Отчет по заданию №1: Метрические алгоритмы классификации

Васильев Руслан ВМК МГУ, 317 группа

15 ноября 2020 г.

Содержание

Введение

В рамках задания требовалось реализовать метод k-ближайших соседей, инструменты для кросс-валидации и функции для вычисления попарных расстояний. В данном отчете описывается ход и результаты экспериментов, произведенных с помощью написанных программ на датасете MNIST. Далее, согласно заданию, предполагается следующее разбиение: первые 60 000 объектов (обучающая выборка) и последние 10 000 (тестовая выборка).

Какой алгоритм работает быстрее? (№1)

В первом эксперименте сравнивается время работы доступных алгоритмов реализованного классификатора. Для этого производится поиск 5 ближайших соседей для каждого объекта тестовой выборки, по евклидовой метрике. Размер

блока для вычислений — 1000 объектов, причем этот параметр используется только для методов, которые строят в памяти матрицу попарных расстояний (`my_own` и `brute`).

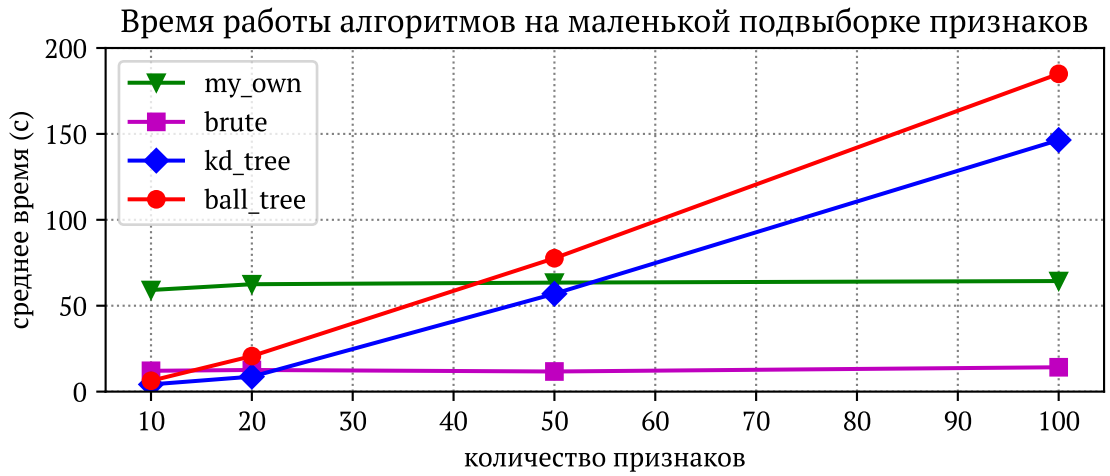

Рис. 1

На рис. 1 видно, что деревья имеют преимущество только в случае малого числа признаков — в данном случае 10 для обоих алгоритмов, а также для 20 в случае `kd_tree`. Сравнивая сами деревья, видим, что Ball tree уступает KD tree. На время работы переборных реализаций количество признаков почти не влияет. Посмотрим, что произойдет, если учитывать еще больше доступных признаков.

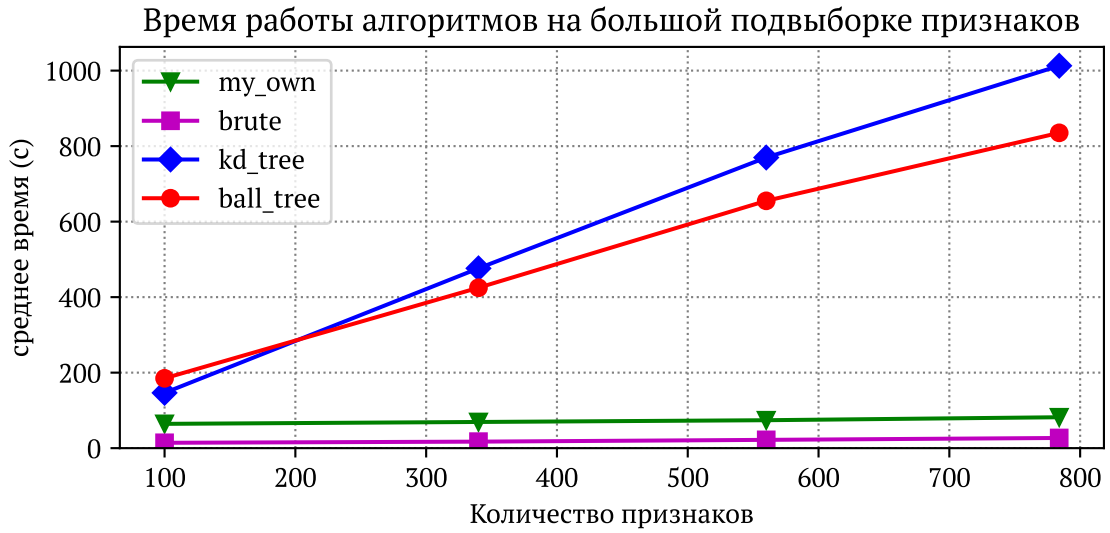

Рис. 2

Рис. 2 показывает, время работы алгоритмов, основанных на деревьях, сильно возрастает с ростом числа признаков. Это неочевидный результат, поскольку в теории время поиска ближайшего соседа имеет (в среднем) время работы зависит логарифмически от размера обучающей выборки, в отличие от полного перебора, где зависимость уже линейная. Такую асимптотику нужно интерпретировать с осторожностью — в ней не учитывается размерность пространства признаков.

Для действительно логарифмической сложности поиска в дереве необходимо потребовать его сбалансированность и равномерное распределение точек в пространстве, что существенно зависит от размерности признакового пространства в случае `kd_tree` и `ball_tree`. Причина невыполнения этих свойств — проклятие размерности: 60 000 даже равномерно распределенных объектов недостаточно для равномерного покрытия по нескольким сотням признаков.

Наконец, время работы `brute` и `my_own` с ростом признаков по-прежнему меняется незначительно, что связано с векторизацией вычислений попарных расстояний. Далее будем использовать только эти алгоритмы, поскольку, как мы показали выше, деревья в данной задаче работают неэффективно.

Какую использовать метрику и нужны ли веса? (№2, №3)

Целью данного эксперимента, кроме ответа на поставленные вопросы, является измерение времени кросс-валидации, а также подбор оптимального значения k — числа ближайших соседей.

Разумным функционалом качества для данной задачи является точность — доля верных предсказаний классификатора. Валидация происходит на 3-фолдах: соседи ищутся в пределах от 1 до 10, в качестве метрик рассматриваются евклидова и косинусная.

В эксперименте использовались алгоритмы и `my_own`, и `brute`, но, как и следовало ожидать, при одинаковых гиперпараметрах они дают одинаковые предсказания — поэтому результаты в измерении точности оказались равными.

Кросс-валидация показывает, что *косинусная метрика дает более точные предсказания, чем евклидова, и использование весов также улучшает качество классификации*. Отметим, что весовые коэффициенты в метрических алгоритмах можно использовать по-разному: они могут зависеть от номера ближайшего соседа, от расстояния до него и от выбора самой функции. В данной реализации голос одного сосседа полагается равным $\frac{1}{d+10^{-5}}$, где d — расстояние между объектом и рассматриваемым соседом.

Также рис. 3 показывает, что оптимально использование метода 4-х ближайших соседей. Четность не мешает как раз из-за весов.

Итак, лучшая средняя точность по кросс-валидации составляет **97.41%** (в тексте отчета будет использоваться выражение точности процентах), которая получается с использованием взвешенного метода $k = 4$ ближайших соседей и косинусной метрикой.

Также по заданию требовалось измерить время поиска соседей в процессе кросс-валидации. Эффективная реализацию заключается в том, чтобы сразу посчитать расстояния до максимального исследуемого числа соседей, а затем последовательно «отбрасывать лишних». Соответствующий результат приведен на рис. 4.

Как и следовало ожидать, единственное трудоемкой операцией является поиск 10 соседей, дальнейшие же вычисления не времязатратны. Наложение кривых связано с одинаковой сложностью вычислений матриц попарных расстояний для

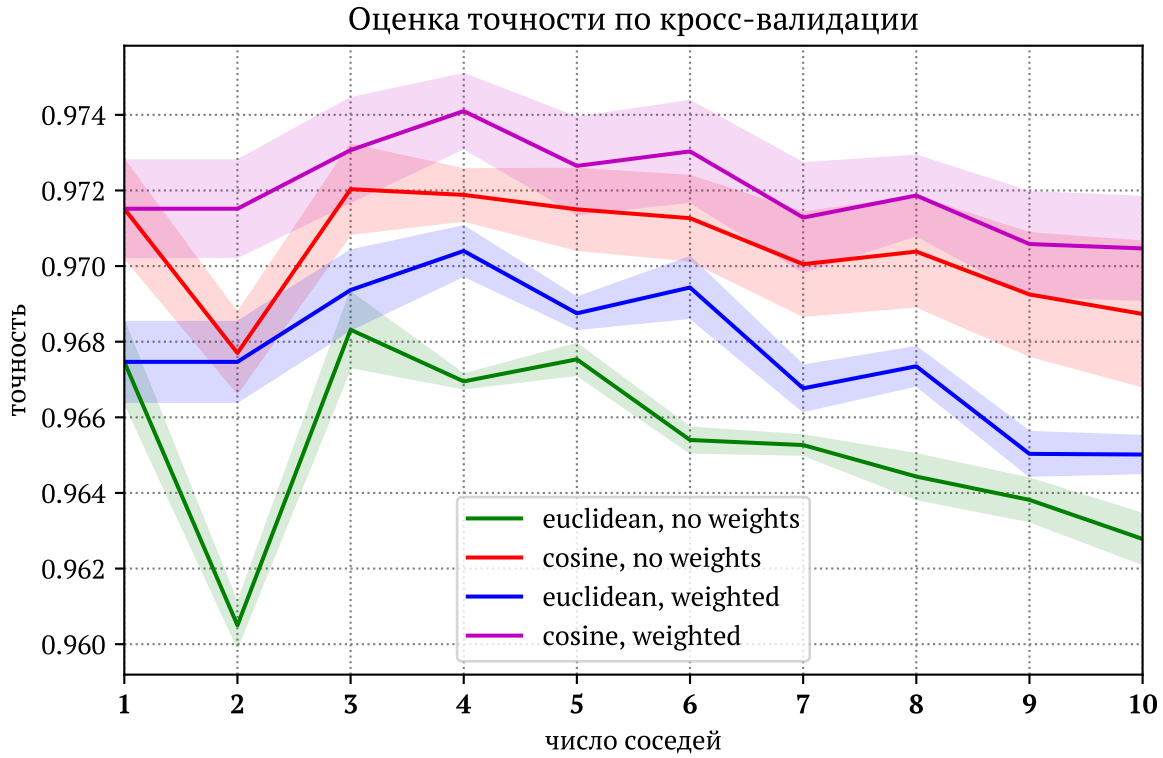

Рис. 3: Полупрозрачным обозначено стандартное отклонение

евклидовой и косинусной метрик. Подобно достаточно известной оптимизации для евклидовой метрики [1], следующее очевидное преобразование позволило быстрее посчитать и матрицу попарных расстояний для косинуса:

$$\rho_{cos}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = 1 - \left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|}, \frac{\mathbf{y}}{\|\mathbf{y}\|} \right\rangle \tag{1}$$

То есть достаточно сначала произвести нормировку признакового описания объектов, а затем умножить матрицы. Такой подход, использованный в реализации библиотеки scikit-learn [2], использует только $(l_1 + l_2) \cdot d$ делений, в то время как нормировка после матричного перемножения потребует $O(l_1 \cdot l_2)$ делений, где d — размерность признакового пространства, а l_1 и l_2 — число объектов (в нашем случае строк) в соответственно первой и второй матрице.

Качественная ли получилась модель? (№4)

В данном эксперименте мы обучим модель с полученными гиперпараметрами на всей обучающей выборки и проанализируем предсказания на тестовой.

Итак, качество на тесте — **97.52%**, что превосходит среднюю точность той же модели на кросс-валидации (97.41%). Различие не столь значительное, хотя обычно точность на тесте получается ниже. На самом деле точность на кросс-валидации зависит от способы разбиения на фолды: можно было бы делать равные пропорции классов, случайно перемешивать, брать другое число подвыборок. В

Время поиска k ближайших соседей на кросс-валидации

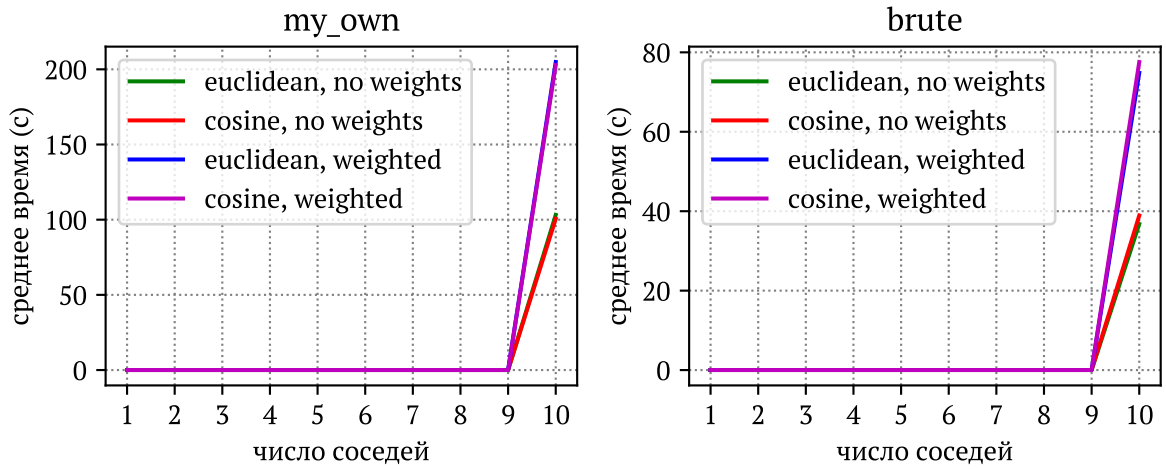

Рис. 4

нашей реализации используется стандартное последовательное разбиение на фолды, поскольку датасет MNIST уже перемешан.

Согласно рейтингу [3], лучшее качество в известных работах — **99.84%** [4], результат 2020-го года. Авторы используют сверточные нейронные сети, предсказания которых улучшются с помощью аугментации и специальных методов ансамблирования.

Вернемся к нашей модели.

истинный класс	0	1	2	3	4	5	6	7	8	9
число ошибок	**3**	6	23	34	36	29	10	30	38	**39**
% среди ошибок	**1.21**	2.42	9.27	13.71	14.52	11.69	4.03	12.10	15.32	**15.73**
% ошибок в классе	**0.31**	0.53	2.23	3.37	3.67	3.25	1.04	2.92	**3.90**	3.87

Таблица 1: Статистики ошибок полученной модели

Построенная матрица ошибок (рис. 5), а также табл. 1 позволяют сделать следующие наблюдения:

- Лучше всего модель классифицирует ***0*** и ***1***. Это связано прежде всего с «простотой» данных символов.
- Самый частый вид ошибки — подмена ***4*** (истина) на ***9*** (предсказание), а также ***7*** на ***9***. Обратные подстановки происходят сравнительно реже. И в основном ошибки не симметричны.
- Тем не менее алгоритм нередко путает ***3*** и ***5*** друг с другом.

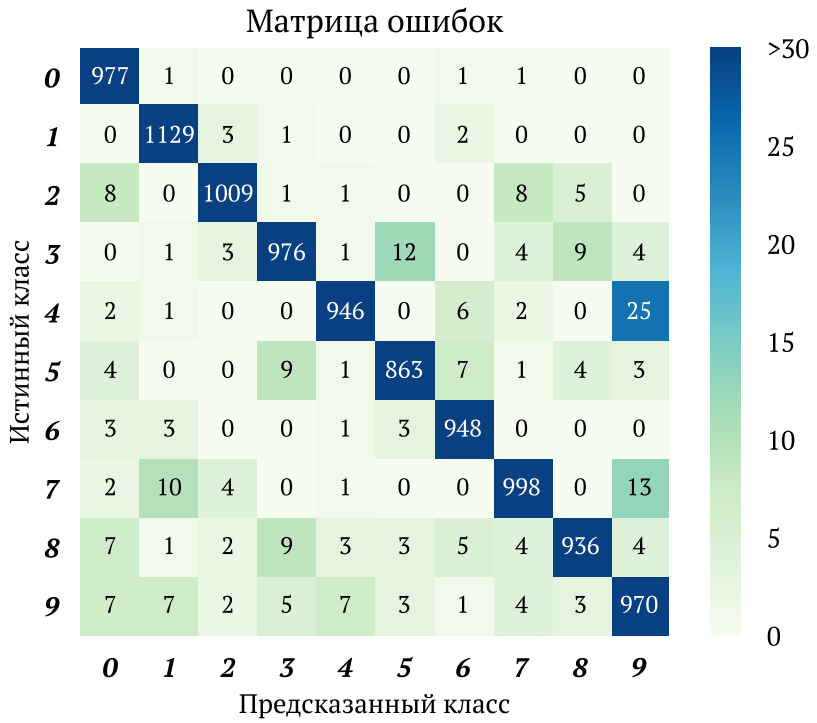

Рис. 5

- Чаще всего алгоритм ошибается на ***8*** и ***9***. Причем ***9*** еще и самая частая подмена.

Среди ошибочных классификаций выделяются разные группы с общими свойствами. Так, например, на рис. 6 видно, что некоторые цифры имеют выраженный поворот покруг центра.

Цифры на некоторых изображениях (рис. 7) могут двояко классифицироваться даже человеком.

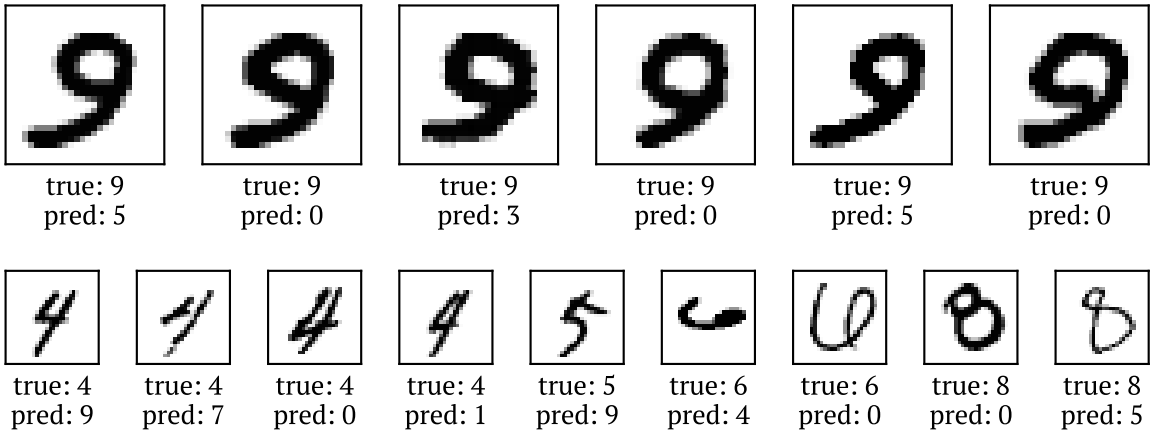

Рис. 6: «Повернутые» цифры

Другие (рис. 8) цифры просто слишком жирно написаны, и их бы следовало несколько размыть.

Наконец, в датасете есть совсем необычно (рис. 9) написанные цифры.

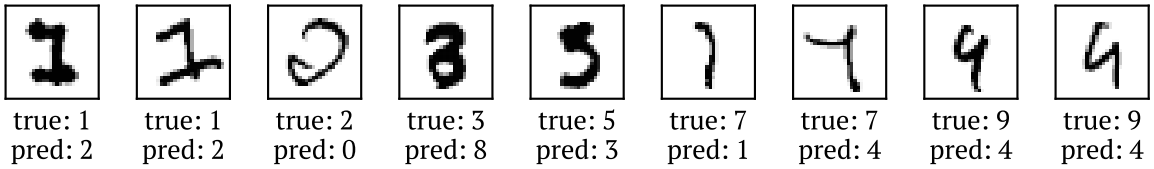

Рис. 7: Цифры, допускающие несколько классификаций

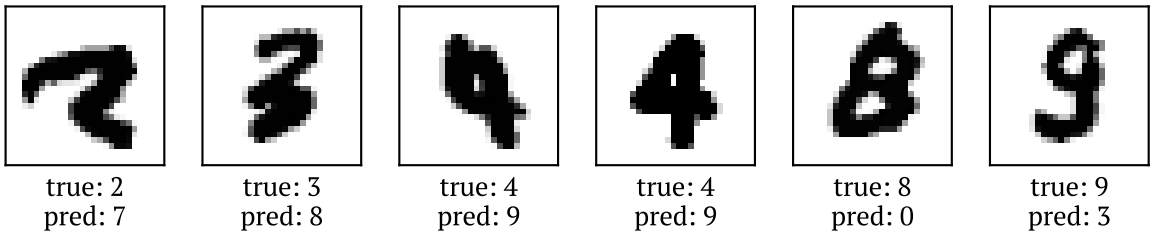

Рис. 8: Слишком жирный шрифт

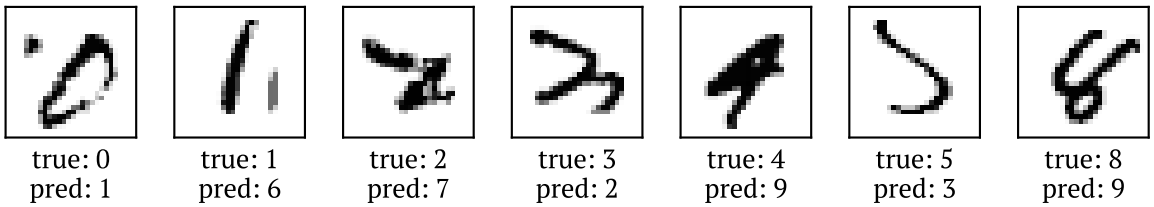

Рис. 9: Трудно классифицируемые цифры

Аугментация данных (№5)

В данном эксперименте требовалось увеличить количество объектов обучающей выборки посредством поворотов, смещений и применения фильтра Гаусса. Функции, связанные с обработкой изображений и аугментацией, реализованы в модуле `augmentation.py`. Из-за удобного API в качестве библиотеки для работы с изображениями используется модуль `scipy.ndimage`. Но его функции принимают на вход единственное изображение, поэтому были написаны распараллеленные (с помощью `joblib` функции трансформации выборки.

С изменением числа объектов в обучающей выборки может измениться оптимальное количество ближайших соседей. Поэтому в процессе кросс-валидации исследуются не только разные параметры трансформаций, но и зависимость от гиперпараметра k.

Преобразования выборки организованы следующим образом: повороты на $\pm 5°$, $\pm 10°$, $\pm 15°$; сдвиги на ± 1, ± 2, ± 3, пикселя по горизонтали и аналогично по вертикали; размытие с помощью фильтра Гаусса с параметром σ, равным 0.5, 1 или 1.5. Например, в случае сдвигов одна размноженная обучающая выборка может состоять из втрое большего числа объектов — сдвиги берутся в паре по одной оси.

Результаты кросс-валидации приведены на рис. 10. Для большей ясности дополнительно пунктиром приводится график лучшей неразмноженной модели из предыдущих экспериментов, гиперпараметры которой используются в моделях с аугментацией. Полупрозрачным цветом по-прежнему обозначается стандартное отклонение.

Кросс-валидация для аугментации

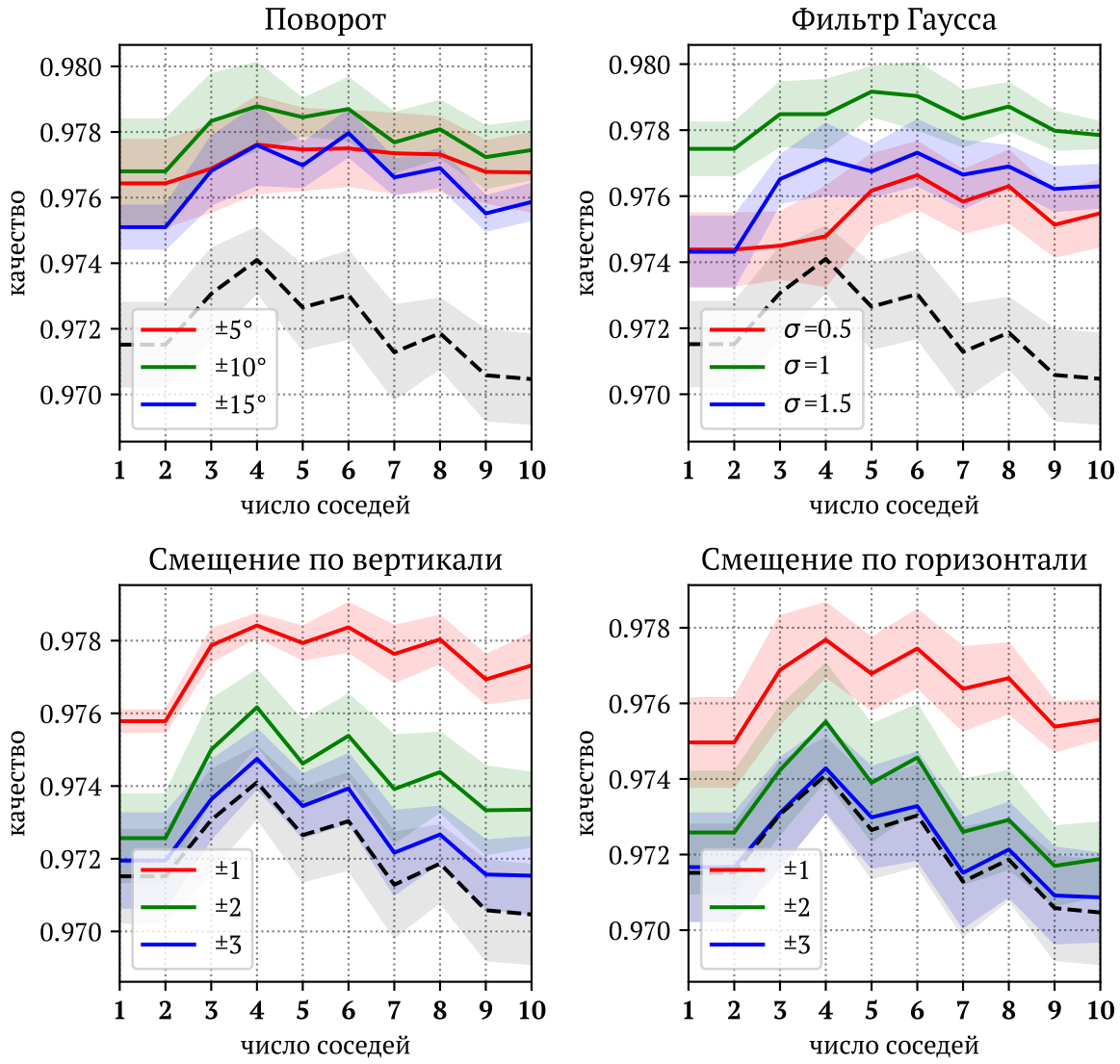

Рис. 10

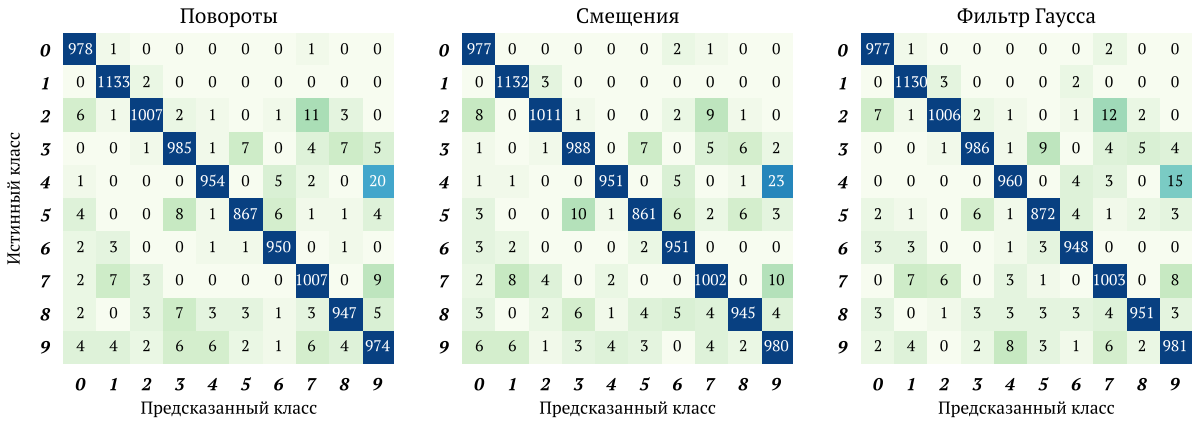

Рис. 11

Теперь мы можем определить лучшие параметры трансформаций: поворот на $\pm 10^\circ$, смещение на ± 1 пиксель и $\sigma = 1$ в фильтре Гаусса. Далее мы объеденим сдвиги по разным осям.

Кросс-валидация показывает, что несмотря на размножение объектов обучающей выборки, параметр $k = 4$ остается достаточно оптимальным (хотя при некоторых преобразованиях лучшим оказываются 5 или 6).

Изменение точности на тестовой выборки при каждом типе преобразований (с лучшими параметрами) и их комбинациями показано в табл. 2. Лучшее качество достигается при сочетании сдвигов и фильтра Гаусса — **98.36%**. Тем не менее использование поворотов дало наиболее интерпретируемый результат — чуть больше половины ошибок, показанных на рис. 6, были исправлены именно с помощью поворотов объектов обучающей выборки. Размытие фильтра Гаусса исправило часть (3 из 6) ошибок, допущенных на чересчур жирных цифрах (рис. 8). Смещения, исправили ошибки на недостаточно центрированных объектах.

Что касается изменения матрицы ошибок рис. 11, то интересно что не все подмены были исправлены трансформациями: например, ***2*** стала только чаще называться ***7*** в любой форме аугментации. Фильтр Гаусса уменьшил число самой распространенной ошибки — подмены ***4*** на ***9***, а также ***7*** на ***9***. Повороты достаточно равномерно уменьшили число ложных предсказаний ***0***. А ***2*** и ***4*** лучше всего предсказываются при смещениях.

Таким образом, с помощью аугментации можно исправлять ошибки алгоритма, причем различиные преобразования влияют на качественно разные ошибки.

Аугментация наоборот? (№6)

Если в предыдущем эксперименте мы размножали объекты обучающей выборки, то в заключительном мы размножим объекты теста. Такой метод часто называют test time augmentation (TTA) [5]. В нашем случае идея заключается в поиске расстояний при различных преобразованиях тестовой выборки. Далее с этими расстояниями можно поступать по-разному. Первым вариантом было усреднение, но в ходе кросс-валидации оказалось, что такой подход ухудшает модель. Более качественный результат дало не усреднение, а выбор *наименьшего* из полученных расстояний — такая эвристика согласуется с идеей метода k ближайших соседей.

Выбор параметров трансформаций производился по той же схеме, что и при аугментации. Результаты можно видеть на рис. 12. Лучшие варианты для поворотов и смещений остались без изменений, а вот фильтр Гаусса во всех случаях ухудшал качество. Для единообразия возьмем $\sigma = 1$.

Как и в случае аугментации, повороты снова помогают исправить ошибки на «повернутых» изображениях рис. 6, а фильтр Гаусса в TTA справился с задачей рис. 8 хуже.

Лучшая точность — **98.35%** достигается в случае комбинации поворотов и сдвигов.

На матрице ошибок рис. 13 особенно хорошо видно, как фильтр Гаусса ухудшает модель в случае TTA. Причина может быть в том, что его применение призвано уменьшить шум, что в случае аугментации действительно полезно. «Стирая» же

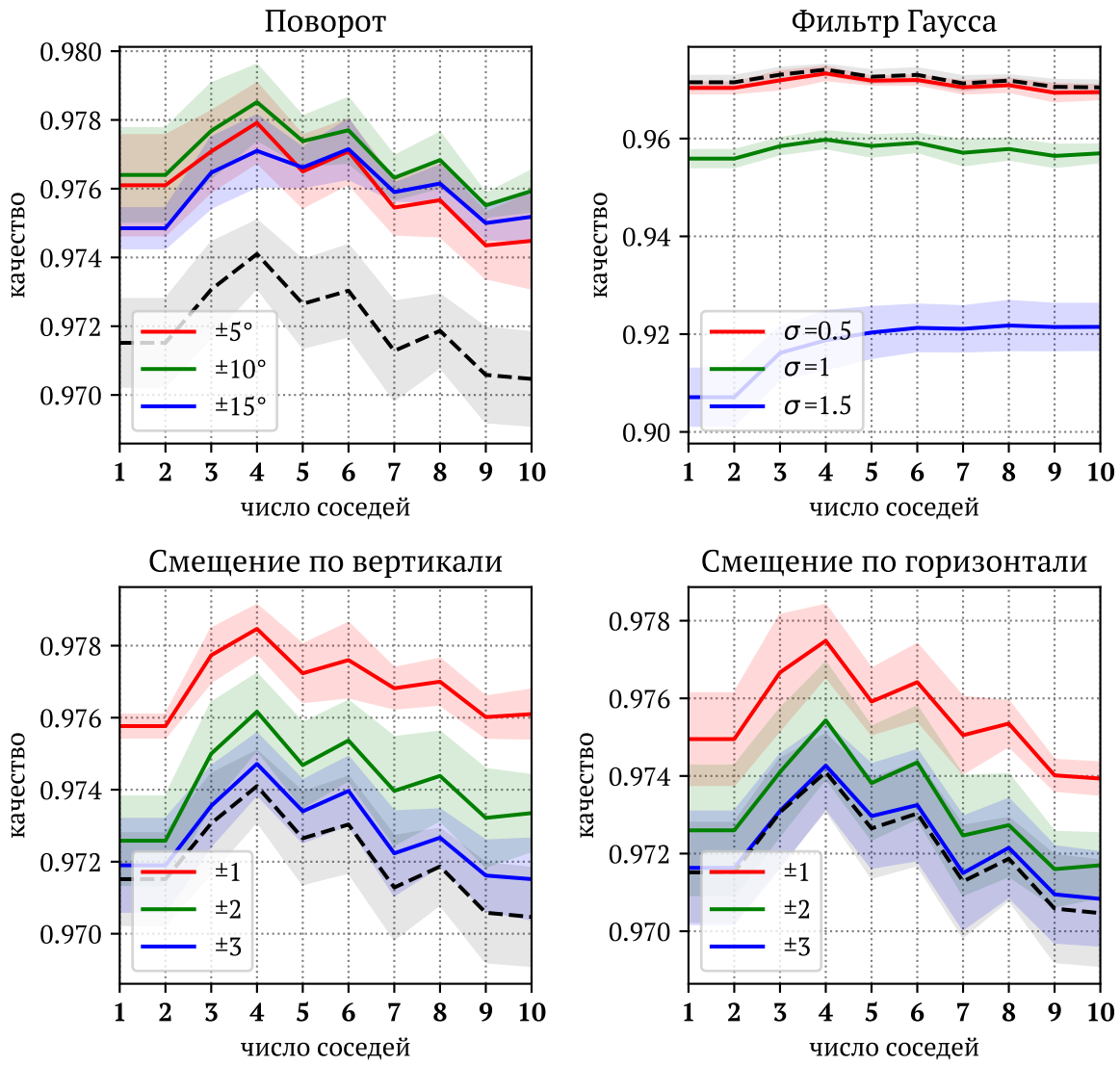
Кросс-валидация для TTA
Поворот
качество
число соседей
±5°
±10°
±15°
Фильтр Гаусса
σ=0.5
σ=1
σ=1.5
Смещение по вертикали
±1
±2
±3
Смещение по горизонтали

Рис. 12

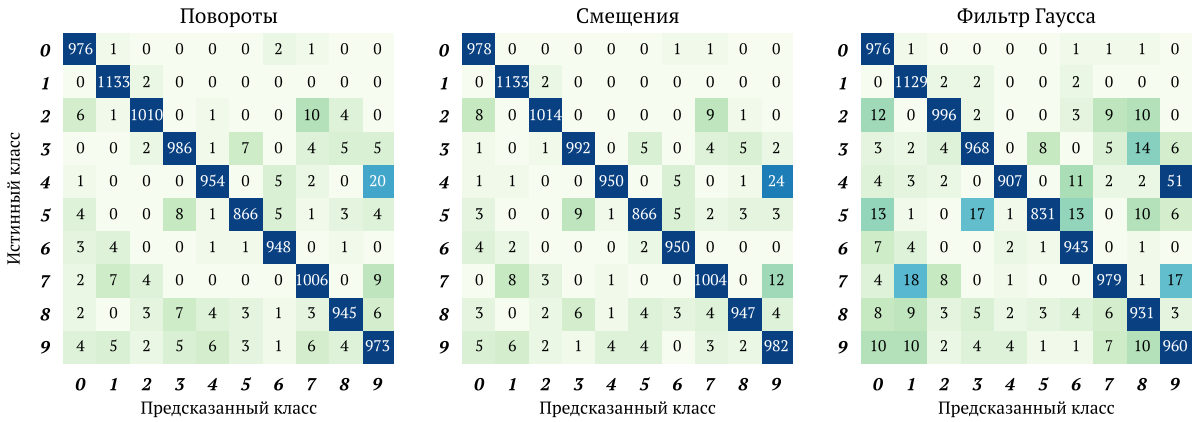
Повороты
Смещения
Фильтр Гаусса
Истинный класс
Предсказанный класс

Рис. 13

часть информации с классифицируемого объекта, мы теряем часть информации и делаем его еще более похожим на другие классы. В то же время, и смещения, и повороты действуют в TTA схожим с аугментацией образом. Однако аугментация более эффективно справляется со сдвигами, в то время как TTA — с поворотами. Финальное сравнение точности двух методов на тестовой выборке при различных комбинациях преобразований приведено на табл. 2.

	повороты $\pm 10^\circ$	сдвиги $(\pm 1, \pm 1)$	фильтр Гаусса $\sigma = 1$	повороты + сдвиги	повороты + Гаусс	сдвиги + Гаусс	повороты, сдвиги, Гаусс
AUG	98.02	97.98	98.14	98.30	98.33	**98.36**	98.30
TTA	97.97	98.16	96.20	**98.35**	96.53	96.52	96.52

Таблица 2: Сопоставление подходов №5 и №6. Точность модели в процентах. Для сравнения, точность, полученная в эксперименте №5 составила **97.52%**.

Заключение

Таким образом, на примере классификации изображений мы провели целый ряд экспериментов с *kNN*. В нашей задаче лучшим образом себя проявила косинусная метрика, которая применяется не столь часто. Были сопоставлены два различных метода размножения выборок, и хотя оба имели свои преимущества, лучший результат показала аугментация, основанная на преобразованиях смещения и фильтра Гаусса. Итоговое качество — **98.36%**.

Параллелизм

В качестве небольшого дополнения отметим роль параллельных вычислений в метрических алгоритмах. Действительно, поиск ближайших соседей может проходить независимо, поэтому нет причин не использовать всю вычислительную способность компьютера. Единственное препятствие — рамер оперативной памяти, но библиотека `joblib`, использованная в реализации модулей, позволяет также использовать отображения фрагментов диска. Таким образом, получается эффективно работать с отдельными сегментами матрицы попарных расстояний, не храня ее полностью в оперативной памяти. Параллельная реализация поиска ближайших соседей написана в методе `KNNClassifier._fkn_parall`. Другое полезное использование параллельных вычислений — трансформации изображений (`augmentation.py`).

References

[1] Samuel Albanie. *Euclidean Distance Matrix Trick*. Visual Geometry Group, University of Oxford. 2019.

[2] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[3] *Image Classification on MNIST*. URL: https://paperswithcode.com/sota/image-classification-on-mnist. accessed: 18 October 2020.

[4] Adam Byerly, Tatiana Kalganova, and Ian Dear. *A Branching and Merging Convolutional Network with Homogeneous Filter Capsules*. 2020. arXiv: 2001.09136 [cs.CV].

[5] Nathan Hubens. *Test Time Augmentation (TTA) and how to perform it with Keras*. URL: https://towardsdatascience.com/test-time-augmentation-tta-and-how-to-perform-it-with-keras-4ac19b67fb4d.