

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 28×28 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 10000. ¹

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.

¹В 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

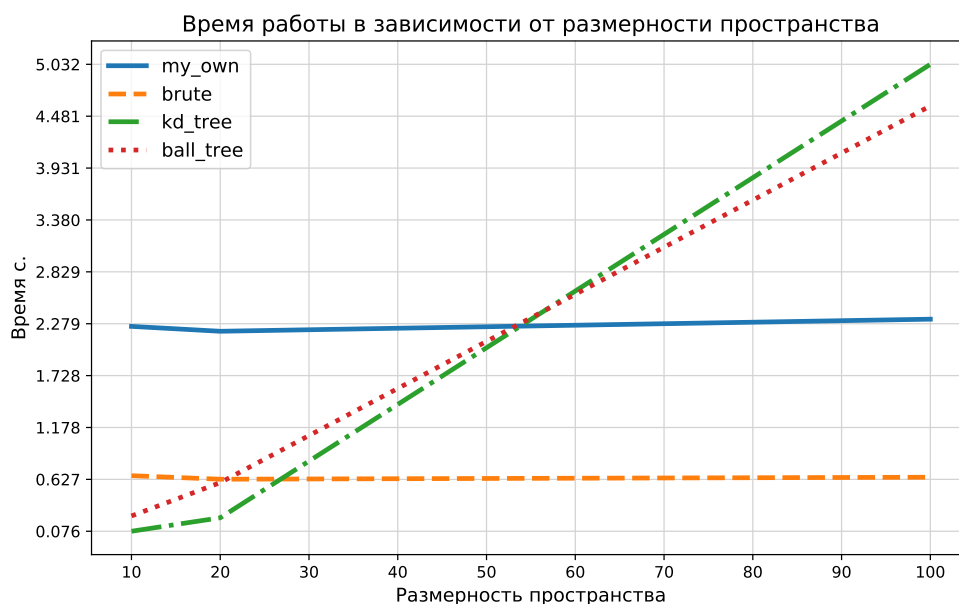


Рис. 1:

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

Время работы kd_tree и ball_tree с ростом размерности пространства увеличивается линейно. Это связано с принципами работы алгоритма и явлением называемым “Проклятие размерности”. Brute и my_own имеют практически константное время работы алгоритма, потому что основаны на простом построении матрицы расстояний, и 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 ²

Задание №2/3

Требуется по кросс-валидации с тремя фолдами оценить точность и время работы алгоритма 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) ³

Кросс-валидация проводилась на обучающей выборке размером 60000

²Кроме 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.

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

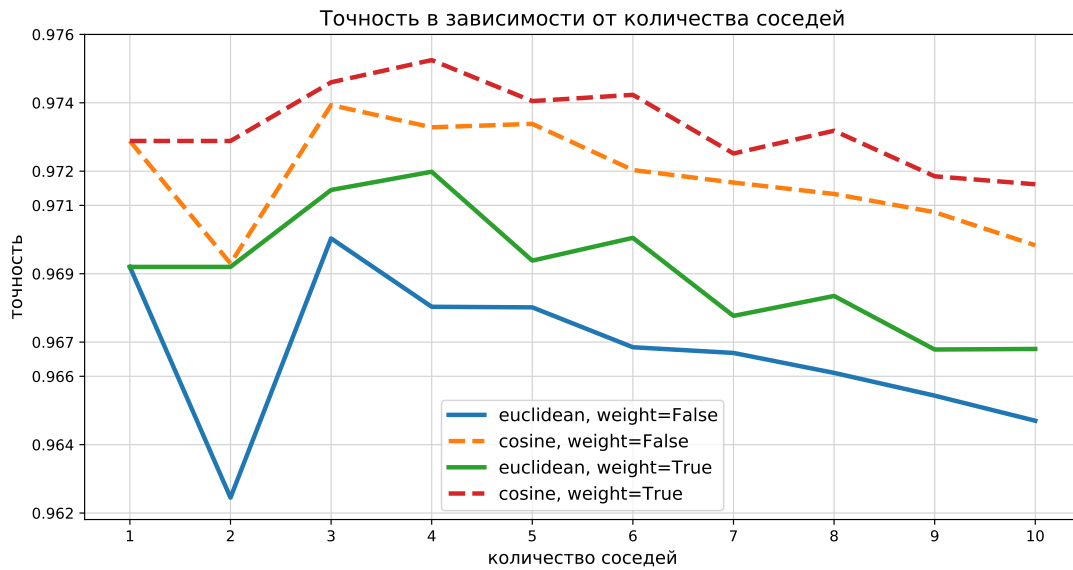


Рис. 2:

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 .
- У всех алгоритмов падает качество при $k > 4$
- 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. Оба используют `my_own`. Размер обучающей выборки 10000.

test size	euclidean		cosine	
	mean(sec)	std(sec)	mean(sec)	std(sec)
1000	0.917	0.015	0.900	0.001
2000	1.710	0.027	1.796	0.113
3000	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). **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
K-Nearest Neighbors (my realisation)	None	2.3
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 с помощью базовой модели мы получаем неплохие результаты. С accuracy 0.9948 в таблице присутствует другой алгоритм KNN [1]

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.

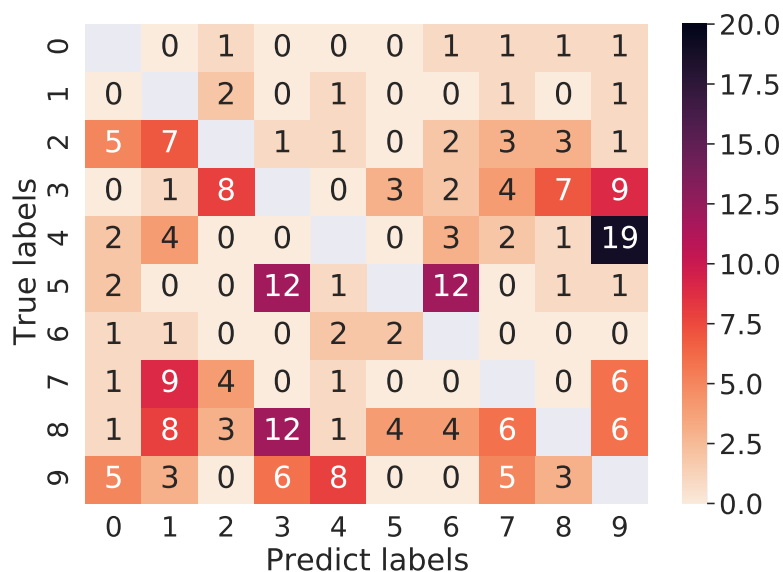


Рис. 3: Матрица ошибок

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 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 слишком сильно повернуты относительно центра изображения. 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

Попробуем улучшить качество нашей модели с помощью аугментации обучающей выборки. Подберем по кросс-валидации лучшие параметры преобразований изображений среди ниже перечисленных:

- rotate deg – -15, -10, -5, 0, 5, 10, 15
- gauss σ – 0.5, 1.0, 1.5
- shift px - -3, -2, -1, 0, 1, 2, 3

Кросс-валидация проводилась на подвыборке размером 40000. В результате отобра лучшие параметры (Таблица 6)

<i>base</i>	<i>rotate, $\pm 10deg$</i>	<i>gauss, $\sigma = 1$</i>	<i>shift_x, $\pm 1px$</i>	<i>shift_y, $\pm 1px$</i>
0.968	0.976	0.976	0.972	0.974

Таблица 6: лучшие параметры

В виду объемности датасета, применим к исходной обучающей выборке только Гауссов фильтр и поворот на -10 градусов (можно было взять и 10). В результате получим обучающую выборку размером 180000.

Точность на тесте: 0.9834

Взглянем на матрицу ошибок после аугментации(Таблица 7):

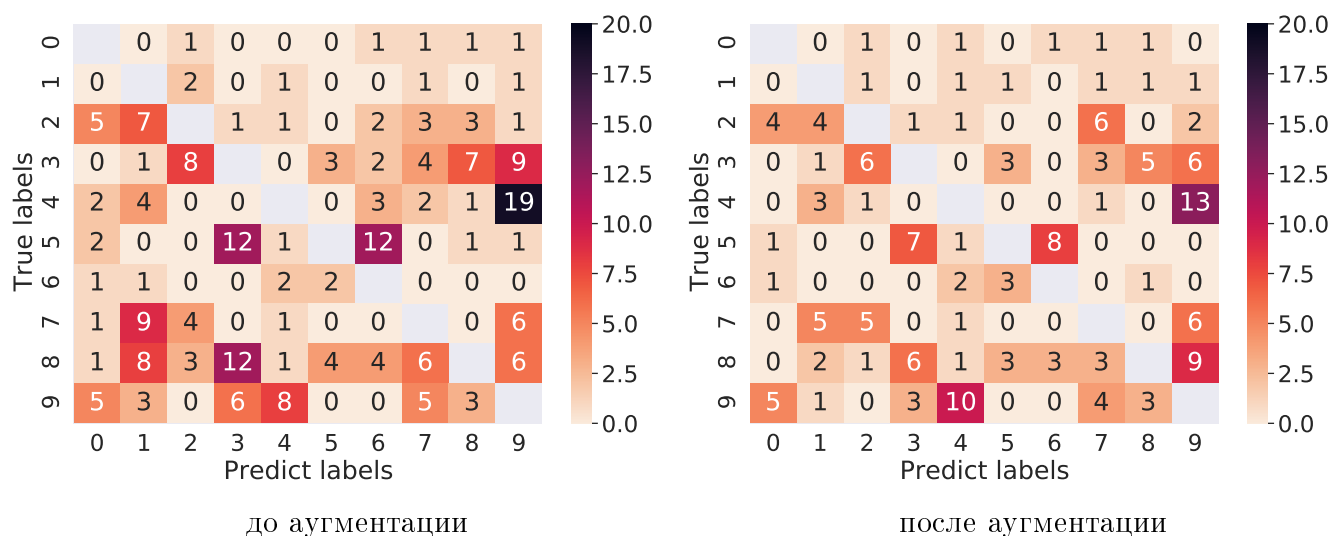


Таблица 7: Изменение матрицы ошибок после аугментации

Даже визуально видно – ошибок стало меньше. Рассмотрим подробнее некоторые объекты, с которыми справился улучшенный алгоритм, а обычный нет(Таблица 8):

picture									
true label	1	2	2	5	7	8	9	9	9
predict (base)	2	1	4	6	9	3	4	4	7
helping transf.	-	r	r	g	r	r	g	g+r	g

Таблица 8: Объекты, которые удалось правильно классифицировать(aug_train)

Аугментация определенно помогает увеличить качество, но можно заметить, что на некоторых клетках матрицы ошибок значения стали больше. В идеале к аугментации нужно подходить более “умно”, возможно стоило бы размножить объекты из определенного класса или научиться понимать в какую сторону лучше повернуть конкретный объект.

Задание №6

Аугментацию можно проводить не только над обучающей выборкой, но и над тестовой. Для этого необходимо найти ближайших соседей обыкновенной тестовой выборки, а также для преобразованной тестовой выборки. Далее можно поступить по-разному, но в данном случае среди двух наборов соседей будем искать k самых

ближайших. По кросс-валидации на подвыборке 25000 лучшими оказались параметры $shift_x = -1$ и $shift_y = -1$.

Точность на тесте: 0.98

Взглянем на матрицу ошибок после аугментации тестовой выборки (Таблица 9)— в некоторых ячейках матрицы ошибок стало сильно меньше ((4,9), (3,9)), но в некоторых местах заметно сильное увеличение количества ошибок по сравнению с базовой моделью ((7,1), (8,3)).

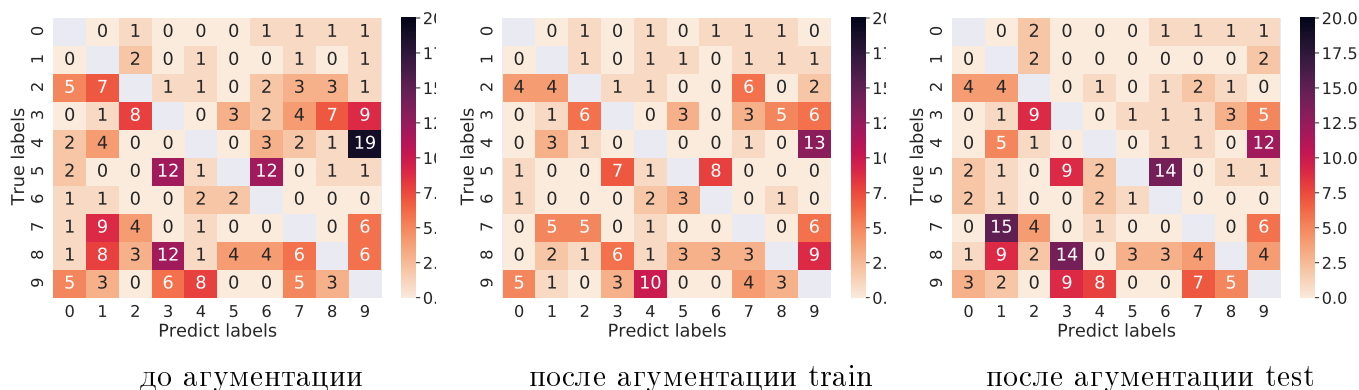


Таблица 9: Изменение матрицы ошибок после аугментции

В Таблице 10 представлены некоторые объекты, с которыми справился алгоритм с аугментацией теста, а базовый ошибся. Сложно сказать из-за какого именно преобразования (сдвиг влево, сдвиг вниз) эти объекты были классифицированы правильно.

picture							
true label	2	7	9	4	5	6	8
predict (base)	0	9	0	9	6	5	2

Таблица 10: Объекты, которые удалось правильно классифицировать(aug_test)

В Таблице 11 представлены некоторые объекты, с которыми справился алгоритм с аугментацией теста, а алгоритм с аугментацией обучающей выборки нет:

picture							
true label	1	2	4	5	7	8	9
predict (aug_train)	5	0	9	6	9	4	0

Таблица 11: Объекты, которые удалось правильно классифицировать(aug_train vs aug_test)

Список литературы

- [1] Keysers D. et al. Deformation models for image recognition //IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2007. – Т. 29. – №. 8. – С. 1422-1435.