

# Исследование работы алгоритма KNearestNeighbors на примере датасета MNIST

Тыцкий Владислав

Октябрь 2020

## Введение

Требуется решить задачу классификации с помощью метрического метода KNearestNeighbors(метод K ближайших соседей) на примере известного датасета MNIST.

MNIST - база данных рукописных цифр. Каждая цифра представляется в виде черно-белого изображения  $28 \times 28$  пикселей, что эквивалентно вектору  $x \in \mathbb{R}^{784}$ . Датасет содержит 70000 размеченных цифр. В данном исследовании мы будем использовать обучающую выборку размера 60000, а тестовую соответственно 10000. <sup>1</sup>

## Задание №1

Сравним различные алгоритмы нахождения ближайших соседей — brute, kd\_tree, ball\_tree и my\_own.

my\_own — самописная реализация, которая вычисляет полную матрицу расстояний  $D^{T \times N}$ , где T — размер тестовой выборки, N — размер обучающей выборки.

brute, kd\_tree, ball\_tree — реализации поиска соседей из библиотеки sklearn.

## Сравнение скорости работы

Так как описанные выше методы нахождения соседей являются детерминированными (100% точными), то главным критерием выбора одного из них для дальнейших исследований будет служить скорость работы. В случае MNIST важно знать как хорошо ведут себя алгоритмы в пространстве большой размерности  $\mathbb{R}^{784}$ .

Для экспериментов были выбраны подпространства размерности 10, 50, 100. В качестве меры расстояния возьмем евклидову метрику.

---

<sup>1</sup>В некоторых частях исследования будет использоваться уменьшенная выборка т.к. вычислительная машина Тыцкого В.И. тяжело справляется с такой нагрузкой. Во всех случаях, где не оговаривается иное, будет использоваться полная выборка

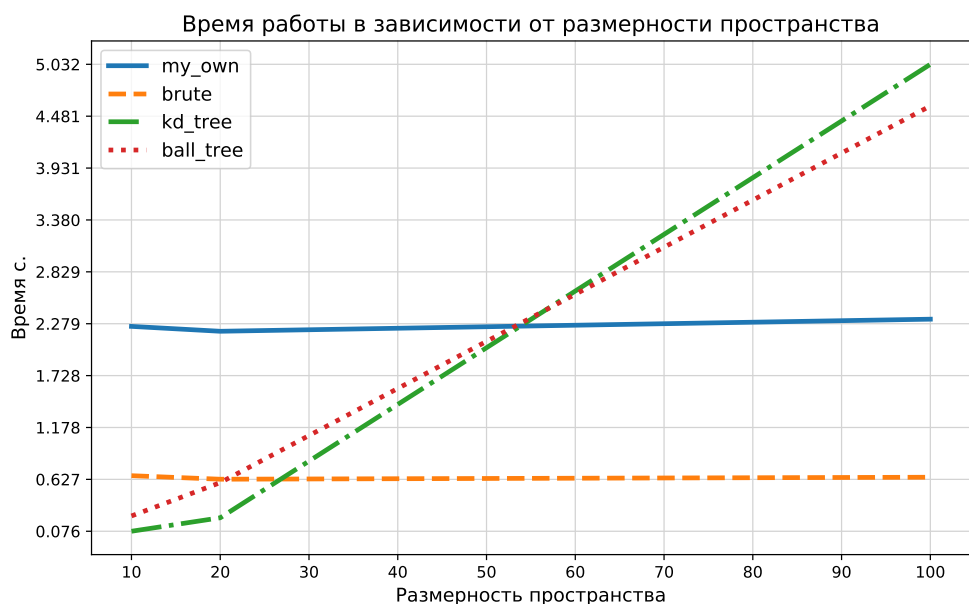


Рис. 1:

На графике (Рис.1) представлены результаты вычисления ближайших соседей для тестовой выборки размера 3000 и 10000 для обучающей выборки.

Из графика время работы kd\_tree и ball\_tree с ростом размерности пространства увеличивается линейно. Это связано с принципами работы алгоритма и так явлением называемым “Проклятие размерности”. Brute и my\_own имеют практически константное время работы алгоритма, потому что основаны на простом построении матрицы расстояний, вычисление нормы разности  $\|x_i - x_j\|$  по сравнению с построением матрицы имеет незначительное количество операций.

Далее везде будем использовать либо brute, либо my\_own <sup>2</sup>

## Задание №2/3

Требуется по кросс-валидации с тремя фолдами оценить точность и время работы в зависимости от следующих факторов:

1. k от 1 до 10 (только точность)
2. евклидова или косинусная метрика
3. используются ли веса или нет (только точность) <sup>3</sup>

Кросс-валидация проводилась на обучающей выборке размером 60000

<sup>2</sup>Кроме евклидовой метрики нам понадобится косинусное расстояние —  $\cos(x, y) = 1 - \frac{(x, y)}{\|x\|_2 \|y\|_2}$ . Sklearn метод, реализующих поиск соседей не поддерживает косинусное расстояние, поэтому часто будем использовать метод my\_own, у которого есть поддержка этого расстояния.

<sup>3</sup> $w_k = \frac{1}{\rho(X, X_k) + 10^{-5}}$ , где w\_k вес K-ого ближайшего соседа X\_k для X

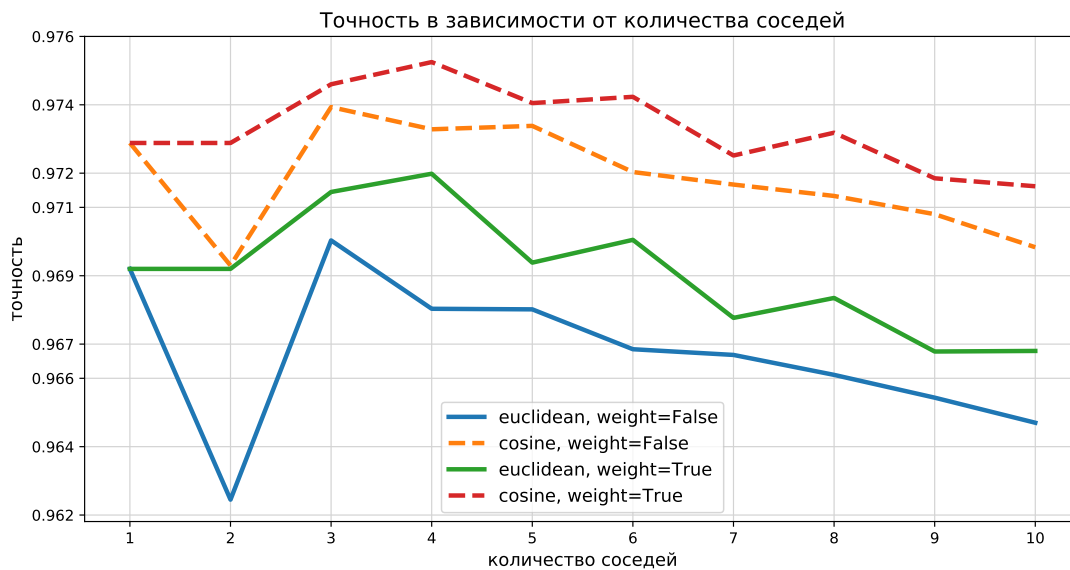


Рис. 2:

На графике (Рис.2) представлены результаты вычисления качества в зависимости от вышеперечисленных параметров

- Косинусное расстояние(с весами и без) лучше евклидова для любых  $k$ .
- Использование весов улучшает качество для любых  $k$ .
- Качество постепенно падает у всех алгоритмов, если  $k > 4$
- Лучшим оказался алгоритм с  $k = 4$  использующий косинусное расстояние и веса. **Его точность: 0.975**

Интересно, что без весов лучшее качество достигается при  $k = 3$ , а с весами при  $k = 4$ . Это говорит о том, что веса в некотором смысле регуляризуют модель — она использует информацию от большего числа соседей, но не “доверяет” слишком далеким объектам.

## Скорость работы

Посмотрим влияет ли метрика на скорость работы алгоритма. В обоих используется `my_own`. Размер обучающей выборки 10000.

test size	euclidean		cosine	
	mean(sec)	std(sec)	mean(sec)	std(sec)
<b>1000</b>	0.917	0.015	0.900	0.001
<b>2000</b>	1.710	0.027	1.796	0.113
<b>3000</b>	2.547	0.021	2.608	0.025

Таблица 1: Сравнение скорости работы в зависимости от метрики

Можно сделать вывод, что метрика не влияет на скорость работы алгоритма. Это вполне логично — обе реализации работают с асимптотикой  $\Theta(n^{2.3727})$  (Алгоритм Копперсмита – Винограда)

## Задание №4

Применим лучший алгоритм (cosine, k=5) для тестовой выборки размером 10000.  
**Точность на тесте: 0.9771**

### Сравнение с лучшими алгоритмами

Рассмотрим таблицу (Таблица 2) лучших алгоритмов для датасета MNIST.

Type	Preprocessing	Error rate (%)
Convolutional neural network	Data augmentation	0.17
Random Multimodel Deep Learning (RMDL)	None	0.18
Convolutional neural network	Expansion of the train data	0.21
Convolutional neural network	Width normalizations	0.23
Convolutional neural network	Expansion of the train data	0.27
Convolutional neural network (CNN)	Expansion of the train data	0.31
Deep neural network	None	0.35
K-Nearest Neighbors	Shiftable edges	0.52
Support-vector machine (SVM)	Deskewing	0.56
Deep neural network	None	0.7
Boosted Stumps	Haar features	0.87
Deep neural network (DNN)	None	1.6
<b>K-Nearest Neighbors (my realisation)</b>	<b>None</b>	<b>2.3</b>
Random Forest	Statistical pixel importance	2.8
Non-linear classifier	None	3.3
Linear classifier	Deskewing	7.6

Таблица 2: Сравнение с лучшими алгоритмами

Даже на бейзлайне мы получаем неплохие результаты. С ассигасу 0.948 в таблице присутствует другой алгоритм KNN [?]

### Ошибки алгоритма

Интересно взглянуть на каких именно объектах наша модель ошибалась. Для этого построим матрицу ошибок(confusion matrix) (Рис.3. Значения на диагонали убраны, в данном случае они нам неинтересны.

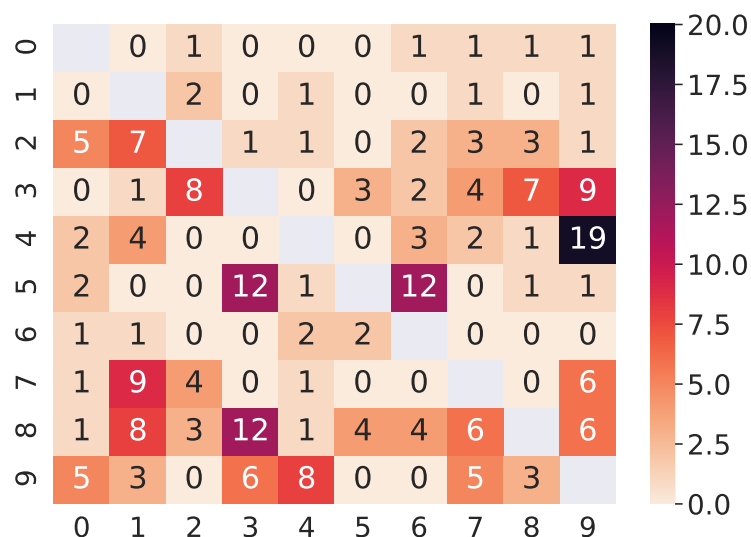


Рис. 3: Матрица ошибок

Можно выделить самые главные причины ошибок - выбросы и непосредственно ошибки модели. Выбросы в данном случае - это такие объекты, которые даже человеку будет тяжело распознать. Эту проблему невозможно исправить никакой моделью. В таблице (Таблица 3) ниже изображено несколько выбросов.

picture									
true label	4	5	5	5	7	8	8	9	9
predict label	9	3	3	8	1	3	3	0	4

Таблица 3: Выбросы

Рассмотрим ошибки модели. Некоторые из них можно было бы решить аугментацией. Расширив обучающую выборку путем изменения исходных объектов (поворот, размытие, сдвиг) мы могли бы лучше решать задачу классификации.

В таблице (Таблица 4) изображены несколько объектов, которые теоретически можно было классифицировать правильно, если бы мы применили к ним фильтр Гаусса.




picture			
true label	0	3	8
predict label	6	2	6

Таблица 4: Фильтр Гаусса

В таблице (Таблица 5) изображены объекты, которые были классифицированы неправильно лишь из-за того, что они повернуты относительно центра изображения слишком сильно. Аугментация могла бы улучшить качество на подобных объектах.







picture						
true label	2	2	3	4	4	6
predict label	4	9	2	1	9	0

Таблица 5: Повороты

Задание №5