Московский Государственный
Университет

Метрические методы классификации

Выполнил: Курцев Д.В.
Группа: 317

Факультет Вычислительной математики и кибернетики
Кафедра Математических методов прогнозирования

Октябрь 2021

1 Введение

Пусть у нас имеется обучающая выборка $X^n = \{x_1, ..., x_n\}$, где каждый $x_i \in \mathbb{R}^d$. И метки классов $Y^n = \{y_1, ..., y_n\}$, $y_i \in \{1, ..., m\} = Y$, где m - количество классов.

В данной работе был реализован и разобран алгоритм kNN - k Nearest Neighbors. Его суть заключается в следующем. На вход подается матрица $X' \subset \mathbb{R}^{l \times d}$. Строится матрица попарных расстояний между X и X'. Для каждого объекта из X' выбирается k ближайших соседей из X. И он соотносится тому классу, чьих объектов из X больше всего среди этих k соседей:

$$a(x, X^n) = \arg\max_{y \in Y} \sum_{i=1}^{k} [y^{(i)} = y] w(i, x)$$

где $w(i, x)$ - вес i-го соседа. В обычной реализации kNN все $w(i, x)$ считаются равными 1. В данной работе был также рассмотрен взвешенный метод kNN, в котором $w(i, x) = 1/(distance(x, x^{(i)}) + 10^{-5})$

Проанализируем как работает kNN в зависимости от разных гиперпараметров. Все эксперименты проводились на датасете MNIST, включающим в себя 60 тыс. объектов обучающей выборки с 784 признаками. И 10 тыс. объектов тестовой выборки с таким же количеством признаков.

2 Зависимость времени поиска k ближайших соседей от стратегии

В данной реализации kNN было рассмотрено 4 метода поиска соседей 'my_own', 'brute', 'kd_tree' и 'ball_tree'. Проверим время поиска 5 ближайших соседей с евклидовой метрикой.

Случайным образом выберем 10/20/100 признаков и засечём время работы алгоритма для каждой стратегии.

Кол-во признаков	**my_own**	**brute**	**kd_tree**	**ball_tree**
10	65.74	11.01	1.60	2.15
20	66.85	10.96	6.68	28.08
100	69.86	14.13	132.34	114.48

Таблица 1: Время работы алгоритмов, с

Из данной таблицы видно, что стратегии *kd_tree* и *ball_tree* быстро работают при небольшой размерности признакого пространства. При этом *ball_tree* уже начинает уступать *brute* при 20 признаках. Но если объекты из большого принакого пространства, то их использование приведёт к неэффективному поиску ближайших соседей. Также отметим, что стратегия *my_own* работает в несколько раз хуже, чем аналогичная ей *brute*. Это связано с неэффективной сортировкой матрицы расстояний. Метод *brute* в своей реализации использует более быстрые сортировки, что делает его более пригодным. В дальнейшем для работы с kNN будем использовать стратегию *brute*.

3 Оценка качества по кросс-валидации

Для выбора остальных гиперпараметров (количество соседей, метрика, учитывание голосов), при которых kNN даёт лучший результат классификации, воспользуемся кросс-валидацией. С помощью неё запустим алгоритм на нескольких комбинациях параметров и выберем лучшую. Разобьём обучающую выборку на 3 фолда. Результаты исследований представлены в следующих четырёх таблицах.

Для начала рассмотрим обычный метод kNN, в котором голос каждого соседа одинаков. Время работы этого алгоритма для $k \in [1, 10]$, для евклидовой метрики равно 162.7 секунд, а для косинусной 160.1.

k	**1-ый фолд**	**2-ой фолд**	**3-ий фолд**	**mean**
1	0.9689	0.9668	0.9667	0.9675
2	0.9603	0.9599	0.9613	0.9605
3	0.9696	0.9683	0.9672	0.9683
4	0.9670	0.9667	0.9671	0.9669
5	0.9681	0.9673	0.9672	0.9675
6	0.9656	0.9649	0.9657	0.9654
7	0.9652	0.9650	0.9656	0.9653
8	0.9642	0.9639	0.9653	0.9644
9	0.9637	0.9632	0.9646	0.9638
10	0.9625	0.9622	0.9637	0.9628

Таблица 2: Евклидова метрика

k	**1-ый фолд**	**2-ой фолд**	**3-ий фолд**	**mean**
1	0.9733	0.9708	0.9705	0.9715
2	0.9689	0.9664	0.9678	0.9677
3	0.9737	0.9716	0.9709	0.9720
4	0.9725	0.9709	0.9716	0.9719
5	0.9728	0.9702	0.9716	0.9715
6	0.9719	0.9697	0.9722	0.9713
7	0.9713	0.9682	0.9708	0.9701
8	0.9713	0.9684	0.9716	0.9704
9	0.9701	0.9670	0.9707	0.9693
10	0.9694	0.9662	0.9707	0.9687

Таблица 3: Косинусная метрика

Теперь посмотрим, как работает взвешенный метод kNN. Время выполнения для евклидовой метрики составило 168 секунд, а работа на косинусной забрала 160 секунду.

k	**1-ый фолд**	**2-ой фолд**	**3-ий фолд**	**mean**
1	0.9689	0.9667	0.9667	0.9675
2	0.9690	0.9668	0.9667	0.9675
3	0.9708	0.9691	0.0.9683	0.9694
4	0.9713	0.9698	0.9701	0.9704
5	0.9693	0.9683	0.9687	0.9687
6	0.9704	0.9685	0.9695	0.9694
7	0.9674	0.9659	0.9670	0.9668
8	0.9676	0.9667	0.9678	0.9674
9	0.9651	0.9643	0.9657	0.9650
10	0.9647	0.9646	0.9657	0.9650

Таблица 4: Евклидова метрика

k	**1-ый фолд**	**2-ой фолд**	**3-ий фолд**	**mean**
1	0.9733	0.9708	0.9704	0.9715
2	0.9733	0.9708	0.9705	0.9715
3	0.9749	0.9725	0.971	0.9731
4	0.9754	0.9732	0.9737	0.9741
5	0.9740	0.9709	0.9730	0.9726
6	0.9747	0.9714	0.9731	0.9730
7	0.9726	0.9693	0.9719	0.9713
8	0.9729	0.9704	0.9722	0.9719
9	0.9717	0.9687	0.9715	0.9706
10	0.9715	0.9686	0.9714	0.9705

Таблица 5: Косинусная метрика

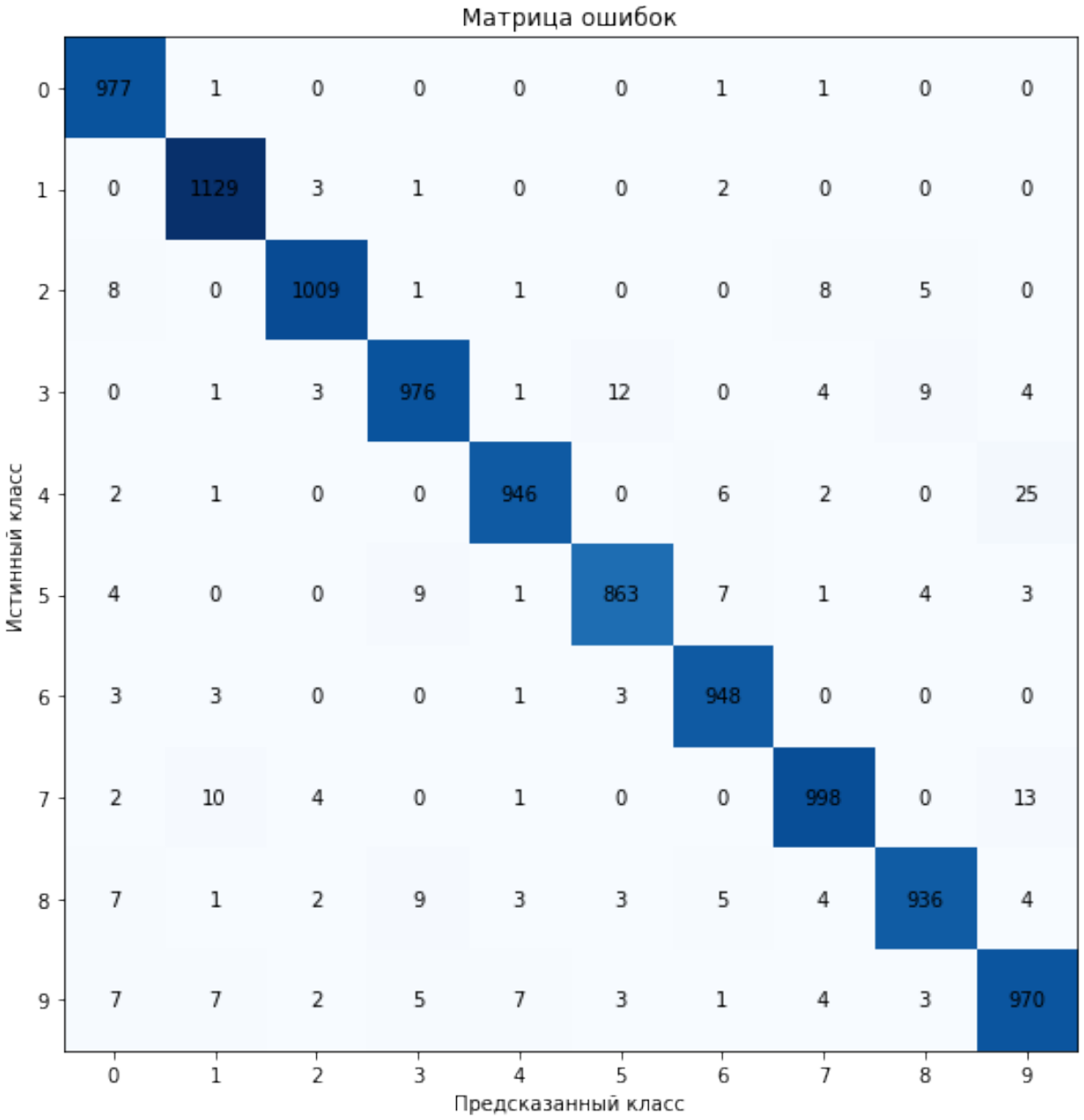
Исходя из экпериментов можно сделать вывод, что невзвешенный метод работает лучше при k = 3, а алгоритм, учитывающий расстояния при k = 4. По таблицам 2, 4 и 3, 5 можно установить, что косинусная метрика лучше подходит для правильной классификации. При этом она даже немного выигрывает по затратам на время работы. И наконец, можно заключить, что особой разницы между обычным алгоритмом работы kNN и взвешенным практически нет. Но в большинстве ситуациях последний работает чуть лучше.

После всех проделанных экпериментов можно сделать вывод, что для датасета MNIST лучше всего взять следующие гиперпараметры kNN. k = 4, strategy = brute, metric = cosine. При этом следует использовать взвешенный метод классификации. В дальнейшем всюду будем брать именно эти параметры.

4 Проверка качества работы kNN

После подбора нужных гиперпараметров обучим наш алгоритм и проверим, как он работает на тестовой выборке. Качество классификации достигает 97.52%.

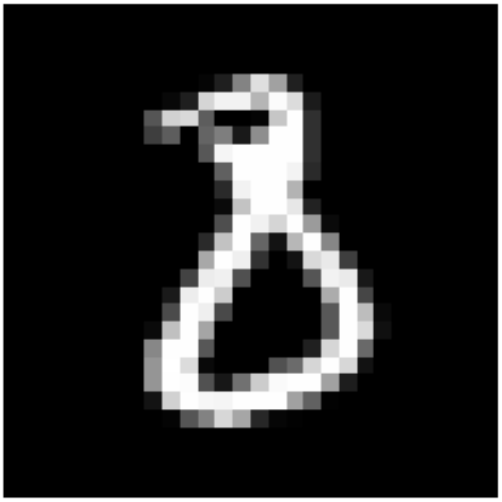
Посмотрим на каких данных наш метод ошибся. Для этого построим матрицу ошибок. Где на пересечении ij-ой строки стоит количество раз, которое алгоритм отнёс объект из i-ого класса j-ому.



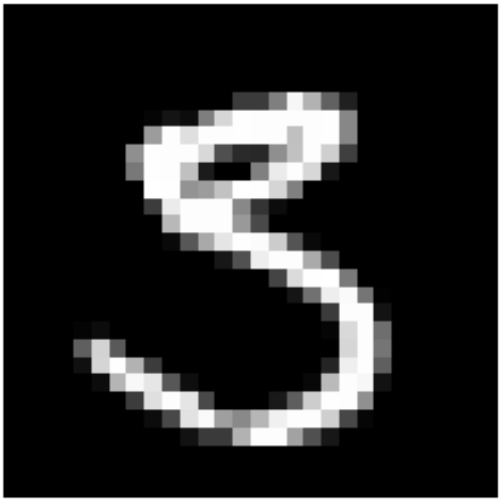
Видим, что наша модель на тестовой выборке отработала чуть лучше, чем на валидации. Это может быть связано с тем, что на каждом этапе кросс-валидации мы отсоединяем слишком большую

часть выборки на тестовую. А в ней могут находиться важные объекты для классификации (какие-нибудь уникальные начертания цифр). На данный момент существует множество методов классификации, работающих лучше kNN, в частности и на датасете MNIST. К примеру, точность свёрточных нейронных сетей достигает до 99.5%.

Можно заметить, что наш алгоритм меньше всего ошибся при классификации '0' и '1', 3 и 6 соответственно раз. Наибольшее количество ошибок он допустил на '8' - 38 раз и '9' - 39 раз. Приведём пример восьмёрки, классифицированной как '3'. Для сравнения выведем ещё цифру 3:

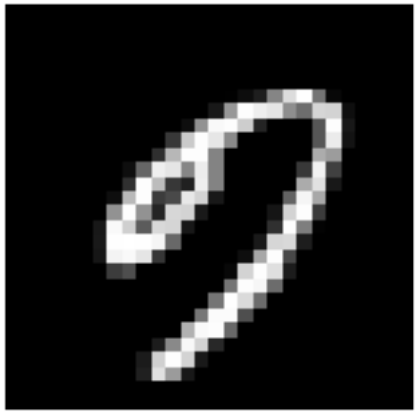


8

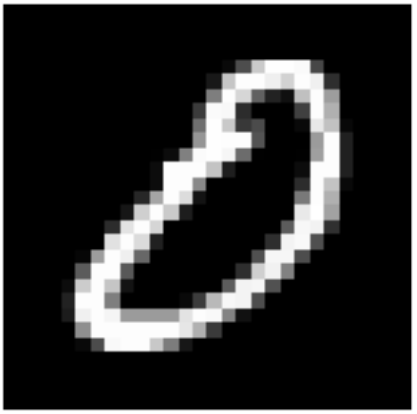


3

Теперь посмотрим пример девятки, которая была воспринята как '0':

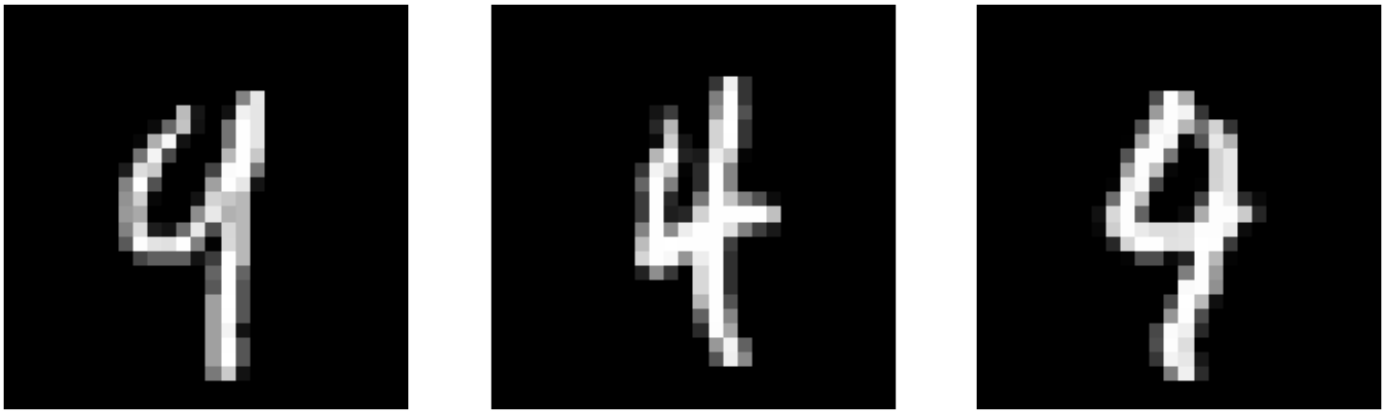


9



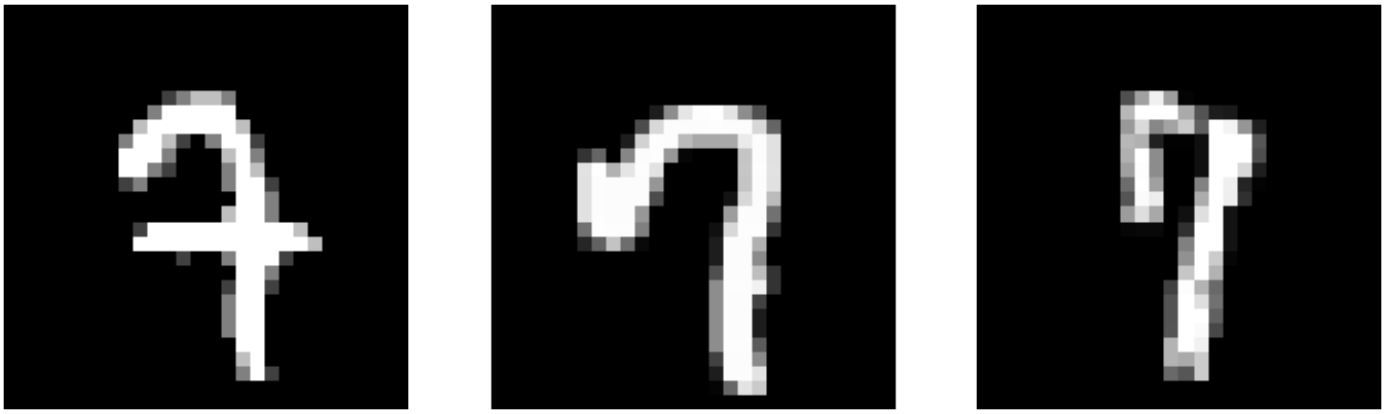
0

Далее обратим внимание на максимальное число неправильно соотнесённых классов к одному и тому же. Можно заметить, что наш алгоритм больше всего ошибается (25 раз) при классификации '4' как '9':



Неправильно классифицированные 4

13 раз он соотнёс '7' к '9':



Неправильно классифицированные 7

Можно заметить, что неправильно класифицированные объекты, действительно, имеют похожие начертания. Усугубляет работу алгоритма плохое написание цифр, что делает объекты из разных классов очень похожими.

5 Аугментация обучающей выборки

Чтобы улучшить качество работы нашей модели, искусственно расширим нашу обучающую выборку в 4 раза. Применим для этого несколько техник:

- Поворот изображения на 5, 10 и 15 градусов (по и против часовой стрелки).
- Сдвиг изображения на 1, 2 и 3 пикселя по каждой из осей во все стороны.
- Применение гауссовского фильтра с дисперсией 0.5, 1 и 1.5

Всевозможных комбинаций огромное количество. Мы рассмотрим лишь некоторые из них. Чтобы хоть как-то исследовать вклад каждого преобразования, поступим следующим образом.

На первом этапе подберём грудус и направление поворота. Для этого случайно выберем параметры сдвига и фильтра и запустим их для всех параметров поворота. То есть получим 6 итераций. По кросс-валидации с 3 фолдами подберём наилучший поворот изображения.

На втором шаге снова рандомно выберем значение гауссовского фильтра. И запустим нашу модель по всем свигам по горизонтальной оси. Для каждого его значения будем рандомно выбирать значение сдвига по вертикальной оси так, чтобы оно не повторялось с предыдущими. Итого ещё 6 итераций.

И на последнем шаге с уже выбранными сдвигами и поворотами запустим наш метод для всех значений фильтров. Это ещё плюс 2 итерации (на прошлом шаге мы уже посчитали для одного из значений).

Таким образом, мы проведём 14 исследований на зависимость различных техник аугментации. Покажем среднее значение точности модели на кросс-валидации (столбец $mean$). Результаты приведены в следующей таблице:

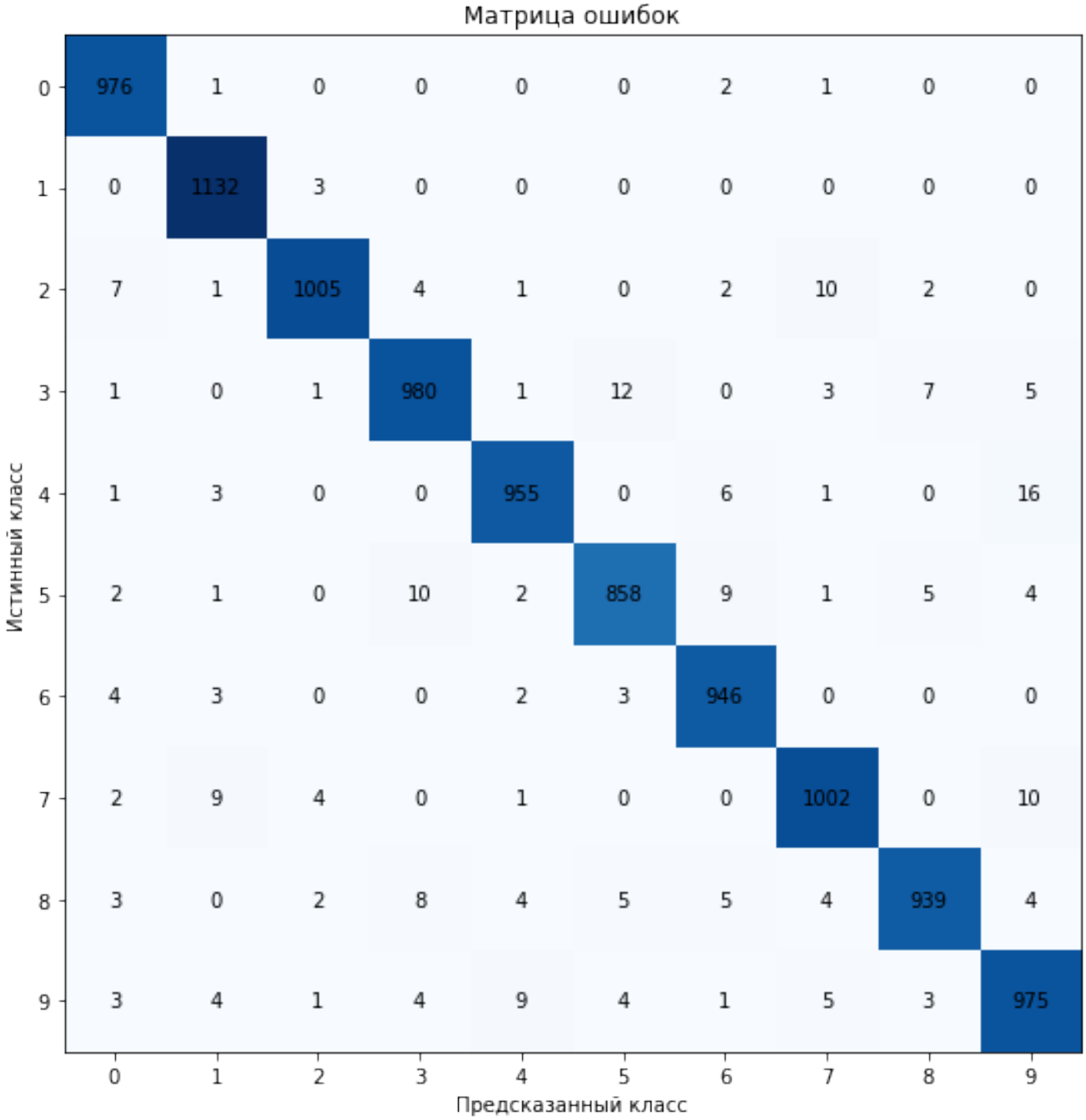
	Поворот, градус	**Сдвиг по X**	**Сдвиг по Y**	**Фильтр**	**mean**
1	-15	2	3	0.5	0.979
2	-10	2	3	0.5	0.985
3	-5	2	3	0.5	0.989
4	5	2	3	0.5	0.989
5	10	2	3	0.5	0.985
6	15	2	3	0.5	0.979
7	-5	-3	-3	1.0	0.989
8	-5	-2	3	1.0	0.989
9	-5	-1	1	1.0	0.992
10	-5	1	-1	1.0	0.993
11	-5	2	-2	1.0	0.989
12	-5	3	2	1.0	0.989
13	-5	3	2	0.5	0.993
14	-5	3	2	1.5	0.988

Таблица 6: Результат аугментации

Можно заметить, что качество нашей модели заметно увеличилось. Это может быть обусловлено тем, что при расширении выборки и разбитии её на фолды, у нас в обучающей и валидационной частях находится много похожих элементов.

Выберем наилучшие параметры (угол поворота $-5°$, сдвиг по оси X равным 1, а по Y = -1, значение гауссовского фильтра 0.5) для аугментации. Проверим, как работает наш алгоритм на тестовой выборке.

Получим более хорошее качество классификации (97.68% правильно распознаных объектов) и следующую матрицу ошибок.



Заметим, что kNN стал меньше ошибаться. Цифру ’9’ алгоритм теперь неправильно относит к другим классам 34 раза, а ’8’ 35 раз. Неправильное отношение ’4’ к ’9’ снизолось до 16 раз, а ’7’ к ’9’ до всего 10 раз. Это говорит о том, что аугментация является полезной техникой для классификации изображений.

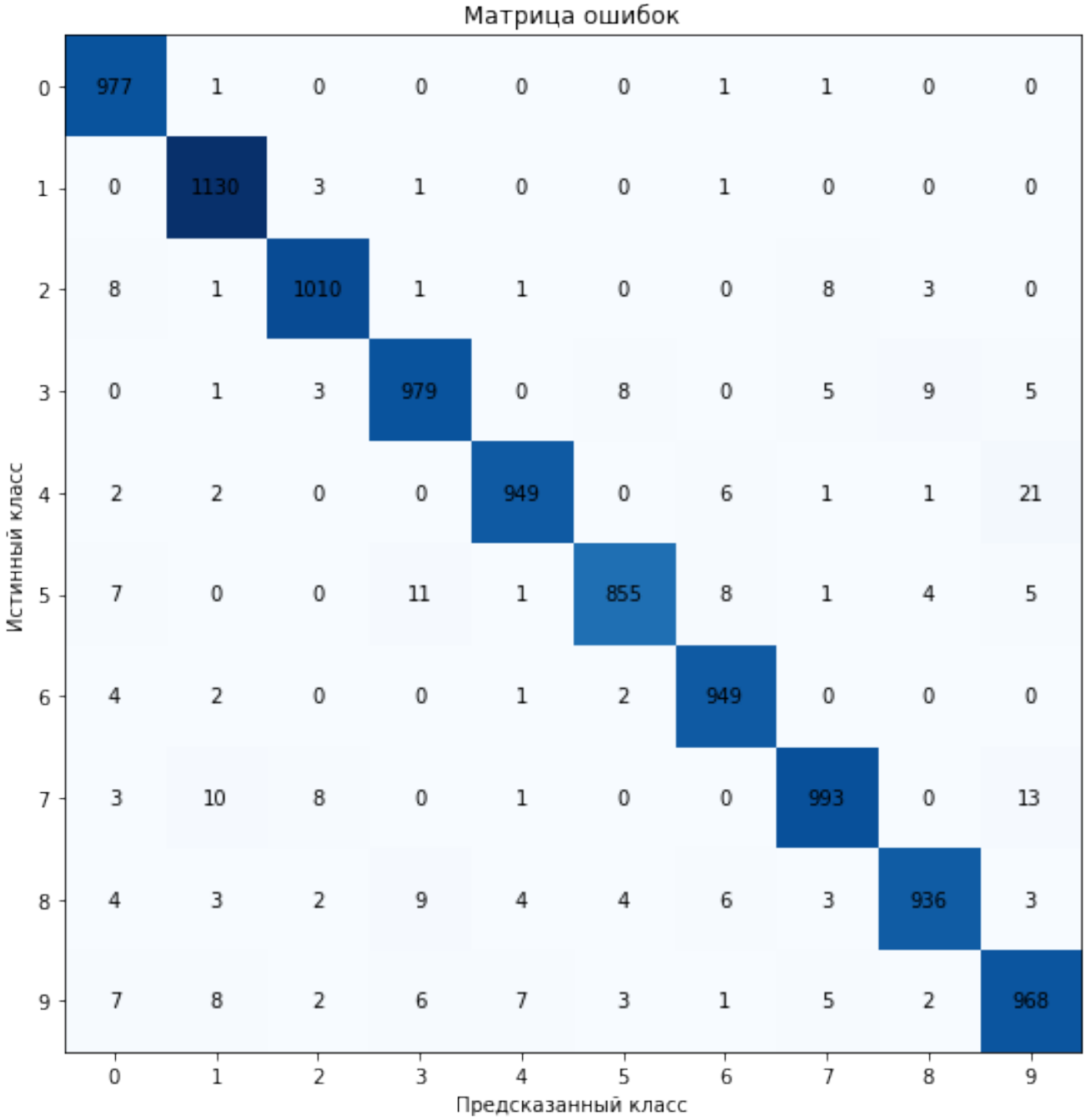
6 Аугментация тестовой выборки

Проведём теперь аугментацию тестовой выборки, оставив обучающую нетронутой. Параметры аугментации возьмём из Таблицы 6. При этом рассмотрим лишь те случаи, на которых был лучший результат на валидации. Протестируем нашу модель на четырёх "разных"выборках и будем предсказывать тот класс, который встретился наибольшее количество раз из этих четырёх подпредсказаний.

	Поворот, градус	**Сдвиг по X**	**Сдвиг по Y**	**Фильтр**	**mean**
1	-5	1	-1	0.5	0.9746
2	-5	1	-1	1.0	0.9737
3	-5	2	3	1.0	0.9729
4	5	2	3	0.5	0.9746
5	-5	3	3	1.0	0.9726
6	-5	1	-1	1.5	0.9731
7	5	-2	3	0.5	0.9741
8	5	3	2	0.5	0.9746
9	-10	1	-1	1.0	0.9732
10	10	-1	-1	1.0	0.9715

Таблица 7: Результат аугментации

Построим матрицу ошибок для наилучшего случая. Возьмём поворот на 5°, сдвиг по оси X = 2, по Y 3, фильтр 0.5.



Заметим, что аугментация тестовой выборки практически не улучшила качество работы алгоритма. '4' была классифицирована, как '9' 21 раз, что хуже, чем при аугментации обучающей выборки. Можно заключить, что расширение обучайющей выборки более эфективно, чем тестовой.

7 Заключение

Был реализован и подробно разобран метод классификации kNN - k Nearest Neighbors. Для более точных предсказаний следует расширять выборку. Подобрать необходимые гиперпараметры модели или аугментации, на которых метод работает более правильно, можно по кросс-валидации.

Содержание