Отчет по практическому заданию №1
Метрические алгоритмы классификации

Каратыщев Дмитрий Иванович

Октябрь 2023

1 Введение

Данное практическое задание направлено на изучение метрических алгоритмов классификации на примере *K Nearest Neighbors*, а также улучшение навыков работы с изображениями. Проводится исследование датасета **MNIST**, который используется для обучения метрического классификатора. Изучаются основные метрики - косинусная и евклидова, а также осуществляется сравнение между различными алгоритмами нахождения ближайших соседей. Целью исследования является выбор лучшего алгоритма и его проверка на аугментированных данных с дальнейшим анализом ошибок. Аугментация данных проводится на основе базовых функций сдвига, поворота, фильтра Гаусса и морфологических операций: эрозия, дилатация, открытие и закрытие.

2 Постановка задачи

Имеется выборка из рукописных изображений цифр **MNIST**. Размер выборки $N = 70000$ элементов. Датасет **MNIST** хранит каждый объект в виде 784-размерного вектора пикселей, который кодирует картинку 28×28. Каждый пиксель имеет значение $p \in [0, 255]$, отражающее оттенок пикселя по шкале серого цвета.

Необходимо разделить исходную выборку объектов $X = \{x_1, x_2, ..., x_N\}$, $Y = \{y_1, y_2, ..., y_N\}$ на обучающую X_{train}, Y_{train} (первые $N_{train} = 60000$ элементов) и тестовую X_{test}, Y_{test} (последние $N_{test} = 10000$ элементов). Здесь и далее X и Y - это множества объектов и откликов на них.

Применяя алгоритм *K Nearest Neighbors* с методами нахождения ближайших соседей *m_ own, brute, kd_ tree, ball_ tree* и метриками расстояния *euclidean и cosine*, нужно выделить лучший на 3-х фолдовой кросс-валидации и провести на нём серию из 6 экспериментов.

2.1 Метрики

Определим базовые функции расстояния, относящиеся к используемым метрикам, а также точность (долю правильных ответов):

$$\rho_{euclidean}(x, y) = \sqrt{\sum_{k=1}^{d} (y_k - x_k)^2} = \sqrt{\sum_{k=1}^{d} y_k^2 - 2y^T x + \sum_{k=1}^{d} x_k^2}$$

$$\rho_{cosine}(x, y) = 1 - \frac{\langle x, y \rangle}{\|x\| \|y\|} = 1 - \frac{y^T x}{\sqrt{\sum_{k=1}^{d} y_k^2} \sqrt{\sum_{k=1}^{d} x_k^2}}$$

$$Accuracy = \frac{\sum_{n=1}^{N_{test}} [y_{test} == y_{predict}]}{N_{test}}$$

Евклидова метрика является базовой функцией расстояния в большинстве задач. Она отражает интуитивные понятия расстояния между двумя точками в пространстве $\mathbb{R}^d$. Основными проблемами, связанными с данной метрикой являются: "проклятие размерности", сильное влияние шумовых признаков, а также ненормализованных признаков

Косинусная метрика не зависит от длин векторов, что позволяет искать схожесть объектов, нормы которых могут сильно отличаться друг от друга. На изображениях датасета **MNIST** это выражается в яркости объектов.

2.2 Взвешенный метод

В процессе проведения экспериментов необходимо реализовать взвешенный метод *K Nearest Neighbors*. В данном методе каждый объект имеет свой вес в зависимости от расстояния до него. Определим вес ближайшего соседа x_i для объекта x_{obj} следующим образом:

$$w_{obj,i} = \frac{1}{\rho(x_{obj}, x_i) + 10^{-5}}$$

2.3 Список экспериментов

1. Оценка скорости работы алгоритмов поиска 5 ближайших соседей на тестовой выборке для 10, 20 и 100 случайно выбранных признаков по евклидовой метрике

2. Оценка точности и времени работы алгоритмов на кросс-валидации с 3-мя фолдами при количестве ближайших соседей $k \in [1, 10]$ для различных метрик

3. Повторение эксперимента №2 с учётом взвешенности алгоритмов и сравнение с результатом невзвешенного лучшего результата

4. Применение лучшего алгоритма к исходным обучающей и тестовой выборкам. Оценка точности и анализ матрицы ошибок

5. Выполнение аугментации обучающей выборки. Подбор оптимальных параметров необходимых трансформаций (поворот, сдвиг, Гауссовский фильтр, морфологические операции) по 3-х фолдовой кросс-валидации.
   5.1. Поворот - $5, 10, 15$ в каждую из сторон
   5.2. Сдвиг - $1, 2, 3$ пикселя по каждой из двух размерностей
   5.3. Дисперсия фильтра Гаусса - $0.5, 1.0, 1.5$. Подбор размеров ядра проводится самостоятельно
   5.4. Морфологические операции - эрозия, дилатация, открытие и закрытие с ядром 2 на 2
6. Выполнение аналогичной аугментации тестовой выборки. Качественное сравнение подходов из 5 и 6 экспериментов

3 Результаты экспериментов

В этой секции рассматриваются результаты каждого из экспериментов. Полученные данные визуализируются при помощи python-модуля matplotlib

3.1 Результаты первого эксперимента

Для подмножества из **10 случайно выбранных признаков** быстрее всего отработал алгоритм **my_own** с показателем времени, равным **45.8** μs. Это значительно отличается от методов *brute*, *kd_tree* и *ball_tree*, ибо на таком количестве признаков алгоритм отработал практически мгновенно. В методах *kd_tree* и *ball_tree* идут дополнительные построения структур данных, что занимает много времени, а имплементация метода *brute* хоть и имеет схожую идею, но, по результатам эксперимента, проигрывает по времени. Лучшее время работы помогает достичь реализованная в *my_own* k-я порядковая статистика

Ранжирование по времени: *my_own* ($45.8\mu s$) $\longrightarrow$ **kd_tree** ($9.83s$) $\longrightarrow$ **ball_tree** ($20.9s$) $\longrightarrow$ **brute** ($48.4s$)

Для подмножества из **20 случайно выбранных признаков** быстрее всего отработал алгоритм **kd_tree** с показателем времени, равным **25.4** s. На таком количестве признаков структура данных, реализованная методом *kd_tree* показывает себя гораздо лучше других алгоритмов.

Ранжирование по времени: **kd_tree** ($25.4s$) $\longrightarrow$ **my_own** ($57.2s$) $\longrightarrow$ **brute** ($59.5s$) $\longrightarrow$ **ball_tree** ($1min, 20s$)

Для подмножества из **100 случайно выбранных признаков** быстрее всего отработал алгоритм **my_own** с показателем времени, равным **1** min **40**s. Интерпретация результата схожа с результатом для подмножества из

10 признаков, ибо построение структур данных требует времени.

Ранжирование по времени: **my_own** $(1min, 40s)$ $\longrightarrow$ **brute** $(2min, 56s)$ $\longrightarrow$ **ball_tree** $(7min, 54s)$ $\longrightarrow$ **kd_tree** $(8min, 18s)$

3.2 Результаты второго эксперимента

Для каждой метрики быстрее всего отработал алгоритм **my_own** с показателями времени, равными $\mathbf{1}min\mathbf{40}s$ при евклидовой и $\mathbf{1}min\mathbf{37}s$ при косинусной.

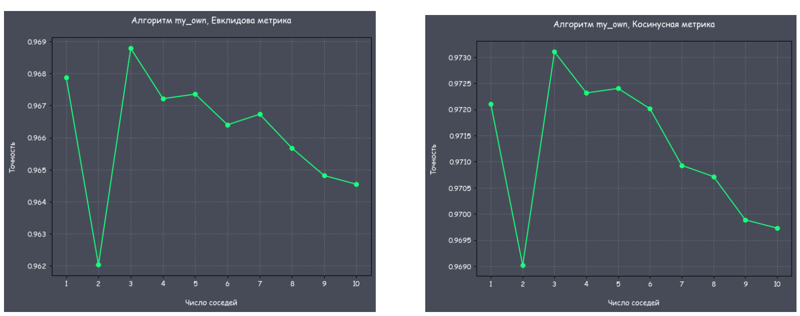
Методы *kd_tree* и *ball_tree* показали достаточно большое время работы при кросс-валидации, так как на каждом новом фолде приходилось очередной раз проводить построение структур данных.

Вычисления оказались незначительно быстрее при косинусной метрике

Ранжирование по времени

Евклидова метрика: **my_own** $(1min, 40s)$ $\longrightarrow$ **brute** $(2min, 36s)$ $\longrightarrow$ **ball_tree** $(39min, 58s)$ $\longrightarrow$ **kd_tree** $(50min, 40s)$
Косинусная метрика: **my_own** $(1min, 37s)$ $\longrightarrow$ **brute** $(2min, 34s)$

Приведём график зависимости точности от числа соседей. Он одинаковый для всех методов, кроме одного - метода **my_own** при косинусной метрике



(a) Основной график

(b) Лучший график, my_own

Лучше всего себя показывает метрика **cosine** в алгоритме **my_own**. Она лучше как по точности, так и по времени работы.

На графиках при $k = 2$ наблюдается резкий спад в точности. Это связано с тем, что меткой для объекта, у которого два ближайших соседа имеют разные классы, может быть как метка первого, так и метка второго соседа. Это невзвешенный метод, поэтому каждая метка имеет равный "вес" и голосование проводится равновероятно по этим меткам

Также видно, что при $k = 3$ имеем самую высокую точность алгоритмов.

С увеличением k точность постепенно падает. Это также связано с равновероятным голосованием по меткам ближайших соседей

3.3 Результаты третьего эксперимента

Для каждой метрики быстрее всего отработал алгоритм **my_own** с показателями времени, равными **1** *min* **38** *s* при евклидовой и **1** *min* **37** *s* при косинусной.

Методы *kd_tree* и *ball_tree* показали достаточно большое время работы при кросс-валидации взвешенного метода, так как на каждом новом фолде приходилось очередной раз проводить построение структур данных, как и не во взвешенном методе.

Вычисления оказались незначительно быстрее при косинусной метрике

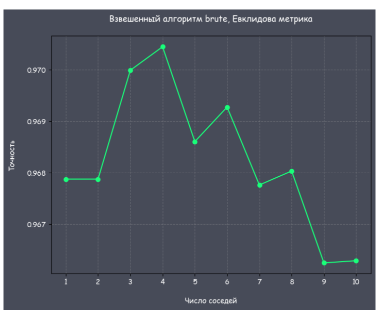
Ранжирование по времени

Евклидова метрика: **my_own** $(1min, 38s) \longrightarrow$ **brute** $(2min, 34s) \longrightarrow$ **ball_tree** $(39min, 58s) \longrightarrow$ **kd_tree** $(50min, 19s)$

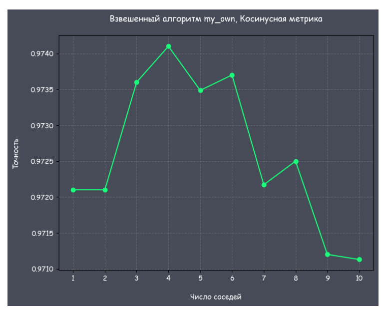
Косинусная метрика: **my_own** $(1min, 37s) \longrightarrow$ **brute** $(2min, 34s)$

По времени работы отличий от невзвешенного метода практически **не наблюдается**

Приведём основной график зависимости точности предсказания от числа соседей. Он одинаковый для всех методов, кроме одного - метода **my_own** на косинусной метрике



(a) Основной график



(b) Лучший график, my_own

На первом графике при $k > 5$ наблюдается постепенный спад в точности. Вероятно, причиной служит то, что в большей окрестности объекта может находится множество различных классов, из которых выбор подходящего будет не совсем точным в предположении гипотезы компактности метрического алгоритма

Также видно, что при $k = 4$ имеем самую высокую точность алгоритмов.

Сравнивая с невзвешенным методом, получаем, что взвешенный метод работает незначительно лучше. Точность лучшего алгоритма в невзвешенном методе - 0.973099. Во взвешенном - 0.974099

Получаем, что лучшая конфигурация, рассматриваемая в данном исследовании для датасета MNIST, - это **взвешенный** алгоритм поиска ближайших соседей **my_own** при использовании метрики **cosine** и числе ближайших соседей $k = 4$

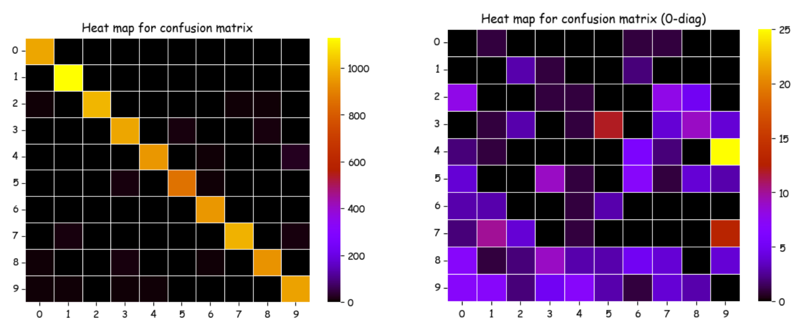
3.4 Результаты четвёртого эксперимента

Точность на тестовой выборке получилась **выше**, чем при кросс-валидации: $0.9752 > 0.97409$. Обученная модель ошиблась на 248 объектах.

Также стоит упомянуть, что лучший результат на датасете **MNIST** на текущий момент (2023 год) был получен в 2020 году для модели **Branching/Merging CNN + Homogeneous Vector Capsules.** Процент ошибок на ней равен 0.13%. Точность $= 0.9987$

Интересно, что модифицированный алгоритм KNN - *Large-Margin KNN* имеет процент ошибок 0.9% на данном датасете и находится на 61 месте в мировом рейтинге

В результате эксперимента получена следующая матрица ошибок:



(a) Матрица ошибок - heat map (b) Матрица ошибок - normalized

Анализ матрицы ошибок показывает, что используемый метрический классификатор больше всего ошибается на паре цифр $(4, 9)$, путая 4 с 9. Также видна сильная связь для ошибок для следующих пар цифр (порядок важен): $3 \mapsto 5$, $7 \mapsto 9$, $7 \mapsto 1$, $5 \mapsto 3$, $8 \mapsto 3$

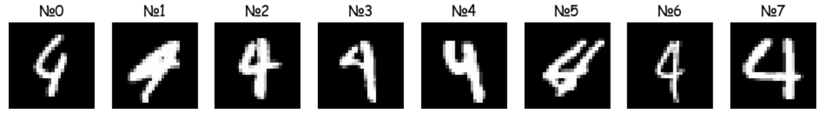
Меньше всего ошибок допускается на цифрах: $0, 1, 6$. Для них число ошибок равно $3, 6, 10$ соответственно

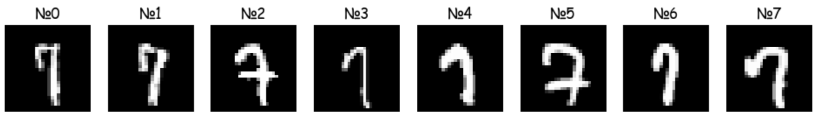
Больше всего ошибок допускается на цифрах: $4, 8, 9$. Для них число ошибок равно $36, 38, 39$ соответственно

Основная характерная черта "ошибочных" объектов в их визуальной дуальности. Заметна разница в толщине линий и повороте цифры.

Заметна тенденция к увеличению числа ошибок для больших цифр, которые классификатор путает чаще, чем меньшие.

Примеры изображений с ошибками в предикте





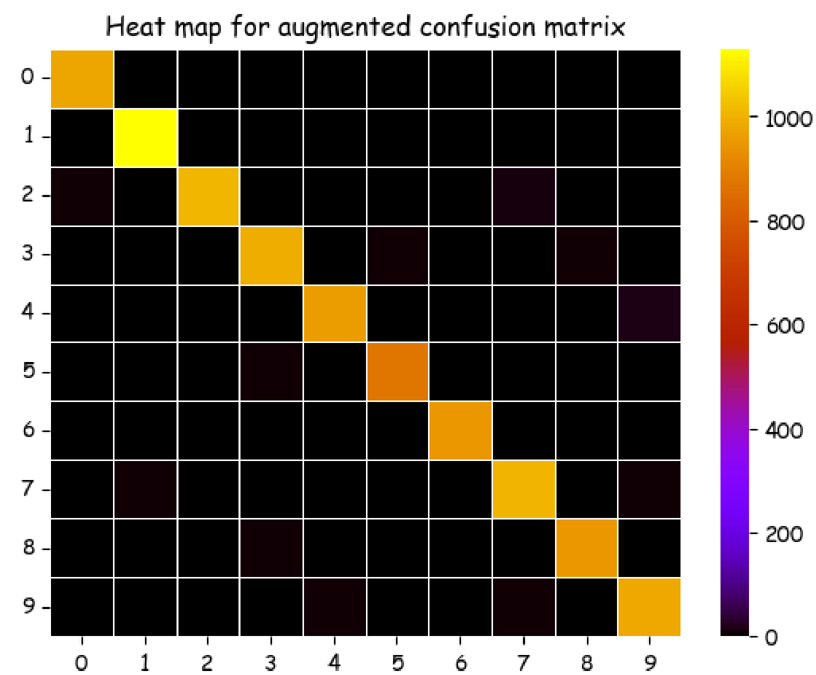
3.5 Результаты пятого эксперимента

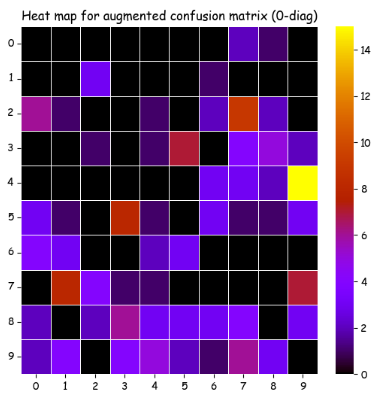
На основе 3-х фолдовой кросс-валидации были получены параметры для всех четырёх трансформаций. Исследование проводилось на изначальной обучающей выборке.

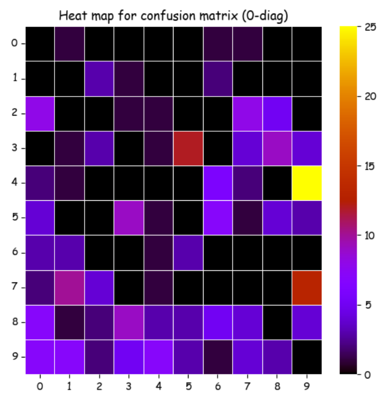
Аугментированная выборка данных состояла из 360000 объектов. К изначальной обучающей выборке были добавлены объекты, показавшие лучший результат по точности на каждой из трансформаций, а также была произведена гибридная трансформация - все четыре типа трансформаций были применены последовательно к изначальной выборке.

Точность повысилась значительно, в сравнении с неаугментированными данными. Разница в точности составляет $0.9822 - 0.9752 = 0.007$, разница в процентном соотношении ошибок составляет $2.48\% - 1.78\% = 0.7\%$

Получены следующие матрицы ошибок:







Видим, что были исправлены ошибки, например, связанные с определением 4 как 0, либо как 1

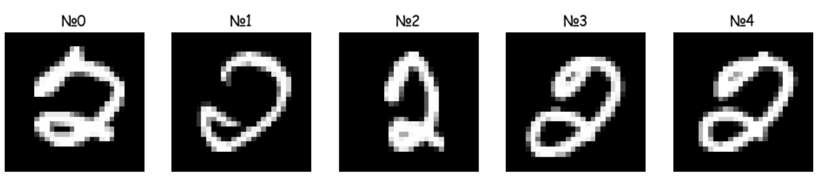
Поворот позволил избавиться от ошибок, при которых несимметричные цифры предсказываются неправильно. Например, $4 \to 1$, либо $2 \to 3$. Также данный тип трансформации позволяет сделать модель более устойчивой к повёрнутым цифрам в выборке

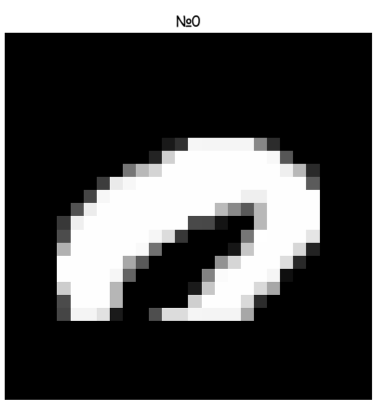
Сдвиг позволил незначительно избавиться от ошибок, связанных с неотцентрированными цифрами

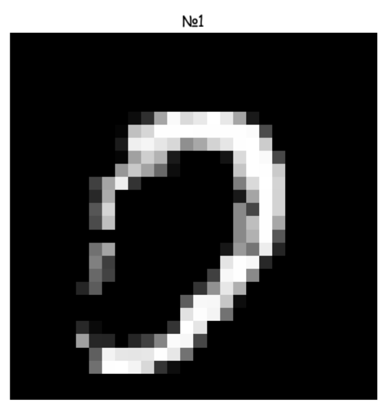
Фильтрация Гаусса помогла незначительно лучше определять цифры, на которых есть шум, однако в выборке мало размытых объектов, из-за чего данный тип аугментации плохо влияет на модель

Закрытие помогло лучше определять цифры с внутренними "отверстиями" (например, цифра 8)

Примеры изображений с ошибками в новом предикте





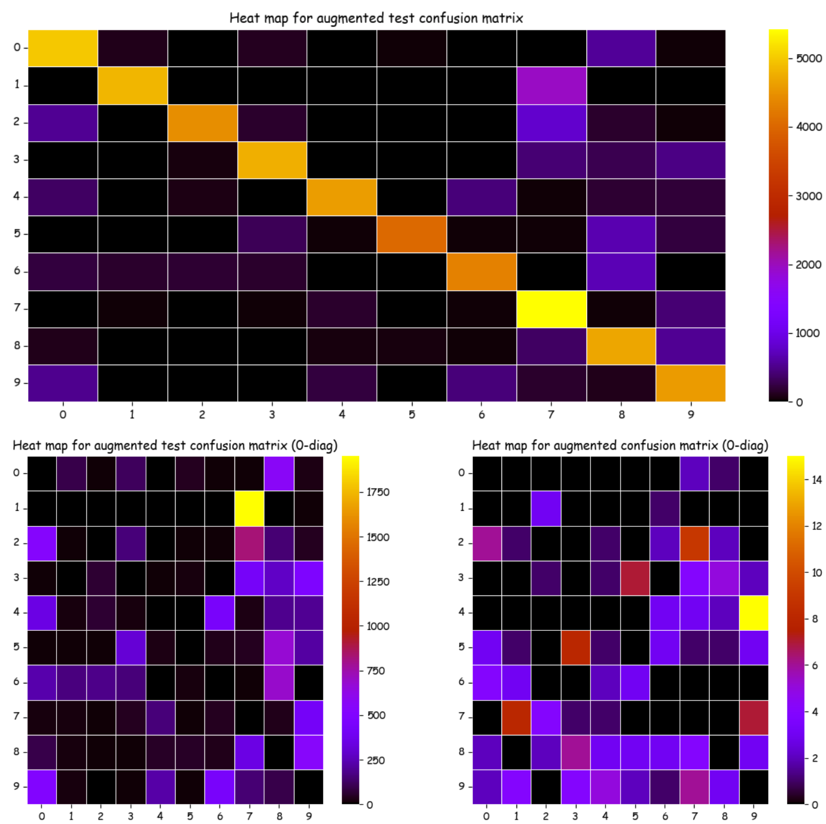


3.6 Результаты шестого эксперимента

Были проведены действия, аналогичные пятому эксперименту. В итоге получена точность 0.7774. Число объектов, на которых была допущена ошибка, получилось равным 13356. В итоге процент ошибок составил 22.26%

Данный подход (в отличие от 5 эксперимента) позволяет взглянуть на то, как модель будет работать на реальных данных, в которых могут быть сильно искажённые цифры. Мы обучаемся на сырых данных без каких-либо аугментаций, используя в качестве тестовой выборки данные, приближённые к реальным.

Heat maps для полученных матриц ошибок



4 Итоговая модель

В итоге получена следующая модель, которая была обучена на аугментированных данных: взвешенный KNN с гиперпараметром $k = 4$, алгоритмом поиска **my_own** и косинусной метрикой. Используемые аугментации:

1. Морфологическое закрытие с ядром $2 \times 2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
2. Поворот на 5 градусов по часовой стрелке
3. Смещение 3 по оси x и 1 по оси y
4. Гауссовский фильтр с дисперсией 1.0
5. Гибрид сразу всех предыдущих трансформаций

Итоговое число объектов в обучающей выборке: $N_{train} = 360000$, а итоговая точность модели составила 0.9822 с процентом ошибок 1.78%

5 Вывод

Был изучен метрический алгоритм классификации *K Nearest Neighbors*. Исследован датасет **MNIST**, изучены две основные метрики - косинусная и евклидова. Выбран лучший алгоритм поиска ближайших соседей на данной выборке **my_own**, основанный на k-й порядковой статистике.

Провелась аугментация данных, были рассмотрены следующие трансформации: сдвиг, поворот, Гауссовский фильтр, дилатация, эрозия, закрытие и открытие.

Получена итоговая модель с точностью 0.9822 с процентом ошибок 1.78%