Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчёт по практикуму

Использование K Nearest Neighbors Classifier для исследования датасета MNIST

Илюхин Владислав
317 группа

Москва 2021

Содержание

1 Постановка задачи

1.1 K Nearest Neighbors Classifier

Первая часть задания - написать самостоятельно на языке *Python* метод ближайших соседей и средства для проверки его точности на кросс-валидации.

1.2 Постановка экспериментов

Вторая часть задания - проведение шести экспериментов с датасетом $MNIST$ при помощи написанного ранее функционала.

2 Реализация функционала

В соотвествии с техническим заданием я реализовал три модуля

- $nearest\_neighbors.py$
- $cross\_validation.py$
- $distances.py$

Первый реализует класс $KNNClassifier$, классифицирующий подаваемые на вход объекты методом ближайших соседей при помощи информации о классах объектов сохраненной выборки.

Второй модуль содержит функции для разбиения выборки на фолды и оценки точности классификации на кросс-валидации.

Третий модуль реализует метрики, используемые для вычисления расстояния в двух других модулях.

Все модули написаны на языке *Python* с использованием библиотеки *Numpy* и успешно сданы в тестирующую систему *Anytask*. Исходный код приложен к отчету.

3 Постановка экспериментов

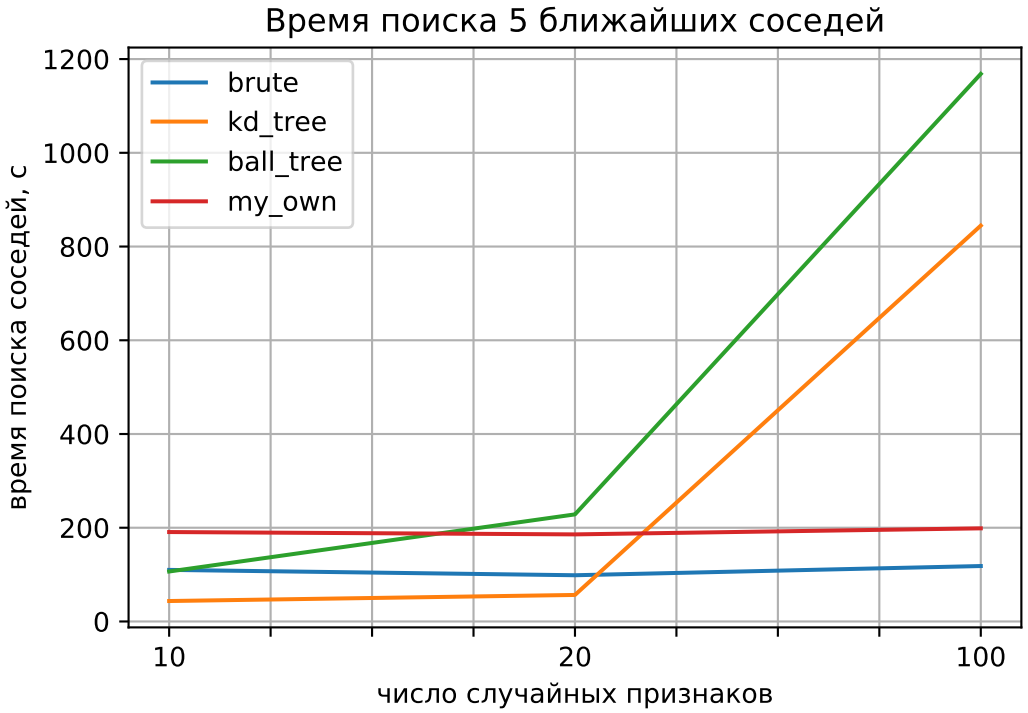
Для постановки экспериментов использовалась среда *JupyterNotebook*. Файл, в котором проводились эксперименты (*MNISTresearch.ipynb*) приложен к отчету.

3.1 Эксперимент 1

Цель первого эксперимента - сравнить скорости работы различных методов поиска ближайших соседей при условии, что используется только евклидова метрика. Методы поиска:

- *brute* - встроенный метод библиотеки *Sklearn*, поиск ближайших соседей путем простого перебора всех попарных расстояний
- *kd_tree* - встроенный метод библиотеки *Sklearn*, построение по данным дерева, в котором каждый лист отвечает за область пространства
- *ball_tree* - встроенный метод библиотеки *Sklearn*, построение по данным дерева, в котором каждый лист отвечает за шар в пространстве
- *my_own* - реализованный мной метод, построение матрицы попарных расстояний между объектами и выделение из нее ближайших соседей с помощью *numpy.argpartition*

Методы тестировались на признаковых пространствах размерностей $10, 20$ и 100:

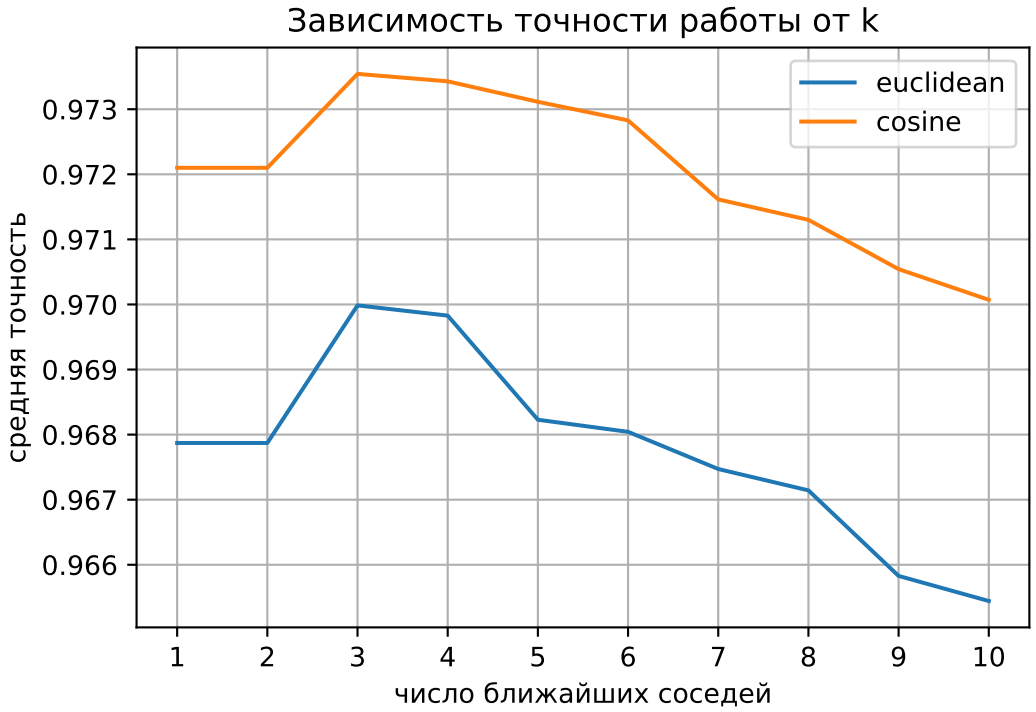


Получается, методы *kd_tree* и *ball_tree* работают быстро на 10 признаков, чуть медленнее на 20 и очень медленно на 100. В то же время my_owm и *brute* отрабатывают во всех трех случаях за примерно равное время. Объяснить вырождение первых двух методов может проклятие размерности - из лекций известно, что при размерности пространства 20 и выше поиск при помощи деревьев по количеству операций сравним с брутфорсом (в то время как все производимые операции значительно более дорогие).

3.2 Эксперимент 2

Цель второго эсперимента - оценить вляение на точность классификации количества взятых в расчет ближайших соседей k и выбранной метрики. $k = \overline{1, 10}$, метрика принимает значение *euclidean* или *cosine*.

Точность была исследована на всей выборке кросс-валидацией с 3 фолдами:



Из графика я сделал вывод, что при любой метрике наибольшая точность достигается при $k = 3$ и почти такая же точность получается при $k = 4$. При больших k точность монотонно убывает, при $k = 1, 2$ точность меньше, чем при $k = 3, 4$.

Также я заметил, что график точности косинусной метрики лежит строго выше графика точности евклидовой метрики, причем в среднем первая дает результат на 0.004 лучше, чем вторая. Я считаю, что это связано с тем, что значение косинусной метрики не зависит от масштаба - так, ярко-черная и сероватая одинаково начертанные цифры при ней будут близки, в то время как евклидова метрика, посчитав, что каждый серый пиксель отличается от черного, заключит, что такие цифры довольно далеки друг от друга.

Далее я проверил на скорость метрики *cosine* и *euclidean*. Поскольку для стратегий *kd_tree* и *ball_tree* валидна только *euclidean* - метрика, для исследования я выбрал только две стратегии *my_own* и *brute*:

Метрика	*my_own*	*brute*
euclidean	260.01	152.29
cosine	238.92	156.05

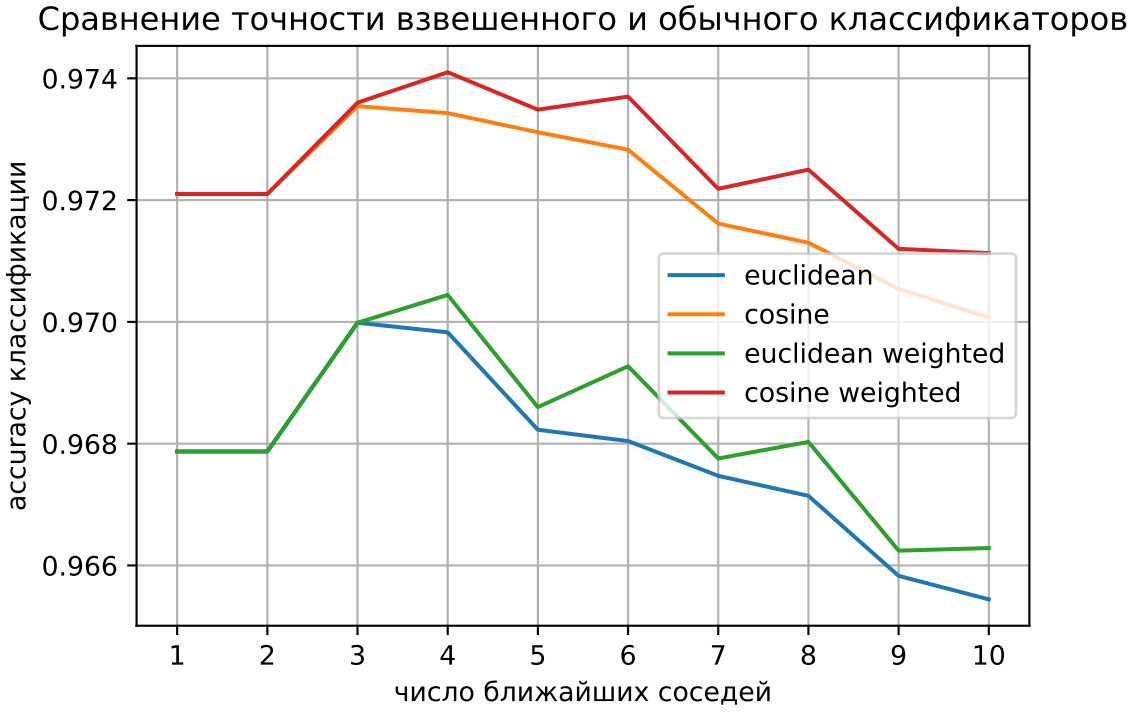
Таблица 1: Скорость работы классификатора при различных используемых метриках, с

Получилось, что *euclidean* для *my_own* работает немного медленнее *cosine*, для *brute* же разницы нет. Я связываю это с тем, что в модуле *distances.py* моя реализация *cosine*-метрики получилась удачнее, чем *euclidean*.

3.3 Эксперимент 3

Цель третьего эксперимента - сравнить точность метода ближайших соседей с весами (голос каждого соседа имеет вес $\frac{1}{distance+\varepsilon}$, где $distance$ - расстояние от объекта до соседа и $\varepsilon = 10^{-5}$) и без весов (голос каждого соседа имеет вес 1).

Я запустил классфикатор 4 раза - получив точность косинусной и евклидовой метрик с весами и без:



Получается, взвешенный метод всегда дает несколько более точный результат, чем невзвешенный. Я считаю, что это связано с тем, что взвешенный алгоритм лучше работает в следующих двух ситуациях:

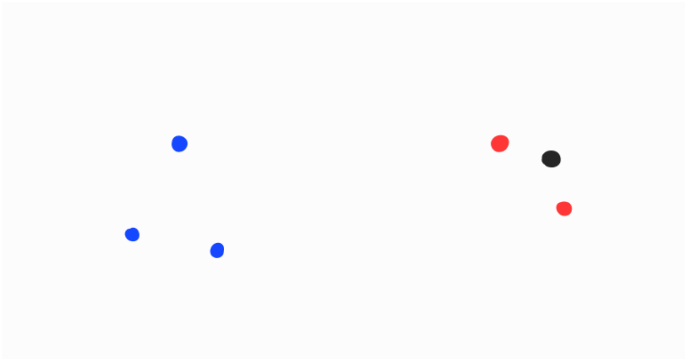


Рис. 1: Маленький кластер объектов одного класса поблизости, большой кластер объектов другого класса далеко

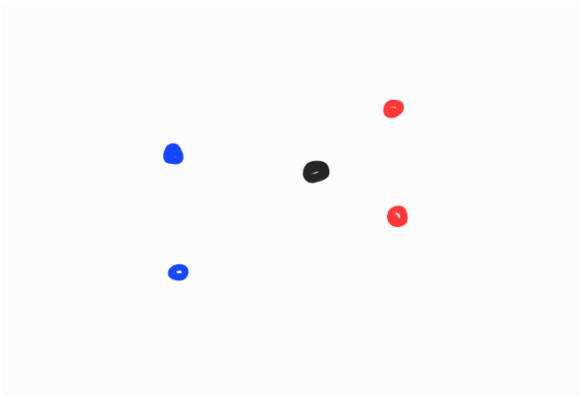


Рис. 2: Граничная точка, более близкая к одному классу, но не имеющая большинство соседей из ближайшего к ней класса

Итак, экспериментальным путем я установил, что самая точная классификация получается при параметрах $metric = cosine$ и $weights = True$.

3.4 Эксперимент 4

Цель первой части четвертого эксперимента - исследовать точнейший алгоритм из реализованных, в том числе сравнить полученную точность с лучшими.

В рамках исследования я обучаю его на обучающей части выборки и смотрю на точность на тестовой, никакой кросс-валидации.

Полученная точноть - 0.9752. Для сравнения, лучшая точность на кросс-валидации была примерно 0.974, а лучшая точность алогритмов из интернета - примерно 0.9881. То есть, построенный классификатор ошибается в среднем в два раза чаще, чем лучший. Это удивительно: ведь мы смотрим только на отдельные пиксели, а не на всю картинку в целом.

Цель второй части четвертого эксперимента - исследовать ошибки, производимые рассматриваемым алгоритмом.

Для начала я построил $confusionmatrix$ - матрицу, на позиции ij в которой записано число объектов класса i, определенных как объект класса j.

977	1	0	0	0	0	1	1	0	0
0	1129	3	1	0	0	2	0	0	0
8	0	1009	1	1	0	0	8	5	0
0	1	3	976	1	12	0	4	9	4
2	1	0	0	946	0	6	2	0	25
4	0	0	9	1	863	7	1	4	3
3	3	0	0	1	3	948	0	0	0
2	10	4	0	1	0	0	998	0	13
7	1	2	9	3	3	5	4	936	4
7	7	2	5	7	3	1	4	3	970

Таблица 2: Confusion matrix

Три максимальных недиагональных значения в матрице ошибок - 25 (4 распознана как 9), 13 (7 распознана как 9) и 12 (3 распознана как 6).

В целом почти все 1 и 2 распознаны верно, а основная путаница происходит между 2 и 7, 7 и 9, 4 и 9, 3 и 5, 3 и 6.

Проанализировать примеры ошибок я решил на основе 4, распознанных как 9.

Первый тип таких рукописных цифр - четверки с соединением двух "палок"сверху, образующим замкнутую область внутри цифры, как у девятки:

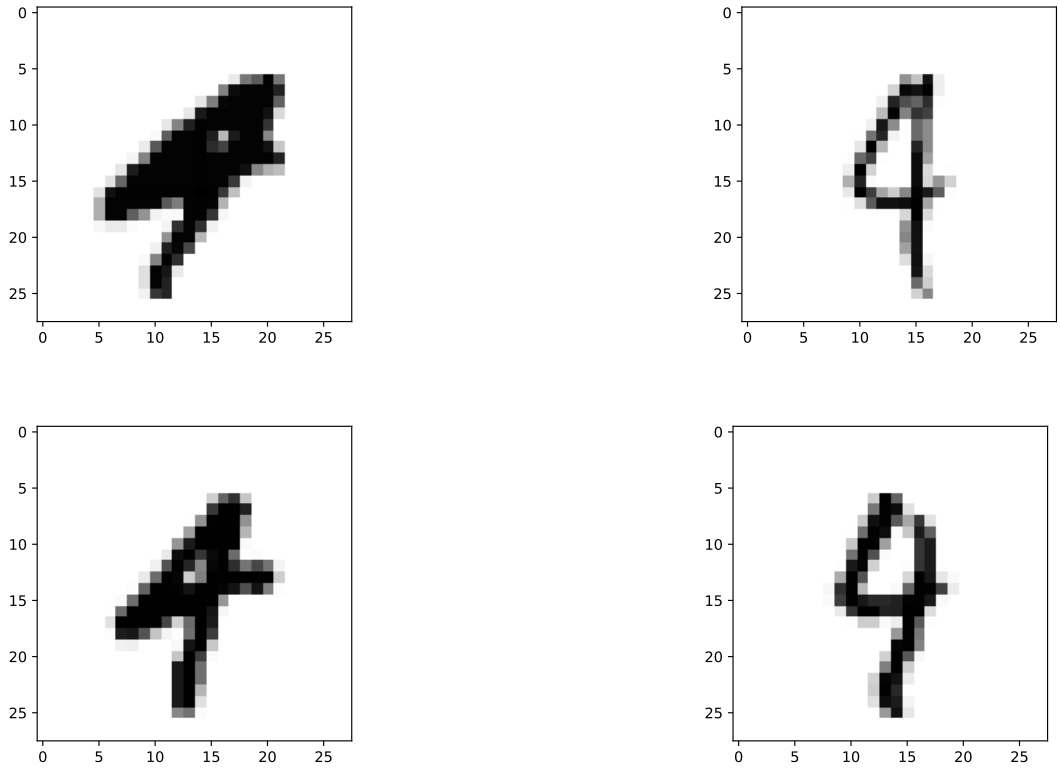


Рис. 3: Четверки первой группы

Второй тип - четверки, у которых верхние "палки"почти сходятся, но между ними все же есть свободное пространство:

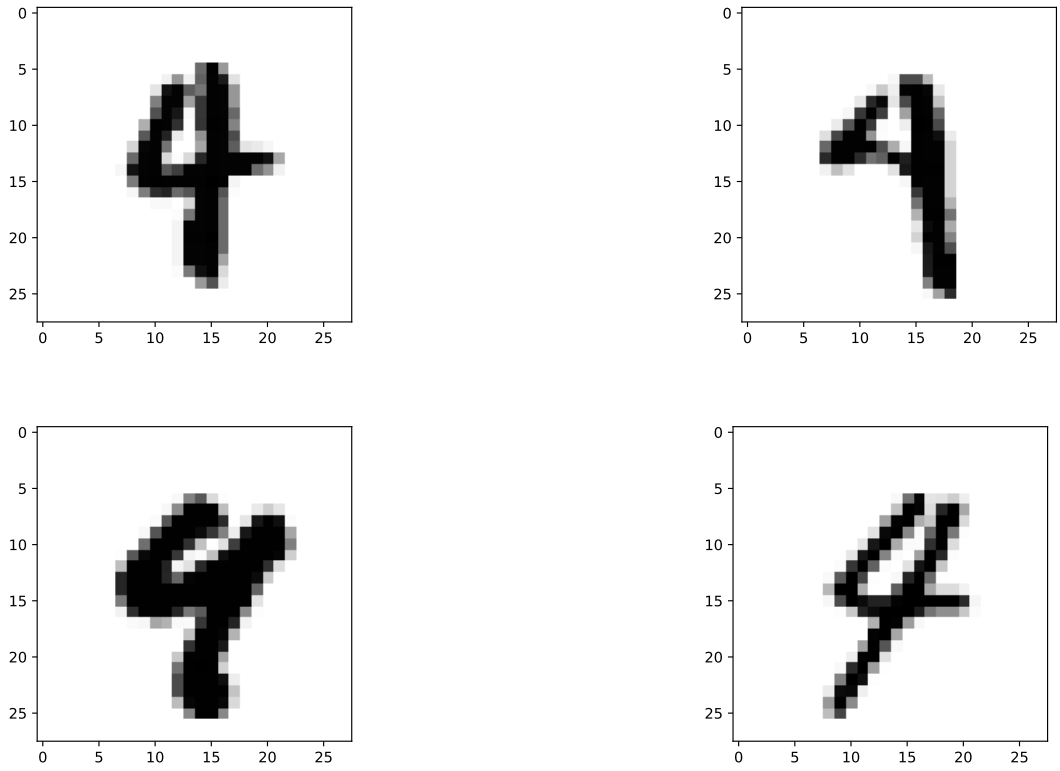


Рис. 4: Четверки второй группы

Третий тип - самые обыкновенные четверки в виде буквы Ц:

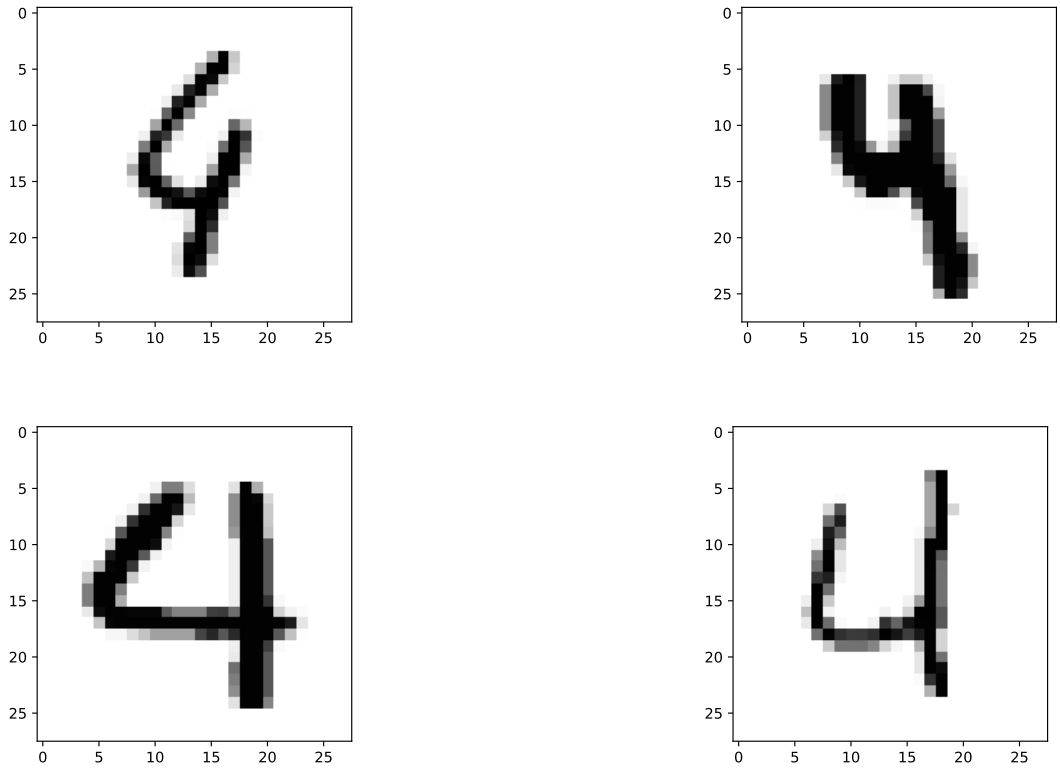


Рис. 5: Четверки третьей группы

Это разбиение интуитивно - мы распознаем цифры, как правило, по таким признакам, как наличие замкнутых областей на ее изображении, наличие закорючек, угол линий. Однако построенная в этом исследовании модель не смотрит на связи между пикселями - она рассматривает их по отдельности, и вряд ли сможет заметить такой признак, как замкнутость части фигуры, требующий анализа немалой окрестности. Проинтерпретируем каждую из выделенных групп на языке модели и выясним, почему же классификатор ошибся на них.

Подумаем о том, из чего состоит девятка. У девятки есть верхняя полуокружность, то есть большое горизонтальное скопление пикселей сверху картинки. Такое же скопление есть у всех четверок первой группы. Вот почему их классфицировало неверно.

Также у девятки есть жирная нижняя полуокружность - все четверки из второй группы также имеют ее. Но поскольку четверке свойственна только горизонтальная линия и две "палки а девятке - еще и верхняя полуокружность, то "страдающие от переизбытка пикселей"четверки второй группы могут восприниматься как девятки.

Накоонец, в третьей группе четверки расположены на картинке нетипично - слишком широко. Поэтому при подсчете расстояния оно будет болльшим до других четверок (склонных компактно находиться в центре) и меньшим до фигур, более склонным "растекаться к краям".

Распространю эту логику на 7, распознаваемые как 9:

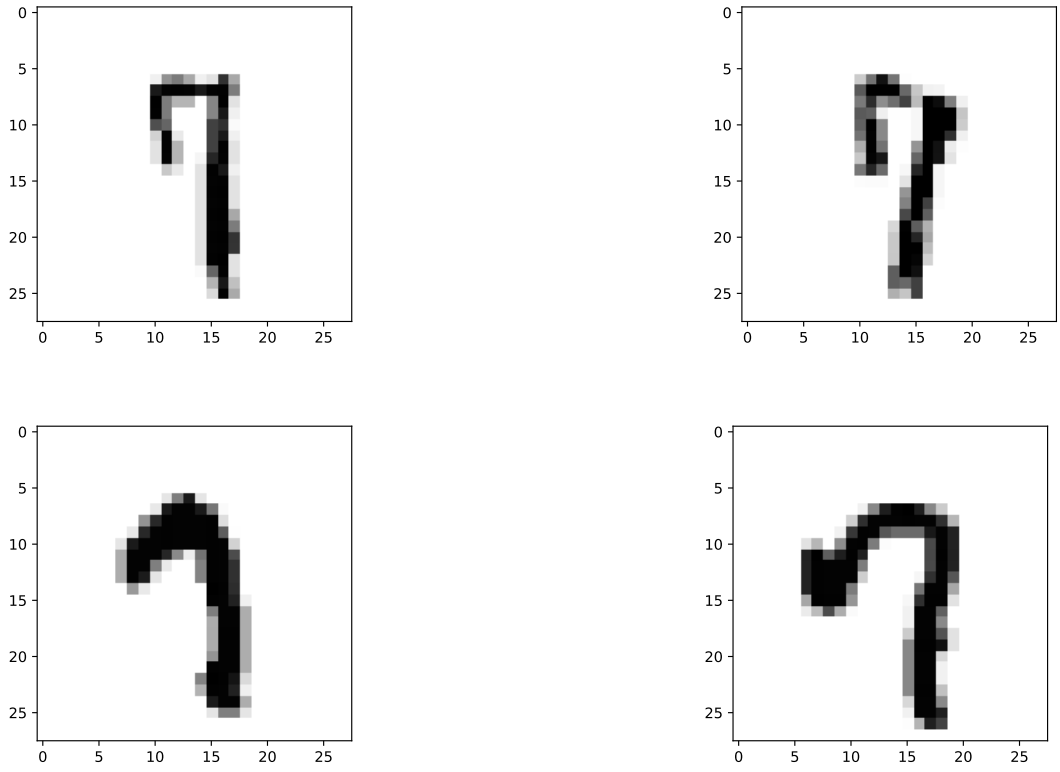


Рис. 6: Типичные семерки, распознанные как девятки

Здесь классификатор скорее всего среагировал на большую загнутую вниз "крышу"семерок, похожую на верхнюю полуокружность девяток.

Делаю вывод: допускаемые классификатором ошибки сходятся в том, что он считает близкими фигуры, у которых совпадает достаточное количество пикселей. Поэтому фигура класса A, содержащая загогулину на пикселях, типично закрашенных для фигур класса B и пустых для фигур класса A, будет распределена в класс B, что и было продемонстрировано.

3.5 Эксперимент 5

Цель пятого эксперимента - посмотреть, как использование аугментации влияет на точность классификаторов. Я рассмотрел три ее типа:

- поворотом (на $-15. - 10, -5, 5, 10, 15$ градусов соответственно)
- сдвигом (на $-3, -2, -1, 1, 2, 3$ пискеля вдоль каждой из осей
- использованием фильтра Гаусса со значениями $\sigma = 0.5, 1, 1.5$ (при $\sigma = 0.5$ размытость минимальна, при $\sigma > 1.5$ данные сильно портятся)

Поставлю такой опыт: применю к обучающей выборке каждый из трех методов аугментации с каждым из возможных значений параметров и посмотрю, как изменились качество и типичные ошибки.

Отмечу специально, что аугментация применяется только к обучающей части выборки, тестовая при этом неизменна.

Сначала применим сдвиги, полученные значения точности я занесу в таблицу:

Ось / величина сдвига	-3	-2	-1	1	2	3
x	0.917	0.954	0.972	0.970	0.950	0.907
y	0.920	0.946	0.970	0.971	0.946	0.901

Таблица 3: Точность классификации при аугментации обучающей выборки сдвигом

При больших сдвигах точность нарушается, так как пиксели у сходим фигур не находятся в конкретных местах полотна.

Теперь фильтр Гаусса:

σ	0.5	1	1.5
$accuracy$	0.971	0.724	0.476

Таблица 4: Точность классификации при аугментации обучающей выборки применением фильтра Гаусса

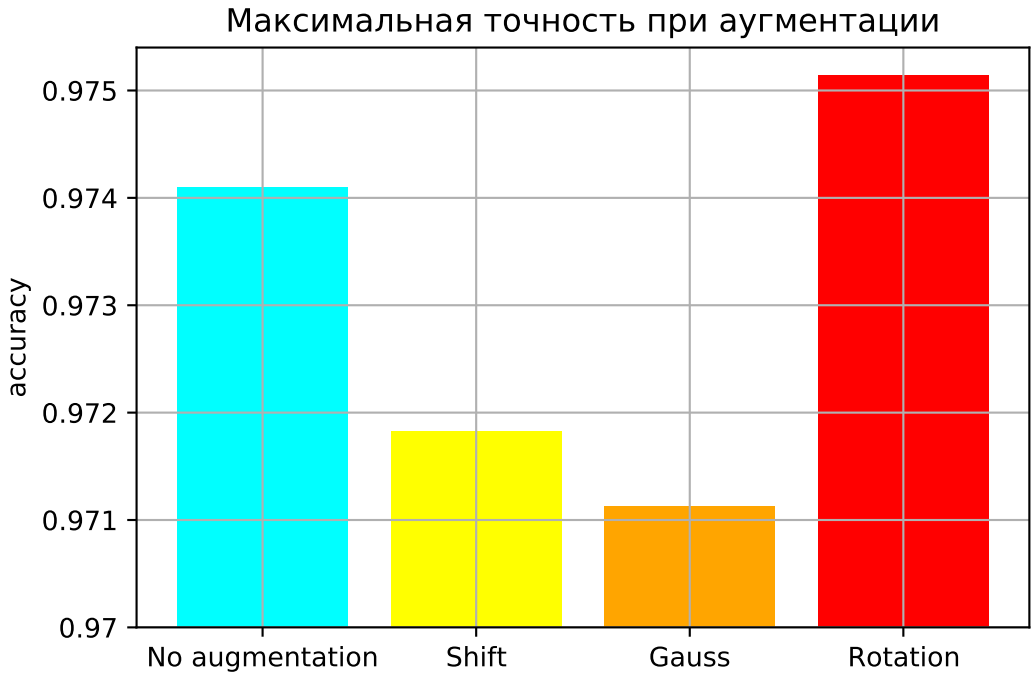
Видим, что большая дисперсия фильтра Гаусса сильно портит классификацию - и немудрено: основные пиксели фигуры размываются и вносят меньший вклад в опредление близости из-за появившегося множества других пикселей по краям.

И повороты:

σ	-15	-10	-5	5	10	15
$accuracy$	0.970	0.973	0.974	0.975	0.974	0.971

Таблица 5: Точность классификации при аугментации обучающей выборки поворотами

Обобщим полученные результаты:



Только повороты позволяют улучшить качество классификации.

Посмотрим на $confusionmatrix$ лучших классификаторов для каждого из способов аугментации:

968	2	2	1	0	4	10	2	4	2
0	1123	2	4	23	0	2	35	1	8
7	3	1014	1	1	0	0	8	4	4
1	0	3	975	0	26	1	0	59	15
0	0	0	0	923	2	1	2	2	7
1	1	0	18	0	841	7	0	10	0
1	5	1	0	7	3	936	0	4	0
2	0	6	7	4	2	0	979	4	59
0	1	4	2	2	6	1	0	881	4
0	0	0	2	22	8	0	2	5	910

Таблица 6: Confusion matrix сдвига

При переносе главные ошибки - 7 распознаются как 9, 3 распознаются как 8, 1 распознаются как 7.

975	0	10	1	2	4	4	0	9	7
1	1131	4	1	4	0	3	15	1	5
0	3	1005	2	0	0	0	4	2	2
0	0	0	978	0	11	0	0	9	5
0	0	1	1	949	1	2	0	4	6
0	0	0	9	0	862	2	0	4	2
2	1	0	0	5	7	947	0	4	1
1	0	10	5	1	1	0	999	4	8
1	0	2	8	0	3	0	0	934	4
0	0	0	5	21	3	0	10	3	969

Таблица 7: Confusion matrix гауссовского фильтра

При использовании Гауссовского фильтра главные ошибки остаются те же, что и без него - 7 распознаются как 9, 4 как 9, 1 как 7.

977	0	8	1	1	4	4	2	3	6
1	1128	3	0	2	0	3	10	2	5
0	3	1003	2	0	0	0	5	3	0
0	1	2	977	0	6	0	0	7	6
0	0	1	1	953	2	1	3	6	9
0	0	0	8	0	859	0	0	5	3
1	3	1	0	7	13	950	0	5	1
1	0	10	7	0	1	0	986	3	3
0	0	4	8	0	2	0	0	934	2
0	0	0	6	19	5	0	22	6	974

Таблица 8: Confusion matrix поворота

При повороте также типы ошибок все те же.

Неожиданный результат - проваливший кросс-валидацию фильтр Гаусса показывает очень хороший результат без нее (0.9749), поворот показывает себя несколько хуже (0.9741), сдвиг же очень плох (0.955).

3.6 Эксперимент 6

Цель шестого эксперимента - посмотреть, насколько аугментация тестовой выборки и последующее усреднение ответов на ней голосованием влияет на качество классификации.

Для эксперимента я устроил три голосвания с разными избирателями:

- оптимальные сдвиг (-1 пиксель вдоль оси X), гауссовский фильтр ($\sigma = 0.5$) и поворот (-5 градусов) из пятого эксперимента
- гауссовский фильтр ($\sigma = 0.5$) и два вращения на -5 и 5 градусов - я выбрал такой набор, потому что сдвиги обычно показываают себя плохо, а вращения на маленькие углы наоборот хорошо
- голосуют все описанные в эксперименте 5 преобразования

Итоговые точности:

- 0.9719
- 0.975
- 0.9776

соотвественно. Первый подход проявил себя плохо, второй продемонстрировал точность, схожую с точностью пятого эксперимента, а вот последний неожиданно дал очень хорошую точность, более чем на 0.002 превышающий лучшую полученную до этого. При этом все эти подходы несильно повлияли на типы ошибок классификации:

0	1	2	3	4	5	6	7	8	9
978	0	10	0	2	6	4	2	6	12
1	1130	1	0	3	1	2	10	0	4
0	2	1005	1	0	0	0	8	3	2
0	1	1	986	0	11	0	0	9	3
0	0	1	1	945	1	1	0	3	7
0	0	0	5	0	856	1	1	1	5
0	2	2	0	6	10	949	0	4	1
1	0	7	4	1	1	0	992	3	5
0	0	5	9	1	4	1	0	942	3
0	0	0	4	24	2	0	15	3	967

Рис. 7: Confusion matrix первого голосования

0	1	2	3	4	5	6	7	8	9
978	0	10	0	2	6	4	2	6	12
1	1130	1	0	3	1	2	10	0	4
0	2	1005	1	0	0	0	8	3	2
0	1	1	986	0	11	0	0	9	3
0	0	1	1	945	1	1	0	3	7
0	0	0	5	0	856	1	1	1	5
0	2	2	0	6	10	949	0	4	1
1	0	7	4	1	1	0	992	3	5
0	0	5	9	1	4	1	0	942	3
0	0	0	4	24	2	0	15	3	967

Рис. 8: Confusion matrix второго голосования

0	1	2	3	4	5	6	7	8	9
976	0	12	0	2	4	7	2	1	13
1	1131	0	0	2	0	3	9	1	7
0	2	1007	2	0	0	0	5	2	2
0	1	0	988	0	10	0	0	5	3
0	0	1	0	951	1	1	1	2	8
0	0	0	5	0	861	0	0	3	2
1	1	0	0	7	8	947	0	2	1
1	0	10	4	1	1	0	999	4	5
1	0	2	6	2	5	0	1	952	4
0	0	0	5	17	2	0	11	2	964

Рис. 9: Confusion matrix третьего голосования

4 Заключение

При помощи самостоятельно написанных классов и процедур я исследовал работу метода ближайших соседей на датасете $MNIST$. Такой классификатор может достигать точности в 0.9776, что немногим меньше точности лучших алгоримов 0.9881. При использовании метода ближайших соседей наибольшую точность демонстрирует косинусная метрика с использованием весов. Аугментация тестовой выборки может сильно улучшить качество классификации, в то время как эффект от аугментации обучающей выборки меньше (хоть и легче интерпретируем).