МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра информатики и программирования

ПРОБЛЕМА ВЫБОРА МЕТРИКИ ДЛЯ ОПРЕДЕЛЕНИЯ СТЕПЕНИ ПОХОЖЕСТИ МЕЖДУ ОБЪЕКТАМИ РАЗЛИЧНОЙ ПРИРОДЫ

КУРСОВАЯ РАБОТА

студента 2 курса 241 группы
направления 02.03.03 — Математическое обеспечение и администрирование
информационных систем
факультета КНиИТ
Салыгина Андрея Юрьевича

Научный руководитель
Зав. кафедрой ИиП, к. ф.-м. н, доцент _____ М. В. Огнева

Заведующий кафедрой ИиП
к. ф.-м. н, доцент _____ М. В. Огнева

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С ростом объема данных, необходимых для анализа и обработки, становится все более важным использование методов кластеризации и классификации. Для эффективного применения этих методов требуется определить понятие метрического пространства, которое включает в себя функции расстояния или метрики, определяющие взаимное расположение объектов в данном пространстве.

Существует множество метрик для различных типов данных, поэтому важно понимать, как рассчитывать разные виды метрик и понимать различия между ними, чтобы правильно оценивать результаты, получаемые при анализе данных.

Ошибки в выборе метрик могут привести к неправильной классификации объектов и неверным выводам, поэтому важно учитывать этот аспект при работе с большими объемами данных.

Цель курсовой работы заключается в реализации сравнительного анализа метрик для данных различной природы.

В частности необходимо решить следующие задачи:

- изучить существующие метрики;
- рассмотреть области применения каждой из метрик;
- исследовать их эффективность на различных наборах данных.

Для исследования будет использован язык Python, в частности, библиотеки:

- sciPy — вычисление метрик;
- scikit-learn — применение метрик в соответствующих методах обработки данных;
- numpy и pandas — очистка и преобразование данных;
- statsmodels — метод линейной регрессии для расстояния Кука;
- nltk и re — обработка текста;
- matplotlib и seaborn — визуализация данных;
- imblearn — метод передискретизации SMOTE;
- Levenshtein и fastDamerauLevenshtein — расстояния Левенштейна и Дамерау-Левенштейна соответственно.

1 Теоретическая часть

Метрикой на множестве X называется отображение d: $X \times X \to \mathbb{R}$ сопоставляющее каждой паре $(x, y) \in X \times X$ вещественное число $d(x, y)$, удовлетворяющее следующим условиям:

— неотрицательность: $d(x, y) \geq 0$ для любых (x, y);

— $d(x, y) = 0$ тогда и только тогда, когда $x = y$;

— симметричность: $d(x, y) = d(y, x)$;

— неравенство треугольника: $d(x, y) \leq d(x, z) + d(z, y)\ \forall x, y, z \in X$.

Множество X вместе с отображением d называется метрическим пространством, и обозначается (X, d).

Метрика является обобщением понятия расстояния на произвольные пространства. Всякое пространство может быть наделено метрикой [1].

1.1 L_p нормы

1.1.1 Расстояние Минковского

Расстояние Минковского (L_p норма) — это обобщение евклидовой, манхэттенской и чебышевской метрик расстояния. Соответственно справедлива формула:

$$d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{1}$$

Расстояние Минковского является метрикой только при $p \geq 1$ в силу неравенства Минковского. При $0 < p < 1$ не является в силу доказательства [2].

При этом при $0 < p < 1$ можно наглядно показать, что это не метрика, с помощью контрпримера:

Пусть $x := (0, 0)$, $y := (1, 1)$, $z := (0, 1)$, тогда:

— $d(x, y) = 2^{1/p}$, которое при $p < 1$ больше 2;

— $d(x, z) = d(y, z) = 1$.

Поэтому $d(x, y) > d(x, z) + d(z, y) \Rightarrow$ расстояние Минковского не является метриков при $0 < p < 1$.

Если $p < 0 \Rightarrow d(x, x) \neq 0 \Rightarrow$ расстояние Минковского не является метрикой при $p < 0$.

Следовательно для того чтобы получить вышеупомянутые метрики:

— Манхэттенское расстояние при $p = 1$;

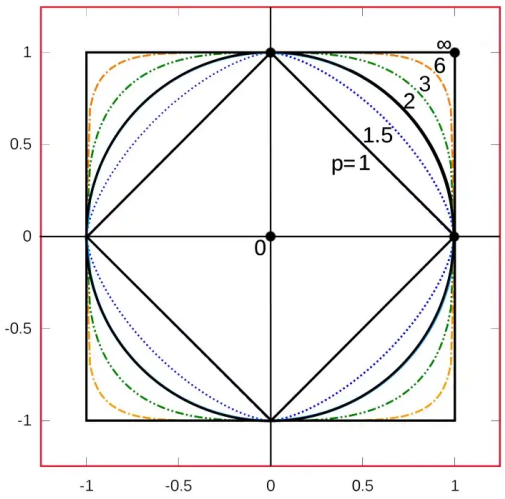

Рисунок 1 – Контуры Минковского для различных значений p.

— Евклидово расстояние при $p = 2$;
— Расстояние Чебышева при $p \to +\infty$ (максимальное расстояние);
— Минимальное расстояние при $p \to -\infty$ (минимальное расстояние) [3].

В качестве примера для L_p норм будет подсчитываться расстояние между средними значениями классов setosa и versicolor по длине чашелистика и длине лепестка датасета ирисов.

Среднее setosa (x): $(5.01, 1.46)$, versicolor (y): $(5.936, 4.26)$

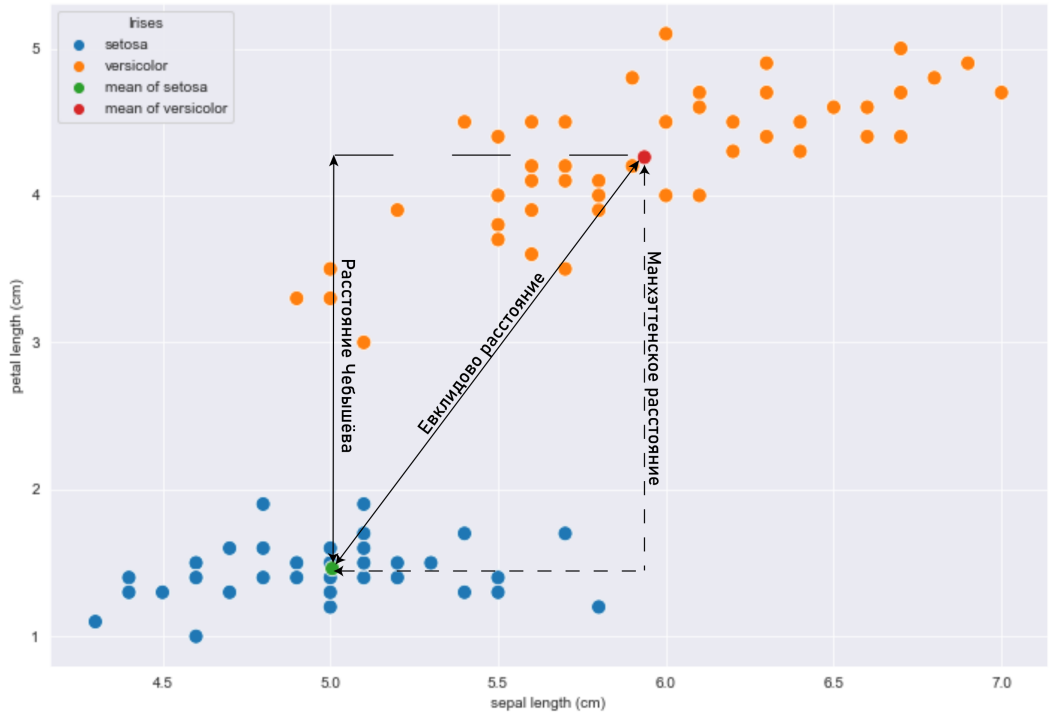

Рисунок 2 – Расстояние между средними значениями ирисов по длине чашелистика и длине лепестка

1.1.2 Евклидово расстояние

Евклидово расстояние (L2 норма) является одной из самых расспространенных метрик для числовых признаков, но не всегда является самой эффективной. Вычисляется по формуле:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

Вычисленные расстояния могут быть искажены в зависимости от единиц измерения признаков, следовательно перед этим необходимо нормализовать данные.

Для нормализации данных можно использовать Min-Max Normalization или Z –Score Normalization.

Методы kNN и HDBSCAN с евклидовым расстоянием показывает хорошие результаты на низкоразмерных данных.

По мере увеличения размерности данных евклидово расстояние становится менее полезным. Это связано с "проклятием размерности которое связано с представлением о том, что многомерное пространство не ведет себя так, как интуитивно ожидается от 2- или 3-мерного пространства.

Несмотря на другие меры, которые учитывают недостатки евклидова расстояния, оно остается одной из наиболее часто используемых мер расстояния, так как интуитивно понятно в использовании, простая в реализации и показывает отличные результаты использования [4].

Пример для ирисов: $d(x, y) = \sqrt{(5.01 - 5.936)^2 + (1.46 - 4.26)^2} \approx 2.94$

1.1.3 Манхэттенское расстояние

Манхэттенское расстояние (L1 норма) — это метрика между двумя точками в N-мерном пространстве, являющейся суммой абсолютной разницы между измерениями двух объектов по всем измерениям [5].

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3}$$

Манхэттенское расстояние хорошо работает для многомерных данных, но при этом менее интуитивно понятна, чем евклидово расстояние, что заметно на многомерных данных.

При этом нужно заметить, что в большинстве случаев оно даст более высокое значение расстояния, чем евклидова метрика, так как манхэттенское расстояние не является кратчайшим из возможных путей.

Манхэттенское расстояние хорошо показывает себя на дискретных и бинарных данных.

Также оно предпочтительнее евклидова, если данные содержат много выбросов [4].

Пример для ирисов: $d(x, y) = |5.01 - 5.936| + |1.46 - 4.26| = 3.726$

1.1.4 Расстояние Чебышева

Расстояние Чебышева (L∞) —— это метрика, которая является максимальным абсолютным расстоянием в одном измерении двух N-мерных точек [6].

$$d(x, y) = \max_i |x_i - y_i| \tag{4}$$

При этом расстояние Чебышева является метрикой, при этом стоит заметить, что если поменять функцию $\max$ на $\min$, то метрикой это расстояние уже являться не будет, в силу того, что не выполняется необходимость условия $d(x, y) = 0 \Leftrightarrow x = y$, так как существует контрпример:

Если $A(5, 1); B(1, 1) \Rightarrow d(A, B) = \min(|x_1 - x_2|, |y_1 - y_2|) = \min(|5 - 1|, 0) = 0 \Rightarrow d(x, y) = 0 \not\Leftrightarrow x = y$

Расстояние Чебышева может подойти под данные цен ценных бумаг, так как позволяет определить какая из бумаг обладает большей разницей между прибылью и потерями [3].

Пример для ирисов: $d(x, y) = max(|5.01 - 5.936|, |1.46 - 4.26|) \approx 2.8$

1.2 Канберрское расстояние

Канберрское расстояние — это взвешенная версия манхэттенского расстояния [7].

$$d(x, y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|} \tag{5}$$

Пример для ирисов: $d(x, y) = \frac{|5.01-5.936|}{|5.01|+|5.936|} + \frac{|1.46-4.26|}{|1.46|+|4.26|} \approx 0.57$

1.3 Косинусное расстояние

Косинусное расстояние — это косинус угла между двумя векторами, вычтенный из 1. Оно используется как способ решения проблемы "проклятия размерности"евклидова расстояния. Два вектора с одинаковой ориентацией имеют его равным 0, а два диаметрально противоположных друг другу 2.

$$d(x,y) = 1 - cos(\widehat{xy}) = \frac{x \cdot y}{\|x\|\|y\|} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \quad (6)$$

Косинусовое расстояние используют, когда имеются многомерные данные и не важна величина векторов. Часто тексты анализируются именно с помощью этой меры, когда данные представлены количеством слов, например: при сравнении двух текстов может оказаться, что в одном какое-то слово встречается чаще, чем в другом тексте, при этом это не означает, что первый текст больше связан с этим словом, так как тексты могут иметь различную длину, а следовательно для того, чтобы не учитывать это, можно использовать косинусное расстояние. Также эта мера может быть использована для отслеживание спама.

При этом стоит отметить, что величина данной меры не говорит о том насколько тот или иной объект похож на другой. Из значения близкого к 0 можно лишь сказать о том, что объекты очень похожи, но на сколько процентов неизвестно [3, 4].

Возвращаясь к преимуществам стоит отметить, что косинусное расстояние неплохо показывает себя на данных, которые имеют огромное число пропусков [8].

Пример вычисления косинусного расстояния на данных об оценке фильмов двух человек:

	Форрест Гамп	Бойцовский клуб
Иван	10	3
Петр	1	10
Яна	3	9

Рисунок 3 – Оценки двух человек по просмотренным фильмам

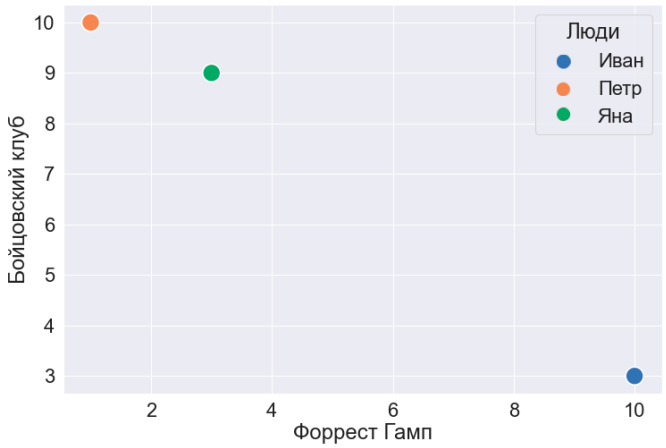

Рисунок 4 – График на основе данных об оценках

Тогда косинусное расстояние вычисляется:

$d(\texttt{Петр},\ \texttt{Иван}) = 1 - \frac{1*10+10*3}{\sqrt{1^2+10^2}\cdot\sqrt{10^2+3^2}} \approx 0.619$

$d(\texttt{Петр},\ \texttt{Яна}) = 1 - \frac{1*3+10*9}{\sqrt{1^2+10^2}\cdot\sqrt{3^2+9^2}} \approx 0.025$

$d(\texttt{Иван},\ \texttt{Яна}) = 1 - \frac{10*3+3*9}{\sqrt{10^2+3^2}\cdot\sqrt{3^2+9^2}} \approx 0.425$

1.4 Метрики для строк

1.4.1 Расстояние Хэмминга

Расстояние Хэмминга — это количество символов или позиций двух строк, на которых соответствующие им символы различны. Оно используется для сравнения двух строк одинаковой длины.

$$H(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{7}$$

Эта метрика является манхэттенское расстоянием для категориальных данных.

Примеры вычисления:

$H(111011, 100001) = 3$

$H(100, 111) = 2$

Если одно из сообщений — это все нули, то тогда расстояние Хэмминга называется весом Хэмминга, и оно равно количеству единиц в ненулевом сообщении [3]:

$H(110011, 000000) = W(110011) = 4$

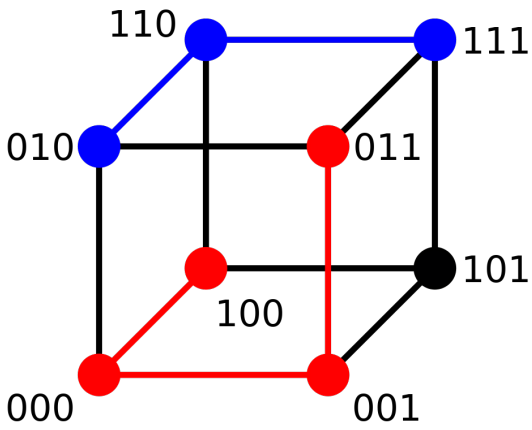

Рисунок 5 – Пример 3-битового расстояния Хэмминга.

В более общем случае расстояние Хэмминга применяется для строк одинаковой длины любых k-ичных алфавитов и служит метрикой различия объектов одинаковой размерности, пример [9]:

$H(\texttt{medication, meditation}) = 1$

1.4.2 Расстояние Левенштейна

Расстояние Левенштейна —— это метрика предназначенная для двух строк, которая измеряет разницу между ними, а именно минимальное количество односимвольных исправлений (вставки, удаления или замены), необходимых для замены одного слова другим. Эта метрика вычисляется по следующей формуле [10]:

$$lev_{x,y}(i,j) = \min \begin{cases} 0, & \texttt{если } i = j = 0, \\ lev_{x,y}(i-1,j) + 1, & \texttt{если } i > 0, \\ lev_{x,y}(i,j-1) + 1, & \texttt{если } j > 0, \\ lev_{x,y}(i-1,j-1) + 1_{(x_i \neq y_j)}, & \texttt{если } i,j > 0 \end{cases} \tag{8}$$

Есть два алгоритма вычисления данной метрики: Вагнера — Фишера и Хиршберга, рассмотрим только первый [11, 12].

Пример вычисления расстояния Левенштейна методом Вагнера — Фишера для слов "вода"и "еда":

$lev_{x,y}(0,0) = 0$

$lev_{x,y}(1,0) = lev_{x,y}(0,0) + 1 = 1$

$lev_{x,y}(2,0) = lev_{x,y}(1,0) + 1 = 2$

...

$lev_{x,y}(0,1) = lev_{x,y}(0,0) + 1 = 1$

$lev_{x,y}(0,2) = lev_{x,y}(0,1) + 1 = 2$

...

Для остальных ячеек используются последние три строчки:

Пример для ячейки (1,1): $lev_{x,y}(1,1) = \min(lev_{x,y}(0,0), lev_{x,y}(1,0) + 1, lev_{x,y}(0,1) + 1) = \min(0+1, 1+1, 1+1) = 1$

Таким образом рассматриваются левый, верхний и верхний-левый "сосед" элемента. Если вычислить остальные элементы аналогично, то получится таблица:

		В	О	Д	А
	0	1	2	3	4
Е	1	1	2	3	4
Д	2	2	2	2	3
А	3	3	3	3	2

Соответственно значение последней ячейки (n, m), где n = 3 (еда), а m = 4 (вода) будет искомым расстоянием Левенштейна, т. е $lev_{x,y} = 2$.

Расстояние Левенштейна используют, если при приближенном соответствии строк нужно найти совпадения коротких строк в длинных текстах, где ожидается небольшое количество различий. Это имеет широкий спектр применений, например, средства проверки орфографии, системы распознавания текстов и облегчение перевода текстов на основе предыдущих переводов.

Расстояние Левенштейна можно находить между двумя длинными строками, но затраты на вычисления, которые пропорциональны произведению двух длин строк, делает это непрактичным [10].

1.4.3 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является расстоянием Левеншнтейна с операцией перестановки, определяется следующим образом [13]:

$$d_{x,y}(i,j) = \min \begin{cases} 0, & \text{если } i = j = 0, \\ d_{x,y}(i-1,j) + 1, & \text{если } i > 0, \\ d_{x,y}(i,j-1) + 1, & \text{если } j > 0, \\ d_{x,y}(i-1,j-1) + 1_{(x_i \neq y_j)}, & \text{если } i,j > 0 \\ d_{x,y}(i-2,j-2) + 1, & \text{если } i,j > 1 \text{ и } x_i = y_{j-1} \text{ и } x_{i-1} = y_j \end{cases} \tag{9}$$

Расстояние Дамерау-Левенштейна применяется в нечётком поиске, компьютерной лингвистике, биоинформатике.

Данное расстояние можно вычислить с алгоритмов Вагнера-Фишера, Укконена или Ландау-Майерса и Шмидта [14].

Пример вычисления расстояния Дамерау-Левенштейна методом Вагнера — Фишера для слов "метрика" и "метирка":

Расчёт первых двух строк и столбцов аналогичен вычислению расстояния Левенштейна. Разница прослеживается, начиная с $i = 2, j = 2$, так как с этого элемента начинает выполняться условие 5-ой строки.

Стоит остановиться на моменте, когда алгоритм доходит до элемента (4, 4) — здесь в первый раз появляется единица из четвертого условия. Позже, на элементе (5, 5) выполняется условие 5-ой строки, так как была найдена перестановка букв "И" и "Р". Если продолжить вычислять элементы, то получится следующая таблица:

		М	Е	Т	Р	И	К	А
	0	1	2	3	4	5	6	7
М	1	0	1	2	3	4	5	6
Е	2	1	0	1	2	3	4	5
Т	3	2	1	0	1	2	3	4
И	4	3	2	1	1	2	3	4
Р	5	4	3	2	2	1	2	3
К	6	5	4	3	3	2	1	2
А	7	6	5	4	4	3	2	1

Таким образом: $d(\texttt{метрика}, \texttt{метирка}) = 1$.

1.5 Расстояние Жаккара

Индекс сходства Жаккара вычисляет схожесть между двумя наборами данных (или же это их пересечение деленное на объединение).

$$J(x, y) = \frac{|x \cap y|}{|x \cup y|} = \frac{|x \cap y|}{|x| + |y| - |x \cap y|} \tag{10}$$

Расстояние Жаккара вычисляет различие между двумя наборами данных и вычисляется по следующей формуле:

$$d(x, y) = 1 - J(x, y) \tag{11}$$

Все свойства метрики для расстояния Жаккара элементарно проверяются, кроме неравенства треугольника. Для того, чтобы доказать выполнимость неравенства, следует воспользоваться преобразованием Штейнхауса. Стоит отметить, что расстояние Жаккара является частным случаем расстояния Штейнхауса [15].

Сверточные нейронные сети, которые занимаются распознованием изображений, применяют индекс Жаккара для рассчета точности обнаружения объектов [16].

Индекс Жаккара может быть применим в семантической сегментации, анализе текста, электронной коммерции и системах рекомендаций [3].

Пример: для системы рекомендации новых друзей в социальной сети будем считать, что чем больше общих друзей у двух пользователей, тем больше шанс того, что эта пара может стать друзьями.

Пусть Петр дружит с $x = \{$Витя, Андрей, Илья, Тимур, Вика и Ульяна$\}$, а Яна дружит с $y = \{$Василиса, Вика, Ульяна, Макар и Илья$\}$. Тогда:

$J(x, y) = \frac{3}{8} = 0.375 \Rightarrow d(x, y) = 1 - 0.375 = 0.625$

$\texttt{независимы} \Rightarrow r_{xy} = 0 \not\Leftrightarrow x$ и y $\texttt{независимы}$.

1.6 Расстояние Махаланобиса

Расстояние Махаланобиса — это многомерная метрика расстояния, которая эффективно измеряет расстояние между вектором и распределением. Это расстояние используется для обнаружения аномалий в многомерных данных, классификации очень несбалансированных наборов данных и одноклассовой классификации.

Если признаки коррелируют друг с другом, то евклидово расстояние между наблюдениями или наблюдением и средним распределения может неправильно показывать, насколько близко точка находится к распределению.

Если рассматривать на примере, то точка P1, если считать евклидовое расстояние, ближе, чем точка P2 к среднему значению распределения (точке M). Но если учитывать распределение данных, то это является неправильным выводом, и в таком случае необходимо использовать расстояние Махаланобиса.

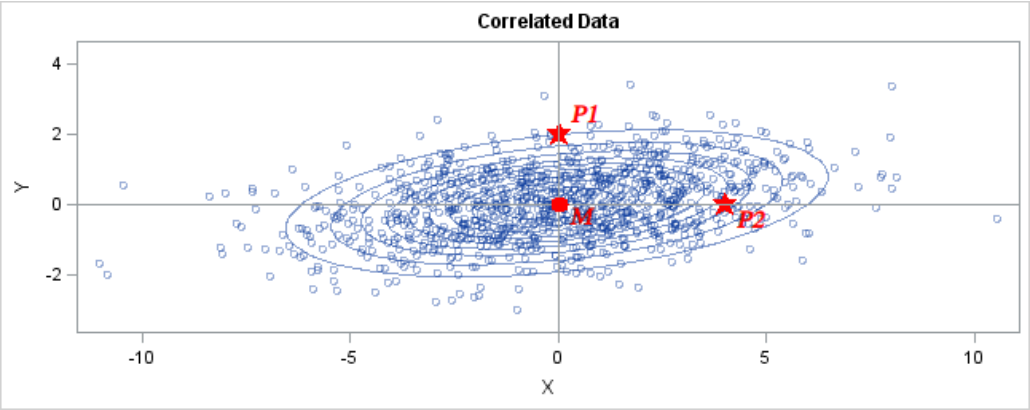

Рисунок 6 – Пример преимущества расстояния Махаланобиса над расстоянием Евклида.

Общая формула:

$$d(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)} \tag{12}$$

Формула для определения расстояния между средним распределения и вектором:

$$d(x,\mu) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)} \tag{13}$$

S^{-1} — обратная ковариационная матрица, с формулой для трёхмерных данных:

$$S^{-1} = \begin{pmatrix} var(x) & cov(x,y) & cov(x,z) \\ cov(y,x) & var(y) & cov(y,z) \\ cov(z,x) & cov(z,y) & var(z) \end{pmatrix}^{-1} \tag{14}$$

При этом согласно свойству симметричности ковариации $\Rightarrow cov(x,y) = cov(y,x)$ и по другим парам аналогично.

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{n} \tag{15}$$

$$var(x) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} \tag{16}$$

Коэффициент растяжения дисперсии вдоль осей (в собственном базисе) — это есть собственное значение, соответствующее собственным векторам ковариационной матрицы (или главным компонентам), то есть $var_i \sim \lambda_i$

Расстояние Махаланобиса, по своему определению, является многомерным вариантом Z-оценки, так как в собственном базисе:

$$S = \begin{pmatrix} \lambda_1 & 0 & 0 & ... \\ 0 & \lambda_2 & 0 & ... \\ ... & ... & ... & .... \\ 0 & 0 & 0 & \lambda_n \end{pmatrix} \Rightarrow S^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & 0 & ... \\ 0 & \frac{1}{\lambda_2} & 0 & ... \\ ... & ... & ... & .... \\ 0 & 0 & 0 & \frac{1}{\lambda_n} \end{pmatrix}$$

Стоит заметить, что расстояние Махаланобиса является частным случаем расстояния Евклида, так как его можно записать в следующем виде:

$$d(x, y) = \sqrt{(x - y)^T E (x - y)}, \tag{17}$$

где E — единичная матрица [17].

1.7 Расстояние Кука

Расстояние Кука используется для оценки влияния наблюдения в регрессионном анализе методом наименьших квадратов. Чем больше расстояние, тем более влиятельной является наблюдение относительно других наблюдений во время регрессионного анализа. То есть больше вероятность того, что оно окажется выбросом, что отрицательно повлияет на производительность модели.

Это расстояние вычисляется путем удаления i-й точки данных из модели и повторного вычисления регрессии. Далее суммируется, насколько изменяются все значения в регрессионной модели при удалении i-го наблюдения. Формула для определения расстояния Кука равна:

$$D_i = \frac{\sum_{j=1}^{n} (Y_j - Y_{j(i)})^2}{(p+1)\sigma^2}, \tag{18}$$

где Y_j — значение наблюдения без удаления j-того признака из модели, $Y_{j(i)}$ — значение наблюдения с удалением j-того признака из модели, p — количество признаков в модели.

Общепринятого правила для отсечения наблюдения нет, но наиболее распространённой считается значение, превышающего $\frac{4}{n}$, где n — количество наблюдений.

1.8 Расстояние Йенсена-Шеннона

Расстояние Йенсена-Шеннона — это квадратный корень из дивергенции Йенсена-Шеннона, которая позволяет определить похожесть двух распределений вероятностей. В основе этой дивергенции лежит дивергенция Кульбака-Лейблера, но при этом в отличии от неё у Йенсена-Шеннона есть симметричность и конечность значения.

$$D_{KL}(p \parallel q) = \sum_{i=1}^{N} p(x_i) \log \left(\frac{p(x_i)}{q(x_i)} \right) \tag{19}$$

$$D_{JS}(p \parallel q) = \frac{D_{KL}(p \parallel \frac{p+q}{2}) + D_{KL}(q \parallel \frac{p+q}{2})}{2} \tag{20}$$

$$d(p, q) = \sqrt{D_{JS}(p \parallel q)} \tag{21}$$

(15) — это дивергенция Кульбака-Лейблера между распределениями p и q, где x_i — это события; (16) — это дивергенция Йенсена-Шеннона; (17) — метрика Йенсена-Шеннона [18, 19].

Йенсон-Шеннон используется для обнаружения дрейфа данных (это одна из причин почему точность модели со временем уменьшается), а также он применяется в GANs (Generative Adversial Networks) в качестве разницы между двумя распределениями: оригинальным и сгенерированным объектом [20].

Пример вычисления расстояния Йенсона-Шеннона для двух распределений: "честной" и "нечестной" монеты. Пусть "нечестная" монета имеет: $p(\mathtt{O}) = 0.95$; $p(\mathtt{P}) = 0.05$, а "честная" монета: $q(\mathtt{O}) = 0.5$; $q(\mathtt{P}) = 0.5$ (O - орёл, P - решка), тогда:

$$D_{KL}(p \parallel \tfrac{p+q}{2}) = p(\mathtt{O}) \ln \left(\frac{p(\mathtt{O})}{\frac{1}{2}(p(\mathtt{O})+q(\mathtt{O}))} \right) + p(\mathtt{P}) \ln \left(\frac{p(\mathtt{P})}{\frac{1}{2}(p(\mathtt{P})+q(\mathtt{P}))} \right) =$$
$$= 0.95 \ln \left(\frac{0.95}{\frac{1}{2}(0.95+0.5)} \right) + 0.05 \ln \left(\frac{0.05}{\frac{1}{2}(0.05+0.5)} \right) = 0.171538$$
$$D_{KL}(q \parallel \tfrac{p+q}{2}) = q(\mathtt{O}) \ln \left(\frac{q(\mathtt{O})}{\frac{1}{2}(p(\mathtt{O})+q(\mathtt{O}))} \right) + q(\mathtt{P}) \ln \left(\frac{q(\mathtt{P})}{\frac{1}{2}(p(\mathtt{P})+q(\mathtt{P}))} \right) =$$
$$= 0.5 \ln \left(\frac{0.5}{\frac{1}{2}(0.95+0.5)} \right) + 0.5 \ln \left(\frac{0.5}{\frac{1}{2}(0.05+0.5)} \right) = 0.113137$$
$$d(p, q) = \sqrt{\frac{0.171538+0.113137}{2}} = 0.377276$$

2 Практическая часть

2.1 Метрики для разреженных данных

В статье [8] было показано, что метод kNN с косинусным расстоянием неплохо справляется на адаптированных для этого расстояния данных. В этой статье были взяты электронные письма компании Enron, и требовалось распределить их на 8 категорий. Утверждается, что точность классификации достигла 76,75% при 48557 признаках из которых ровно 40 не были равны нулю.

Было решено проверить данные результаты путём выбора похожего по устройству датасета, причем дополнительно изучить: какая из метрик будет показывать наибольшую точность модели. Были взяты твиты и предложения различных эмоций, и они были объединены в один датасет размером в 38061 строк (в исходном исследовании использовались 2400 строк). Стоит учитывать, что в собственном датасете присутствуют записи различной длины и датасет содержит довольно необработанные сообщения.

Изначальные датасеты содержали различное количество эмоциональных окрасок, при их объединении методом input_data() получились следующие значения:

- грусть — 11430 строк;
- нейтральное — 8638 строк;
- счастье — 12238 строк;
- злость — 3103 строки;
- страх — 2652 строки.

Стоит отметить, что также методом input_data() из датасета были убраны эмоции: скука, скорбь, восторг, веселье, ненависть, любовь, тревожность и удивление в связи с тем, что количество наблюдений данных классов оказалось достаточно малым. Также это связано с ограниченными ресурсами памяти и вычислительными мощностями.

Метод clearing_data() привёл слова к похожей форме, используя стеммер Портера и лемматизацию, обработка предложений производилась с помощью регулярных выражений:

```
for i in range(0, len(X_concatted)):
    # очистка текста
    tweets = re.sub("@[^ ]+", "", X_concatted[i])
    tweets = re.sub("#[a-zA-Z0-9_]+", "", tweets)
    tweets = re.sub("&quot", "", tweets)
```

```
    tweets = re.sub("https?:\/\/.*?[\s+]", "", tweets)
    tweets = re.sub("[^a-zA-Z]", " ", tweets)
    tweets = tweets.lower()
    tweets = tweets.split()
    # применение стеммера Портера
    tweets_stem = [
        ps.stem(word)
        for word in tweets
        if not word in stopwords.words("english")
    ]
    # лемматизация
    tweets_lem = [
        wnl.lemmatize(word)
        for word in tweets
        if not word in stopwords.words("english")
    ]
```

В частности было выявлено, что точность модели ухудшается при замене стеммера на лемматизацию. Также точность модели измерялась с полным и сокращённом по частотности словаре. При сокращенном словаре, в котором удалены все слова, которые упоминаются только 1 раз, точность модели не изменяется, но при этом многократно увеличивается скорость обучения модели.

Метод data_transformation() создал сокращёный словарь из начального в 21210 слов; сокращённый: 9852 слов. При удалении более 2-ух слов точность модели понижается.

Стоит отметить, что такие аномальные слова как "alwayyyyyyyssssssss" модель не исправляет, и не убирает, оно входит в полный словарь. В сокращенный словарь, который был построен в соответствии с частотностью слов, такое слово с большой долей вероятностью не войдет, так как в точности такие "опечатки" в основном делаются максимум несколько раз за весь датасет.

Были созданы тренировочные и тестовые выборки методом train_test_split(). Коэффициенты для kNN подбирались от 1 до 30, где значение соседей равное 8 показало хороший результат. Нужно отметить, что 8 — это не наилучшее значение для всех расстояний, но оно является наиболее оптимальным. Наилучшее значение не вычислялось в силу большого времени обработки данных. Также данные не были избавлены от возможных выбросов в силу тех же причин.

Далее была произведена нормализация методом Normalize и выравнивание классов методом SMOTE при количестве соседей в 25 (метод ADASYN показывал худшие результаты).

В качестве метрик для kNN были выбраны косинусное, евклидово, манхэттенское, канберрское, чебышевское, корреляционное, квадрат евклидова расстояния и расстояние жаккара.

Таблица 1 — Результаты обучения модели на различных метриках (стеммер)

Метрика	Precision	Recall	F1 score
Euclidean	0.67	0.52	0.50
Squared euclidean	0.67	0.52	0.50
Cosine	0.64	0.63	0.61
Manhattan	0.70	0.51	0.51
Canberra	0.69	0.64	0.64
Chebyshev	0.46	0.39	0.39
Hamming	0.60	0.22	0.11
Jaccard	0.65	0.66	0.65

Таблица 2 — Результаты обучения модели на различных метриках (лемматизация)

Метрика	Precision	Recall	F1 score
Euclidean	0.65	0.47	0.45
Squared euclidean	0.65	0.47	0.45
Cosine	0.62	0.61	0.59
Manhattan	0.66	0.45	0.43
Canberra	0.65	0.57	0.57
Chebyshev	0.42	0.34	0.33
Hamming	0.53	0.21	0.10
Jaccard	0.64	0.65	0.64

Согласно таблицам 1 и 2 наилучшими расстояниями при использовании стеммера Портера и лемматизации оказались расстояние Жаккара, канберрское и косинусное в порядке убывания.

Необходимо учитывать то, что точность модели снизилась из-за нейтрального класса, так как это класс, который довольно сильно пересекается с такими классами как "счастье"и "грусть". Проблема может также возникнуть из-за дрейфа данных, и для проверки на это следует использовать метрику Йенсена-Шеннона.

2.2 Метрики для выявления выбросов в данных

Обычно, для выявления выбросов в данных с маленькой размерностью используют 1.5 интерквартильных расстояния, но минусом такого способа для многомерных данных является то, что его необходимо считать для каждой колонки отдельно, и при этом он должен быть уникален для каждого столбца, что является не очень удобным и занимает много времени.

Недостатком такого способа является и то, что он не может отследить взаимосвязь компонентов, в отличие от расстояния Махаланобиса или расстояния Кука.

Расстояния подобно евклидову не способны правильно оценивать расстояние наблюдения от исходного распределения, так как они не могут учитывать его, следовательно они не предназначены для отслеживания выбросов в данных. Тем не менее для оценки выбросов можно использовать такие расстояния, но только вместе с такими методами, например, как класстеризация DBSCAN.

Поэтому для оценки датасета будут использованы расстояния Махаланобиса, Кука и метод DBSCAN с евклидовым расстоянием.

В качестве датасета были выбраны многомерные данные о домах, которые содержат как количественные, так и категориальные признаки. В изначальном датасете присутствовал 21 признак, но многие из них изначально коррелировали между собой.

Было проведено два исследования, в первом из которых было удалено только 7 признаков из датасета (те, которые не предоставляли для модели значения), а во втором 12. Первый датасет имел между признаками сильную корреляцию, во втором — сильную коррелирующие и схожие по смыслу признаки были удалены.

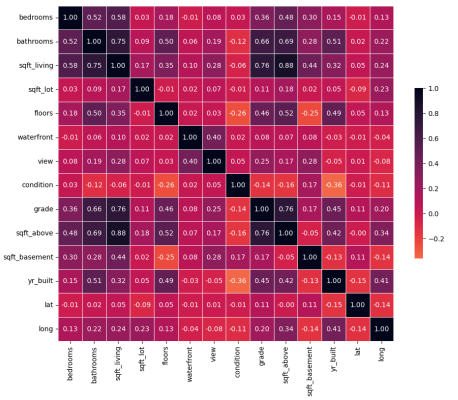

Рисунок 7 – Тепловая карта первого датасета

Рисунок 8 – Тепловая карта второго датасета

2.2.1 Расстояние Махаланобиса

Метод `mahalanobisCleaning()` используется для очистки от выбросов расстоянием Махаланобиса.

Для этого расстояния из формулы (16) необходимо посчитать обратную ковариационную матрицу и среднее по каждому из столбцов.

```
# вычисление ковариационной матрицы
inv_cov_mats = [
    np.linalg.matrix_power(dfs[0].cov(), -1),
    np.linalg.matrix_power(dfs[1].cov(), -1),
]
mean_values = [np.mean(dfs[0], axis=0), np.mean(dfs[1], axis=0)]
mah_dfs = [dfs[0].copy(), dfs[1].copy()]
mah_dfs[0]["mah_dist"] = 0
mah_dfs[1]["mah_dist"] = 0
```

Можно заметить, что для того, чтобы найти выбросы, одного расстояния Махаланобиса недостаточно и необходимо использовать распределение хи-квадрат. Математически доказано, что расстояние Махаланобиса в квадрате соответствует распределению хи-квадрат [21]:

Расчёт расстояния Махаланобиса для каждого наблюдения по всем столбцам:

```
columns = [14, 9]
mah_borders = (
    chi2.ppf(0.99, columns[0]),
    chi2.ppf(0.99, columns[1]),
)
mah_outliers = [pd.DataFrame(), pd.DataFrame()]
# вычисление расстояния
for i in range(0, 2):
    for j, values in mah_dfs[i].iterrows():
        values = values[0 : columns[i]]
        mah_dfs[i].loc[j, "mah_dist"] = (
            mahalanobis(values, mean_values[i], inv_cov_mats[i])
```

```
            ** 2
        )
```

Используя распределение хи-квадрат отсекутся выбросы:

```
mah_outliers[i] = mah_dfs[i][
    mah_dfs[i]["mah_dist"] > mah_borders[i]
].index
```

В результате из датасета было убрано 1984 строки для данных, которые имели сильную корреляцию и 1106 строк для данных без мультиколлинеарности или 9.18% и 5.12% от всего датасета соответственно.

2.2.2 Расстояние Кука

Метод `cooksCleaning()` используется для очистки от выбросов расстоянием Кука. Для нахождения выбросов изначально происходит обучение множественной линейной регрессии:

```
models = [
    sm.regression.linear_model.OLS(np.asarray(target), Xs[0]).fit(),
    sm.regression.linear_model.OLS(np.asarray(target), Xs[1]).fit(),
    ]
```

Скорректированный R-квадрат первой модели: 0.695, второй — 0.468.

Вычисляется влияние каждого наблюдения и записывается в новую колонку:

```
for i in range(0, 2):
        inflce = models[i].get_influence()
        inflce_lsts[i] = inflce.cooks_distance[0]
        cooks_dfs[i]["influence"] = inflce_lsts[i]
```

Устанавливается порог, который определяет, является ли влияние строки данных высоким. В качестве порога используется 4/n, где n —количество наблюдений.

В результате для первого датасета количество удалённых строк составило 1076, а для второго — 1140 или 4.97% и 5.28% от всего датасета соответственно.

2.2.3 DBSCAN

Метод `dbscanCleaning()` используется для очистки от выбросов методом DBSCAN.

Для начала происходит поиск максимального числа выбросов с начальным `eps = 0.5`. После этого начинается подбор `eps` с шагом в 100, чтобы количество выбросов было менее, чем у соответствующего расстояния Махаланобиса:

```
while count_of_outliers[i] > len_of_mah_outliers[i]:
    dbscans[i] = DBSCAN(eps=eps)
    dbscans[i].fit(dfs[i])
    labels_df = pd.DataFrame(dbscans[i].labels_, columns=["labels"])
    labels_df.index = dfs[i].index
    removed_df = labels_df[labels_df["labels"] == -1]
    removed_indexes[i] = removed_df.index
    count_of_outliers[i] = len(removed_indexes[i])
    eps += 100
```

В результате для первого датасета количество удалённых строк составило 1591, а для второго — 888 или 7.36% и 4.1% от всего датасета соответственно

2.2.4 Выводы

На основе вышеприведенных действий можно сделать вывод о том, что расстояние Махаланобиса является эффективным методом для избавления от выбросов на данных с отсутствующей мультиколлинеарностью, но при этом сложно интерпретируем, в отличие от расстояние Кука, в котором можно проследить насколько важно то или иное наблюдение или признак. Стоит отметить, что расстояние Кука работает только если существует линейная зависимость между признаками и целевой переменной, в противном случае оно может быть не таким эффективным.

Таблица 3 — Процент выбросов на различных методах

Метод	Данные с мультиколлинеарностью	Данные без мультиколлинеарности
Махаланобис	9.18%	5.12%
Кук	4.97%	5.28%
DBSCAN (Евклид)	7.36%	4.11%

Оба этих расстояния позволяют определить насколько одно наблюдение далеко от другого, в отличие от метода DBSCAN. При этом его сложно оптимизировать для конкретных наборов данных, потому что подбор ϵ и минимального размера выборки может занять много времени и вычислительной мощности. Также этот метод может быть проблемным, так как он не учитывает распределение данных, что может привести к не таким точным результатам. В силу этого на других расстояниях исследование не проводилось. К достоинствам данного метода относится то, что он достаточно простой и понятный в реализации.

2.3 Метрики для строк

Расстояние Левенштейна и Дамерау-Левенштейна было использовано для исправления пользовательского ввода названий фильмов, для чего был выбран соответствующий датасет с 10000 наблюдениями.

Для того, чтобы расстояние Левенштейна работало более быстро и корректно, оно начинает вычисляться для всех названий из датасета, как только размер входного названия превышает 3 символа, и для фильмов, которые содержат введенное название (без исправлений), если его размер не более трёх символов.

Это сделано потому что в датасете присутствуют фильмы, которые имеют очень маленькое название, следовательно данные расстояния будут иметь маленькое значение и соответствующий неверный фильм будет найден. Число "3" было выбрано так как фильмов с данным количеством букв в названии относительно немного, в сравнение к исходному датасету (1 процент), к тому же вероятность получить опечатку на слове такой длины небольшая.

Для расстояний Левенштейна и Дамерау-Левенштейна используются методы `search_correction_lev()` и `search_correction_dam_lev()` соответственно. Эти методы идентичны за исключением использованного расстояния, они возвращают пять наиболее близких к входному названию фильмов.

```
def search_correction_lev(input_title, input_df):
    lev_distances = np.array([])
    if len(input_title) <= 3:
        # оставляем все те строки, в которых есть входная строка
        df = input_df.loc[input_df["title"].str.contains(input_title, case=False)]
        # считаем расстояния между ними
        for title in df["title"]:
            lev_distances = np.append(lev_distances, distance(input_title, title))
        df["distances"] = lev_distances
```

```
        # сортируем, с наименьшим расстоянием возвращаем
        df = df.sort_values(by="distances")
        return df["title"].iloc[0:5]
    else:
        for title in input_df["title"]:
            lev_distances = np.append(lev_distances, distance(input_title, title))
        input_df["distances"] = lev_distances
        input_df = input_df.sort_values(by="distances")
        return input_df["title"].iloc[0:5]
```

Например, если необходимо найти фильм "alien" при входной строке "Aline" , то расстояние Левенштейна не справится с этой задачей и найдёт только: "alive" , "alice" , "mine" и "life" . Расстояние Дамерау-Левенштейна найдёт этот фильм и выдаст его как наиболее подходящий в силу того, что оно обладает операцией перестановки: "alien" , "alice" , "alive" и "life" .

Выбор того или иного расстояния зависит от конкретных целей.

2.4 Метрики для смешанных данных

2.4.1 Предобработка данных

В качестве смешанных данных (имеющие численные и категориальные признаки) были взяты данные из статьи [22], которая посвящена проблеме завершения студентами обучения в высших учебных заведений.

Из исходного датасета были убраны все люди, которые имели тип Enrolled, и были оставлены только Dropout и Graduate.

Также из датасета были убраны колонки 4 колонки, которые отвечали за количество учебных единиц и которые коррелировали с другими учебными единицами.

Для изображения полученных класстеров использовался метод PCA.

Так как изначальное число наблюдений в классах было различно (2209 и 1421), то был применём метод SMOTE с количеством соседей равным 3. Для генерации данных применялся метод ADASYN, но он оставался таким же по эффективности или был хуже, чем SMOTE.

```
X_resampled_train, y_resampled_train = SMOTE(
    k_neighbors=3, random_state=41
).fit_resample(X_train, y_train)
X_resampled_test, y_resampled_test = SMOTE(
```

```
    k_neighbors=3, random_state=41
).fit_resample(X_test, y_test)
```

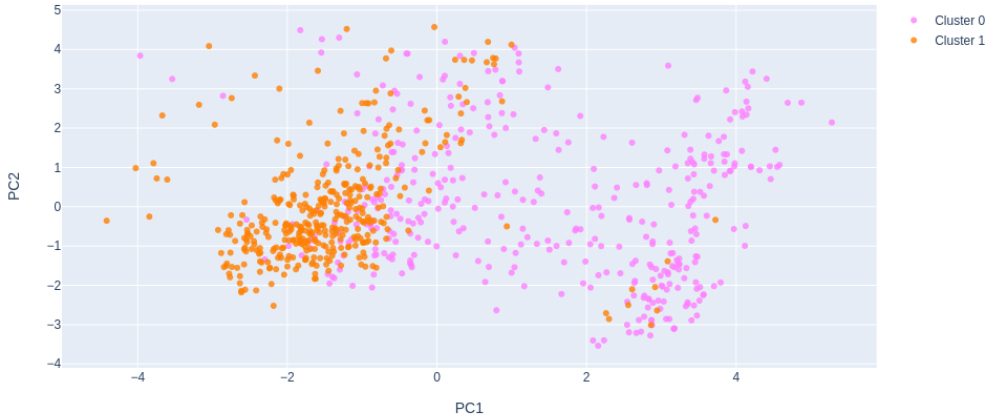

Рисунок 9 – Правильное распределение наблюдений на классы (стандартизированные)

Далее данные были нормализованы с помощью StandardScaler и применив расстояние Махаланобиса, был отсечен хвост у распределения Хи-Квадрат с 29 степенями свободы и 0.99 значением ppf.

2.4.2 KNN

Использовалось несколько видов кластеризации и метод ближайших соседей в качестве классификатора.

Далее приведена точность модели для различных расстояний и метода ближайших соседей используя GridSearchCV:

Таблица 4 — Результаты обучения KNN на различных метриках

Метрика	Precision	Recall	F1 score	Количество соседей
Euclidean	0.85	0.85	0.85	3
Squared euclidean	0.85	0.85	0.85	3
Cosine	0.86	0.86	0.86	4
Manhattan	0.88	0.87	0.87	10
Canberra	0.89	0.88	0.88	10
Hamming	0.87	0.87	0.87	10
Chebyshev	0.82	0.82	0.82	5
Jaccard	0.85	0.84	0.84	8

Таким образом, получилось, что наилучшим расстоянием для данной задачи является канберрское, манхэттенское и расстояние Хэмминга с количеством

соседей равным 10. При этом для расстояний Хэмминга и Жаккара использовались не стандартизированные данные, в силу определения этих расстояний.

Рисунок 10 – Распределение наблюдений на классы методом KNN (канберрское расстояние)

2.4.3 KMeans

В качестве следующего метода для данной задачи был использован KMeans. Процесс очистки данных аналогичен. Результат получился немного хуже, чем при использовании KNN, значения precision, recall и f1-score получились равные 0.81, 0.75 и 0.74 соответственно.

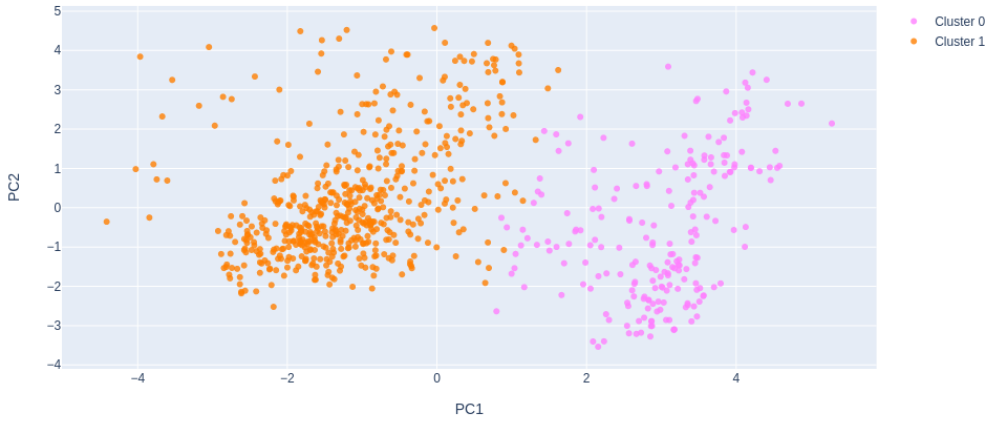

Рисунок 11 – Распределение наблюдений на классы методом KMeans

2.4.4 K-Medoids

Метод K-Medoids был взят для сравнения с KMeans. Этот метод использует медоиды, которые находятся близко к центроидам, но в отличие от них являются объектом, принадлежащего кластеру. Для данного метода получились следующие результаты:

Таблица 5 — Результаты обучения K-Medoids на различных метриках

Метрика	Precision	Recall	F1 score
Euclidean	0.51	0.51	0.51
Squared euclidean	0.51	0.51	0.51
Cosine	0.54	0.53	0.52
Manhattan	0.53	0.53	0.53
Canberra	0.79	0.71	0.69
Hamming	0.66	0.66	0.66
Chebyshev	0.54	0.54	0.54
Jaccard	0.80	0.71	0.69

Следовательно, лучшими метриками для данного метода оказались расстояние Жаккара, Канберрское и Хэмминга с f1-score равными 0.69, 0.69 и 0.66 соответсвенно.

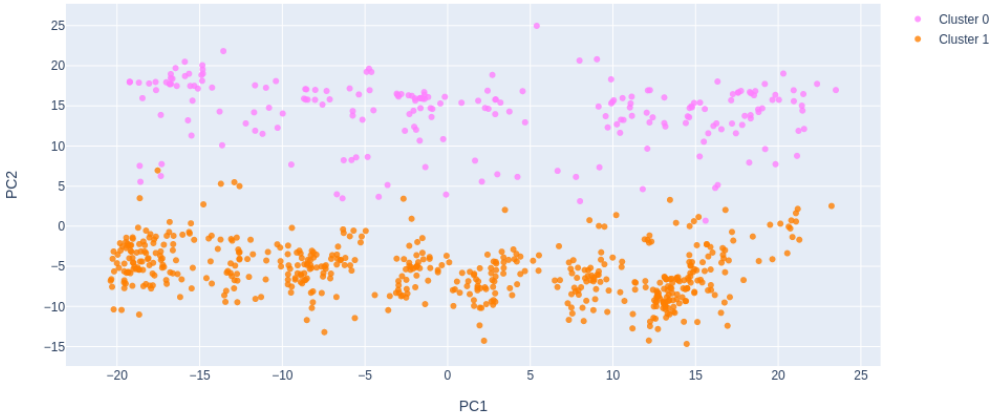

Рисунок 12 – Распределение наблюдений на классы методом K-Medoids (Расстояние Жаккара)

При этом для этого метода использовались не стандартизированные данные, так как на стандартизированных оказалось, что метод работает плохо.

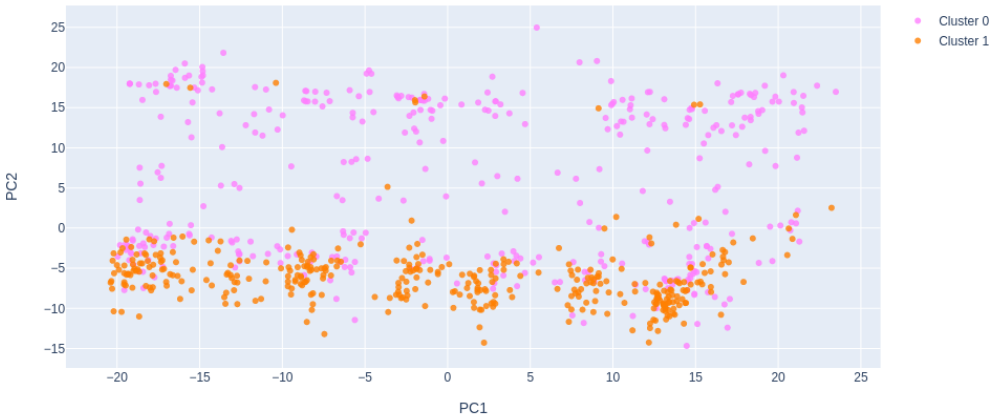

Рисунок 13 – Правильное распределение наблюдений на классы (нестандартизированные)

2.4.5 K-Prototype

Метод K-Prototype является объединением методов KMeans и KModes для численных и категориальных данных соответсвенно. Из этого следует, что результаты ожидаются чуть лучше, чем при использовании KMeans в силу определения метода, но на практике отличия в точности этих методов зафиксировано не было. Применив данный метод, значение precision — 0.81, recall — 0.75, f1-score — 0.74.

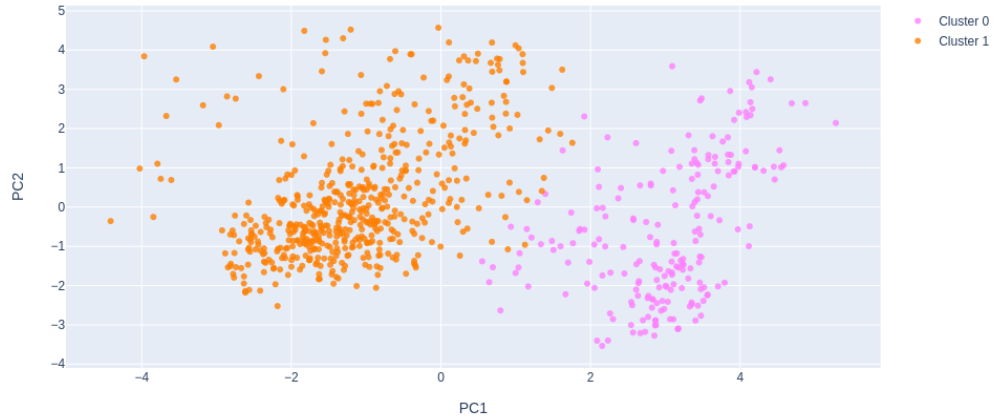

Рисунок 14 – Распределение наблюдений на классы методом K-Prototype

2.4.6 Выводы

Таким образом, получилось, что наилучшим методом для данной задачи оказался KNN с канберрским расстоянием с точностью 0.88%. При этом, стоит заметить, что эту задачу можно решить лучше, если использовать более сложные, но менее интерпретируемые методы, что выходит за пределы данной

работы. Преимуществом такого метода как KNN является то, что его достаточно легко интерпретировать, засчёт того, что можно сказать насколько одно наблюдение далеко от другого.

ЗАКЛЮЧЕНИЕ

В данной работе было проведено тщательное исследование существующих метрик, используемых в анализе данных. В теоретической части были рассмотрены особенности каждой метрики, их преимущества и недостатки, а также их применимость к разным типам данных.

В практической части исследования было проведено тестирование метрик на разных типах данных, таких как разреженные, строковые, смешанные. Кроме того, были рассмотрены конкретные сценарии использования метрик, такие как выявление выбросов, моделирование эмоциональной окраски текстов, определение будущей успеваемости студента в университете на основе данных при поступлении и исправление ошибок в написанном запросе.

В результате исследования было выявлено, что наилучшие результаты для разреженных данных достигаются при использовании канберрского расстояния и расстояния Жаккара со стеммером Портера. Расстояние Махаланобиса оказалось наиболее эффективным для выявления выбросов. Для смешанных данных наилучшей конфигурацией оказалось использование метода kNN и канберрского расстояния с количеством соседей равным 10.

Важно отметить, что основной целью данного исследования было выявление применимости различных метрик в разных ситуациях, а не достижение максимальной точности моделей. Для этого могут использоваться более сложные, но менее интерпретируемые методы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Метрика [Электронный ресурс]. — URL: http://www.machinelearning.ru/wiki/index.php?title=Метрика (Дата обращения 16.11.2022). Загл. с экр. Яз. рус.

2 Çolakoğlu, H. A generalization of the minkowski distance and new definitions of the central conics / H. Çolakoğlu // Turkish Journal of Mathematics. — 2020. — Т. 44. — С. 319–333.

3 17 types of similarity and dissimilarity measures used in data science. [Электронный ресурс]. — URL: https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681 (Дата обращения 16.11.2022). Загл. с экр. Яз. англ.

4 9 Distance Measures in Data Science [Электронный ресурс]. — URL: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa (Дата обращения 16.11.2022). Загл. с экр. Яз. англ.

5 Manhattan distance [Explained] [Электронный ресурс]. — URL: https://iq.opengenus.org/manhattan-distance/ (Дата обращения 16.11.2022). Загл. с экр. Яз. англ.

6 Chebyshev distance [Электронный ресурс]. — URL: https://iq.opengenus.org/chebyshev-distance/ (Дата обращения 16.11.2022). Загл. с экр. Яз. англ.

7 Canberra distance [Электронный ресурс]. — URL: http://www.scientificlib.com/en/Mathematics/LX/CanberraDistance.html (Дата обращения 16.11.2022). Загл. с экр. Яз. англ.

8 Barahimi, F. Massive enhanced extracted email features tailored for cosine distance / F. Barahimi // CoRR. — 2022. — Т. abs/2205.04819.

9 Расстояние Хэмминга [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=Расстояние_Хэмминга (Дата обращения 18.11.2022). Загл. с экр. Яз. рус.

10 The Levenshtein Algorithm [Электронный ресурс]. — URL: https://www.cuelogic.com/blog/the-levenshtein-algorithm (Дата обращения 19.11.2022). Загл. с экр. Яз. англ.

11 Задача о редакционном расстоянии, алгоритм Вагнера-Фишера [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=Задача_о_редакционном_расстоянии,_алгоритм_Вагнера-Фишера (Дата обращения 19.11.2022). Загл. с экр. Яз. рус.

12 Расстояние Левенштейна для чайников [Электронный ресурс]. — URL: https://habr.com/ru/post/676858/ (Дата обращения 19.11.2022). Загл. с экр. Яз. рус.

13 Задача о расстоянии Дамерау-Левенштейна [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=Задача_о_расстоянии_Дамерау-Левенштейна (Дата обращения 04.12.2022). Загл. с экр. Яз. англ.

14 Damerau Levenshtein distance [Электронный ресурс]. — URL: https://iq.opengenus.org/damerau-levenshtein-distance/ (Дата обращения 04.12.2022). Загл. с экр. Яз. англ.

15 Kosub, S. A note on the triangle inequality for the jaccard distance / S. Kosub // Pattern Recognition Letters. — 2019. — Т. 120. — С. 36–38.

16 Jaccard Index [Электронный ресурс]. — URL: https://deepai.org/machine-learning-glossary-and-terms/jaccard-index (Дата обращения 19.11.2022). Загл. с экр. Яз. англ.

17 Mahalanobis Distance – Understanding the math with examples (python) [Электронный ресурс]. — URL: https://www.machinelearningplus.com/statistics/mahalanobis-distance/ (Дата обращения 23.11.2022). Загл. с экр. Яз. англ.

18 Jensen Shannon Divergence [Электронный ресурс]. — URL: https://notesonai.com/Jensen-Shannon+Divergence (Дата обращения 04.12.2022). Загл. с экр. Яз. англ.

19 scipy.spatial.distance.jensenshannon [Электронный ресурс]. — URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.jensenshannon.html (Дата обращения 04.12.2022). Загл. с экр. Яз. англ.

20 Jensen Shannon Divergence [Электронный ресурс]. — URL: https://iq.opengenus.org/jensen-shannon-divergence/ (Дата обращения 04.12.2022). Загл. с экр. Яз. англ.

21 The Relationship between the Mahalanobis Distance and the Chi-Squared Distribution [Электронный ресурс]. — URL: https://markusthill.github.io/mahalanbis-chi-squared/ (Дата обращения 02.03.2023). Загл. с экр. Яз. англ.

22 Realinho, V. Predicting student dropout and academic success / V. Realinho, J. Machado, L. Baptista, M. V. Martins // Data. — 2022. — Т. 7, № 11.

Приложение А
Разреженные данные

```
import numpy as np
import pandas as pd
import nltk
import seaborn as sns
import re
import matplotlib.pyplot as plt
import nltk

from nltk.corpus import stopwords
from sklearn import preprocessing
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from nltk.stem.porter import PorterStemmer
from sklearn.metrics import classification_report
from nltk.stem import PorterStemmer, WordNetLemmatizer
from collections import Counter
from sklearn.preprocessing import Normalizer


def input_data():
    # Загружаем датасет:
    df_init = pd.read_csv("Emotion_final.csv")

    # Выбрасываем пустые строчки:
    df_init = df_init.dropna()

    # Заменяем названия, для объединения датасетов:
    df_init = df_init.rename(columns={"Text": "content", "Emotion": "sentiment"})

    # Загружаем другой датасет:
    df_init_tweet = pd.read_csv("tweet_emotions.csv")
    df_tweet = df_init_tweet.drop(labels="tweet_id", axis=1)

    # Выбрасываем пустые строчки:
    df_tweet = df_tweet.dropna()

    # Заменяем название, для объединения датасетов:
    df_tweet.sentiment = df_tweet.sentiment.replace("happiness", "happy")

    # Соединим датасеты
    df_concatted = pd.DataFrame(pd.concat([df_tweet, df_init]))

    # Перезагружаем индексы:
    df_concatted.reset_index(drop=True, inplace=True)
```

```
46
47      # Удалим все те эмоции, которые имеют слишком малое количество:
48      to_delete = [
49          "empty",
50          "boredom",
51          "relief",
52          "enthusiasm",
53          "fun",
54          "hate",
55          "love",
56          "worry",
57          "surprise",
58      ]
59
60      for val in to_delete:
61          df_concatted.drop(
62              df_concatted[df_concatted["sentiment"] == val].index, inplace=True
63          )
64
65      # Обновим индексы:
66      df_concatted.reset_index(drop=True, inplace=True)
67
68      return df_concatted
69
70
71  def clearing_data(X):
72      # Представление данных для проверки эффективности расстояний на данных
73      # с множественным пропуском
74      nltk.download("wordnet")
75      ps = PorterStemmer()
76      wnl = WordNetLemmatizer()
77
78      cleared_tweets_stem = []
79      cleared_tweets_lem = []
80      for i in range(0, len(X)):
81          # очистка от упоминаний других пользователей
82          tweets = re.sub("@[^ ]+", "", X[i])
83
84          # очистка от хештегов
85          tweets = re.sub("#[a-zA-Z0-9_]+", "", tweets)
86
87          # очистка от кавычек
88          tweets = re.sub("&quot", "", tweets)
89
90          # очистка от внешних ссылок
91          tweets = re.sub("https?:\/\/.*?[\s+]", "", tweets)
92
93          # очистка от знаков препинаний
```

```
        tweets = re.sub("[^a-zA-Z]", " ", tweets)

        # приведение всего к одному регистру
        tweets = tweets.lower()

        # разделение твита на отдельные слова
        tweets = tweets.split()

        # очистка от шумовых слов (стоп-слов) ( предлоги, цифры, частицы и т. п.)
        # применение стеммера Портера
        tweets_stem = [
            ps.stem(word) for word in tweets if not word in stopwords.words("english")
        ]

        # лемматизация
        tweets_lem = [
            wnl.lemmatize(word)
            for word in tweets
            if not word in stopwords.words("english")
        ]

        # добавление его в общий список
        cleared_tweets_stem.append(tweets_stem)
        cleared_tweets_lem.append(tweets_lem)
    return cleared_tweets_stem, cleared_tweets_lem


def data_transformation(cleared_data):
    words_of_tweets_stem = [a for b in cleared_data[0] for a in b]
    words_of_tweets_lem = [a for b in cleared_data[1] for a in b]

    # подсчитаем количество всех слов
    d_stem = Counter(words_of_tweets_stem)
    d_lem = Counter(words_of_tweets_lem)

    # формируем датасеты
    d_stem_df = pd.DataFrame(list(d_stem))
    d_lem_df = pd.DataFrame(list(d_lem))

    for val in [[d_stem_df, d_stem], [d_lem_df, d_lem]]:
        val[0]["value"] = list(val[1].values())
        val[0].drop(val[0][(val[0].value <= 1)].index, inplace=True)
        val[0].reset_index(inplace=True)
        val[0].rename({0: "word"}, axis=1, inplace=True)
        val[0] = val[0].sort_values(by="value", ascending=False)

    # формируем строки в которых указываем количество повторений слова
    words_of_tweet_lst_stem = []
```

```
142     words_of_tweet_lst_lem = []
143 
144     list_of_d_stem_df = list(d_stem_df.word)
145     list_of_d_lem_df = list(d_lem_df.word)
146 
147     for val in [
148         [words_of_tweet_lst_stem, cleared_data[0], list_of_d_stem_df],
149         [words_of_tweet_lst_lem, cleared_data[1], list_of_d_lem_df],
150     ]:
151         for i in range(0, len(val[1])):
152             d = dict()
153             for inner_val in val[1][i]:
154                 # если слово присутствует в словаре
155                 if inner_val in d:
156                     if inner_val in val[2]:
157                         d[inner_val] += 1
158                 # если слово отсутствует в словаре
159                 else:
160                     if inner_val in val[2]:
161                         d[inner_val] = 1
162             val[0].append(d)
163     return words_of_tweet_lst_stem, words_of_tweet_lst_lem
164 
165 
166 # Представление данных в удобочитаемом виде для модели
167 df = input_data()
168 X = df["content"]
169 y = df["sentiment"]
170 
171 cleared_data = clearing_data(X)
172 
173 transformed_data = data_transformation(cleared_data)
174 
175 # Создание датасетов для обучения:
176 df_stem = pd.DataFrame.from_dict(transformed_data[0])
177 df_stem.fillna(0, inplace=True)
178 
179 df_lem = pd.DataFrame.from_dict(transformed_data[1])
180 df_lem.fillna(0, inplace=True)
181 
182 # Кодирование таргета
183 le = preprocessing.LabelEncoder()
184 target = le.fit_transform(y)
185 
186 X_train_stem, X_test_stem, y_train_stem, y_test_stem = train_test_split(
187     df_stem, target, test_size=0.1, random_state=41
188 )
189 
```

```
190 X_train_lem, X_test_lem, y_train_lem, y_test_lem = train_test_split(
191     df_lem, target, test_size=0.1, random_state=41
192 )
193
194 # Балансирование классов
195 X_resampled_stem_train, y_resampled_train_stem = SMOTE(
196     k_neighbors=25, random_state=41
197 ).fit_resample(X_train_stem, y_train_stem)
198 X_resampled_lem_train, y_resampled_train_lem = SMOTE(
199     k_neighbors=25, random_state=41
200 ).fit_resample(X_train_lem, y_train_lem)
201 X_resampled_stem_test, y_resampled_test_stem = SMOTE(
202     k_neighbors=25, random_state=41
203 ).fit_resample(X_test_stem, y_test_stem)
204 X_resampled_lem_test, y_resampled_test_lem = SMOTE(
205     k_neighbors=25, random_state=41
206 ).fit_resample(X_test_lem, y_test_lem)
207
208 # Нормирование данных
209 transformer_stem_train = Normalizer().fit(X_resampled_stem_train)
210 transformer_lem_train = Normalizer().fit(X_resampled_lem_train)
211 transformer_stem_test = Normalizer().fit(X_resampled_stem_test)
212 transformer_lem_test = Normalizer().fit(X_resampled_lem_test)
213
214 X_normalized_stem_train = transformer_stem_train.transform(X_resampled_stem_train)
215 X_normalized_lem_train = transformer_lem_train.transform(X_resampled_lem_train)
216 X_normalized_stem_test = transformer_stem_test.transform(X_resampled_stem_test)
217 X_normalized_lem_test = transformer_lem_test.transform(X_resampled_lem_test)
218
219 metrics = [
220     "jaccard",
221     "euclidean",
222     "sqeuclidean",
223     "cosine",
224     "manhattan",
225     "canberra",
226     "chebyshev",
227     "correlation",
228     "hamming",
229 ]
230
231 for val in metrics:
232     print("Stemming")
233     knn = KNeighborsClassifier(n_neighbors=8, metric=val, n_jobs=-1)
234     knn.fit(X_normalized_stem_train, y_resampled_train_stem)
235     y_pred = knn.predict(X_normalized_stem_test)
236     report = classification_report(y_resampled_test_stem, y_pred)
237     print(report)
```

```
238
239     print("Lemmatization")
240     knn = KNeighborsClassifier(n_neighbors=8, metric=val, n_jobs=-1)
241     knn.fit(X_normalized_lem_train, y_resampled_train_lem)
242     y_pred = knn.predict(X_normalized_lem_test)
243     report = classification_report(y_resampled_test_lem, y_pred)
244     print(report)
```

Приложение Б
Метрики для выявления выбросов в данных

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.spatial.distance import mahalanobis
from scipy.stats import chi2
from sklearn.cluster import DBSCAN
from statsmodels.formula.api import ols

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

df["bathrooms"] = df["bathrooms"].astype(int)
df["floors"] = df["floors"].astype(int)

dfs = (df.copy(), df.copy())

dfs[0].drop(
    columns=[
        "date",
        "id",
        "sqft_living15",
        "sqft_lot15",
        "zipcode",
        "price",
        "yr_renovated",
    ],
    inplace=True,
)
dfs[1].drop(
    columns=[
        "date",
        "id",
        "sqft_living15",
        "sqft_lot15",
        "zipcode",
        "sqft_living",
        "price",
        "grade",
        "sqft_above",
        "yr_built",
        "waterfront",
        "yr_renovated",
    ],
```

```
    inplace=True,
)

target = df["price"]

# очистка от выбросов расстоянием Махаланобиса
def mahalanobisCleaning(dfs):
    # вычисление ковариационной матрицы
    inv_cov_mats = [
        np.linalg.matrix_power(dfs[0].cov(), -1),
        np.linalg.matrix_power(dfs[1].cov(), -1),
    ]
    mean_values = [np.mean(dfs[0], axis=0), np.mean(dfs[1], axis=0)]
    mah_dfs = [dfs[0].copy(), dfs[1].copy()]
    mah_dfs[0]["mah_dist"] = 0
    mah_dfs[1]["mah_dist"] = 0

    columns = [14, 9]

    mah_borders = (chi2.ppf(0.99, columns[0]), chi2.ppf(0.99, columns[1]))
    mah_outliers = [pd.DataFrame(), pd.DataFrame()]

    # вычисление расстояния
    for i in range(0, 2):
        for j, values in mah_dfs[i].iterrows():
            values = values[0 : columns[i]]
            mah_dfs[i].loc[j, "mah_dist"] = (
                mahalanobis(values, mean_values[i], inv_cov_mats[i]) ** 2
            )

        # находятся выбросы
        mah_outliers[i] = mah_dfs[i][mah_dfs[i]["mah_dist"] > mah_borders[i]].index

        # удаляются выбросы
        print("Количество удалённых строк:", len(mah_outliers[i]))
        print(f"Это {len(mah_outliers[i]) / len(mah_dfs[i]) * 100}% датасета")

        mah_dfs[i].drop(index=mah_outliers[i], inplace=True)
        mah_dfs[i].reset_index(inplace=True)
        mah_dfs[i].drop(columns=["index", "mah_dist"], inplace=True)


# очистка от выбросов расстоянием Кука
def cooksCleaning(dfs, target):
    cooks_dfs = (dfs[0].copy(), dfs[1].copy())

    # выбираем колонки
    Xs = [
```

```
        cooks_dfs[0][cooks_dfs[0].columns[0:14]],
        cooks_dfs[1][cooks_dfs[1].columns[0:9]],
    ]

    Xs[0] = sm.tools.tools.add_constant(Xs[0])
    Xs[1] = sm.tools.tools.add_constant(Xs[1])

    # обучение моделей
    models = [
        sm.regression.linear_model.OLS(np.asarray(target), Xs[0]).fit(),
        sm.regression.linear_model.OLS(np.asarray(target), Xs[1]).fit(),
    ]

    inflce_lsts = [[], []]

    # получение расстояний Кука
    for i in range(0, 2):
        inflce = models[i].get_influence()
        inflce_lsts[i] = inflce.cooks_distance[0]
        cooks_dfs[i]["influence"] = inflce_lsts[i]

    # удаляются выбросы
    cooks_borders = [4 / len(cooks_dfs[0]), 4 / len(cooks_dfs[1])]
    cooks_outliers = [pd.DataFrame(), pd.DataFrame()]
    for i in range(0, 2):
        cooks_outliers[i] = cooks_dfs[i][cooks_dfs[i]["influence"] > cooks_borders[i]]
        print("Количество удалённых строк:", len(cooks_outliers[i]))
        print(f"Это {len(cooks_outliers[i]) / len(cooks_dfs[i]) * 100}% датасета")

        cooks_dfs[i].drop(index=cooks_outliers[i].index, axis=0, inplace=True)
        cooks_dfs[i].reset_index(inplace=True)
        cooks_dfs[i].drop(columns=["index", "influence"], inplace=True)


# очистка от выбросов методом DBSCAN
def dbscanCleaning(dfs, border1, border2):
    dbscan_1 = DBSCAN()
    dbscan_2 = DBSCAN()

    dbscans = [dbscan_1.fit(dfs[0]), dbscan_2.fit(dfs[1])]
    unique_values = [[], []]
    count_of_outliers = [0, 0]
    removed_indexes = [[], []]
    len_of_mah_outliers = [border1, border2]

    # настройка DBSCAN
    for i in range(0, 2):
        eps = 0.5
```

```
        for value in dbscans[i].labels_:
            if value not in unique_values[i]:
                unique_values[i].append(value)
            if value == -1:
                count_of_outliers[i] += 1

        # настройка eps
        while count_of_outliers[i] > len_of_mah_outliers[i]:
            dbscans[i] = DBSCAN(eps=eps)
            dbscans[i].fit(dfs[i])
            labels_df = pd.DataFrame(dbscans[i].labels_, columns=["labels"])
            labels_df.index = dfs[i].index
            removed_df = labels_df[labels_df["labels"] == -1]
            removed_indexes[i] = removed_df.index
            count_of_outliers[i] = len(removed_indexes[i])
            eps += 100

    print("Количество удалённых строк:", count_of_outliers[0])
    print(f"Это {count_of_outliers[0] / len(dfs[0]) * 100}% датасета")

    print("Количество удалённых строк:", count_of_outliers[1])
    print(f"Это {count_of_outliers[1] / len(dfs[1]) * 100}% датасета")

    dbscan_dfs = [pd.DataFrame(), pd.DataFrame()]
    for i in range(0, 2):
        dbscan_dfs[i] = dfs[i].drop(index=removed_indexes[i])
        dbscan_dfs[i].reset_index(inplace=True)
        dbscan_dfs[i].drop(columns="index", inplace=True)

mahalanobisCleaning(dfs)
cooksCleaning(dfs, target)
dbscanCleaning(dfs, 1984, 1106)
```

Приложение В
Метрики для строк

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from Levenshtein import distance
from fastDamerauLevenshtein import damerauLevenshtein

df = pd.read_csv("TMDb_updated.csv")
df = df["title"].to_frame()
df = df["title"].str.lower().to_frame()
df["title"].astype(str)

def search_correction_lev(input_title, input_df):
    lev_distances = np.array([])
    input_title = input_title.lower()
    if len(input_title) <= 3:
        # оставляем все те строки, в которых есть входная строка
        df = input_df.loc[input_df["title"].str.contains(input_title, case=False)]
        # считаем расстояния между ними
        for title in df["title"]:
            lev_distances = np.append(lev_distances, distance(input_title, title))
        df["distances"] = lev_distances
        # сортируем, с наименьшим расстоянием возвращаем
        df = df.sort_values(by="distances")
        return df["title"].iloc[0:5]
    else:
        for title in input_df["title"]:
            lev_distances = np.append(lev_distances, distance(input_title, title))
        input_df["distances"] = lev_distances
        input_df = input_df.sort_values(by="distances")
        return input_df["title"].iloc[0:5]

def search_correction_dam_lev(input_title, input_df):
    lev_distances = np.array([])
    input_title = input_title.lower()
    if len(input_title) <= 3:
        # оставляем все те строки, в которых есть входная строка
        df = input_df.loc[input_df["title"].str.contains(input_title, case=False)]
        # считаем расстояния между ними
        for title in df["title"]:
            lev_distances = np.append(
                lev_distances, damerauLevenshtein(input_title, title, similarity=False)
            )
```

```
        df["distances"] = lev_distances
        # сортируем, с наименьшим расстоянием возвращаем
        df = df.sort_values(by="distances")
        return df["title"].iloc[0:5]
    else:
        for title in input_df["title"]:
            lev_distances = np.append(
                lev_distances, damerauLevenshtein(input_title, title, similarity=False)
            )
        input_df["distances"] = lev_distances
        input_df = input_df.sort_values(by="distances")
        return input_df["title"].iloc[0:5]

print(search_correction_lev("Aline", df))
print(search_correction_dam_lev("Aline", df))
```

Приложение Г
Метрики для смешанных данных

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from kmodes.kprototypes import KPrototypes
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from scipy.spatial.distance import mahalanobis
from scipy.stats import chi2

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

# убираем лишний класс
df = df.drop(df[df.Target == "Enrolled"].index)
df = df.reset_index()
df.drop(columns="index", inplace=True)

df.drop(
    columns=[
        "Curricular units 1st sem (without evaluations)",
        "Curricular units 2nd sem (without evaluations)",
        "Curricular units 1st sem (enrolled)",
        "Curricular units 2nd sem (enrolled)",
    ],
    inplace=True,
)

target = df["Target"]
le = preprocessing.LabelEncoder()

y = le.fit_transform(target)
X = df.drop(columns="Target")

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=41
)
```

```
46
47 # увеличиваем количество данных
48 X_resampled_train, y_resampled_train = SMOTE(
49     k_neighbors=3, random_state=41
50 ).fit_resample(X_train, y_train)
51 X_resampled_test, y_resampled_test = SMOTE(k_neighbors=3, random_state=41).fit_resample(
52     X_test, y_test
53 )
54
55 # нормализуем данные
56 standart_scaler_train = StandardScaler()
57 standart_scaler_train.fit(X_resampled_train)
58 X_standart_train = standart_scaler_train.transform(X_resampled_train)
59
60 standart_scaler_test = StandardScaler()
61 standart_scaler_test.fit(X_resampled_test)
62 X_standart_test = standart_scaler_test.transform(X_resampled_test)
63
64
65 def mahal(X, y):
66     matrix_cov_rev = np.linalg.matrix_power(pd.DataFrame(X).cov(), -1)
67     mean_values = np.mean(X, axis=0)
68     mah_df99 = pd.DataFrame(X).copy()
69     mah_df99["mah_dist"] = 0
70     for i, values in mah_df99.iterrows():
71         values = values[0:30]
72         mah_df99.loc[i, "mah_dist"] = (
73             mahalanobis(values, mean_values, matrix_cov_rev) ** 2
74         )
75     mah_border99 = chi2.ppf(0.99, 29)
76     mah_outliers99 = mah_df99[mah_df99["mah_dist"] > mah_border99].index
77     mah_df99 = pd.DataFrame(X).drop(index=mah_outliers99)
78     mah_df99 = mah_df99.reset_index()
79     mah_df99 = mah_df99.drop(columns="index")
80     y_mah = pd.DataFrame(y).drop(index=mah_outliers99)
81     return mah_df99, y_mah
82
83
84 # очищаем от выбросов
85 X_standart_train, y_resampled_train_standart = mahal(
86     X_standart_train, y_resampled_train
87 )
88 X_standart_test, y_resampled_test_standart = mahal(X_standart_test, y_resampled_test)
89
90 X_resampled_train, y_resampled_train_n_standart = mahal(
91     X_resampled_train, y_resampled_train
92 )
93 X_resampled_test, y_resampled_test_n_standart = mahal(
```

```
    X_resampled_test, y_resampled_test
)

# KNN
metrics = [
    ["euclidean", "sqeuclidean", "cosine", "manhattan", "canberra", "chebyshev"],
    ["jaccard", "hamming"],
]

for i in range(0, 2):
    knn = KNeighborsClassifier()
    k_range = list(range(1, 20))
    param_grid = dict(
        n_neighbors=k_range,
        algorithm=["auto", "ball_tree", "kd_tree", "brute"],
        metric=metrics[i],
    )

    # ищем наилучший классификатор
    grid = GridSearchCV(
        knn,
        param_grid,
        cv=10,
        scoring="f1_weighted",
        return_train_score=False,
        n_jobs=-1,
    )
    grid.fit(X_standart_train, y_resampled_train_standart[0])
    y_pred = grid.predict(X_standart_test)

# K-Means
kmeans = KMeans(n_clusters=2).fit(X_standart_train)
y_pred = kmeans.predict(X_standart_test)

# K-Prototype
X_prot = X_standart_train
X_prot.columns = X.columns.values

# указываем категориальные признаки
list_of_cat_features = [
    "Marital status",
    "Nacionality",
    "Displaced",
    "Gender",
    "International",
    "Father's qualification",
    "Mother's qualification",
    "Father's occupation",
```

```
    "Mother's occupation",
    "Educational special needs",
    "Debtor",
    "Tuition fees up to date",
    "Scholarship holder",
    "Application mode",
    "Course",
    "Daytime/evening attendance",
    "Previous qualification",
]
for feature in list_of_cat_features:
    X_prot[feature] = X_prot[feature].astype(object)


cat_columns = [
    X_prot.columns.get_loc(col) for col in list(X_prot.select_dtypes("object").columns)
]

k_prot = KPrototypes(n_jobs=-1, n_clusters=2, random_state=41)
k_prot.fit(X_prot, categorical=cat_columns)
y_pred = k_prot.predict(X_standart_test, categorical=cat_columns)

# K-Medoids
km_model = KMedoids(
    n_clusters=2,
    metric="jaccard",
    method="pam",
    init="k-medoids++",
    random_state=41,
).fit(X_resampled_train)
clusters = km_model.predict(X_resampled_test)
```

Приложение Д
CD-диск с отчётом о выполненной работе

На приложенном диске содержится полный код исследования.
code.zip — исходный код исследования.