Метрический алгоритм классификации kNN

Сорокин Олег, 317

каф. ММП, ВМК МГУ
14.10.2022

Содержание

1 Введение **2**

2 Список экспериментов **2**
2.1 Сравнение по времени алгоритмов поиска ближайших соседей 2
2.1.1 Дизайн эксперимента 2
2.1.2 Результаты 2
2.1.3 Выводы 2
2.2 Влияние гиперпараметров модели на её точность 3
2.2.1 Дизайн эксперимента 3
2.2.2 Результаты 3
2.2.3 Выводы 4
2.3 Влияние наличия весов, зависящих от расстояний, на точность предсказаний . . 5
2.3.1 Дизайн эксперимента 5
2.3.2 Результаты 5
2.3.3 Выводы 6
2.4 Анализ множества ошибочных предсказаний лучшего алгоритма 6
2.4.1 Дизайн эксперимента 6
2.4.2 Результаты 6
2.4.3 Выводы 6
2.5 Влияние расширения выборки путём аугментации на точность предсказаний . . 7
2.5.1 Дизайн эксперимента 7
2.5.2 Результаты 8
2.5.3 Выводы 8
2.6 Влияние расширения тестовой выборки аналогичной аугментацией на точность 8
2.6.1 Дизайн эксперимента 8
2.6.2 Результаты 8
2.6.3 Выводы 8

3 Выводы **8**

4 Аппендикс **9**
4.1 Пояснение к сравнению качества работы при различных метриках 9
4.2 Разности матриц из эксперимента 5 10

1 Введение

Алгоритм KNN - метрический алгоритм, применяющийся в задачах классификации или регрессии. Является достаточно простым и популярным (в том числе и в промышленности).

Далее будем рассматривать только модель классификации на основе KNN. Пользуясь собственной реализацией на Python с использованием библиотеки numpy и алгоритмами sklearn на примере данных из MNIST:

- сравним эффективность различных алгоритмов поиска в условиях большой размерности признакового пространства;
- рассмотрим проблемы роста временных затрат с ростом числа данных или их размерности;
- предложим возможные методы подбора гиперпараметров с целью увеличения качества модели. Ориентиром качества выберем модели классификации, упоминания о которых есть на Kaggle, и с помощью которых решалась задача с такими же данными;
- рассмотрим иные методы увеличения точности классификации.

2 Список экспериментов

2.1 Сравнение по времени алгоритмов поиска ближайших соседей

2.1.1 Дизайн эксперимента

Выборка содержит 70тыс. grayscale-изображений цифр от 0 до 9 размера 28x28 пикселей. Разобьём её на тренировочную (первые 60тыс.) и тестовую (остальные 10тыс.). Будем искать $k = 5$ ближайших соседей из второй выборки для каждого объекта первой выборки, сравнивая время, затрачиваемое каждым алгоритмом. Предварительно получим данные разных размерностей, отбросив часть признаков для объектов обеих выборок следующим образом:

- Каждое из изображений по строкам вытягиваем в вектор длины 28*28 = 784
- От каждого из векторов $(x_1, x_2, ..., x_{784})$ выберем подвекторы $(x_1, x_2, ..., x_s)$, где $s \in \{10, 20, 100\}$

Измерения будем производить, подавая на вход часть описание единственного объекта тестовой выборки. Зафиксируем гиперпараметры модели: метрика - евклидова, $k = 5$ - выбрано ранее. Сравним время выполнения поставленной задачи для алгоритмов поиска: brute-перебор, kd-tree, ball-tree из библиотеки sklearn, а также для реализации my_own на основе построения матрицы попарных расстояний.

2.1.2 Результаты

Средние значения времени по каждой из рассматриваемых размерностей приведены в таблице:

Алгоритм	$t_{cp}, dim = 10$	$t_{cp}, dim = 20$	$t_{cp}, dim = 100$
my_own	0.00205	0.0034	0.01246
brute	0.00124	0.00144	0.00486
kd-tree	0.00133	0.00208	0.00742
ball-tree	0.00119	0.00165	0.00637

2.1.3 Выводы

В данной задаче лучше всего себя проявил bruteforce алгоритм. Его сложность, согласно документации, составляет $O(DN^2)$, где D - размерность данных, N - число элементов. Для сравнения, сложность, например, kd-tree составляет $O(D * N * logN)$. Из полученных результатов следует, что константа О большого в общем случае может быть велика. Значит, для достаточно малого числа элементов и достаточно малых размерностей эффективнее использовать bruteforce.

Стоит, однако, заметить, что результаты получены при значительном ограничении $N = 1$. С ростом N kd-tree начинает проявлять свою эффективность.

2.2 Влияние гиперпараметров модели на её точность

2.2.1 Дизайн эксперимента

Исследуем, как метрика и число ближайших соседей влияют на точность классификации рассматриваемых данных из MNIST. Каждой из двух исследуемых метрик (евклидова и косинусная) поставим в соответствие классификатор. Зафиксируем стратегию поиска классификаторов и схему использования весов. Например, возьмём алгоритм my_own с равномерными весами для каждого соседа. Далее проверим все $k = 1, ..., 10, 250$ для классификатора с евклидовой метрикой и для классификатора с косинусным расстоянием. При этом будем использовать кросс-валидацию с тремя фолдами. Усреднив результаты по фолдам, получим средние точности для каждого k. Далее сделаем выводы о том, при каких k показал лучшую точность каждый из классификаторов.

2.2.2 Результаты

Результаты эксперимента отражены на рисунках 1-3:

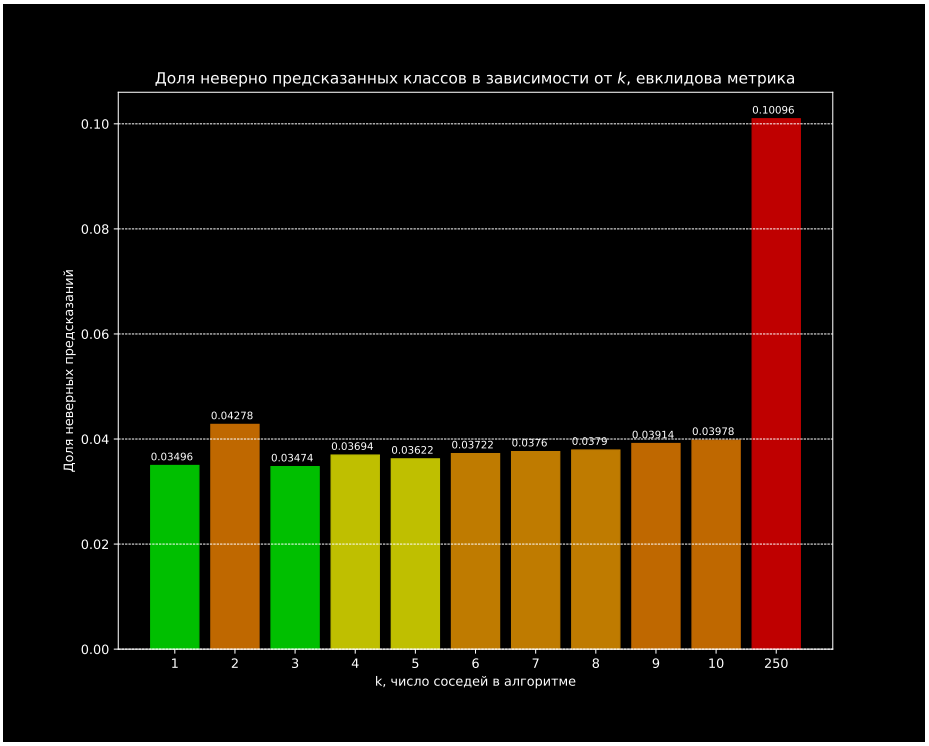

Рис. 1: Результат для евклидовой метрики

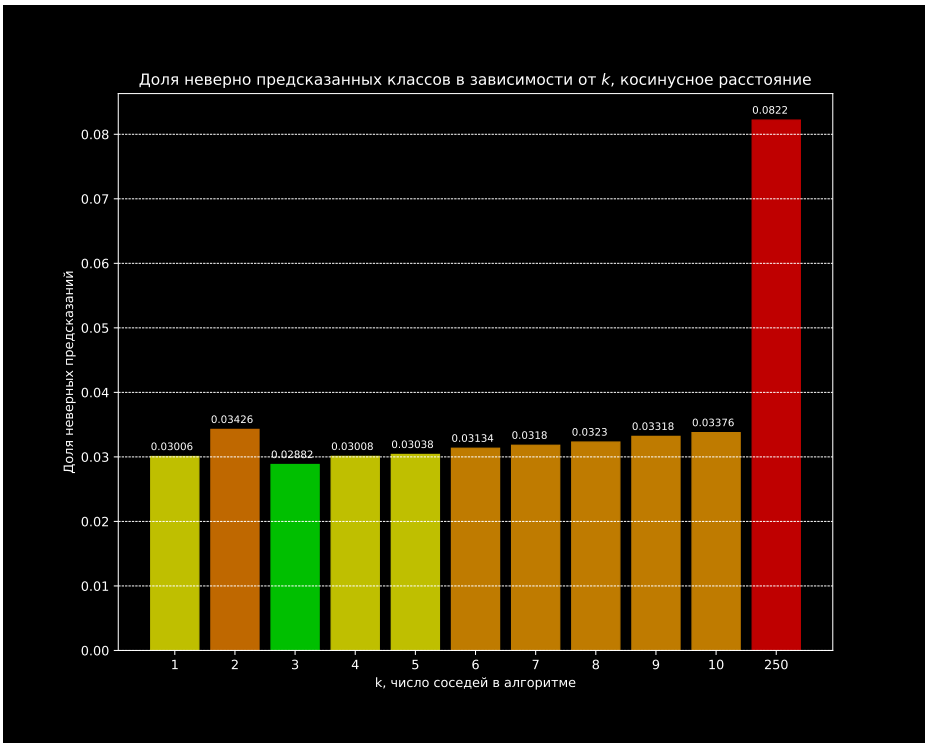

Рис. 2: Результат для косинусного расстояния

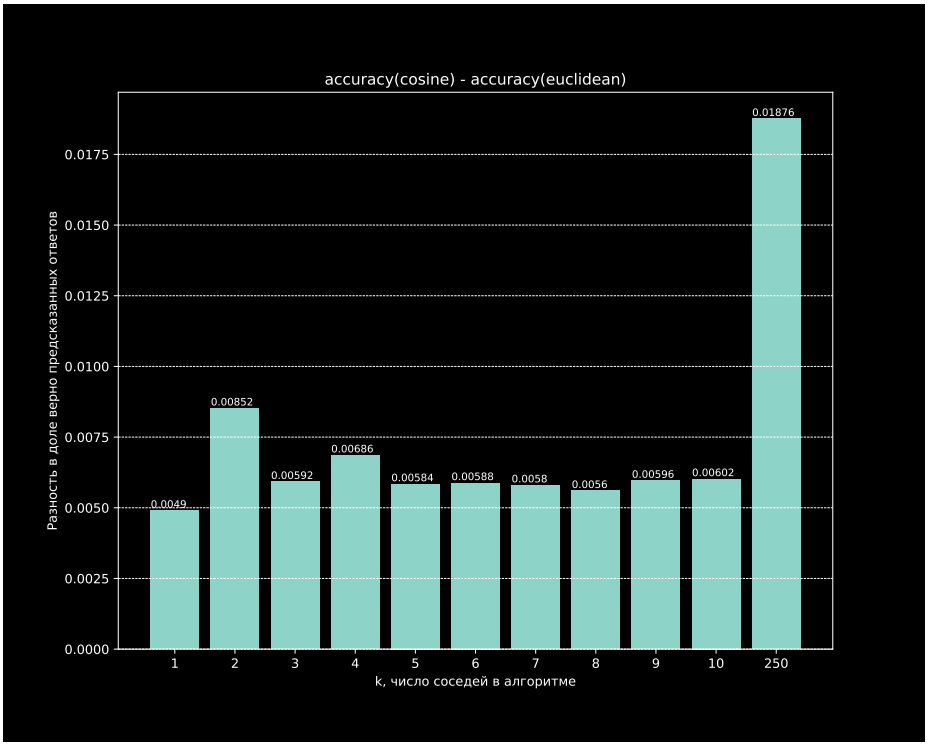

Рис. 3: Разностное сравнение двух метрик

2.2.3 Выводы

Наибольшая точность достигается при некотором промежуточном значении k, то есть $k > 1$ и $k << |X_{train}|$. Действительно, $k = 1$ соответствует самой сложной модели в том смысле, что происходит запоминание позиций, что ведёт к переобучению модели. И напротив, большие значения k означают, что модель переупрощена, то есть неспособна выискивать паттерны в виде групп ближайших соседей одного класса. Частично эта проблема может быть решена с

помощью весовых схем (см. след. эксперимент). На графиках можно видеть, что некоторые значения могут работать лучше соседних. Объяснение этому состоит в том, что в выборке в кластерах объектов могут присутствовать выбросы. При малых значениях k модель может обнаружить только эти выбросы и ошибиться, а при больших - обнаружить объекты других классов за пределами кластера, что тоже приведёт к ошибочной классификации.

На рис. 3 можно видеть, что при всех значениях k лучше себя показало косинусное расстояние. Возможные объяснения:

- Евклидова метрика подвержена проклятию высокой размерности (784 признака).
- В многомерном пространстве для однозначного определения точки нужно выбрать совокупность углов, откладываемых от осей и длину вектора. Длина вектора при фиксированных углах в рассматриваемой задаче соответствует яркости изображения, при этом при изменении длины вектора класс соответствующего объекта не меняется. Косинусное расстояние, в отличие от евклидовой метрики, игнорирует монотонные перемены в яркости изображения, а некоторые изображения из датасета действительно могут отличаться лишь своей интенсивностью (см. раздел 4.1, аппендикс).

2.3 Влияние наличия весов, зависящих от расстояний, на точность предсказаний

2.3.1 Дизайн эксперимента

Проверим, увеличится ли точность при замене единичных весов на убывающие от расстояния. Зафиксируем некоторую конфигурацию модели, например, my_own с косинусным расстоянием. Для $k = 1, ..., 10, 250$ сравним средние точности по трём фолдам для алгоритма без весов и для алгоритма, где вес i-го соседа объекта x определяется как

$$W(i, x) = \frac{1}{\rho(x, x^{(i)}) + \epsilon}, \epsilon = 10^{-5}$$

2.3.2 Результаты

Результат сравнения точностей представлен на графике:

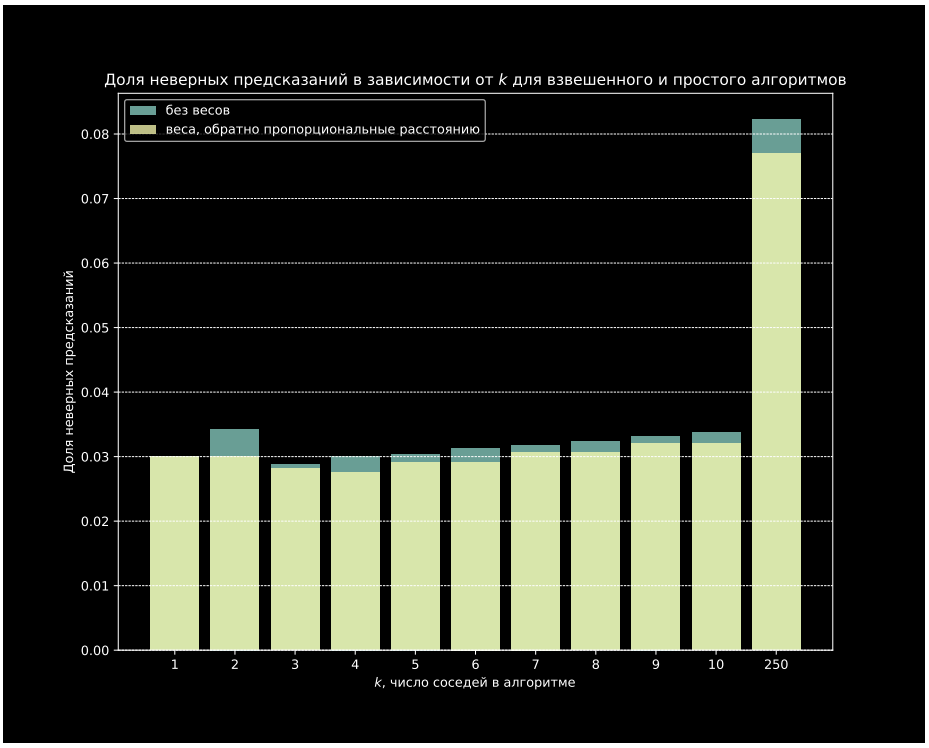

Рис. 4: Результат сравнения весовых схем KNN

2.3.3 Выводы

Наличие весов, заданных указанным способом, повышает точность классификации.

2.4 Анализ множества ошибочных предсказаний лучшего алгоритма

2.4.1 Дизайн эксперимента

Из предыдущих экспериментов следует, что лучшим набором параметров в смысле точности является весовой алгоритм с косинусным расстоянием $k = 4$. В качестве алгоритма поиска выберем bruteforce.

Определим тренировочную и тестовую выборки как в эксперименте 1. Подсчитаем точность алгоритма на тестовой выборке, сравним с точностью по кросс-валидации. Выполним анализ ошибок на объектах тестовой выборки.

2.4.2 Результаты

На тренировочной выборке точность достигла 1.0, на тестовой выборке – 0.9752. При разбиении по кросс-валидации (с тремя фолдами) исходной выборки для этих же параметров получим точности (0.97366053, 0.97174057, 0.97131885).

Матрица ошибок имеет вид, указанный на рис. 5.

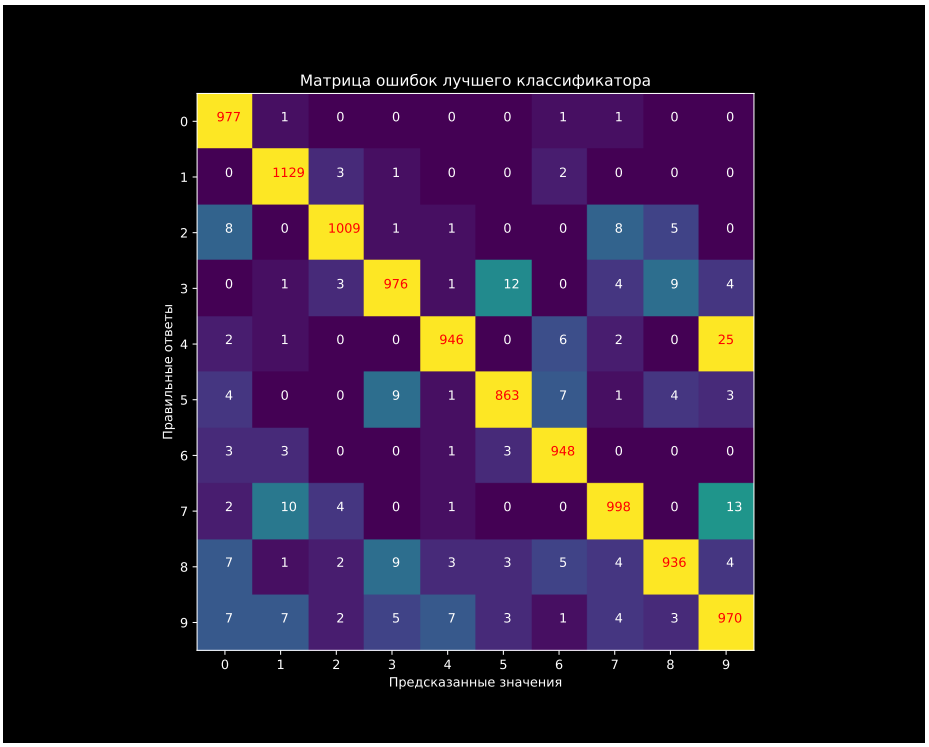

Рис. 5: Матрица ошибок лучшего классификатора

2.4.3 Выводы

Лучший алгоритм на тестовой выборке показал примерно ту же точность, что и на кросс-валидации. Практически все современные алгоритмы показывают более высокие результаты (а у топовых алгоритмов точность на MNIST достигает порядка 0.99).

Анализируя матрицу ошибок, можно сказать следующее:

- Больше всего ошибок суммарно допущено при распознавании цифры 9 (39 ошибок).
- Самая частая ошибка классификатора - отклик 9 на объекте класса 4 (25 ошибок).

На рис. 6 приведены примеры изображений, распознанных неверно. Среди их особенностей можно выделить нетипичность написания отдельных элементов цифры (некоторые и вовсе не дописаны). И, как можно видеть из примеров, это зачастую делает цифры плохо различимыми.

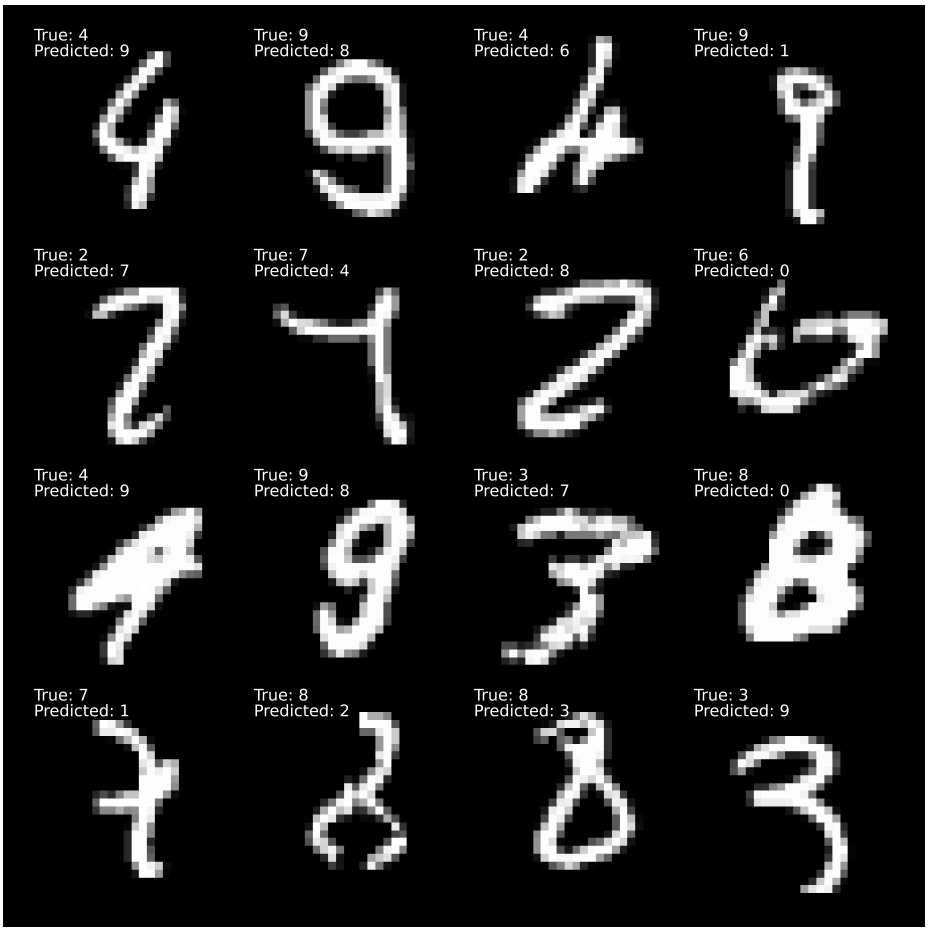

Рис. 6: Примеры неверно классифицированных изображений

2.5 Влияние расширения выборки путём аугментации на точность предсказаний

2.5.1 Дизайн эксперимента

Будем рассматривать следующие виды аугментации:

1) Случайный поворот в обоих направлениях на 5, 10, 15 градусов.

2) Смещение на 1, 2, 3 пикселя по обеим осям.

3) Гауссовский фильтр с ядром 15, дисерсиями 0.5, 1.0, 1.5.

4) Морфологические преобразования: эрозия, дилатация, открытие, закрытие с ядром 2.

Зафиксируем модель, описанную в предыдущем эксперименте. Реализуем жадный алгоритм поиска приводящих к увеличению точности преобразований над списом из n преобразований:

0. Преобразования к тренировочной выборке не применены, считаем точность модели.

1. Перебираем все параметры первого преобразования, выбираем наилучшее. Если качество ухудшилось по сравнению с предыдущей итерацией, переходим к следующей итерации с исходной выборкой, иначе - с преобразованной.

2...n). Аналогично шагу 1.

На шаге n мы находим оптимальную комбинацию преобразований.

Рост выборки - экспоненциальный, поэтому аугментировать на каждой итерации будем 20000 объектов.

2.5.2 Результаты

Изначальная точность – 0.9752. В результате работы алгоритма были последовательно отобраны преобразования:

- Поднятие точности до 0.978 поворотами на 10 градусов.
- Поднятие точности до 0.9787 сдвигами на пиксель.
- Поднятие точности до 0.9791 применением гауссовского фильтра с дисперсией 1.5.
- Поднятие точности до 0.9797 применением открытия с ядром 2.

2.5.3 Выводы

В разделе 4.2 (аппендикс) приведены 4 матрицы, каждая из которой есть разница новой и старой матриц ошибок (до и после аугментации). Используем их для отслеживания динамики ошибок в различных случаях.

Поворот изображений наиболее поднял точность распознавания 3 и 9, а именно уменьшился процент предсказания вместо них 5 и 3 соответственно. Кроме того, лучше распознаётся 7 (не путается с единицей).

Сдвиг незначительно увеличил процент распознавания 8 и 9.

В исходном датасете находилось много недописанных цифр 8, гауссовский фильтр частично исправил эту ситуацию, уменьшив процент неверных предсказаний восьмёрок. Однако в связи с наложением шума на похожие изображения возникли ошибки с парами 3-5, 9-7, поэтому изменение точности не очень велико.

Морфологические операции незначительно повлияли на некоторые ошибки, возникшие при аугментации с помощью фильтра Гаусса.

2.6 Влияние расширения тестовой выборки аналогичной аугментацией на точность

2.6.1 Дизайн эксперимента

Обучим алгоритм на оригинальной подвыборке из 60000 элементов. Подобно предыдущему эксперименту будем применять жадный алгоритм, но аугментировать в этот раз будем тестовую выборку. Сделаем выводы о динамике ошибок, построив соответствующие матрицы.

2.6.2 Результаты

В результате работы жадного алогоритма нашлось только одно преобразование тестовой выборки, повышающее точность - гауссовский фильтр с дисперсией 0.5. Соответствующая матрица приведена на рис. 7.

2.6.3 Выводы

Метод, описанный в эксперименте 5, привёл к значительному увеличению точности. Аугментирование тестовой выборки практически не повлияло на точность. Возможно, её изменение обусловлено тем, что фильтр Гаусса сильнее всего повлиял на объекты, которые модель часто классифицировала неверно.

3 Выводы

Из существующих алгоритмов поиска ближайших соседей нельзя выбрать универсальный. Как было показано, даже bruteforce алгоритм может быть эффективен в задачах высокой размерности при малых данных, тогда как деревья могут проигрывать ему в этой ситуации.

Не существует и единого подхода в выборе гиперпараметров: в задачах низких размерностей, рассматриваемых на Kaggle, высокие результаты показывает евклидова метрика. В рассматриваемой здесь задаче по указанным ранее причинам косинусное расстояние приводит к большему увеличению точности модели.

Однозначно положительный эффект на работу модели оказывает аугментация обучающей выборки. В условиях недостаточного количества данных (а при высокой размерности признакового пространства выборка не содержит достаточно информации о каждом единичном объекте) она является надёжным методом повышения точности.

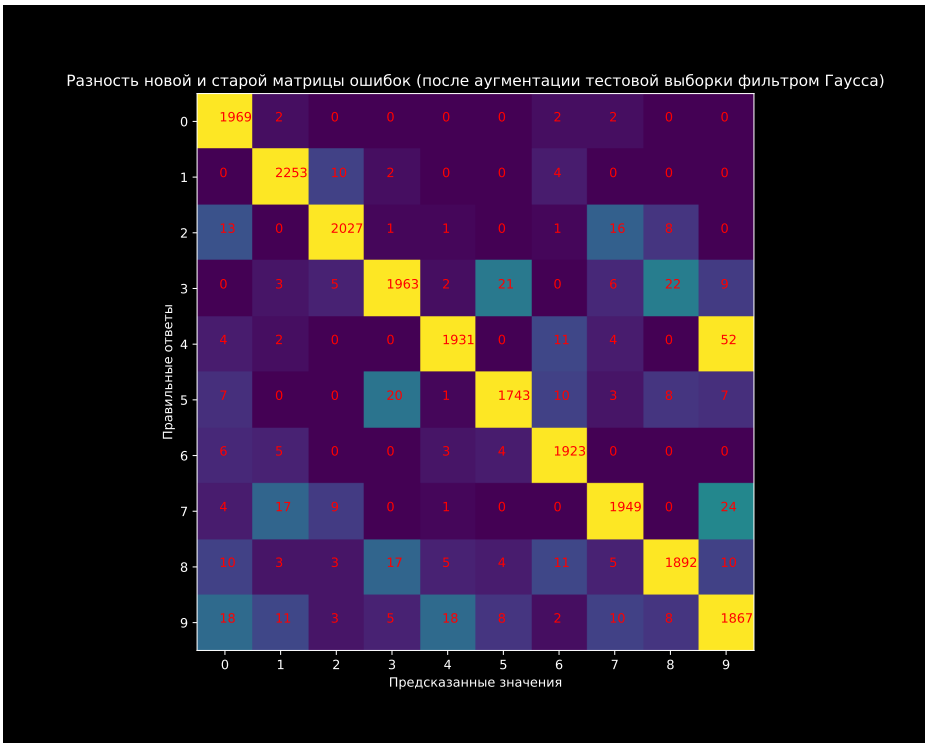

Рис. 7: Матрица ошибок после аугментации тестовой выборки фильтром Гаусса

4 Аппендикс

4.1 Пояснение к сравнению качества работы при различных метриках

Рассмотрим следующие изображения (второе инвертировано для лучшей различимости):

Рис. 8: Иллюстрация к примеру о сравнении метрик

Пронумеруем их слева направо от 1 до 3. Матрица M попарных евклидовых расстояний имеет вид:

	1	2	3
1	0	3160	3139
2	3160	0	2225
3	3139	2225	0

Для попарных косинусных расстояний имеем:

	1	2	3
1	0.0	0.14	0.59
2	0.14	0.0	0.62
3	0.59	0.62	0.0

Объекты 1 и 2 следует отнести к одному классу, 1 и 3 - к разным. Однако при этом евклидово расстояние между объектами 1 и 2 больше расстояния между объектами 1 и 3, что может

привести к ошибке классификации. Косинусное расстояние, как было описано ранее, менее чувствительно к смене яркости.

4.2 Разности матриц из эксперимента 5

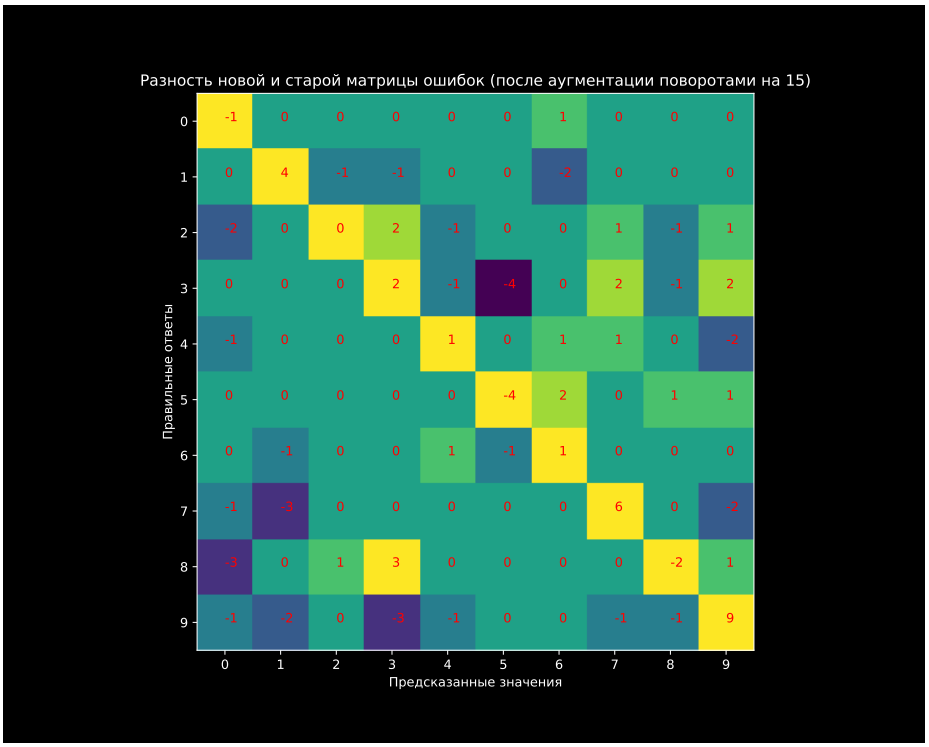

Рис. 9: Динамика ошибок при применении поворотов

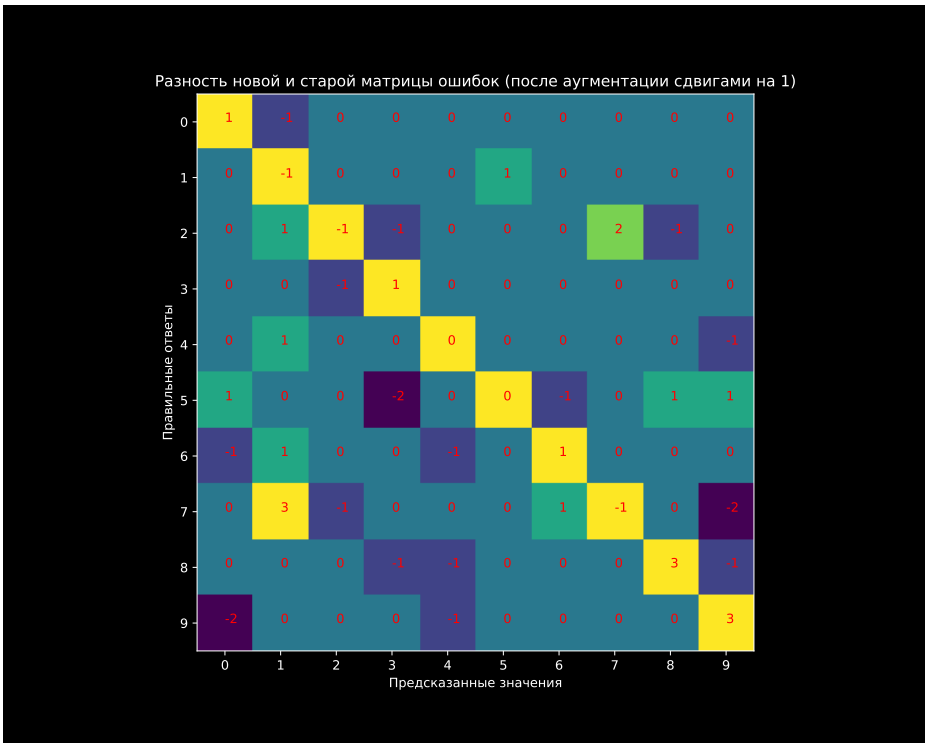

Рис. 10: Динамика ошибок при применении сдвига

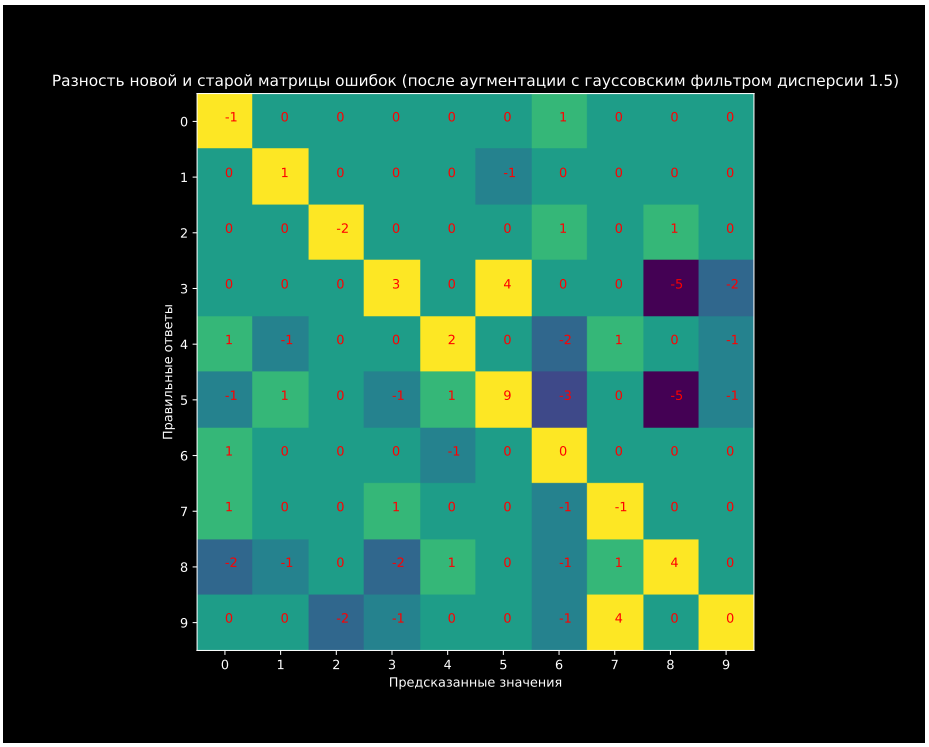

Рис. 11: Динамика ошибок при применении гауссова фильтра

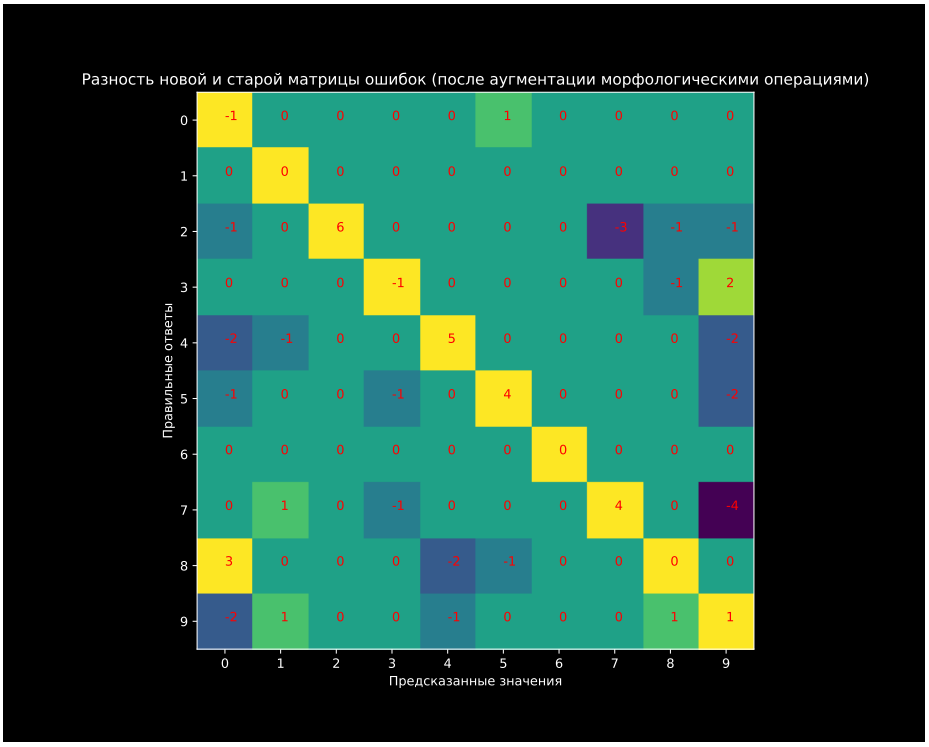

Рис. 12: Динамика ошибок при применении эрозии, дилации, открытия и закрытия