Отчет о выполненной работе «Метрические алгоритмы классификации»

Федоров Илья Сергеевич
курс «Практикум на ЭВМ» ММП ВМК МГУ

14 октября 2019 г.

Постановка задачи

В качестве задачи предлагалось реализовать на языке Python классический метрический алгоритм классификации - метод k ближайших соседей, а также провести серию экспериментов для исследования зависимости времени работы алгоритма и качества классификации от метода поиска ближайших соседей, гиперпараметров алгоритма и размера входных данных. В качестве исследуемого набора данных использовалась база образцов рукописных цифр MNIST, разбитая на обучающую выборку (60 тыс. объектов) и тестовую (10 тыс. объектов).

Время работы

Рассмотрим время работы алгоритма в зависимости от метода поиска ближайших соседей и количества признаков (признаки выбирались случайным образом).

Метод	**Кол-во признаков**	**Время работы, с**
my_own	10	100.7
my_own	20	103.1
my_own	100	141.0
brute	10	10.9
brute	20	10.7
brute	100	12.2
kd_tree	10	2.5
kd_tree	20	4.2
kd_tree	100	86.9
ball_tree	10	3.2
ball_tree	20	9.7
ball_tree	100	76.1

Таблица 1: Измерения времени работы алгоритма, k = 5

Прежде всего заметим, что реализованный своими руками метод поиска ближайших соседей `my_own` работает медленнее, чем предоставляемый библиотекой `sklearn` метод `brute`, работающий по аналогичному алгоритму: вычисление матрицы попарных расстояний. Было установлено, что `my_own` теряет в производительности на моменте сортировки указанной матрицы. Этот результат вполне ожидаем: действительно, зачастую, количество ближайших соседей k значительно меньше количества объектов в обучающей выборке, поэтому логично предположить, что полная сортировка является излишней операцией, и существуют используемые `brute` эффективные методы поиска k первых порядковых статистик в массиве. Таким образом, приходим к гипотезе о том, что при k близком к количеству объектов в обучающей выборке, время работы метода `brute` будет ближе к времени работы метода `my_own`. Рассмотрим экспериментальные данные.

Метод	**Кол-во признаков**	**Время работы, с**
my_own	10	99.5
my_own	20	103.2
my_own	100	136.1
brute	10	57.1
brute	20	57.1
brute	100	59.1

Таблица 2: Измерения времени работы алгоритма, k = 60000

Действительно, мы видим, что при увеличении гиперпараметра k время работы метода `brute` увеличивается и становится сравнимым по порядку с временем работы метода `my_own`. Таким образом, наша гипотеза оказалась верна.

Другим важным выводом из таблицы 1 является тот факт, что для признаковых пространств высокой размерности (в данном случае 100), структуры данных для их разбиения на области, становятся малоэффективными. А именно, методы `kd_tree` и `ball_tree`, использующие одноименные структуры данных, начинают показывать результаты, лишь незначительно превосходящие «наивный» алгоритм поиска ближайших соседей. Более того, известно [1], что не существует структур данных, которые существенно ускоряют поиск ближайших соседей в пространствах высоких размерностей. В этом случае следует использовать приближенные методы.

Оценка качества классификации на кросс-валидации

Проведем оценку качества классификации алгоритма с помощью кросс-валидации. Валидироваться будем на трех фолдах, а исследуемыми гиперпараметрами будут:

- Число ближайших соседов k (изменяется от 1 до 10)
- Метрика (евклидова или косинусная)

В качестве метода поиска ближайших соседей будем использовать `brute`, поскольку, как мы убедились прежде, структуры данных не имеют существенного влияния на время работы для пространств высокой размерности (а сейчас мы имеем 784 признака). Измерять качество будем по метрике `accuracy`, т. е. по доле правильных ответов. Для фиксированной метрики кросс-валидация выполняется примерно 90-100 секунд. Поскольку для каждой метрики перебирается 10 значений параметра k, проход кросс-валидации для трех фолдов занимает около 10 секунд. Рассмотрим результаты работы кросс-валидации:

k	**1-й фолд**	**2-й фолд**	**3-й фолд**	**Среднее арифметическое**
1	0.968	0.966	0.966	0.967
2	0.968	0.966	0.966	0.967
3	0.970	0.969	0.968	0.969
4	0.970	0.969	0.970	0.970
5	0.969	0.968	0.968	0.968
6	0.970	0.968	0.969	0.969
7	0.967	0.965	0.967	0.966
8	0.967	0.966	0.967	0.967
9	0.965	0.964	0.965	0.965
10	0.964	0.964	0.965	0.965

Таблица 3: Результаты работы кросс-валидации, евклидова метрика

k	**1-й фолд**	**2-й фолд**	**3-й фолд**	**Среднее арифметическое**
1	0.973	0.970	0.970	0.971
2	0.973	0.970	0.970	0.971
3	0.974	0.972	0.971	0.972
4	0.975	0.973	0.973	0.974
5	0.973	0.970	0.972	0.972
6	0.974	0.971	0.973	0.973
7	0.972	0.969	0.971	0.970
8	0.972	0.970	0.972	0.971
9	0.970	0.968	0.971	0.970
10	0.971	0.968	0.971	0.970

Таблица 4: Результаты работы кросс-валидации, косинусная метрика

Как видим, для классификации рукописных цифр методом k ближайших соседей несколько лучше работает косинусная метрика, а оптимальным значением k является 4.

Рассмотрим теперь качество классификации для взвешенного алгоритма k ближайших соседей. Проведем аналогичные описанным выше оценки точности. Время работы кросс-валидации остается приблизительно таким же, как и для невзвешенного варианта: 90-100 секунд для фиксированной метрики.

k	1-й фолд	2-й фолд	3-й фолд	Среднее арифметическое
1	0.968	0.966	0.966	0.967
2	0.968	0.966	0.966	0.967
3	0.970	0.969	0.968	0.969
4	0.971	0.969	0.970	0.970
5	0.969	0.968	0.968	0.968
6	0.970	0.968	0.969	0.969
7	0.967	0.965	0.967	0.966
8	0.967	0.966	0.967	0.967
9	0.965	0.964	0.965	0.965
10	0.964	0.964	0.965	0.965

Таблица 5: Результаты работы кросс-валидации, евклидова метрика

k	1-й фолд	2-й фолд	3-й фолд	Среднее арифметическое
1	0.973	0.970	0.970	0.971
2	0.973	0.970	0.970	0.971
3	0.975	0.972	0.971	0.973
4	0.975	0.973	0.973	0.974
5	0.974	0.970	0.973	0.973
6	0.975	0.971	0.973	0.973
7	0.973	0.969	0.972	0.971
8	0.973	0.970	0.972	0.971
9	0.972	0.969	0.971	0.970
10	0.971	0.968	0.971	0.970

Таблица 6: Результаты работы кросс-валидации, косинусная метрика

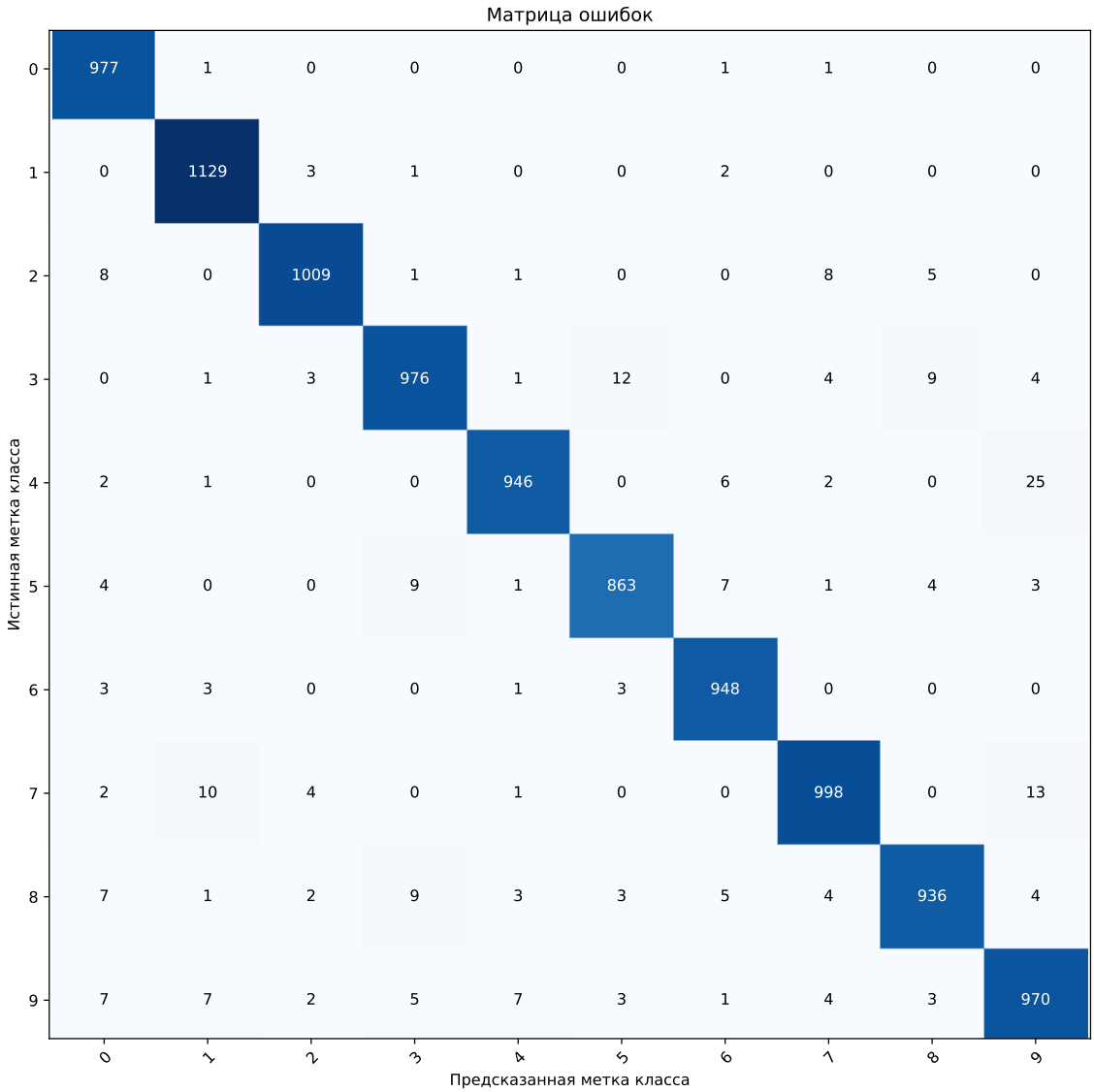
Сравнив таблицы 3-4 и 5-6, заметим, что взвешенный алгоритм k ближайших соседей в некоторых случаях работает чуть более точно. Оптимальное для невзвешенного алгоритма значение k=4 остается таким же и для взвешенного.

Оценка качества классификации на отложенной выборке

Подобрав по кросс-валидации оптимальные гиперпараметры модели, обучим нашу финальную модель и оценим её точность на отложенной выборке. Напомним, что для обучения используются 60 тысяч объектов, а для валидации 10 тысяч. В качестве модели используем взвешенный алгоритм k ближайших соседей при k=4 с использованием косинусной метрики. Получаем точность классификации 0.9752 по метрике `accuracy`. Заметим, что на кросс-валидации качество было немного хуже. Это может быть связано с тем, что фолды имели большой размер, и каждый раз в обучающую выборку не попадали некоторые важные образцы рукописных цифр. На сегодняшний день лучшие результаты по классификации объектов набора данных

MNIST показывают сверточные нейронные сети (CNN): их точность превышает 0.99 по метрике `accuracy`.

Проанализируем теперь матрицу ошибок, допущенных при классификации.



Как видим, чаще всего наша модель ошибается при попытке классифицировать цифру 4, которую она относит к классу 9. Рассмотрим несколько примеров, на которых модель допустила ошибку.

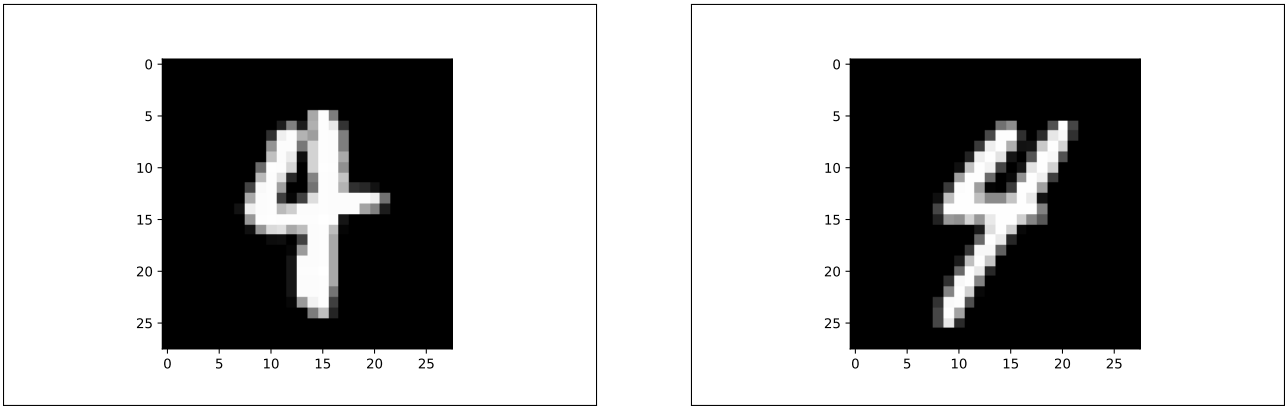


Рис. 1: Примеры объектов, на которых модель ошиблась

Следующей по популярности ошибкой является попытка классифицировать цифру 7 как цифру 9.

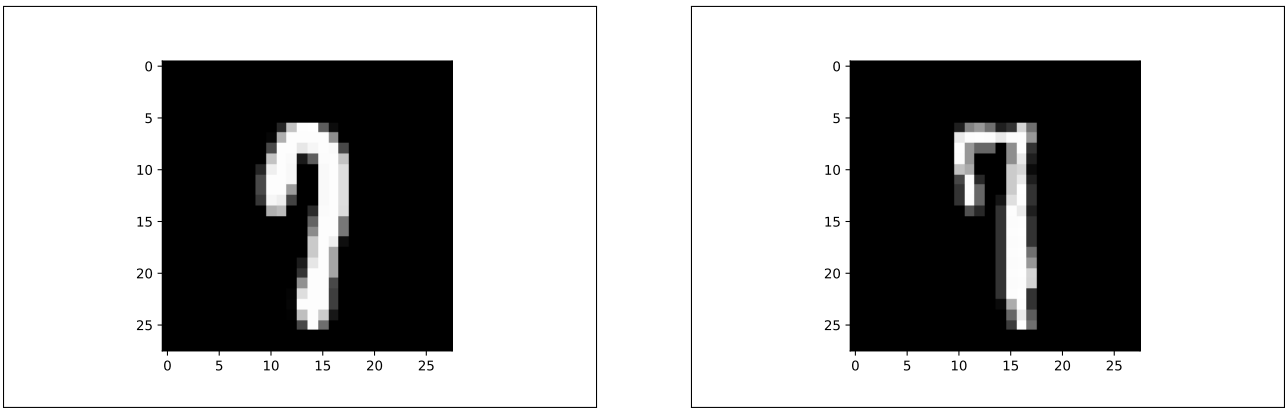


Рис. 2: Примеры объектов, на которых модель ошиблась

И, напоследок, рассмотрим ошибку, связанную с попыткой классифицировать цифру 3 как цифру 5.

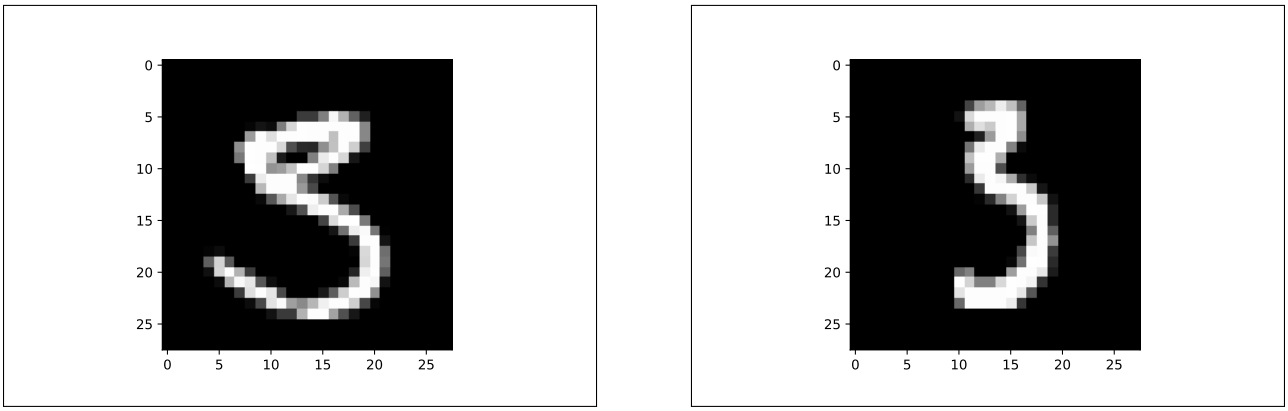


Рис. 3: Примеры объектов, на которых модель ошиблась

Можно заметить, что пары цифр, на которых модель ошибается, часто имеют общие элементы в написании, что способствует уменьшению расстояния между ними. Иногда, если цифра написана неаккуратно, даже человеку может быть сложно её классифицировать (см. изображения выше).

Аугментация

Часто для увеличения качества классификации модели при работе с изображениями используют искусственное увеличение выборки: к картинкам применяют различные преобразования, такие как отражения, сдвиги, размытия, повороты и т.д. Поскольку при классификации рукописных цифр важна ориентация, мы рассмотрим лишь последние 3 преобразования. Рассмотрим следующие преобразования:

- Повороты изображения против и по часовой оси на 5, 10 и 15 градусов
- Сдвиги изображений по осям X и Y на 1, 2 и 3 пикселя (в обе стороны)
- Размытия по Гауссу с $\sigma = 0.5, 1.0, 1.5$

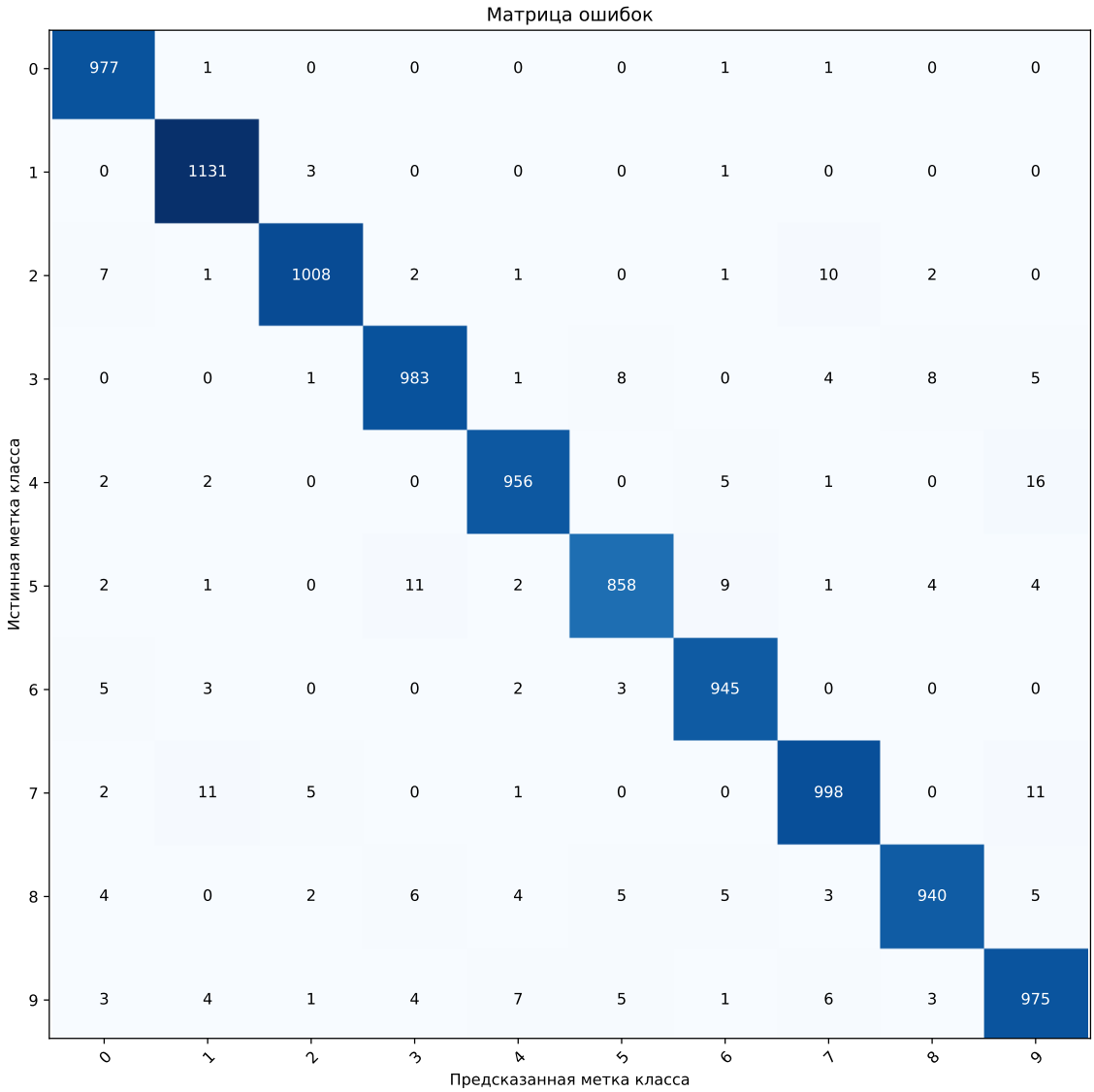
Так как рассмотреть всевозможные варианты этих преобразований является достаточно тяжелой в плане вычислительных ресурсов задачей, будем выбирать преобразования случайный образом (из возможных 648 вариантов), применять их к урезанной (опять же, в соображениях экономии ресурсов) обучающей выборке, пополнять её преобразованными изображениями и измерять качество на кросс-валидации. Рассмотрим полученные результаты:

№	Поворот	Сдвиг по X	Сдвиг по Y	Параметр σ размытия	accuracy (mean)
1	-15	-1	-1	1.5	0.968
2	-10	3	-3	1	0.975
3	-5	-2	1	0.5	0.984
4	-5	2	1	1	0.982
5	-5	3	1	0.5	0.984
6	5	-3	-3	0.5	0.984
7	5	3	-2	1.5	0.974
8	5	3	3	0.5	0.984
9	15	-2	-2	1	0.970
10	15	-2	1	0.5	0.974

Таблица 7: Результаты аугментации

Как видим, в среднем, качество классификации повышается. Возьмем один из лучших (среди имеющихся) вариантов преобразования: поворот на -5 градусов, сдвиг по X на 3 пикселя, сдвиг по Y на 1 пиксель, размытие гаусса с $\sigma = 0.5$ и применим его для раздутия всей обучающей выборке (60 тыс. объектов). Применив полученный

классификатор на отложенной выборке (10 тыс. объектов), получим значение 0.977 по метрике `accuracy`. Таким образом, аугментация действительно улучшает качество классификации. Для ещё большего повышения точности, можно перебрать больше комбинаций преобразований. Рассмотрим теперь матрицу ошибок.

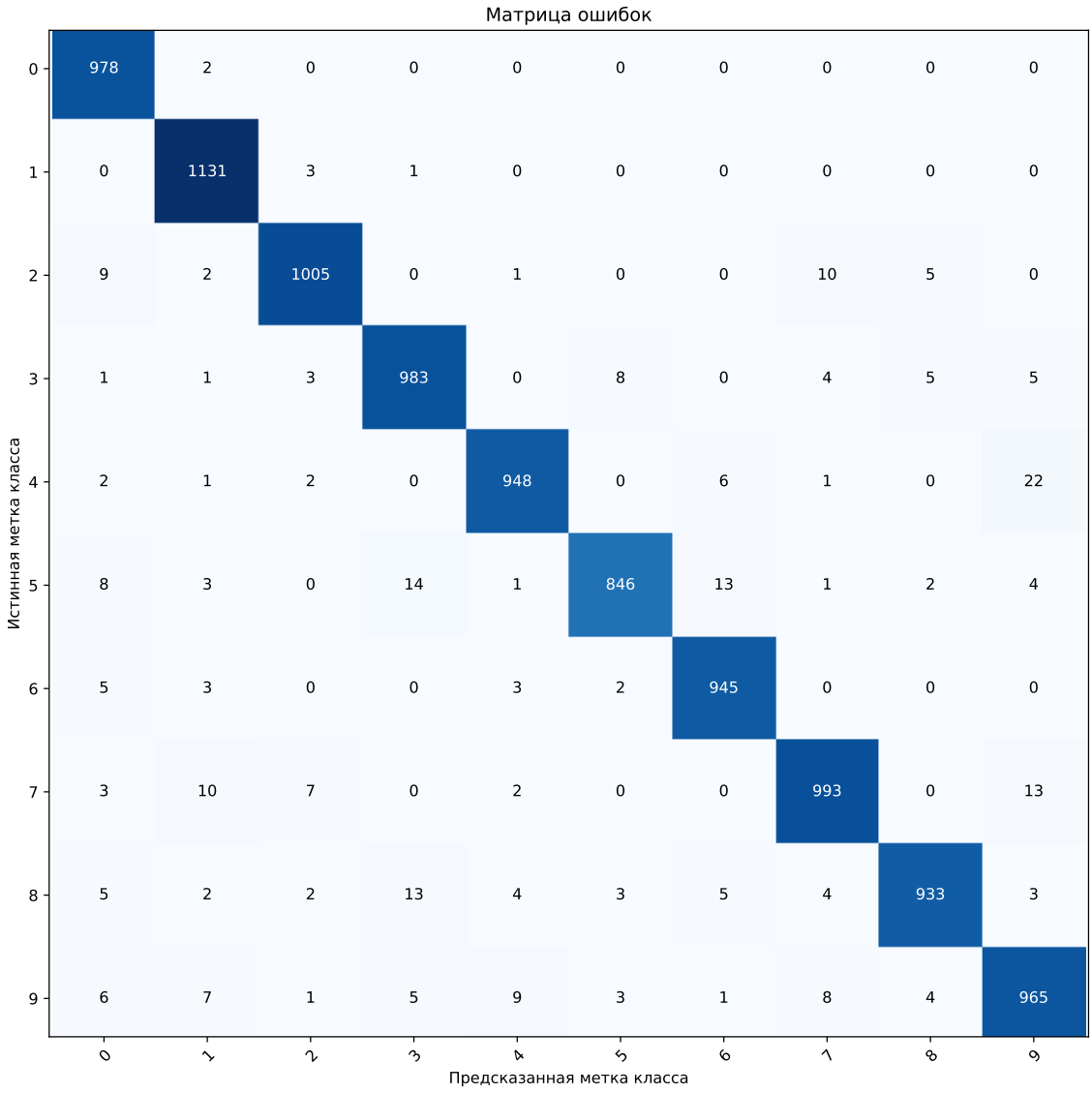


Как видим, алгоритм стал меньше ошибаться на описанных выше примерах. Возможно, наибольший вклад внесло преобразование поворотом, поскольку люди часто пишут под разным наклоном, а размытие позволило иметь похожим цифрам чуть меньшее расстояние, поскольку теперь каждый «выделенный» пиксель обязательно имел в своей окрестности информацию о том, что он близко.

Другой подход к размножению выборки

Если в предыдущем методе мы искусственным образом пытались увеличить обучающую выборку, то сейчас попробуем преобразовывать тестовую выборку: для каж-

дого объекта получим несколько его трансформированных версий, для каждой из них будем искать k ближайших соседей в обучающей выборке, и некоторым образом подытоживать полученные результаты. Например, будем выбирать среди полученных 4k соседей объекты с наименьшим расстоянием, и брать среди них самый популярный класс. Подбором по урезанной (из соображений времени работы) обучающей выборке выберем параметры преобразования: поворот на 15 градусов, сдвиг по X на -2 пикселя, по оси Y на 1 пиксель, параметр размытия $\sigma = 0.5$. Применим полученные параметры для использования данного метода на неурезанных выборках. Результатом будет являться незначительное улучшение точности: 0.9757 по `accuracy`. Рассмотрим матрицу ошибок.



Видим небольшие улучшения по сравнению с KNN без размножения выборки. Возможно, нам «не повезло» и среди перебираемых параметров преобразований не нашлось оптимальных (напомним, что из соображений экономии ресурсов, мы перебирали лишь часть комбинаций).

В нашем случае эффективнее оказалось размножить обучающую выборку, а не тестовую. Для более точной классификации следует перебирать больше комбинаций параметров, а также, для каждой из них перебирать некоторые значения k (в нашем случае всегда использовался вариант с k=4). Кроме того, возможно, стоит попробовать скомбинировать оба подхода: размножить как обучающую выборку, так и тестовую (в указанном выше смысле). Второй должен работать лучше, если тестовые данные неточные или зашумленные, а в обучающей выборке представлена широкая коллекция качественных экземпляров (такая ситуация достаточно часто может встречаться на практике: обучающая выборка долго чистилась и подготовливалась, а поступающие на вход данные требуют быстрой обработки).

Вывод

В данном отчете были представлены некоторые подходы к классификации с помощью метода k ближайших соседей, было исследовано качество классификации различных модификаций данного алгоритма на наборе данных рукописных цифр MNIST.

Список литературы

[1] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB*, pages 194–205. Morgan Kaufmann, 1998.