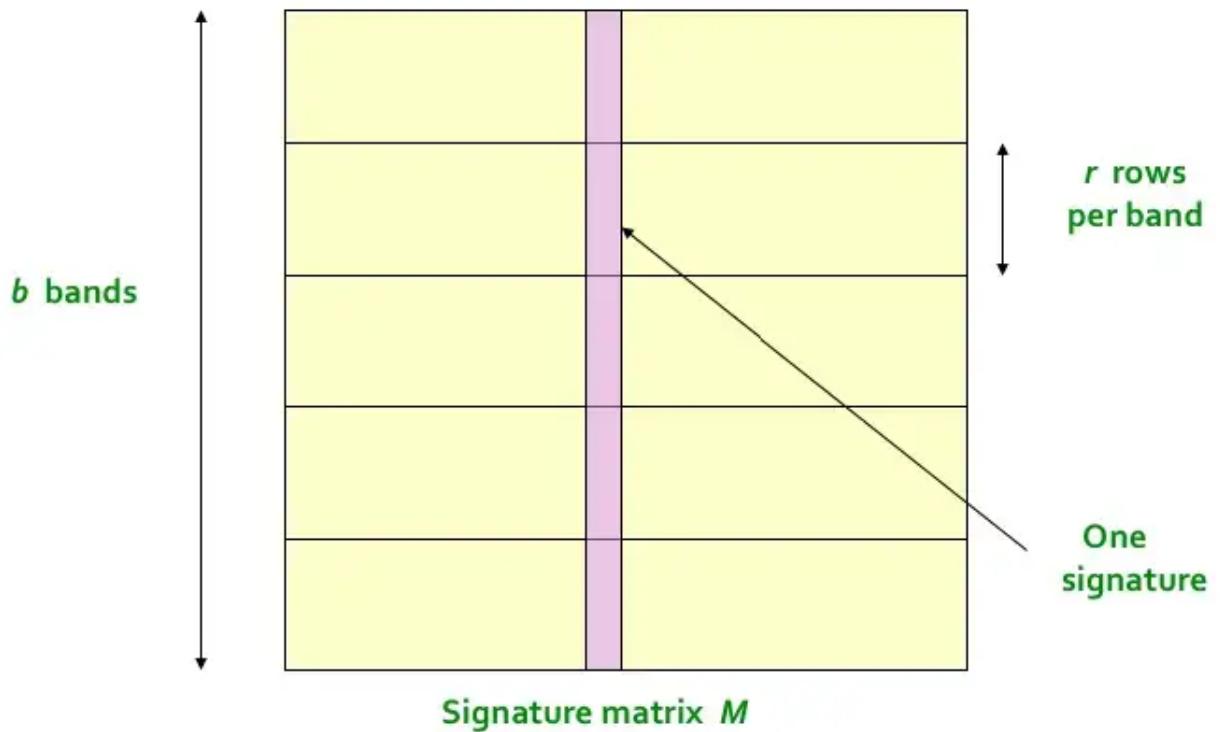
Значит, по матрице сигнатур можно быстро и довольно точно вычислять схожесть текстов!

Конечно, чем больше перестановок используется, тем точнее считается схожесть.

**15.4.3.4. Locality-sensitive hashing (LSH)** LSH - это алгоритм, который позволяет найти похожие документы с некоторой фиксированной вероятностью. То есть, мы можем зафиксировать порог вероятности, скажем, 80%, и с помощью алгоритма LSH находить документы, похожие друг на друга с этой вероятностью.

Алгоритм LSH заключается в следующем:

- Мы разбиваем матрицу сигнатур на несколько равных полос по строкам (они называются bands):



- Затем для каждой полосы делаем следующее: разбиваем все столбики на корзины (buckets) таким образом, что если два столбика совпали, то они попадают в одну корзину, а различные столбики попадают в разные корзины.

Здесь очень важно для понимания не забыть - что это матрица сигнатур, а не матрица текстов.

#### 15.4.3.5. Гиперпараметры LSH

В алгоритме LSH мы выбираем несколько гиперпараметров:

- Число перестановок  $N$  для получения матрицы сигнатур
- Число полос  $b$ .

Чем больше число полос  $b$ , тем большая вероятность того, что два каких-то кусочки попадут в один bucket при хешировании.

- Во-первых, чем больше  $b$ , тем меньше длины  $r$  кусочков (векторы меньшей длины с большей вероятностью совпадут)
- Во-вторых, чем больше  $b$ , что у каких-то двух документов совпали хотя бы в одной строке кусочки (если  $b = 10$ , то таких шансов 10, а если 20 - то таких шансов 20)

## 16. Лекция 9. Быстрый поиск ближайших соседей (LSH + kNN). Наша лекция

### 16.1. Метод случайных проекций

#### 16.1.1. Загрузка данных

```
import pandas as pd

df = pd.read_csv("Tweets.csv", delimiter=',', error_bad_lines=False)

df = df[['text']].dropna()
df.reset_index(drop=True, inplace=True)
df['id'] = np.arange(len(df))

df.head()
```

Получились какие-то очень длинные тексты из Твиттера.

#### 16.1.2. Кодирование TfIdf

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(
    analyzer='char',
    ngram_range=(1, 3),
    min_df=0,
    stop_words='english')

X_tfidf = tfidf.fit_transform(df['text'])
```

#### 16.1.3. Найдём похожие слова

```
import numpy as np

def get_similarity_items(X_tfidf, item_id, topn=5):
    """
    Get the top similar items for a given item id
    The similarity measure here is based on cosine distance.
    """
    query = X_tfidf[item_id]
    scores = X_tfidf.dot(query.T).toarray().ravel()
    best = np.argpartition(scores, -topn)[-topn:]
    return sorted(zip(best, scores[best])), key=lambda x: -x[1])

similar_items = get_similarity_items(X_tfidf, item_id=15)

# an item is always most similar to itself, in real-world
# scenario we might want to filter itself out from the output

for similar_item, similarity in similar_items:
    print(similar_item, similarity)
    item_description = df.iloc[similar_item]['text']
    print('similar_item_id:', similar_item)
    print('cosine_similarity:', similarity)
    print('item_description:', item_description)
    print()
```

Здесь мы видим, что, естественно, само слово похоже на себя с похожестью 1.

#### 16.1.4. Сгенерируем для LSH случайные плоскости

```
def generate_random_vectors(dim, n_vectors):
    """
    generate random projection vectors
    the dims comes first in the matrix's shape,
    so we can use it for matrix multiplication.
    """
    return np.random.randn(dim, n_vectors)

vocab_size = len(tfidf.get_feature_names())
print('vocabulary_size:', vocab_size)

np.random.seed(0)
n_vectors = 16
random_vectors = generate_random_vectors(vocab_size, n_vectors)
print('dimension:', random_vectors.shape)
random_vectors
```

#### 16.1.5. Получим наши LSH

```
data_point = X_tfidf[0]

#True is positive sign; False if negative sign
bin_indices_bits = data_point.dot(random_vectors) >= 0
print('dimension:', bin_indices_bits.shape)
bin_indices_bits
```

#### 16.1.6. Переводим из массива True/False в десятичный вид

```
bin_indices_bits = data_point.dot(random_vectors) >= 0

# https://wiki.python.org/main/BitwiseOperators
# x << y is the same as multiplying x by 2**y
power_of_two = 1 << np.arange(n_vectors - 1, -1, step=-1)
print(powers_of_two)

#final integer representation of individual bins
bin_indices = bin_indices_bits.dot(power_of_two)
print(bin_indices)
```

#### 16.1.7. Соберём всё в одну функцию

```
from collections import defaultdict

def train_lsh(X_tfidf, n_vectors, seed=None):
    if seed is not None:
        np.random.seed(seed)

    dim = X_tfidf.shape[1]
    random_vectors = generate_random_vectors(dim, n_vectors)

    # partition data points into bins,
    # and encode bin index bits into integers
    bin_indices_bits = X_tfidf.dot(random_vectors) >= 0
    powers_of_two = 1 << np.arange(n_vectors - 1, -1, step=-1)
    bin_indices = bin_indices_bits.dot(powers_of_two)

    #update 'table' so that 'table[i]' is the list of document ids with bin index equal to
```

```

table = defaultdict(list)
for idx, bin_index in enumerate(bin_indices):
    table[bin_index].append(idx)

# note that we are storing the bin_indices here
# so we can do some ad-hoc checking with it,
# this isn't actually required
model = {
    'table': table,
    'random_vectors': random_vectors,
    'bin_indices': bin_indices,
    'bin_indices_bits': bin_indices_bits
}

return model

#train the model
n_vectors = 16
model = train_lsh(X_tfidf, n_vectors, seed=143)
model

```

### 16.1.8. Посмотрим на похожие объекты

```

#comparison
similar_item_ids = [similar_item for similar_item, _ in similar_items]

bits1 = model['bin_indices_bits'][similar_item_ids[0]]
bits2 = model['bin_indices_bits'][similar_item_ids[1]]

print('bits_1:', bits1)
print('bits_2:', bits2)
print('Number_of_agreed_bins:', np.sum(bits1 == bits2))

```

### 16.1.9. Напишем алгоритм поиска похожих слов, который учитывает количество различающихся позиций

```
from itertools import combinations
```

```

def search_nearly_bins(query_bin_bits, table, search_radius=3, candidate_set=None):
    """
    For a given query vector and trained LSH model's table
    return all candidate neighbors with the specified search radius.

```

*Example*

---

```

model = train_lsh(X_tfidf, n_vectors=16, seed=143)
query = model['bin_index_bits'][0]
candidates = search_nearly_bins(query, model['table'])
"""

if candidate_set is None:
    candidate_set = set()

n_vectors = query_bin_bits.shape[0]
powers_of_rwo = 1 << np.arange(n_vectors - 1, -1, step=-1)

for different_bits in combinations(range(n_vectors), search_radius):
    #flip the bits (n_1, n_2, ..., n_r) of the query bin to produce a new bit vector
    index = list(different_bits)
    alternate_bits = query_bin_bits.copy()
    alternate_bits[index] = np.logical_not(alternate_bits[index])

```

```

#convert the new bit vector to an integer index
nearby_bin = alternate_bits.dot(powers_of_two)

#fetch the list of documents belonging to
# the bin indexed by the new bit vector,
# then add those documents to candidate_set;
# make sure that the bin exists in the table
if nearby_bin in table:
    candidate_set.update(table[nearby_bin])
return candidate_set

```

### 16.1.10. kNN

Всё это мы делали, чтобы применить ускоренный kNN. Мы будем делать поиск ближайших соседей не среди всех соседей, а только среди похожих.

```

from sklearn.metrics.pairwise import pairwise_distances

def get_nearest_neighbours(X_tfidf, query_vector, model, max_search_radius=3):
    table = model['table']
    random_vectors = model['random_vectors']

    # compute bin index for the query vector, in bit representation.
    bin_index_bits = np.ravel(query_vector.dot(random_vectors) >= 0)

    # search nearby bins and collect candidates
    candidate_set = set()
    for search_radius in range(max_search_radius + 1):
        candidate_set = search_nearby_bins(bin_index_bits, table, search_radius, candidate_set)

    # sort candidates by their true distances from the query
    candidate_list = list(candidate_set)
    candidates = X_tfidf[candidate_list]
    distance = pairwise_distances(candidates, query_vector, metric='cosine').flatten()

    distance_col = 'distance'
    nearest_neighbours = pd.DataFrame({
        'id': candidate_list,
        distance_col: distance
    }).sort_values(distance_col).reset_index(drop=True)
    return nearest_neighbours

```

### 16.1.11. Пример применения

```

print('original_similar_items:\n' + str(similar_items))

item_id = 15
query_vector = X_tfidf[item_id]
nearest_neighbours = get_nearest_neighbours(X_tfidf, query_vector, model, max_search_radius)
print('dimension:', nearest_neighbours.shape)
nearest_neighbours.head()

```

### 16.1.12. Проверка на адекватность

Давайте попробуем склеить полученных соседей и принадлежащие им строки

```

# we can perform a join with the original table to get the description
# for sanity checking purpose
nearest_neighbours.head().merge(df, on='id', how='inner')

```

## 16.2. MiniHashing

### 16.2.1. Сделаем шинглы

```
document = 'Lorem_Ipsum_dolor_sit_amet'
#shingles and discard the last 5 as these are just the last n<5 characters from the document
shingles = [document[i:i+5] for i in range(len(document))][-5:]
```

### 16.2.2. Сделаем шинглы ещё одного текста

```
other_document = 'Lorem_Ipsum_dolor_sit_amet_is.How_dummy_text_starts'
#shingles and discard the last 5 as these are just the last n<5 characters from the document
other_shingles = [other_document[i:i+5] for i in range(len(other_document))][-5:]

# Jaccard distance is the size of set intersection divided by the size of set union
len(set(shingles) & set(other_shingles)) / len(set(shingles) | set(other_shingles))
```

### 16.2.3. Проведём эксперименты minihash помощью библиотеки

```
from lsh import minihash

for _ in range(5):
    hasher = minihash.Minihasher(seeds=100, char_ngram=5)
    fingerprint0 = hasher.fingerprint('Lorem_Ipsum_dolor_sit_amet')
    fingerprint1 = hasher.fingerprint('Lorem_Ipsum_dolor_sit_amet_is.How_dummy_text_starts')
    print(sum(fingerprint0[i] in fingerprint1 for i in range(hasher.num_seeds)) / hasher.n)
```

### 16.2.4. Увеличим кол-во перестановок

```
from lsh import minihash

for _ in range(5):
    hasher = minihash.Minihasher(seeds=1000, char_ngram=5)
    fingerprint0 = hasher.fingerprint('Lorem_Ipsum_dolor_sit_amet')
    fingerprint1 = hasher.fingerprint('Lorem_Ipsum_dolor_sit_amet_is.How_dummy_text_starts')
    print(sum(fingerprint0[i] in fingerprint1 for i in range(hasher.num_seeds)) / hasher.n)
```

### 16.2.5. Сделаем LSH по матрице сигнатур

```
import itertools

from lsh import cache, minihash

# a pure python shingling function that will be used in comparing
# LSH to true Jaccard similarities
def shingles(text, char_ngram=5):
    return set(text[head:head + char_ngram] for head in range(0, len(text) - char_ngram))

def jaccard(set_a, set_b):
    intersection = set_a & set_b
    union = set_a | set_b
    return len(intersection) / len(union)

def candidate_duplicates(document_feed, char_ngram=5, seeds=100, bands=5, hashbytes=4):
    sims = []
    hasher = minihash.MiniHasher(seeds=seeds, char_ngram=char_ngram, hashbytes=hashbytes)
    if seeds % bands != 0:
```

```

raise ValueError(f 'Seeds_hat_to_be_a_multiple_of_bands.{seeds} % {bands} != 0')

lshcache = cache.Cache(num_bands=bands, hasher=hasher)
for line in document_feed:
    line = line.decode('utf-8')
    docid, headline_text = line.split('\t', 1)
    fingerprint = hasher.fingerprint(headline_text.encode('utf8'))

# in addition to storing the fingerprint store the line
# number and document ID to help analysis later on
    lshcache.add_fingerprint(fingerprint, doc_id=docid)

candidate_pairs = set()
for b in lshcache.bins:
    for bucket_id in b:
        if len(b[bucket_id]) > 1:
            pairs_ = set(itertools.combinations(b[bucket_id], r=2))
            candidate_pairs.update(pairs_)

```

## 17. Статья про SHAP

Статья - <https://habr.com/ru/post/428213/>

### 17.1. Какую проблему решает SHAP

Дисклаймер: какова проблема, которую решает SHAP?

В стеке sklearn, в пакетах xgboost, lightGBM были встроенные методы оценки важности фичей (feature importance) для «деревянных моделей»:

- Gain. Эта мера показывает относительный вклад каждой фичи в модель. Для расчета мы идем по каждому дереву, смотрим в каждом узле дерева какая фича приводит к разбиению узла и насколько снижается неопределенность модели согласно метрике (Gini impurity, information gain). Для каждой фичи суммируется её вклад по всем деревьям.
- Cover. Показывает количество наблюдений для каждой фичи. Например, у вас 4 фичи, 3 дерева. Предположим, фича 1 в узлах дерева содержит 10, 5 и 2 наблюдения в деревьях 1, 2 и 3 соответственно. Тогда для данной фичи важность будет равна 17 ( $10 + 5 + 2$ ).
- Frequency. Показывает, как часто данная фича встречается в узлах дерева, то есть считается суммарное количество разбиений дерева на узлы для каждой фичи в каждом дереве.

Основная проблема во всех этих подходах, что непонятно, как именно данная фича влияет на предсказание модели. Например, мы узнали, что уровень дохода важен для оценки платежеспособности клиента банка для выплаты кредита. Но как именно? Насколько сильно более высокий доход смещает предсказания модели?

Мы, конечно, можем сделать несколько предсказаний, меняя уровень дохода. Но что делать с другими фичами? Ведь мы попадаем в ситуацию, что надо получить понимание влияние дохода независимо от других фичей, при их некотором среднем значении.

Есть этакий среднестатистический клиент банка «в вакууме». Как будут меняться предсказания модели в зависимости от изменения дохода?

Тут-то на помощь и приходит библиотека SHAP.

### 17.2. Рассчитываем важность фичей с помощью SHAP

#### 17.2.1. Вступление из теории игр

Для рассчёта берётся значение Шэпли. Что это такое давайте проясним:

Допустим группа людей играет в карты. Как распределить призовой фонд между ними в соответствии с их вкладом?

Делается ряд допущений:

- Сумма вознаграждения каждого игрока равна общей сумме призового фонда
- Если два игрока сделали равный вклад в игру, они получают равную награду
- Если игрок не внес никакого вклада, он не получает вознаграждения
- Если игрок провел две игры, то его суммарное вознаграждение состоит из суммы вознаграждений за каждую из игр

Мы представляем фичи модели в качестве игроков, а призовой фонд — как итоговое предсказание модели.

#### 17.2.2. Пример рассчёта значения Шэпли

Формула для рассчёта значения Шэпли для  $i$ -й фичи выглядит так:

$$\phi_i(p) = \sum_{S \subseteq N/\{i\}} \frac{|S|!(n-|S|-1)!}{n!} \cdot (p(S \cup \{i\}) - p(S))$$

где

- $S$  - произвольный набор фичей без  $i$ -й фичи
- $n$  - количество фичей

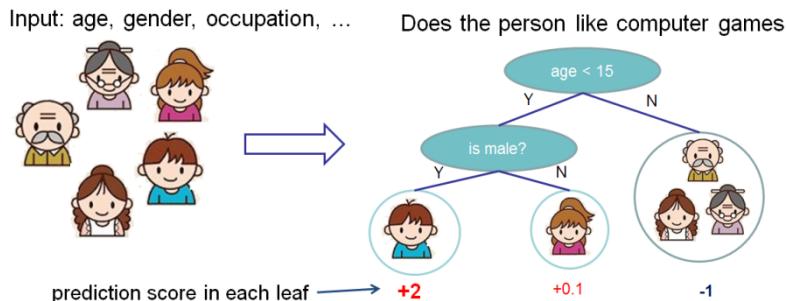
- $p(S)$  - предсказание без  $i$ -й фичи
- $p(S \cup \{i\})$  - предсказание модели с  $i$ -й фичей

Значение Шэпли для  $i$ -той фичи рассчитывается для каждого сэмпла данных (например, для каждого клиента в выборке) на всех возможных комбинациях фичей (включая отсутствие всех фичей), затем полученные значения суммируются по модулю и получается итоговая важность  $i$ -той фичи.

Данные вычисления чрезвычайно затратны, поэтому под капотом используются различные алгоритмы оптимизации вычислений.

Допустим есть задача: хотим оценить важность фичей для предсказания, нравятся ли человеку компьютерные игры.

В этом примере для простоты у нас есть две фичи: age (возраст) и gender (пол). Gender (пол) принимает значения 0 и 1.



Возьмём Bobby (маленький мальчик в самом левом узле дерева) и посчитаем значение Шэпли для фичи age (возраст).

У нас есть два набора фичей  $S$ :

- $\{\}$  - нет фичей
- $gender$  - есть только фича пол

**17.2.2.1. Ситуация, когда нет значений фичей** Разные модели по-разному работают с ситуациями, когда для сэмпла данных нет фичей, то есть для всех фичей значения равны NULL.

Будет считать в данном случае, что модель усредняет предсказания по веткам дерева, то есть предсказание без фичей будет  $[(2 + 0.1)/2 + (-1)]/2 = 0.025$ .

Если же мы добавим знание возраста, то предсказание модели будет  $(2 + 0.1)/2 = 1.05$ .

В итоге значение Шэпли для случая отсутствия фичей будет:

$$\frac{|S|!(n - |S| - 1)!}{n!} \cdot (p(S \cup \{i\}) - p(S)) = \frac{1(2 - 0 - 1)!}{2!} \cdot 1.025 = 0.5125$$

**17.2.2.2. Ситуация, когда знаем пол** Для Bobby для  $gender$  предсказание без фичи возраст, только с фичей пол, равно  $[(2 + 0.1)/2 + (-1)]/2 = 0.025$ . Если же мы знаем возраст, то предсказание — это самое левое дерево, то есть 2.

В итоге значение Шэпли для этого случая:

$$\frac{|S|!(n - |S| - 1)!}{n!} \cdot (p(S \cup \{i\}) - p(S)) = \frac{1}{(2 - 1 - 1)!2!} \cdot (1.975) = 0.9875$$

**17.2.2.3. Суммируем** Итогое значение Шэпли для фичи age (возраст):

$$\phi_{AgeBobby} = 0.9875 + 0.5125 = 1.5$$

### 17.3. Реальный пример из бизнеса

Библиотека SHAP обладает богатым функционалом визуализации, который помогает легко и просто объяснить модель как для бизнеса, так и для самого аналитика, чтобы оценить адекватность модели.

На одном из проектов я анализировал отток сотрудников из компании. В качестве модели использовался xgboost.

```

import shap

shap_test = shap.TreeExplainer(best_model).shap_values(df)
shap.summary_plot(shap_test, df,
                  max_display=25, auto_size_plot=True)

```

Получившийся график важности фичей:



Как его читать:

- значения слева от центральной вертикальной линии — это negative класс (0), справа — positive (1)
- чем толще линия на графике, тем больше таких точек наблюдения
- чем краснее точки на графике, тем выше значения фичи в ней

Из графика можно сделать интересные выводы и проверить их адекватность:

- чем меньше сотрудник повышают зарплату, тем выше вероятность его ухода
- есть регионы офисов, где отток выше
- чем моложе сотрудник, тем выше вероятность его ухода
- ...

Можно сразу сформировать портрет уходящего сотрудника: ему не повышали зарплату, он достаточно молод, холост, долгое время на одной позиции, не было повышений грейда, не было высоких годовых оценок, он стал мало общаться с коллегами.

Просто и удобно!

## 18. Статья про LIME

Статья - <https://habr.com/ru/company/ods/blog/599573/>

В этом обзоре мы рассмотрим, как методы LIME и SHAP позволяют объяснять предсказания моделей машинного обучения, выявлять проблемы сдвига и утечки данных, осуществлять мониторинг работы модели в production и искать группы примеров, предсказания на которых объясняются схожим образом.

Также поговорим о проблемах метода SHAP и его дальнейшем развитии в виде метода Shapley Flow, объединяющего интерпретацию модели и многообразия данных.

### 18.1. Интерпретация моделей машинного обучения

Модели машинного обучения (такие как нейронные сети, машины опорных векторов, ансамбли решающих деревьев) являются "прозрачными" в том смысле, что все происходящие внутри них вычисления известны. Но тем не менее часто говорят, что модели машинного обучения плохо интерпретируемые. Здесь имеется в виду то, что процесс принятия решения не удается представить в понятной человеку форме, то есть:

1. Понять, какие признаки или свойства входных данных влияют на ответ
2. Разложить алгоритм принятия решения на понятные составные части
3. Объяснить смысл промежуточных результатов, если они есть
4. Описать в текстовом виде алгоритм принятия решения (возможно, с привлечением схем или графиков)

Достичь полной интерпретируемости в машинном обучении, как правило, не удается, но даже частичная интерпретация может существенно помочь. Обзор способов интерпретации моделей машинного обучения можно найти, например, в Linardatos et al., 2020 и Li et al., 2021. Чем же может помочь интерпретация модели?

Во-первых, интерпретировав алгоритм, мы можем открыть для себя что-то новое о свойствах исследуемых данных (например, какие признаки в табличных данных на наибольшей степени влияют на ответ).

Во-вторых, интерпретация модели помогает оценить ее качество. Если мы узнаем, на что именно обращает модель, какими правилами руководствуется при предсказании, то сможем оценить правдоподобность этих правил.

Казалось бы, разве недостаточно знать метрику качества модели на тестовой выборке? Часто может оказаться, что после развертывания модель работает в среднем хуже, чем на тестовой выборке из-за ряда проблем, таких как сдвиг и утечка данных.

#### 18.1.1. Распределение данных

На протяжении всего обзора будет использоваться понятие распределения данных, поэтому сначала повторим значение этого понятия.

Если модель использует  $N$  входных признаков, и  $i$ -й признак принимает значения из множества  $X_i$ , то все пространство признаков равно  $X = X_1 \times X_2 \times \dots \times X_N$ . Однако на практике далеко не все сочетания этих признаков возможны, то есть область реальных данных представляет собой лишь малую часть пространства  $X$ .

Как правило, в машинном обучении используется статистический подход (statistical learning framework), при котором имеющийся датасет  $D = \{x_i, y_i\}_{i=1}^{\ell}$  рассматривается как выборка из совместного распределения данных  $P(x, y) = P(x)P(y|x)$ , иногда называемого также генеральной совокупностью. Конечно, представление датасета  $D$  как взятого из распределения  $P(x, y)$  довольно условно, потому что обычно мы имеем лишь конечную выборку данных, но не имеем строгого определения для  $P(x, y)$ . Но в целом мы считаем, что  $P(x)$  наиболее велико для "типовых" примеров  $x$ , и равно нулю для невозможных примеров.  $P(x, y)$  также задает вероятность для любого подмножества примеров.

Например, пусть мы имеем датасет из объявлений о продаже автомобилей. Для нашего датасета верно, например, следующее:

- Количество авто "Lada Granta" превосходит количество авто "Москвич-412"
- Количество авто "Победа" с двигателем мощностью 500 л. с. равно нулю

Тогда мы можем считать датасет выборкой из распределения, в котором для  $P(x)$  верно следующее:

- $P(x|\text{марка}(x) = "LadaGranta") > P(x|\text{марка}(x) = "Москвич-412")$
- $P(x|\text{марка}(x) = "Победа" \& \text{мощность}(x) \approx "500 л.с.") \approx 0$

Таким образом,  $P$  описывает соотношение разных типов примеров, и датасет рассматривается как выборка из  $P$ . На таком подходе основана большая часть теоретических исследований, посвященных машинному обучению, в которых доказывается эффективность тех или иных методов.

Модели машинного обучения делятся в основном на дискриминативные и генеративные, при этом генеративные моделируют  $P(x)$  или  $P(x, y)$ , а дискриминативные - только  $P(y|x)$ , при этом в регрессии, как правило, упрощая его до мат. ожидания  $E_{x \sim P(x)}[y|x]$ . Если ваша модель предсказывает целевой признак по исходным - то это дискриминативная модель. Но такие модели не вычивают никакой информации о распределении  $P(x)$ , то есть о том, насколько правдоподобно сочетание входных признаков и лежит ли оно в области реальных данных, то есть само многообразие данных не моделируется.

### 18.1.2. Сдвиг данных

Сдвиг данных (data shift) означает, что данные, на которых модель будет применяться, среднестатистически отличаются от тех, на которых модель обучалась и тестировалась, то есть распределение входных данных отличается при тестировании и применении:  $P_{test}(X, Y) \neq P_{usage}(X, Y)$ . Поскольку  $P(X, Y) = P(Y|X)P(X)$ , то двумя вариантами сдвига данных являются:

1. Сдвиг в распределении исходных данных  $P(X)$
2. Сдвиг в условном распределении целевой переменной  $P(Y|X)$

**Проблема 1.** Большинство метрик качества (accuracy, MSE, logloss, F1 и другие) зависят от распределения исходных данных  $P(X)$ , то есть, упрощенно говоря, от соотношения разных типов примеров в датасете. Например, пусть модель тестировалась на датасете, в котором 80% изображений были высокого качества (HQ), а применяться будет в условиях, когда, наоборот, 80% изображений будут низкого качества (LQ). Пусть мы сравниваем две модели: на HQ-изображениях точность первой модели лучше, чем второй, а на LQ-изображениях, наоборот, точность второй модели лучше, чем первой. Если при тестировании большая часть изображений были HQ, то мы сделаем вывод, что первая модель лучше, тогда как на самом деле лучше была бы вторая.

**Проблема 2.** В обучающих и тестовых данных могут присутствовать такие корреляции, которые не обобщаются на другие выборки. Например, мы классифицируем животных по изображению, но большинство изображений рыб, которые у нас имеются, содержат также пальцы рыбака (эти изображения мы используем как при обучении, так и при тестировании). Модель может научиться классифицировать как рыбу изображение, содержащее пальцы, что в целом неверно и может не работать на других выборках (Brendel and Bethge, 2019).

**Проблема 3.** Обучающие данные часто недостаточно разнообразны, то есть не покрывают все типы примеров, на которых желательна корректная работа модели (либо покрывают их в неправильном соотношении). Поэтому при применении модели могут встречаться примеры таких типов, которые никогда не встречались при обучении или встречались редко, то есть лежали вне распределений  $P_{train}$  и  $P_{test}$  (out-of-distribution, OOD). Иными словами,  $\{x | P_{train}(x) > 0\} \subsetneq \{x | P_{usage}(x) > 0\} \subsetneq X$  (рис. 1), и модель, обучающаяся на  $P_{train}$ , может не научиться корректно работать на  $P_{usage}$ . Таким образом, примеры некоторых типов не встречались при обучении, но на них желательна корректная работа модели, как минимум для защиты от атак, "обманывающих" модель (рис. 1) (см. Brown et al., 2017; Akhtar and Mian, 2018).

Например, в датасете из фотографий практически исключено появление фотографий собаки, покрашенной в радужные цвета, или фотографий лица человека, которому на лоб приkleено фото другого человека. Такие изображения считаются OOD-данными. Также OOD-данными можно считать примеры с очень редкими значениями каких-либо признаков, которые имеют мало шансов попасть в обучающую выборку. Диагностика сдвига данных является важной практической задачей (Yang et al., 2021), а способность модели обобщаться на большее разнообразие примеров и ситуаций, чем те, что встречались при обучении, является открытой проблемой в машинном обучении и одной из метрик "интеллектуальности" модели (Shen et al., 2021; Goyal and Bengio, 2020; Battaglia et al., 2018; Chollet, 2019).

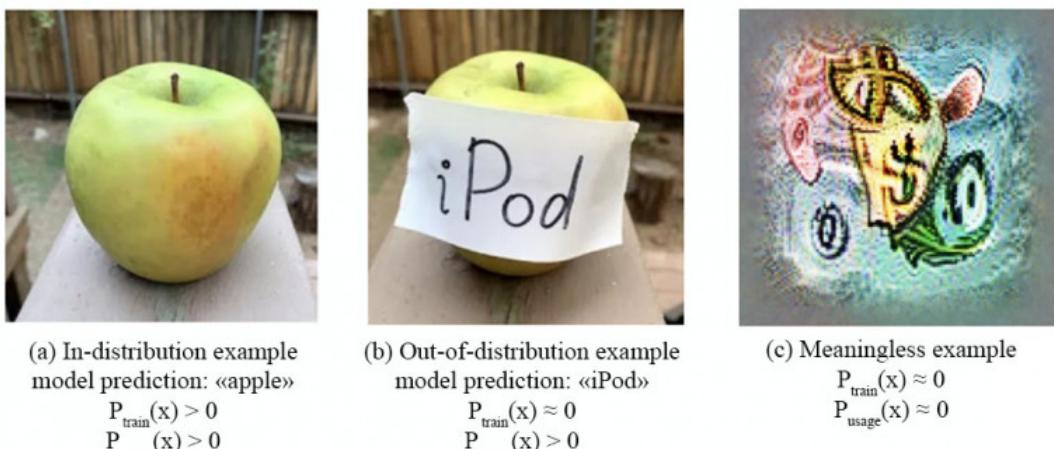


Рис. 1. Примеры некоторых типов не встречались при обучении, но на них желательна корректная работа модели, как минимум для защиты от атак, "обманывающих" модель.

### 18.1.3. Утечка данных

Утечкой данных (data leakage, или target leakage) называется ситуация, когда существует некий признак, который при обучении содержал больше информации о целевой переменной, чем при последующем применении модели на практике. Например, ID пациента может сильно коррелировать с диагнозом, но только в текущем датасете (который поделен на обучающую и тестовую часть). Модель, предсказывающая диагноз по ID, будет иметь высокую точность на тестовом датасете, но в целом очевидно, что в данной задаче такой способ предсказания некорректен и не будет хорошо работать на других данных. Утечка данных является частным случаем сдвига данных, поскольку зависимость  $ID \rightarrow$  диагноз была в  $P_{\text{train}}$  и  $P_{\text{test}}$ , но ее не будет в  $P_{\text{usage}}$ . Диагностировать утечку данных не всегда просто.

Если удастся интерпретировать модель (хотя бы приблизительно), то мы получим дополнительную информацию, которая поможет надежнее оценить ее качество в условиях возможной утечки и сдвига данных.

### 18.1.4. Локальная интерпретация моделей

Вместо попыток интерпретировать модель целиком, что может быть очень сложно, мы можем рассмотреть задачу интерпретации ответа модели  $f$  на конкретном, фиксированном примере  $x_0$ . Например, если на данном изображении модель распознала собаку, то почему она распознала собаку? Какие части и свойства изображения повлияли на предсказание модели?

Для ответа на этот вопрос мы можем изменять  $x_0$  и смотреть, как изменится при этом ответ модели, то есть мы изучаем зависимость  $f(x_0 + \Delta x)$  от  $\Delta x$ . При этом возможно удастся с хорошей точностью аппроксимировать эту зависимость простой функцией  $g(\Delta x)$ . Такой подход называется локальной аппроксимацией модели в окрестности точки  $x_0$ .

Например, рассчитав градиент  $\nabla f(x)$  в точке  $x_0$  мы узнаем, как изменится ответ при очень малых изменениях  $\Delta x$ . При этом мы получаем локальную линейную аппроксимацию (в курсе высшей математики такая аппроксимация называется дифференциалом функции  $f$ ). Такой подход используется, например, при расчете так называемых saliency maps в компьютерном зрении (Simonyan et al., 2013) - производных выходных значений сети по отдельным пикселям изображения. Но такая аппроксимация далеко не всегда адекватна:

- Производная локальна и не говорит о том, как изменится ответ при существенных изменениях  $\Delta x$ . Например, если один из признаков достиг состояния "насыщения" то есть значение данного признака более чем достаточно, чтобы сделать какой-то вывод о целевой переменной, то производная по нему почти равна нулю. Эффект будет лишь если мы сильно изменим данный признак.
- В некоторых моделях (решающих деревьях) производная либо равна нулю, либо не существует.
- Для бинарных признаков производная не всегда информативна, поскольку малое изменение признака ведет в "невозможную" область нецелого значения, в котором модель и не обязана работать корректно.
- Для бинарных признаков производная не всегда информативна, поскольку малое изменение признака ведет в "невозможную" область нецелого значения, в котором модель и не обязана работать корректно.

Есть и другие подходы к локальной интерпретации модели. В качестве  $\Delta x$  мы можем рассматривать некий набор осмысленных, не бесконечно малых изменений входных данных - такой подход используется в методе **LIME** (Ribeiro et al., 2016).

Также мы можем интерпретировать не само предсказание  $f(x_0)$ , а одну из следующих величин:

- Разницу между двумя предсказаниями  $\Delta y = f(x_0) - f(x_{background})$
- Разницу между текущим и усредненным предсказанием  $\Delta y = f(x_0) - E[f(x)]$ .

Это означает, что мы пытаемся объяснить изменение в предсказании, вызванное изменением входных признаков (в первом случае) или появлением информации о входных признаках (во втором случае). При этом мы вычисляем вклад каждого признака в  $\Delta y$ . Такой подход используется в методе **SHAP** (Lundberg and Lee, 2017).

Методы LIME и SHAP можно применить к любой модели машинного обучения, поскольку они никак не используют информацию о том, как устроена модель "изнутри" то есть являются model-agnostic методами (хотя существуют вычислительно эффективные реализации для конкретных видов моделей, такие как Tree SHAP).

Существуют также специфические способы интерпретации, позволяющие объяснить  $\Delta y$  для конкретных типов моделей, например DeepLIFT (Shrikumar et al., 2017) и Integrated Gradients (Sundararajan et al., 2017) для нейронных сетей (т. е. дифференцируемых моделей). Эти методы в данном обзоре мы не будем рассматривать. Обзор других методов локальной интерпретации моделей можно найти, например, в Lundberg et al., 2019, раздел "Methods 7".

## 18.2. LIME: Local Interpretable Model-agnostic Explanations

LIME (Ribeiro et al., 2016) - это подход к интерпретации ответа модели  $f(x_0)$  на конкретном тестовом примере  $x_0$  с помощью вычисления значений  $f(x_0 + \Delta x)$  для некоторого конечного набора значений  $\Delta x$ . Иллюстрация работы метода LIME приведена на рис. 2. Чтобы формально описать метод LIME, нам понадобится ввести ряд обозначений, которые мы будем использовать и в следующих частях обзора.

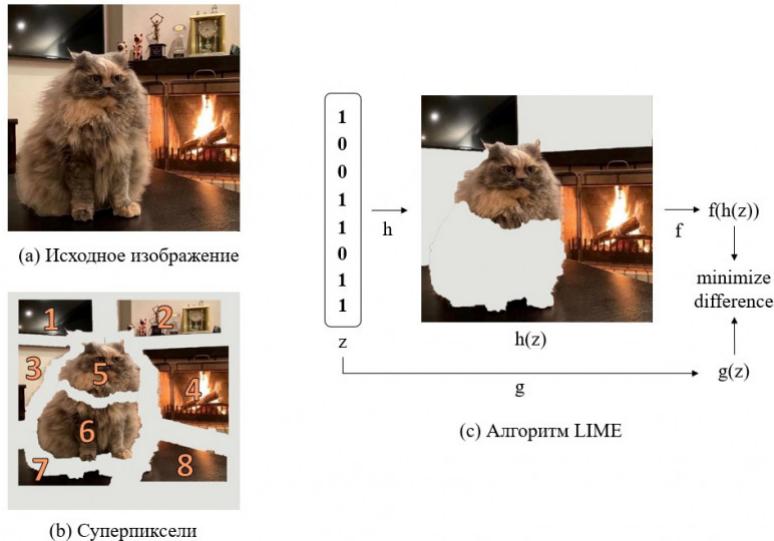


Рис. 2. Иллюстрация работы метода LIME.

Мы интерпретируем модель  $f$  на примере  $x_0$ :

- $f : X \rightarrow Y$  - исходная модель
- $x_0 \in X$  - выбранный тестовый пример, предсказание на котором  $f(x_0)$  интерпретируется

### 18.2.1. Локальное упрощенное представление

Для примера  $x_0$  вводим  $M$  осмысленных, интерпретируемых изменений  $\Delta_i$ . Например, для изображений таким изменением может быть удаление отдельного суперпикселя, то есть участка изображения с похожим содержимым и четкими границами (рис. 1b). Каждое изменение бинарно, то есть оно либо есть, либо нет. Соответственно, мы получаем  $2^M$  различных вариантов  $\Delta x$ . Наличие или отсутствие каждого из изменений можно описать числом 0 или 1: из этих чисел можно собрать бинарный вектор  $z$  размерностью  $M$ , который будем называть упрощенным представлением (рис. 1c).

- $\{\Delta_i\}_{i=1}^M$  - изменения примера  $x_0$
- $z_i \in \{0, 1\}$  - индикатор изменения  $\Delta_i$
- $z \in \{0, 1\}^M$  - вектор упрощённого представления
- $h : \{0, 1\}^M \rightarrow X$  - функция, преобразующая вектор упрощённого представления  $z$  в  $x_0 + \Delta x$

Например, на рис. 1 функция  $h$  работает следующим образом: все "присущие" суперпиксели ( $z_i = 1$ ) рисуются без изменений. Все "отсутствующие" суперпиксели ( $z_i = 0$ ) заполняются белым цветом (либо, как вариант, усредненным цветом соседних суперпикселей). При этом  $h([1, 1, \dots, 1]) = x_0$ , поскольку вектор из единиц означает отсутствие всех изменений. Фактически выбор функции  $h$  равносителен выбору изменений  $\Delta_i$ , то есть семантики упрощенного представления.

Функцию  $h$  мы можем выбрать произвольно, но так, чтобы отдельные изменения  $\Delta_i$  были интерпретируемы. При этом мы надеемся, что объяснить предсказание модели на  $x_0$  можно интерпретировать, изучая, как влияют на ответ эти изменения. Конечно, функция  $h$  может быть выбрана неудачно. Например, если модель определяет стиль фотографии как "ретро" при наличии оттенка "сепия то изменения отдельных суперпикселей не помогут интерпретировать модель. Если  $\Delta_i$  были выбраны неудачно, то всегда можно попробовать заново с другими изменениями  $\Delta_i$ .

Для табличных данных мы можем в качестве  $\Delta_i$  рассматривать замену одного из признаков на ноль, среднее или медианное значение по обучающему датасету.

### 18.2.2. Объясняющая модель

Теперь мы можем обучить модель  $g$  предсказывать значение  $f(h(z))$  по вектору упрощенного представления  $z$ . При этом модель  $g$  должна быть простой и интерпретируемой (поскольку смысл метода LIME в интерпретации). Например, это может быть линейная модель или решающее дерево. Чтобы обучить модель, нужно собрать обучающую выборку. Для этого нам потребуется получить ответ модели  $f$  для разных  $z$  (то есть для разных  $x_0 + \Delta x$ , поскольку  $z$  определяет  $\Delta x$ ) и таким образом собрать обучающую выборку для модели  $g$ .

- $g : \{0, 1\}^M \rightarrow Y$  - объясняющая модель
- $\{z^{(i)}, f(h(z^{(i)}))\}_{i=1}^N$  - обучающая выборка для объясняющей модели

Максимально возможный размер обучающей выборки равен  $2^M$ , но обычно  $M$  велико и приходится ограничиться перебором лишь некоторых значений  $z$ . Авторы предлагают уделять основное внимание таким  $z$ , которые близки к вектору из единиц: это соответствует небольшим изменениям в  $x_0$  (чем больше нулей в  $z$ , тем больше одновременных изменений  $x_0$  мы рассматриваем). Введём функцию  $\pi(z)$ , определяющую меру близости  $h(z)$  к  $x_0$ , и назначим веса  $\pi(z^{(i)})$  примерам из обучающей выборки.

- $\pi : \{0, 1\}^M \rightarrow \mathbb{R}$  - мера близости  $h(z)$  к  $x_0$
- $\{w^{(i)} = \pi(z^{(i)})\}_{i=1}^N$  - веса примеров из обучающей выборки

*Примечание.* В kernel SHAP, который мы рассмотрим в следующих разделах, в качестве  $\pi$  берется функция, назначающая большие веса как векторам с большим количеством единиц, так и векторам с большим количеством нулей. Так мы акцентируем внимание в том числе на значениях  $z$  с большим количеством нулей, то есть на отдельных интерпретируемых компонентах (например, отдельных суперпикселях в случае изображений).

Для обучения модели  $g$  осталось выбрать функцию потерь - например, среднеквадратичное отклонение. Эта функция будет сравнивать предсказания моделей  $f$  и  $g$ . Авторы предлагают также использовать "штраф за сложность"  $\Omega(g)$  - например, количество ненулевых весов в линейной модели. Если в качестве функции потерь выбрано среднеквадратичное отклонение, то задача оптимизации формулируется следующим образом:

$$\sum_{i=1}^N w^{(i)} \left( g(z^{(i)}) - f(h(z^{(i)})) \right)^2 + \Omega(g) \rightarrow \min_g$$

В данной формуле мы считаем квадрат разности предсказаний объясняющей модели  $g(z^{(i)})$  и исходной модели  $f(h(z^{(i)}))$ , и считаем взвешенную сумму по обучающей выборке, используя веса  $w^{(i)}$ . Кроме того мы прибавляем штраф за сложность объясняющей модели  $\Omega(g)$ .

Таким образом, суть подхода LIME в том, что мы аппроксимируем предсказание модели  $f$  в окрестности тестового примера  $x_0$  более простой, легко интерпретируемой моделью  $g$ , которая использует упрощенное представление  $z$ . Например, если модель  $g$  линейна, то каждому изменению  $\Delta_i$  (например, суперпиксели в изображении) сопоставляется некий вес.

При этом мы надеемся, что такая аппроксимация адекватна, то есть наличие  $i$ -го изменения линейно влияет на предсказание модели  $f$ . В некоторых случаях это может оказаться совсем не так, и модель  $g$  не сможет хорошо обучиться (функция потерь останется высокой). Например, в случае изображений это может означать, что мы не можем линейно влиять на предсказание модели, удаляя отдельные суперпиксели. Возможно, модель ориентируется не на отдельные объекты, а на цвет изображения в целом. Тогда можно попробовать использовать другое упрощенное представление, элементами которого является информация об усредненном цвете изображения.

Работа алгоритма LIME не зависит от вида модели  $f$  (нейронная сеть, решающие деревья и т. д.) и никак явно не использует информацию о том, как модель устроена "изнутри то есть LIME является "model-agnostic" алгоритмом интерпретации.

### 18.2.3. Примеры и обсуждение

На рис. 3 мы видим объяснение предсказания сверточной нейронной сети Inception (Szegedy et al., 2014). Сначала мы рассматриваем выходной нейрон, соответствующий классу "Electric guitar" и пытаемся аппроксимировать значение на этом нейроне с помощью линейной модели  $g$ , которая использует информацию о наличии или отсутствии суперпикселей ( $M$  бинарных признаков, где  $M$  - количество суперпикселей). В результате для каждого суперпикселя мы получаем вес, то есть вклад этого суперпикселя в предсказание "Electric guitar" и выделяем суперпиксели с наибольшим весом. Далее повторяем тот же алгоритм для двух других выходных нейронов, соответствующих классам "Acoustic guitar" и "Labrador". Как можно видеть из примера, алгоритм LIME концептуально достаточно прост.

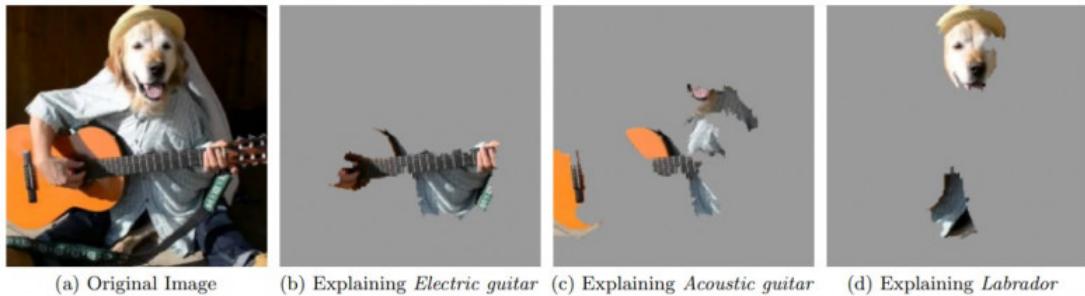


Рис. 3. Пример результатов, полученных с помощью метода LIME. Отмечены суперпиксели с наибольшими весами в линейной объясняющей модели.

Работа алгоритма LIME определяется выбором набора изменений  $\Delta_i$ , этот выбор осуществляется вручную и в каких-то случаях может быть неудачным. Поэтому LIME можно рассматривать как метод, в котором мы сначала формулируем гипотезу о том, как можно было бы объяснить предсказание модели, а затем проверяем ее.

Интересно было бы попробовать применить алгоритм LIME для интерпретации предсказаний человека на различных задачах. Такой подход мог бы дать лучшее понимание плюсов, минусов и границ применимости алгоритма.

Один из минусов заключается в том, что измененные примеры  $x_0 + \Delta x$  могут быть неестественными, ненатуральными (например, изображение, в котором стерта часть суперпикселей). Модель  $f$ , напротив, обучалась только на натуральных примерах, и чаще всего от нее не требуется корректная работа на ненатуральных примерах (не лежащих в многообразии исходных данных, в терминологии из книги Deep Learning, раздел 5.11.3). Хотелось бы проверять работу  $f$  только для тех  $x_0 + \Delta x$ , которые также выглядят натурально. Об этой проблеме мы еще поговорим в дальнейшем в разделе "Соблюдение границ многообразия данных".

Конечно, можно разрабатывать более эффективные способы интерпретации для конкретных предметных областей, но особенность LIME именно в том, что это очень общий подход, который может быть применен к широкому классу моделей. При этом многие детали в нем, в частности вид упрощенного представления, могут быть выбраны произвольно.

### 18.2.4. LIME-SP: объединение локальных интерпретаций в глобальную

Авторы также предлагают надстройку над алгоритмом LIME, называемую submodular pick (SP), которая может помочь интерпретировать модель в целом, а не только на конкретном примере. Для этого выбирается набор тестовых примеров, и каждый пример интерпретируется алгоритмом LIME, при этом к каждому примеру мы применяем одни и те же по смыслу изменения. Используя линейную модель  $g$ , в результате мы получаем матрицу  $W$ , строкой которой является номер примера, столбцом - позиция в векторе упрощенного представления, значением - вес данной позиции на данном примере.

Например, в случае модели, работающей с текстом и использующей bag-of-words, столбец матрицы  $W$  будет соответствовать номер слова в словаре. Однако с суперпикселями так не получится, поскольку на каждом тестовом примере суперпиксели разные по смыслу.

Получив матрицу  $W$ , мы можем рассчитать глобальный вес каждой позиции, найдя норму каждого столбца. Также мы можем попытаться выбрать небольшой набор строк матрицы  $W$  такой, чтобы в этом наборе для каждого столбца хотя бы раз встретилось большое значение. Этот набор строк соответствует набору тестовых примеров, которых предположительно может быть достаточно для глобальной интерпретации модели.

### 18.3. SHAP: Shapley Additive Explanation Values

В данном разделе мы рассмотрим подход SHAP (Lundberg and Lee, 2017), позволяющий оценивать важность признаков в произвольных моделях машинного обучения, а также может быть применен как частный случай метода LIME.

#### 18.3.1. Shapley values в теории игр

Теория игр - это область математики, изучающей взаимодействие (игру) между игроками, преследующими некие цели и действующими по некоторым правилам. Кооперативной игрой называется такая игра, в которых группа игроков (коалиция) действует совместно. С середины XX века (Shapley, 1952) известны так называемые Shapley values, которые позволяют численно оценить вклад каждого игрока в достижение общей цели.

**Определение (Shapley values).** Пусть существует характеристическая функция  $v$ , которая каждому множеству игроков сопоставляет число - эффективность данной коалиции игроков, действующей совместно. Тогда Shapley value для каждого игрока  $i$  - это число, рассчитываемое по достаточно простой формуле. Обозначим за  $\Delta(i, S)$  прирост эффективности от добавления игрока  $i$  в коалицию игроков  $S$ :

$$\Delta(i, S) = v(S \cup i) - v(S) \quad (1)$$

Пусть всего есть  $N$  игроков. Рассмотрим множество  $\Pi$  всех возможных упорядочиваний игроков, и обозначим за  $(\text{players before } i \text{ in } \pi)$  множество игроков, стоящих перед игроком  $i$  в упорядочивании  $\pi$ . Shapley value для игрока  $i$  рассчитывается таким образом:

$$\phi(i) = \frac{1}{n!} \sum_{\pi \in \Pi} \Delta(i, (\text{players before } i \text{ in } \pi)) \quad (2)$$

То есть мы считаем средний прирост эффективности от добавления  $i$ -го игрока в коалицию игроков, стоящих перед ним, по всем возможным упорядочиваниям игроков (количество элементов суммы равно  $N!$ ).

Формула (2) задается аксиоматически, то есть есть формулируется ряд необходимых свойств и доказывается, что данное решение является единственным, которое им удовлетворяет. Поскольку  $\Delta(i, S)$  не зависит от порядка игроков в  $S$ , то можно объединить равные друг другу слагаемые и переписать формулу (2) в следующем эквивалентном виде:

$$\phi(i) = \sum_{S \subseteq \{1, 2, \dots, N\} \setminus i} \frac{|S|!(|N| - |S| - 1)!}{N!} \Delta(i, S) \quad (3)$$

Формула (3) является взвешенной суммой по всем подмножествам игроков, не содержащих игрока  $i$ , в которой веса принимают наибольшие значения при  $|S| \approx 0$  или  $|S| \approx |N|$  и наименьшие значения при  $|S| \approx \frac{|N|}{2}$ .

#### 18.3.2. Shapley regression values

Shapley values можно применить в машинном обучении, если игроками считать наличие отдельных признаков, а результатом игры - ответ модели на конкретном примере  $x$ . Shapley values применяются в машинном обучении еще с XX века (Kruskal, 1987).

Shapley regression values (Lipovetsky and Conklin, 2001) позволяют оценить вклад каждого признака в ответ модели  $f$ . Зафиксируем конкретный тестовый пример  $x$  и обучающую выборку, и за характеристическую функцию множества признаков будем считать предсказание модели, обученной только на этих признаках:

$$v(S) = f_S(x_S) \quad (4)$$

Следовательно,  $\Delta(i, S)$  - изменение в предсказании  $x$  между моделью  $f_{S \cup \{i\}}$ , обученной на признаках  $S \cup \{i\}$  и моделью  $f_S$ , обученной на признаках  $S$ :

$$\Delta(i, S) = (f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)) \quad (5)$$

Тогда вклад отдельных признаков в величину предсказания модели можно оценивать по формулам (2) и (3). Отметим, что мы рассматриваем не вклад каждого признака в точность модели, а вклад каждого признака в величину предсказания модели на конкретном тестовом примере, что помогает интерпретировать это предсказание. Если модель предназначена для классификации, то в выходными значениями считаются logits - значения до операции softmax/sigmoid. Но можно рассматривать и вклад признаков в величину метрики качества, об этом см. в разделе "Диагностика сдвига данных с помощью SHAP loss values".

В целом есть некоторые проблемы в представлении признака как игрока. Дело в том, что для игрока существует лишь два состояния: либо он есть, либо его нет. Признак же существует как минимум в трех состояниях: два разных значения и неопределенное значение (отсутствие признака). В Shapley regression values сравнивается текущее значение признака на примере  $x$  с его полным отсутствием при обучении и тестировании.

Минусом такого подхода является высокая сложность вычислений: для расчета Shapley regression values нужно обучать модель на всех возможных подмножествах признаков, что в большинстве случаев невыполнимо. Однако мы можем намного быстрее посчитать приблизительное значение Shapley values, если будем считать не все элементы суммы (3), а лишь некоторые, обладающие большими весами. Для этого мы можем использовать веса как вероятности при семплировании элементов суммы (аппроксимация семплированием, Shapley sampling values - Štrumbelj and Kononenko, 2014).

### 18.3.3. SHAP values

Можем ли мы аппроксимировать Shapley regression values, обучая всего одну модель на всех признаках? В этом случае нам нужно получать предсказание модели в случаях, когда многие из признаков имеют неопределенные значения. Большинство моделей не умеют работать с такими данными, что является проблемой.

Применим статистический подход и будем считать, что обучающие и тестовые данные взяты из некоторого распределения вероятностей. Пусть часть признаков в примере  $x$  известны, часть пропущены. Обозначим за  $x_S$  известные признаки. В SHAP характеристическая функция множества признаков  $S$  для примера  $x$  и модели  $f$  задается как условное мат. ожидание:  $v(S) = E[f(x)|x_S]$ . Данная формула означает, что за  $v(S)$  мы берем мат. ожидание предсказания  $f$  на примерах  $x'$ , взятых из распределения данных, таких, что  $x'_S = x_S$ . О способе подсчета таких величин мы поговорим далее, но сначала дадим формальное определение SHAP values (Lundberg and Lee, 2017).

**Определение (SHAP values).** Пусть мы имеем модель  $f$ , распределение данных и некий тестовый пример  $x$  и хотим оценить важность текущих значений каждого признака по сравнению с их неопределенными значениями. SHAP (SHapley Additive exPlanation) values для признаков на примере  $x$  - это Shapley values, рассчитываемые для следующей кооперативной игры:

- Играками являются признаки (наличие  $i$ -го игрока означает текущее значение  $i$ -го признака на примере  $x$ , отсутствие  $i$ -го игрока означает неопределенное значение  $i$ -го признака - так же, как в Shapley regression values).
- Характеристической функцией  $v(S)$  коалиции признаков  $S$  является условное мат. ожидание  $E[f(x)|x_S]$  по распределению данных.

Таким образом, алгоритм расчета SHAP values следует формулам (2) и (3): для каждого возможного упорядочивания признаков мы берем все признаки, стоящие перед  $i$ -м признаком (обозначим их за  $S$ ) и считаем величину  $\Delta_f(i, S) = E[f(x)|x_S \cup i] - E[f(x)|x_S]$  (о способе подсчета см. далее), после чего усредняем полученные значения по всем упорядочиваниям. Это означает, что SHAP values описывают ожидаемый прирост выходного значения модели при добавлении  $i$ -го признака в текущем примере.

Наиболее важным свойством, которым обладают Shapley values, рассчитанные по формулам (2) и (3), является состоятельность (consistency). Это свойство означает, что если (записав пример  $x$ ) мы рассчитываем Shapley values для двух моделей  $f$  и  $f'$ , и для любых  $S, i$  верно, что вклад признака  $i$  при имеющемся множестве признаков  $S$  в первой модели не меньше, чем во второй, то Shapley value этого признака в первой модели также не меньше, чем во второй:

$$\forall S, i : \Delta_f(i, S) \geq \Delta_{f'}(i, S) \implies \phi_f(x) \geq \phi_{f'}(x)$$

Отличие SHAP values от Shapley regression values в том, что в последних характеристической функцией группы признаков  $x_S$  является значение  $f_{x_S}(x_S)$ , а в SHAP values -  $\mathbb{E}[f(x)|x_S]$ . В целом эти значения близки, так как  $f_{x_S}$  как правило моделирует  $\mathbb{E}[y|x_S]$ . Но Shapley regression values требуют многократного обучения модели и таким образом являются характеристикой обучаемой модели, тогда как SHAP values являются характеристикой обученной модели.

#### 18.3.4. Проблемы и ограничения SHAP values

На практике расчет SHAP values позволяет интерпретировать модель, выявлять скрытые проблемы в модели и данных и даже выполнять кластеризацию, что мы увидим далее в разделе "SHAP на практике". Однако такой подход имеет свои проблемы и ограничения. Проблемы SHAP можно поделить на три класса: вычислительные проблемы, проблемы ограниченной применимости и концептуальные проблемы.

- Вычислительные проблемы. Определение SHAP не говорит о том, как именно рассчитывать  $\mathbb{E}[f(x)|x_S]$  в условиях ограниченной по размеру выборки данных. Если бы мы имели прямой доступ к распределению  $\mathcal{D}$ , неограниченные вычислительные ресурсы и могли бы бесконечно семплировать из  $\mathcal{D}$ , то была бы возможна оценка  $\mathbb{E}[f(x)|x_S]$  с любой точностью. Однако обычно мы имеем лишь обучающую и тестовую выборки, взятые из  $\mathcal{D}$ . Их может быть недостаточно, чтобы надежно оценить  $\mathbb{E}[f(x)|x_S]$ . SHAP values можно искать либо введением различных упрощений (Kernel SHAP), либо для конкретных видов моделей (Tree SHAP), что мы рассмотрим в следующих разделах.
- Проблемы ограниченной применимости. Shapley values являются достаточно гибким инструментом, который можно применять во многих случаях для оценки вклада "участников процесса" в результат. Но SHAP values не являются универсальным способом интерпретации как минимум потому, что выходные данные могут иметь сложный формат, а входные признаки могут быть не интерпретируемы.
- Концептуальные проблемы возникают из вопроса о том, можно ли вообще SHAP values считать мерой важности признаков в модели? Дело в том, что SHAP values зависят не только от модели, но и от распределения данных, при этом даже признаки, которые никак не используются моделью, могут иметь ненулевые SHAP values. Об этом мы поговорим позже, так как этот вопрос требует отдельного раздела.

#### 18.3.5. Independent SHAP

Задачу оценки  $\mathbb{E}[f(x)|x_S]$ , можно сильно упростить, если предположить, что наличие каждого признака линейно и независимо влияет на ответ модели. Тогда значения признаков, не входящих в  $S$ , не зависят от  $x_S$  и друг от друга и линейно влияют на ответ, и  $\mathbb{E}[f(x)|x_S]$  можно рассчитать, заменив пропущенные значения их мат. ожиданиями, которые можно приблизить средним значением по выборке данных. Обозначив пропущенные признаки за  $\bar{S}$ , получим:

$$\mathbb{E}[f(x)|x_S] \approx f([x_S, \mathbb{E}[x_{\bar{S}}]])$$

В формуле стоит знак приблизительного равенства потому, что линейность и независимость - это лишь предположения: чем ближе они к истине, тем точнее аппроксимация. С помощью данной формулы мы можем рассчитать  $f(x_S)$  для любого подмножества признаков  $x_S$ , а значит получили возможность рассчитывать SHAP values на практике по формуле (3), дополнительно применяя также аппроксимацию семплированием.

Не является ли линейность и независимость влияния признаков чрезмерно грубым упрощением? В некоторых случаях возможно является, но вспомним, что в методе LIME мы использовали такое же допущение, обучая линейную объясняющую модель  $g$ . О связи SHAP и LIME мы поговорим далее.

#### 18.3.6. Kernel SHAP

Авторы сопоставляют SHAP с несколькими появившимися в 2010-х годах методами интерпретации моделей машинного обучения, которые используют локальную аппроксимацию, в первую очередь с LIME, а также с layer-wise relevance propagation (Bach et al., 2015) и activation difference propagation (DeepLIFT) (Shrikumar et al., 2017), которые мы сейчас не будем рассматривать.

Как LIME, так и DeepLIFT рассматривают локальное упрощенное представление в виде вектора  $z$  (где  $z_i = 0$  означает наличие изменения  $\Delta_i$  относительно примера  $x_0$ ). В разделе, посвященном LIME, мы говорили о таком представлении и об объясняющей модели  $g$ , аппроксимирующей  $f(h(z))$ . Авторы SHAP рассматривают случай, когда объясняющая модель линейна (такие способы интерпретации авторы

называют additive feature attribution methods):

$$g(z) = \phi_0 + \sum_{i=1}^M \phi_i z_i, \quad \phi_i \in \mathbb{R}, z_i \in \{0, 1\} \quad (6)$$

Важным является случай, когда  $\Delta_i$  означает удаление некоего признака из  $x_0$ , то есть при  $z_i = 0$  один из признаков в  $x_0$  заменяется на неопределенное значение. В этом случае будем говорить, что  $z_i$  задает наличие признака. При этом постановка задачи становится эквивалентной той, на которой основан SHAP.

Вспомним, что в LIME при выборе линейной модели  $g$  мы затем искали ее веса  $\phi_i$ , но для этого мы должны были выбрать функцию потерь  $\mathcal{L}$ , метрику сходства  $\pi$  и штраф сложности  $\Omega$ . Авторы SHAP выводят единственно возможные значения для  $\mathcal{L}$ ,  $\pi$  и  $\Omega$  такие, что полученные веса  $\phi_i$  будут равны SHAP values, рассчитанным по формуле (3), при обучении на всех возможных  $z$ . Если же мы будем обучать  $g$  не на всех  $z$ , то полученные веса будут аппроксимировать  $\phi_i(f, x_0)$ .

- $\mathcal{L}$  - среднеквадратичное отклонение
- $\Omega(g) = 0$
- $\pi(z) = \frac{M-1}{\binom{M}{|z|}|z|(M-|z|)}$ , где  $\binom{M}{|z|}$  - биномиальный коэффициент.

Основное отличие от примера, предложенного авторами LIME (Ribeiro et al., 2016), состоит именно в  $\pi(z)$ . Как мы видели в разделе "Объясняющая модель  $\pi(z)$  фактически является метрикой сходства между векторами, такие метрики часто называют ядрами (kernels), поэтому предложенная формула для  $\pi(z)$  получила название **Shapley kernel**, а метод назван **kernel SHAP**.

Shapley kernel назначает большие веса примерам как с большим количеством единиц, так и с большим количеством нулей. Это означает, что мы обращаем основное внимание на примеры с (почти) максимальным и (почти) минимальным количеством компонентов (игроков). На рис. 4 показано отличие Shapley kernel от L2 distance и cosine similarity, которые предлагалось использовать в LIME.

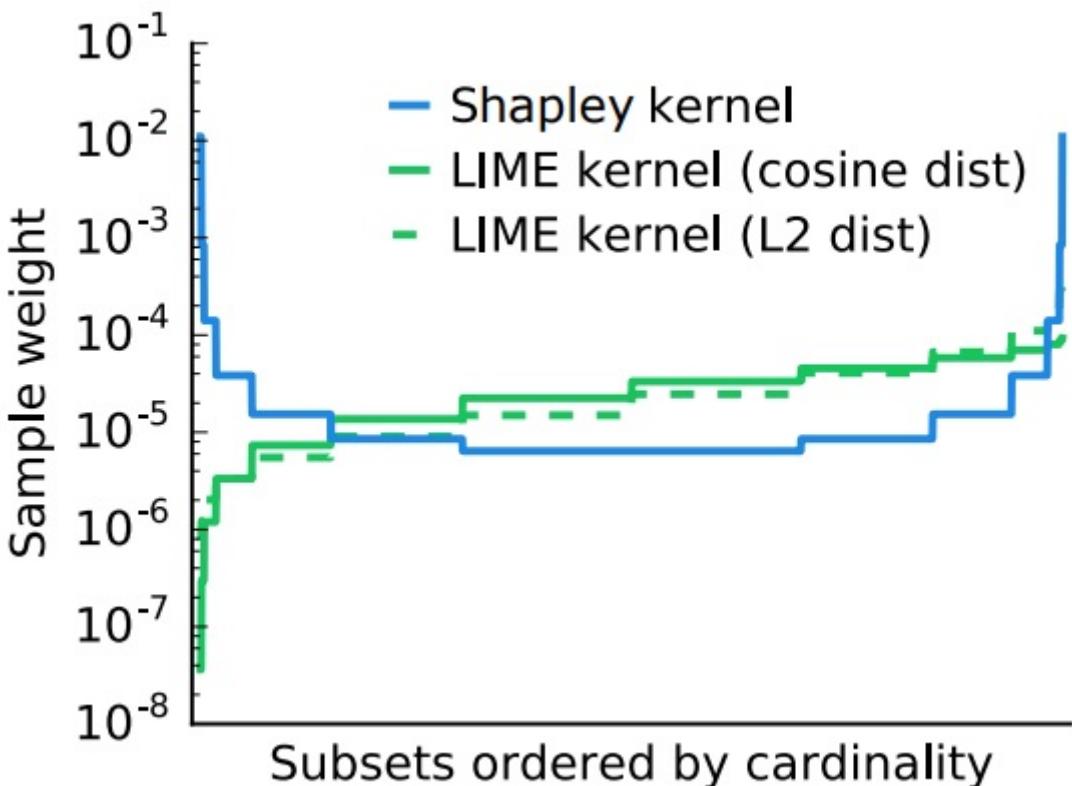


Рис. 4. Shapley kernel в сравнении с L2 distance и cosine similarity.

Формулировка метода Kernel SHAP не говорит о том, как именно рассчитывать или аппроксимировать  $\mathbb{E}[f(x)|x_S]$ . В частности, можно использовать Independent SHAP (в python-библиотеке shap за это отвечает параметр algorithm объекта shap.KernelExplainer).

Таким образом,

- **Kernel SHAP** - это способ приблизительного расчета суммы (3) подсчетом не всех, а лишь части ее элементов

- **Independent SHAP** - это способ приблизительного расчета элемента суммы (3), т. е.  $\mathbb{E}[f(x)|x_S]$

Как видим, Kernel SHAP и Independent SHAP конкретизируют разные подзадачи и могут быть использованы совместно.

### 18.3.7. Tree SHAP

SHAP values основаны на условном мат. ожидании  $\mathbb{E}[f(x)|x_S]$ , но не всегда понятно, как посчитать это значение (см. раздел "SHAP values"). Модели, основанные на решающих деревьях (градиентный бустинг, случайный лес) позволяют работать с произвольным количеством пропусков в данных, поэтому в таких моделях можно напрямую рассчитать значение  $\mathbb{E}[f(x)|x_S]$ , сделав предсказание на признаках  $x_S$ .

Получить предсказание решающего дерева для примера с пропущенными значениями можно следующим образом. Рассмотрим пример  $x$  и дерево  $D$ , которое начинается разделяющим правилом, ведущим к двум поддеревьям  $D_1$  и  $D_2$ . Если данное правило основано на признаке, который известен, тогда на основе значения признака мы выберем нужное поддерево  $D_i(x)$  и возьмем его ответ:  $D(x) = D_i(x)$ . Если же признак пропущен, то в качестве ответа возьмем величину  $D(x) = (c_1 D_1(x) + c_2 D_2(x))$ , где  $c_1$  и  $c_2$  - доли обучающих примеров, которые попали в первое и во второе дерево. Таким образом мы аппроксимируем значение  $\mathbb{E}[f(x)|x_S]$  с помощью обучающей выборки.

SHAP values можно посчитать по формуле (3), но количество слагаемых экспоненциально зависит от количества признаков, и алгоритм расчета SHAP values в общем случае является NP-полным. Однако если ограничиться только решающими деревьями, то для расчета формулы (3) существует алгоритм с полиномиальной сложностью низкого порядка, предложенный в Lundberg et al., 2018 и реализованный в python-библиотеке shap.

## 18.4. SHAP на практике

Для подсчета SHAP values существует python-библиотека shap, которая может работать со многими ML-моделями (XGBoost, CatBoost, TensorFlow, scikit-learn и др) и имеет документацию с большим количеством примеров. С помощью библиотеки SHAP можно строить различные схемы и графики, описывающие важность признаков в модели и их влияние на ответ. Рассмотрим примеры из документации.

### 18.4.1. Waterfall plot

На рис. 5 показан waterfall plot, объясняющий предсказание на первом тестовом примере из датасета Boston housing. SHAP values получены с помощью метода Tree SHAP. Схема читается снизу вверх, и признаки упорядочены по возрастанию их SHAP values. Например, SHAP value  $-0.43$  для признака CRIM (имеющего значение  $0.006$ ) говорит о том, что значение CRIM=0.006 на данном примере уменьшает величину предсказания модели, по сравнению с отсутствием признака CRIM, при произвольном наличии других признаков (см. "Shapley values в теории игр").

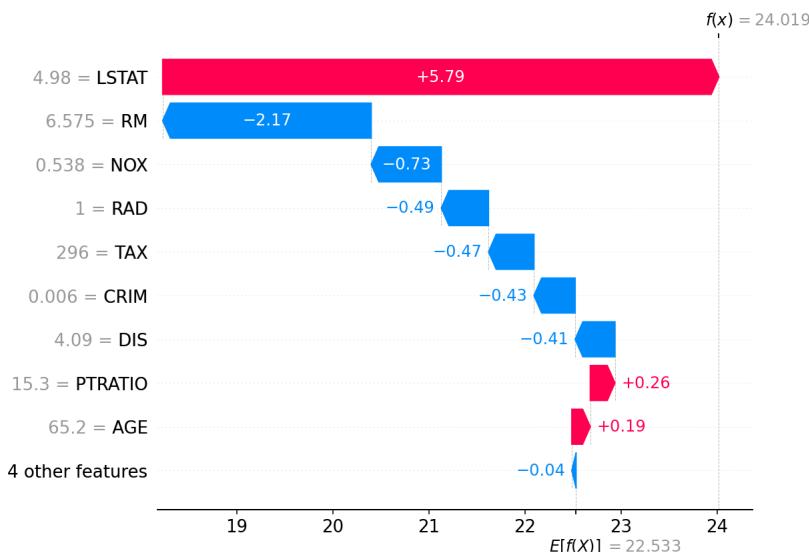


Рис. 5. Waterfall plot.

#### 18.4.2. Summary plot

Рассчитав SHAP value для каждого признака на каждом примере с помощью `shap.Explainer` или <https://shap-lrjball.readthedocs.io/en/latest/generated/shap.KernelExplainer.html> (есть и другие способы, см. документацию), мы можем построить summary plot, то есть summary plot объединяет информацию из waterfall plots для всех примеров.

На рис. 6 summary plot построен для модели, обученной на датасете, описывающем влияние различных медицинских анализов на вероятность смерти в течение следующих 12 лет (Cox et al., 1997). Рисунок взят из Lundberg et al., 2019. SHAP values при этом получены с помощью метода Tree SHAP.

Каждая горизонтальная линия соответствует одному признаку, и на этой линии отмечаются точки, соответствующие тестовым примерам: координата точки на линии соответствует SHAP value, цвет точки - значению признака. Если в каком-то участке линии не хватает места для всех точек, линия начинает расти в ширину. Таким образом, для каждого признака схема представляет собой слившееся множество точек, по одной точке для каждого примера.

Слева на рис. 6 показана важность признаков, рассчитанная как средний модуль величины SHAP values для данного признака.

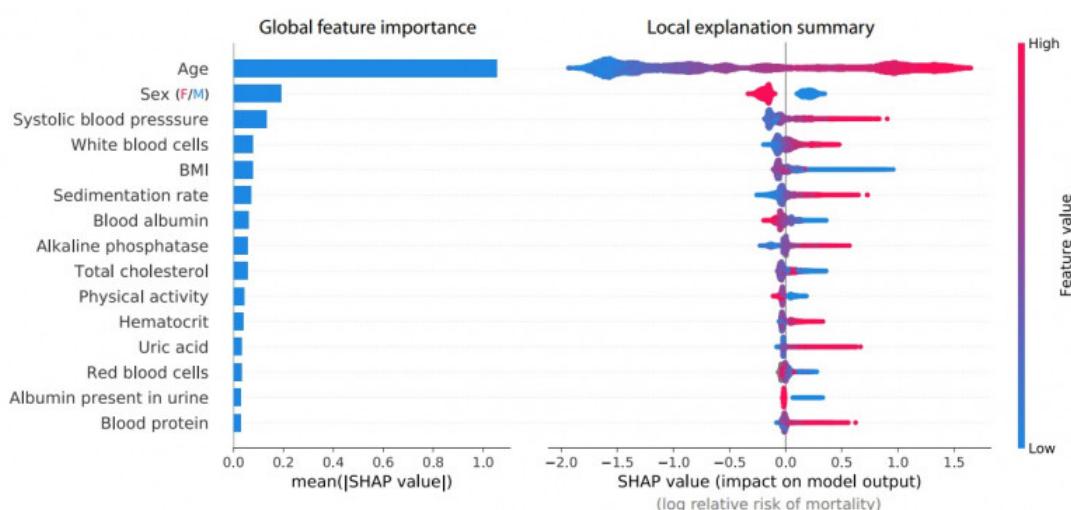


Рис. 6. Summary plot.

Очевидное влияние на риск смерти имеет возраст, а также пол (женщины среднестатистически живут дольше). Мы видим, что чем больше возраст (красный цвет), тем большее SHAP value (горизонтальная ось) назначается этому признаку. Большое значение SHAP value, в свою очередь, означает, что удаление этого признака (замена значения возраста на неопределенное) существенно уменьшит предсказанную вероятность смерти в течение 12 лет.

Можно также заметить, что большинство SHAP values, соответствующих медицинским анализам (строки 3 и далее), имеют либо окончательные, либо положительные значения SHAP values. Это означает, что есть много значений анализов, увеличивающих предполагаемый риск смерти (для данной модели), но нет таких значений, которые бы его сильно уменьшали.

Одно из преимуществ SHAP summary plot по сравнению с глобальными методами оценки важности признаков (такими, как mean impurity decrease или permutation importance) состоит в том, что на SHAP summary plot можно различить 2 случая:

- признак имеет слабое влияние, но во многих примерах
- признак имеет большое влияние, но в немногих примерах

#### 18.4.3. Dependence plot

На рис. 7C мы видим совместное распределение SHAP value для признака Systolic blood pressure и значения этого признака по тестовому датасету. Фактически это та же самая информация, что показана в summary plot для этого признака. На рис. 7D показаны SHAP interaction values (их способ расчета мы не рассматривали в этом обзоре) для пары признаков: Systolic blood pressure + Age. На рис. 7B показана схема (SHAP dependence plot), объединяющая информацию из схем на рис. 7C и 7D. Рисунки взяты из Lundberg et al., 2019.

Как можно видеть, совместное распределение на рис. 7B имеет "два хвоста". Для возраста 30-40 (синий цвет) систолическое давление 160+ mmHg повышает предсказанную вероятность смерти в течение 12 лет,

тогда как для возраста 60-70+ это же давление является нормальным (соответствующим возрасту) и дополнительно не повышает вероятность смерти в течение 12 лет.

Следует помнить, что при этом мы изучаем обученную модель, а не только данные. Модель может ошибаться, и тогда выводы о целевой зависимости, сделанные на основе SHAP values, тоже будут неверны.

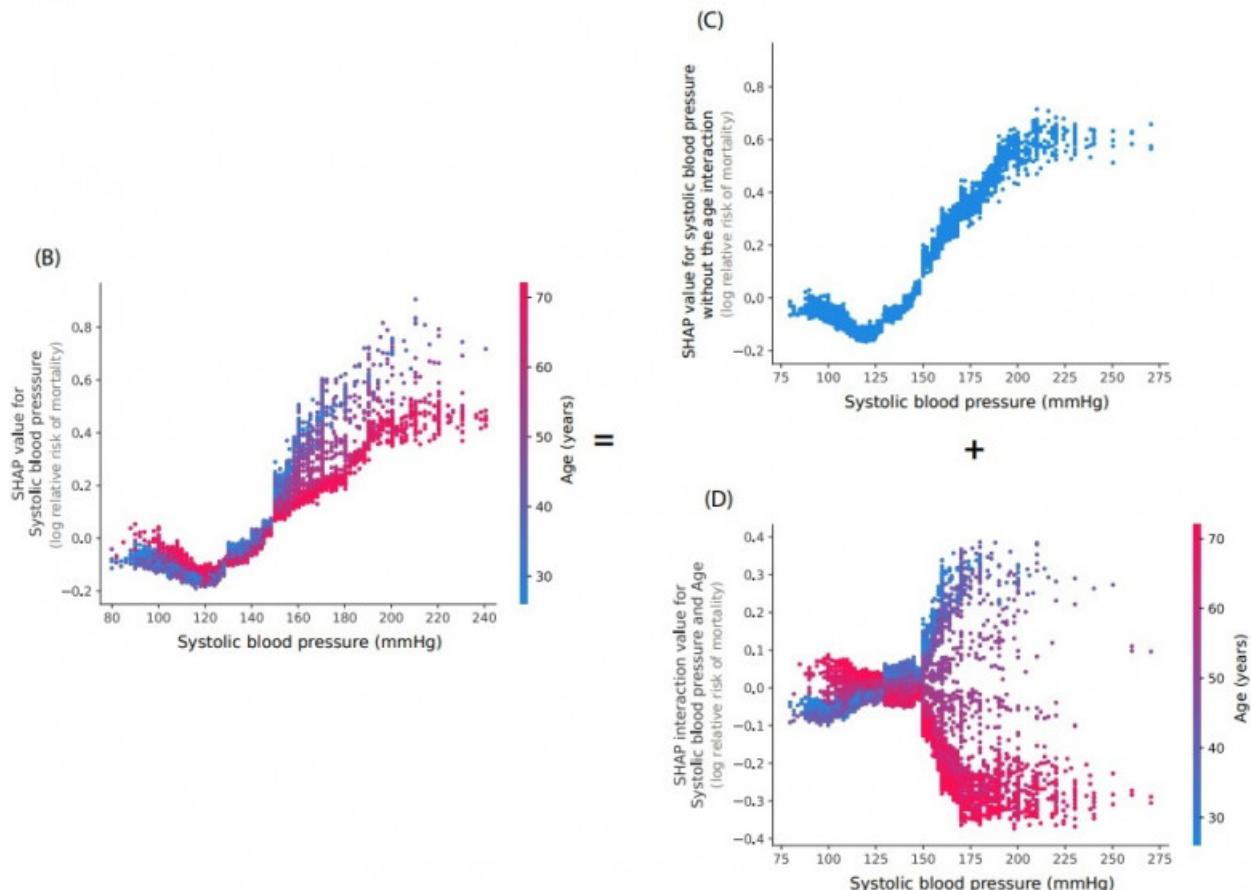


Рис. 7. Dependence plot.

#### 18.4.4. Диагностика сдвига данных с помощью SHAP loss values

Сдвиг данных является серьезной проблемой в машинном обучении. Он может быть вызван недостаточным разнообразием обучающих данных, изменениями в методах расчета признаков и ошибками в их обработке после развертывания модели.

Для диагностики сдвига данных можно использовать Shapley values, считая результатом кооперативной игры не предсказание модели, а функцию потерь. Тогда мы сможем определить, какие признаки вносят положительный или отрицательный вклад в точность модели.

В работе "Explainable AI for Trees" (Lundberg et al., 2019) авторы проводят следующий эксперимент. В качестве данных используется датасет, целевым признаком в котором является длительность процедуры анестезии перед операцией. Датасет содержит данные за 4 года (2185 признаков и ок. 147000 пациентов). Данные за первый год используются для обучения, данные за следующие три года - для тестирования (симуляции работы системы по назначению). Отметим, что это не временной ряд: каждый пример независим.

График метрики качества на тестовых данных (в зависимости от даты) показан на рис. 8. Такие графики используют для мониторинга качества работы системы. Метрика качества на обучающих данных (первый год) намного лучше, что естественно.

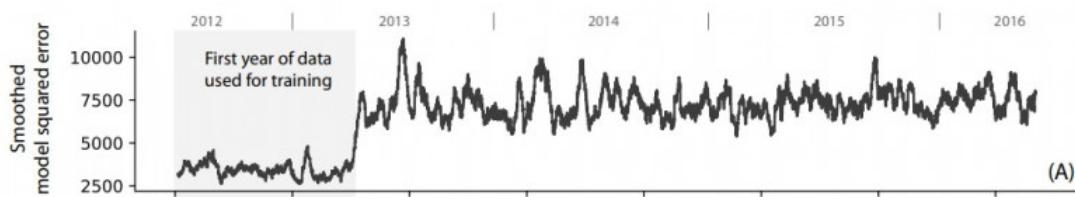


Рис. 8. Метрика качества на обучающих и тестовых данных в датасете "hospital procedure duration".

На каждом тестовом примере можно рассчитать SHAP loss values, считая игроками наличие признаков, а характеристической функцией набора признаков - точность предсказания на данном примере. Поскольку каждый тестовый пример соответствует определенной дате, то мы можем построить для каждого признака график зависимости Shapley loss values от даты. На Рис. 9 показаны такие графики для трех бинарных признаков (цветом обозначено значение признака).

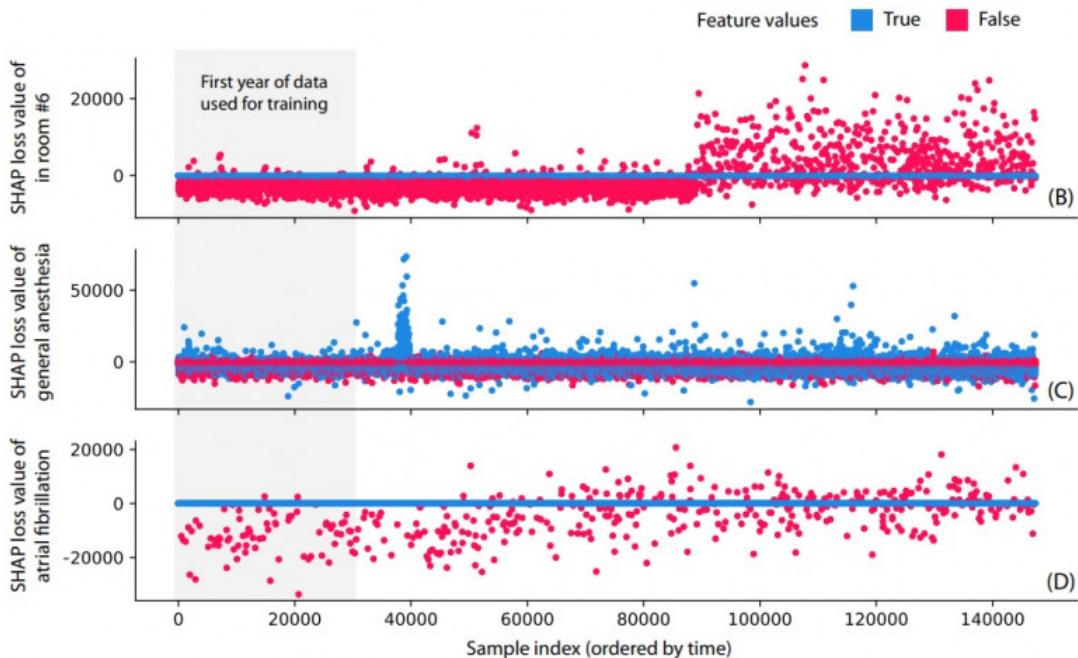


Рис. 9. Мониторинг SHAP loss values обнаруживает проблемы, не видимые по графику метрики качества, в датасете "hospital procedure duration".

Первый признак "in room #6" означает, что процедура проводилась в комнате номер 6. При обучении этот признак понижал ошибку предсказания времени процедуры, но при симуляции работы по назначению с какого-то момента стал резко повышать ошибку, то есть наличие этого признака стало вредным. Причина кроется в том, что в коде обработки признаков была допущена ошибка: перепутаны номера комнат 6 и 13. Данная ошибка была допущена исследователями намеренно, чтобы показать эффективность SHAP loss values в обнаружении подобных проблем. При этом на графике функции потерь этой проблемы не видно, так как она затрагивает лишь небольшой процент данных.

Следующие две проблемы, напротив, были ненамеренными и обнаружены случайно. Признак "general anesthesia" в определенный момент резко снижал точность предсказания. Как выяснилось, проблема была связана с ошибкой в конфигурации электронного оборудования.

На признаке "atrial fibrillation" виден дрейф значений SHAP loss values: со временем этот признак становится все менее полезным, и в итоге начинает понижать точность предсказания времени процедуры. Как выяснилось, это вызвано изменениями в длительности процедуры аблации фибрилляции предсердий, которая связана с изменениями в технологиях и персонале.

#### 18.4.5. Supervised-кластеризация данных с помощью SHAP

Из SHAP values можно составить SHAP-вектор для каждого обучающего или тестового примера. Lundberg et al., 2018 заметили, что SHAP-векторы можно использовать для кластеризации данных намного эффективнее, чем векторы исходных признаков.

Задача кластеризации данных достаточно нетривиальна. Как правило, мы не можем выполнить кластеризацию просто на основе евклидова расстояния между векторами признаков, так как признаки различаются по важности, имеют разный масштаб, и некоторые признаки могут дублировать друг друга.

Элементы SHAP-вектора соответствуют отдельным признакам, но при этом каждый элемент SHAP-вектора представлен в одной и той же шкале. При этом величина SHAP value означает важность признака (в контексте задачи предсказания целевого признака). Исходя из этого, можно выполнять supervised-кластеризацию в пространстве SHAP-векторов.

На рис. 10 показана кластеризация с помощью SHAP на датасете, описывающем риск смерти в течение 12 лет на основе медицинских анализов (об этом датасете см. также раздел "Summary plot"). Каждый столбец представляет собой SHAP-вектор для одного пациента, и эти векторы упорядочены (локально) по сходству друг с другом с помощью алгомеративной иерархической кластеризации и (глобально) по значению целевого признака, объединяясь в кластеры людей со сходной симптоматикой.

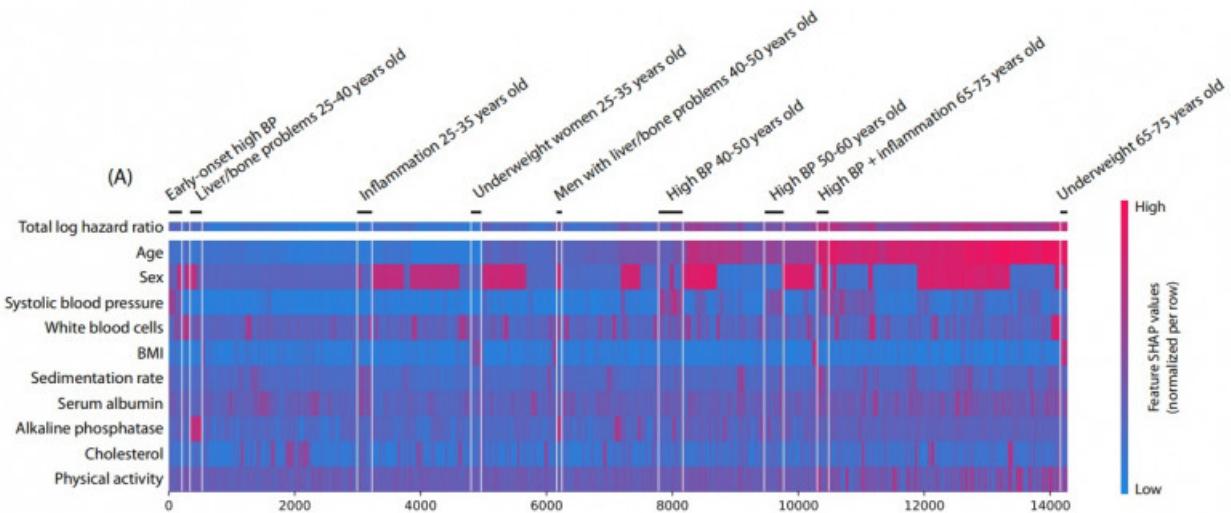


Рис. 10. Кластеризация данных с помощью SHAP values.

Другой пример из Lundberg et al., 2018 изображен в трехмерном формате (рис. 11). В данном случае показаны примеры из датасета, в котором целевой переменной является вероятность месячного заработка больше \$50K. Примеры здесь тоже кластеризованы с помощью их SHAP-векторов ("sorted by explanation similarity"), то есть мы ищем группы примеров, предсказания на которых объясняются схожим образом. Полученным кластерам даны текстовые описания.

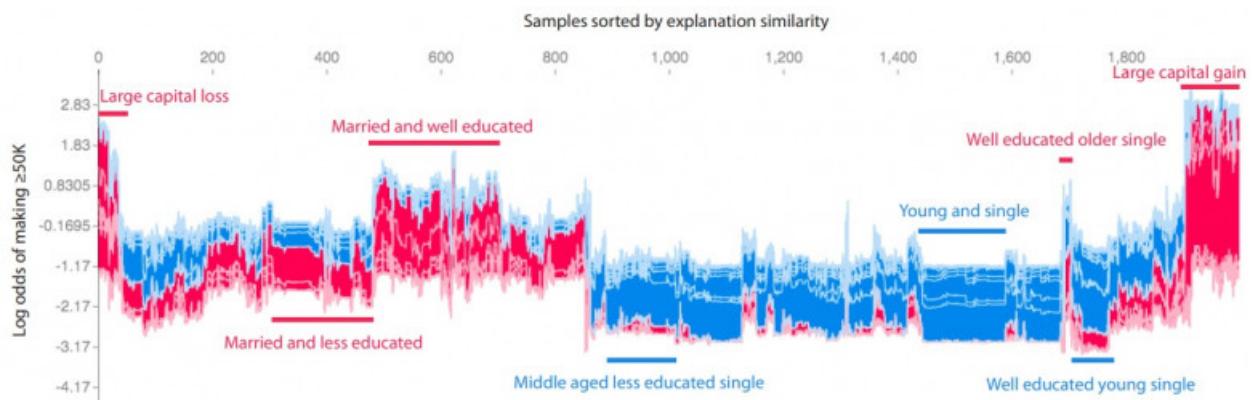


Рис. 11. Еще один пример кластеризации данных с помощью SHAP values.

## 18.5. Проблемы SHAP values

Можно ли SHAP values считать адекватной мерой важности признаков в модели? Как мы увидим далее, SHAP values может быть не равно нулю даже для тех признаков, которые никак не используются в модели.

Если рассуждать более глобально, то проблема SHAP values в том, что выбор характеристической функции  $v(S) = \mathbb{E}[f(x)|x_S]$  является не единственным возможным, существуют и другие варианты, и каждый вариант может быть по-своему хорош или плох в разных ситуациях.

### 18.5.1. SHAP values в условиях взаимной зависимости признаков

Ранее мы рассматривали определение SHAP values и видели, что значение  $\mathbb{E}[f(x)|x_S]$  может либо точно рассчитываться (в Tree SHAP для решающих деревьев), либо аппроксимироваться (в Independent SHAP).

**tree SHAP** рассчитывает непосредственно SHAP values:  $v(S) = \mathbb{E}[f(x)|x_S]$ , аппроксимируя их с помощью обучающей выборки. При этом значение  $v(S)$  зависит не только от модели  $f$ , но и от распределения данных. Таким образом, SHAP values являются характеристикой не только модели, а объединенной системы "модель + данные". На разных распределениях данных SHAP values для одной и той же модели может оказаться разной.

Пусть два признака А и В коррелируют, при этом модель использует для предсказания только признак А. Признак В влияет на мат. ожидание признака А, а отсюда и на мат. ожидание ответа модели, следовательно признак В может иметь ненулевое значение SHAP value. Этот признак никак не используется моделью, но он позволяет предсказывать значения других признаков и таким образом влияет на ожидаемый ответ модели. Более подробный пример можно найти в Janzing et al., 2019, см. Example 1. В

целом если два признака всегда равны друг другу, то SHAP values для них тоже будут равны, даже если модель использует лишь один из этих признаков.

Таким образом, не используемые моделью признаки могут иметь ненулевые SHAP values, потому что распределение данных таково, что эти признаки позволяют предсказать ожидаемый ответ модели. Поэтому интерпретировать SHAP values как важность признаков в модели не всегда корректно.

**Independent SHAP** аппроксимирует SHAP values следующим образом:  $v(S) = f([x_S, \mathbb{E}[x_{\bar{S}}]])$ . При этом зависимость  $v(S)$  от распределения данных сохраняется, но в Independent SHAP те признаки, которые не используются моделью, всегда будут иметь нулевые SHAP values. С другой стороны, игнорирование зависимости признаков друг от друга в Independent SHAP может привести к тому, что определенные сочетания признаков  $[x_S, \mathbb{E}[x_{\bar{S}}]]$  будут необычны или вовсе некорректны, что означает, что мы будем изучать предсказания модели на данных за пределами обучающего распределения (out-of-distribution).

### 18.5.2. Соблюдение границ многообразия данных

Множество реальных данных как подмножество  $X$  часто называют **многообразием данных** (Carlsson, 2009; Fefferman et al., 2013). Это не обязательно многообразие в математическом смысле, но что-то общее между ними есть. В математике многообразие - это подмножество некоего пространства, являющееся локально евклидовым, то есть в нем можно ввести локальные координаты и, изменяя их, "путешествовать" по многообразию. Имея набор данных с большим количеством признаков часто можно предположить, что все значения признаков обусловлены небольшим количеством независимых факторов (см. факторный анализ). Изменяя значения этих факторов, можно "путешествовать" по многообразию данных, не выходя за его пределы. В данных, как правило, есть случайный шум, поэтому данные можно считать лежащими не на многообразии, а в окрестности многообразия.

В предыдущем разделе мы рассматривали проблему ненулевых SHAP values для признаков, которые не используются в модели. Часто эту проблему рассматривают под таким углом (Chen et al., 2020): при локальной интерпретации модели  $f$  в окрестности примера  $x_0$  стоит ли ограничиться изучением поведения модели в рамках многообразия данных (Tree SHAP), или нужно также рассматривать и поведение модели на таких изменениях  $x_0$ , которые дают нереалистичные примеры (Independent SHAP)?

Справедливо ради надо отметить, что не всегда следует избегать выхода за пределы многообразия обучающих данных. Конечно, при этом мы можем получить совершенно невозможные сочетания признаков, и на таких сочетаниях от модели не требуется корректная работа. За пределами многообразия реальных данных модель может вести себя как угодно (рис. 12), и ее интерпретация в этих областях неинформативна. Но с другой стороны, существует проблема сдвига данных, когда распределение данных меняется.

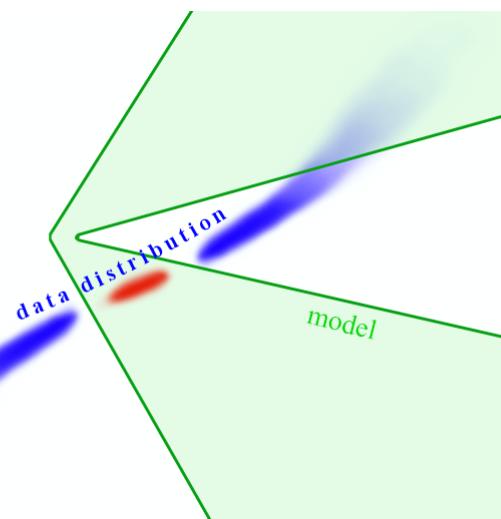


Рис. 12. Модель может хорошо работать в пределах распределения данных, но вести себя необычно за его пределами. В данном примере один из "хвостов" распределения данных не попал в обучающую выборку из-за ее ограниченного разнообразия, из-за чего модель неверно работает в этой области (проблема сдвига данных). Однако есть такие области, в которых поведение модели нас вообще не интересует, так как таких данных не бывает.

**Резюме.** Выходя за пределы многообразия обучающих данных, мы можем получить нереалистичные или некорректные данные, но оставаясь на многообразии, мы можем получить сильную взаимосвязь признаков, что создает сложности в оценке их независимого влияния на предсказания. Более подробно об этой проблеме можно почитать в статье "True to the Model or True to the Data?"(Chen et al., 2020).

## 18.6. Shapley Flow

Shapley Flow (Wang et al., 2020) - это еще один метод интерпретации предсказания модели на конкретном примере, являющийся обобщением метода SHAP. Его можно применять тогда, когда о многообразии данных известна какая-то информация. Рассмотрим алгоритм действий в методе Shapley Flow.

Программную реализацию Shapley Flow можно найти в данном репозитории, в нем же есть тьюториал.

### 18.6.1. Использование метода Shapley Flow

**18.6.1.1. Шаг 1.** Строится граф причинно-следственных связей (causal graph) во входных данных (рис. 13). Вершинами графа являются признаки, направленные ребра обозначают наличие причинно-следственной связи. К этому графу добавляется еще одна вершина - предсказание модели. Этот граф используется как входные данные для алгоритма Shapley Flow.

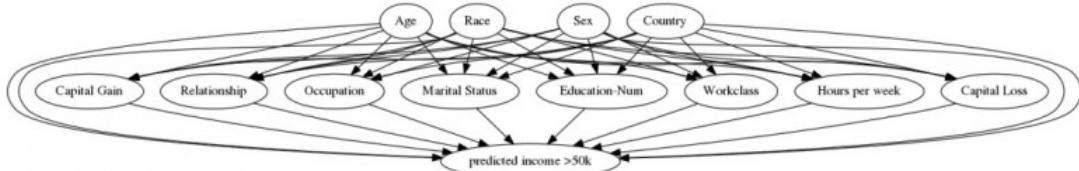


Рис. 13. Граф причинно-следственных связей в данных. Также к графу добавляется вершина, из которой идут ребра во все остальные вершины: эта вершина означает случайные эффекты, она не показана на рисунке для лучшей читаемости. Данный граф причинно-следственных связей является упрощенным и используется лишь для иллюстрации работы метода.

**18.6.1.2. Шаг 2.** Выбираются два примера:  $x_0$  и  $x_{background}$ . Метод Shapley Flow нацелен на интерпретацию различия в предсказаниях между этими примерами, основываясь на различии исходных признаков, то есть интерпретируется  $\Delta y = f(x_0) - f(x_{background})$ . Например, если модель работает с медицинскими данными, то в качестве  $x_{background}$  можно взять типичные значения анализов для какой-то болезни. Также авторы упоминают о том, что Shapley Flow позволяет использовать в качестве  $x_{background}$  одновременно несколько значений или распределение значений.

**18.6.1.3. Шаг 3.** Алгоритм Shapley Flow оценивает вклад в  $\Delta y$  для каждого ребра в графе. Вклад каждой вершины (признака) при этом считается как сумма вкладов выходящих из нее ребер. Пример результата показан на рис. 14 (показаны 10 ребер с наибольшими по модулю вкладами).

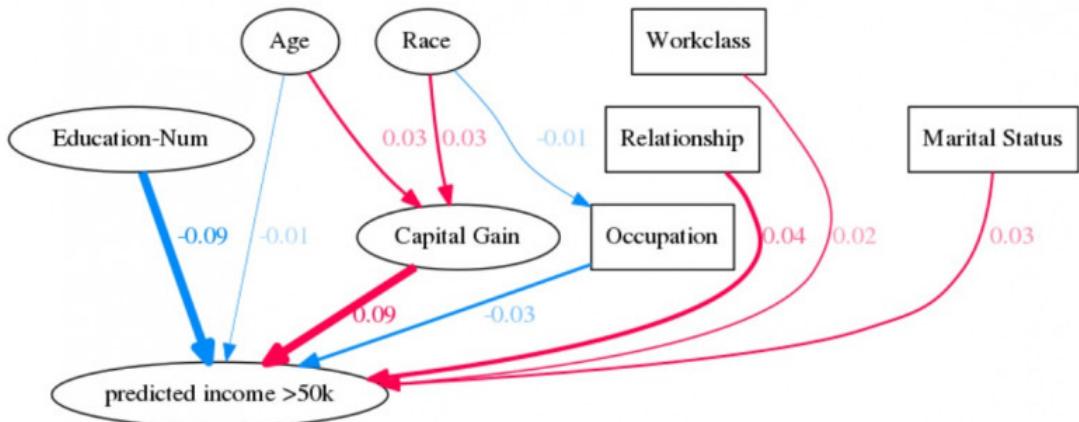


Рис. 14. Иллюстрация работы метода Shapley Flow. Показаны 10 ребер с наибольшими по модулю вкладами.

### 18.6.2. Принцип работы метода Shapley Flow

Здесь мы не будем углубляться в детали метода Shapley Flow, но рассмотрим общую идею. Согласно авторам метода Shapley Flow, дилемма "interventional" против "conditional" интерпретации связана с выбором "границ объяснения"(boundary of explanation). Границей объяснения называется деление причинно-следственного графа на две части  $D$  и  $F$  таким образом, что ребра могут идти из  $D$  в  $F$ , но не могут идти обратно. При этом  $F$  можно рассматривать как модель, а  $D$  - ее входные данные.

На рис. 15 показана простая модельная задача (Pearl and Russell, 2000).

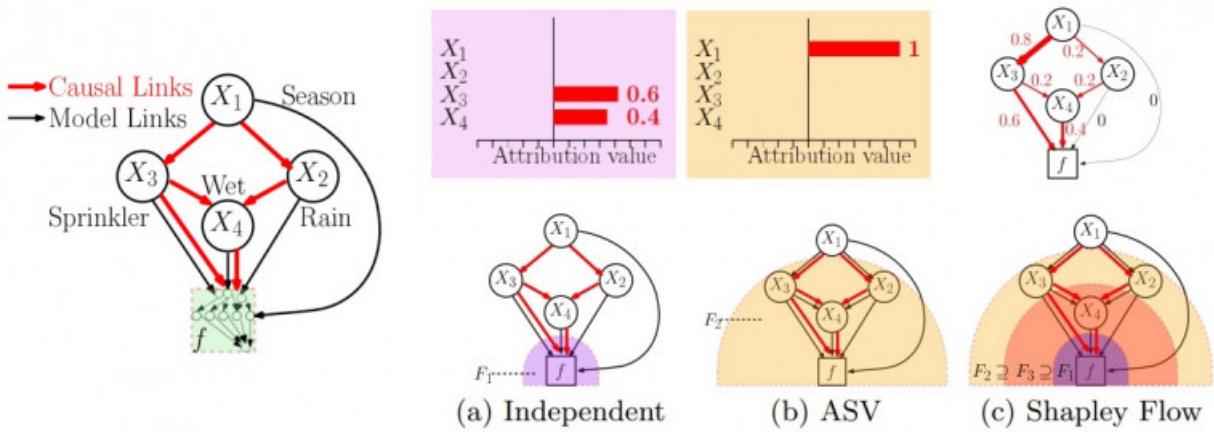


Рис. 15. Сравнение методов SHAP, Assymmetric SHAP и Shapley Flow на модельной задаче.

В задаче есть пять переменных:

- $X_1$  - сезон года
- $X_2$  - идет ли дождь?
- $X_3$  - включен ли разбрызгиватель воды?
- $X_4$  - сырое ли на улице?
- $Y$  - скользко ли на улице?

Красными стрелками показаны причинно-следственные статистические связи между признаками. С определенной долей вероятности можно утверждать следующее:

1.  $X_1 \rightarrow X_3$ : если сезон сухой, то разбрызгиватель включён
2.  $X_1 \rightarrow X_2$ : если сезон влажный, то идёт дождь
3.  $X_3 \rightarrow X_4$ : если разбрызгиватель включён, то покрытие мокрое
4.  $X_2 \rightarrow X_4$ : если идёт дождь, то покрытие мокрое
5.  $X_4 \rightarrow Y$ : если покрытие мокрое, то на улице скользко

Модель  $f$  предсказывает значение  $Y$ , используя все четыре переменные  $X_1, X_2, X_3, X_4$ . При этом модель  $f$  делает предсказание только на основе  $X_3$  и  $X_4$  (которые коррелируют), но это заранее не известно. Нужно определить, от чего зависит ответ модели  $f$ . При этом график причинно-следственных связей известен.

На рис. 15 (а-с) проиллюстрирована работа трех методов: Independent SHAP, Assymmetric Shapley values (ASV, Frye et al., 2019) и Shapley Flow. Метод Independent SHAP относится к классу "interventional" (см. предыдущий раздел) и не учитывает границы многообразия данных: признаки в нем считаются независимыми. Этот метод назначит ненулевой вклад только признакам  $X_3$  и  $X_4$ . Его можно рассматривать как "interventional" по отношению к "границе объяснения"  $F_1$ , то есть вершины, имеющие непосредственные связи с  $F_1$ , изменяются независимо, и таким образом изучается поведение модели.

Метод Assymmetric Shapley values является обобщением SHAP, которое позволяет использовать информацию о причинно-следственных связях в исходных данных. Его можно рассматривать как "interventional" по отношению к "границе объяснения"  $F_2$ , то есть вершины, имеющие непосредственные связи с  $F_2$  (а это только вершина  $X_1$ ), изменяются независимо. Этот метод назначит ненулевой вклад только вершине  $X_1$ .

Авторы Shapley Flow стремятся объединить оба подхода. При этом рассматриваются все возможные "границы объяснения" и важность вклада назначается не вершинам, а ребрам. Таким образом, во-первых мы видим признаки, которые непосредственно влияют на ответ модели ( $X_3$  и  $X_4$ ), а во-вторых видим признаки, которые влияют косвенно ( $X_1$ ) и тем самым могут объяснить наличие этих значений в данном примере.

Алгоритм Shapley Flow, как и Shapley values, задается аксиоматически, и доказывается единственность решения. Если SHAP основан на Shapley values, то Shapley Flow основан на Owen values - расширении понятия Shapley values. Более детальное описание этого метода можно найти в Wang et al., 2020.

## 18.7. Заключение

Интерпретация моделей машинного обучения не является простой задачей, поэтому существенная часть этого обзора была посвящена различным проблемам при применении методов LIME и SHAP.

В **LIME** мы пытаемся найти локальную аппроксимацию модели (обычно линейную), но получившаяся аппроксимация может быть неточной и иногда вводить в заблуждение.

**SHAP** направлен на интерпретацию моделей, но сами результаты этого метода не всегда легко интерпретировать. SHAP values не имеют простого и интуитивно понятного определения, и при этом имеют разновидности (Tree SHAP, Independent SHAP, Kernel SHAP, Deep SHAP), которые существенно отличаются:

- **Tree SHAP** позиционируется как "high-speed exact algorithm однако именно в нем результаты зависят от распределения данных, что континтуитивно. При этом признаки, которые вообще не используются моделью, могут иметь ненулевые SHAP values.
- **Independent SHAP** не имеет такой проблемы, но работает намного медленнее и иногда оценивает работу модели на некорректных сочетаниях признаков.

Несмотря на это, SHAP values могут быть очень полезны и могут помочь не только в интерпретации предсказаний, но и в отладке работы модели в условиях сдвига данных, а также в кластеризации данных.

**Shapley Flow** является наиболее общим, но и наиболее сложным методом, в котором требуется строить граф причинно-следственных связей в данных. Этим по-видимому и вызвана его пока что малая распространенность на практике.

## 19. Лекция 10. Интерпретация моделей: SHAP и LIME

### 19.1. Практика

#### 19.1.1. Загружаем данные

Хотим спрогнозировать энергоэффективность зданий в Нью-Йорке (Energy Star Score)

```
train_features = pd.read_csv('training_features.csv')
test_features = pd.read_csv('testing_features.csv')
train_labels = pd.read_csv('training_labels.csv')
test_labels = pd.read_csv('testing_labels.csv')

train_features.head()
```

#### 19.1.2. Используем модель с уже найденными гиперпараметрами

```
# Create an imputer object with a median filling strategy
imputer = SimpleImputer(strategy='median')
```

```
# Train on the training features
imputer.fit(train_features)
```

```
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
```

```
# Sklearn wants the labels as one-dimensional vectors
y = np.array(train_labels).reshape((-1,))
y_test = np.array(test_labels).reshape((-1,))
```

```
model = GradientBoostingRegressor(
    max_depth=5,
    max_features=None,
    min_samples_leaf=6,
    min_samples_split=6,
    n_estimators=800,
    random_state=42
)
```

```
model.fit(X, y)
model_pred = model.predict(X_test)
```

```
print(f'Final Model Performance on the test set: MAE={mae(y_test, model_pred)}')
```

#### 19.1.3. Стандартный feature\_importances

Мы можем посмотреть стандартный feature\_importances:

```
# Extract the feature importances into a dataframe
feature_results = pd.DataFrame(
    {'feature': list(train_features.columns),
     'importance': model.feature_importance_})

```

```
# Show the top 10 most important
```

```
feature_results = feature_results.sort_values('importance', ascending=False).reset_index()
feature_results.head(10)
```

Но чем это плохо? Мы не видим что происходит при изменении конкретного признака, как меняется предсказание, как зависит оно от конкретной фичи.

#### 19.1.4. Нарисует то же самое в виде гистограммы

```
figsize(8, 6)
plt.style.use('fivethirtyeight')

# Plot the 10 most important features in a horizontal bar chart
feature_results.loc[:9, :].plot(
    x='feature',
    y='importance',
    edgecolor='k',
    kind='barh',
    color='blue'
)
plt.xlabel('Relative Importance', size=20)
plt.ylabel('')
plt.title('Feature Importances from Random Forest', size=30)
```

#### 19.1.5. Оставляем только 10 самых важных признаков

```
# Extract the names of the most important features
most_important_features = feature_results['feature'][:10]

# Find the index that corresponds to each feature name
indices = [list(train_features.columns).index(x) for x in most_important_features]

# Keep only the most important features
X_reduced = X[:, indices]
X_test_reduced = X_test[:, indices]

print('Most_important_training_features_shape:', X_reduced.shape)
print('Most_important_testing_features_shape:', X_test_reduced.shape)
```

#### 19.1.6. Обучение линейной модели

Давайте возьмём линейную модель и попробуем её обучить на всех признаках и только на тех, которые мы отобрали:

```
lr = LinearRegression()

# Fit on full set of features
lr.fit(X, y)
lr_full_pred = lr.predict(X_test)

# Fit on reduced set of features
lr.fit(X_reduced, y)
lr_reduced_pred = lr.predict(X_test_reduced)

# Display results
print(f'Linear_Regression_Full_Results: MAE={mae(y_test, lr_full_pred)}')
print(f'Linear_Regression_Reduced_Results: MAE={mae(y_test, lr_reduced_pred)}')
```

Как мы видим, МАЕ увеличилось, а значит на всех признаках лучше работает.

#### 19.1.7. Как изменился результат у бустинга

```
# Create the model with the same hyperparameters
model_reduced = GradientBoostingRegressor(
    max_depth=5,
    max_features=None,
    min_samples_leaf=6,
    min_samples_split=6,
```

```

n_estimators=800,
random_state=42
)

# Fit and test on the reduced set of features
model_reduced.fit(X_reduced, y)
model_reduced_pred = model_reduced.predict(X_test_reduced)

print(f'Gradient_Boosted_Reduced_Results : MAE={mae(y_test, model_reduced_pred)}')

```

Качество тоже ухудшилось, просто не так сильно.

## 19.2. Применяем LIME

Проанализируем модель на примере с очень плохим предсказанием (а потом с очень хорошим).

### 19.2.1. Найдём максимально верные и максимально неверные предсказания

```

# Find the residuals
residuals = abs(model_reduced_pred - y_test)

# Exact the worst and best prediction
wrong = X_test_reduced[np.argmax(residuals), :]
right = X_test_reduced[np.argmin(residuals), :]

print(np.argmax(residuals), np.argmin(residuals))

```

### 19.2.2. Используем LIME

```

# Create a lime explainer object
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_reduced,
    mode='regression',
    training_labels=y,
    feature_names=list(most_important_features)
)

```

### 19.2.3. Посмотрим на объяснение в том случае, когда модель предсказывает максимально плохо

```

# Display the predicted and true value for the wrong instance
print(f'Prediction:{model_reduced.predict(wrong.reshape(1,-1))}')
print(f'Actual_Value:{y_test[np.argmax(residuals)]}')

# Explanation for wrong prediction
wrong_exp = explainer.explain_instance(
    data_row=wrong,
    predict_fn=model_reduced.predict
)

# Plot the prediction explanation
wrong_exp.as_pyplot_figure()
plt.title('Explanation_of_Prediction', size=28)
plt.xlabel('Effect_on_Prediction', size=22)

```

### 19.2.4. Представление в ноутбуке

```

wrong_exp.show_in_notebook(show_predicted_value=False)
right_exp.show_in_notebook(show_predicted_value=False)

```

## 19.3. Используем SHAP

### 19.3.1. Объясним самое плохое предсказание с помощью SHAP

```
import shap
shap.initjs()

gb_explainer = shap.KernelExplainer(model_reduced.predict, X_test_reduced)[1000:1100]
gb_shap_values = gb_explainer.shap_values(X_test_reduced)[1000:1100]

shap.initjs()
shap.force_plot(
    gb_explainer.expected_value,
    gb_shap_values[23, :],
    X_test_reduced[23, :],
    feature_names=list(most_important_features)
)
```

### 19.3.2. Общие результаты

```
shap.summary_plot(gb_shap_values, X_test_reduced[1000:1100], plot_type="bar", features_nam
```

ДОПИСАТЬ ПОЗЖЕ

## 20. Лекция 11. Кластеризация и визуализация данных. Лекция ФЭН

### 20.1. Постановка задачи

Даны объекты  $x_1, \dots, x_l, x_i \in X$

Требуется выявить в данных  $K$  кластеров - таких областей, что объекты внутри одного кластера похожи друг на друга, а объекты из разных кластеров друг на друга не похожи.

Формализация задачи: необходимо построить алгоритм  $\alpha : X \rightarrow \{1, \dots, K\}$ , сопоставляющий каждому объекту  $x$  номер кластера.

### 20.2. Метрики качества

- **Внешние метрики** - используют информацию об истинных метках объектов
- **Внутренние метрики** - оценивают качество кластеризации, основываясь только на наборе данных.

#### 20.2.1. Внутрекластерное расстояние

Пусть  $c_k$  - центр  $k$ -го кластера

Внутри кластера все объекты максимально похожи, поэтому наша цель - минимизировать внутрекластерное расстояние:

$$\sum_{k=1}^K \sum_{i=1}^l [\alpha(x_i) = k] \rho(x_i, c_k) \rightarrow \min_{\alpha}$$

#### 20.2.2. Межкластерное расстояние

Объекты из разных кластеров должны быть как можно менее похожи друг на друга, поэтому мы максимизируем межкластерное расстояние:

$$\sum_{i,j=1}^l [\alpha(x_i) \neq \alpha(x_j)] \rho(x_i, x_j) \rightarrow \max_{\alpha}$$

#### 20.2.3. Индекс Данна

Это композиция внутрекластерного и межкластерного расстояния:

Хотим минимизировать внутрекластерное расстояние и одновременно максимизировать межкластерное расстояние

$$\frac{\min_{1 \leq k < k' \leq K} d(k, k')}{\max_{1 \leq k \leq K} d(k)} \rightarrow \max_{\alpha}$$

где:

- $d(k, k')$  - расстояние между кластерами  $k$  и  $k'$
- $d(k)$  - внутрекластерное расстояние для  $k$ -го кластера

### 20.3. Как считать расстояние?

Когда мы говорим про расстояние между точками, то не совсем понятно как его считать.

#### 20.3.1. Евклидово расстояние

Евклидово расстояние - расстояние между точками в общепринятом понимании, то есть геометрическое расстояние между двумя точками:

$$\rho(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

В общем случае для векторов длиной  $n$

$$\rho(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

### 20.3.2. Манхэттенское расстояние

Манхэттенское расстояние ещё называют расстоянием городских кварталов:

$$\rho(a, b) |x_1 - x_2| + |y_1 - y_2|$$

В общем случае для векторов длиной  $n$ :

$$\rho(a, b) = \sum_{i=1}^n |a_i - b_i|$$

## 20.4. Алгоритмы кластеризации

### 20.4.1. K-means

Дано: выборка  $x_1, \dots, x_l$

Параметр: число кластеров  $K$

Начало: случайно выбрать центры кластеров  $c_1, \dots, c_K$

#### 20.4.1.1. Алгоритм .

1. Каждый объект отнести к ближайшему центру кластера.
2. Пересчитать центры полученных кластеров по центру тяжести принадлежащих им точек.
3. Повторять шаги 1 и 2 до стабилизации кластеров.

Очень важно, что этот метод сходится: после какого-то шага точки не будут перекрашиваться.

#### 20.4.1.2. Идея метода. Идея метода - минимизация внутrikластерного расстояния

$$\sum_{k=1}^K \sum_{i=1}^l [\alpha(x_i) = k] \rho(x_i, c_k) \rightarrow \min_{\alpha}$$

и если мы возьмём  $\rho(a, b) = (a - b)^2$ , то

$$\sum_{k=1}^K \sum_{i=1}^l [\alpha(x_i) = k] (a - b)^2 \rightarrow \min_{\alpha}$$

Нужно также сказать, что  $K - means$  очень чувствителен к первоначальной инициализации точек. Чувствительность можно сравнить с изменением решающего дерева, если поменять выборку на которой мы его строили.

### 20.4.2. Графовые методы кластеризации

Выборка представляется в виде графа, где в вершинах стоят объекты, а на рёбрах - расстояние между ними.

Дальше удаляются все рёбра, которые больше какого-то веса  $R$ , и все попавшие в одну компоненту связности - причислены к одному кластеру.

### 20.4.3. Иерархическая кластеризация

Иерархия кластеров:

- на нижнем уровне -  $l$  кластеров, каждый из которых состоит из одного объекта
- на верхнем уровне - один большой кластер

#### 20.4.3.1. Алгоритм Ланса-Уильямса

1. Делаем каждый объект один кластером
2. На каждом следующем шаге объединяем два наиболее похожих кластера (по некоторой мере схожести  $d$ ) с предыдущего шага

#### 20.4.4. Density-Based Clustering

Сам метод называется *DBSCAN*.

Метод разделяет объекты на 3 типа:

- Границные объекты - объекты, которые попали на границы кластеров
- Основные объекты - объекты, которые попали внутрь кластеров
- Шумовые - в привычном понимании шум

Гипер-параметры метода:

- $\text{eps}$  - размер окрестности
- $\text{min\_samples}$  - минимальное число объектов в окрестности (включая сам объект), для определения основных точек

##### 20.4.4.1. Алгоритм DBSCAN

1. Выбрать точку без метки
2. Если в окрестности меньше, чем  $\text{min\_pts}$  точек, то пометить её как шумовую
3. Создать кластер, поместить в него текущую точку (если это не шумовая точка)
4. Для всех точек из окрестности  $S$ :
  - Если точка шумовая - отнести к данному кластеру, но не использовать для расширения
  - Если точка основная - отнести к данному кластеру, а её окрестность добавить к  $S$
5. Перейти к шагу 1

### 20.5. Метрики качества

*Disclaimer:* предполагается, что известны истинные метки объектов.

Мера зависит не от самих значений меток, а от разбиения выборки на кластеры.

Пусть:

- $a$  - число пар объектов с одинаковыми метками и находящихся в одном кластере
- $b$  - число пар объектов с различными метками и находящихся в разных кластерах
- $N$  - число объектов в выборке

Тогда

#### 20.5.1. Rand Index (RI)

$$RI = \frac{a + b}{C_N^2} = \frac{2(a + b)}{N(N - 1)}$$

Где  $RI$  - доля объектов, для которых исходное и полученное разбиения согласованы. Выражает похожесть двух различных разбиений выборки.

#### 20.5.2. Adjusted Rand Index (ARI)

$RI$  нормируется так, чтобы величина всегда принимала значения из отрезка  $[-1; 1]$  независимо от числа объектов  $N$  и числа кластеров, получается  $ARI$ :

$$ARI = \frac{RI - \mathbb{E}[RI]}{\max(RI) - \mathbb{E}[RI]}$$

- $ARI > 0$  - разбиения похожи ( $ARI = 1$  - совпадают)
- $ARI \approx 0$  - случайные разбиения
- $ARI < 0$  - непохожие разбиения

### 20.5.3. Mutual Information (AMI)

Метрика похожа на ARI.

Индекс MI - это взаимная информация для двух разбиений выборки на кластеры:

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P_{UV}(i, j) \frac{\log(P_{UV}(i, j))}{P_U(i) \cdot P_V(j)}$$

где

- $P_{UV}(i, j)$  - вероятность, что объект принадлежит кластеру  $U_i \subset U$  и кластеру  $V_j \subset V$
- $P_U(i)$  - вероятность, что объект принадлежит кластеру  $U_i \subset U$
- $P_V(j)$  - вероятность, что объект принадлежит кластеру  $V_j \subset V$

AMI в отличии от MI уже относительная метрика - лежит только в  $[0; 1]$  - чем ближе к 1, тем более похожи разбиения.

### 20.5.4. Гомогенность, Полнота, V-мера

Пусть  $H$  - энтропия:  $H = -\sum_{i=1}^{|U|} P(i) \log(P(i))$ . Тогда

$$h = 1 - \frac{H(C|K)}{H(C)}, c = 1 - \frac{H(K|C)}{H(K)}$$

где

- $K$  - результат кластеризации
  - $C$  - истинное разбиение выборки на классы
- при этом
- $h$ (гомогенность) измеряет, насколько каждый кластер состоит из объектов одного класса
  - $c$ (полнота) измеряет, насколько объекты одного класса относятся к одному кластеру

Гомогенность и полнота принимают значения из отрезка  $[0; 1]$ . Большие значения соответствуют более точной кластеризации.

**Эти метрики не нормализованы (как ARI и AMI), то есть они зависят от числа кластеров.**

- При большем количестве кластеров и малом числе объектов лучше использовать ARI и AMI
- При более 1000 объектов и числе кластеров меньше 10 проблема не так сильно выражена, поэтому её можно игнорировать.

### 20.5.5. Силуэт (Silhouette)

Не требует знания истинных меток.

Пусть  $a$  - среднее расстояние от объекта до всех объектов из того же кластера,  $b$  - среднее расстояние от объекта до объектов из ближайшего (не содержащего объект) кластера. Тогда силуэт данного объекта:

$$s = \frac{b - a}{\max(a, b)}$$

Силуэт выборки ( $S$ ) - средняя величина силуэта по объектам.

Силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров.

При этом  $S \in [-1; 1]$ :

- $S \approx -1$  - плохие (разнозненные) кластеризации
- $S \approx 0$  - кластеры накладываются друг на друга
- $S \approx 1$  - чётко выраженные кластеры

С помощью силуэта можно выбирать число кластеров  $k$  (если оно заранее неизвестно) - выбирается  $k$ , для которого метрика максимальна.

Силуэт зависит от формы кластеров и достигает больших значений на более выпуклых кластерах.

## 20.6. Визуализация

Задача визуализации состоит в отображении объектов в 2x или 3x мерное пространство с сохранением отношений между ними.

### 20.6.1. MultiDimensional Scaling (MDS)

Идея метода - минимизация квадратов отклонений между исходными и новыми попарными расстояниями:

$$\sum_{i \neq j}^l (\rho(x_i, x_j) - \rho(z_i, z_j))^2 \rightarrow \min_{z_1, \dots, z_l}$$

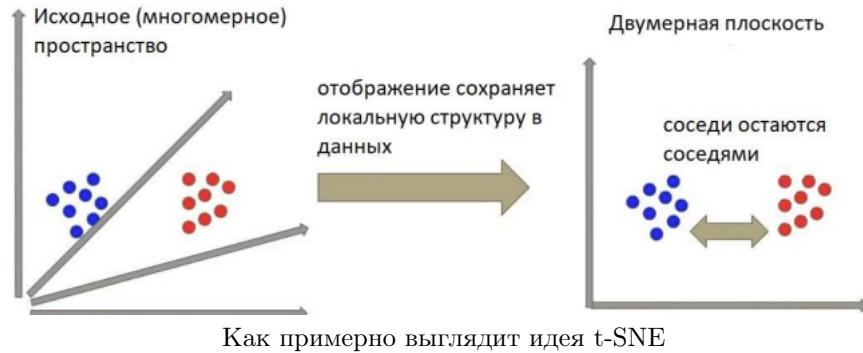
где  $z_i$  и  $z_j$  - это проекции на  $\mathbb{R}^2$  или  $\mathbb{R}^3$ .

В теории хорошо, на практике плохо.

### 20.6.2. t-SNE(T-distributed stochastic neighbour embedding)

Идея в том, что нам важно не сохранение расстояний между объектами, а сохранение пропорций:

$$\rho(x_1, x_2) = \alpha \rho(x_1, x_3) \Rightarrow \rho(z_1, z_2) = \alpha \rho(z_1, z_3)$$



Как выглядит близость объектов в исходном пространстве?

$$p(i|j) = \frac{\exp\left(\frac{-||x_i - x_j||^2}{2\sigma_j^2}\right)}{\sum_{k \neq j} \exp\left(\frac{-||x_k - x_j||^2}{2\sigma_j^2}\right)}$$

Объекты из окрестности  $x_j$  приближаются нормальным распределением

Чем кучнее объекты из этой окрестности, тем меньше берётся значение  $\sigma_j^2$

#### 20.6.2.1. Обучение t-SNE Для построения проекций $z_i$ объектов $x_i$ будем минимизировать расстояние между исходным и полученным распределениями (минимизируем дивергенцию Кульбака-Лейблера)

$$KL(p||q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \rightarrow \min_{z_1, \dots, z_l}$$

## 21. Кластеризация. Обучение без учителя. Конспект Соколова

До сих пор мы изучали методы обучения с учителем - то есть методы, которые восстанавливают зависимость по объектам с известными ответами. Если задать семейство моделей и функционал ошибки, то обучение сводится к выбору лучшей модели с точки зрения этого функционала.

Также существует большой класс обучения без учителя (*unsupervised learning*), в которых отсутствует целевая переменная, и требуется восстановить некую скрытую структуру в данных. Примером может служить визуализация - задача изображения многомерной выборки на двухмерной плоскости. Чтобы визуализация была осмысленной, при таком отображении нужно сохранить основные закономерности данных. Формализовать требование "сохранить основные закономерности" тяжело, и поэтому строго оценить качество решения данной задачи не представляется возможным.

Мы рассмотрим несколько типов задач обучения без учителя, обсудим методы их решения и подходы к измерению качества.

### 21.1. Кластеризация

Пусть дана выборка объектов  $X = (x_i)_{i=1}^l, x_i \in \mathbb{X}$ . В задаче кластеризации требуется выявить в данных  $K$  кластеров - таких областей, что объекты внутри одного кластера похожи друг на друга, а объекты из разных кластеров друг на друга не похожи. Более формально, требуется построить алгоритм  $\alpha : \mathbb{X} \rightarrow \{1, \dots, K\}$ , определяющий для каждого объекта номер его кластера; число кластеров  $K$  может либо быть известно, либо являться параметром.

Кластеризовать можно многое что: новости по сюжетам, пиксели на изображении по принадлежности к объекту, музыку по жанрам, сообщения на форуме по темам, клиентов по типу поведения.

В нашей постановке задачи много неточностей - в частности, мы не указали, как измеряется сходство объектов. Как и раньше, начнём обсуждение задачи с метрик качества.

#### 21.1.1. Метрики качества кластеризации

Существует два подхода к измерению качества кластеризации, внутренний и внешний; Внутренний основан на некоторых свойствах выборки и кластеров, а внешний использует дополнительные данные - например, информацию об истинных кластерах.

Приведём несколько примеров внутренних метрик качества. Будем считать, что каждый кластер характеризуется своим центром  $c_k$ .

1. Внутрикластерное расстояние:

$$\sum_{k=1}^K \sum_{i=1}^l [\alpha(x_i) = k] \rho(x_i, c_k)$$

где  $\rho(x, z)$  - некоторая функция расстояния. Данный функционал требуется минимизировать, поскольку в идеале все объекты кластера должны быть одинаковыми.

2. Межкластерное расстояние:  $\sum_{i,j=1}^l [\alpha(x_i) \neq \alpha(x_j)] \rho(x_i, x_j)$

Данный функционал нужно максимизировать, поскольку объекты из разных кластеров должны быть как можно менее похожими друг на друга.

3. Индекс Данна (Dunn Index):

$$\frac{\min_{1 \leq k < k' \leq K} d(k, k')}{\max_{1 \leq k \leq K} d(k)}$$

где  $d(k, k')$  - расстояние между кластерами  $k$  и  $k'$  (например, евклидово расстояние между их центрами), а  $d(k)$  - внутрикластерное расстояние для  $k$ -го кластера (например, сумма расстояний от всех объектов этого кластера до его центра). Данный индекс необходимо максимизировать (мы хотим увеличить минимальное расстояние между кластерами, а также уменьшить внутреннее расстояние кластеров).

Внешние метрики возможно использовать, если известно истинное распределение объектов по кластерам. В этом случае задачу кластеризации можно рассматривать как задачу многоклассовой классификации, и использовать любую метрику оттуда -  $F$ -меру с микроД или макроусреднением.

### 21.1.2. K-Means

Одним из наиболее популярных методов кластеризации является  $K - Means$ , который оптимизирует внутрикластерное расстояние (21.1), в котором используется квадрат евклидовой метрики.

Заметим, что в данном функционале имеется две степени свободы: центры кластеров  $c_k$  и распределение объектов по кластерам  $\alpha(x_i)$ . Выберем для этих величин произвольные начальные приближения, а затем будем оптимизировать их по очереди:

1. Зафиксируем центры кластеров. В этом случае внутрикластерное расстояние будет минимальным, если каждый объект будет относиться к тому кластеру, чей центр является ближайшим:  $\alpha(x_i) = \arg \min_{1 \leq k \leq K} \rho(x_i, c_k)$

2. Зафиксируем распределение объектов по кластерам. В этом случае внутрикластерное расстояние с квадратом евклидовой метрики можно проанализировать по центрам кластеров и вывести

$$\text{аналитические формулы для них: } c_k = \frac{\sum_{i=1}^l [\alpha(x_i)=k] x_i}{\sum_{i=1}^l [\alpha(x_i)=k]}$$

Повторяя эти шаги до сходимости, мы получим некоторое распределение объектов по кластерам. Новый объект относится к тому кластеру, чей центр является ближайшим.

Результат работы метода  $K - Means$  существенно зависит от начального приближения. Существует большое количество подходов к инициализации; одни из наиболее успешных считается  $k - means + +$ .

### 21.1.3. Графовые методы

Графовые методы кластеризации - это простые методы, которые основаны на построении графа близости. Его вершинами являются объекты, а выбор рёбер зависит от конкретного алгоритма. Например, рёбра могут быть проведены между объектами, расстояния между которыми меньше определённого порога. Кластерами же объявляются группы объектов, попадающих в одну компоненту связности.

Такие подходы очень простые, но при грамотном выборе функции расстояния (скажем, обученной под конкретную задачу) могут показывать очень хорошие результаты.

### 21.1.4. Иерархическая кластеризация

Описанные выше методы кластеризации находят "плоскую" структуру кластеров. В некоторых задачах возникает потребность в построении иерархии кластеров, в которой верхним уровнем является один большой кластер, а нижним -  $l$  кластеров, каждый из которых состоит из одного объекта. Например, при кластеризации новостей можно рассчитывать, что чем ниже мы спускаемся по иерархии, тем более тонкие различия между сюжетами будут выделяться.

Одним из подходов является восходящая кластеризация. Она начинается с нижнего уровня, на котором все объекты принадлежат к отдельным кластерам:  $C^l = \{\{x_1\}, \dots, \{x_l\}\}$ . Каждый следующий уровень  $C^j$  получается путём объединения двух наиболее похожих кластеров с предыдущего уровня  $C^{j+1} = \{X_1, \dots, X_{j+1}\}$ . Схожесть кластеров определяется с помощью некоторой функции  $d(X_m, X_n)$  - например, это может быть расстояние между центрами кластеров.

## 21.2. Визуализация

Как уже упоминалось выше, задача визуализации состоит в отображении объектов в двух или трёхмерное пространство с сохранением отношений между ними. Под сохранением отношений обычно понимают близость попарных расстояний в исходном и новом пространствах.

Так, в методе многомерного шкалирования (multidimensional scaling, MDS) минимизируются квадраты отклонений между исходными и новыми попарными расстояниями:

$$\sum_{i \neq j} (\rho(x_i, x_j) - \rho(z_i, z_j))^2 \rightarrow \min_{z_1, \dots, z_l}$$

где

- $x_i \in \mathbb{R}^D$  - исходные объекты
- $z_i \in \mathbb{R}^d$ , где  $2 \leq d \leq 3$  - их низкоразмерные проекции.

Обратим внимание на две особенности данного подхода:

- Исходные объекты не обязаны принадлежать евклидову пространству - достаточно лишь уметь вычислять расстояния между ними. Благодаря этому можно визуализировать даже сложные объекты вроде строк.
- Проекции объектов ищутся непосредственно, без какой-либо параметрической зависимости между ними и исходными представлениями объектов. Из-за этого затруднительно добавить к визуализации новые данные. Впрочем, это и не нужно - мы ведь хотим просто нарисовать объекты и посмотреть на них. Если же требуется отображать и новые, тестовые данные, то следует пользоваться методами понижения размерности, которые преобразуют объекты с помощью некоторой модели.

Одним из наиболее популярных методов визуализации на сегодняшний день является *t-distributed stochastic neighbor embedding* (t-SNE), который исправляет несколько ключевых проблем многомерного шкалирования.

Для начала заметим, что нам не так важно точное сохранение расстояний после проецирования - достаточно лишь сохранять пропорции. Например, если  $\rho(x_1, x_2) = \alpha \rho(x_1, x_3)$ , то в новом пространстве достаточно выполнения такого же равенства, чтобы соотношения между этими тремя объектами были сохранены:  $\rho(z_1, z_2) = \alpha \rho(z_1, z_3)$ . Будем использовать нормальную плотность для измерения сходства объектов в исходном пространстве:

$$\rho(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Отнормируем эти близости так, чтобы получить вектор распределений расстояний от объекта  $x_j$  до всех остальных объектов:

$$p(i|j) = \frac{\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma_j^2}\right)}{\sum_{k \neq j} \exp\left(\frac{-\|x_k - x_j\|^2}{2\sigma_j^2}\right)}$$

Данные величины не являются симметричными, что может добавить нам дополнительных сложностей при дальнейшей работе. Симметризуем их:

$$p_{ij} = \frac{p(i|j) + p(j|i)}{2l}$$

Благодаря данному способу симметризации невозможна ситуация, в которой для некоторого отдалённого объекта  $x_i$  все близости  $p_{ij}$  будут близки к нулю - можно показать, что всегда выполнено  $\sum_j p_{ij} > \frac{1}{2l}$

Перейдём теперь к измерению сходства в новом низкоразмерном пространстве. Известно, что в пространствах высокой размерности можно разместить объекты так, что их попарные расстояния буду близки - а вот сохранить это свойство в низкоразмерном пространстве вряд ли возможно. Поэтому будем измерять сходства между объектами с помощью распределения Коши, которое имеет тяжёлые хвосты и не так сильно штрафует за увеличение расстояний между объектами:

$$q_{ij} = \frac{(1 + \|z_i - z_j\|^2)^{-1}}{\sum_{k \neq m} (1 + \|z_k - z_m\|^2)^{-1}}$$

Теперь мы умеем измерять расстояния между объектами как в исходном, так и в новом пространствах, и осталось лишь задать функционал ошибки проектирования. Будем измерять ошибку с помощью дивергенции Кульбака-Лейблера, которая часто используется для измерения расстояний между распределениями:

$$KL(p||q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \rightarrow \min_{z_1, \dots, z_l}$$

Решать данную задачу оптимизации можно с помощью стохастического градиентного спуска.

### 21.3. Обучение представлений

Ещё одной разновидностью задач обучения без учителя является обучение представлений (representation learning), которое состоит в построении некоторых числовых представлений исходных объектов с сохранением свойств этих объектов.

Мы уже сталкивались с этой областью - выходы одного из последних слоёв свёрточной сети являются представлениями изображения, и их можно использовать как признаки при решении той или иной задачи. В данном разделе мы разберём *word2vec*, способ обучения представлений для слов.

В лингвистике существует дистрибутивная гипотеза, согласно которой слова, встречающиеся в похожих контекстах, имеют похожие смыслы. Будем строить представления для слов, опираясь на эту гипотезу: чем в более похожих контекстах встречаются два слова, тем ближе должны быть соответствующие им векторы.

Итак, мы хотим для каждого слова  $w$  из словаря  $W$  найти вектор  $\vec{w} \in \mathbb{R}^d$ . Пусть дан некоторый текст  $x = (w_1, \dots, w_n)$ . Контекстом слова  $w_j$  будем называть слова, находящиеся от него на расстоянии не более  $K$  - то есть слова  $w_{j-K}, \dots, w_{j-1}, w_{j+1}, \dots, w_{j+K}$ . Определим через векторы слов вероятность встретить слово  $w_i$  в контексте слова  $w_j$ :

$$p(w_i|w_j) = \frac{\exp(\langle \vec{w}_i, \vec{w}_j \rangle)}{\sum_{w \in W} \exp(\langle \vec{w}, \vec{w}_j \rangle)}$$

Тогда для выборки текстов  $X = \{x_1, \dots, x_l\}$ , где текст  $x_i$  имеет длину  $n_i$ , можно определить правдоподобие и максимизировать его:

$$\sum_{i=1}^l \sum_{j=1}^{n_i} \sum_{k=-K, k \neq 0}^K \log p(\vec{w}_{j+k} | \vec{w}_j) \rightarrow \max_{\{\vec{w}\}_{w \in W}}$$

Данный функционал можно оптимизировать стохастическим градиентным спуском. В результате обучения мы получим представления для слов, которые, как показывает практика, будут обладать многими интересными свойствами - и, в том числе, близкие по смыслу слова будут иметь близкие векторы.

## 22. Перевод статьи DBSCAN Clustering — Explained

Статья - <https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556>

Кластеризация - это способ сгруппировать набор точек данных таким образом, чтобы похожие точки данных группировались вместе. Поэтому алгоритмы кластеризации ищут сходства или различия между точками данных. Кластеризация - это неконтролируемый метод обучения, поэтому метки, связанные с точками данных, отсутствуют. Алгоритм пытается найти закономерности в данных.

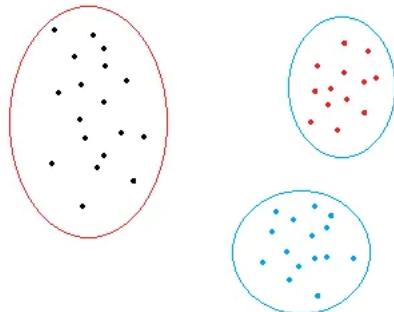
Существуют различные подходы и алгоритмы для выполнения задач кластеризации, которые можно разделить на три вида:

- Кластеризация на основе разделения данных: например  $k - Means$  или  $k - Median$
- Иерархическая кластеризация: например агломеративная или разделяющая кластеризация
- Кластеризация на основе плотности: например DBSCAN

### 22.1. Кластеризация на основе плотности

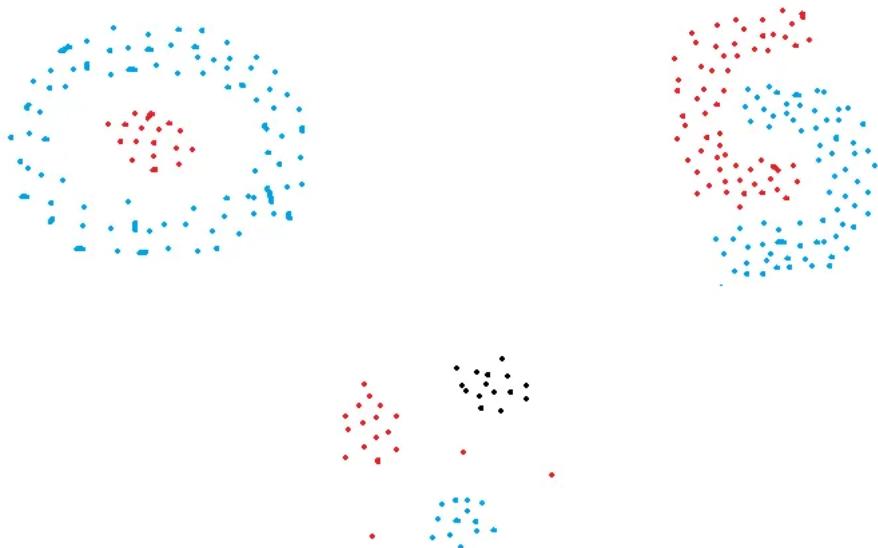
Методы кластеризации на основе разделения данных и иерархические очень эффективны для кластеров нормальной формы. Однако, когда дело доходит до кластеров произвольной формы или обнаружения выбросов - методы, основанные на плотности, более эффективны.

Например, набор данных на рисунке ниже можно легко разделить на три кластера, используя алгоритм  $k - Means$



Кластеризация  $k - Means$

Рассмотрим следующие изображения:



Точки, которые обозначают объекты, на этих рисунках сгруппированы в произвольные формы или включают выбросы. Алгоритмы кластеризации на основе плотности очень эффективны при поиске областей с высокой плотностью и выбросами. Очень важно обнаруживать выбросы для какой-либо задачи, например, обнаружения аномалий.

## 22.2. Алгоритм DBSCAN

DBSCAN (density-based) способен находить кластеры произвольной формы и кластеры с шумом.

Основная идея DBSCAN заключается в том, что точка принадлежит кластеру, если она близка ко многим точкам из этого кластера.

У DBSCAN есть два гиперпараметра:

- $\text{eps}$  - расстояние, определяющее окрестности. Две точки считаются соседями, если расстояние между ними меньше или равно  $\text{eps}$ .
- $\text{minPts}$ : минимальное количество точек данных для определения кластера.

На основе этих двух гиперпараметров точки классифицируются на:

- Основные - которые лежат внутри кластера, в области радиусом  $\text{eps}$  вокруг неё по крайней мере  $\text{minPts}$  точек (включая саму точку).
- Пограничные - точки, достижимые из основных, но не имеющие  $\text{minPts}$  соседей.
- Выбросы - не являющиеся ни основными, ни пограничными точки.

Сама работа алгоритма:

1. Определяются  $\text{minPts}$  и  $\text{eps}$ .
2. Выбирается случайным образом начальная точка.
  - (a) Если в окрестности есть  $\text{minPts}$  точек, то точка помечается как основная и начинается формирование кластера. Все соседние точки добавляются в тот же кластер, что и начальная. Если эти точки тоже являются основными - произвести действия рекурсивно.
  - (b) Если в окрестности нет  $\text{minPts}$  точек, то точка помечается шумовой.
3. Пока остались непомеченные точки - повторять шаг 2.

Применяя эти шаги, алгоритм DBSCAN способен находить области с высокой плотностью и отделять их от областей с низкой плотностью.

Кластер включает в себя основные точки, которые являются соседними (т. Е. Достижимыми Друг от друга), и все пограничные точки этих основных точек. Необходимым условием для формирования кластера является наличие хотя бы одной базовой точки. Хотя это очень маловероятно, у нас может быть кластер только с одной основной точкой и ее пограничными точками.

## 22.3. Scikit-learn реализация

### 22.3.1. Импортование библиотек

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

### 22.3.2. Генерация датасета

```
#Determine centroids
centers = [[0.5, 2], [-1, -1], [1.5, -1]]
#Create dataset
X, y = make_blobs(n_samples=400, centers=centers,
    cluster_std=0.5, random_state=0)
#Normalize the values
X = StandardScaler().fit_transform(X)
```

### 22.3.3. Визуализация полученного датасета

```
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1], c=y, cmap='Paired')
```

#### **22.3.4. Использование DBSCAN**

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.4, min_samples=20)
db.fit(X)
```

#### **22.3.5. Визуализация результатов работы DBSCAN**

```
y_pred = db.fit_predict(X)
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1], c=y_pred, cmap='Paired')
plt.title("Clusters_determined_by_DBSCAN")
```

### **22.4. Плюсы и минусы DBSCAN**

Плюсы:

- Не требует предварительного указания количества кластеров.
- Хорошо работает с кластерами произвольной формы.
- DBSCAN устойчив к выбросам и способен обнаруживать выбросы.

Минусы:

- В некоторых случаях определение подходящего расстояния соседства (eps) непросто и требует знаний предметной области.
- Если кластеры сильно отличаются по плотности внутри кластера, DBSCAN не очень подходит для определения кластеров. Характеристики кластеров определяются комбинацией параметров eps-minPts. Поскольку мы переходим к алгоритму в одной комбинации eps-minPts, он не может хорошо общаться на кластеры с сильно отличающейся плотностью.

## **23. HDBSCAN - перевод статьи**

**СДЕЛАТЬ ПОЗЖЕ**

## 24. Лекция 11. Кластеризация. Наша лекция

### 24.1. Практика

#### 24.1.1. Генерация датасета

```
from sklearn.cluster import KMeans
import numpy as np
from matplotlib import pylab as plt
%pylab inline

X = np.zeros((150, 2))

np.random.seed(seed=42)
X[:50, 0] = np.random.normal(loc=0.0, scale=0.3, size=50)
X[:50, 1] = np.random.normal(loc=0.0, scale=0.3, size=50)

X[50:100, 0] = np.random.normal(loc=2.0, scale=0.5, size=50)
X[50:100, 1] = np.random.normal(loc=-1.0, scale=0.2, size=50)

X[100:150, 0] = np.random.normal(loc=-1.0, scale=0.2, size=50)
X[100:150, 1] = np.random.normal(loc=2.0, scale=0.5, size=50)

plt.figure(figsize=(12,8))
plt.scatter(X[:,0], X[:,1], s=50, cmap='viridis')
plt.xlabel('x')
plt.ylabel('y')
```

#### 24.1.2. Применение K-Means

```
kmeans = KMeans(n_clusters=3, random_state=1)
kmeans.fit(X)
print(kmeans.labels_)
```

#### 24.1.3. Визуализация предсказания K-Means

```
plg.figure(figsize=(12,8))
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, s=50, cmap='viridis')
plt.xlabel('x')
plt.ylabel('y')
```

#### 24.1.4. Применение иерархической кластеризации

```
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist

distance_mat = pdist(X)

Z = hierarchy.linkage(distance_mat, 'single')
plt.figure(figsize=(12,8))
dn = hierarchy.dendrogram(Z, color_threshold=0.5)
```

#### 24.1.5. Что будет если подобрать неверное количество кластеров?

**24.1.5.1. K-Means** Для  $K - Means$  трудности не предоставляет - сколько нужно найти кластеров, столько и найдёт:

```

plt.figure(figsize=(15,8))
for n_c in range(2, 8):
    kmeans = KMeans(n_clusters=n_c)
    kmeans = kmeans.fit(X)
    clusters = kmeans.predict(X)
    plt.subplot(2, 3, n_c-1)
    plt.scatter(X[:, 0], X[:, 1], c=clusters)
    plt.title(f'n_clusters={n_c}')
    print('n=', n_c, 'score:', silhouette_score(X, clusters))

plt.show()

```

А что делать, если мы не знаем сколько кластеров у нас есть?

- Можно использовать другие алгоритмы кластеризации
- Можно использовать другие метрики качества кластеризации

И тут нужно понимать, что внешние метрики у нас нужны для сравнения двух кластеризаций, а внутренние для оценки качества кластеризации:

- Насколько кластеры удалены друг от друга
- Насколько кластеры хорошей формы

Как раз для оценки кластеризации и есть silhouette\_score. Но нужно сказать, что для этой метрики все те кластеры, что не является шаром или кругом - плохие кластеры и нужно учитывать это.

#### 24.1.5.2. DBSCAN

Посмотри на результаты кластеризации при разном выборе параметров *eps* и *min\_samples*

```

from sklearn.cluster import DBSCAN

plt.figure(figsize=(15,23))
i = 1
for samples in [2, 4, 8]:
    for e in [0.1, 0.2, 0.5, 1, 2]:
        dbscan = DBSCAN(eps=e, min_samples=samples)
        clusters = dbscan.fit_predict(X)
        plt.subplot(6, 3, i)
        plt.scatter(X[:,0], X[:,1], c=clusters)
        plt.title(f'eps={e}, n={samples}')
        try:
            print(f'eps={e}, n={samples}, score:{silhouette_score(X, clusters)}')
        except:
            print(f'eps={e}, n={samples}, score:-1')
        i += 1
    i += 1

plt.show()

```

Пример визуализации работы DBSCAN - <https://www.naftaliharris.com/blog/visualizing-dbscan-clustering>

## 24.2. Пример: кластеризация игроков NBA

### 24.2.1. Получаем данные

```

import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

nba = pd.read_csv('nba_2019.csv')
nba.head()
nba.columns

```

#### 24.2.2. Применим K-Means с 5-ю кластерами только к числовым столбцам

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

numeric_cols = nba._get_numeric_data().dropna(axis=1)

kmeans = KMeans(n_clusters=5, random_state=3)
kmeans.fit(numeric_cols)
```

#### 24.2.3. Визуализируем данные с помощью метода главных компонент (PCA)

```
pca = PCA(n_components=2)
res = pca.fit_transform(numeric_cols)

plt.figure(figsize=(12,8))
plt.scatter(res[:,0], res[:,1], c=kmeans.labels_, s=50, cmap='viridis')
plt.xlabel('x')
plt.ylabel('y')
```

#### 24.2.4. Посмотрим, какое смысловое значение несут кластеры

1. Визуализируем точки в осях `nba['pts']` (total points) и `nba['ast']` (total assistances) и раскрасим их в цвета кластеров.
2. Визуализируем точки в осях `nba['age']` (age) и `nba['mp']` (minutes played) и раскрасим их в цвета кластеров.

```
plt.figure(figsize=(12, 8))
plt.scatter(nba['PTS'], nba['AST'], c=kmeans.labels_, s=50, cmap='viridis')
plt.xlabel('points')
plt.ylabel('assistances')

plt.figure(figsize=(12, 8))
plt.scatter(nba['Age'], nba['MP'], c=kmeans.labels_, s=50, cmap='viridis')
plt.xlabel('age')
plt.ylabel('minutes_played')
```

### 24.3. Сжатие изображений с помощью K-Means

#### 24.3.1. Возьмём изображение

```
import matplotlib.image as mpimg
img = mpimg.imread('duck.jpg')[..., 1]
plt.figure(figsize=(15,9))
plt.axis('off')
plt.imshow(img, cmap='gray')
```

#### 24.3.2. Сжимаем изображение

Если у нас уменьшить кол-во возможных цветов у пикселей, то понадобится меньше памяти, чтобы закодировать один пиксель, а следовательно и всё изображение:

```
from sklearn.cluster import MiniBatchKMeans
from scipy.stats import randint

X = img.reshape((-1, 1)) # img to vec
k_means = MiniBatchKMeans(n_clusters=5)
k_means.fit(X)
values = k_means.cluster_centers_ # mean color
```

```

labels = k_means.labels_

img_compressed = values[labels].reshape(img.shape) # vec to img

plt.figure(figsize=(15,9))
plt.axis('off')
plt.imshow(img_compressed, cmap='gray')

```

Даже с 5-ю цветами картинка хорошо видна

## 24.4. Нахождение тем в текстах

### 24.4.1. Загрузка данных

Применим *KMeans* для кластеризации текстов из 4 новостных категорий.

```

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer
from sklearn.preprocessing import Normalizer
from sklearn import metrics
from time import time

categories = [
    'talk.politics.misc',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space'
]
print('Loading 20 newsgroups dataset for categories :')
print(categories)

dataset = fetch_20newsgroups(subset='all', categories=categories,
                             shuffle=True, random_state=42)
print(f'{len(dataset.data)} documents')
print(f'{len(dataset.target_names)} categories')

labels = dataset.target
true_k = np.unique(labels).shape[0]

```

### 24.4.2. TF-IDF

```

print('Extracting features from the training dataset using a sparse vectorizer')
vectorizer = TfidfVectorizer(max_df=0.5, max_features=1000,
                            min_df=2, stop_words='english')
X = vectorizer.fit_transform(dataset.data)
print(f'n_samples:{X.shape[0]}, n_features:{X.shape[1]}' )

```

### 24.4.3. Применим K-Means к получившимся векторам и выведем метрики качества кластеризации

```

km = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)

print(f'Clustering sparse data with {str(km)}')

t0 = time()
km.fit(X)

print(f'Homogeneity:{metrics.homogeneity_score(labels, km.labels_)}')
print(f'Completeness:{metrics.completeness_score(labels, km.labels_)}')
print(f'V-measure:{metrics.v_measure_score(labels, km.labels_)}')
print(f'Adjusted_Rand_Index:{metrics.adjusted_rand_score(labels, km.labels_)}')

```

```

print(f'Adjusted_mutual_info_score:{metrics.adjusted_mutual_info_score(labels, km.labels_)}
print(f'Silhouette_Coefficient:{metrics.silhouette_score(labels, km.labels_, sample_size=order_centroids = km.cluster_centers_.argsort()[:, :-1]

```

#### **24.4.4. Выведем слова, соответствующие самым весомым компонентам центров кластеров**

```

terms = vectorizer.get_feature_names()
for i in range(true_k):
    print(f'Cluster {i+1}:', end=' ')
    for ind in order_centroids[i, :10]:
        print(f'{terms[ind]}', end=' ')
    print()

```

### **24.5. Кластеризация рукописных цифр**

#### **24.5.1. Загрузка данных**

```

from sklearn.datasets import load_digits

digits = load_digits()

X, y = digits.data, digits.target()
Im = digits.images

```

Выведем на экран первые 20 цифр:

```

for i in range(20):
    plt.figure(figsize=(2, 2))
    plt.imshow(Im[i], cmap='gray')
    plt.show()

```

#### **24.5.2. Обучим K-Means с 10-ю кластерами**

```

km = KMeans(n_clusters=10, init='k-means++', max_iter=100, n_init=1)
km.fit(X)

```

#### **24.5.3. Посмотрим на центры кластеров**

```

_, axes = plt.subplots(2, 5)
for ax, center in zip(axes.ravel(), km.cluster_centers_):
    ax.matshow(center.reshape(8, 8), cmap=plt.cm.gray)
    ax.set_xticks(())
    ax.set_yticks(())

```

#### **24.5.4. Визуализация с помощью PCA**

```

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

print(f'Projecting {X.shape[1]}-dimensional data to 2D')

plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=km.labels_)
plt.show()

```

## 24.6. HDBSCAN

### 24.6.1. Подключение библиотек

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets as data
%matplotlib inline
sns.set_context('poster')
sns.set_style('white')
sns.set_color_codes()
plot_kwds = {'alpha': 0.5, 's': 80, 'linewidths': 0}
```

### 24.6.2. Генерация данных

```
plt.figure(figsize=(12,8))

moons, _ = data.make_moons(n_samples=50, noise=0.05)
blobs, _ = data.make_blobs(n_samples=50, centers=[(-0.75, 2.25), (1.0, 2.0)], cluster_std=0.5)
test_data = np.vstack([moons, blobs])
plt.scatter(test_data.T[0], test_data.T[1], color='b', **plot_kwds)
```

### 24.6.3. Кластеризуем с помощью hdbscan

```
clusterer = hdbscan.HDBSCAN(min_cluster_size=5, gen_min_span_tree=True)
clusterer.fit(test_data)
```

### 24.6.4. Алгоритм вкратце

1. Апроксимируем распределение данных с помощью расстояния до  $k$ -го соседа
2. Строим минимальное остовное дерево
3. Строим иерархию кластеров
4. Схлопываем кластеры по параметру  $\min_{cluster\_size}$
5. Извлекаем итоговые кластеры

### 24.6.5. Визуализация мин остова для наших данных

```
plt.figure(figsize=(12,8))

clusterer.minimim_spanning_tree.plot(
    edge_cmap='viridis',
    edge_alpha=0.6,
    node_size=80,
    edge_linewidth=2
)
```

### 24.6.6. Визуализация дендограммы для наших данных

```
plt.figure(figsize=(12,8))

clusterer.single_linkage_tree.plot(cmap='viridis', colorbar=True)
```

#### 24.6.7. Схлопываем кластеры

```
plt.figure(figsize=(12,8))

clusterer.condensed_tree.plot()

Можно добавить параметр select_clusters = True и алгоритм за нас обведёт нужные кластеры:

plt.figure(figsize=(12,8))

clusterer.condensed_tree.plot(select_clusters=True, selection_palette=sns.color_palette())
```

#### 24.6.8. Найдём итоговые кластеры на наших данных

```
plt.figure(figsize=(12,8))

palette = sns.color_palette()
clusters_colors = [
    sns.desaturate(palette[col], sat)
    if col >= 0 else (0.5, 0.5, 0.5) for col, sat in
    zip(clusterer.labels_, clusterer.probabilities_)

plt.scatter(test_data.T[0], test_data.T[1], c=clusters_colors, **plot_kwds)
```

#### 24.6.9. Сравнение с DBSCAN

```
test_data = np.load('clusterable_data.npy')
plt.figure(figsize(12, 8))
plt.scatter(test_data.T[0], test_data.T[1], color='b', **plot_kwds)

from sklearn.cluster import DBSCAN

db_clusterer = DBSCAN(eps=0.025)
db_clusterer.fit(test_data)

plt.figure(figsize=(12,8))

palette = sns.color_palette()
clusters_colors = [
    sns.desaturate(palette[col], 0.5)
    if col >= 0 else (0.5, 0.5, 0.5) for col in
    db_clusterer.labels_]

plt.scatter(test_data.T[0], test_data.T[1], c=clusters_colors, **plot_kwds)
```

## 25. Кластеризация. Продолжение. Конспект Соколова

Вернёмся к одной из задач обучения без учителя - кластеризации. Пусть у нас есть выборка  $X = \{x_i\}_{i=1}^l$ ,  $x_i \in \mathbb{X}$ , и мы хотим построить алгоритм  $\alpha : \mathbb{X} \rightarrow \{1, \dots, K\}$ , который ставил бы в соответствие объекту номер кластера. Ранее предлагалось использовать номера кластеров как новый признак для обучения с учителем. Сегодня мы рассмотрим модель, которая будет генерировать для нас сколько угодно таких кластерных признаков.

### 25.1. Графовые методы

#### 25.1.1. От выборки к графу

В курсе МО-1 мы рассматривали несколько алгоритмов кластеризации, а именно:

- $K - Means$  - метрический алгоритм, оптимизирующий внутрекластерное расстояние
- $DBSCAN$  - алгоритм, основанный на плотности расположения объектов

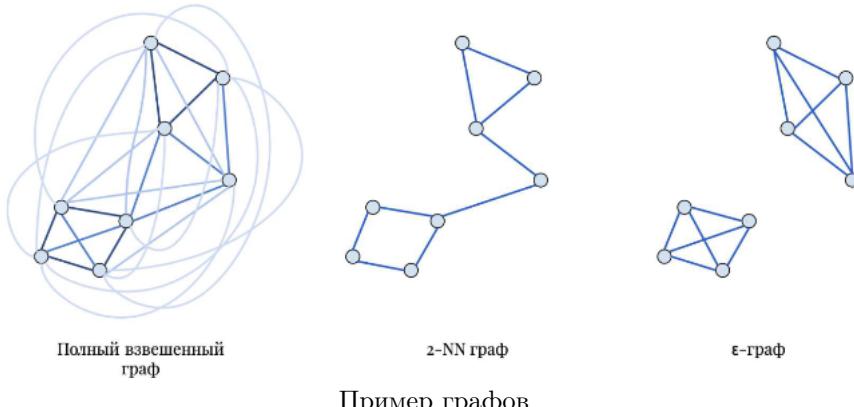
Попробуем изобрести немного другой подход. Мы можем представить объекты из выборки в виде вершин некоторого неориентированного графа  $G = (V, E)$ ,  $V = X = \{x_1, \dots, x_l\}$ . Рассмотрим несколько вариантов, как в таком представлении можно задать рёбра  $E$ :

- Граф  $G$  может быть полным с рёбрами, вес которых определяется по некоторой формуле, например

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Гиперпараметр  $\sigma$  определяет, насколько нам важны далёкие объекты.

- $G$  можно задать как  $kNN$ -граф, то есть объект  $x_i$  будет связан с  $k$  его ближайшими соседями.
- Вершина  $x_i$  может быть связана с теми вершинами, расстояние до которых меньше выбранного  $\varepsilon$ , то есть  $\rho(x_i, x_j) < \varepsilon$ . Такой граф будет называться  $\varepsilon$ -графом



Пример графов

#### 25.1.2. Лапласиан графа

Обозначим через  $W$  матрицу смежности графа  $G$ . Степени вершин будем считать как  $d_i = \sum_{j=1}^l w_{ij}$ . Пусть  $D = diag(d_1, \dots, d_l)$ . Тогда матрица  $L = D - W$  будет называться лапласианом графа  $G$ . Рассмотрим несколько свойств лапласиана  $L$ :

1. Пусть  $f \in \mathbb{R}^n$ . Тогда имеет место следующая формула:

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^l w_{ij} (f_i - f_j)^2$$

Проверка формулы:

$$\begin{aligned}
f^T L f &= f^T D f - f^T W f = \sum_{i=1}^l d_i f_i^2 - \sum_{i,j=1}^l w_{ij} f_i f_j = \sum_{i=1}^l \left( \sum_{j=1}^l w_{ij} \right) f_i^2 - \sum_{i,j=1}^l w_{ij} f_i f_j = \sum_{i,j=1}^l w_{ij} f_i^2 - \\
\sum_{i,j=1}^l w_{ij} f_i f_j &= \frac{1}{2} \sum_{i,j=1}^l w_{ij} f_i^2 + \frac{1}{2} \sum_{i,j=1}^l w_{ij} f_i^2 - \sum_{i,j=1}^l w_{ij} f_i f_j = \frac{1}{2} \sum_{i,j=1}^l w_{ij} f_i^2 + \frac{1}{2} \sum_{i,j=1}^l w_{ij} f_j^2 - \sum_{i,j=1}^l w_{ij} f_i f_j = \\
\frac{1}{2} \sum_{i,j=1}^l w_{ij} (f_i^2 - 2f_i f_j + f_j^2) &= \frac{1}{2} \sum_{i,j=1}^l w_{ij} (f_i - f_j)^2
\end{aligned}$$

2.  $L$  - симметрическая неотрицательно определённая матрица. Симметричность вытекает из неориентированности графа. Свойство неотрицательной определённости легко следует из первого пункта. Действительно, в обсуждённых методах построения графа  $w_{ij} \geq 0$ , притом  $(f_i - f_j)^2 \geq 0$ . Следовательно,  $f^T L f \geq 0$ , что и означает неотрицательную определённость.

Однако у лапласиана также есть свойство, которое поможет нам в задаче кластеризации. Сформулируем его в виде теоремы.

### 25.1.3. Теорема Лапласиана Графа

Пусть  $L$  - лапласиан графа  $G$ . Тогда выполнены следующие два пункта:

1. Собственное значение  $\lambda = 0$  матрицы  $L$  имеет кратность, равную числу компонент связности  $k$ .
2. Пусть  $A_1, \dots, A_k$  - компоненты связности графа  $G$ . Тогда векторы  $f_1, \dots, f_k$ , определяемые по формуле  $f_i = ([x_j \in A_i])_{j=1}^l$ , будут являться собственными векторами для  $\lambda = 0$ .

Доказательство:

Сперва рассмотрим случай  $k = 1$ . Поймём, почему  $\lambda = 0$  вообще является собственным значением матрицы  $L$ . Для этого рассмотрим вектор  $f = (1, \dots, 1)$ :

$$Lf = Df - Wf = \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_l \end{pmatrix} \times \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1l} \\ w_{21} & w_{22} & \dots & w_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{l1} & w_{l2} & \dots & w_{ll} \end{pmatrix} \times \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_l \end{pmatrix} - \begin{pmatrix} w_{11} + \dots + w_{1l} \\ \vdots \\ w_{l1} + \dots + w_{ll} \end{pmatrix} = 0$$

Теперь предположим, что существует собственный вектор  $f' \in \mathbb{R}^n : \exists p \neq q \rightarrow f'_p \neq f'_q$ , то есть неконстантный вектор, соответствующий  $\lambda = 0$ . Тогда  $Lf' = 0 \Rightarrow f'^T L f' = 0$ . Но рассматриваемый граф является связным. Значит существует путь  $p = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_{n-1} \rightarrow i_n = q$ . Поскольку вершины  $i_r$  и  $i_{r+1}$  соединены ребром, то  $w_{i_r i_{r+1}} > 0$ , а значит  $f'_{i_r} = f'_{i_{r+1}}$  (иначе получим  $f'^T L f' > 0$ ). Отсюда  $f'_p = f'_{i_1} = \dots = f'_{i_{n-1}} = f'_q$  - константный вектор. Получили противоречие  $\Rightarrow f'$  не является собственным вектором для  $\lambda = 0$ . Значит, мы доказали, оба пункта для  $k = 1$ .

Теперь пусть  $k > 1$ . Можно упорядочить вершины так, чтобы лапласиан  $L$  стал блочно-диагональной матрицей:

$$L = \begin{pmatrix} L_1 & 0 & \dots & 0 \\ 0 & L_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & L_k \end{pmatrix}$$

Блоки  $L_1, \dots, L_k$  будут являться лапласианами компонент связности графа  $G \Rightarrow \lambda = 0$  имеет кратность  $k$ , а собственный векторы  $f_1, \dots, f_k$  задаются по формуле

$$f_i = ([x_j \in A_i])_{j=1}^l$$

### 25.1.4. Гипотеза

Пусть есть похожие объекты  $x_j$  и  $x_k$ , то есть расстояние между ними невелико. Тогда у собственных векторов  $f_i$ , соответствующих маленьким собственным значениям, выполнено  $f_{ij} \approx f_{ik}$

Эта гипотеза имеет лишь эмпирическое доказательство. Однако алгоритм, построенный на этой гипотезе, позволяет достигать успеха в задаче кластеризации. Данный алгоритм называется спектральной кластеризацией.

### 25.1.5. Спектральная кластеризация

1. Строим по объектам граф  $G$  и лапласиан  $L = D - W$  за  $O(l^2)$
2. Находим нормированные собственные векторы  $u_1, \dots, u_m$  матрицы  $L$  за  $O(l^3)$
3. Составляем матрицу  $U \in \mathbb{R}^{l \times m} : U = (u_1 | \dots | u_m)$  за  $O(lm)$
4. Обучаем на этой матрице алгоритм  $K - Means$  с  $k$  кластерами за  $O(l^{mk+1})$

Если мы верим в нашу гипотезу, то для похожих объектов соответствующие координаты векторов  $u_1, \dots, u_m$  будут близки. Матрица  $U$  имеет размерность  $l \times m$ , поэтому можно посмоотреть на неё, как на матрицу объекты-признаки. При этом новыми признаками будут как раз собственные векторы  $u_1, \dots, u_m$ . В описанном выше алгоритме предлагается обучать на них  $K - Means$ , но никто не запрещает использовать эти признаки для других задач.

Как видно, алгоритм обладает высокой вычислительной сложностью. Но утверждается, что он позволяет достигать впечатляющих результатов в задаче кластеризации. Однако как оценить, насколько хорошо алгоритм справляется с задачей классификации? Этому вопросу посвящён остаток конспекта.

## 25.2. Оценка качества кластеризации

### 25.2.1. Метрика на разметке

Пусть существует разметка  $(y_1, \dots, y_n)$ , не участвующая при обучении. Мы не использовали эту разметку в качестве дополнительного признака, так как нам не хочется мотивировать модель данным признаком. Тогда предлагается ввести оценку качества алгоритма кластеризации при помощи этой разметки, саму же разметку тогда называют *gold standard*. Введём несколько требований к внешней метрике качества  $Q$ :

#### ВСТАВИТЬ КАРТИНКИ

- Гомогенность. Базовое свойство разделения разных объектов в разные кластеры.
- Полнота. Один кластер не должен дробиться на несколько маленьких.
- Rab-bag. Весь мусор должен быть в одном "мусорном" кластере, чтобы остальными кластеры были чистыми
- Cluster size vs. quantity. Лучше испортить один кластер с целью улучшить качество множества других.

### 25.2.2. BCubed

Единственной известной на данный момент метрикой, обладающей всеми четырьмя названными свойствами является BCubed. Она считается следующим способом. Пусть  $L(x)$  - gold standard,  $C(x)$  - номер кластера, выдаваемый рассматриваемым алгоритмом. Тогда рассмотрим несколько величин:

- $Correctness(x, x') = \begin{cases} 1 & C(x) = C(x') \wedge L(x) = L(x') \\ 0 & otherwise \end{cases}$
- $Precision - BCubed = Avg_x [Avg_{x':C(x)=C(x')} Correctness(x, x')]$
- $Recall - BCubed = Avg_x [Avg_{x':L(x)=L(x')} Correctness(x, x')]$

Тогда  $F$ -мера от определённых точности и полноты будет удовлетворять всем нужным нам требованиям.

## 26. Перевод статьи про спектральную кластеризацию

Статья - <https://towardsdatascience.com/spectral-clustering-aba2640c0d5b>

### 26.1. Введение

В этой статье мы рассмотрим все тонкости спектральной кластеризации для графов и других данных. Кластеризация - одна из основных задач в машинном обучении без учителя. Цель состоит в том, чтобы разметить немаркированные данные по кластерам, где, как мы надеемся, похожие точки объекты будут назначены в один и тот же кластер.

Спектральная кластеризация - это метод, уходящий корнями в теорию графов, где этот подход используется для идентификации сообществ узлов в графе на основе соединяющих их ребер. Этот метод является гибким и позволяет нам также группировать данные, не относящиеся к графу.

Спектральная кластеризация использует информацию из собственных значений (спектра) специальных матриц, построенных на основе графа или объектов. Мы узнаем, как построить эти матрицы, интерпретировать их спектр и использовать собственные векторы для распределения наших данных по кластерам.

### 26.2. Собственные векторы и собственные значения

Решающее значение для этого обсуждения имеет концепция собственных значений и собственных векторов. Для матрицы  $A$ , если существует вектор  $x$ , который не полностью равен 0, и скаляр  $\lambda$  такой, что  $Ax = \lambda x$ , то  $x$  называется собственным вектором  $A$  с соответствующим собственным значением  $\lambda$ .

Мы можем думать о матрице  $A$  как о операторе переноса. Большинство векторов окажутся где-то совершенно в другом месте, когда к ним будет применён оператор  $A$ , но собственные векторы изменяются только по величине. Величина, на которую собственный вектор масштабируется при применении оператора, зависит от  $\lambda$ .

#### 26.2.1. Реализация поиска собственных значений в Python

```
import numpy as np

# a 2x2 matrix
A = np.array([[0,1],[-2,-3])

# find eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(A)

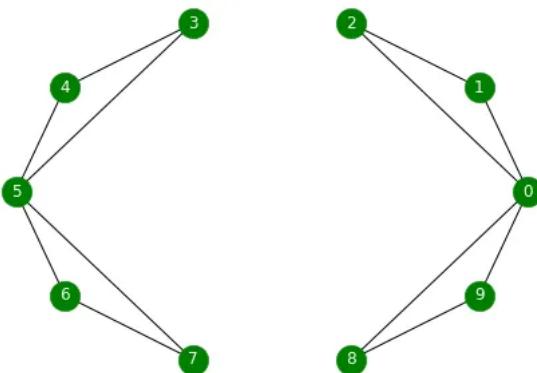
# print results
for i, value in enumerate(vals):
    print("Eigenvector:", vecs[:, i], ", Eigenvalue:", value)

# Eigenvector: [ 0.70710678 -0.70710678] , Eigenvalue: -1.0
# Eigenvector: [-0.4472136   0.89442719] , Eigenvalue: -2.0
```

### 26.3. Графы

Сеть маршрутизаторов в Интернете можно легко представить в виде графа. Маршрутизаторы - это узлы, а рёбра - это соединения между парами маршрутизаторов. Некоторые маршрутизаторы могут разрешать трафик только в одном направлении, поэтому рёбра могут быть направленными. Веса могут отображать скорость передачи данных. С помощью этой настройки мы могли бы затем запросить график, чтобы найти эффективные пути для передачи данных от одного маршрутизатора к другому по сети.

Давайте рассмотрим неориентированный график для примера:



Граф для примера

У этого графа 10 вершин и 12 рёбер. Также у этого графа есть две компоненты связности -  $\{0, 1, 2, 8, 9\}$  и  $\{3, 4, 5, 6, 7\}$ .

Делать кластеризацию на основе компонент связности - отличная идея. Но есть сложности в виде того, что весь граф может быть связан или что компоненты больше, чем нужно.

#### 26.4. Матрица смежности

Мы можем представить граф в виде матрицы смежности, где индексы строк и столбцов означают вершины, а записи представляют отсутствие или присутствие ребра между вершинами. Матрица смежности для нашего примера графика выглядит следующим образом:

```
A = np.array([
    [0, 1, 1, 0, 0, 0, 0, 0, 1, 1],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 1, 1, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 1, 0]
])
```

Если в  $i$ -й строке  $j$ -м столбце стоит 1, значит есть ребро из вершины  $i$  в вершину  $j$ . В противном случае ребра нет. Так как в нашем случае граф не ориентированный, то  $\forall i, j : A[i][j] = A[j][i]$ .

#### 26.5. Степень матрицы

Степенью узла называется кол-во рёбер, которые из него исходят. В случае ориентированных графов есть входящая и исходящая степень, но в нашем случае это не так. Для примера - вершина 0 имеет степень 4.

Матрица степеней - это матрица, на главной диагонали которой стоят степени вершин:

```
D = np.diag(A.sum(axis=1))
print(D)
```

```
# [[4 0 0 0 0 0 0 0 0 0]
# [0 2 0 0 0 0 0 0 0 0]
# [0 0 2 0 0 0 0 0 0 0]
# [0 0 0 2 0 0 0 0 0 0]
# [0 0 0 0 2 0 0 0 0 0]
# [0 0 0 0 0 4 0 0 0 0]
# [0 0 0 0 0 0 2 0 0 0]
# [0 0 0 0 0 0 0 2 0 0]
# [0 0 0 0 0 0 0 0 2 0]
# [0 0 0 0 0 0 0 0 0 2]]
```

## 26.6. Лапласиан Графа

Теперь мы собираемся найти Лапласиан графа. Лапласиан - это ещё одно метрическое представление графа. Он обладает некоторыми прекрасными свойствами, которыми мы воспользуемся для спектральной кластеризации. Чтобы вычислить нормальный Лапласиан (есть ещё несколько вариантов), мы просто вычтем из матрицы смежности диагональную матрицу степеней:  $L = D - A$

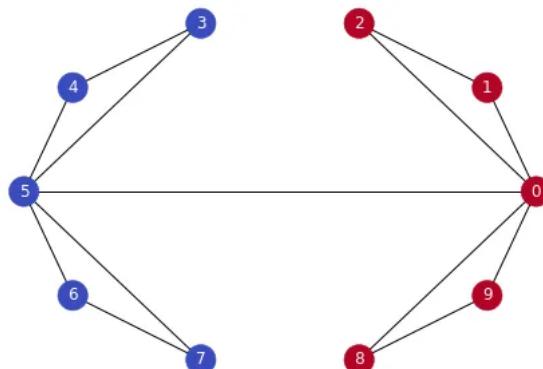
## 26.7. Собственные значения Лапласиана графа

Как уже упоминалось, лапласиан обладает некоторыми прекрасными свойствами. Чтобы понять это, давайте рассмотрим собственные значения, связанные с лапласианом, когда я добавляю ребра к нашему графику:

- Когда рёбер у графа нет - все собственные значения Лапласиана равны нулю.
- При добавлении рёбер собственные значения увеличиваются
- Количество собственных значений равных нулю отображает количество компонент связности.

Первое нулевое (в отсортированном по возрастанию смысле) значением называется **спектральным разрывом**. Спектральный разрыв даёт нам некоторое представление о плотности графа. Если бы этот граф был плотным, то спектральный разрыв был бы равен 10 ( $n$ , где  $n$  - количество вершин графа).

Второе собственное ненулевое значение называется значением **Фидлера**. Значение Фидлера приблизительно равно минимальному разрезу графа, необходимому для того, чтобы разделить на две компоненты связности. Если бы наш график состоял из двух связных компонент, то значение Фидлера было бы равно 0. Каждое значение в векторе Фидлера даёт нам информацию о том, к какой стороне разреза принадлежит этот узел. Давайте раскрасим узлы в зависимости от того, является ли их запись в векторе собственных значений положительной или нет:



Раскраска графа

Этот простой трюк разделил наш график на два кластера! Почему это работает? Помните, что нулевые собственные значения представляют собой связанные компоненты. Собственные значения, близкие к нулю, говорят нам о том, что существует почти полное разделение двух компонентов. Здесь у нас есть единственное преимущество, которое, если бы его не существовало, у нас было бы два отдельных компонента. Таким образом, второе собственное значение невелико.

Подводя итог тому, что мы знаем на данный момент: первое собственное значение равно 0, потому что у нас есть одна компонента связности. Второе собственное значение близко к 0, потому что мы находимся на расстоянии одного ребра от разреза компоненты связности на две разные компоненты связности. Мы также увидели, что вектор, связанный с этим значением, подсказывает нам, как разделить вершины на эти компоненты.

Наличие четырех собственных значений перед разрывом указывает на то, что, вероятно, существует четыре кластера. Векторы, связанные с первыми тремя положительными собственными значениями, должны дать нам информацию о том, какие три разреза необходимо сделать на графике, чтобы назначить каждый узел одному из четырех приближенных компонентов. Давайте построим матрицу из этих трех векторов и выполним кластеризацию K-средних для определения назначений:

```
from sklearn.cluster import KMeans
```

```
# our adjacency matrix
print("Adjacency Matrix:")
print(A)
```

```

# Adjacency Matrix:
# [[0.  1.  1.  0.  0.  1.  0.  0.  1.  1.]
# [1.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
# [1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
# [0.  0.  0.  0.  1.  1.  0.  0.  0.  0.]
# [0.  0.  0.  1.  0.  1.  0.  0.  0.  0.]
# [1.  0.  0.  1.  1.  0.  1.  1.  0.  0.]
# [0.  0.  0.  0.  1.  0.  1.  0.  0.  0.]
# [0.  0.  0.  0.  1.  1.  0.  0.  0.  0.]
# [1.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
# [1.  0.  0.  0.  0.  0.  0.  1.  0.  0.]]

# diagonal matrix
D = np.diag(A.sum(axis=1))

# graph laplacian
L = D-A

# eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(L)

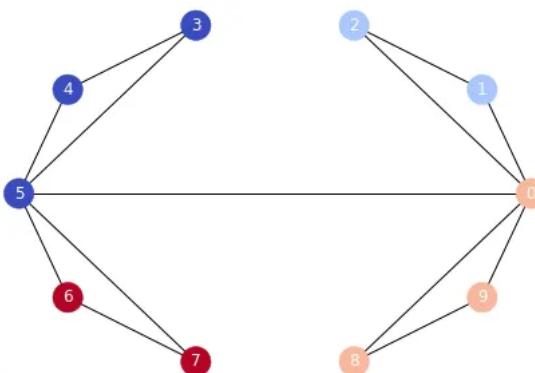
# sort these based on the eigenvalues
vecs = vecs[:, np.argsort(vals)]
vals = vals[np.argsort(vals)]

# kmeans on first three vectors with nonzero eigenvalues
kmeans = KMeans(n_clusters=4)
kmeans.fit(vecs[:, 1:4])
colors = kmeans.labels_

print("Clusters:", colors)

# Clusters: [2 1 1 0 0 0 3 3 2 2]

```



Спектральная кластеризация для 4-ёх кластеров

График был разделен на четыре кластера, причем вершины 0 и 5 произвольно назначены одному из кластеров. Это действительно круто, и это спектральная кластеризация!

Подводя итог:

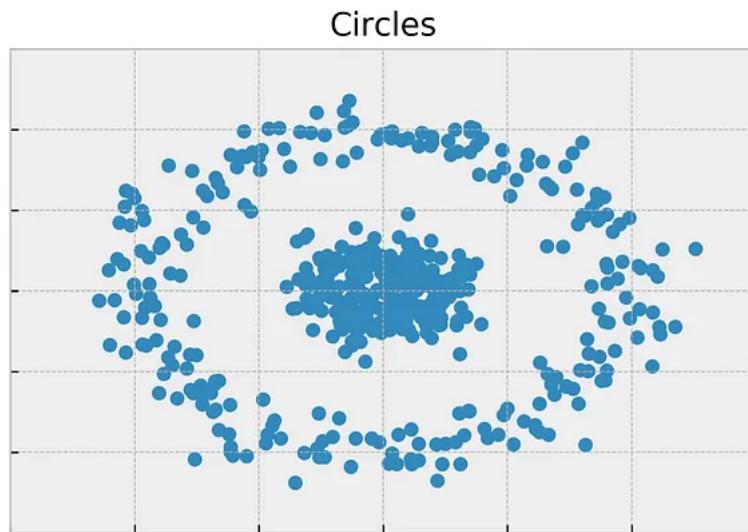
1. Мы взяли граф
2. Построили матрицу смежности
3. Нашли Лапласиан графа
4. Нашли собственные значения Лапласиана
5. Нашли собственные векторы Лапласиана

6. Применили  $K - Means$  для нахождения кластеров

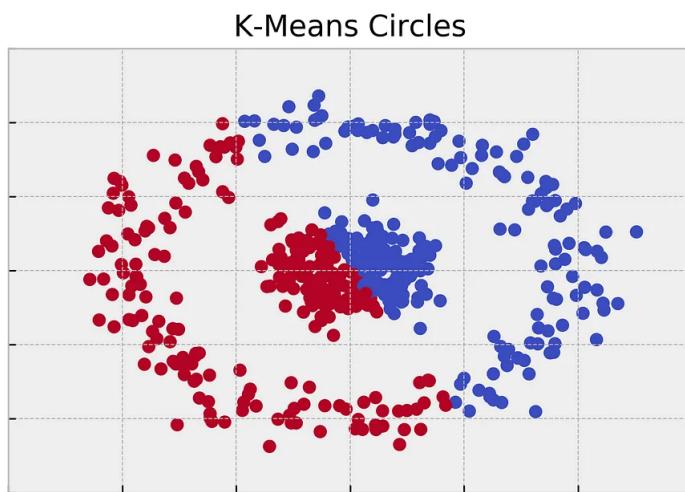
Теперь мы посмотрим как это сделать для произвольных данных.

## 26.8. Спектральная кластеризация для произвольных данных

Посмотрите на приведенные ниже данные. Точки нарисованы из двух концентрических окружностей с добавлением некоторого шума. Мы бы хотели, чтобы алгоритм мог сгруппировать эти точки в две окружности, которые их генерировали.



Эти данные представлены не в виде графа. Давайте просто попробуем алгоритм K-Means. K-Means найдет две центроиды и пометит точки, основываясь на том, к какому центроиду они также находятся ближе всего. Вот результаты для  $K - Means$  с количеством кластеров равным двум:



Естественно,  $K - Means$  не смог обработать окружность в окружности. Он работает на евклидовом расстоянии и предполагает, что кластеры имеют примерно сферическую форму. Эти данные (и часто данные из реального мира) опровергают эти предположения. Давайте попробуем решить эту проблему с помощью спектральной кластеризации.

## 26.9. Граф ближайших соседей

Есть несколько способов представить наши данные в виде графа. Самый простой способ - построить граф  $k$ -ближайших соседей. Граф  $k$ -ближайших соседей рассматривает каждый объект как вершину в графе. Затем от каждой вершины проводится ребро к его  $k$  ближайшим соседям в исходном пространстве. Как правило, алгоритм не слишком чувствителен к выбору  $k$ . Меньшие числа, такие как 5 или 10, обычно работают довольно хорошо.

Посмотрите на изображение данных еще раз и представьте, что каждая вершина связана со своими 5 ближайшими соседями. Любая вершина во внешнем кольце будет иметь только рёбра в это же внешнее кольцо, но во внутренний круг не будет ни одного рёбер. Довольно легко увидеть, что этот граф будет состоять из двух компонент связности: внешнего кольца и внутреннего круга.

Поскольку мы разделяем эти данные только на два компонента, мы должны быть в состоянии использовать наш предыдущий трюк с вектором Фидлера. Вот код, который я использовал для выполнения спектральной кластеризации этих данных:

```
from sklearn.datasets import make_circles
from sklearn.neighbors import kneighbors_graph
import numpy as np

# create the data
X, labels = make_circles(n_samples=500, noise=0.1, factor=.2)

# use the nearest neighbor graph as our adjacency matrix
A = kneighbors_graph(X, n_neighbors=5).toarray()
print(A)

# [[0.  0.  0.  ... 0.  0.  0.]
#  [0.  0.  0.  ... 0.  0.  0.]
#  [0.  0.  0.  ... 0.  0.  0.]
#  ...
#  [0.  0.  0.  ... 0.  1.  0.]
#  [0.  0.  0.  ... 0.  0.  0.]
#  [0.  0.  0.  ... 0.  0.  0.]]

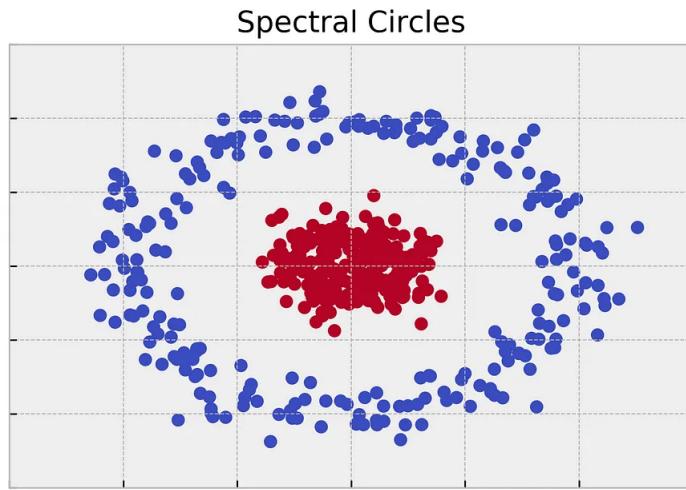
# create the graph laplacian
D = np.diag(A.sum(axis=1))
L = D-A

# find the eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(L)

# sort
vecs = vecs[:, np.argsort(vals)]
vals = vals[np.argsort(vals)]

# use Fiedler value to find best cut to separate data
clusters = vecs[:, 1] > 0
```

А вот и результат применения:



Применение спектральной кластеризации

## 26.10. Другие подходы

Граф ближайших соседей - хороший подход, но он основан на том факте, что “близкие” точки должны принадлежать одному кластеру. Иногда это может быть не так. Более общий подход заключается в построении матрицы сходства. Матрица сходства похожа на матрицу смежности, за исключением того, что значение для пары вершин выражает, насколько эти вершины похожи друг на друга. Если пары вершин абсолютно различны, то сходство должно быть равно 0. Если вершины идентичны, то сходство может быть равно 1. Таким образом, сходство действует подобно весам для ребер на нашем графике.

Как решить, что означает, что два объекта должны быть похожими, - это один из самых важных вопросов в машинном обучении. Часто знание предметной области является лучшим способом построения меры подобия. Если у вас есть доступ к экспертам по предметной области, задайте им этот вопрос.

Существуют также целые области, посвященные изучению того, как создавать показатели сходства непосредственно из данных. Например, если у вас есть некоторые помеченные данные, вы можете обучить классификатор предсказывать, похожи ли два входных сигнала или нет, основываясь на том, имеют ли они одинаковую метку. Затем этот классификатор можно использовать для присвоения сходства немаркированных точек.

## 26.11. Заключение

Мы рассмотрели теорию и применение спектральной кластеризации как для графов, так и для произвольных данных. Спектральная кластеризация - это гибкий подход к поиску кластеров, когда ваши данные не соответствуют требованиям других распространенных алгоритмов.

Сначала мы сформировали граф из наших данных. Рёбра в графе отражают сходство между точками. Собственные значения лапласиана графа затем могут быть использованы для нахождения наилучшего числа кластеров, а собственные векторы могут быть использованы для нахождения самих кластеров.

## 27. Лекция 12. Кластеризация-2. Наша лекция

### 27.1. Спектральная кластеризация

#### 27.1.1. Подключение библиотек

```
from sklearn.datasets import make_circles
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.cluster import SpectralClustering
import matplotlib.pyplot as plt
import pylab as pl
import networkx as nx
```

#### 27.1.2. Генерация данных

```
X_small, y_small = make_circles(n_samples=(250, 500),
                                random_state=3, noise=0.07, factor=0.1)
X_large, y_large = make_circles(n_samples=(250, 500),
                                 random_state=3, noise=0.07, factor=0.6)

y_large[y_large == 1] = 2

df = pd.DataFrame(np.vstack([X_small, X_large]), columns=['x1', 'x2'])
df['label'] = np.hstack([y_small, y_large])
df.label.value_counts()
sns.scatterplot(data=df, x='x1', y='x2', hue='label', style='label', palette='bright')
```

#### 27.1.3. Попробуем кластеризовать с помощью K-Means

```
x1 = np.expand_dims(df['x1'].values, axis=1)
x2 = np.expand_dims(df['x2'].values, axis=1)
X = np.concatenate((x1, x2), axis=1)
y = df['label'].values

from sklearn.cluster import KMeans

clustering = KMeans(n_clusters=3).fit(X)

colors = ['r', 'g', 'b']
colors = np.array([colors[label] for label in clustering.labels_])
plt.scatter(X[y == 0], X[y == 0, 1], c=colors[y == 0], marker='X')
plt.scatter(X[y == 1], X[y == 1, 1], c=colors[y == 1], marker='o')
plt.scatter(X[y == 2], X[y == 2, 1], c=colors[y == 2], marker='*')
plt.show()
```

#### 27.1.4. Попробуем кластеризовать с помощью DBSCAN

```
from sklearn.cluster import DBSCAN

clustering = DBSCAN(eps=10, min_samples=5).fit(X)

colors = ['r', 'g', 'b']
colors = np.array([colors[label] for label in clustering.labels_])
plt.scatter(X[y == 0], X[y == 0, 1], c=colors[y == 0], marker='X')
plt.scatter(X[y == 1], X[y == 1, 1], c=colors[y == 1], marker='o')
plt.scatter(X[y == 2], X[y == 2, 1], c=colors[y == 2], marker='*')
plt.show()
```

### 27.1.5. Попробуем спектральную кластеризацию

```
from sklearn.cluster import SpectralClustering

clustering = SpectralClustering(n_clusters=3, gamma=1000).fit(X)

colors = [ 'r' , 'g' , 'b' ]
colors = np.array([ colors [label] for label in clustering.labels_ ])
plt.scatter(X[y == 0], X[y == 0, 1], c=colors[y == 0], marker='X')
plt.scatter(X[y == 1], X[y == 1, 1], c=colors[y == 1], marker='o')
plt.scatter(X[y == 2], X[y == 2, 1], c=colors[y == 2], marker='*')
plt.show()
```

### 27.2. Сегментация изображений при помощи спектральной кластеризации

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction import image
from sklearn.cluster import spectral_clustering

l = 100
x, y = np.indices((l, l))

center1 = (28, 24)
center2 = (40, 50)
center3 = (67, 58)
center4 = (24, 70)

radius1, radius2, radius3, radius4 = 16, 14, 15, 14

circle1 = (x - center1[0]) ** 2 + (y - center1[1]) ** 2 < radius1 ** 2
circle2 = (x - center2[0]) ** 2 + (y - center2[1]) ** 2 < radius2 ** 2
circle3 = (x - center3[0]) ** 2 + (y - center3[1]) ** 2 < radius3 ** 2
circle4 = (x - center4[0]) ** 2 + (y - center4[1]) ** 2 < radius4 ** 2

#####
# 4 circles
img = circle1 + circle2 + circle3 + circle4

# We use a mask that limits to the foreground: the problem that we are
# interested in here is not separating the objects from the background,
# but separating them one from the other.
mask = img.astype(bool)

img = img.astype(float)
img += 1 + 0.2 * np.random.randn(*img.shape)

# Convert the image into a graph with the value of the gradient on the
# edges.
graph = image.img_to_graph(img, mask=mask)

# Take a decreasing function of the gradient: we take it weakly
# dependent from the gradient the segmentation is close to a voronoi
graph.data = np.exp(-graph.data / graph.data.std())

# Force the solver to be arpack, since amg is numerically
# unstable on this example
labels = spectral_clustering(graph, n_clusters=4, eigen_solver='arpack')
label_im = np.full(mask.shape, -1.)
```

```

label_im [ mask ] = labels

plt . matshow ( img )
plt . matshow ( label_im )

# ##### # 2 circles
img = circle1 + circle2
mask = img . astype ( bool )
img = img . astype ( float )

img += 1 + 0.2 * np . random . randn (*img . shape)

graph = image . img_to_graph ( img , mask=mask
graph . data = np . exp ( -graph . data / graph . data . std () )

labels = spectral_clustering ( graph , n_clusters=2 , eigen_solver='arpack' )
label_im = np . full ( mask . shape , -1 .)
label_im [ mask ] = labels

plt . matshow ( img )
plt . matshow ( label_im )

plt . show ()

```

## 27.3. Optuna

### 27.3.1. Пример использования optuna

```

import optuna

def objective ( trial ):
    x = trial . suggest_float ( 'x' , -10 , 10 )
    return ( x - 2 ) ** 2

study = optuna . create_study ()
study . optimize ( objective , n_trials=10 )

study . best_params

```

### 27.3.2. Ключевая идея Optuna

1. Определяем целевую функцию `objective`, через аргументы она будет получать специальный объект `trial`. С его помощью можно назначать различные гиперпараметры. Например, как в примере выше, мы задаём  $x$  в интервале от  $-10$  до  $10$ .
2. Далее создаём объект обучения с помощью метода `optuna.create_study`
3. Запускаем оптимизацию целевой функции `objective` на 10 итераций  $n\_trials = 10$ . Происходит 10 вызовов нашей функции с различными параметрами от  $-10$  до  $10$ . Какие именно параметры выбирает `optuna` будет описано ниже.

### 27.3.3. Что под капотом?

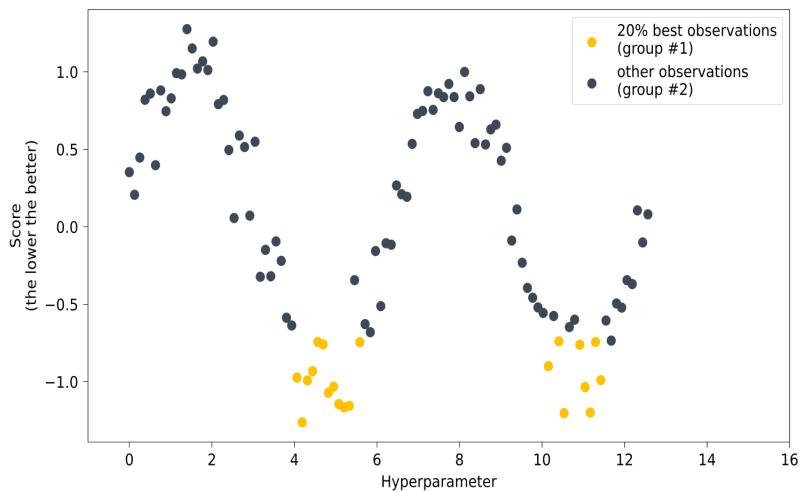
- Samplers - это набор алгоритмов для поиска гиперпараметров
- Pruners - это набор алгоритмов для прореживания экспериментов. Pruning - это механизм, который позволяет обрывать эксперименты, которые с большей долей вероятности приведут не к оптимальным результатам.

#### 27.3.4. Sampler: Tree-structured Parzen Estimator (TPE)

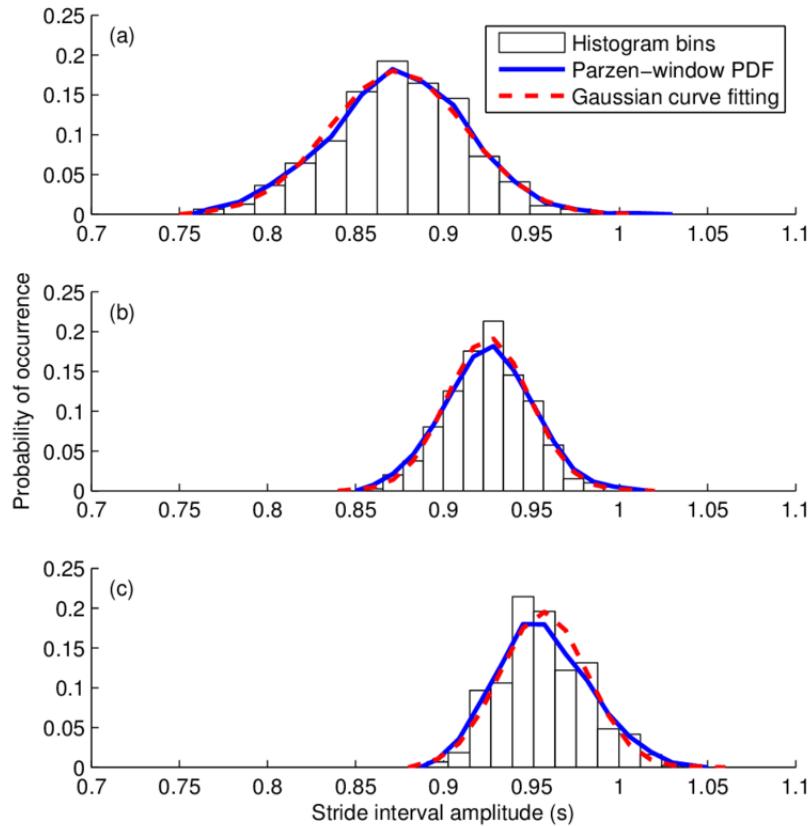
Общая идея: мы хотим чаще проверять гиперпараметры, близкие к тем, при которых модель показала хорошее качество на предыдущих попытках.

Предположим сначала, что мы хотим сделать поиск оптимального значения для одного гиперпараметра.

- На нескольких первых итерациях алгоритму требуется "разогрев": нужно иметь некоторую группу значений данного гиперпараметра, на которой известно качество модели. Самый простой способ собрать такие наблюдения - провести несколько итераций Random Search.
- Следующим шагом будет разделение собранных во время разогрева данных на две группы. В первой группе будут те наблюдения, для которых модель продемонстрировала лучшее качество, а во второй - все остальные. Размер доли лучших наблюдений задаётся пользователем: чаще всего это 10-25% от всех наблюдений.

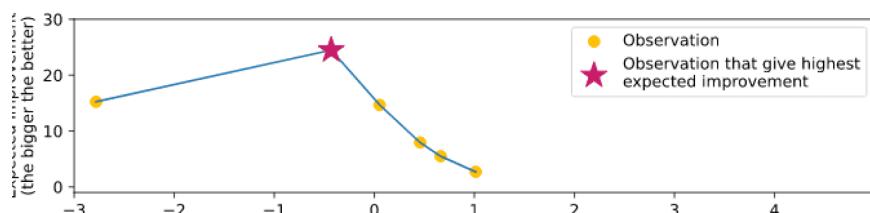
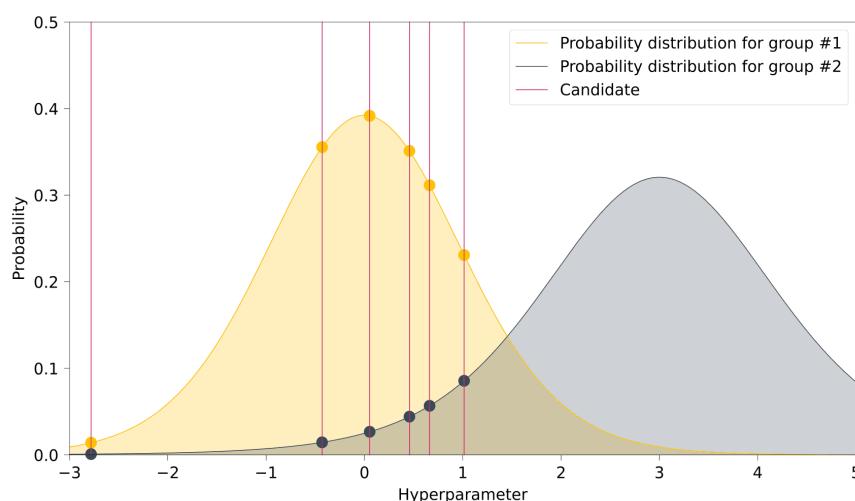


- Далее некоторым образом строятся оценки распределения  $l(x)$  лучших наблюдений и распределения  $g(x)$  всех остальных в пространстве значений рассматриваемого гиперпараметра.



4. На следующем шаге алгоритма мы сэмплируем несколько значений-кандидатов из распределения  $l(x)$ . Из насемплированных кандидатов мы хотим найти тех, кто с большей вероятностью окажется в первой группе (состоящей из лучших наблюдений), чем во второй. Для этого для каждого кандидата  $x$  вычисляется *Expected Improvement*:

$$EI(x) = \frac{l(x)}{g(x)}$$



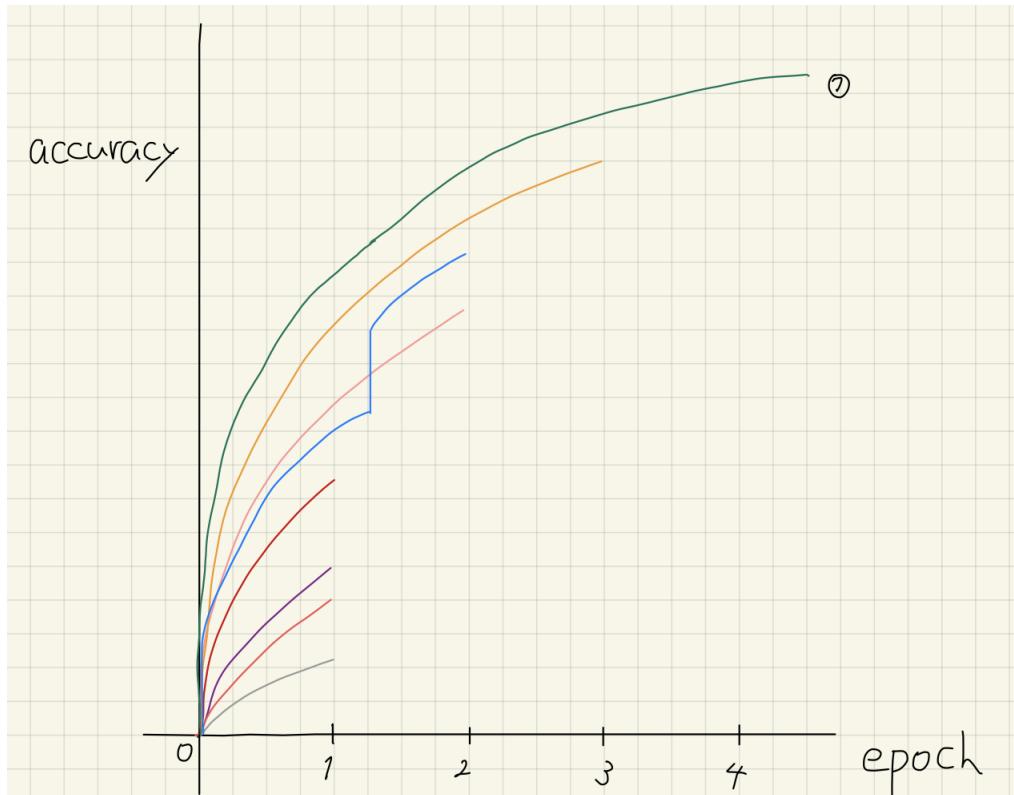
- После того как было выбрано значение-кандидат, максимизирующее  $EI$ , обучается модель с этим значением гиперпараметра.

После обучения мы замеряем её качество на валидационной выборке и в соответствии с этим результатом обновляем распределения  $l(x)$  и  $g(x)$ :

- снова ранжируем всех имеющихся кандидатов по качеству модели с учётом последнего, из топ 10 – 25%
- формируется обновлённое  $l(x)$ , из остальных —  $g(x)$

Так происходит столько раз, сколько итераций алгоритма мы задали.

#### 27.3.5. Pruner: Median Pruner



Эта картинка иллюстрирует вот что: мы можем сразу запустить всё на всех параметрах, но спустя определённое количество epochs обучения обрезать часть (в данном случае половину) веток.

#### 27.3.6. Successive Halving Algorithm (SHA) of Pruning

- Optuna параллельно запускает несколько процессов поиска гиперпараметров (несколько trials)
- Периодически оцениваем learning curves этих процессов:
  - изначально запускаем trials с минимальными ресурсами (мало обучающих примеров, мало итераций)
  - на каждом следующем шаге отсекаем половину trials с худшим качеством + увеличиваем ресурсы

#### 27.3.7. Пример запуска

```
import optuna

import sklearn.datasets
import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm
```

```

def objective(trial):
    iris = sklearn.datasets.load_iris()
    x, y = iris.data, iris.target

    classifier_name = trial.suggest_categorical("classifier", ["SVC", "RandomForest"])
    if classifier_name == "SVC":
        svc_c = trial.suggest_float("svc_c", 1e-10, 1e10, log=True)
        classifier_obj = sklearn.svm.SVC(C=svc_c, gamma="auto")
    else:
        rf_max_depth = trial.suggest_int("rf_max_depth", 2, 32, log=True)
        classifier_obj = sklearn.ensemble.RandomForestClassifier(
            max_depth=rf_max_depth, n_estimators=10
        )

    score = sklearn.model_selection.cross_val_score(classifier_obj, x, y, n_jobs=-1, cv=3)
    accuracy = score.mean()
    return accuracy

```

```

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)

```

В данном примере перебираются не все гиперпараметры, а в зависимости от выбранных других параметров.

### 27.3.8. Ещё один пример запуска

```

import numpy as np
import optuna

import catboost as cb
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


def objective_cb(trial):
    data, target = load_breast_cancer(return_X_y=True)
    train_x, valid_x, train_y, valid_y = train_test_split(data, target, test_size=0.3)

    param = {
        "objective": trial.suggest_categorical("objective", ["Logloss", "CrossEntropy"]),
        "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1),
        "depth": trial.suggest_int("depth", 1, 12),
        "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered", "Plain"]),
        "bootstrap_type": trial.suggest_categorical("bootstrap_type", ["Bayesian", "Bernoulli"]),
        "used_ram_limit": "3gb",
    }

    if param["bootstrap_type"] == "Bayesian":
        param["bagging_temperature"] = trial.suggest_float("bagging_temperature", 0, 10)
    elif param["bootstrap_type"] == "Bernoulli":
        param["subsample"] = trial.suggest_float("subsample", 0.1, 1)

    gbm = cb.CatBoostClassifier(**param)

    gbm.fit(train_x, train_y, eval_set=[(valid_x, valid_y)], verbose=0, early_stopping_rounds=10)

```

```

preds = gbm.predict(valid_x)
pred_labels = np.rint(preds)
accuracy = accuracy_score(valid_y, pred_labels)
return accuracy

study = optuna.create_study(direction="maximize")
study.optimize(objective_cb, n_trials=100, timeout=600)

print("Number_of_finished_trials : {}" .format(len(study.trials)))

print("Best_trial:")
trial = study.best_trial

print("Value:{}" .format(trial.value))

print("Params:")
for key, value in trial.params.items():
    print("{}:{}" .format(key, value))

```

Здесь как раз гиперпараметры задаются древоидно: для разных гиперпараметров можно задавать новые гиперпараметры.

#### **27.3.9. Как добавить optuna для своей модели?**

У optuna есть модуль optuna.integration, вот его нужно почитать и там будет как интегрировать.

## 28. Лекция 13. Рекомендательные системы. Наша лекция

### 28.1. Что такое рекомендательная система?

Рекомендательная система автоматически предсказывает товары/фильмы/музыку, которые могут заинтересовать пользователя на основе:

- прошлого поведения
- похожести на других пользователей
- похожести товаров/фильмов/музыки
- контекста (например: пользователь находится в поисковой выдаче по запросу "ipad")

### 28.2. Основные подходы

- **Collaborative Filtering**: рекомендуем товары, основываясь на прошлом поведении пользователи и всех остальных пользователей
- **Content-based**: рекомендации, основанные на похожести свойств товаров
- **Matrix Factorization**: рекомендации, основанные на разложении матрицы оценок "пользователь-товар" в произведение матриц меньшей размерности
- **Neural Networks**: рекомендации, полученные с помощью нейросетевых подходов

### 28.3. Collaborative Filtering

Рассмотрим матрицу взаимодействий "пользователь-товар"

	SHERLOCK	HOUSE OF CARDS	AVENGERS	ARRESTED DEVELOPMENT	Breaking Bad	THE WALKING DEAD
1	1	0		1		
0		1	1			1
				1	1	0
		1	1		0	
		1				1

У нас есть пользователи и контент (фильмы). Эта матрица отображает

- 1 - если пользователю понравился фильм
- 0 - если пользователю не понравился фильм
- пропуск - если пользователь не смотрел фильм

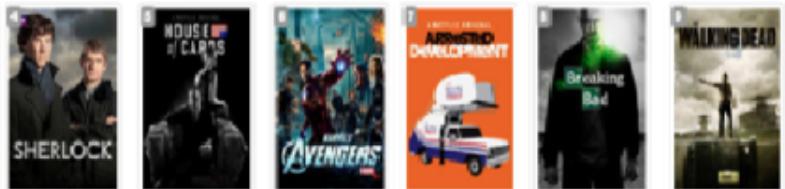
У нас есть на последней строчке женщина. Что же ей порекомендовать посмотреть?

### 28.3.1. User-based CF

Один из подходов называется Used-based CF.

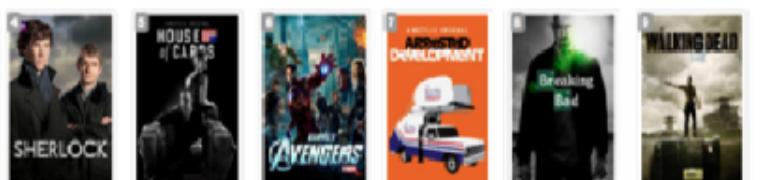
Идея: давайте найдём похожих на user пользователей и порекомендуем ему понравившиеся им товары.

На нашего пользователя похожи все хоть чуть-чуть (совпала хотя бы одна оценка), кроме третьего пользователя



	SHERLOCK	HOUSE OF CARDS	AVENGERS	ARRESTED DEVELOPMENT	BREAKING BAD	THE WALKING DEAD
1	1	1	0		1	
0	0	1	1			1
				1	1	0
	<td>1</td> <td>1</td> <td></td> <td>0</td> <td></td>	1	1		0	
G		1				1

Нужно среди этих пользователей найти такой фильм, что он понравился большинству и наш пользователь этот фильм ещё не смотрел:



	SHERLOCK	HOUSE OF CARDS	AVENGERS	ARRESTED DEVELOPMENT	BREAKING BAD	THE WALKING DEAD
1	1	1	0		1	
0	0	1	1			1
				1	1	0
	<td>1</td> <td>1</td> <td></td> <td>0</td> <td></td>	1	1		0	
G		1				1

Вопрос того, как это всё поставить в формальную оценку

**28.3.1.1. Явная и неявная оценка** Взаимодействие пользователя с товаром можно оценить не только по шкале "смотрел/не смотрел" "купил/не купил". Можно рассматривать оценки, выставленные пользователем товару.

- Явная оценка (explicit) это та оценка, которую пользователь сам осознано сделал: например, поставил оценку фильму по пятибалльной шкале.
- Неявная оценка (implicit) это та, которая получается от пользователя от обычных действий: например, факт просмотра товара/фильма, чтения поста в сети

Обычно такие матрицы разрежены, потому что пользователю очень сложно потребить весь контент сайта, а что уж тогда говорить про большинство пользователей.

При этом очевидно, что неявные оценки могут быть более запутанными: мы могли посмотреть фильм, он нам не понравился, но мы не сказали об этом. Но это хоть какая-то оценка

Как можно посчитать похожесть пользователей?

#### 28.3.1.2. Корреляция оценок

Пусть:

- $I_{uv}$  - множество товаров, оценённых пользователями  $u, v$
- $r_u, r_v$  - средняя оценка у пользователей  $u, v$

Тогда похожесть этих пользователей можно посчитать по формуле (корреляция Пирсона):

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - r_u)(r_{vi} - r_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - r_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - r_v)^2}}$$

Мы хотим попробовать предсказывать какую оценку пользователь сам может поставить товару.

#### 28.3.1.3. Прогноз

Прогноз - это средняя оценка пользователя + добавки от похожих пользователей с весами: Пусть

- $nn(u)$  - множество пользователей, похожих на пользователя  $u$  (можно найти как пользователей с корреляцией больше какой-то константы)
- $sim(u, v)$  - похожесть пользователей  $u$  и  $v$
- $p_{u,i}$  - оценка, которую пользователь  $u$  поставит товару  $i$

тогда

$$p_{u,i} = \frac{\sum_{v \in nn(u)} sim(u, v)(r_{v,i} - r_v)}{\sum_{v \in nn(i)} |sim(u, v)|}$$

#### 28.3.1.4. Минусы

- У большинства пользователей не так много оценок, что приводит к неуверенной оценки похожести пользователей
- Оценки конкретного пользователя меняются во времени, поэтому при добавлении хотя бы одной новой оценки его похожесть на других пользователей может сильно измениться

### 28.3.2. Item-based CF

Рассмотрим другой подход - item based collaborative filtering.

Идея: к оценённым пользователем товарам найдём наиболее похожие на них и порекомендуем пользователю

Все выкладки аналогичны User-based подходу, только вместо строк-пользователей теперь рассматриваем столбцы-товары.

Item-based решает проблему, когда у нас мало данных о пользователе.

#### 28.3.2.1. Плюсы Item-base CF

- Для популярных товаров можно получить надёжную оценку похожести (много оценок юзеров)
- Можно обновлять похожести товаров реже, например, раз в день

### 28.3.3. Обзор Collaborative Filtering

Плюсы:

- Достаточно неплохие рекомендации при большом количестве явных оценок

Минусы:

- Два пользователя должны оценивать одинаковые товары, оценка очень похожих товаров не учитывается в их близости.
- Проблема холодного старта: не знаем, что делать с новым товаром или пользователем.
- Сильная разреженность матрицы приводит к плохим рекомендациям: например, вероятность того, что два пользователя, которые купили 100 книжек каждый, имеют хотя бы одну общую покупку (в каталоге из миллиона книг) равна 0.01 (в случае 50 покупок и 10 миллионов получаем 0.00025)

## 28.4. Матричные разложения

### 28.4.1. Векторы интересов

Решаем задачу рекомендации пользователям различных фильмов.

Можно описать пользователя и фильм векторами интересов:

- для пользователя - насколько он интересуется каждым жанром
- для фильма - насколько он относится к каждому жанру

### 28.4.2. Рейтинг

Будем определять **заинтересованность** как **скалярное произведение** вектора пользователя и вектора фильма.

### 28.4.3. Модели со скрытыми переменными

У нас есть матрица рейтингов для задачи пользователь-фильм.

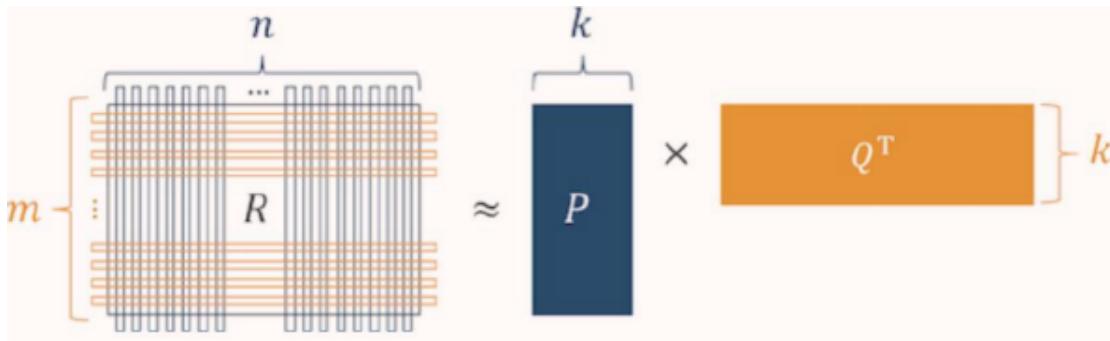
Цель: найти такие векторы пользователей и векторы фильмов, скалярное произведение которых максимально близко к рейтингам из таблицы.

	(0.9, 0.05)	(0.02, 1.1)	(1.05, 0.01)
(2.1, 5)	2	5	
(4.6, 0)	5		4
(0, 1)		1	
(4.9, 0.9)		1	5

Далее заполняем оставшуюся таблицу и даём пользователю фильм, у которого наибольшая оценка. На картинке эти векторы для размерности 2 уже найдены. А как их находить?

#### 28.4.4. Матричные разложения

Задачу нахождения этих векторов можно представить с помощью матричной факторизации, а именно, представить матрицу рейтингов как произведение двух матриц:



где

- $k$  - количество жанров
- Матрица  $P$  - векторы интересы пользователей
- Матрица  $Q$  - векторы фильмов

Почему мы хотим получить приблизительное разложение, а не точное? В матрице  $m$  много пропусков - мы не можем раскладывать матрицу с пропусками. Тогда давайте заполним пропуски нулями. Если мы будем точно восстанавливать разложение - наши пропуски-нули так и останутся нулями, а нам не очень хочется получать такие результаты. Тогда восстанавливать разложение будем приближённо.

Это называется матричной факторизацией.

Давайте для нашей задачи применим SVD. Собственные значения приблизительно равные нулю в матрице  $\Sigma$  просто занулим. А итоговая матрица после разложения  $SVD$  и будет нашей матрицей товарных предпочтений. Она не будет в точности равна исходной, но будет очень сильно на неё похожа. Вместо нулей в пропущенных значениях мы будем иметь какую-то оценку.

При этой операции у пользователя и товара появляются «латентные» признаки. Это признаки, показывающие «скрытое» состояние пользователя и товара.

Общее семейство подобных алгоритмов называется **NMF (non-negative matrix factorization)**. Как правило вычисление таких разложений весьма трудоёмко, поэтому на практике часто прибегают к их приближённым итеративным вариантам.

### 28.5. Хорошие свойства рекомендательной системы

Какими свойствами должна обладать хорошая рекомендательная система?

Если вы хотите купить туалетную бумагу, заходите на сайт, чтобы её купить, а вам предлагается купить туалетную бумагу - с одной стороны это хорошо, рекомендательная система предсказывает вашу покупку. С другой стороны это плохо - вы бы и так купили туалетную бумагу и нет никакого профита от того, что вам её порекомендовали купить.

#### 28.5.1. Diversity(Разнообразие)

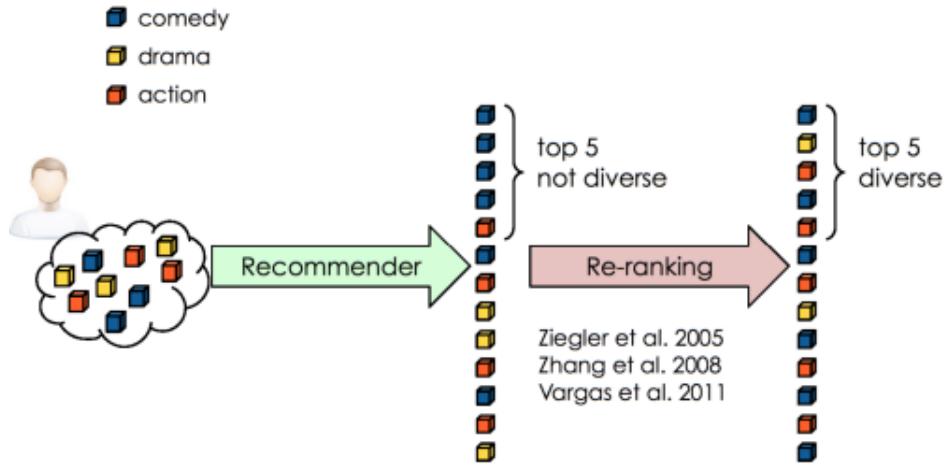
Если вы слушаете музыку и вам всё время подсказывают очень похожих между собой артистов - вам может это не очень сильно нравится, потому что вы можете хотеть попробовать что-то ещё. Какое-то разнообразие.

С точки зрения метрик сделать разнообразие невозможно.

Как же тогда достичь разнообразия?

Рекомендательная система обычно не говорит это пользователю показать/это не показать. Обычно модели выдают вероятность того, что пользователю понравится объект или какую-то ранговую характеристику относительно других объектов.

Давайте делать Re-ranking - если мы рекомендуем топ-5, то после отбора кандидатов мы делаем так, чтобы в реальном топе были разные товары, а не одинаковые:



В случае на картинке сверху, мы ухудшаем качество модели и нашей метрики, но увеличили Diversity и это хорошо.

### 28.5.2. Novelty (новизна)

Это нужно, чтобы постоянно не рекомендовать вам одно и то же. Вы когда заходите на ютуб, смотрите предложенные ролики, обновляете страницу - у вас могут попасться абсолютно новые ролики с новыми темами, которые вы до этого не смотрели.

Как оценить? Как улучшить?

### 28.5.3. Serendipity (способность удивить)

Порекомендовать Star Wars 2 тому, кто посмотрел Star Wars 1 - очевидно и бесполезно.

Опять же, мы спотыкаемся о проблему, что чтобы добавить serendipity - нужно ухудшать метрику модели, что опять невозможно.

Как оценить? Как улучшить?

## 28.6. Метрики

Если предсказываем рейтинг - можно использовать метрики качества регрессии.

Если предсказываем бинарное действие (купит/не купит) - метрики классификации.

Обычно показываем пользователю только  $k$  наиболее релевантных товаров  $R_u(k)$ , так что нужны еще такие метрики:

- $hitrate@k = [R_u(k) \cap L_u \neq 0]$ , где  $L_u$  - товары, которые пользователь купит
- $precision@k = \frac{|R_u(k) \cap L_u|}{|R_u(k)|}$
- $recall@k = \frac{|R_u(k) \cap L_u|}{|L_u|}$

### 28.6.1. Метрики качества ранжирования

Пусть:

- $a_{ui}$  - предсказание модели для пользователя  $u$  и товара  $i$  (вероятность, с которой пользователь купит товар)
- Отсортируем товары по убыванию предсказаний
- Для товара  $i_p$  на позиции  $p$  можно посчитать его полезность  $g(r_{ui_p})$  и штраф  $d(p)$

$$DCG@k(u) = \sum_{p=1}^k g(a_{ui_p})d(p)$$

Например,  $g(r) = 2^r - 1$ ,  $d(p) = \frac{1}{\log(p+1)}$

Пусть  $maxDCG@k$  - значение  $DCG$  при идеальном ранжировании, тогда

$$nGCD@k(u) = \frac{DCG@k}{MmaxDCG@k}$$

## 28.7. Практика

### 28.7.1. Импорт библиотек

```
%pylab inline

import numpy as np
import pandas as pd
import math

from tqdm import tqdm_notebook
```

### 28.7.2. Загрузка данных

Загрузим данные, в которых хранится логи платформы за 1 год, где пользователи читают статьи:

```
from google.colab import files

for fn in uploaded.keys():
    print(f'User_uploaded_file_{fn}' with length {len(uploaded[fn])} bytes. ')

# Then move kaggle.json into the folder where the API expected to find it
!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d gsmoreira/articles-sharing-reading-from-cit-deskdrop
!unzip articles-sharing-reading-from-cit-deskdrop.zip

articles_df = pd.read_csv('shared_articles.csv')
articles_df = articles_df[articles_df['eventType'] == 'CONTENT_SHARED']
articles_df.head(2)

interactions_df = pd.read_csv('users_interactions.csv')
interactions_df.head(10)
```

### 28.7.3. Предобработка данных

```
interactions_df.personId = interactions_df.personId.astype(str)
interactions_df.contentId = interactions_df.contentId.astype(str)
articles_df.contentId = articles_df.contentId.astype(str)

event_type_strength = {
    'VIEW': 1.0,
    'LIKE': 2.0,
    'BOOKMARK': 2.5,
    'FOLLOW': 3.0,
    'COMMENT CREATED': 4.0
}

interactions_df['eventStrength'] = interactions_df.eventType.apply(lambda x: event_type_strength[x])

interactions_df['eventStrength']
```

Просто так просуммировать взаимодействие пользователя - не очень хорошо, так как будет очень большой разброс. Мы мозьмём логарифм от нашей суммы.

Также мы хотели бы взять только тех пользователей, которые провзаимодействовали бы хотя бы с 5-ю материалами, чтобы не подвергаться проблеме холодного старта.

```
users_interactions_count_df = (
    interactions_df
    .groupby(['personId', 'contentId'])
    .first()
```

```

    .reset_index()
    .groupby('personId').size()
)
print('#users:', len(users_interactions_count_df))

users_with_enough_interactions_df = \
    users_interactions_count_df[users_interactions_count_df >= 5].reset_index()[['personId']]
print('#users with at least 5 interactions:', len(users_with_enough_interactions_df))

```

#### 28.7.4. Построим гистограмму взаимодействия

```
users_interactions_count_df.hist(bins=30)
```

Из этого видим, что в целом у пользователей не очень много взаимодействий и качество будет не очень хорошее.

Оставим только те взаимодействия, которые касаются только отфильтрованных пользователей:

```

interactions_from_selected_users_df = interactions_df.loc[np.in1d(interactions_df['personId'],
    users_with_enough_interactions_df)]
print('#interactions before:', interactions_df.shape)
print('#interactions after:', interactions_from_selected_users_df.shape)

```

#### 28.7.5. Сгладим количество взаимодействий

В данной постановке каждый пользователей мог взаимодействовать с каждой статьёй более 1 раза (как минимум совершая различные действия). Предлагается "схлопнуть" все действия в одно взаимодействие с весом, равным сумме весов.

Однако полученное число будет в том числе тем больше, чем больше действий произвёл человек. Чтобы уменьшить разброс предлагается взять логарифм от полученного числа (можно придумывать другие веса действиям и по-другому обрабатывать значения).

Также сохраним последнее значение времени взаимодействия для разделения выборки на обучение и контроль.

```

def smooth_user_preference(x):
    return math.log(1+x, 2)

interactions_full_df = (
    interactions_from_selected_users_df
    .groupby(['personId', 'contentId']).eventStrength.sum()
    .apply(smooth_user_preference)
    .reset_index().set_index(['personId', 'contentId'])
)
interactions_full_df['last_timestamp'] = (
    interactions_from_selected_users_df
    .groupby(['personId', 'contentId'])['timestamp'].last()
)
interactions_full_df = interactions_full_df.reset_index()
interactions_full_df.head(20)

```

#### 28.7.6. Разобъём выборку на train и test опираясь на временной фактор

```

from sklearn.model_selection import train_test_split

split_ts = 1475519530
interactions_train_df = interactions_full_df.loc[interactions_full_df.last_timestamp < split_ts]
interactions_test_df = interactions_full_df.loc[interactions_full_df.last_timestamp >= split_ts]

print('#interactions on Train set: %d' % len(interactions_train_df))
print('#interactions on Test set: %d' % len(interactions_test_df))

interactions_train_df

```

### 28.7.7. Изменим формат данных

Для удобства подсчёта качества запишем данные в формате, где строка соответствует пользователю, а столбцы будут истинными метками и предсказаниями в виде списков.

```
interactions = (
    interactions_train_df
    .groupby('personId')['contentId'].agg(lambda x: list(x))
    .reset_index()
    .rename(columns={'contentId': 'true_train'})
    .set_index('personId')
)

interactions['true_test'] = (
    interactions_test_df
    .groupby('personId')['contentId'].agg(lambda x: list(x))
)
# interactions.loc[pd.isnull(interactions.true_test), 'true_test'] = [
#     '' for x in range(len(interactions.loc[pd.isnull(interactions.true_test), 'true_test']))]

interactions.head(5)
```

### 28.7.8. Baseline (модель по популяции)

Самой простой моделью рекомендаций (при этом достаточно сильной!) является модель, которая рекомендует наиболее популярные предметы.

Реализуем её. Давайте считать, что рекомендуем мы по 10 материалов (такое ограничение на размер блока на сайте).

Посчитаем популярность каждой статьи, как сумму всех "оценок" взаимодействий с ней. Отсортируем материалы по их популярности.

```
popular_content = (
    interactions_train_df
    .groupby('contentId')
    .eventStrength.sum().reset_index()
    .sort_values('eventStrength', ascending=False)
    .contentId.values
)

print(articles_df.loc[articles_df.contentId == popular_content[1]]['title'].values)
print(articles_df.loc[articles_df.contentId == popular_content[2363]]['title'].values)
```

Теперь необходимо сделать предсказания для каждого пользователя. Не забываем, что надо рекомендовать то, что пользователь ещё не читал (для этого нужно проверить, что материал не встречался в true\_train).

```
top_k = 10

interactions['prediction_popular'] = (
    interactions.true_train
    .apply(
        lambda x:
            popular_content[~np.in1d(popular_content, x)][:top_k]
    )
)
interactions['prediction_popular'][0]
```

Настало время оценить качество. Посчитаем precision@10 для каждого пользователя (доля угаданных рекомендаций). Усредним по всем пользователям. Везде далее будем считать эту же метрику.

Что делать, если у нас в тесте меньше 10 статей для пользователя (например 5)? Как считать качество?

Мы возьмём долю угадывания не из 10, а из 5. Добавляем маленько 0.001, потому что в тесте может вообще ничего не быть.

```
def calc_precision(column):
    return (
        interactions
        .apply(
            lambda row:
                len(set(row['true_test'])) .intersection(
                    set(row[column]))) /
                min(len(row['true_test']) + 0.001, 10.0),
            axis=1).mean()

calc_precision('prediction_popular')
```

Качество получилось низкое, но с чем это связано:

- Матрица очень разреженая
- У пользователей мало интеракций

### 28.7.9. Коллаборативная фильтрация

Перейдём к более сложному механизму рекомендаций, а именно коллаборативной фильтрации. Суть коллаборативной фильтрации в том, что учитывается схожесть пользователей и товаров между собой, а не факторы, которые их описывают.

Для начала для удобства составим матрицу "оценок" пользователей. Нули будут обозначать отсутствие взаимодействия.

```
ratings = pd.pivot_table(
    interactions_train_df,
    values='eventStrength',
    index='personId',
    columns='contentId').fillna(0)
```

#### 28.7.9.1. Memory-based .

Посчитаем схожести пользователей с помощью корреляции Пирсона. Для каждой пары учитываем только ненулевые значения.

Для скорости работы лучше переходить от pandas к numpy.

Будем брать с рейтингами больше двух, чтобы мы смогли посчитать корреляцию.

```
ratings_m = ratings.values

similarity_users = np.zeros((len(ratings_m), len(ratings_m)))

for i in tqdm_notebook(range(len(ratings_m)-1)):
    for j in range(i+1, len(ratings_m)):

        # nonzero elements of two users
        mask_uv = (ratings_m[i] != 0) & (ratings_m[j] != 0)

        # continue if no intersection
        if np.sum(mask_uv) == 0:
            continue

        # get nonzero elements
        ratings_v = ratings_m[i, mask_uv]
        ratings_u = ratings_m[j, mask_uv]

        # for nonzero std
        if len(np.unique(ratings_v)) < 2 or len(np.unique(ratings_u)) < 2:
            continue
```

```

similarity_users[i, j] = np.corrcoef(ratings_v, ratings_u)[0, 1]
similarity_users[j, i] = similarity_users[i, j]

```

Теперь у нас есть матрицы схожести пользователей. Их можно использовать для рекомендаций. Для каждого пользователя:

1. Найдём пользователей с похожестью больше  $\alpha$  на нашего пользователя.
2. Посчитаем для каждой статьи долю пользователей (среди выделенных на первом шаге), которые взаимодействовали со статьёй.
3. Порекомендуем статьи с наибольшими долями со второго шага (среди тех, которые пользователь ещё не видел).

В нашем примере данных не очень много, поэтому возьмём  $\alpha = 0$ .

После того, как будут сделаны предсказания (новый столбец в interactions), посчитаем качество по той же метрике.

```

prediction_user_based = []

for i in tqdm_notebook(range(len(similarity_users))):
    users_sim = similarity_users[i] > 0

    if len(users_sim) == 0:
        prediction_user_based.append([])
    else:
        tmp_recommend = np.argsort(ratings_m[users_sim].sum(axis=0))[-1:-len(users_sim)-1:-1]
        tmp_recommend = ratings.columns[tmp_recommend]
        recommend = np.array(tmp_recommend)[~np.in1d(tmp_recommend, interactions.iloc[i])]
        prediction_user_based.append(list(recommend))

interactions['prediction_user_based'] = prediction_user_based
calc_precision('prediction_user_based')

```

### 28.7.10. Модель со скрытыми переменными

Реализуем подход с разложением матрицы оценок. Для этого сделаем сингулярное разложение (svd в scipy.linalg), на выходе вы получите три матрицы.

Заметим, что мы используем матрицу с нулями, будто отсутствующие взаимодействия негативные, что странно.

Если бы мы учили модель со скрытыми переменными с помощью стохастического градиентного спуска, то неизвестные взаимодействия могли бы не использовать.

```

from scipy.linalg import svd

U, sigma, V = svd(ratings)

print(ratings.shape, U.shape, sigma.shape, V.shape)

Sigma = np.zeros((1112, 2366))
Sigma[:1112, :1112] = np.diag(sigma)

new_ratings = U.dot(Sigma).dot(V)

print(sum(sum((new_ratings - ratings.values) ** 2)))

```

Значения у матрицы с сингулярными числами отсортированы по убыванию. Допустим мы хотим оставить только первые 100 компонент (и получить скрытые представления размерности 100). Для этого необходимо оставить 100 столбцов в матрице U, оставить из sigma только первые 100 значений (и сделать из них диагональную матрицу) и 100 столбцов в матрице V. Перемножим преобразованные матрицы ( $\hat{U}$ ,  $\hat{\sigma}$ ,  $\hat{V}^T$ ), чтобы получить восстановленную матрицу оценок.

```
K = 100
```

```
sigma[K:] = 0
Sigma = np.zeros((1112, 2366))
Sigma[:1112, :1112] = np.diag(sigma)
```

Посчитаем качество аппроксимации матрицы по норме Фробениуса (среднеквадратичную ошибку между всеми элементами соответствующими элементами двух матриц). Сравним его с простым бейзлайном с константным значением, равным среднему значению исходной матрицы. У аппроксимации ошибка должна получиться ниже.

```
new_ratings = U.dot(Sigma).dot(V)

print(sum(sum((new_ratings - ratings.values) ** 2)))
print(sum(sum((ratings.values.mean() - ratings.values) ** 2)))
```

Теперь можно делать предсказания по матрице. Сделаем их (не забываем про то, что уже было просмотрено пользователем), оценим качество. Для этого необходимо для каждого пользователя найти предметы с наибольшими оценками в восстановленной матрице.

```
new_ratings = pd.DataFrame(new_ratings, index=ratings.index, columns=ratings.columns)

predictions = []

for personId in tqdm_notebook(interactions.index):
    prediction = (
        new_ratings
        .loc[personId]
        .sort_values(ascending=False)
        .index.values
    )

    predictions.append(
        list(prediction[~np.in1d(
            prediction,
            interactions.loc[personId, 'true_train'])])[:top_k])
)

interactions['prediction_svd'] = predictions

calc_precision('prediction_svd')
```

## 29. Рекомендательные системы. Признаки в рекомендательных системах. Конспект Соколова. Первый конспект

Задача рекомендательной системы - выбрать для позователя объекты из некоторого множества, на которых он с наибольшей вероятностью совершил интересные нам действия. Это могут быть:

- Товары, которые пользователь захочет купить
- музыка, которую пользователь захочет дослушать до конца
- статьи, которые пользователь дочитает до конца
- видео, которые пользователь досмотрит до конца
- и многое другое

Мы будем рассуждать в терминах пользователей (*users*,  $U$ ), и товаров (*items*,  $I$ ), но все методы подходят для рекомендаций любых объектов. Будем считать, что для некоторых пар пользователей  $u \in U$  и товаров  $i \in I$  известны оценки  $r_{ui}$ , которые отражают степень заинтересованности пользователя в товаре. Вычисление таких оценок - отдельная тема. Например, в интернет-магазине заинтересованность может складываться из покупок товара и просмотров его страницы, причём покупки должны учитываться с большим весом. В социальной сети заинтересованность в материале может складываться из времени просмотра, кликов и явного отклика (лайки, репосты), это всё тоже должно суммироваться с различными весами. Не будем сейчас останавливаться на этом вопросе, а перейдём к основной задаче.

Требуется по известным рейтингам  $r_{ui}$  научиться строить для каждого пользователя  $u$  набор из  $k$  товаров  $I(u)$ , наиболее подходящих данному пользователю - то есть таких, для которых рейтинг  $r_{ui}$  окажется максимальным.

Получается понятная задача: для объекта  $x_{ui}$  "пользователь-товар" нужно предсказать значение цепевой переменной  $r_{ui}$ . Здесь требуют уточнения три пункта

1. целевая переменная
2. признаки
3. функционал ошибки

Первое мы затронули выше, третье обсудим позже, а сейчас поговорим о том, какими признаками можно охарактеризовать пару "пользователь-товар"

### 29.1. Коллаборативная фильтрация

Как понять, что пользователю может понравиться товар? Первый вариант - поискать похожих на него пользователей и посмотреть, что нравится им. Также можно поискать товары, похожие на те, которые этот пользователь уже покупал. Методы коллаборативной фильтрации строят рекомендации для пользователя на основе похожестей между пользователями и товарами. Мы рассмотрим два подхода к определению сходства.

#### 29.1.1. Memory-based

Два пользователя похожи, если они ставят товарам одинаковые оценки. Рассмотрим двух пользователей  $u$  и  $v$ . Обозначим через  $I_{uv}$  множество товаров  $i$ , для которых известны оценки обоих пользователей:

$$I_{uv} = \{i \in I | \exists r_{ui} \wedge \exists r_{vi}\}$$

Тогда сходство двух данных пользователей можно вычислить через корреляцию Пирсона:

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}}$$

где  $\bar{r}_u$  и  $\bar{r}_v$  - средние рейтинги пользователей по множеству товаров  $I_{uv}$ .

Чтобы вычислять сходства между товарами  $i$  и  $j$ , введём множество пользователей  $U_{ij}$ , для которых известны рейтинги этих товаров:

$$U_{ij} = \{u \in U | \exists r_{ui} \wedge \exists r_{uj}\}$$

Тогда сходство двух товаров можно вычислить через корреляцию Пирсона:

$$w_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}}$$

где  $\bar{r}_i$  и  $\bar{r}_j$  - средние рейтинги товаров по множеству пользователей  $U_{ij}$ . Отметим, что существуют и другие способы вычисления похожестей - например, можно вычислять скалярные произведения между векторами рейтингов двух товаров.

Мы научились вычислять сходства товаров и пользователей - разберём теперь несколько способов определения товаров, которые стоит рекомендовать пользователю  $u_0$ . В подходе на основе сходств пользователей, (user-based collaborative filtering) определяется множество  $U(u_0)$  пользователей, похожих на данного:

$$U(u_0) = \{v \in U | w_{u_0 v} > \alpha\}$$

После этого для каждого товара вычисляется, как часто он покупался пользователями из  $U(u_0)$ :

$$p_i = \frac{|\{u \in U(u_0) | \exists r_{ui}\}|}{|U(u_0)|}$$

Пользователю рекомендуются  $k$  товаров с наибольшими значениями  $p_i$ . Данный подход позволяет строить рекомендации, если для данного пользователя найдутся похожие. Если же пользователь является нетипичным, то подобрать что-либо не получится.

Также существует подход на основе сходств товаров (item-based collaborative filtering). В нём определяется множество товаров, похожих на те, которые интересовали данного пользователя:

$$I(u_0) = \{i \in I | \exists r_{u_0 i}, w_{i_0 i} > \alpha\}$$

Затем для каждого товара из этого множества вычисляется его сходство с пользователем:

$$p_i = \max_{i_0: \exists r_{u_0 i_0}} w_{i_0 i}$$

Пользователю рекомендуются  $k$  товаров с наибольшими значениями  $p_i$ . Даже если пользователь нетипичный, то данный подход может найти товары, похожие на интересные ему - и для этого необязательно иметь пользователя со схожими интересами.

### 29.1.2. Модели со скрытыми переменными

Все описанные выше подходы требуют хранения разреженной матрицы  $R = \{r_{ui}\}$ , которая может быть достаточно большой. Более того, они весьма эвристичны и зависят от выбора способа вычисления сходства, способа генерации товаров-кандидатов, способа их ранжирования. Алтернативой являются подходы на основе моделей со скрытыми переменными (latent factor models).

Мы будем пытаться построить для каждого пользователя  $u$  и товара  $i$  векторы  $p_u \in \mathbb{R}^d$  и  $q_i \in \mathbb{R}^d$ , которые будут характеризовать "категории интересов". Например, каждую компоненту такого вектора можно интерпретировать как степень принадлежности данного товара к определённой категории. Разумеется, никак не будет гарантироваться, что эти компоненты соответствуют каким-то осмысленным категориям, если только мы специально не потребуем этого от модели. По сути, векторы пользователей и товаров являются представлениями (embeddings), позволяющими свести эти сущности в одно векторное пространство.

Сходство пользователя и товара будем вычислять через скалярное произведение их представлений:

$$r_{ui} \approx \langle p_u, q_i \rangle$$

Также через скалярное произведение можно вычислять сходство двух товаров или двух пользователей. Мы можем записать функционал ошибки, исходя из способа вычисления сходства:

$$\sum_{(u,i) \in R} (r_{ui} - \bar{r}_u - \bar{r}_i \langle p_u, q_i \rangle)^2 \rightarrow \min_{P,Q} \quad (1)$$

Суммирование здесь ведётся по всем парам пользователей и товаров, для которых известен рейтинг  $r_{ui}$ . Заметим, что если  $R'$  - матрица  $R$  с центрированными строками и столбцами, то данная задача сводится к низкоранговому матричному разложению:

$$\|R; -P^T Q\|^2 \rightarrow \min_{P,Q}$$

Здесь представления пользователей и товаров записаны в столбцах матриц  $P$  и  $Q$ . Существуют модификации, в которых к скалярным произведениям добавляется масштабирующий множитель  $\alpha \in \mathbb{R}$ :

$$\|R' - \alpha P^T Q\|^2 \rightarrow \min_{P, Q, \alpha}$$

Данный функционал можно регуляризовать:

$$\sum_{(u, i) \in R} (r_{ui} - \bar{r}_u - \bar{r}_i - \langle p_u, q_i \rangle)^2 + \lambda \sum_{u \in U} \|p_u\|^2 + \mu \sum_{i \in I} \|q_i\|^2 \rightarrow \min_{P, Q} \quad (2)$$

Описанная модель носит название Latent Factor Model (LFM).

Отметим, что использование среднеквадратичной ошибки не всегда имеет смысл - в рекомендациях требуется выдать более высокие предсказания для товаров, которые более интересны пользователю, но вовсе не требуется точно предсказывать рейтинги. Впрочем, среднеквадратичную ошибку удобно оптимизировать. Более того, именно она использовалась в качестве функционала в конкурсе Netflix Prize, который во многом определили развитие рекомендательных систем, и в котором было предложено много популярных сейчас методов.

Существует два основных подхода к решению задачи (1). Первый - стохастический градиентный спуск, который на каждом шаге случайно выбирает пару  $(u, i) \in R$ :

$$\begin{aligned} p_{uk} &:= p_{uk} + \eta q_{ik} (r_{ui} - \bar{r}_u - \bar{r}_i - \langle p_u, q_i \rangle) \\ q_{ik} &:= q_{ik} + \eta p_{uk} (r_{ui} - \bar{r}_u - \bar{r}_i - \langle p_u, q_i \rangle) \end{aligned}$$

Второй подход основан на особенностях функционала (1) и называется ALS (alternating least squares). Можно показать, что этот функционал не является выпуклым в совокупности по  $P$  и  $Q$ , но при этом становится выпуклым, если зафиксировать либо  $P$ , либо  $Q$ . Более того, оптимальное значение  $P$  при фиксированном  $Q$  (и наоборот) можно выписать аналитически, - но оно будет содержать обращение матрицы:

$$\begin{aligned} p_u &= \left( \sum_{i: \exists r_{ui}} q_i q_i^T \right)^{-1} \sum_{i: \exists r_{ui}} r_{ui} q_i \\ q_i &= \left( \sum_{u: \exists r_{ui}} p_u p_u^T \right)^{-1} \sum_{u: \exists r_{ui}} r_{ui} p_u \end{aligned}$$

(здесь через  $p_u$  и  $q_i$  мы обозначили столбцы матриц  $P$  и  $Q$ ).

Чтобы избежать сложной операции обращения, будем фиксировать всё, кроме одной строки  $p_k$  матрицы  $P$  или одной строки  $q_k$  матрицы  $Q$ . В этом случае можно найти оптимальное значение  $p_k$  и  $q_k$ :

$$\begin{aligned} p_k &= \frac{q_k - \left( R - \sum_{s \neq k} p_s q_s^T \right)^T}{q_k q_k^T} \\ q_k &= \frac{p_k - \left( R - \sum_{s \neq k} p_s q_s^T \right)}{p_k p_k^T} \end{aligned}$$

Данный подход носит название Hierarchical alternating least squares (HALS).

### 29.1.3. Учёт неявной информации

Выше мы обсуждали, что интерес пользователя к товару может выражаться по-разному. Это может быть как явный (выставление рейтинга или лайк, написание рецензии с оценкой), так и неявный (просмотр видео, посещение страницы) сигнал. Неявным сигналам нельзя доверять слишком сильно - пользователь мог по многим причинам смотреть страницу товара. При этом неявной информации гораздо больше, и поэтому имеет смысл использовать её при обучении моделей.

Один из способов учёта неявной информации предлагается в методе Implicit ALS (iALS). Введём показатель неявного интереса пользователя к товару:

$$s_{ui} = \begin{cases} 1 & \exists r_{ui} \\ 0 & \text{otherwise} \end{cases}$$

Здесь мы считаем, что даже если пользователь поставил низкую оценку товару, то это все равно лучше ситуации, в которой пользователь совсем не поставил оценку. Это не очень сильные рассуждения - пользователь мог просто не найти товар, и в таком случае неправильно судить об отсутствии интереса. Поэтому введём веса  $c_{ui}$ , характеризующие уверенность в показателе интереса  $s_{ui}$ :

$$c_{ui} = 1 + \alpha r_{ui}$$

Коэффициент  $\alpha$  позволяет регулировать влияние явного рейтинга на уверенность в интересе. Теперь мы можем задать функционал:

$$\sum_{(u,i) \in D} c_{ui} (s_{ui} - \bar{s}_u - \bar{s}_i - \langle p_u, q_i \rangle)^2 + \lambda \sum_u \|p_u\|^2 + \mu \sum_i \|q_i\|^2 \rightarrow \min_{P,Q}$$

Как и раньше, обучать его можно с помощью стохастического градиентного спуска, ALS или HALS. Предложенные способы вычисления  $s_{ui}$  и  $c_{ui}$  могут изменяться в зависимости от специфики задачи.

#### 29.1.4. Факторизационные машины

Рассмотрим признаковое пространство  $\mathbb{R}^d$ . Допустим, что целевая переменная зависит от парных взаимодействий между признаками. В этом случае представляется разумным строить полиномиальную регрессию второго порядка:

$$\alpha(x) = w_0 + \sum_{j=1}^d w_j x_j + \sum_{j_1=1}^d \sum_{j_2=j_1+1}^d w_{j_1 j_2} x_{j_1} x_{j_2}$$

Данная модель состоит из  $\frac{d(d-1)}{2} + d + 1$  параметров. Если среди признаков есть категориальные с большим числом категорий (например, идентификатор пользователя), то после их бинарного кодирования числовых параметров станет слишком большим. Чтобы решить проблему, предположим, что вес взаимодействия признаков  $j_1$  и  $j_2$  может быть аппроксимирован произведением низкоразмерных скрытых векторов  $v_{j_1}$  и  $v_{j_2}$ , характеризующих эти признаки. Мы получим модель, называемую факторизационной машиной (factorization machine, FM):

$$\alpha(x) = w_0 + \sum_{j=1}^d w_j x_j + \sum_{j_1=1}^d \sum_{j_2=j_1+1}^d \langle v_{j_1}, v_{j_2} \rangle x_{j_1} x_{j_2}$$

Благодаря описанному трюку, число параметров снижается до  $dr + d + 1$ , где  $r$  - размерность скрытых векторов.

Данная модель является обобщением моделей с матричными разложениями. Задачу (1) можно сформулировать как задачу построения регрессии с двумя категориальными признаками: идентификатором пользователя и идентификатором товара. Целевым признаком является рейтинг  $r_{ui}$ . Для некоторого подмножества пар (пользователь, товар) мы знаем рейтинг. Для остальных мы хотим его восстановить. После бинаризации признаков получим, что каждый объект  $x$  описывается  $|U| + |I|$  признаками, причём ненулевыми являются ровно два из них: один соответствует номеру пользователя  $u$ , второй - номеру товара  $i$ . Тогда факторизационная машина пример следующий вид:

$$\alpha(x) = w_0 + w_u + w_i + \langle v_{u_i}, v_i \rangle$$

Данная форма полностью соответствует модели (1). По сути, факторизационная машина позволяет строить рекомендательные модели на основе большого количества категориальных и вещественных признаков.

Существует несколько методов настройки факторизационных машин, из которых наиболее совершенным считается метод Монте-Карло на основе марковских цепей. Реализацию можно найти в библиотеке libFM.

#### 29.1.5. FFM

Недавно было предложено расширение факторизационных машин, позволившее авторам победить в конкурсах Criterio и Avazu по предсказанию кликов по рекламным объявлениям. В обычных факторизационных машинах у каждого признака имеется всего один скрытый вектор, отвечающий за взаимодействие с остальными признаками. Допустим, что признаки можно некоторым образом сгруппировать - например, в задаче рекомендаций музыкальных альбомов в бинарном векторе, отвечающем за композиции, будет стоять несколько единиц, соответствующих всем композициям из альбома. Все единицы из этого вектора можно

объединить в одну группу. Расширим модель, введя для каждого признака разные скрытые векторы для взаимодействия с разными группами:

$$\alpha(x) = w_0 + \sum_{j=1}^d w_j x_j + \sum_{j_1=1}^d \sum_{j_2=j_1+1}^d \langle v_{j_1 f_{j_2}}, v_{j_2 f_{j_1}} x_{j_1} x_{j_2} \rangle$$

где  $f_{j_1}$  и  $f_{j_2}$  - индексы групп признаков  $x_{j_1}$  и  $x_{j_2}$ . Данная модель носит название field-aware factorization machines (FFM).

## 29.2. Контентные модели

В колаборативной фильтрации используется информация о предпочтении пользователей и об их сходствах, но при этом никак не используются свойства самих пользователе или товаров. При этом может быть полезно находить товары, которые своим описанием похожи на товары из истории пользователя. Особенно релевантно это может быть для рекомендательных систем контента (музыки, статей, видео), где пользователю, скажем, захочется познакомиться с музыкой, похожей на музыку его любимых исполнителей.

Как правило, это приводит к следующей идее: все товары описываются с помощью векторов (представлений, embeddings), и затем измеряется сходство между вектором нового товара и векторами товаров из истории пользователя. Можно вычислять минимальное или среднее расстояние до векторов из истории. Можно обучить линейную модель, которая для данного пользователя предсказывает целевую переменную на основе представления товара:

$$\sum_{i \in I: \exists r_{ui}} (\langle w_u, q_i \rangle - r_{ui})^2 \rightarrow \min_{w_u}$$

и затем с помощью этой модели оценивать, насколько пользователю подойдут другие товары. Можно обучить граф вычислений, который по всем данным о товаре и о пользователе пытается предсказать целевую переменную. Существует много методов, и какой из них подойдёт для данной задачи - заранее предсказать нельзя.

## 29.3. Статистические признаки

Важны и более простые типы факторов: конверсия просмотра данного товара в покупку за всю историю магазина, число покупок данного пользователя в категории данного товара, число покупок данного пользователя и т.д. Если товар или пользователь уже набрали достаточно статистики, то зачастую такие признаки оказываются самыми главными при принятии решения, поскольку уже содержат в себе достаточно информации о предпочтениях.

## 30. Рекомендательные системы. Метрики качества рекомендаций. Конспект Соколова. Второй конспект

Существует достаточно много метрик качества рекомендательных систем = некоторые связаны с точностью предсказания, а некоторые оценивают продуктивные аспекты (например, среднее качество тех фильмов, которые обычно рекомендуются). Нет общих советов по поводу того, на какую метрику имеет смысл обращать внимание. Один из возможных подходов - выбрать ключевую с точки зрения бизнеса онлайн-метрку (например, среднее время, которое пользователь проводит на сайте интернет-магазина, средний чек или что-то ещё), а затем выбрать офлайн-метрику или линейную комбинацию офлайн-метрик, которая лучше всего коррелирует с ключевой метрикой. Здесь под онлайн-метрикой понимается показатель, который можно измерить только при запуске рекомендательной системы на реальных пользователях, а под офлайн-метрикой - функцию, которую можно оценить, построив предсказания модели на исторических данных. Также иногда пытаются найти промежуточную онлайн-метрику, которая коррелирует с основной, но при этом быстрее реагирует на изменения в работе рекомендательной системы - но эту тему мы пока затрагивать не будем. Разберём несколько офлайн-метрик.

### 30.1. Качество предсказаний

Поскольку рекомендательная система обучается предсказывать оценки  $r_{ui}$ , логично оценивать качество решения именно этой задачи.

#### 30.1.1. Предсказание рейтингов

Если модель предсказывает рейтинг или другую вещественную величину (например, длительность просмотра), то качество может измеряться через MSE, RMSE, MAE или другие регрессионные метрики.

#### 30.1.2. Предсказание событий

Если модель предсказывает вероятность некоторого события (клика, покупки, просмотра, добавления в корзину), то качество можно измерять с помощью метрик качества классификации - доля правильных ответов, точность, полнота, F-мера, AUC-ROC, AUC-PR, log-loss и т.д.

Также можно учитывать, что мы показываем пользователю только  $k$  товаров, получивших самые высокие предсказывания модели, и нас интересует лишь качество этих товаров. Если через  $R_u(k)$  обозначить лучшие  $k$  товаров для пользователя  $u$  с точки зрения модели, а через  $L_u$  товары, для которых действительно произошло интересующее нас событие, то можно ввести следующие метрики:

- Наличие верной рекомендации:  $hitrate@k = [R_u(k) \cap L_u \neq \emptyset]$
- Точность:  $precision@k = \frac{|R_u(k) \cap L_u|}{R_u(k)}$
- Полнота:  $recall@k = \frac{|R_u(k) \cap L_u|}{|L_u|}$

#### 30.1.3. Качество ранжирования

Вообще говоря, нам не очень важно, насколько точно модель предсказывает рейтинг или вероятность клика - от неё лишь требуется дать более релевантным товарам более высокие предсказания. Это значит, что модель должна правильно ранжировать (или сортировать) товары.

Одной из популярных метрик качества ранжирования является  $nDCG$ . Обозначем через  $a_{ui}$  предсказание модели для пользователя  $u$  и товара  $i$ . Отсортируем все товары по убыванию предсказания  $a_{ui}$ . Тогда для товара  $i_p$  на позиции  $p$  можно вычислить его полезность  $g(r_{ui_p})$  и штраф за позицию  $d(p)$ . Метрика  $DCG$  задаётся как

$$DCG@k(u) = \sum_{p=1}^k g(r_{ui_p})d(p)$$

Примерами конкретных функций могут служить  $g(r) = 2^r - 1$  и  $d(p) = \frac{1}{\log(p+1)}$ . Чтобы значение метрики легче было интерпретировать, её можно поделить на значение  $DCG$  при идеальном ранжировании - в этом случае получим метрику  $nDCG$  (normalized DCG):

$$nDCG@k(u) = \frac{DCG@k(u)}{\max DCG@k(u)}$$

Далее значение  $nDCG$  можно усреднить по всем пользователям.

**30.1.3.1. Недостатки оценок качества предсказания** Основная проблема состоит в том, что качество предсказания само по себе не определяет пользу рекомендательной системы. Модель может идеально угадывать то, что купил пользователь - но, возможно, он приобрёл бы эти товары и без рекомендаций. Поскольку мы никогда не можем узнать, повлияли ли рекомендации на намерения пользователя, имеет смысл анализировать и другие метрики качества, которые могут косвенно говорить о пользе предсказаний модели.

## 30.2. Покрытие

### 30.2.1. Покрытие товаров

Полезно обращать внимание на то, какая доля товаров в принципе рекомендуется пользователям - так, может оказаться, что модель показывает только самые популярные товары, а большая часть ассортимента игнорируется. В качестве простейшей метрики можно использовать *покрытие каталога*, которое вычисляется как доля товаров, порекомендованных хотя бы один раз.

Также можно оценить общее разнообразие рекомендаций. пусть  $p(i)$  - доля показа товара  $i \in I$  среди всех показов для данной рекомендательной системы. Тогда разнообразие можно определить как энтропию такого распределения:

$$H(p) = - \sum_{i \in I} p(i) \log p(i)$$

### 30.2.2. Покрытие пользователей

Рекомендательная система может быть устроена так, что некоторым пользователям вообще ничего не рекомендуется - например, из-за низкой уверенности классификаторов или отсутствия тех или иных признаков для модели. Имеет смысл вычислять долю пользователей, для которых не рекомендуется ни одного товара, чтобы отслеживать проблемы с покрытием в модели рекомендаций.

## 30.3. Новизна

Под новизной понимается доля новых для пользователя товаров среди рекомендованных. При этом под новыми понимаются те товары, которые пользователь видит впервые глобально, а не только на нашем сайте - в идеале хочется уметь угадывать, какие товары пользователь встречал раньше на других ресурсах.

Можно предложить несколько подходов к измерению новизны:

- Для каждого рекомендованного товара добавить в интерфейсе возможность сообщить о том, что этот товар пользователь уже видел
- Удалить из обучающей выборки часть товаров, которые пользователь купил или посмотрел - тем самым мы будем моделировать ситуацию, в которой пользователь когда-то раньше узнал про этот товар, но в наших данных это не отражено. Далее будем оценивать новизну на основе того, как часто эти удалённые товары попадают в рекомендации.
- Можно считать, что пользователь с большей вероятностью встречал раньше популярные товары и с меньшей - непопулярные. Тогда новизну можно вычислять как долю угаданных рекомендательной системой товаров, где каждый товар имеет вес, обратно пропорциональный популярности этого товара.

## 30.4. Прозорливость (serendipity)

Под прозорливостью понимается способность рекомендательной системы предлагать товары, которые отличаются от всех купленных пользователем ранее. Например, если пользователь читал только книги конкретного автора, то рекомендацию хорошей с точки зрения пользователя книги, но от другого автора мы будем называть прозорливой.

Прозорливость можно измерять как долю рекомендаций, которые далеки от всех оценённых пользователем товаров. Рассмотрим пример с рекомендациями книг. Допустим, мы хотим измерить расстояние  $d(b, B)$  между новой книгой  $b$  и множеством уже оценённых книг  $B$ . Обозначим через  $c_{B,w}$  число книг от автора  $w$  в множестве  $B$ , а через  $c_B$  - максимальное количество книг от одного автора в  $B$ . Тогда расстояние можно определить как

$$d(b, B) = \frac{1 + c_B - C_B, w(b)}{1 + c_B}$$

где  $w(b)$  - автор книги  $b$ .

### **30.5. Разнообразие**

Под разнообразием понимается степень сходства товаров внутри одной пачке рекомендаций (то есть тех товаров, которые одновременно рекомендуются пользователю). Логично ожидать, что полезность набора из 10 чехлов фотоаппаратов ниже, чем набора из чехла, линзы, объектива, батареек и т.д. Именно это и должна оценивать метрика разнообразия. Можно её задавать как, например, среднее попарное расстояние между товарами в одной пачке. Расстояние может измеряться по каталогу (как далеко в дереве категорий товаров находятся эти два товара) или, например, по аналогии с item-to-item рекомендациями (насколько эти два товара пересекаются по множествам купивших их пользователей).

### **30.6. Архитектура рекомендательных систем**

В рекомендательной системе может участвовать очень большое количество товаров. При каждом посещении пользователем веб-страницы, где есть блок рекомендаций, необходимо выдать ему  $k$  наиболее подходящих товаров, причём достаточно быстро (пользователь не может ждать минуту, пока загрузится страница). В хорошей рекомендательной системе участвуют сотни признаков - их вычисление для каждого товара, а затем ещё и применение ко всем товарам градиентного бустинга или графа вычислений вряд ли получится успеть сделать за 1 секунду. Из-за этого рекомендательные системы работают в несколько этапов: обычно всё начинается с отбора кандидатов, где быстрая модель выбирает небольшое количество (тысячи или десятки тысяч) товаров, а затем только для этих товаров вычисляется полный набор признаков и применяется полноценная модель. В качестве быстрой модели может выступать линейная модель на нескольких самых важных признаках или, например, простая колаборативная модель.

# 31. Статья про FM и модификации. Перевод

## 31.1. Введение

В этой статье мы представим факторизационные машины (FM) как гибкую и мощную платформу моделирования для рекомендаций по collaborative filtering. Затем мы опишем, как специализированные функции потерь, непосредственно оптимизирующие порядок ранжирования товаров, позволяют применять FM-модели к данным с учётом неявной информации. В заключение мы продемонстрируем эти моменты на реальном наборе данных с неявной информацией с использованием новой библиотеки FM-моделирования с открытым исходным кодом. Вперед!

## 31.2. Рекомендательные системы

Поскольку цифровая экономика продолжает расширяться в размерах и становиться все более изощренной, роль рекомендательных систем в предоставлении персонализированного, релевантного контента каждому отдельному пользователю становится более важной, чем когда-либо. Сайты электронной коммерции с постоянно увеличивающимися каталогами товаров могут одновременно предоставлять индивидуальные показы миллионам пользователей. Поставщики цифрового контента помогают пользователям найти тот самый контент, который им нравится. А ведь если бы пользователи пытались сделать это сами - могли бы и за всю жизнь не найти. Например, Netflix сообщила в 2015 году, что ее рекомендательная система влияла примерно на 80% часов потокового вещания на сайте, и далее оценила стоимость системы более чем в 1 млрд долларов в год.

Два основных подхода к рекомендационным системам - это фильтрация на основе содержимого (Content-Based Filtering, CBF) и коллаборативная фильтрация (Collaborative Filtering, CF). Модели CBF представляют пользователей и товары в виде векторов атрибутов или характеристик (например, возраст, местоположение, доход, уровень активности пользователя; отдел, категория, жанр, цена товара). В отличие от этого, методы CF опираются только на прошлое поведение пользователей: модель анализирует со-путствующие шаблоны, чтобы определить сходства между пользователями и/или товарами и пытается вывести предпочтения пользователя по невидимым товарам, используя только записанные взаимодействия пользователя. Подходы на основе CF имеют преимущества в том, что они не зависят от домена (т.е. не требуется специальных знаний о бизнесе или инженерном обеспечении функций), а также обычно более точны и масштабирумы, чем модели CBF.

## 31.3. Матричное разложение

Многие из наиболее популярных и успешных подходов к коллаборативной фильтрации (CF) основаны на матричном разложении (MF), развитие которого ускорилось благодаря конкурсу Netflix Prize в 2006-2009 годах, в котором победитель использовал техники MF, включая теперь популярный алгоритм SVD++. Модели MF пытаются научиться низкоразмерным представлениям, или вложениям, пользователей и товаров в общем скрытом пространстве факторов. По сути, наблюдаемая разреженная матрица взаимодействия пользователей и товаров "факторизуется" на приближенное произведение двух матриц малого ранга, содержащих вложения пользователей и товаров. После того, как эти скрытые факторы изучены, можно вычислить сходство между пользователями/товарами и определить неизвестные предпочтения, сравнивая скрытые представления пользователей/товаров.

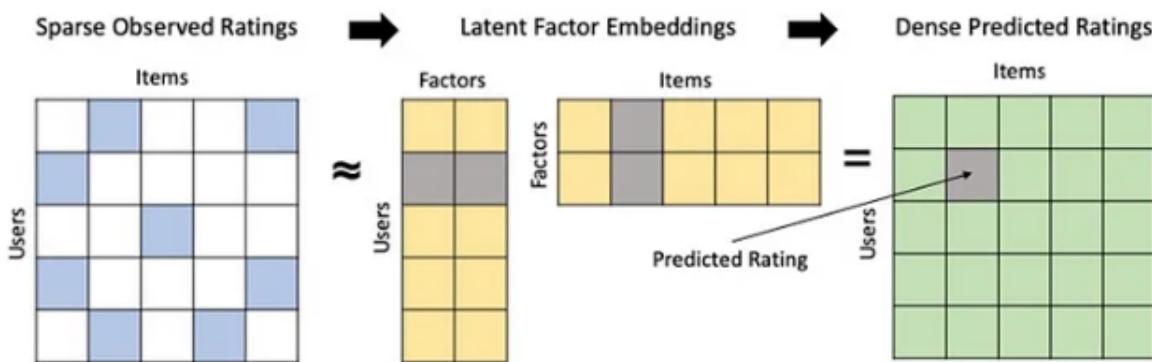


Image by Author

Многие популярные алгоритмы MF учатся скрытым факторам пользователей и товаров, минимизируя квадратичную ошибку между наблюдаемыми и предсказанными оценками, где предсказанные оценки

вычисляются как скалярное произведение соответствующих скрытых факторов пользователя и товара. Некоторые специальные модели дополнительно включают глобальные смещения пользователей/товаров и/или регуляризационные члены для предотвращения переобучения. Обычная функция потерь для MF может быть выражена следующим образом:

$$\sum_{(u,i) \in S} (r_{ui} \langle v_u, v_i \rangle)^2 + \lambda(\|v_u\|^2 + \|v_i\|^2) \rightarrow \min_{u,i}$$

Однако эта формулировка не работает при работе со скрытыми факторами, где вы не наблюдаете явных числовых оценок или положительных/отрицательных ответов, а только сырое поведение пользователя (например, просмотры, просмотры страниц, покупки, клики). Скрытые факторы являются более распространенными фичами в контексте рекомендаций в реальном мире, и на самом деле рекомендательные системы, построенные только на основе явной информации (даже если она есть), обычно плохо работают из-за того, что оценки отсутствуют случайным образом, а вместо этого сильно коррелируют со скрытыми предпочтениями пользователя. Для адаптации подхода MF к неявной информации ввели концепцию оценок как бинарных подразумеваемых предпочтений (видел ли пользователь товар или нет) с весом уверенности, представляющим предполагаемую силу этого бинарного предпочтения. Эта модельная формулировка является основой популярного неявного алгоритма обратной связи ALS, реализованного как в библиотеке SparkML, так и в библиотеке Implicit Python:

$$\sum_{(u,i) \in S} c_{ui} (p_{ui} - \langle v_u, v_i \rangle)^2 + \lambda(\|v_u\|^2 + \|v_i\|^2) \rightarrow \min_{v_u, v_i}$$

Несмотря на простоту и эффективность, этот подход имеет несколько ключевых недостатков. Во-первых, при представлении данных в виде матрицы взаимодействий пользователя/элемента невозможно включить дополнительные функции, такие как атрибуты пользователя/элемента, используемые в моделях CBF, и/или другую контекстную информацию о самом взаимодействии. Это большая упущененная возможность, когда существуют богатые вспомогательные функции, а также не позволяет модели генерировать информативные прогнозы для новых пользователей/элементов, что часто называют проблемой холодного старта. Во-вторых, кодирование неявной обратной связи пользователя в виде бинарных (0, 1) оценок и минимизация ошибки предсказания модели является очень косвенным представлением пользовательских предпочтений — вы не знаете наверняка, что наблюдаемые элементы на самом деле являются негативными для пользователя, и, конечно же, некоторые положительные (и отрицательные) элементы предпочтительнее других с точно таким же закодированным (0, 1) рейтингом.

Подумайте, как большинство рекомендаций на самом деле предоставляется пользователям: в виде одного или нескольких упорядоченных списков элементов. Таким образом, интуитивно понятно, что истинная цель должна состоять в том, чтобы получить правильное ранжирование элементов для каждого пользователя. Это позволит нам генерировать рекомендации, в которых все элементы упорядочены по релевантности для каждого пользователя, при этом наиболее релевантные элементы отображаются в самом верху списка рекомендуемых элементов каждого пользователя.

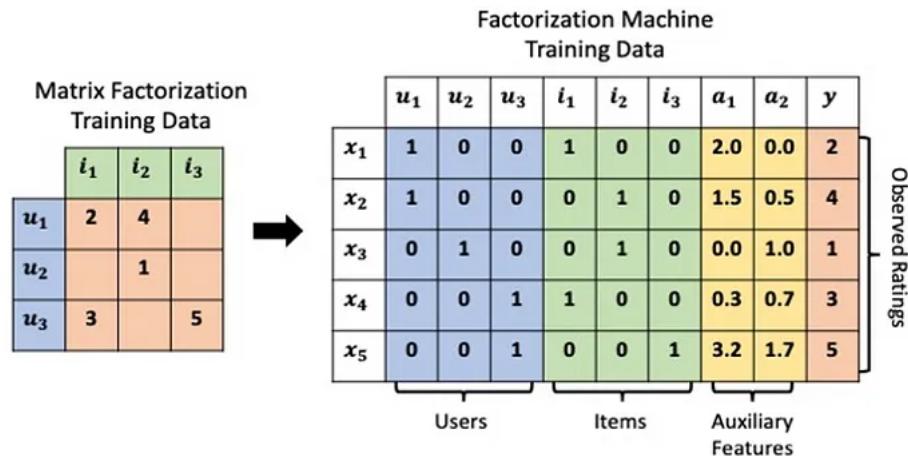
Чтобы преодолеть эти ограничения, нам нужна более общая структура модели, которая может расширить подход со скрытыми факторами, чтобы включить произвольные вспомогательные функции и специализированные функции потерь, которые напрямую оптимизируют порядок ранжирования элементов, используя неявные данные обратной связи. Добро пожаловать в мир машин факторизации и обучение ранжированию.

### 31.4. Машины факторизации

Машины факторизации (FM) — это общие модели обучения с учителем, которые отображают произвольные вещественные признаки в низкоразмерное пространство скрытых факторов и могут естественным образом применяться к широкому кругу задач прогнозирования, включая регрессию, классификацию и ранжирование. FM могут точно оценивать параметры модели на очень разреженных данных и обучаться с линейной сложностью, что позволяет масштабировать их до очень больших наборов данных — эти характеристики делают FM идеальным решением для реальных задач рекомендаций. В отличие от рассмотренной выше классической модели MF, которая вводит матрицу взаимодействия пользователя и элемента, модели FM представляют взаимодействия пользователя и элемента в виде кортежей действительных векторов признаков и числовых целевых переменных.

Обычно для совместной фильтрации базовыми фичами будут бинарные векторы индикаторы пользователя и элемента, так что каждая обучающая выборка имеет ровно две ненулевые записи, соответствующие данной комбинации пользователя/элемента. Однако эти индикаторы пользователя/элемента могут быть дополнены произвольными вспомогательными функциями, например, атрибутами пользователя или эле-

мента и/или контекстными функциями, относящимися к самому взаимодействию (например, день недели, порядок добавления в корзину и т. д.).



Уравнение модели FM состоит из  $n$ -сторонних взаимодействий между функциями. Модель второго порядка (безусловно, наиболее распространенная) включает веса для каждой базовой функции, а также условия взаимодействия для каждой попарной комбинации функций.

$$f(x) = w_0 + \sum_{p=1}^P w_p x_p + \sum_{p=1}^{P-1} \sum_{q=p+1}^P w_{p,q} x_p x_q$$

Эта формулировка модели может показаться знакомой — это просто квадратичная линейная регрессия. Однако, в отличие от полиномиальных линейных моделей, которые оценивают каждый член взаимодействия отдельно, FM вместо этого используют факторизованные параметры взаимодействия: веса взаимодействия признаков представлены как внутреннее произведение вложений скрытого факторного пространства двух признаков:

$$f(x) = w_0 + \sum_{p=1}^P w_p x_p + \sum_{p=1}^P \sum_{q=p+1}^P \langle v_p, v_q \rangle x_p x_q$$

Это значительно уменьшает количество оцениваемых параметров и в то же время облегчает более точную оценку, нарушая строгие критерии независимости между скрытыми взаимодействиями. Рассмотрим реалистичный набор данных рекомендаций с 1 000 000 пользователей и 10 000 объектов. Квадратичная линейная модель должна была бы оценить  $|U| + |I| + |UI| \approx 10^{10}$  параметров. FM-модели размерности  $F = 10$  потребовалось бы только  $|U| + |I| + |F|(|U| + |I|) \approx 11 \cdot 10^6$  параметров. Кроме того, многие распространённые алгоритмы MF (включая SVD++, ALS) могут быть переформулированы как частные случаи более общего/гибкого класса моделей FM.

Однако не сразу становится очевидным, как адаптировать модели FM для данных неявной обратной связи. Один наивный подход состоял бы в том, чтобы пометить все наблюдаемые взаимодействия пользователя и элемента как (1), а все ненаблюденные взаимодействия как (-1) и обучить модель, используя общую функцию потерь классификации, такую как log loss. Но с реальными рекомендательными наборами данных это потребовало бы создания миллиардов ненаблюдаемых обучающих выборок пользовательских элементов и привело бы к серьезному дисбалансу классов из-за разреженности взаимодействия. Этот подход также имеет ту же концептуальную проблему, что и рассмотренная выше адаптация MF со скрытой обратной связью: он по-прежнему минимизирует ошибку прогнозирования рейтинга вместо прямой оптимизации ранжирования элементов.

### 31.5. Learning-to-Rank

Методы оптимизации, которые изучают порядок ранжирования напрямую, а не минимизируют ошибку прогнозирования, называются обучением-рангу (LTR). Модели LTR обучаются на парах или списках обучающих выборок, а не на отдельных наблюдениях. Функции потерь основаны на относительном порядке элементов, а не на их необработанных оценках. Модели, использующие LTR, дали самые современные результаты в области поиска, извлечения информации и совместной фильтрации. Эти методы являются ключом к адаптации FM-моделей к проблемам рекомендаций с неявной обратной связью.

Одним из самых популярных методов LTR для рекомендации товаров является байесовский персонализированный рейтинг (BPR). BPR пытается изучить правильное ранжирование элементов для каждого пользователя, максимизируя апостериорную вероятность (MAP) параметров модели с учетом набора данных наблюдаемых предпочтений пользователя в отношении элементов и выбранного априорного распределения. Предполагается, что наблюдаемые элементы каждого пользователя (неявная обратная связь) предпочтительнее ненаблюдаемых элементов, а все парные предпочтения считаются независимыми. Чтобы изучить эти предпочтения, создаются обучающие выборки, состоящие из кортежей [пользователь ( $u$ ), наблюдаемый элемент ( $i$ ), ненаблюдаемый элемент ( $j$ )], и максимизируется следующая функция логарифмического правдоподобия по отношению к параметрам модели.

$$\ln[p(>_u | \theta)p(\theta)] \rightarrow \max_{\theta}$$

где  $(>_u | \theta)$  - предсказанный моделью рейтинг элемента для пользователя ( $u$ ). Это можно узнать, используя обучающие данные, описанные выше, путем максимизации совместной вероятности того, что наблюдаемые пользователями элементы предпочтительнее, чем их ненаблюдаемые элементы. Мы можем определить эту вероятность как разницу между прогнозируемыми показателями полезности наблюдаемых ( $i$ ) и ненаблюдаемых ( $j$ ) элементов пользователя, отображенных на  $[0, 1]$  с помощью сигмоидной функции:

$$p(>_u | \theta) = \prod_{(u,i,j) \in S} \sigma[f(u, i|\theta) - f(u, j|\theta)]$$

Где действительные оценки полезности пользовательского элемента  $f(u, i|\theta)$  и  $f(u, j|\theta)$  генерируются с использованием основного уравнения модели FM, приведенного выше. Собрав все вместе и включив термин регуляризации, критерий максимизации становится следующим:

$$\sum_{(u,i,j) \in S} \ln(\sigma[f(u, i|\theta) - f(u, j|\theta)]) - \lambda \|\theta\|^2 \rightarrow \max_{\theta} \rightarrow \max_{\theta}$$

Делая шаг вперед, взвешенный приблизительный парный ранг (WARP) не просто выбирает ненаблюдаемые элементы ( $j$ ) случайным образом, а выбирает множество ненаблюдаемых элементов для каждой наблюдаемой обучающей выборки, пока не найдет изменение ранга для пользователя, что дает более информативное обновление градиента. Это особенно важно в контексте с большим количеством элементов и сильно искаженной популярностью элементов (очень часто).

Алгоритм:

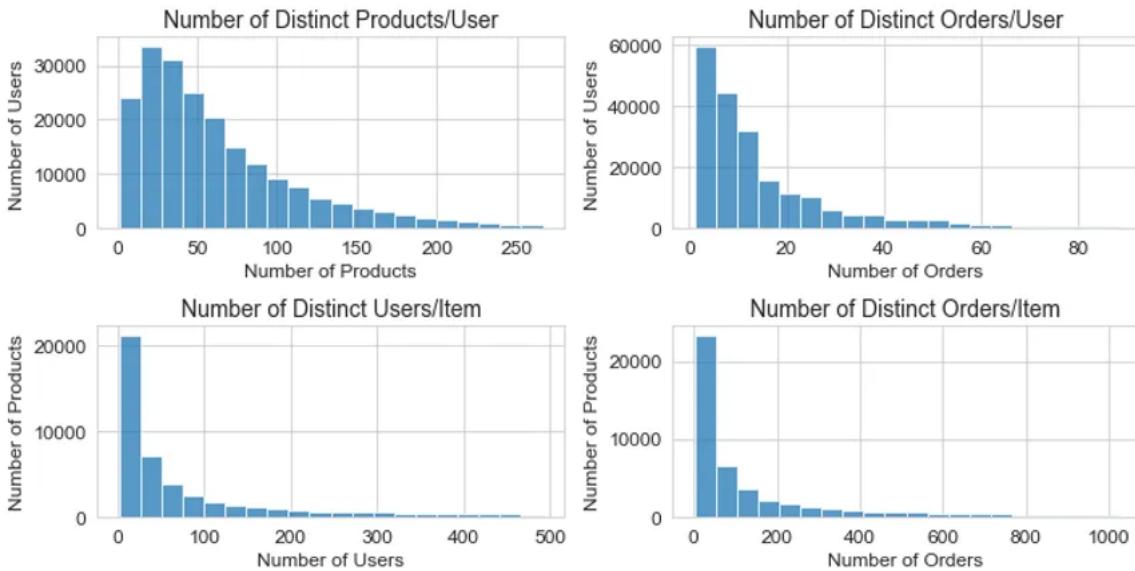
- Произвольно выберите ненаблюдаемый элемент для пользователя и вычислите его показатель полезности. Если оценка ненаблюдаемого элемента превышает оценку наблюдаемого элемента плюс фиксированный запас, выполните обновление градиента, в противном случае продолжайте выборку отрицательных элементов.
- Масштабируйте величину обновления градиента на основе количества выбранных отрицательных элементов, прежде чем обнаруживать нарушение поля — делайте меньшие обновления, если было выбрано больше отрицательных элементов, поскольку более вероятно, что модель в настоящее время правильно ранжирует пользовательские предпочтения.

Фактически, если вы масштабируете величину обновлений градиента с помощью множителя  $(0, 1]$ , BPR можно рассматривать как частный случай WARP, где максимальное количество отрицательных выборок равно единице, что приводит к постоянному множителю обновления градиента из 1. Использование WARP увеличивает время обучения на эпоху по сравнению с BPR, но часто приводит к более быстрой сходимости и превосходной производительности модели.

### 31.6. Оценка модели

Теперь, когда вся теория убрана, давайте посмотрим, как эти компоненты объединяются для получения высококачественных рекомендаций на основе известного набора реальных данных. Мы обучим модель FM с неявной обратной связью, используя новый авторский пакет RankFM, который реализует описанные выше методы, и сравним его производительность с популярным алгоритмом MF с неявной обратной связью, обсуждавшимся ранее (через пакет Implicit). Цель состоит в том, чтобы показать, что модель FM, включающая дополнительные функции и обученная с использованием методов оптимизации LTR, обеспечивает более высокую производительность по сравнению с классической моделью MF с аналогичными спецификациями.

For this exercise, we'll use the 2017 Instacart Orders Data. It contains 200,000 users, 50,000 items, 3.4 million orders, and over 32 million total recorded user-item interactions. A quick exploratory look at the data reveals high sparsity: the median user has purchased only 48 items across 9 orders and the median item has been purchased by only 35 users across 60 orders. The overall interaction sparsity rate is 99.87%.



Мы случайным образом разделим общие данные о взаимодействии, используя 75% для обучения модели, а оставшиеся 25% — для оценки показателей проверки и оценки производительности модели. Чтобы отразить идею о том, что повторные покупки несут более сильные сигналы предпочтения, мы включим веса выборки, определенные с помощью журнала подсчета покупок каждого пользовательского товара. Мы дополним основные данные взаимодействия пользователя с товаром набором характеристик товара, обозначающих отдел супермаркета и проход каждого товара. К сожалению, в этом наборе данных нет дополнительных пользовательских функций.

```
import numpy as np
import pandas as pd

valid_pct = 0.25
interactions[ 'random' ] = np.random.random( size=len( interactions ) )

train_mask = interactions[ 'random' ] < ( 1 - valid_pct )
valid_mask = interactions[ 'random' ] >= ( 1 - valid_pct )

interactions_train = interactions[train_mask].groupby([ 'user_id', 'product_id' ]).size().to_
interactions_valid = interactions[valid_mask].groupby([ 'user_id', 'product_id' ]).size().to_

sample_weight_train = np.log2( interactions_train[ 'orders' ] + 1 )
sample_weight_valid = np.log2( interactions_valid[ 'orders' ] + 1 )

interactions_train = interactions_train[ [ 'user_id', 'product_id' ] ]
interactions_valid = interactions_valid[ [ 'user_id', 'product_id' ] ]

train_users = np.sort( interactions_train.user_id.unique() )
valid_users = np.sort( interactions_valid.user_id.unique() )
cold_start_users = set( valid_users ) - set( train_users )

train_items = np.sort( interactions_train.product_id.unique() )
valid_items = np.sort( interactions_valid.product_id.unique() )
cold_start_items = set( valid_items ) - set( train_items )

item_features_train = item_features[item_features.product_id.isin( train_items )]
item_features_valid = item_features[item_features.product_id.isin( valid_items )]
```

Теперь давайте обучим модель RankFM. Мы будем использовать 50 скрытых факторов, потерю WARP с максимальным количеством отрицательных выборок 100 и график обучения с обратным масштабированием, который со временем снижает скорость обучения, чтобы помочь с конвергенцией SGD. RankFM распечатает логарифмическую вероятность каждой эпохи обучения, чтобы вы могли отслеживать как время обучения, так и прирост производительности от эпохи к эпохе:

```
from rankfm.rankfm import RankFM
```

```
model = RankFM(factors=50, loss='warp', max_samples=100, learning_schedule='invscaling')
model.fit(interactions_train, item_features=item_features_train, sample_weight=sample_weight)

training epoch: 0
log likelihood: -291678

training epoch: 1
log likelihood: -285965
```

```
training epoch: 38
log likelihood: -232521
```

```
training epoch: 39
log likelihood: -231900
```

Мы можем сгенерировать действительные оценки полезности, используя основное уравнение модели FM с методом прогнозирования(). Пользователи и элементы, отсутствующие в обучающих данных, могут быть либо удалены, либо в результирующем векторе оценок могут быть установлены оценки пр.пап. Мы можем использовать метод рекомендацию() для создания рекомендуемых элементов TopN для каждого пользователя в наборе проверки. Есть полезный флагок, который позволяет вам выбрать, следует ли включать ранее наблюдаемые учебные элементы или генерировать только рекомендации новых элементов. Результатом является pandas DataFrame со значениями индекса UserID и рекомендуемыми элементами каждого пользователя в столбцах, упорядоченных слева направо в соответствии с ожидаемыми предпочтениями:

Наконец, мы оценим метрики производительности на заблокированных данных проверки. RankFM включает в себя отдельный модуль оценки, который реализует многие популярные рекомендательные показатели, включая hit rate, reciprocal rank, discounted cumulative gain, и precision/recall.

```
from rankfm.evaluation import hit_rate, precision, recall
```

```
model_hrt = hit_rate(model, interactions_valid, k=10)
model_pre = precision(model, interactions_valid, k=10)
model_rec = recall(model, interactions_valid, k=10)
```

```
model hit rate: 0.815
model precision: 0.254
model recall: 0.139
```

Используя набор данных с 50 000 уникальных элементов, где средний пользователь покупает менее 50, мы можем получить процент попаданий при проверке выше 80% с очень небольшим проектированием функций и необходимой подготовкой данных. Не слишком потерянный.

### 31.7. Сравнение с бейзлайном

Теперь давайте сравним производительность RankFM с нашим базовым алгоритмом ALS. Для этого нам сначала нужно преобразовать наши данные о взаимодействии пользователя с элементом в разреженную матрицу CSR. Мы будем использовать точно такое же разделение данных и вводить наши сгенерированные веса выборки в качестве показателей достоверности для обучения модели ALS.

```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
```

```
# create zero-based index position <-> user/item ID mappings
```

```

index_to_user = pd.Series(np.sort(np.unique(interactions_train['user_id'])))
index_to_item = pd.Series(np.sort(np.unique(interactions_train['product_id'])))

# create reverse mappings from user/item ID to index positions
user_to_index = pd.Series(data=index_to_user.index, index=index_to_user.values)
item_to_index = pd.Series(data=index_to_item.index, index=index_to_item.values)

# convert user/item identifiers to index positions
interactions_train_imp = interactions_train.copy()
interactions_train_imp['user_id'] = interactions_train['user_id'].map(user_to_index)
interactions_train_imp['product_id'] = interactions_train['product_id'].map(item_to_index)

# prepare the data for CSR creation
data = sample_weight_train
rows = interactions_train_imp['user_id']
cols = interactions_train_imp['product_id']

# create the required user-item and item-user CSR matrices
user_items_imp = csr_matrix((data, (rows, cols)), shape=(n_users_train, n_items_train))
item_users_imp = user_items_imp.T.tocsr()

Теперь, когда у нас есть данные в требуемом формате ввода, мы можем подогнать модель ALS и сгенерировать рекомендации TopN для пользователей проверки. Мы будем использовать то же измерение скрытого фактора, что и RankFM, а в остальном будем использовать значения по умолчанию.

from implicit.als import AlternatingLeastSquares

```

```

# initialize and fit the model
imp_model = AlternatingLeastSquares(factors=50)
imp_model.fit(item_users_imp)

# generate recommendations for all users and map back to original user/item ID values
recs_imp = imp_model.recommend_all(user_items=user_items_imp, N=10, filter_already_liked_it=True)
recs_imp = pd.DataFrame(recs_imp, index=index_to_user.values).apply(lambda c: c.map(index_to_user))

```

Наконец, мы рассчитаем hit rate, precision и recall для того же набора удерживающих взаимодействий, который использовался для оценки нашей модели RankFM.

```

valid_user_items = interactions_valid.groupby('user_id')['product_id'].apply(set).to_dict()
combined_users = set(train_users) & set(valid_users)

imp_hrt = np.mean([int(len(set(recs_imp.loc[u])) & valid_user_items[u]) > 0] for u in combined_users)
imp_pre = np.mean([len(set(recs_imp.loc[u])) & valid_user_items[u]) / len(recs_imp.loc[u]) for u in combined_users])
imp_rec = np.mean([len(set(recs_imp.loc[u])) & valid_user_items[u]) / len(valid_user_items[u]) for u in combined_users])

hit_rate: 0.780
precision: 0.240
recall: 0.125

```

Хотя полный анализ должен был бы соответствующим образом настроить обе модели для различных комбинаций гиперпараметров, мы можем увидеть скромный прирост производительности (5–10%) с формулировкой модели FM и методами оптимизации LTR с использованием идентичных данных взаимодействия и примерно эквивалентных спецификаций модели (т. е. количество скрытые факторы). Вероятно, мы бы увидели еще больший прирост производительности по сравнению с базовой моделью ALS MF, если бы у пользователей было меньше зарегистрированных взаимодействий (в данных Instacart у каждого пользователя есть как минимум 3 заказа) и/или у нас были бы более информативные вспомогательные функции пользователя/элемента.

## 31.8. Заключение

В этой статье мы рассмотрели машины факторизации (FM) как универсальную, но мощную модельную структуру, особенно хорошо подходящую для решения проблем рекомендаций по совместной фильтрации. В отличие от традиционных подходов матричной факторизации (MF), модели FM могут быть естественным образом расширены, чтобы включать пользователя, элемент или контекстуальные вспомогательные

функции для увеличения основных данных взаимодействия. Затем мы показали, как функции потерь на основе обучения к рангу (LTR), такие как байесовское персонализированное ранжирование (BPR) и взвешенный приблизительный парный ранг (WARP), являются ключом к успешной адаптации моделей FM к неявным данным обратной связи. Чтобы продемонстрировать эти моменты, мы продемонстрировали модель FM с неявной обратной связью, которая превосходит популярный базовый алгоритм ALS MF на хорошо известном наборе данных с рекомендациями по неявной обратной связи с открытым исходным кодом. Наконец, мы представили RankFM: новый пакет Python для построения и оценки моделей FM для задач рекомендаций с неявными данными обратной связи.

## 32. Лекция 14. Факторизационные машины. Наша лекция

Когда мы говорили про колаборативную фильтрацию и SVD - мы работали с матрицей рейтингов. В таких моделях мы больше ничего добавить не можем. Но вообще есть еще признаки товара и пользователя, которые можно использовать. Поэтому можно перейти к классической формулировке от матрицы рейтингов к матрице объект-признак. Что у нас будет один объект?

### 32.1. Контентные модели

Категориальный признак юзера, категориальный признак объекта, признаки пользователя, признаки товара.

Получается, что одна строчка - характеристика пары  $User - Item$ . Предсказывать мы будем рейтинг.

На такой постановке задачи мы можем обучать классические модели ML/DL. Это называется контентные модели, когда помимо самого взаимодействия пользователя, мы учитываем информацию про пользователя и про товар.

### 32.2. Факторизационные машины

Давайте мы будем делать предсказание не как линейную регрессию, а как квадратичную регрессию. То есть

$$\alpha(x) = \sum_i w_i x_i + \sum_{i,j} w_{ij} x_i x_j$$

Но тут есть несколько проблем:

- Матрица получается разреженая
- Признаков очень много, весов  $w_{ij}$  будет квадратично от признаков. Для обучения такой модели нужно много времени и ресурсов.

Мы можем представить матрицу  $W$  как произведение матриц меньшего размера:  $w_{ij} = \langle v_i, v_j \rangle >$ . Сколько мы будем экономить?

Пусть у нас  $l = 100$  признаков, тогда у нас  $|\{w_{ij}\}| = \frac{10^4}{2}$ . А размерности векторов  $v_i, v_j$  мы можем задать сами. Допустим у нас длина одного вектора 5, тогда координат векторов  $v_i$  получается 500. Это меньше в 10 раз! Из 5000 неизвестных получили 500.

Это и есть факторизационная машина.

### 32.3. Метрики

Если мы говорим про метрики рекомендации, то на них можно смотреть по разному:

- Как классификацию: купит пользователь товар или нет. Тогда можем использовать метрики классификации
- Как задачу регрессии: можем предсказывать непрерывный рейтинг и использовать обычные метрики регрессии

Но также мы можем также использовать и метрики ранжирования: обычно на основе наших моделей мы рекомендуем пользователю не один товар, а несколько, и надеемся, что какой-то товар пользователю понравится. Мы как-то наши рекомендации ранжируем и каждой рекомендации даём вес, ранжируем по уменьшению предсказанного веса и рекомендуем товары с наибольшим возможным рейтингом/весом. Здесь и нужны метрики качества ранжирования. Это можно делать по разному

#### 32.3.1. Precision at K

Precision at K (p@K) - точность на  $K$  элементах - базовая метрика ранжирования для одного объекта. Допустим, наш алгоритм ранжирования выдал оценки релевантности для каждого элемента  $\{r(e)\}_{e \in E}$ . Отобрав среди них первые  $K \leq M$  элементов с наибольшим  $r(e)$ , можно посчитать долю релевантных. Именно это и делает precision at K:

$$p@K = \frac{\sum_{k=1}^K r^{true}(\pi^{-1}(k))}{K} = \frac{\text{релевантных элементов}}{K}$$

где  $\pi^{-1}(k)$  - элемент  $e \in E$ , который в результате перестановки  $\pi$  оказался на  $k$ -й позиции. Так  $\pi^{-1}(1)$  - элемент с наибольшим  $r(e)$ , итд.

Эта метрика плоха вот чем: представим два разных пользователя. Каждому мы предложили 10 товаров. Первый пользователь пролистал все товары и только последний купил. Второй пользователь купил тоже один, но самый первый. С точки зрения метрики эти ситуации одинаковые, но вторая лучше.

### 32.3.2. Average precision at K

Эта метрика как раз закрывает минус предыдущей. Она учитывает расположение выдачи.

Эта метрика равна сумме  $p@k$  по индексам  $k$  от 1 до  $K$  только для релевантных элементов, делённому на  $K$ :

$$@K = \frac{1}{K} \sum_{k=1}^K r^{true}(\pi^{-1}(k)) \cdot p@k$$

Представьте, что предлагается только три продукта. Если пользователь купил только третий, то метрика будет  $ap@3 = \frac{1}{3}(0+0+\frac{1}{3})$ , а если пользователь купит первый товар из списка, то  $ap@3 = \frac{1}{3}(\frac{1}{1}+\frac{1}{2}+0)$ , что несомненно лучше. Если бы были куплены все товары, то  $ap@3 = \frac{1}{3}(\frac{1}{1}+\frac{2}{2}+\frac{3}{3}) = 1$

### 32.3.3. Mean average precision at K

Mean average precision at K ( $map@K$ ) - одна из наиболее часто используемых метрик качества ранжирования. В  $p@K$  и  $ap@K$  качество ранжирования оценивается для отдельно взятого объекта (пользователя, поискового запроса). На практике объектов множество: мы имеем дело с сотнями тысяч пользователей, миллионами поисковых запросов и т.д. Идея  $map@K$  заключается в том, чтобы посчитать  $ap@K$  для каждого объекта и усреднить:

$$map@K = \frac{1}{N} \sum_{j=1}^N ap@K_j$$

*Замечание:* идея вполне логична, если предположить, что все пользователи одинаково нужны и одинаково важны. Если же это не так, то вместо простого усреднения можно использовать взвешенное, домножив  $ap@K$  каждого объекта на соответствующий его "важности" вес.

### 32.3.4. Cumulative Gain at K

Мы будем штрафовать объект за его позицию в поисковой выдаче, и у каждого объекта есть нелинейный вес. Вес объекта будет уменьшаться с его позицией в поисковой выдаче:

$$DCG@K = \sum_{k=1}^K \frac{2^{r^{true}(\pi^{-1}(k))} - 1}{\log_2(k+1)}$$

*Замечание:* если  $r^{true}$  принимает значения только 0 и 1, то  $2^{r^{true}(\pi^{-1}(k))} = r^{true}(\pi^{-1}(k))$ , и формула принимает более простой вид:

$$DCG@K = \sum_{k=1}^K \frac{r^{true}(\pi^{-1}(k)) - 1}{\log_2(k+1)}$$

Если у нас три товара, и пользователь купил только первый, то  $DCG@3 = \frac{1}{\log_2(2)} + 0 + 0 = 1$ . Если пользователь купил только третий товар, то  $DCG@3 = 0 + 0 + \frac{1}{\log_2(4)} = \frac{1}{2}$ .

Использование логарифма как функции дисконтирования можно объяснить следующими интуитивными соображениями: с точки зрения ранжирования позиции в начале списка отличаются гораздо сильнее, чем позиции в его конце. Так, в случае поискового движка между позициями 1 и 11 целая пропасть (лишь в нескольких случаях из ста пользователь заходит дальше первой страницы поисковой выдачи), а между позициями 101 и 111 особой разницы нет - до них мало кто доходит. Эти субъективные соображения прекрасно выражаются с помощью логарифма.

Discounted cumulative gain решает проблему учёта позиции релевантных элементов, но лишь усугубляет проблему с отсутствием нормировки: если  $CG@K$  варьируется в пределах  $[0; K]$ , то  $DCG@K$  уже принимает значения на не совсем понятном отрезке.

## 32.4. Практика

### 32.4.1. Загрузка библиотек

```
%pylab inline

import numpy as np
import pandas as pd

from tqdm import tqdm_notebook
```

### 32.4.2. Загрузка и обработка данных

Если что-то непонятно - пошагово описывалось в предыдущей лекции

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('User_uploaded_file_{name}_with_length_{length}_bytes'.format(
        name=fn, length=len(uploaded[fn])))

!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json

! kaggle competitions list
! kaggle datasets download -d gspmoreira/articles-sharing-reading-from-cit-deskdrop
! unzip articles-sharing-reading-from-cit-deskdrop.zip

articles_df = pd.read_csv('shared_articles.csv')
articles_df = articles_df[articles_df['eventType'] == 'CONTENT_SHARED']
articles_df.head(2)

interactions_df = pd.read_csv('users_interactions.csv')
interactions_df.head(10)

interactions_df.personId = interactions_df.personId.astype(str)
interactions_df.contentId = interactions_df.contentId.astype(str)
articles_df.contentId = articles_df.contentId.astype(str)

event_type_strength = {
    'VIEW': 1.0,
    'LIKE': 2.0,
    'BOOKMARK': 2.5,
    'FOLLOW': 3.0,
    'COMMENT CREATED': 4.0,
}

interactions_df['eventStrength'] = interactions_df.eventType.apply(lambda x: event_type_strength[x])
interactions_df['eventStrength']

users_interactions_count_df = (
    interactions_df
    .groupby(['personId', 'contentId'])
    .first()
    .reset_index()
    .groupby('personId').size()
)
print('#users:', len(users_interactions_count_df))

users_with_enough_interactions_df = \
    users_interactions_count_df[users_interactions_count_df >= 5].reset_index()[['personId']]
```

```

print('# users with at least 5 interactions:', len(users_with_enough_interactions_df))

users_interactions_count_df.hist(bins=30)

interactions_from_selected_users_df = interactions_df.loc[np.in1d(interactions_df.personId,
    users_with_enough_interactions_df)]

print('# interactions before:', interactions_df.shape)
print('# interactions after:', interactions_from_selected_users_df.shape)

def smooth_user_preference(x):
    return math.log(1+x, 2)

interactions_full_df = (
    interactions_from_selected_users_df
    .groupby(['personId', 'contentId']).eventStrength.sum()
    .apply(smooth_user_preference)
    .reset_index().set_index(['personId', 'contentId'])
)
interactions_full_df['last_timestamp'] = (
    interactions_from_selected_users_df
    .groupby(['personId', 'contentId'])['timestamp'].last()
)
interactions_full_df = interactions_full_df.reset_index()
interactions_full_df.head(20)

from sklearn.model_selection import train_test_split

split_ts = 1475519530
interactions_train_df = interactions_full_df.loc[interactions_full_df.last_timestamp < split_ts]
interactions_test_df = interactions_full_df.loc[interactions_full_df.last_timestamp >= split_ts]

print('# interactions on Train set: %d' % len(interactions_train_df))
print('# interactions on Test set: %d' % len(interactions_test_df))

interactions_train_df

interactions = (
    interactions_train_df
    .groupby('personId')['contentId'].agg(lambda x: list(x))
    .reset_index()
    .rename(columns={'contentId': 'true_train'})
    .set_index('personId')
)
interactions['true_test'] = (
    interactions_test_df
    .groupby('personId')['contentId'].agg(lambda x: list(x))
)
# fill nans with empty lists
interactions.loc[pd.isnull(interactions.true_test), 'true_test'] = [
    []] for x in range(len(interactions.loc[pd.isnull(interactions.true_test), 'true_test']))

interactions.head(5)

def calc_precision(column):
    ##### There your code #####
    return (
        interactions

```

```

    .apply(
        lambda row:
            len(set(row['true_test'])) . intersection(
                set(row[column]))) /
            min(len(row['true_test']) + 0.001, 10.0),
            axis=1).mean()

#####
ratings = pd.pivot_table(
    interactions_train_df,
    values='eventStrength',
    index='personId',
    columns='contentId').fillna(0)

```

### 32.4.3. Контентные модели

В этой части реализуем альтернативный подход к рекомендательным системам — контентные модели.

Теперь мы будем оперировать не матрицей с оценками, а классической для машинного обучения матрицей объекты-признаки. Каждый объект будет характеризовать пару user-item и содержать признаки, описывающие как пользователя, так и товар. Кроме этого признаки могут описывать и саму пару целиком.

Матрица со всеми взаимодействиями уже получена нами на этапе разбиения выборки на 2 части.

Будем обучать классификатор на взаимодействие, а для него нужны отрицательные примеры. Добавим случайные отсутствующие взаимодействия как отрицательные.

Заметим, что модель оценивает каждую пару потенциального взаимодействия, а значит, надо подготовить выборку из всех возможных пар из пользователей и статей.

```

test_personId = np.repeat(interactions.index, len(ratings.columns))
test_contentId = list(ratings.columns) * len(interactions)
test = pd.DataFrame(
    np.array([test_personId, test_contentId]).T,
    columns=['personId', 'contentId'])

interactions_train_df = pd.concat((
    interactions_train_df,
    test.loc[
        np.random.permutation(test.index)[
            :4*len(interactions_train_df)]], ignore_index=True)
interactions_train_df.eventStrength.fillna(0, inplace=True)

```

Придумаем и добавим признаков о пользователях и статьях. Сначала добавим информацию о статьях в данные о взаимодействиях.

```

interactions_train_df = interactions_train_df.merge(articles_df, how='left', on='contentId')
interactions_test_df = interactions_test_df.merge(articles_df, how='left', on='contentId')

# first feature index
features_start = len(interactions_train_df.columns)

```

После обучения модели нам придётся делать предсказания на тестовой выборке для всех возможных пар статья-пользователь. Подготовим такую матрицу, чтобы параллельно посчитать признаки для неё.

```

test_personId = np.repeat(interactions.index, len(articles_df))
test_contentId = list(articles_df.contentId) * len(interactions)
test = pd.DataFrame(
    np.array([test_personId, test_contentId]).T,
    columns=['personId', 'contentId'])
test = test.merge(articles_df, how='left', on='contentId')

test.head()

```

Добавим признаки-индикаторы возможных значений contentType.

```

interactions_train_df[ 'is_HTML' ] = interactions_train_df.contentType == 'HTML'
interactions_train_df[ 'is_RICH' ] = interactions_train_df.contentType == 'RICH'
interactions_train_df[ 'is_VIDEO' ] = interactions_train_df.contentType == 'VIDEO'

test[ 'is_HTML' ] = test.contentType == 'HTML'
test[ 'is_RICH' ] = test.contentType == 'RICH'
test[ 'is_VIDEO' ] = test.contentType == 'VIDEO'

Добавим признаки "длина названия" и "длина текста" + некоторые проверки на ключевые слова.

interactions_train_df[ 'title_length' ] = interactions_train_df.title.fillna( '' ).apply( len )
interactions_train_df[ 'text_length' ] = interactions_train_df.text.fillna( '' ).apply( len )

test[ 'title_length' ] = test.title.fillna( '' ).apply( len )
test[ 'text_length' ] = test.text.fillna( '' ).apply( len )

interactions_train_df[ 'has_new' ] = \
    interactions_train_df.title.fillna( '' ).apply( lambda x: 'new' in x.lower() )
interactions_train_df[ 'has_why' ] = \
    interactions_train_df.title.fillna( '' ).apply( lambda x: 'why' in x.lower() )
interactions_train_df[ 'has_how' ] = \
    interactions_train_df.title.fillna( '' ).apply( lambda x: 'how' in x.lower() )
interactions_train_df[ 'has_ai' ] = \
    interactions_train_df.title.fillna( '' ).apply( lambda x: 'ai' in x.lower() )

test[ 'has_new' ] = \
    test.title.fillna( '' ).apply( lambda x: 'new' in x.lower() )
test[ 'has_why' ] = \
    test.title.fillna( '' ).apply( lambda x: 'why' in x.lower() )
test[ 'has_how' ] = \
    test.title.fillna( '' ).apply( lambda x: 'how' in x.lower() )
test[ 'has_ai' ] = \
    test.title.fillna( '' ).apply( lambda x: 'ai' in x.lower() )

```

Добавим признаки-индикаторы языка.

```

interactions_train_df[ 'is_lang_en' ] = interactions_train_df.lang == 'en'
interactions_train_df[ 'is_lang_pt' ] = interactions_train_df.lang == 'pt'

test[ 'is_lang_en' ] = test.lang == 'en'
test[ 'is_lang_pt' ] = test.lang == 'pt'

```

Обучим на полученных признаках градиентный бустинг.

```

import catboost

model = catboost.CatBoostClassifier()
model.fit( interactions_train_df[ interactions_train_df.columns[ features_start : ] ] ,
           np.array( interactions_train_df.eventStrength > 0, dtype=int ) )

```

Сделаем предсказания на тестовой выборке, сформируем из них рекомендации.

```

top_k = 10

predictions = model.predict_proba( test[ interactions_train_df.columns[ features_start : ] ] )[ :, 1 ]
test[ 'predictions' ] = predictions

test = test.sort_values( 'predictions' , ascending=False )
predictions = test.groupby( 'personId' )[ 'contentId' ].aggregate( list )
tmp_predictions = []

for personId in tqdm_notebook( interactions.index ):
    prediction = np.array( predictions.loc[ personId ] )

    tmp_predictions.append(

```

```

list(prediction[~np.in1d(
    prediction,
    interactions.loc[personId, 'true_train'])])[:top_k])

interactions['prediction_content'] = tmp_predictions

```

Делаем предсказание

```
calc_precision('prediction_content')
```

### 32.4.4. Факторизационная машина

Вспомним, что факторизационная машина учитывает попарные взаимодействия признаков, что приводит сразу и к использованию контента (сами признаки), и к обучению скрытых представлений (индикаторы пользователей и статей).

Попробуем факторизационные машины из библиотеки pyFM (так как можно работать прямо из питона). <https://github.com/coreylynch/pyFM>

```
from pyfm import pylibfm
from sklearn.feature_extraction import DictVectorizer
```

Перейдём к обобщению матричных разложений — факторизационным машинам, которые могут работать с контентной информацией. Вспомним, какие данные у нас изначально были:

В факторизационную машину можно загрузить "айдишикни" пользователей и статей (то есть сделать аналог коллаборативной фильтрации) и одновременно различные признаки.

Удобно обрабатывать категориальные переменные (id и другие) можно с помощью DictVectorizer. Например, процесс может выглядеть вот так:

```

train = [
    {"user": "1", "item": "5", "age": 19},
    {"user": "2", "item": "43", "age": 33},
    {"user": "3", "item": "20", "age": 55},
    {"user": "4", "item": "10", "age": 20},
]
v = DictVectorizer()
X = v.fit_transform(train)
y = np.repeat(1.0, X.shape[0])
fm = pylibfm.FM()
fm.fit(X, y)
fm.predict(v.transform({"user": "1", "item": "10", "age": 24}))

```

Сгенерируем таблицу с признаками в таком виде, где будут id пользователя, статьи и автора статьи и несколько признаков, которые вы сможете придумать.

```

train_data = []

for i in tqdm_notebook(range(len(interactions_train_df))):
    features = {}
    features['personId'] = str(interactions_train_df.iloc[i].personId)
    features['contentId'] = str(interactions_train_df.iloc[i].contentId)

    try:
        article = articles_df.loc[features['contentId']]
        features['authorId'] = str(article.authorPersonId)
        features['authorCountry'] = str(article.authorCountry)
        features['lang'] = str(article.lang)
    except:
        features['authorId'] = 'unknown'
        features['authorCountry'] = 'unknown'
        features['lang'] = 'unknown'

    train_data.append(features)

```

Повторим эту процедуру для тестовой выборки. Заметим, что модель оценивает каждую пару потенциального взаимодействия, а значит, надо подготовить выборку из всех возможных пар из пользователей и статей.

```

from copy import deepcopy

test_data = []

for i in tqdm_notebook(range(len(interactions))):
    features = {}
    features[ 'personId' ] = str( interactions.index[ i ] )
    for j in range(len(ratings.columns)):

        features[ 'contentId' ] = str( ratings.columns[ j ] )

    try:
        article = articles_df.loc[ features[ 'contentId' ] ]
        features[ 'authorId' ] = str( article.authorPersonId )
        features[ 'authorCountry' ] = str( article.authorCountry )
        features[ 'lang' ] = str( article.lang )
    except:
        features[ 'authorId' ] = 'unknown'
        features[ 'authorCountry' ] = 'unknown',
        features[ 'lang' ] = 'unknown'

    test_data.append(deepcopy(features))

```

Векторизуем, получим разреженные матрицы.

Мы будем обучать регрессор на силу взаимодействия, а для него нужны отрицательные примеры. Добавим некоторое количество случайных примеров как негативные (матрица взаимодействий разреженная, поэтому шансы взять как негативное взаимодействие некоторое положительное мало).

```
dv = DictVectorizer()
```

```

train_features = dv.fit_transform(
    train_data + list(np.random.permutation(test_data)[:100000]))
test_features = dv.transform(test_data)

```

```
train_features
```

```
y_train = list(interactions_train_df.eventStrength.values) + list(np.zeros(100000))
```

Укажем размером скрытого представления 10, сделаем 10 итераций.

```
fm = pylibfm.FM(num_factors=10, num_iter=30, task='regression')
```

```
fm.fit(train_features, y_train)
```

Предскажем и оценим качество.

```
test_features = dv.transform(test_data)
```

```
y_predict = fm.predict(test_features)
```

```
new_ratings = y_predict.reshape((1112, 2366))
```

```
predictions = []
```

```

for i, person in enumerate(interactions.index):
    user_prediction = ratings.columns[np.argsort(new_ratings[ i ])[:-1]]
    predictions.append(
        user_prediction[~np.in1d(user_prediction,
                                interactions.loc[ person, 'true_train' ])][:top_k])

```

```
interactions[ 'fm_prediction' ] = predictions
```

```
calc_precision('fm_prediction')
```

## 33. Статья. Дисбаланс Классов

### 33.1. Что понимается под дисбалансом классов?

Рассмотрим ситуацию несбалансированных классов – что нужно уточнить при выработке стратегии решения задачи классификации, какие стратегии бывают, как отвечать на вопрос про дисбаланс на собеседовании. Приведём результаты экспериментов, дадим код и практические советы. Уровень для читателя — средний (достаточно знать основы машинного обучения и иметь небольшой опыт в решении задач классификации).

В задаче классификации данные называются **несбалансированными** (**Imbalanced Data**), если в обучающей выборке доли объектов разных классов существенно различаются, также говорят, что «**классы не сбалансираны**». Есть понятие несбалансированности и для задач регрессии, но там оно ограничивается наличием аномалий в данных, поэтому здесь мы будем рассматривать только задачи классификации и для простоты – бинарной классификации. На рис. 1 (слева) схематически показан дисбаланс: розовым цветом – объекты «большого» класса 0 и синим – объекты «малого» класса 1. Не надо путать дисбаланс с разреженностью, на рис. 1 (справа) показаны данные в виде матрицы (например «user-item»), лишь для небольшого числа элементов матрицы известна метка, поэтому говорят о разреженности данных, но пропорции классов примерно равны.

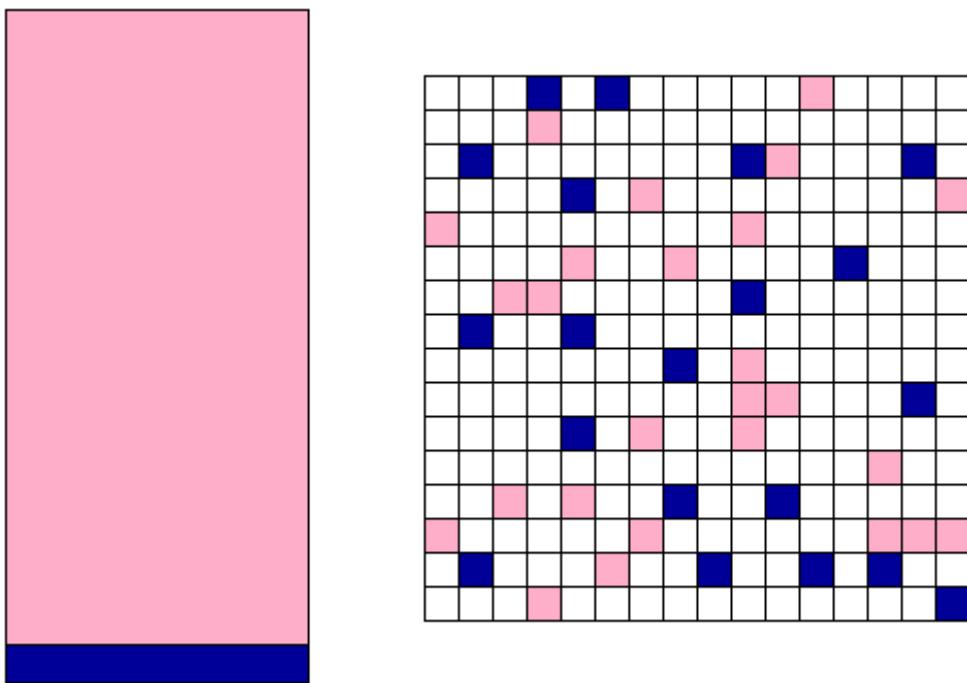


Рис. 1. Несбалансированная задача классификации (слева) и разреженная сбалансированная задача (справа).

Задачи с дисбалансом чаще всего возникают, когда какой-то из классов соответствует очень редко наблюдаемым или диагностируемым явлениям (дефолт, поломка, редкая болезнь, мошенничество и т.п.)

### 33.2. Что делать при дисбалансе классов?

Такой вопрос часто задают на собеседованиях, есть блог-заметки и ютуб-ролики на эту тему, почти все они дают ложное представление о дисбалансе. Обычно рекомендуют давать такой ответ – надо сделать **перебалансировку данных**: недо- или пере- сэмплирование (определим дальше), иногда вспоминают аббревиатуру SMOTE. Сейчас поговорим о том, что подобный ответ не учитывает теорию и практику классификации.

**Во-первых, при ответе на вопрос надо обязательно уточнить природу задачи:** в чём причина дисбаланса, сколько классов, насколько серьёзный дисбаланс, какими данными и мета-данными мы располагаем. Например, очень часто к дисбалансу приводит наличие дубликатов, первое напрашивавшееся действие – устраниТЬ дубликаты. Также часто дисбаланс возникает в задаче с очень большим числом классов и некоторые классы малы по естественным причинам, например это генотипы представителей малых народов. Удивительно, но в этом случае бывает не так важно, относит ли алгоритм какие-то объекты к малым классам. Если позитивных (класса 1) объектов в обучении крайне мало, например 1-3 на 1 миллион объектов (а такие задачи бывают, например, когда позитивный класс – катастрофы или

большие экономические кризисы), то задачу логично решать как детектирование аномалий (здесь мы не будем подробно описывать пайплайн решения). Дисбаланс может меняться со временем, например пропорции классов могут сильно отличаться в обучении и контроле – это отдельная ситуация и здесь логично «выравнивать» пропорции классов в обучении и контроле (чтобы обучение было похоже на контроль). Дисбаланс может быть из-за недостатка размеченных данных (например, порция данных с объектами класса 1 не была полностью размечена), тут есть варианты использования, например, синтетических данных. Наконец, всего перечисленного может не быть, у нас обычная бинарная задача классификации, пропорции классов не меняются со временем и процент объектов класса 1 – от 2% до 10%. Такую ситуацию и рассмотрим дальше (она соответствует, например, задаче банковского скоринга в стабильной экономической обстановке).

**Во-вторых, надо уточнить функцию ошибки (функционал качества).** Почему это важно? Если используется LogLoss, то каких-то «танцев с перебалансировкой» делать не только не нужно, но и недопустимо, поскольку это делает решение (алгоритм) неоткалиброванным. Если используется ROC-AUC, то перебалансировка и многие другие рецепты не влияют на значение функционала (будет изменение в третьем знаке после запятой). Если используется F1-мера, то тут уже интереснее – подобные функции ниже и рассмотрим.

Но прежде чем говорить о настройке на F1-меру (и другие похожие на неё функционалы), давайте осознаем, а что означает эта настройка. Допустим у нас две корзины, в каждой 10 шаров, в первой – 1 чёрный, во второй тоже 1 чёрный, все остальные шары белые, см. рис. 2. Сейчас мы вытащим шар из корзины (это будет случайная корзина, они равновероятны, но мы будем знать из какой корзины берётся шар), надо угадать его цвет. Рассмотрим прогноз «шар белый», его точность (accuracy)

$$0.5 \times 0.9 + 0.5 \times 0.9 = 0.9$$

его F1-мера равна нулю. Рассмотрим прогноз «шар из первой корзины белый, а из второй – чёрный», его точность (accuracy)

$$0.50.9 + 0.5 \times 0.1 = 0.5$$

его F1-мера равна  $\frac{2}{\frac{1}{0.1} + \frac{1}{0.5}} = \frac{1}{6}$ . Мы увеличили F1-меру, но что означает это увеличение? Стал ли наш прогноз более адекватным и вообще, научились ли мы угадывать цвет? Очевидно, нет! Кстати, противоположный прогноз «шар из первой корзины чёрный, а из второй – белый» обладает такими же показателями качества. Дальше мы не будем расписывать, что из этого следует, наиболее сообразительные догадаются сами;) Но самое главное – **F1-мера может расти не из-за того, что прогноз становится «адекватнее»**, а как раз из-за того, что он становится нетривиальным (как и многие другие показатели качества).

В-третьих, нужно уточнить контекст вопроса. Есть выражение «проблема дисбаланса», но на самом деле, **никакой проблемы дисбаланса нет!** Дисбаланс – это естественное свойство данных. Когда говорят, что в этом случае использование точности неправильно, т.к. формально высокая точность получается у константного решения (см. пример с шарами), то это проблема выбора функционала качества при дисбалансе, а не самого дисбаланса. Когда говорят, что при настройке моделей они превращаются в константные, то это проблема настройки, выбора loss-функции и порога бинаризации. Если ошибки первого и второго рода имеют разную цену (обычно ошибочная классификация объектов малого класса стоит дороже), то это надо напрямую учитывать при обучении алгоритмов (с помощью весов классов, об этом ниже). Также **возможно в вопросе про дисбаланс имеется в виду изменение пайплайна решения задачи, например использование стратифицированного контроля** (его обязательно надо использовать, хотя в большинстве современных библиотек машинного обучения он включён по умолчанию).

Наконец, самое любопытное (мало кто об этом задумывается), чтобы понять, как учитывать дисбаланс необходимо понять, насколько хорошо наши модели в данной задаче могут быть откалиброваны, какая геометрия данных и распределения классов! Ниже на модельных задачах объясним, что имеется в виду.

### 33.3. Перебалансировка данных/изменение выборки

На рис. 2 показана идея перебалансировки: мы делаем классы сбалансированными, для этого заменяем большой класс подвыборкой по мощности равной малому классу – это называется **недосэмплированием (Undersampling the majority class)** или «увеличиваем в размерах малый класс» – это называется **пересемплированием (Oversampling the minority class)**. Простейшая стратегия недосэмплирования – взять случайную подвыборку, простейшая стратегия пересемплирования – продублировать объекты малого класса. О более «умных» поговорим дальше. У пересемплирования качество, как правило выше, т.к. мы используем все данные, однако недосэмплирование позволяет учить модель на маленькой выборке (можно кстати, строить ансамбль над алгоритмами, обученными на разных недосэмплированиях).

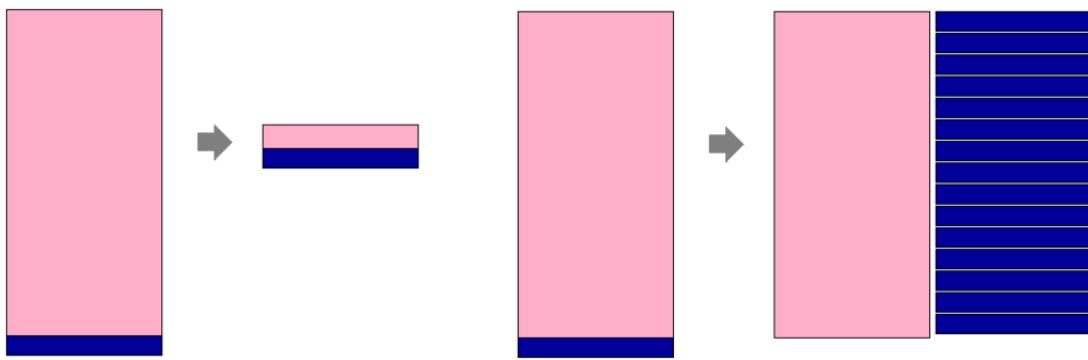


Рис. 2. Undersampling (слева) и Oversampling (справа).

### 33.4. Nearmiss1/2, Tomek links, Edited nearest neighbors (ENN)

Основная идея «умного недосэмплирования» – брать из большого класса только объекты важные для решения рассматриваемой задачи.

- Стратегия метода NearMiss-1 – из большего класса выбираем объекты, у которых среднее расстояний до N ближайших малого класса наименьшее, т.е. из граничной зоны (первая картинка рис. 3.1).
- Стратегия метода NearMiss-2 – из большего класса выбираем объекты, у которых среднее расстояний до N дальних малого класса наименьшее (вторая картинка рис. 3.1).
- Стратегия метода Tomek links – удалить объекты большого класса, образующие связи Томека (объекты двух разных классов образуют связь Томека, если нет объекта, который ближе к одному из них при этом являясь объектом другого класса, см. рис. 3.1).
- Стратегия ENN – сделать скользящий контроль, например по 10 фолдам, удалить объекты большого класса, на которых ближайший сосед ошибается. В последних двух методах не гарантируется выравнивание классов по мощности. Заметим, что последний метод может быть использован вместе с какой-то моделью алгоритмов (и «отвязан» от конкретной метрики), а предыдущие методы ориентировались на метрику. Есть и другие стратегии недосэмплирования, но общая идея должна быть понятна.

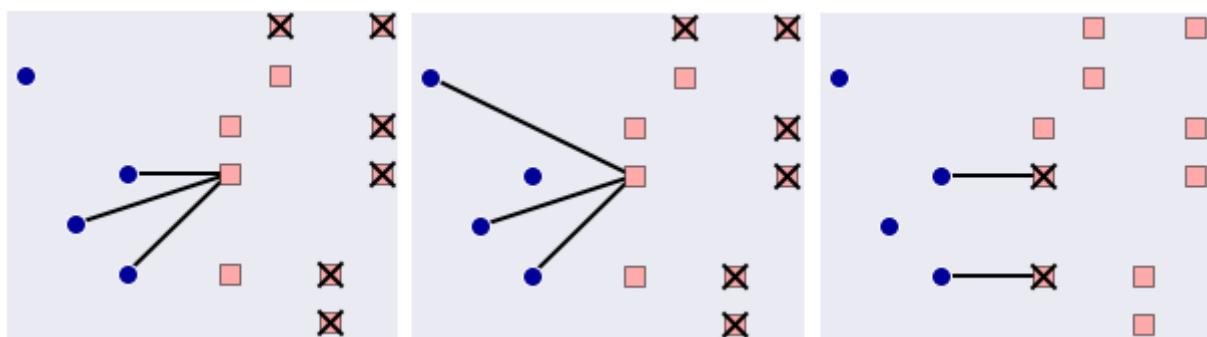


Рис. 3.1. Слева-направо: ближайшие соседи из малого класса, дальние соседи из малого класса, связи Томека.

На рис. 3.2 показано применение различных стратегий недосэмплирования от случайной до ENN, в подписи они названы по именам соответствующих функций из библиотеки imblearn. Исходные данные изображены на первой картинке рис. 4.2.

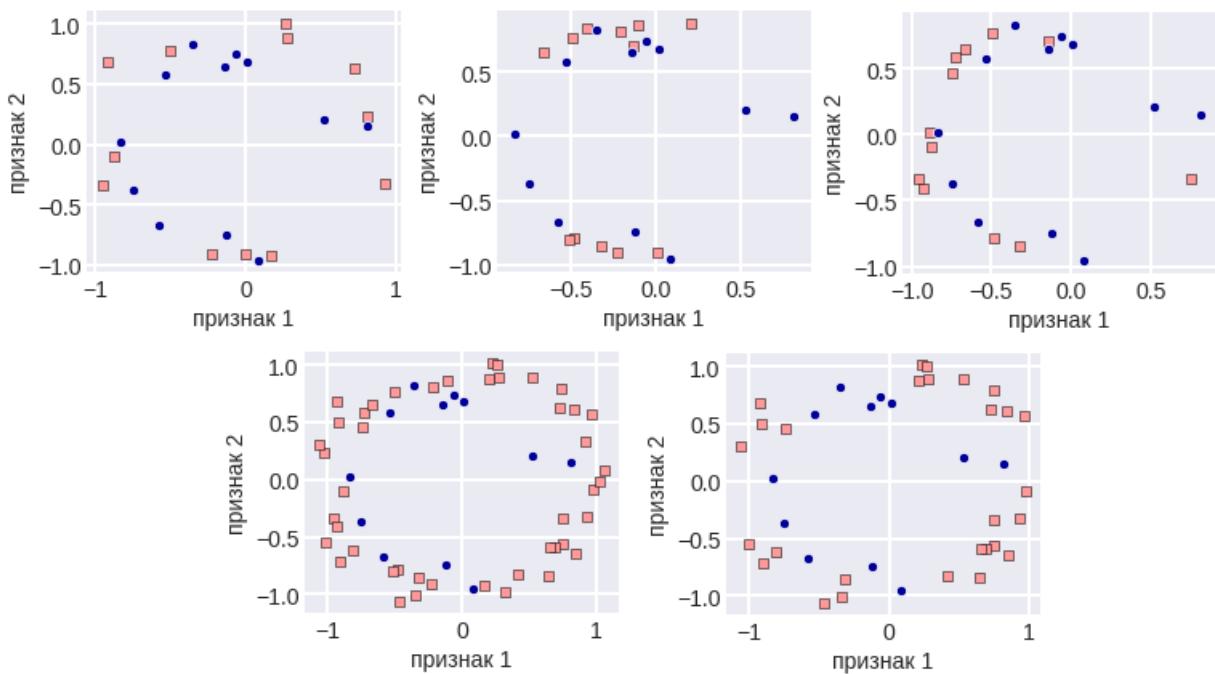


Рис. 3.2. Иллюстрация стратегий недосэмплирования (слева направо): RandomUnderSampler, NearMiss(version=1), NearMiss(version=2), TomekLinks(), EditedNearestNeighbours().

### 33.5. SMOTE = Synthetic Minority Oversampling Techniques, ADASYN = Adaptive Synthetic

Идея метода SMOTE: увеличить малый класс за счёт представителей выпуклых комбинаций пар (см. рис. 4.1). Подобная идея реализуется в современных методах аугментации, например в MixUp. В методе SMOTE для точки малого класса выбирается один из  $k$  ближайших соседей и на отрезке между ними случайно выбирается новый объект. Почему-то нигде в описаниях не указывается, но в качестве  $k$  ближайших соседей рассматриваются только объекты малого класса. Есть разные модификации метода, например такие, в которых все точки делятся на группы, в зависимости от процента «чужих» в окрестности (от этого зависит вероятность порождения нового объекта).

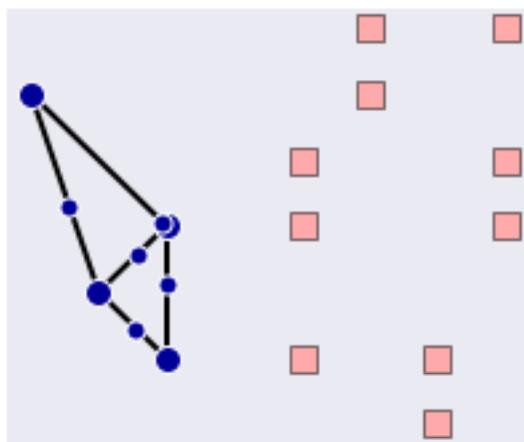


Рис. 4.1. Идея метода SMOTE.

Метод ADASYN аналогичен SMOTE, но число объектов, которые генерируются с помощью объекта малого класса, пропорционально числу чужаков (объектов большого класса) в его окрестности.

На рис. 4.2 показано применение различных стратегий пересэмплирования. Первая картинка визуально совпадает с изображением исходного датасета, т.к. здесь представители малого класса просто дублируются.

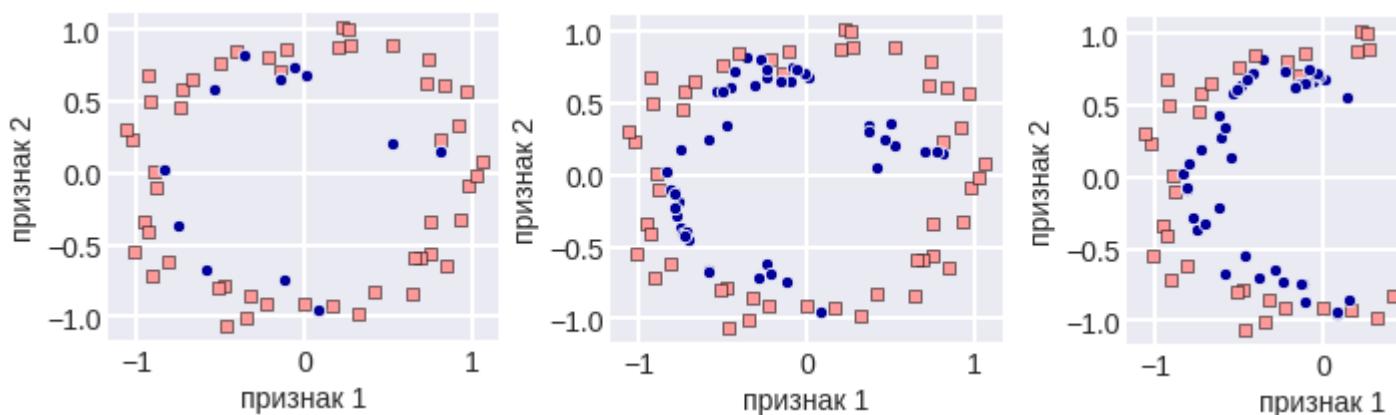


Рис. 4.2. Иллюстрация методов пересэмплирования, слева-направо: RandomOverSampler, SMOTE, ADASYN.

### 33.6. Взвешивание объектов

Большинство методов реализованных в sklearn имеет параметр «веса объектов» или «веса классов», обычно берут веса объектов большого класса  $a$  равным 1, а веса объектов малого класса  $b = \frac{m_0}{m_1}$ , где  $m_0, m_1$  – число объектов в классах  $a$  и  $b$  соответственно. Обычно при настройке алгоритма классификации минимизируют эмпирический риск

$$\sum_i L(y_i, \alpha(x_i))$$

в котором почти все слагаемые относятся к большому классу, в весовых схемах добавляются веса штрафов на конкретных объектах. Если объекты одного класса имеют одинаковый вес, получаем функцию, которую логично минимизировать при дисбалансе:

$$\sum_i C_{y_i} L(y_i, \alpha(x_i))$$

Нетрудно видеть, что весовые схемы обобщают идею сэмплирования, но являются более удобной, простой и гибкой техникой. Чтобы не задумываться о значении весов можно в sklearn выбрать `class_weight='balanced'`.

### 33.7. Решающее правило: выбор порога

Обычно модель получает некоторые оценки принадлежности к классам, а сама классификация – это результат бинаризации (по умолчанию порог = 0.5). Но порог можно подбирать, мы рассмотрим простую стратегию: при скользящем контроле по 10 фолдам получим оценки принадлежности классу 1 на обучении (функция `cross_val_predict` в sklearn), потом для заданного функционала качества подберём оптимальный порог бинаризации (при котором значение функционала максимально). Этот же порог будем потом использовать на teste.

Технику выбора порога можно использовать совместно с перевзвешиванием выборки и балансировкой данных. На рис. 5 (слева) показано, как значения качества зависят от выбора порога, на рис. 5 (справа) аналогичные графики приведены после перевзвешивания, видно что «графики растягиваются» и оптимальные пороги смещаются вправо, но при этом оптимальные значения порогов отличны от 0.5.

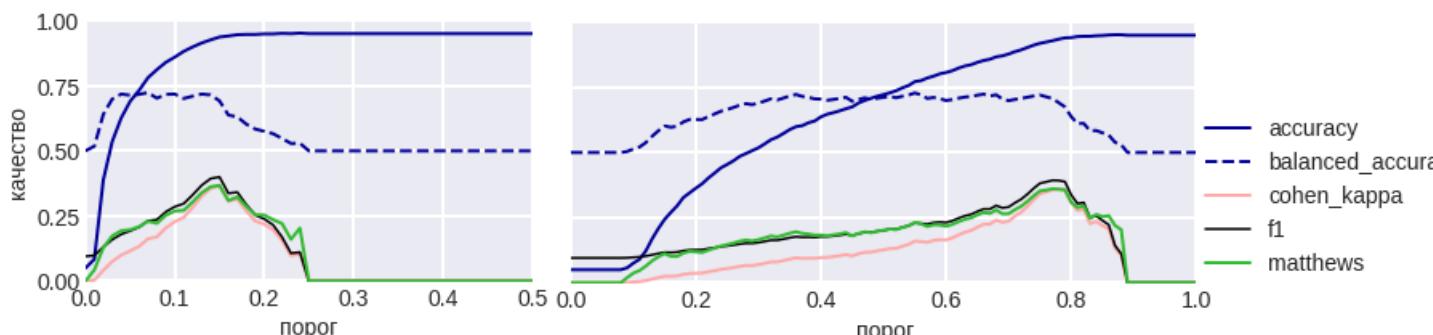


Рис. 5. Показатели качества от значения порога бинаризации до перевзвешивания выборки (слева) и после (справа).

### 33.8. Что использовать на практике

Давайте проведём несколько простых экспериментов, они довольно наглядно пояснят, что и как работает в разных ситуациях на практике. Решаемые задачи (реальные или модельные) не так важны, поэтому мы взяли их из стандартных генераторов sklearn-a. Но вот геометрия данных и используемые модели будут важны. Рассмотрим задачу «два полумесяца» с сильным дисбалансом и разной степенью зашумлённости данных, см. рис. 6.

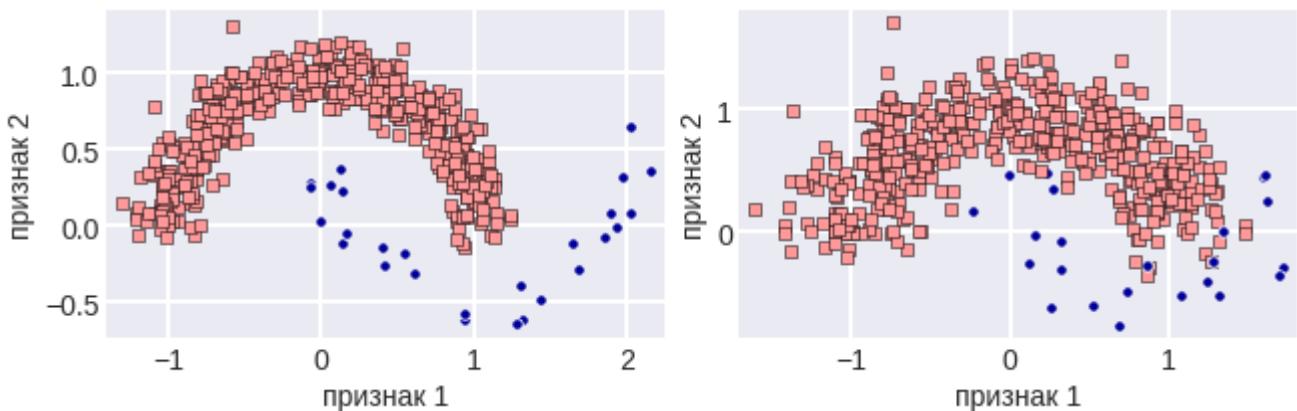


Рис. 6. Решаемая задача «два полумесяца» с разной степенью шума.

Сначала решим задачу с помощью логистической регрессии (хотя здесь нет линейной зависимости, решение должно получиться неплохим), а также с помощью градиентного бустинга (воспользуемся библиотекой LightGBM). Гиперпараметры в обоих методах не будем настраивать и возьмём значения рекомендованные «по умолчанию». В таблицах ниже каждая строка соответствует своему показателю качества, первые пять (точность, сбалансированная точность, каппа Коэна, F1-мера и коэффициент Мэттьюса) зависят от бинаризации (именно они наиболее интересны), остальные (logloss, площади под ROC и PR кривыми) – не зависят. Первый столбец (None) – алгоритм с параметрами по умолчанию, второй (Weights) – использование взвешивания классов, третий (Th-d) – подбор порога на 10-fold-контроле, четвёртый (Th-d + W) – совмещение взвешивания и подбора порога, следующие три столбца – разные техники пересэмплирования, последние пять столбцов – разные техники недосэмплирования.

	None	Weights	Th-d	Th-d + W	RandOS	SMOTE	ADASYN	RandUS	NM1	NM2	TLinks	ENN
accuracy_score	0.970	0.866	0.975	0.974	0.869	0.867	0.939	0.831	0.950	0.895	0.970	0.970
balanced_accuracy_score	0.731	0.881	0.861	0.864	0.880	0.882	0.863	0.857	0.854	0.824	0.731	0.731
cohen_kappa_score	0.619	0.377	0.715	0.714	0.382	0.380	0.558	0.311	0.603	0.397	0.619	0.619
f1_score	0.633	0.431	0.728	0.727	0.435	0.433	0.589	0.372	0.629	0.445	0.633	0.633
matthews_corrcoef	0.669	0.459	0.738	0.732	0.461	0.461	0.578	0.403	0.612	0.441	0.669	0.669
– log_loss	-0.100	-0.291	-0.100	-0.291	-0.283	-0.285	-0.135	-0.361	-0.336	-0.352	-0.100	-0.100
roc_auc_score	0.960	0.962	0.960	0.962	0.962	0.962	0.961	0.956	0.962	0.933	0.960	0.960
average_precision_score	0.788	0.789	0.788	0.789	0.789	0.788	0.784	0.783	0.788	0.704	0.788	0.788

Табл. Задача 1 + логистическая регрессия

	None	Weights	Th-d	Th-d + W	RandOS	SMOTE	ADASYN	RandUS	NM1	NM2	TLinks	ENN
accuracy_score	0.961	0.857	0.963	0.962	0.858	0.867	0.843	0.848	0.904	0.882	0.963	0.966
balanced_accuracy_score	0.665	0.872	0.870	0.871	0.872	0.871	0.870	0.859	0.812	0.770	0.684	0.723
cohen_kappa_score	0.477	0.358	0.623	0.612	0.359	0.374	0.336	0.335	0.411	0.329	0.516	0.583
f1_score	0.492	0.414	0.640	0.648	0.415	0.427	0.395	0.394	0.457	0.382	0.531	0.598
matthews_corrcoef	0.551	0.441	0.577	0.569	0.442	0.451	0.427	0.420	0.446	0.366	0.578	0.621
– log_loss	-0.110	-0.316	-0.110	-0.316	-0.313	-0.293	-0.345	-0.356	-0.392	-0.423	-0.108	-0.107
roc_auc_score	0.952	0.952	0.952	0.952	0.952	0.951	0.951	0.948	0.919	0.878	0.952	0.952
average_precision_score	0.722	0.722	0.722	0.722	0.722	0.718	0.720	0.722	0.609	0.439	0.722	0.722

Табл. Задача 2 + логистическая регрессия

	None	Weights	Th-d	Th-d + W	RandOS	SMOTE	ADASYN	RandUS	NM1	NM2	TLinks	ENNs
accuracy_score	0.992	0.992	0.992	0.992	0.987	0.991	0.991	0.845	0.966	0.917	0.992	0.992
balanced_accuracy_score	0.972	0.970	0.972	0.970	0.941	0.965	0.963	0.840	0.843	0.859	0.972	0.972
cohen_kappa_score	0.929	0.927	0.929	0.927	0.879	0.921	0.920	0.321	0.682	0.482	0.929	0.929
f1_score	0.934	0.932	0.934	0.932	0.886	0.926	0.925	0.380	0.700	0.522	0.934	0.934
matthews_corrcoef	0.930	0.928	0.930	0.928	0.879	0.922	0.920	0.399	0.682	0.519	0.930	0.930
-log_loss	-0.065	-0.051	-0.065	-0.051	-0.130	-0.088	-0.088	-0.468	-0.442	-0.636	-0.065	-0.065
roc_auc_score	0.988	0.995	0.988	0.995	0.967	0.964	0.983	0.908	0.909	0.845	0.988	0.988
average_precision_score	0.970	0.972	0.970	0.972	0.857	0.889	0.878	0.393	0.695	0.440	0.970	0.970

Табл. Задача 1 + градиентный бустинг

	None	Weights	Th-d	Th-d + W	RandOS	SMOTE	ADASYN	RandUS	NM1	NM2	TLinks	ENNs
accuracy_score	0.983	0.983	0.983	0.983	0.973	0.981	0.980	0.846	0.927	0.264	0.982	0.963
balanced_accuracy_score	0.892	0.902	0.926	0.930	0.843	0.923	0.916	0.840	0.784	0.482	0.901	0.935
cohen_kappa_score	0.832	0.837	0.826	0.832	0.728	0.825	0.817	0.320	0.453	-0.005	0.828	0.715
f1_score	0.841	0.846	0.835	0.841	0.742	0.835	0.828	0.380	0.491	0.101	0.837	0.734
matthews_corrcoef	0.834	0.838	0.829	0.834	0.730	0.825	0.818	0.399	0.465	-0.019	0.828	0.730
-log_loss	-0.103	-0.102	-0.103	-0.102	-0.191	-0.130	-0.140	-0.384	-0.397	-0.699	-0.111	-0.255
roc_auc_score	0.976	0.964	0.976	0.964	0.968	0.962	0.960	0.909	0.885	0.738	0.975	0.976
average_precision_score	0.884	0.874	0.884	0.874	0.837	0.872	0.843	0.449	0.464	0.403	0.878	0.758

Табл. Задача 2 + градиентный бустинг

Какие выводы можно сделать?

Подбор порога (без совмещения с любой другой техникой) – идеальная стратегия для «нешумных данных». На самом деле, только это и надо использовать, когда геометрия данных относительно проста, модель хорошо описывает данные (и особенно, если хорошо откалибрована). Обратим внимание, что качество признакового пространства (шум и геометрия) в классическом ML зависит исключительно от Вас, поэтому, **если Вы умеете решать задачи, то кроме подбора порога Вам ничего не нужно**. Бустинг, который идеально справился с задачей, показывает хорошее качество по умолчанию или с использованием весовой схемы (т.е. достаточно оптимизировать гиперпараметры). Вообще, полезно запомнить – **хорошие признаки и правильно подобранная модель это самое главное в ML**, всё остальное от лукавого (и не важно есть дисбаланс или нет его).

Неужели описанные схемы сэмплирования особо не нужны? На самом деле, для любой схемы можно найти применение (надо только взять «не очень подходящую» модель и/или увеличить шум). В табл. ниже показано качество метода случайный лес, видим, что SMOTE здесь явно предпочтителен (хотя сами показатели значительно просели).

	None	Weights	Th-d	Th-d + W	RandOS	SMOTE	ADASYN	RandUS	NM1	NM2	TLinks	ENNs
accuracy_score	0.976	0.978	0.965	0.977	0.980	0.979	0.975	0.947	0.322	0.344	0.976	0.973
balanced_accuracy_score	0.822	0.831	0.895	0.922	0.866	0.914	0.902	0.908	0.597	0.636	0.841	0.848
cohen_kappa_score	0.737	0.764	0.692	0.787	0.793	0.805	0.777	0.622	0.029	0.042	0.753	0.735
f1_score	0.749	0.775	0.711	0.799	0.804	0.817	0.790	0.648	0.131	0.142	0.765	0.749
matthews_corrcoef	0.747	0.775	0.694	0.787	0.797	0.806	0.777	0.645	0.100	0.137	0.758	0.736
-log_loss	-0.194	-0.201	-0.201	-0.154	-0.159	-0.172	-0.308	-0.193	-1.367	-0.967	-0.163	-0.204
roc_auc_score	0.952	0.949	0.950	0.962	0.961	0.967	0.945	0.969	0.672	0.851	0.959	0.953
average_precision_score	0.819	0.845	0.822	0.862	0.844	0.838	0.760	0.758	0.453	0.633	0.838	0.772

Табл. Задача 2 + случайный лес

### 33.9. Какие ещё методы существуют?

В DL есть свои методы учёта дисбаланса. Перебалансировка выборки здесь чаще производится на уровне батчей (например, так делали при формировании батчей для обучения R-CNN), причём её можно совместить с аугментацией (про MixUp мы уже вспоминали). Также есть специфические задачи, в которых дисбаланс связан ещё и с эффективностью вычислений, см. например Negative Sampling. Кроме того, есть специальные функции ошибок для обучения нейросетей при дисбалансе, например focal loss (она же используется для калибровки сетей). Ниже дадим ссылку на очень хороший обзор.

## 34. Одноклассовые методы и обнаружение аномалий. Конспект Соколова

В задачах кластеризации, о которых шла речь ранее, требуется разделить выборку на группы так, чтобы внутри каждой группы объекты были похожи друг на друга. Теперь мы изучим немного другую постановку - поиск аномалий. В ней даётся выборка "нормальных" объектов, и требуется построить некоторую модель, описывающую данную выборку. Далее для новых объектов требуется определять, принадлежат ли они тому же распределению, что и эта выборка, или же являются выбросами или аномалиями. Такие методы применяются, например, в задачах обнаружения мошеннического поведения или раннего обнаружения неполадок оборудования

### 34.1. Несбалансированная классификация

В некоторых задачах примеры аномалий могут быть даны, но в небольших объёмах - например, при анализе данных систем самолёта может быть известно несколько аномальных ситуаций из прошлого. Такую задачу можно рассматривать как классификацию с несбалансированными классами. При решении обычными методами классификатору может оказаться выгоднее относить все объекты к одному классу, поэтому имеет смысл модифицировать процедуру обучения.

Самые простые методы борьбы с несбалансированностью - undersampling и oversampling. Первый из них удаляет случайные объекты доминирующего класса до тех пор, пока соотношение классов не станет приемлемым. Второй дублирует случайные объекты минорного класса. Оптимально число объектов для удаления или дублирования следует подбирать с помощью кросс-валидации. Отметим, что данные методы применяются лишь к обучающей выборке, а контрольная выборка остаётся без изменений.

Более сложный метод SMOTE заключается в дополнении минорного класса синтетическими объектами. Генерация нового объекта производится следующим образом. Выбирается случайный объект  $x_1$  минорного класса, для него выделяются  $k$  ближайших соседей из этого же класса ( $k$  - настраиваемый параметр), из этих соседей выбирается один случайный  $x_2$ . Новый объект вычисляется как точка на отрезке между  $x_1$  и  $x_2$ :  $\alpha x_1 + (1 - \alpha)x_2$ , для случайного  $\alpha \in (0; 1)$ .

### 34.2. Одноклассовая классификация

Ниже мы будем обсуждать обнаружение точечных аномалий - объектов, которые существенно отличаются от заданной выборки. При этом выделяют и другие типы. Так, контекстными аномалиями называют наблюдения, отличающиеся от наблюдений, близких по некоторому параметру. Например, температура  $-10^\circ$  является нормальной в январе, но аномальной в июне.

#### 34.2.1. Статистические методы

В статистических методах предлагается восстановить плотность выборки  $p(x)$ , и затем определять аномальность объекта на основе того, насколько вероятно его получить из данной плотности. Например, это можно делать через отклонение от среднего  $[\rho(x, \mu) > d]$  (порог может подбираться, если известно некоторое количество примеров аномалий), сравнение значения плотности с порогом  $[p(x) < d]$  или с помощью статистических тестов. Существует два подхода к восстановлению плотности: параметрический и непараметрический.

**34.2.1.1. Непараметрический подход** Начнём с одномерных величин. Согласно одному из определений неотрицательная функция  $p(x)$  является плотностью распределения случайной величины  $\xi$ , если её значение в каждой точке равно пределу

$$p(x) = \lim_{h \rightarrow 0} \frac{1}{2h} \mathbb{P}(\xi \in [x - h, x + h])$$

Воспользуемся этим определением и построим эмпирическую оценку плотности:

$$\hat{p}(x) = \frac{1}{2lh} \sum_{i=1}^l [\|x - x_i\| < h] = \frac{1}{lh} \sum_{i=1}^l \frac{1}{2} \left[ \frac{\|x - x_i\|}{h} < 1 \right]$$

где  $h$  - ширина окна, регулирующая гладкость эмпирической плотности. Чем больше объектов обучающей выборки в окрестности точки, тем выше будет плотность.

В указанной оценке используется индикатор, что приводит к отсутствию гладкости. Чтобы устранить это, заменим индикатор того, что расстояние меньше ширины окна, на некоторую гладкую функцию  $K(z)$ :

$$\hat{p}(x) = \frac{1}{lh} \sum_{i=1}^l K\left(\frac{x - x_i}{h}\right)$$

Здесь  $K(z)$  - ядро (не путайте с ядрами Мерсера!), которое должно удовлетворять четырём требованиям:

- чётность:  $K(-z) = K(z)$
- нормированность:  $\int K(z)dz = 1$
- неотрицательность:  $K(z) \geq 0$
- невозрастание при  $z > 0$

Примером может служить гауссово ядро  $K(z) = (2\pi)^{-1/2}e^{-0.5z^2}$ .

Оценку плотности легко обобщить на многомерный случай, заменив разность  $\|x - x_i$  на некоторую метрику  $\rho(x, x_i)$ :

$$\hat{p}(x) = \frac{1}{lV(h)} \sum_{i=1}^l K\left(\frac{\rho(x, x_i)}{h}\right)$$

где  $V(h) = \int K\left(\frac{\rho(x, x_i)}{h}\right) dx$  - нормировочная константа. Следует помнить, что число объектов, необходимое для качественной оценки плотности, растёт экспоненциально по мере роста числа признаков. Из-за этого непараметрические методы подходят только для обнаружения аномалий в маломерных пространствах.

**34.2.1.2. Параметрический подход** Параметрический подход состоит в приближении плотности с помощью распределения  $p(x|\theta)$  из некоторого семейства  $\{p(x|\theta)|\theta \in \Theta\}$  с помощью метода максимального правдоподобия:

$$\sum_{i=1}^l \log p(x_i|\theta) \rightarrow \max_{\theta}$$

В качестве распределений могут выступать, например, нормальные или смеси нормальных. В пространствах большой размерности может иметь смысл наивное байесовское предположение о котором пойдёт речь на семинарах.

### 34.2.2. Метрические методы

Метрический подход основан на выделении объектов, которые расположены от других существенно дальше, чем объекты в среднем удалены друг от друга. А именно, объект  $x$  объявляется аномальным, если  $p$  или меньше процентов объектов имеют до него расстояние меньше  $\varepsilon$ :

$$\frac{1}{l} \sum_{i=1}^l [\rho(x, x_i) < \varepsilon] \leq p$$

Пороги  $p$  и  $\varepsilon$  являются параметрами, которые должны настраиваться по известным примерам аномалий или исходя из априорных предположений.

### 34.2.3. Одноклассовый метод опорных векторов

Заметим, что похожим образом можно применять любую модель для обнаружения аномалий - достаточно обучить её так, чтобы прогнозы для объектов из обучения были близки к нулю или, наоборот, как можно сильнее отделены от нуля.

Выше мы пытались описать данные с помощью распределения или использовать метрику, чтобы оценить аномальность объекта. Далее мы разберём подход на основе моделей. Действительно, можно взять любую модель машинного обучения и настроить её так, чтобы на нормальных объектах она принимала близкие к нулю или, например, положительные значения. Тогда можно будет считать, что если на новом объекте прогноз сильно отличается от прогнозов на обучающей выборке, то этот объект скорее аномальный. Мы поговорим о двух методах: на основе SVM и на основе решающих деревьев.

Для обнаружения аномалий, по сути, необходимо построить некоторую функцию  $\alpha(x)$ , которая принимает значение 1 на области как можно меньшего объёма, содержащей как можно больше объектов выборки. Во всех остальных точках она должна иметь значение 0. Такая функция будет компактно описывать обучающую выборку, и можно рассчитывать, что на аномальных объектах она будет отрицательной.

Будем строить линейную функцию  $\alpha(x) = \text{sign}(\langle w, x \rangle)$ , и потребуем, чтобы она отделяла выборку от начала координат с максимальным отступом. Соответствующая оптимизационная задача будет иметь вид

$$\begin{cases} \frac{1}{2} \|w\|^2 + \frac{1}{vl} \sum_{i=1}^l \xi_i - \rho \rightarrow \min_{w, \xi, \rho} \\ \langle w, x_i \rangle \geq \rho - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Здесь гиперпараметр  $v$  отвечает за корректность на обучающей выборке - можно показать, что он является верхней границей на число аномалий (объектов выборки, на которых  $\alpha(x) = -1$ ). Решающее правило будет иметь вид

$$\alpha(x) = \text{sign}(\langle w, x \rangle - \rho)$$

где ответ  $-1$  будет соответствовать выбросу. Получается, что мы ищем гиперплоскость так, что:

- она отделяет как можно больше объектов выборки от нуля (чем меньше  $v$ , тем больше объектов мы будем отделять) - за это отвечает слагаемое  $\frac{1}{vl} \sum_{i=1}^l \xi_i$  в функционале
- она имеет большой отступ  $\frac{1}{\|w\|^2}$
- она при этом как можно сильнее отдалена от нуля (то есть  $\rho$  как можно большее значение)

Для данной задачи можно выписать двойственную и сделать ядровой переход в ней:

$$\begin{cases} \frac{1}{2} \sum_{i,j=1}^l \lambda_i \lambda_j K(x_i, x_j) \rightarrow \min_{\lambda} \\ 0 \leq \lambda_i \leq \frac{1}{vl}, i = 1, \dots, l \\ \sum_{i=1}^l \lambda_i = 1 \end{cases}$$

Модель при этом будет иметь вид

$$\alpha(x) = \text{sign} \left( \sum_{i=1}^l \lambda_i K(x, x_i) - \rho \right)$$

Заметим, что при использовании гауссова ядра, данная модель будет очень похожа на метод, который строит непараметрическую оценку плотности (34.2.1.1) с гауссовым ядром и сравнивает её значение с порогом  $\rho$ .

При использовании подходящих ядер можно действительно получить функцию, которая точно описывает обучающую выборку в исходном пространстве. Также можно показать, что объекты из того же распределения, из которого сгенерирована обучающая выборка, будут с не очень большой вероятностью попадать в область с отрицательным значением  $\alpha(x)$ .

#### 34.2.4. Isolation forest

Ранее мы обсуждали, что случайный лес вводит функцию расстояния - чем чаще два объекта попадают в один лист, тем более похожими их можно считать. Похожий подход можно использовать и для обнаружения аномалий. Метод, который мы разберём называют изоляционным лесом (Isolation forest).

На этапе обучения будем строить лес, состоящий из  $N$  деревьев. Каждое дерево будем строить стадартным жадным алгоритмом, но при этом признак и порог будет выбирать случайно. Строить дерево будем до тех пор, пока в вершине не окажется ровно один объект, либо пока не будет достигнута максимальная высота. Высоту дерева можно ограничить величиной  $\log_2(l)$ .

Метод основан на предположении о том, что чем сильнее объект отличается от большинства, тем быстрее он будет отделён от основной выборки с помощью случайных разбиений. Соответственно, выбросами будем считать те объекты, которые оказались на небольшой глубине.

Чтобы вычислить оценку аномальности объекта  $x$ , найдём расстояние от соответствующего ему листа до корня в каждом дереве. Если лист, в котором оказался объект, содержит только его, то в качестве оценки  $h_n(x)$  от данного  $n$ -го дерева будем брать саму глубину  $k$ . Если же в листе оказалось  $m$  объектов,

то в качестве оценки возьмём величину  $h_n(x = k + c(m))$ . Здесь  $c(m)$  - средняя длина пути от корня до листа в бинарном дереве поиска, которая вычисляется по формуле

$$c(m) = 2H(m-1) - 2 \frac{m-1}{m}$$

а  $H(i) \approx \ln(i) + 0.5772156649$  -  $i$ -е гармоническое число. Оценку аномальности вычислим на основе средней глубины, нормированной на среднюю длину пути в дереве, построенном на выборке размера  $l$ :

$$\alpha(x) = 2^{-\frac{\sum_{n=1}^N h_n(x)}{c(l)}}$$

Для ускорения работы можно строить каждого дерева на подвыборке размера  $s$ . В этом случае во всех формулах выше нужно заменить  $l$  на  $s$

## 35. Лекция 15. Дисбаланс классов и поиск аномалий. Наша лекция

### 35.1. Практическая работа с балансировкой данных

#### 35.1.1. Импортируем библиотеки

```
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, roc_auc_score
from sklearn.model_selection import train_test_split

from collections import Counter

import matplotlib.pyplot as plt
import seaborn as sns

#plt.rcParams['figure.figsize'] = [20, 7]
```

#### 35.1.2. Загружаем датасет

```
# https://disk.yandex.ru/i/3my_swzwW4d5Q
data = pd.read_csv(r'creditcard.csv')
data.head()
data.shape
data.Class.value_counts()
```

Здесь видно, что классы представлены в разной пропорции.

#### 35.1.3. Предобрабатываем данные

```
# separate fraudulent and non fraudulent data
data_0 = data[data['Class'] == 0]
data_1 = data[data['Class'] == 1]

data_0 = data_0.sample(n=9000)

data = data_1.append(data_0)

data.Class.value_counts()

# save as csv
data.to_csv('credit-card.csv')

# check the number of 1s and 0s
count = data['Class'].value_counts()

print('Fraudulent "1": ', count[1])
print('Not Fraudulent "0": ', count[0])

# print the percentage of question where target == 1
print(count[1]/count[0]* 100)

# plot the no of 1's and 0's
g = sns.countplot(data['Class'])
g.set_xticklabels(['Not Fraud', 'Fraud'])
plt.show()
```

### 35.1.4. Разбиение на train и test

```
x = data.iloc[:, :-1]
y = data.iloc[:, -1]

# check length of 1's and 0's
one = np.where(y==1)
zero = np.where(y==0)
len(one[0]), len(zero[0])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

### 35.1.5. Обучим логрег

```
# create the object
model = LogisticRegression()

model.fit(x_train, y_train)

y_predict = model.predict(x_test)

roc_auc_score(y_test, y_predict)

confusion_matrix(y_test, y_predict)
```

### 35.1.6. Попробуем обучить бустинг

```
# import library
from xgboost import XGBClassifier

xgb_model = XGBClassifier().fit(x_train, y_train)

# predict
xgb_y_predict = xgb_model.predict(x_test)

# accuracy score
xgb_score = accuracy_score(xgb_y_predict, y_test)

print('Roc_auc_score:', roc_auc_score(xgb_y_predict, y_test))

confusion_matrix(y_test, xgb_y_predict)
```

### 35.1.7. Посмотрим сколько у нас объектов каждого класса

```
# class count
class_count_0, class_count_1 = data['Class'].value_counts()

# divide class
class_0 = data[data['Class'] == 0]
class_1 = data[data['Class'] == 1]

# print the shape of the class
print('class_0:', class_0.shape)
print('\n', 'class_1:', class_1.shape)
```

### 35.1.8. Случайный undersampling

```

class_0_under = class_0.sample(class_count_1)

test_under = pd.concat([class_0_under, class_1], axis=0)

print("total class of 1 and 0:\n", test_under['Class'].value_counts())

test_under['Class'].value_counts().plot(kind='bar', title='Count_(target)')
plt.show()

```

### 35.1.9. Случайный oversampling

```

class_1_over = class_1.sample(class_count_0, replace=True)

test_under = pd.concat([class_1_over, class_0], axis=0)

# print the number of class count
print('class_count_of_1_and_0:\n', test_under['Class'].value_counts())

# plot the count
test_under['Class'].value_counts().plot(kind='bar', title='Count_(target)')
plt.show()

```

### 35.1.10. undersampling с библиотекой imblearn

```

# import library
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42, replacement=True)

# fit predictor and target varialbe
x_rus, y_rus = rus.fit_resample(x_train, y_train)

print('original_dataset_shape:', Counter(y_train))
print('Resample_dataset_shape', Counter(y_rus))

model.fit(x_rus, y_rus)

predict = model.predict_proba(x_test)[:, 1]

roc_auc_score(y_test, predict)

```

### 35.1.11. oversampling с библиотекой imblearn

```

# import library
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)

# fit predictor and target varaible
x_ros, y_ros = ros.fit_resample(x_train, y_train)

print('Original_dataset_shape', Counter(y_train))
print('Resample_dataset_shape', Counter(y_ros))

model.fit(x_ros, y_ros)

predict = model.predict_proba(x_test)[:, 1]

roc_auc_score(y_test, predict)

```

### 35.1.12. undersampling Tomek links

```
# load library
from imblearn.under_sampling import TomekLinks

t1 = TomekLinks(sampling_strategy='majority')

# fit predictor and target variable
x_t1, y_t1 = t1.fit_resample(x_train, y_train)

print('Original_dataset_shape:', Counter(y_train))
print('Resample_dataset_shape:', Counter(y_t1))

model.fit(x_t1, y_t1)

predict = model.predict_proba(x_test)[:, 1]

roc_auc_score(y_test, predict)
```

### 35.1.13. Синтетический oversampling меньшего класса (SMOTE)

```
# load library
from imblearn.over_sampling import SMOTE

smote = SMOTE()

# fit target and predictor variable
x_smote, y_smote = smote.fit_resample(x_train, y_train)

print('Original_dataset_shape:', Counter(y_train))
print('Resample_dataset_shape:', Counter(y_smote))

model.fit(x_smote, y_smote)

predict = model.predict_proba(x_test)[:, 1]

roc_auc_score(y_test, predict)
```

### 35.1.14. NearMiss

```
from imblearn.under_sampling import NearMiss

nm = NearMiss()

x_nm, y_nm = nm.fit_resample(x_train, y_train)

print('Original_dataset_shape:', Counter(y_train))
print('Resample_dataset_shape:', Counter(y_nm))

model.fit(x_nm, y_nm)

predict = model.predict_proba(x_test)[:, 1]

roc_auc_score(y_test, predict)
```

### 35.1.15. Penalize algorithm (cost-sensitive training)

```
# load library
from sklearn.linear_model import LogisticRegression
```

```

# we can add class_weight='balanced' to add penalize mistake
model = LogisticRegression(class_weight='balanced')

model.fit(x_train, y_train)

predict = model.predict_proba(x_test)

print('ROC-AUC-score: ', roc_auc_score(y_test, predict))

```

### 35.1.16. Xgboost с лучшей техникой семплирования

```

xgb_model = XGBClassifier().fit(x_rus, y_rus)

# predict
xgb_y_predict = xgb_model.predict(x_test)

print('Roc_auc_score: ', roc_auc_score(xgb_y_predict, y_test))

```

## 35.2. Поиск аномалий

В поиске аномалии разница с дисбалансом классов в том, что дисбаланс ещё более жёсткий и что у нас нет разметки. Поэтому можно решать задачи дисбаланса классов поиском аномалий, а вот обратно не работает.

```

import numpy as np
from numpy import genfromtxt
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import multivariate_normal
from sklearn.metrics import f1_score

import warnings
warnings.filterwarnings("ignore")

```

### 35.2.1. Введение

**Задача поиска аномалий (Anomaly Detection)** - один из вариантов обучения без учителя (Unsupervised Learning): обычно примеров аномалий или нет, или их достаточно мало.

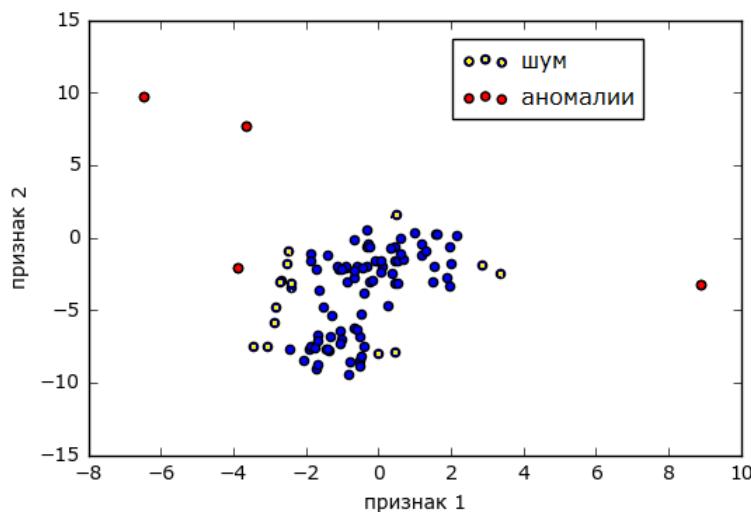
Можно выделить два направления, занимающихся поиском аномалий:

- Детектирование выбросов (Outlier Detection)
- Детектирование «новизны» (Novelty Detection)

В статистике **выбросом** называют результат измерения, выделяющийся из общей выборки. Выбросы являются следствием:

- ошибок в данных (неточности измерения, округления, неверной записи и т.п.)
- наличия шумовых объектов (неверно классифицированных объектов)
- присутствия объектов «других» выборок (например, показания сломавшегося датчика).

«**Новый объект**», как и выброс, — это объект, который отличается по своим свойствам от объектов (обучающей) выборки. Но в отличие от выброса, его в самой выборке пока нет (он появится через некоторое время, и задача как раз и заключается в том, чтобы обнаружить его при появлении). **Новизна**, как правило, появляется в результате принципиально нового поведения объекта.



На рисунке видно, что шум (noise) — это выброс «в слабом смысле» (он может немного размывать границы класса/кластера). Нас же интересуют, прежде всего, выбросы «в сильном смысле», которые искажают эти границы.

Практические приложения:

- Обнаружение подозрительных банковских операций (Credit-card Fraud)
- Обнаружение вторжений (Intrusion Detection)
- Обнаружение нестандартных игроков на бирже (инсайдеров)
- Обнаружение неполадок в механизмах по показаниям датчиков
- Медицинская диагностика (Medical Diagnosis)
- Сейсмология

### 35.2.2. Простые методы

#### 35.2.2.1. Box plot

Будем работать с набором данных Boston house prices dataset.

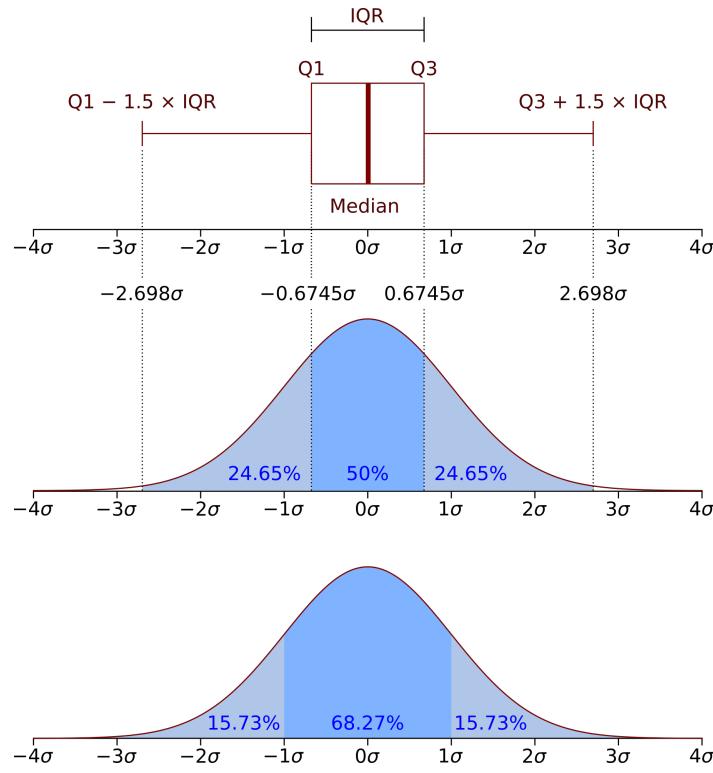
```
from sklearn.datasets import fetch_california_housing

# load data
boston = fetch_california_housing(as_frame=True)

X = boston.data
y = boston.target

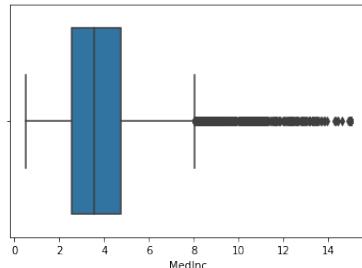
# Show data
columns = boston.feature_names
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
boston_df.head()
```

Построение Box plot



Построим box plot для признака DIS.

```
sns.boxplot(x=boston_df['MedInc']);
```



Как видно из графика, присутствует некоторое количество выбросов. Такой способ удобен для визуализации, но не для нахождения самих выбросов.

Исходя из методики построения ящика с усами, выбросами считаются точки, чьё значение признака  $x$ :

- $x > Q_3 + 1.5 \times IQR$
- $x < Q_1 - 1.5 \times IQR$

где **интерквартильный размах**  $IQR = Q_3 - Q_1$

Какие недостатки у этого метода?

- Вся теория работает, если мы работаем с нормально-распределёнными признаками, если это не так, то мы не можем просто основываться на ящике с усами для детектирования аномалий.
- Во-вторых - это однофакторный метод, который смотрит на каждый признак по отдельности. Объект - это набор признаков, и объект может быть аномалией по двум признакам, но не аномалией по одному и подобное. Нам нужен метод, который работает с многомерной точкой.

Но давайте посчитаем всё равно boxplot вручную и выкинем "выбросы"

```
Q1 = boston_df['MedInc'].quantile(0.25)
Q3 = boston_df['MedInc'].quantile(0.75)
IQR = Q3 - Q1
```

```
outlier_idx = boston_df[(boston_df['MedInc'] < (Q1 - 1.5 * IQR)) | (boston_df['MedInc'] >
sns.boxplot(x=boston_df['MedInc'].drop(outlier_idx));
```

**35.2.2.2. Z-score Стандартизованная оценка** (z-оценка, standard score, z-score) - это мера относительного разброса наблюдаемого или измеренного значения, которая показывает сколько стандартных отклонений составляет его разброс относительного среднего значения.

**Интуиция.** Вычисляя Z-score, мы масштабируем и центрируем данные и смотрим на точки, которые находятся далеко от 0.

$$z = \frac{x - \bar{x}}{std}$$

Точки, которые достаточно далеко от 0, считаются выбросами. В большинстве случаев используется порог 3 или -3, т.е. если Z-score больше 3 или меньше -3, то точка считается выбросом.

**Замечание:** правило трёх сигм гласит, что приблизительно с вероятностью 0,9973 значение нормально распределённой случайной величины лежит в интервале  $3\sigma$ .

Давайте посчитаем это руками:

```
x_mean = boston_df[ 'MedInc' ].mean()
x_std = boston_df[ 'MedInc' ].std()

z_score = (boston_df[ 'MedInc' ] - x_mean) / x_std
z_score_abs = abs(z_score)

outlier_idx = boston_df[z_score_abs > 3].index

sns.boxplot(x=boston_df[ 'MedInc' ].drop(outlier_idx))
```

Опять же Z-score имеет те же недостатки, что и boxplot.

### 35.2.3. Elliptic Envelope

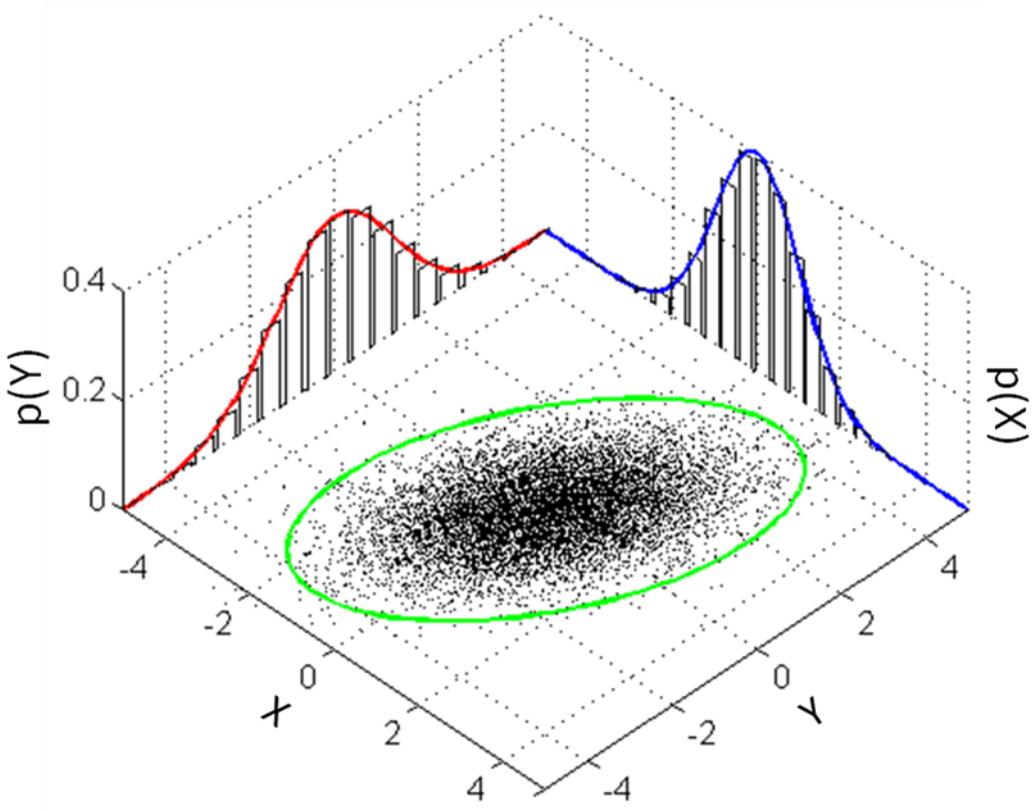
Будем считать, что распределение данных подчиняется многомерному нормальному распределению. Будем оценивать параметры этого распределения  $\mu$  и  $\Sigma$  по данным, где

- $\mu$  - вектор средних значений
- $\Sigma$  - ковариационная матрица

И плотность вероятности:

$$f_X(x_1, \dots, x_k) = \frac{1}{\sqrt{2\pi}^k |\Sigma|} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

По сути мы строим эллипс, и всё, что в него не попадает - считаем выбросами.



Давайте запрограммируем это дело:

```

from scipy.stats import multivariate_normal
from sklearn.metrics import f1_score

# Returns the estimated distribution parameters
def estimateGaussian(dataset):
    #
    mu = np.mean(dataset, axis=0)
    sigma = np.cov(dataset.T)
    #
    return mu, sigma

# Returns the probability for each object
def multivariateGaussian(dataset, mu, sigma):
    #
    p = multivariate_normal(mean=mu, cov=sigma)
    return p.pdf(dataset)
    #

# Returns the best f1 value and the best threshold
def selectThresholdByCV(probs, target):
    best_epsilon = 0
    best_f1 = 0
    f = 0

    stepsize = (max(probs) - min(probs)) / 1000;
    epsilons = np.arange(min(probs), max(probs), stepsize)

    for epsilon in np.nditer(epsilons):
        #
        predictions = (probs < epsilon)
        f = f1_score(target, predictions, average='binary')
        if f > best_f1:
            best_f1 = f

```

```

best_epsilon = epsilon
#
```

```
return best_f1, best_epsilon
```

Для этой задачи мы возьмём данные, мы возьмём данные, про которые мы примерно представляем где у нас выбросы и по ним подберём порог. Порог отбираем по  $f1$  score.

Загрузим данные:

```
! wget https://raw.githubusercontent.com/hse-ds/iad-applied-ds/master/2022/seminars/sem14/
! wget https://raw.githubusercontent.com/hse-ds/iad-applied-ds/master/2022/seminars/sem14/
! wget https://raw.githubusercontent.com/hse-ds/iad-applied-ds/master/2022/seminars/sem14/
```

Разделим на тренинг и тест:

```
columns_names = [ 'Latency_(ms)', 'Throughput_(mb/s)' ]

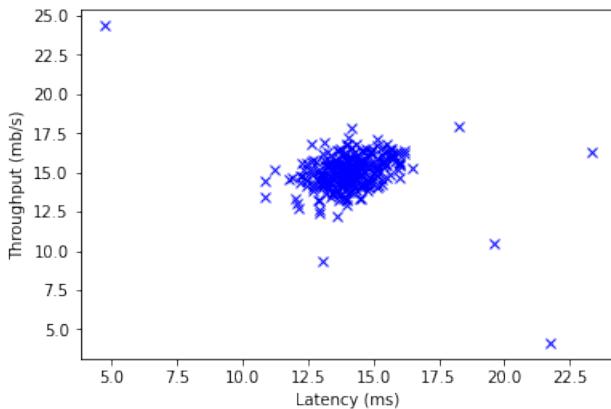
# train data
data_train = pd.read_csv('tr_server_data.csv', names=columns_names)

# validation data
data_val = pd.read_csv('cv_server_data.csv', names=columns_names)

# target for validation
data_val_tr = pd.read_csv('gt_server_data.csv', names=['target'])
```

Построим график наших данных

```
plt.xlabel('Latency_(ms)')
plt.ylabel('Throughput_(mb/s)')
plt.plot(data_train.iloc[:,0], data_train.iloc[:,1], 'bx')
plt.show()
```



Теперь:

1. Оценим параметры распределения по тренировочным данным
2. Оценим вероятность для каждого объекта тренировочной выборки
3. Оценим вероятность для каждого объекта валидационной выборки
4. Выберем оптимальное значение порога
5. Найдём индексы аномальных объектов в тренировочной выборке
6. Построим график, на котором аномальные объекты отображены красным цветом

```
#1
mu, sigma = estimateGaussian(data_train)
```

```

#2
p = multivariateGaussian(data_train, mu, sigma)

#3
p_cv = multivariateGaussian(data_val, mu, sigma)

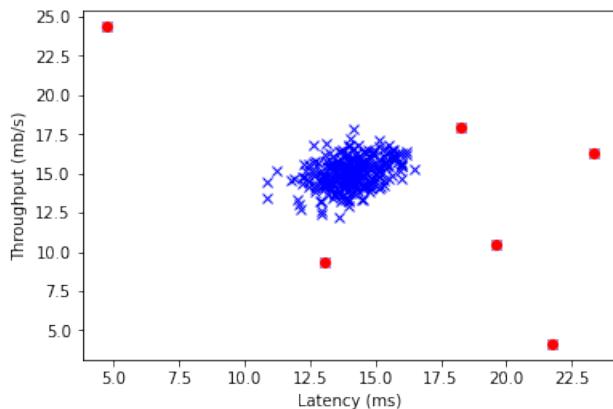
#4
fscore, ep = selectThresholdByCV(p_cv, data_val_tr)

print(fscore, ep)

#5
outliers = np.asarray(np.where(p < ep))

#6
plt.figure()
plt.xlabel('Latency_(ms)')
plt.ylabel('Throughput_(mb/s)')
plt.plot(data_train.iloc[:,0], data_train.iloc[:,1], 'bx')
plt.plot(data_train.iloc[outliers[0],0], data_train.iloc[outliers[0],1], 'ro')
plt.show()

```



При этом мы нашли выбросы скорее, а не шумы, потому что шумы это скорее те крестики, которые немного слева от окружности, а вот красные квадраты - выбросы.

#### 35.2.4. Одноклассовый SVM

Вариант метода опорных векторов, который отделяет выборку от начала координат. Использование ядер позволяет обойти сомнительное предположение о том, что объекты должны располагаться вдали от начала координат. По умолчанию, в качестве ядра используют лишь rbf (радиальные базисные функции), остальные ядра показывают плохие результаты.

**OneClassSVM** это скорее алгоритм поиска новизны, а не выбросов, т.к. «затачивается» под обучающую выборку.

Загрузим данные:

```

from sklearn import svm

data_train = pd.read_csv('tr_server_data.csv', names=columns_names)

Обучим нашу модель:

clf = svm.OneClassSVM(nu=0.05, kernel="rbf", gamma=0.1)
clf.fit(data_train)

pred = clf.predict(data_train)

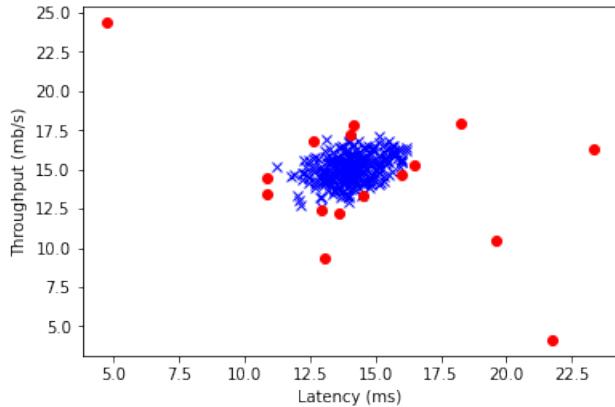
normal = data_train[pred == 1]
abnormal = data_train[pred == -1]

```

```

plt.figure()
plt.plot(normal.iloc[:, 0], normal.iloc[:, 1], 'bx')
plt.plot(abnormal.iloc[:, 0], abnormal.iloc[:, 1], 'ro')
plt.xlabel('Latency (ms)')
plt.ylabel('Throughput (mb/s)')

```



Обычный SVM пытается построить прямую, которая будет разделяющей гиперплоскостью. Тут же у нас SVM с ядрами, а конкретно с радиальным ядром - старается завернуть обычные данные в замкнутую фигуру. Это и есть одноклассовый SVM.

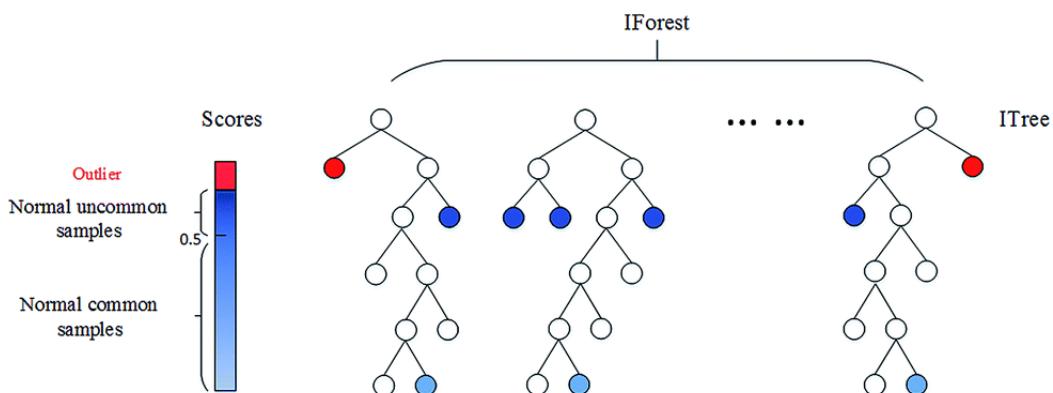
### 35.2.5. Изолирующий лес (Isolation Forest)

Isolation Forest (оригинальная статья) «изолирует» наблюдения следующим образом:

1. сначала, случайным образом выбирается признак,
2. затем, случайным образом выбирает разделяющее значение (split value) между максимальным и минимальным значениями выбранного признака.

Идея в том, что если у нас аномалия - она быстро отделяется от всех остальных.

Поскольку рекурсивное разбиение может быть представлено древовидной структурой, количество разбиений, необходимых для выделения объекта, эквивалентно длине пути от корневого узла до конечного узла.



Если есть аномалия, то любым деревом, абсолютно произвольным, эта аномалия на первом или втором уровне отделяется.

Мы для точки считаем среднюю глубину по всем деревьям.

Эта длина пути, усредненная по лесу таких случайных деревьев, является мерой нормальности нашей решающей функции.

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

где

- $n$  - количество внешних узлов
- $h(x)$  - длина пути до наблюдения  $x$
- $c(n)$  - средняя длина пути неудачного поиска в бинарном дереве поиска

Сгенерируем данные и посмотрим на них.

```

rng = np.random.RandomState(42)

# test data
X_train = 0.2 * rng.randn(1000, 2)
X_train = np.r_[X_train + 3, X_train]
X_train = pd.DataFrame(X_train, columns = [ 'x1' , 'x2' ])

# test data - new "normal" data
X_test = 0.2 * rng.randn(200, 2)
X_test = np.r_[X_test + 3, X_test]
X_test = pd.DataFrame(X_test, columns = [ 'x1' , 'x2' ])

# anomaly data
X_outliers = rng.uniform(low=-1, high=5, size=(50, 2))
X_outliers = pd.DataFrame(X_outliers, columns = [ 'x1' , 'x2' ])

```

Построим график:

```

plt.figure(figsize=(10,10))
plt.title("Data")

p1 = plt.scatter(X_train.x1, X_train.x2, c='white', s=20*4, edgecolor='k')
p2 = plt.scatter(X_test.x1, X_test.x2, c='green', s=20*4, edgecolor='k')
p3 = plt.scatter(X_outliers.x1, X_outliers.x2, c='red', s=20*4, edgecolor='k')

plt.axis('tight')
plt.xlim((-2, 5))
plt.ylim((-2, 5))
plt.legend([p1, p2, p3], ["train", "test_normal", "abnormal"], loc="lower_right")

plt.show()

```

Теперь:

1. Обучим IsolationForest:
2. Посчитаем долю верных ответов (accuracy) для тестовой "нормальной" выборки и для аномалий
3. Построим scatter plot, на котором видно какие наблюдения изолирующий лес считает аномалиями

```

# 1
clf = IsolationForest(max_samples=100, contamination=0.1, random_state=rng)
clf.fit(X_train)

# prediction
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)

# test "normal" sample
print("Accuracy (normal data):", len(y_pred_test[y_pred_test==1])/len(y_pred_test))

# anomalies
print("Accuracy (outliers):", len(y_pred_outliers[y_pred_outliers== -1])/len(y_pred_outlier))

# adding the predicted label
X_outliers_pred = X_outliers.assign(y = y_pred_outliers)

```

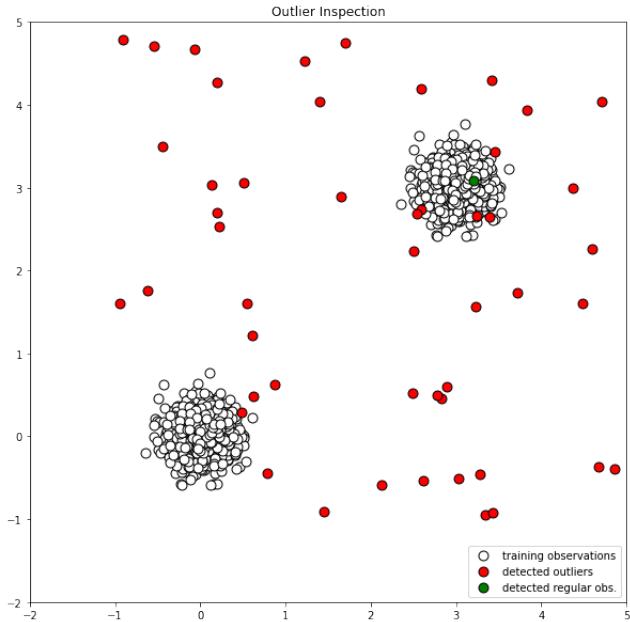
```

plt.figure(figsize=(10,10))
plt.title("Outlier Inspection")

p1 = plt.scatter(X_train.x1, X_train.x2, c='white', s=20*4, edgecolor='k')
p2 = plt.scatter(X_outliers_pred.loc[X_outliers_pred.y == -1, ['x1']], 
                  X_outliers_pred.loc[X_outliers_pred.y == -1, ['x2']], 
                  c='red', s=20*4, edgecolor='k')
p3 = plt.scatter(X_outliers_pred.loc[X_outliers_pred.y == 1, ['x1']], 
                  X_outliers_pred.loc[X_outliers_pred.y == 1, ['x2']], 
                  c='green', s=20*4, edgecolor='k')

plt.axis('tight')
plt.xlim((-2, 5))
plt.ylim((-2, 5))
plt.legend([p1, p2, p3], ["training observations", "detected outliers", "detected regular obs."])
plt.show()

```



Почти все аномалии нашли, но некоторые объекты тоже называли аномалиями.

### 35.2.6. Разделяющие поверхность для разных алгоритмов

Давайте посмотрим на то, какие разделяющие поверхности строят разные алгоритмы. Возьмём эти алгоритмы:

```
from sklearn.covariance import EllipticEnvelope
from sklearn.ensemble import IsolationForest
```

```
import matplotlib
from matplotlib import rc
from scipy import stats
```

Сгенерируем данные

```
n = 100 # size of sample
n_out = 5 # count of outliers
```

```
from sklearn.datasets import make_blobs
```

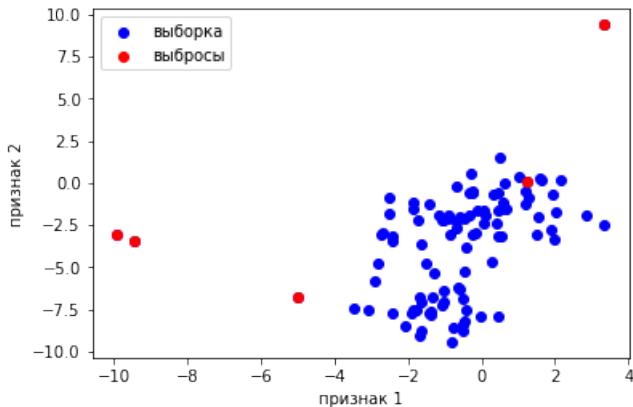
```
X = make_blobs(n_samples=n, n_features=2, centers=3, random_state=2, center_box=(-8.0, 8.0))
```

```

X[:n_out,:] = 20*np.random.rand(n_out, 2) - 10
y = np.ones(n)
y[:n_out] = -1

plt.scatter(X[:,0], X[:,1], c='#0000FF', label='sample')
plt.scatter(X[:n_out,0], X[:n_out,1], c='#FF0000', label='outliers')
plt.xlabel(u'признак 1')
plt.ylabel('признак 2')
plt.legend();

```



Напишем вспомогательную функцию для отрисовки областей

```

xx, yy = np.meshgrid(np.linspace(-10, 10, 500), np.linspace(-10, 10, 500))
outliers_fraction = 0.1

def run_and_plot(clf, X, outliers_fraction, draw_legend=True, title=''):
    clf.fit(X)
    print(clf)

    a_prob = clf.decision_function(X)
    threshold = stats.scoreatpercentile(a_prob, 100 * outliers_fraction)

    print('error=' + str((clf.predict(X) != y).mean()))

    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(8, 8))
    plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), Z.max(), 20), cmap=plt.cm.binary)
    a_ = plt.contour(xx, yy, Z, levels=[threshold], linewidths=1, colors='yellow')
    b_ = plt.scatter(X[y>0, 0], X[y>0, 1], c='white')
    c_ = plt.scatter(X[y<0, 0], X[y<0, 1], c='red')
    plt.axis('tight')
    if draw_legend:
        plt.legend([
            [a_.collections[0], b_, c_],
            [u'split_spase', u'normal_objects', u'outliers'],
            prop=matplotlib.font_manager.FontProperties(size=11), loc='upper_right'])
    plt.title(title)

```

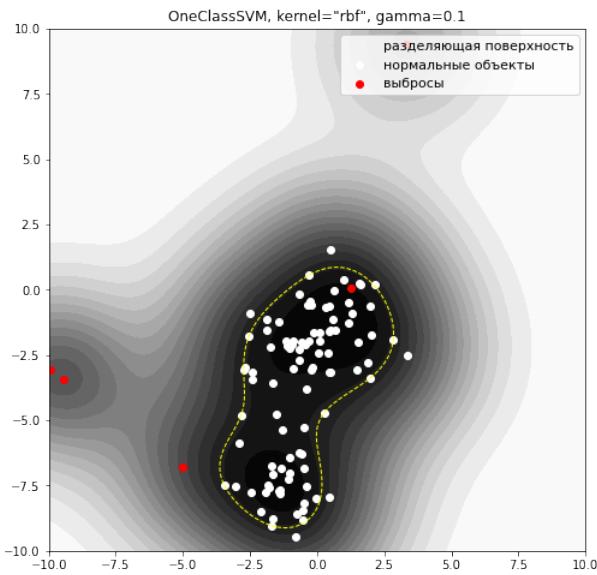
### 35.2.6.1. OneClassSVM .

```

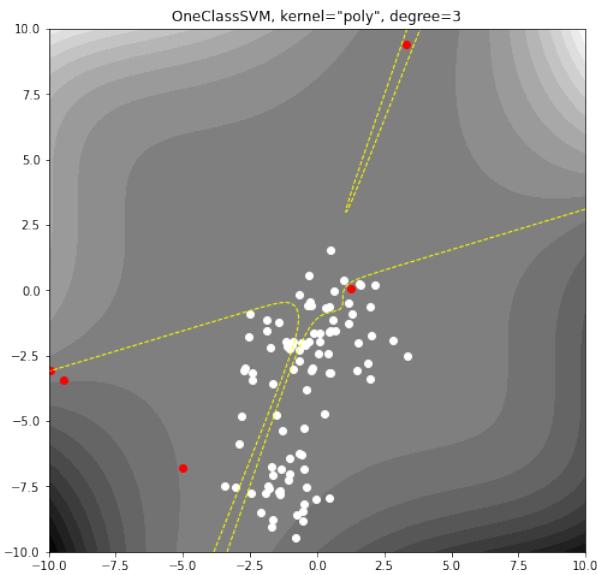
clf = svm.OneClassSVM(nu=0.95 * outliers_fraction + 0.05, kernel="rbf", gamma=0.1)

run_and_plot(clf, X, outliers_fraction=outliers_fraction, draw_legend=True, title='OneClass')

```



```
clf = svm.OneClassSVM(nu=0.95 * outliers_fraction + 0.05, kernel="poly", degree=5)
run_and_plot(clf, X, outliers_fraction=outliers_fraction, draw_legend=False, title='OneClassSVM, kernel="poly", degree=5')
```

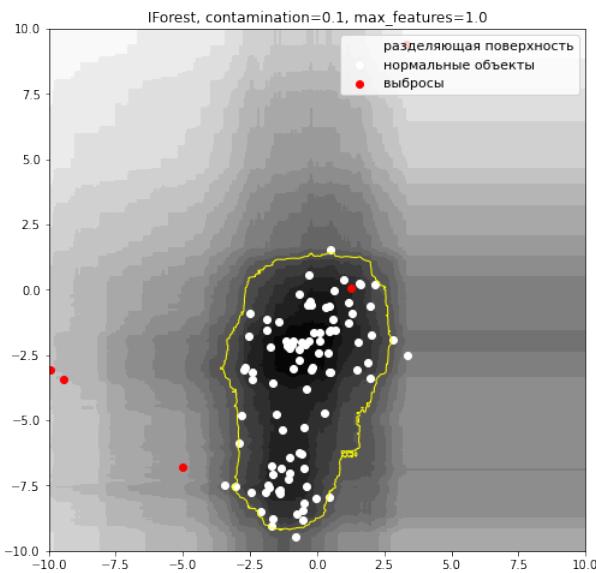


Вот как раз здесь наглядно видно разницу использования ядер в SVM.

### 35.2.6.2. Isolation Forest .

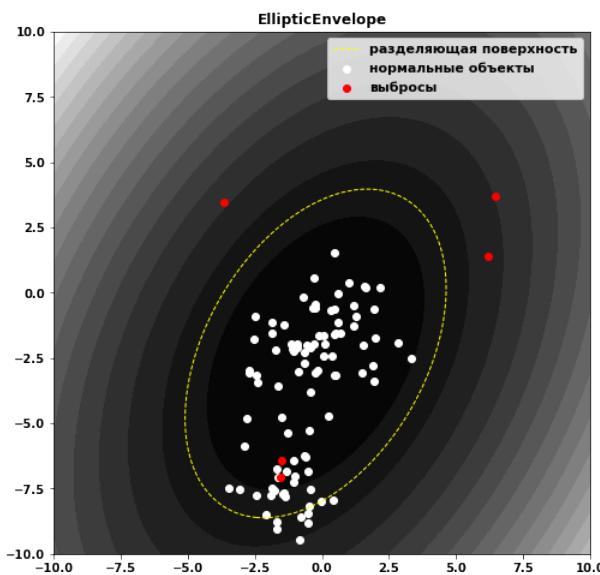
```
clf = IsolationForest(n_estimators=n, max_samples='auto', contamination=0.1, max_features=bootstrap=False, n_jobs=1, random_state=None, verbose=0)

run_and_plot(clf, X, outliers_fraction=outliers_fraction, draw_legend=True, title='IForest')
```



### 35.2.6.3. Elliptic Envelope .

```
clf = EllipticEnvelope(store_precision=True, assume_centered=False, support_fraction=None,
run_and_plot(clf, X, outliers_fraction=outliers_fraction, draw_legend=True, title='Elliptic
```



## 35.2.7. Библиотека PyOD

### 35.2.7.1. Имортим библиотеки .

```
from __future__ import division
from __future__ import print_function

import os
import sys
from time import time

import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
```

```

from sklearn.model_selection import train_test_split
from scipy.io import loadmat

from pyod.models.abod import ABOD
from pyod.models.clof import CBLOF
# from pyod.models.feature_bagging import FeatureBagging
from pyod.models.hbos import HBOS
from pyod.models.iforest import IForest
from pyod.models.knn import KNN
from pyod.models.lof import LOF
from pyod.models.mcd import MCD
from pyod.models.ocsvm import OCSVM
from pyod.models.pca import PCA

from pyod.utils.utility import standardizer
from pyod.utils.utility import precision_n_scores
from sklearn.metrics import roc_auc_score

```

### 35.2.7.2. Сгенерируем набор точек для задачи поиска выбросов .

```

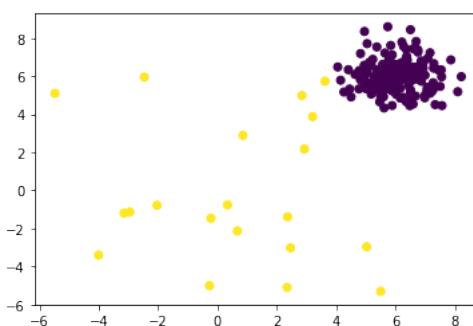
from pyod.utils.data import generate_data
from matplotlib import pylab as plt
%pylab inline

contamination = 0.1 # percentage of outliers
n_train = 200 # number of training points
n_test = 100 # number of testing points

# Generate sample data
X_train, X_test, y_train, y_test = \
    generate_data(n_train=n_train,
                  n_test=n_test,
                  n_features=2,
                  contamination=contamination,
                  random_state=42)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.show()

```



### 35.2.7.3. Применим KNN для нахождения выбросов .

```

from pyod.models.knn import KNN # kNN detector

# train kNN detector
clf_name = 'KNN'
clf = KNN()
clf.fit(X_train)

# get the prediction label and outlier scores of the training data

```

```

y_train_pred = clf.labels_ # binary labels (0: inliers, 1: outliers)
y_train_scores = clf.decision_scores_ # raw outlier scores

# get the prediction on the test data
y_test_pred = clf.predict(X_test) # outlier labels (0 or 1)
y_test_scores = clf.decision_function(X_test) # outlier scores

```

#### 35.2.7.4. Посмотрим на качество алгоритма .

```

from pyod.utils.data import evaluate_print

# evaluate and print the results
print("\nOn_Training_Data:")
evaluate_print(clf_name, y_train, y_train_scores)
print("\nOn_Test_Data:")
evaluate_print(clf_name, y_test, y_test_scores)

```

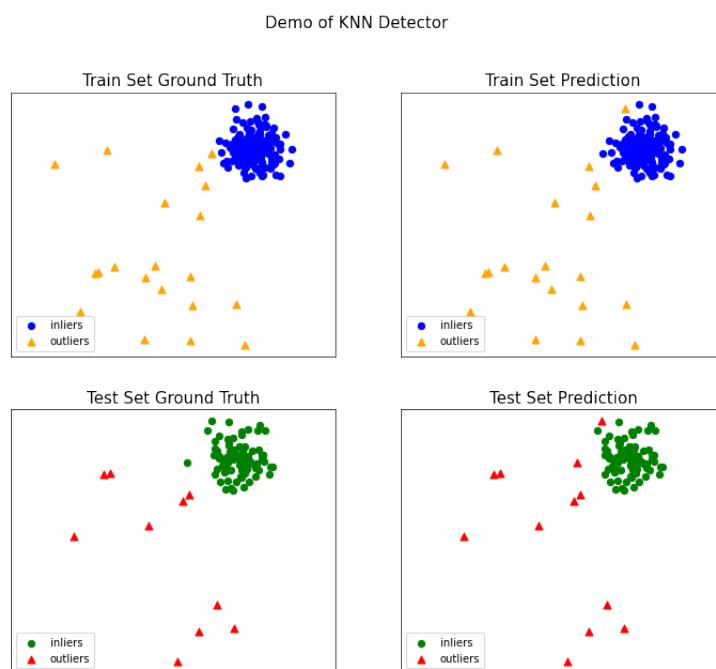
#### 35.2.7.5. Визуализируем результат .

```

from pyod.utils.example import visualize

visualize(clf_name, X_train, y_train, X_test, y_test, y_train_pred,
          y_test_pred, show_figure=True, save_figure=False)

```



#### 35.2.7.6. Сравним различные алгоритмы для нахождения выбросов по времени работы и качеству .

Загрузим данные и информацию о проценте выбросов

```

mat_file = 'arrhythmia.mat'

random_state = np.random.RandomState(42)

df_columns = [ 'Characteristics', 'Data', '#Samples', '#Dimensions', 'Outlier_Perc',
               'ABOD', 'CBLOF', 'HBOS', 'IForest', 'KNN', 'LOF', 'MCD',
               'OCSVM', 'PCA' ]

mat = loadmat(mat_file)

```

```

X = mat[ 'X' ]
y = mat[ 'y' ].ravel()
outliers_fraction = np.count_nonzero(y) / len(y)
outliers_percentage = round(outliers_fraction * 100, ndigits=4)

# construct containers for saving results
roc_list = [ 'ROC-AUC', mat_file[:-4], X.shape[0], X.shape[1], outliers_percentage ]
prn_list = [ 'Precision@N', mat_file[:-4], X.shape[0], X.shape[1], outliers_percentage ]
time_list = [ 'Time', mat_file[:-4], X.shape[0], X.shape[1], outliers_percentage ]

Разобъем данные на тренировочную и валидационную части и масштабируем их.

# 60% data for training and 40% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=random_state)

# standardizing data for processing
X_train_norm, X_test_norm = standardizer(X_train, X_test)

Применим различные алгоритмы нахождения выбросов

from time import time

classifiers = { 'Angle-based_Outlier_Detector_(ABOD)': ABOD(
    contamination=outliers_fraction),
    'Cluster-based_Local_Outlier_Factor': CBLOF(
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    # 'Feature Bagging': FeatureBagging(contamination=outliers_fraction,
    #                                     random_state=random_state),
    'Histogram-base_Outlier_Detection_(HBOS)': HBOS(
        contamination=outliers_fraction),
    'Isolation_Forest': IForest(contamination=outliers_fraction,
                                 random_state=random_state),
    'K_Nearest_Neighbors_(KNN)': KNN(contamination=outliers_fraction),
    'Local_Outlier_Factor_(LOF)': LOF(
        contamination=outliers_fraction),
    'Minimum_Covariance_Determinant_(MCD)': MCD(
        contamination=outliers_fraction, random_state=random_state),
    'One-class_SVM_(OCSVM)': OCSVM(contamination=outliers_fraction),
    'Principal_Component_Analysis_(PCA)': PCA(
        contamination=outliers_fraction, random_state=random_state),
}

for clf_name, clf in classifiers.items():
    t0 = time()
    clf.fit(X_train_norm)
    test_scores = clf.decision_function(X_test_norm)
    t1 = time()
    duration = round(t1 - t0, ndigits=4)
    time_list.append(duration)

    roc = round(roc_auc_score(y_test, test_scores), ndigits=4)
    prn = round(precision_n_scores(y_test, test_scores), ndigits=4)

    print('{clf_name}_ROC:{roc}, precision_{@rank_n}:{prn}, '
          'execution_time:{duration}'.format(
              clf_name=clf_name, roc=roc, prn=prn, duration=duration))

    roc_list.append(roc)
    prn_list.append(prn)

```

```

df_time = pd.DataFrame([time_list], columns=df_columns)
df_roc = pd.DataFrame([roc_list], columns=df_columns)
df_prn = pd.DataFrame([prn_list], columns=df_columns)

df_res = pd.concat([df_time, df_roc, df_prn], axis=0)

```

### 35.2.7.7. Попробуем определить долю выбросов в датасете .

В этом случае - у нас нет разметки. Что делать?

```

data = pd.read_csv("heart.csv")
X = data.copy()
y = data['target']
del X['target']
print(data.shape)

```

```

Q1 = X.quantile(0.25)
Q3 = X.quantile(0.75)
IQR = Q3 - Q1
print(IQR)

```

```

X_wo_out = X[~((X < (Q1 - 3 * IQR)) | (X > (Q3 + 3 * IQR))).any(axis=1)] #search big outliers
y_wo_out = y[~((X < (Q1 - 3 * IQR)) | (X > (Q3 + 3 * IQR))).any(axis=1)]

```

%%time

```

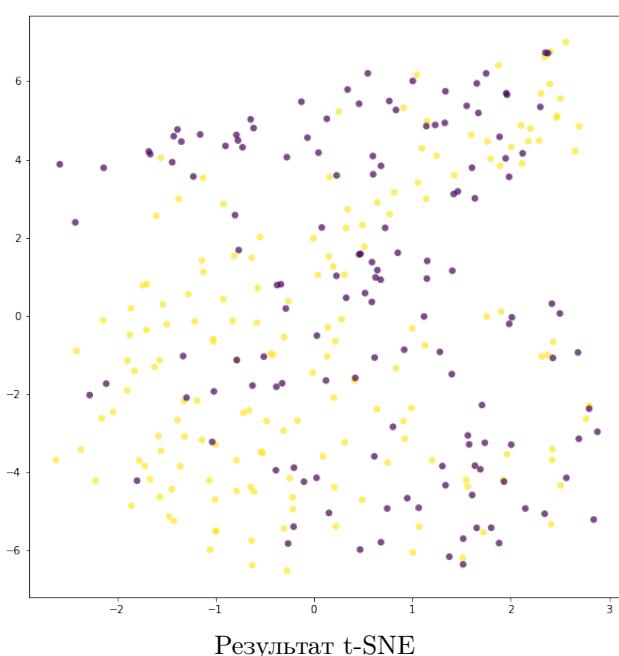
from sklearn.manifold import TSNE

tsne = TSNE(perplexity=100)

X_tsne = tsne.fit_transform(X)

plt.figure(figsize=(10,10))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, edgecolor='none', alpha=0.7, s=40)
plt.show()

```



А теперь попробуем с помощью наших моделей оценить - если качество улучшилось - то мы убрали выбросы и должно стать получше:

```

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

cross_val_score(LogisticRegression(), X, y, cv=3).mean() # on all data

cross_val_score(LogisticRegression(), X_wo_out, y_wo_out, cv=3).mean() # drop outliers by

outliers_fraction = 0.15

classifiers = { 'Angle-based_Outlier_Detector_(ABOD)': ABOD(
    contamination=outliers_fraction),
    'Cluster-based_Local_Outlier_Factor': CBLOF(
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    # 'Feature Bagging': FeatureBagging(contamination=outliers_fraction,
    #                                     random_state=random_state),
    'Histogram-base_Outlier_Detection_(HBOS)': HBOS(
        contamination=outliers_fraction),
    'Isolation_Forest': IForest(contamination=outliers_fraction,
                                  random_state=random_state),
    'K_Nearest_Neighbors_(KNN)': KNN(contamination=outliers_fraction),
    'Local_Outlier_Factor_(LOF)': LOF(
        contamination=outliers_fraction),
    'Minimum_Covariance_Determinant_(MCD)': MCD(
        contamination=outliers_fraction, random_state=random_state),
    'One-class_SVM_(OCSVM)': OCSVM(contamination=outliers_fraction),
    'Principal_Component_Analysis_(PCA)': PCA(
        contamination=outliers_fraction, random_state=random_state),
}

for clf_name, clf in classifiers.items():
    clf.fit(X)
    y_pred = clf.labels_

    X_wo = X[y_pred==0]
    y_wo = y[y_pred==0]

    print(clf_name, cross_val_score(LogisticRegression(), X_wo, y_wo, cv=3).mean())

```

36. Лекция 16. Прогнозирование временных рядов. Наша лекция