Отчет о практическом задании «Метрические алгоритмы классификации».

Практикум 317 группы, ММП ВМК МГУ.

Марьясов Максим Михайлович.

Октябрь 2024.

Содержание

1 Введение

Данное задание направлено на изучение метрических алгоритмов классификации на основе датасета изображений MNIST. Цель исследования - изучение зависимости точности и времени от выбранного алгоритма, метрики сравнения, и изучение методов аугментации изображений. Также в практическом задании рассматриваются роль использованных аугментаций на улучшения качества классификации.

2 Постановка задачи

2.1 Постановка задачи классификации в метрических методах

Дано:

X - объекты, Y - ответы.

Обучающая выборка $X_l = \{x_i, y_i\}_{i=1}^l$

Задача классификации в метрических методах основана на **гипотезе компактности**, которая гласит, что **близкие объекты, как правило, лежат в близких классах**.

Для определения расстояния между объектами используется **евклидова метрика** и **косинусная метрика**.

Евклидова метрика:

$p(x, x_i) = \sqrt[2]{\sum_{i=1}^{n} |x^j - x_i^j|^2}$, где $x = (x^1, x^2, .., x^n)$ и $x_i = (x_i^1, x_i^2, .., x_i^n)$ - вектор признаков, соответственно, x и x_i.

Косинусная метрика: $p(x, x_i) = 1 - \frac{x * x_i^T}{||x|| * ||x_i||}$, где $x = (x^1, x^2, .., x^n)$ и $x_i = (x_i^1, x_i^2, .., x_i^n)$ - вектор признаков, соответственно, x и x_i.

2.2 Метод k ближайших соседей (KNN)

Метод k ближайших соседей (k-nearest neighbors, KNN) - метрический алгоритм машинного обучения.

Для произвольного $x \in X$ отранжируем объекты $x_1, ..., x_l$:

$p(x, x^{(1)}) \leq p(x, x^{(2)} \leq ... \leq p(x, x^{(l)}$,

где $x^(i)$ - i-ый сосед обьекта x среди $x_1, ..., x_l$;

$y^(i)$ - ответ на i-ом соседе объекта x.

Метрический алгоритм классификации относите объект x к тому классу, которому принадлежат ближайшие его соседи.

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

где $w(i, x)$ - вес, степень близости к объекту x его i-ого соседа, не отрицателен, не возврастает по i.

Рассмотрим различные методы выбора $w(i, x)$:

$w(i, x) = [i \leq 1]$ - метод ближайшего соседа. $w(i, x) = [i \leq k]$ - метод k ближайших соседей.

Значительный плюс данного алгоритма это то, что происходит **lazy learning**, потому что алгоритм во время обучения только запоминает обучающую выборку. Затем уже на её основе предсказывает класс объекта x.

Параметр k оптимизируем по **leave-one-out**:

$LOO(k, X^l) = \sum_{i=1}^{l} [a(x_i; X^l \backslash \{x_i\}, k) \neq y_i] \to \min$

Так как расстояние до соседей не учитываются, то происходит расхождения, и данный пробел решает **метод взвешенных k ближайших соседей**:

$w(i, k) = [i \leq k] w_i$, где w_i - вес, зависящий только от номера соседа.

2.3 Методы поиска KNN

Наивный алгоритм.

Отсортировать всю выборку по значению $p(q,)$ и взять срез из первых k элементов. Сложность по времени $O(nlogn)$.

Поиск k-ой порядковой статистики.

Существует алгоритм поиска k-ой порядковой статистики, сложность которого $O(n)$ сравнений. Этот алгоритм изменяет порядок в массиве таким образом, что первые k элементов меньше, чем все последующие.

kd-tree.

Данный метод заключается в построении сбалансированного дерева, листья которого соответствуют объектам выборки X.

Рассмотрим процедуру построения такого дерева. Пусть $x_i \in R$

n — это j-ый столбец матрицы признаков.

1. Найти медиану x_1 и с ее помощью разделить X на две половины.
2. Для каждой из половин отдельно найти медианы x_2 и разделить эти половины еще раз пополам.
3. Повторять эту процедуру вплоть до x_d и затем продолжить с x_1.
4. Если какое-то разбиение содержит лишь один объект, то получен лист.
5. И так далее до тех пор, пока все объекты не будут распределены в свои листья.

Если размерность пространства небольшая (10-20), то сложность построения дерева $O(n * log(n))$. Заметим, что функция расстояния не используется при построении дерева.

Поиск осуществляется путем передвижения от корня к листьям и постоянными проверками на факт того, нет ли в соседнем сплите более близких соседей, чем в данном. Экспериментально было установлено, что в пространствах большой размерности сложность поиска ближайшего соседа в kd-дереве сильно ухудшается и приобретает линейный порядок сложности.

Ценность kd-дерева в том, что его можно построить один раз и затем многократно использовать для поиска. Процедуру построения структуры данных с такой целью называют индексированием, а саму структуру – индексом.

ball-tree.

То же самое, но вместо полупространств шары. Вместо медианы на каждом шаге нужно искать геометрический центр и радиус шара.

При малых размерностях сложность построения $O(n * log(2n))$, а сложность поиска $O(log(n))$.

2.4 Реализация KNN

В данных экспериментах используется библиотека scikit-learn и класс NearestNeighbors из модуля neighbors.

Используется четыре алгоритма: kd_tree, ball_tree, brute для класса NearestNeigbors и собственная реализация my_own, которая работает полным перебором соседей и поиска ближайших k.

На основе данных алгоритмов написана собственная реализация классификатора с помощью метода KNN - класс KNNClassifier. Относительно выбранного параметров strategy, k и metric происходит отбор ближайших соседей.

В классе KNNClassifier представлена косинусная и евклидова метрика.

Также для полной работы классификации написана собственная реализация функции predict. На результат работы функции predict влияет булевый параметр KNNClassifier weights, который при True используют взвешенный KNN, где $w(i, x) = 1/(\text{distance} + 10^{-5})$ и distance - расстояние между x и i-ым соседом.

2.5 Реализация кросс-валидации

Для подсчета точности на фолдах обучающей выборки были реализованы функция kfold, которая возвращает подразбиения для теста и обучения классификатора, и функция knn_cross_val_score, которая проводит кросс-валидацию и считает метрику, указанную в параметре score (по умолчанию, точность ("accuracy"), при передаче обучающей выборке и результат работы функции kfold (если не указано, то проводит разбиение по трем фолдам).

3 Эксперименты

3.1 Задание №1

Данный эксперимент направлен на изучение времени работы различных алгоритмов поиска ближайших соседей, таких как: "kd_tree "ball_tree "brute"и "my_own где "my_own собственная реализация алгоритма. Сравнения алгоритмов было проведено относительно количества признаков (выбираемое случайным образом), по которым считается метрика, в данном случае она евклидова.

Для каждого алгоритма и подразбиения признаков (выбираются один раз для всех алгоритмов) было подсчитано время. Полученный результат был разделен на 10000 объектов, чтобы получить время работы на одном объекте. Результаты представлены на **Рис.1**.

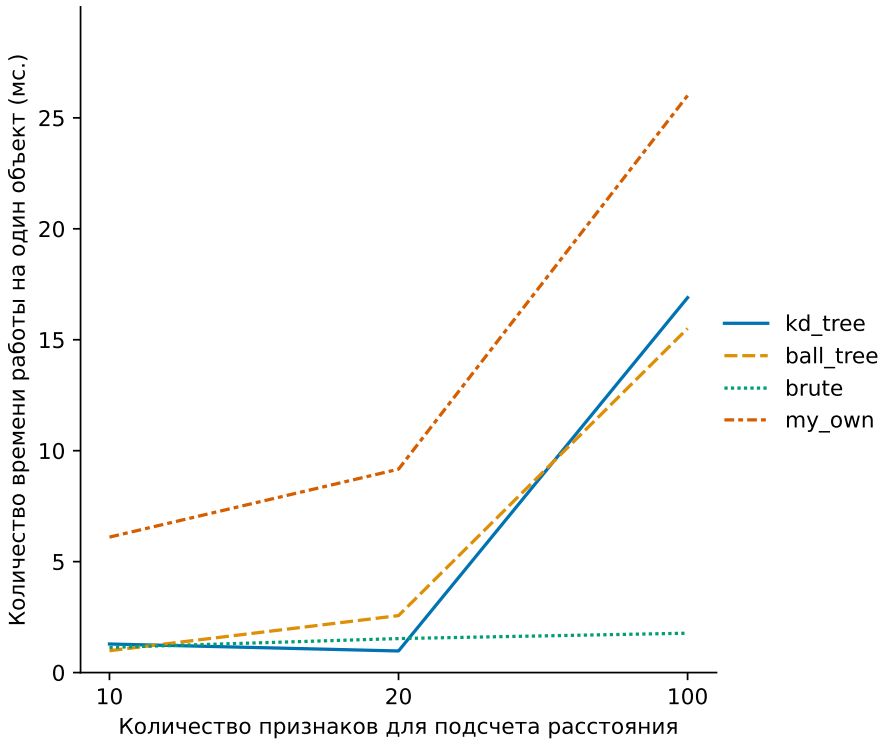

Рис. 1: Сравнение различных алгоритмов KNN на основе времени работы на одном объекте

На графике можно заметить лидирующий алгоритм по времени на подразбиении 100 - это алгоритм "brute". Остальные реализации заметно уступают ему по времени, особенно собственная реализация алгоритма, имеющая заметное отставание от других реализаций.

Алгоритм ball_tree и kd_tree имеют схожие показатели времени, что понятно из их реализации поиска ближайших.

Итогом данного эксперимента можно считать определение быстрого по времени работы алгоритма 'brute' для использования в последующих экспериментах.

3.2 Задание №2

Цель данного эксперимента - сравнить косинусную и евклидову метрику по времени работы и точности ("accuracy") классификации, а также найти лучшее число k, чтобы алгоритм классификации имел высокую точность ('accuracy') и выбрать метрику, дающую качественную классификацию.

Дальнейшие эксперименты были проведены на основе алгоритма поиска ближайших соседей 'brute'.

Сравнение косинусной и евклидовой метрики было проведено относительно времени для $k \in \{1, 2, 3\}$. Результаты приведены на Рис. 2.

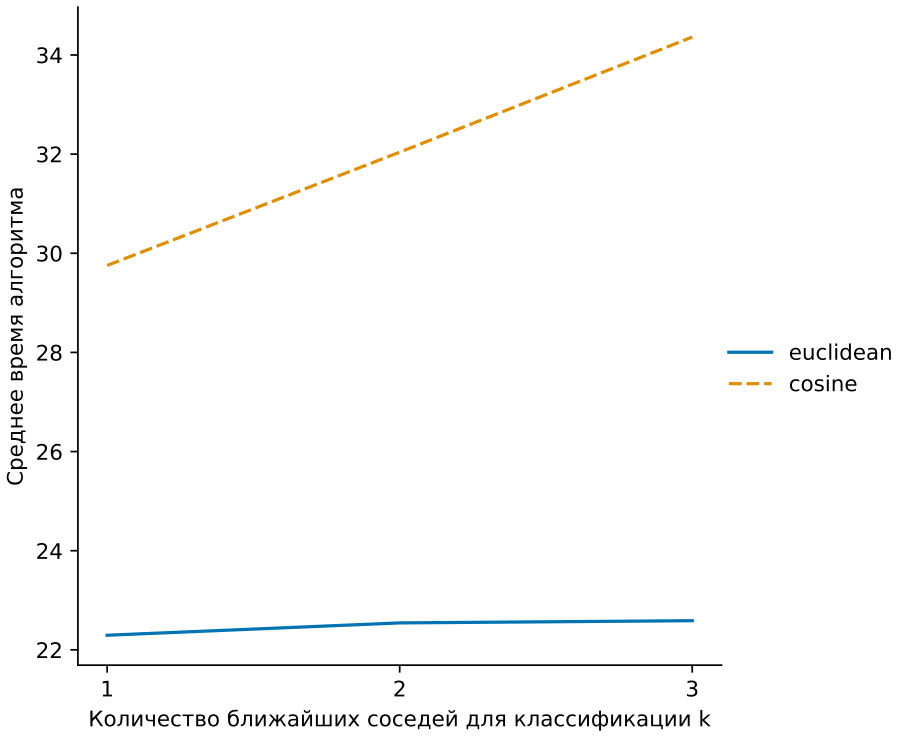

Рис. 2: Сравнение евклидовой и косинусной метрики на основе времени работы алгоритма классификации

Можно заметить, что евклидова метрика работает немного быстрее того же алгоритма на основе косинусной метрики.

Сравнение косинусной и евклидовой метрики на основе метрики точности классификации ('accuracy') было проведено при $k \in \{1, 2, ..., 10\}$. Результаты сравнения представлены на Рис. 3.

По Рис. 3 можно сделать выводы, что алгоритм, основанный на косинусной метрике имеет выше точность относительно евклидовой метрики. Если сравнить прошлый результат сравнения по времени и данный результат, то можно сделать вывод, что **косинусная метрика имеет выше качество, чем евклидовая метрика, а прирост точности имеет больший вклад, чем дополнительное время затраченное на классификацию**.

Также на Рис. 3 можно заметить, что при количестве ближайших соседей k равных 2 точность классификации понижается, затем скачком повышается, в дальнейшем можно заметить, что при нечетном количестве соседей точность повышается, относительно прошлого результата, а при четном количестве соседей - точность уменьшается. Можно сделать предположение, что при нечетном количестве соседей решающий голос остается за большинством (если поблизости 2 класса), а при четном - решающий, если голоса имеют ровно количество (2 класса), остается за тем классом, чье число меньше.

Итогом данного эксперимента можно считать определение косинусной метрики для дальнейшего использования в экспериментах, как лучшего по качеству (не пренебрегая временем).

3.3 Задание №3

Данный эксперимент направлен на сравнение взвешенного метода k ближайших соседей, где голос объекта равен $1/(distance + 10^{-5})$, где $distance$ расстояние между основным объектом и объектом-кандидатом, с методом без весов при тех же параметрах(была выбрана косинусная метрика и алгоритм 'brute', количество фолдов - 3) на основе метрики точности ("accuracy").

Сравнение было выполнено при $k \in \{1, 2, ..5\}$. Результаты представлены на **Рис. 4**.

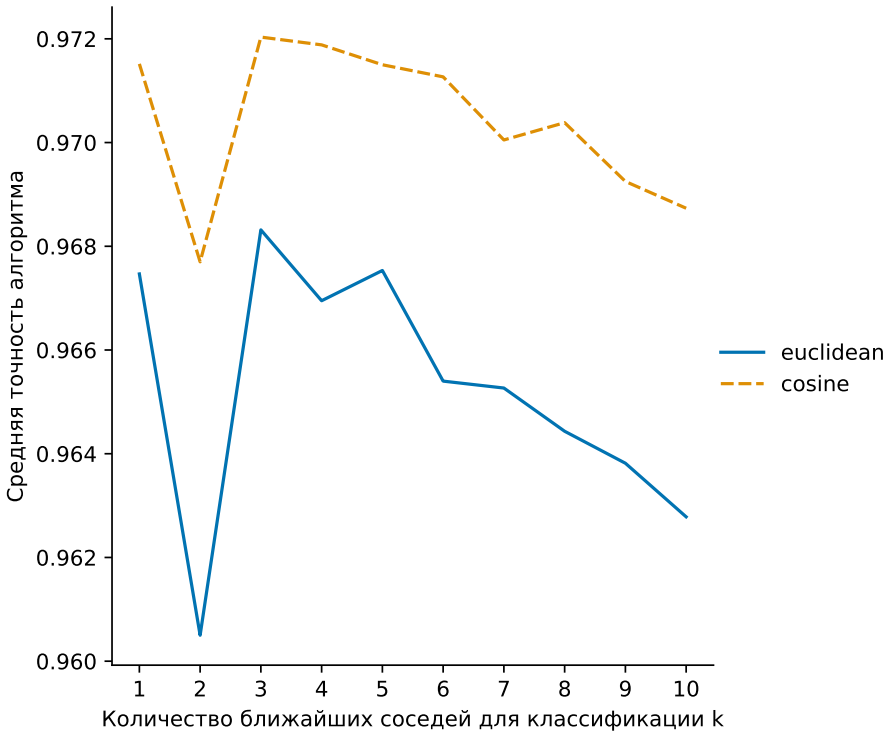

Рис. 3: Сравнение евклидовой и косинусной метрики на основе метрики точности (’accuracy’) для классификации

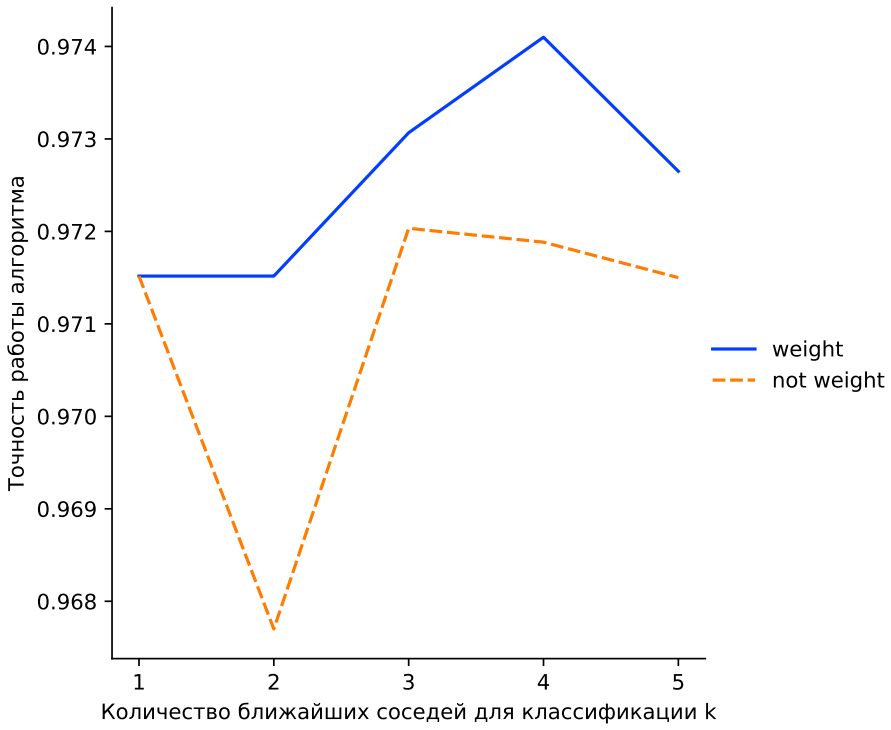

Рис. 4: Сравнение взвешенного метода k ближайших соседей и метода без весов.

На графике можно заметить лидирующее положение взвешенного метода k ближайших соседей

по метрике "accuracy". Данный результат дает нам возможность определить параметры KNNClassifier для того, чтобы обеспечить высокую точность ("accuracy") и быструю скорость.

3.4 Задание №4

Данный эксперимент направлен на подсчет метрики точности ('accuracy') на обучающей и тестовой выборке с параметрами для KNNClassifier, которые мы определили в прошлых трех экспериментах, и сравнения её с метрикой точности на кросс-валидации. Также мы изучим матрицу ошибок и рассмотрим пару объектов, на который модель ошибается.

В итоге была получена **точность равная 0.9752**. Сравнивая данный результат с кросс-валидацией при тех же параметрах (лучший результат на Рис.4) 0.9741, то можно увидеть заметный прирост в точности. Лучшие алгоритмы, для которых подсчитана метрика точности ("accuracy"), имеют точность 0.9987-0.9984, к чему пока не может приблизится наша модель.

Дальше, чтобы понять над чем нужно поработать, чтобы увеличить точность классификации модели, нужно оценить матрицы ошибок (confusion matrix) на Рис. 5.

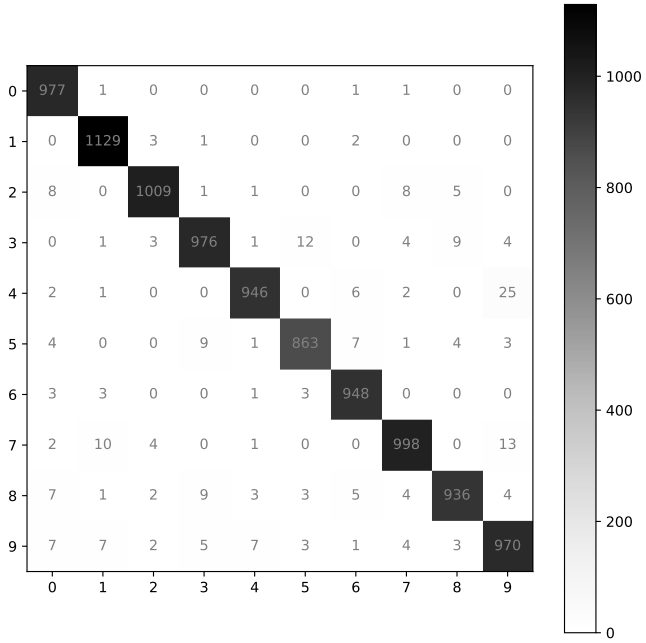

Рис. 5: Матрица ошибок модели классификации.

Анализируя матрицу ошибок, можно понять, что случаются коллизии, когда, к примеру цифра 4 становится 9, или 7 становится 9. Чтобы изучить, почему получаются данные ошибки, визуализируем случайно 22 объекта, с которыми возникают проблемы (Рис. 6).

Используя данные визуализации, мы можем определить пробелы в обработке обучающей выборки для последующей классификации. К примеру, ошибочными примерами являются:

1. Очень жирные и тонкие цифры.
2. Цифры, повернутые относительно базового положения.
3. Шумовые объекты.

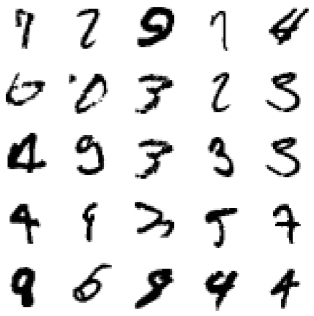

Рис. 6: Картинки цифр, на которых возникают ошибки в классификации.

4. Выбросы, которые нельзя адекватно классифицировать.

Дальнейшие эксперименты будут направлены на обработку данных объектов, чтобы повысить точность модели, на основе аугментации обучающей (задание №5) или тестовой (задание №6) выборки.

3.5 Задание №5

В данном эксперименте, цель которого улучшить точность классификации на основе аугментации, мы рассмотрим 7 аугментаций, реализованных в библиотеках cv2, skimage.

В этом эксперименте мы не будем рассматривать комбинации аугментаций, ограничимся только отдельными аугментациями. Для каждой аугментации будем по кросс-валидации на обучающей выборке оценивать точность, где train фолды мы будем увеличивать в два раза (обычный фолд + аугментированный), а test фолды оставлять неизменными.

3.5.1 Аугментация - поворот

Мы рассмотрели, что у нас есть проблемы с цифрами, которые повернуты. Оценим градус поворота из списка [-15, -10, -5, 5, 10, 15], при котором у нас достигается лучшая точность. Данные результаты приведены на Рис. 7.

В итоге, гиперпараметром для данной аугментации будет являться величина поворота равная 10.

3.5.2 Аугментация - смещение

Данный эксперимент направлен на подбор гиперпараметра для аугментации смещения. Подбор происходит перебором по списку сдвига в пикселях по горизонтали и сдвигу в пикселях по вертикали (от 0 до 3 пикселя, в разных направлениях).

Результат данного эксперимента приведен на Рис. 8.

По итогу, гиперпараметром для данной аугментации будет являться величина смещения (-1, 0), где -1 смещение по пикселям по горизонтали, а 0 - смещение по вертикале.

3.5.3 Аугментация - фильтр Гаусса

Проблема жирных цифр не только в том, что они имеют большой размер, но и то, что насыщенность черного цвета сильна, а значит вклад в расстояние будет еще больше. Фильтр Гаусса борется с данной проблемой, осветляя цифру (Рис. 9).

Были рассмотрены различные ядра, но остановились на фильтре Гаусса с ядром 3.

Дальше на кросс-валидации была проанализирован размер дисперсии фильтра Гаусса с ядром 3 из набора [0.5, 1.0, 1.5].

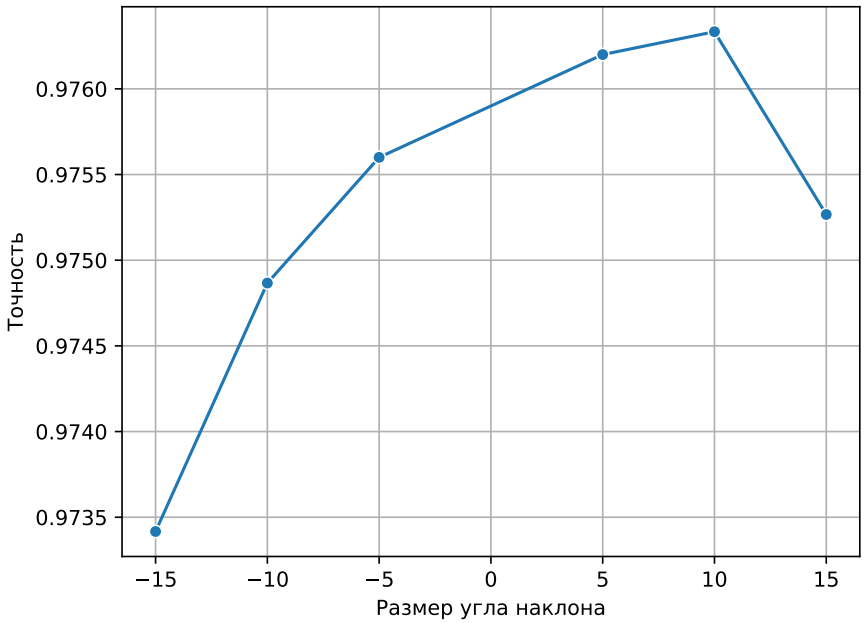

Рис. 7: График зависимости точности на кросс-валидации от величины поворота для аугментации.

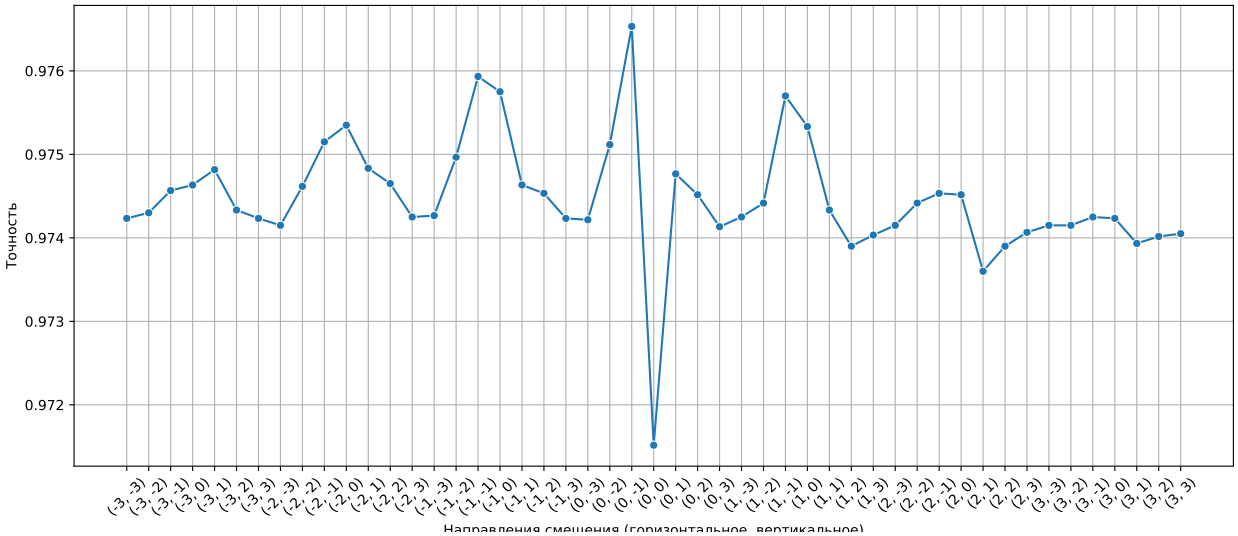

Рис. 8: График зависимости точности на кросс-валидации от величины смещения для аугментации.

В итоге, гиперпараметром для данной аугментации будет являться величина дисперсии равная 1.0 и ядро равное 3.

3.5.4 Аугментация - эрозия с ядром 2

Одним из типов ошибочных объектов являлись жирные цифры. Аугментация эрозия позволяет хорошо справляться с данной ошибкой. В данном эксперименте мы будем использовать эрозию с ядром 2 (Рис. 11).

Можем заметить, что она уменьшает размер объекта, а следовательно избавляется от жирных

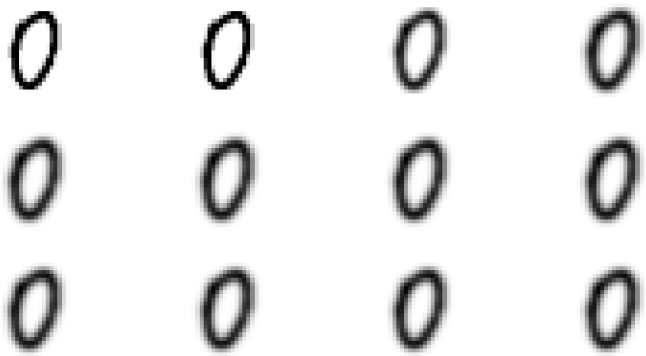

Рис. 9: Пример работы фильтра Гаусса с различными ядрами

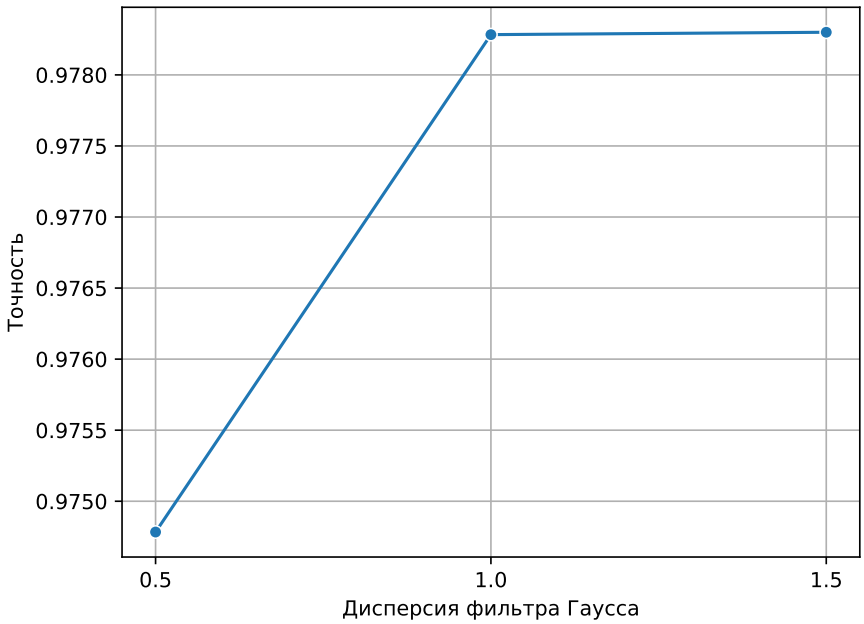

Рис. 10: График зависимости точности кросс-валидации от дисперсии фильтра Гаусса с ядром 3

цифр.

Далее, найдем количество итераций эрозий, которое улучшает точность модели. Результаты приведены на Рис. 12.

Данный результат показывает, что лучшая эрозия с одной итерацией, но точность, относительно безлайна без аугментации показала лишь небольшой прирост, оставив вопрос в использовании данной аугментации на потом.

3.5.5 Аугментация - дилатация с ядром 2

Если прошлая аугментация боролась с жирными цифрами, то дилатация - это обратная аугментация для эрозии. Она борется с тонкими цифрами, увеличивая их жирноту (Рис. 13). В данном эксперименте используется дилатация с ядром 2.

Рис. 11: Пример работы эрозии при итерациях от 0 до 3.

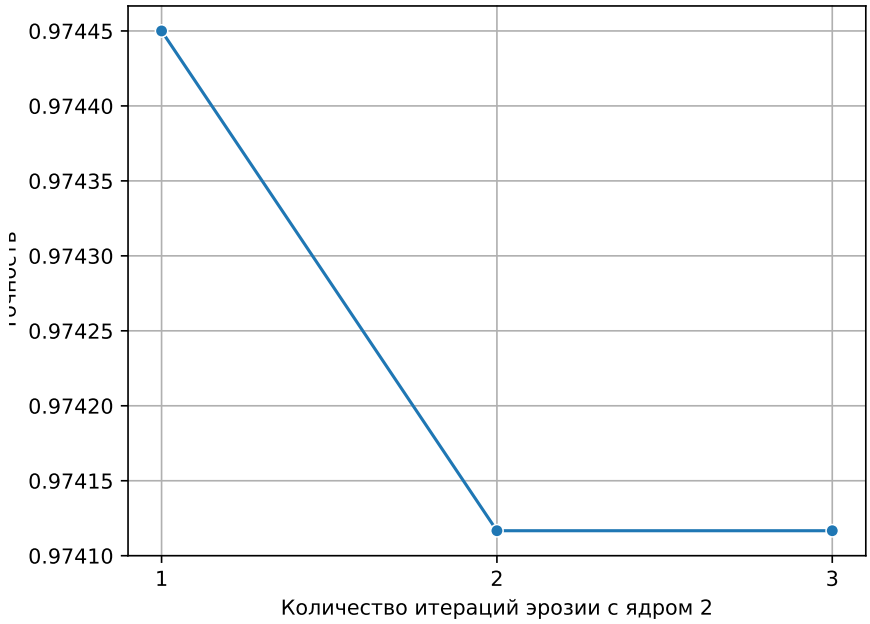

Рис. 12: График зависимость точности на кросс-валидации от количества итераций эрозии.

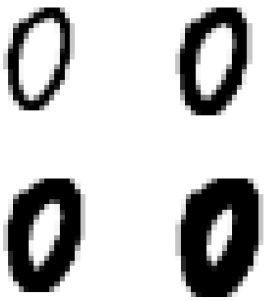

Рис. 13: Цифра в зависимости от итераций дилатации с ядром 2 (от 0 до 3).

Далее, найдем количество итераций дилатации, которое улучшает точность модели. Результаты приведены на Рис. 14.

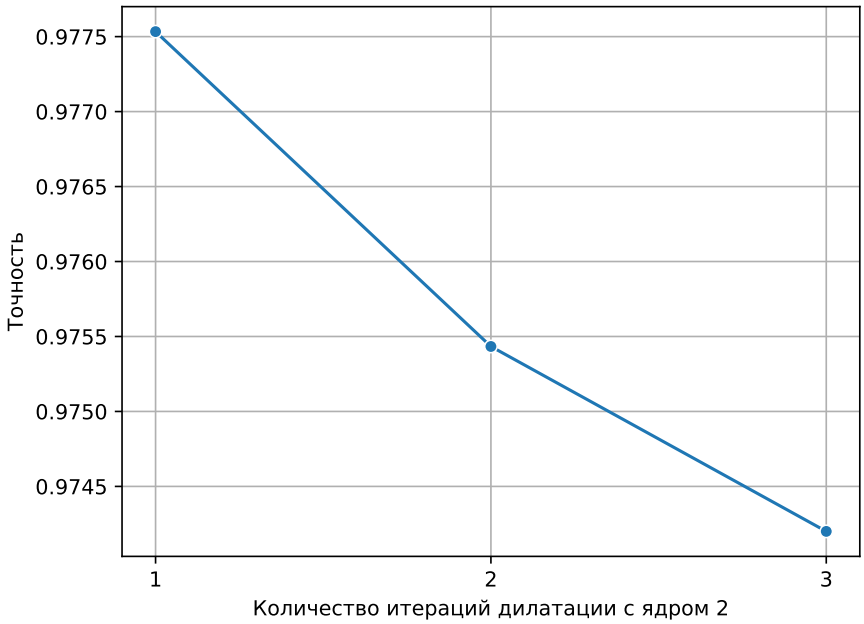

Рис. 14: График зависимости точности на кросс-валидации от итерации дилатации с ядром 2.

По итогу, гиперпараметр для дилатации равен 1 итерации. На графике заметно, что дилатация хорошо сказывается на точности.

3.5.6 Аугментация - открытие с ядром 2

Открытие с ядром используют для удаления мелких объектов и шумов. Она сочетает в себе дилатацию и эрозию. Она будет бороться с шумами объектов-выбросов, пример работа открытия с ядром 2 на Рис. 15.

Рис. 15: Цифра в зависимости от итераций открытия с ядром 2 (от 0 до 3).

Далее, найдем количество итераций открытия, которое улучшает точность модели. Результаты приведены на Рис. 16.

По итогу, оптимальный гиперпараметр для открытия с ядром 2 - это 1 итерация.

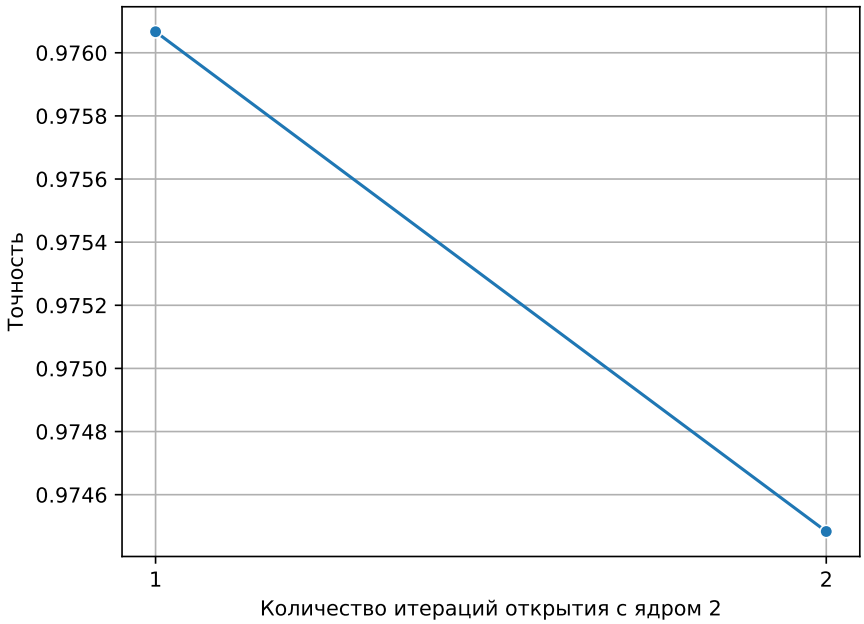

Рис. 16: График зависимости точности на кросс-валидации от итерации открытия с ядром 2.

3.5.7 Аугментация - закрытие с ядром 2

Аугментация закрытие с ядром сглаживает границы объектов и заполняет мелкие отверстия в объектах. Она также, как и открытие, сочетает в себе дилатацию и эрозию. Пример работы закрытия с ядром 2 на Рис. 17.

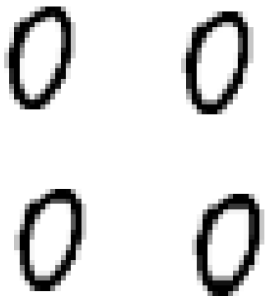

Рис. 17: Цифра в зависимости от итераций закрытия с ядром 2 (от 0 до 3).

Далее, найдем количество итераций закрытия, которое улучшает точность модели. Результаты приведены на Рис. 18.

По итогу, оптимальный гиперпараметр для закрытия с ядром 2 - это 1 итерация.

3.5.8 Результат

Данный эксперимент направлен на применение полученных аугментаций к обучающей выборке с теми гиперпараметрами, которые мы подобрали в прошлом эксперименте, и обучения модели на модифицированной выборке для подсчета точности на тестовой выборке.

Рис. 18: График зависимости точности на кросс-валидации от итерации закрытия с ядром 2.

В итоге, точность на модифицированной выборке равна **0.9836**. Следовательно, модель повысила качество классификации цифр после применения аугментаций.

Далее, рассмотрим матрицу ошибок и проанализируем, что изменилось (Рис. 19).

Можем заметить, что многие ошибки ушли, но не все. Рассмотрим 25 случайно выбранных цифр, на которых модель ошибается и визуализируем их на Рис. 20.

В ошибочных цифрах можем увидеть несколько видов ошибок:

1. Кривое написание цифры
2. Очень жирное написание цифры
3. Сильно шумные объекты
4. Выбросы

3.6 Задание №6

Данный эксперимент направлен на изучения другого подхода работы с аугментациями. Мы будем их применять к тестовой выборке и предсказывать по полученным объектам класс, затем проводить голосование между классами полученными таким способом для каждого объекта (учитывая, что полученные аугментации также являются тем же объектом).

мы будем использовать те же гиперпараметры, полученные в прошлом эксперименте, но в обратную сторону. То есть если сдвиг был в одну сторону, то мы делаем его в противоположную. Гиперпараметры дилатации обучающей выборки превращаются в параметры эрозии теста, и аналогично для эрозии. Поворот цифр берется в обратную сторону.

В итоге, после обучения модели на обычном датасете и предсказании таким способом точность модели стала равна **0.9741**.

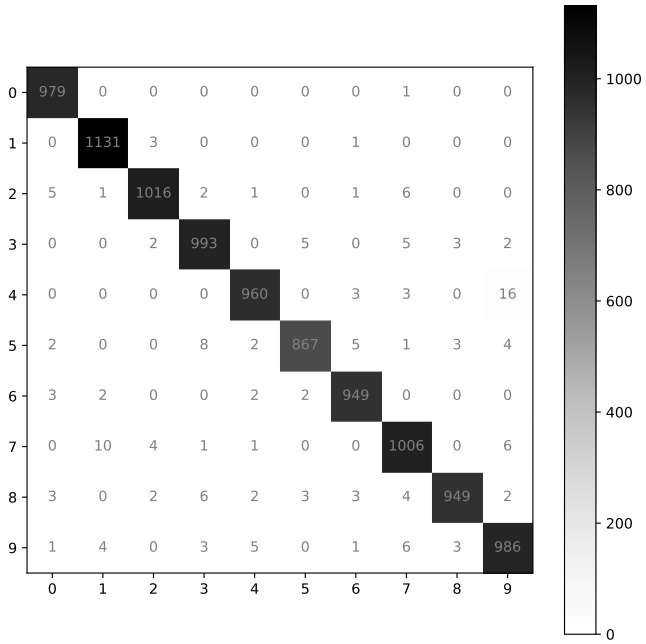

Рис. 19: Матрица ошибок для модели, обученной на аугментации обучающей выборки.

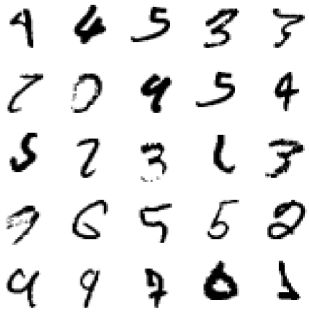

Рис. 20: Часть цифр, на которых ошибается модель, обученная на модифицированной выборке.

4 Заключение

Результатом данной работы являются навыки работы с аугментациями, полученные при проведении экспериментов, и навык подбора гиперпараметров модели. Также были изучены подходы к увеличению точности модели KNNClassifier на датасете MNIST.

Первые четыре эксперимента были направлены на определение лучшей по точности модели для классификации тестовой выборки, а последние два эксперимента - на получение результатов точности выше модели, обученной на оригинальном датасете.

Есть несколько проблем метрического метода в данной задаче - одна из них это то, что мы считаем метрику между разными объектами, но на одном и том же пикселе, но в реальных примерах это далеко не так. Мы пробовали исправить данную коллизию сдвигами, какие-то примеры были исправ-

лены, но это не решает проблему полностью. Особенно проблематичными являются цифры с разным написанием, потому что в векторах - это очень разные объекты. Иногда в тестовой выборке возникают "выбросы к примеру, когда появляется "закорючка которая сильно влияет на подсчет расстояния.

Для решения данных проблем нужно искать новые методы аугментации, либо новые подходы к построению моделей.

5 Библиография

1. Хэндбук Яндекса, параграф 2.2 Метрические методы . https://education.yandex.ru/handbook/ml/article/metricheskiye-metody
2. Лекция ММО К.В.Воронцова по метрическим методам.
3. Семинарский конспект ММРО №2 по метрическим методам.