Отчет о практическом задании
«Метрические алгоритмы классификации»

Практикум 317 группы, ММП ВМК МГУ

Борисов Иван Максимович

16.10.2023

Содержание

Аннотация

Данное практическое задание посвящено изучению основных метрических методов классификации, их улучшению путем подбора оптимальных гиперпараметров и расширения множества обучающей выборки (аугментацией) в задаче распознавания рукописных цифр датасета MNIST.

1 Введение

Классической задачей машинного обучения является задача *классификации*. Пусть задано произвольное множество объектов и для каждого объекта указано, к какому классу он принадлежит. Тогда под задачей классификации понимается отнесение новых объектов к более «подходящим» классам на основе сходств с заданным множеством. Задача сводится к определению понятия более «подходящего» или «ближайшего» класса. Для определения этой близости в машинном обучении используются различные методы, такие как функциональные, вероятностные и метрические.

В данной работе исследуется метрический метод называемый методом «ближайших соседей». Его идея заключается в нахождении попарных расстояний между заданным объектом и объектами из множества с известной классификацией. Тогда для определения класса выбирается фиксированное число объектов с наименьшими расстояниями и по ним определяется, к какому классу принадлежит наш объект.

2 Пояснения к задаче

2.1 Математическая постановка задачи

Пусть задано множество объектов $X = \{x\}_{i=1}^{n}$ и ответов на них $Y = \{y\}_{i=1}^{n}$,

$$x_i \in \mathbb{R}^d,\ y_i \in \{0, 1, ..., m\}, m \in \mathbb{N}$$

Будем называть y_i — **меткой класса** (меткой) объекта x_i. Пусть также в $\mathbb{R}^n$ определено расстояние $\rho : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \cup \{0\}$

Пусть теперь необходимо определить класс объекта $q \in \mathbb{R}^n$. Для решения этой задачи фиксируем «число ближайших соседей» $k \in \mathbb{N}$.
Далее для любого $x_i \in X$ находим $\rho_i = \rho(x_i, q)$. Выбираем из $\{\rho_i\}_{i=1}^{n}$ k минимальных по модулю $\rho_{i_1}, ..., \rho_{i_k}$. Им соответствуют объекты $x_{i_1}, ..., x_{i_k}$, по ним определяем метки $y_{i_1}, ..., y_{i_k}$.
Теперь с помощью «голосования» (см. 3.3) определяем метку нашего класса.

2.2 Алгоритмы нахождения k ближайших соседей

Пусть необходимо найти k ближайших соседей для $Q \in \mathbb{R}^{m \times d}$, где каждая строка Q — объект, для которого необходимо найти ближайших соседей.

2.2.1 Brute method

Суть «грубого» метода заключается в построении матрицы попарных расстояний и нахождении минимальных расстояний по строкам построенной матрицы.
Минусы наивного алгоритма заключаются в необходимости хранить матрицу размера $m \times n$ и сложности поиска $O(n \cdot m \cdot d)$.

2.2.2 Kd-tree

Для ускорения алгоритма перебора можно строить специальное дерево, которое и было названо kd-tree.
Основная идея: фиксируем признак и находим объект со «средним» значением по этому признаку, затем делим множество оставшихся объектов по принципу большие в правое поддерево, меньшие - в левое. Далее фиксируем другой признак и повторяем алгоритм.
Теперь для нахождения ближайших соседей необходимо совершать обход дерева в глубину, что существенно ускоряет время работы: сложность алгоритма $O(\log n \cdot m \cdot d)$.
Но при достаточно больших d сложность алгоритма может деградировать до $O(n \cdot m \cdot d)$.
Что брать за «среднее» значение, как перебирать признаки и какой использовать критерий останова может зависеть от конкретной реализации. Поэтому для более подробной информации см. sklearn.neighbors.

2.2.3 Ball-tree

Для решения проблемы деградации сложности kd-tree при больших размерах признакового пространства была придумана модификация kd-tree, названная ball-tree и хорошо зарекомендовавшая себя при больших значениях выборки.
Для построения дерева вместо фиксации декартовой координаты, находим **центроиду** (т.е. геометрический центр множества объектов). Рассчитываем максимальный радиус от центроиды до остальных точек и запоминаем его. Повторяем процедуру для точек, попавших в различные полусферы. Теперь при обходе дерева достаточно сравнивать расстояние от объекта, для которого ищется расстояние, до центроиды с максимальным радиусом в правой и левой вершинах. Тем самым осуществляется обход дерева в глубину. Получаем сложность $O(\log n \cdot m \cdot d)$ уже устойчивую к размерности признакового пространства.
Аналогично для более подробной информации о конкретной реализации стоит обратиться к sklearn.neighbors.

2.3 Расстояния

В 2.1 предполагалась фиксация $\rho : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \cup \{0\}$. Рассмотрим метрики используемые в рамках исследования.

2.3.1 Евклидово расстояние

$$\forall x, y \in \mathbb{R}^d \Rightarrow \rho(x, y) = (\sum_{i=1}^{d}(x_i - y_i)^2)^{1/2}$$

2.3.2 Косинусное расстояние

$$\forall x, y \in \mathbb{R}^d \Rightarrow \rho(x, y) = 1 - \frac{< x, y >}{||x||_2||y||_2}$$

$$< x, y >= \sum_{i=1}^{d} x_i y_i \ , \ ||x||_2 = (\sum_{i=1}^{d} {x_i}^2)^{1/2}$$

2.4 Голосование

В этом разделе речь пойдет о способе определении метки объекта q из 2.1 по определенным меткам $y_1, ..., y_k$.

2.4.1 Наивное голосование

Объявим за метку q моду $\{y_i\}_{j=1}^{k}$. Если их несколько, то выберем произвольную, так как модель «не уверена» в ответе.

2.4.2 Взвешенное голосование

Возможна ситуация, когда в множество ближайших соседей для q попало большое число объектов с «большими» расстояниями и несколько с «маленькими». Но разница между «большими» и «маленькими» расстоянием настолько велика, что хотелось бы определять класс для q по тем немногочисленным объектам с «маленькими» расстояниям.
В связи с этим желанием для каждой метки из $\{y_i\}_{i=1}^{k}$ в соответствие ставится «голос» $v_i = \frac{1}{\rho_i + \varepsilon}$,

$$\varepsilon = 10^{-5}$$

Для получения метки объекта q необходимо просуммировать все голоса при одинаковых y_i и взять максимум по полученным суммам, тогда меткой для q будет метка соответствующая всем слагаемым из суммы.

3 Эксперименты

Теперь, когда задача сформулирована и объявлены все доступные инструменты, исследуем метод ближайших соседей на датасете MNIST [1]. Выборка состоит из 70 тыс. картинок размера 28×28. Разделим ее на обучающую и тренировочную в отношении 6 к 1. Признаками в данном случае выступают отдельные пиксели, то есть $d = 784$.

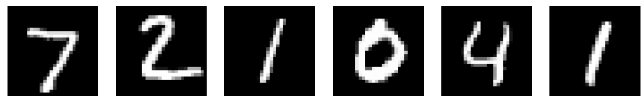


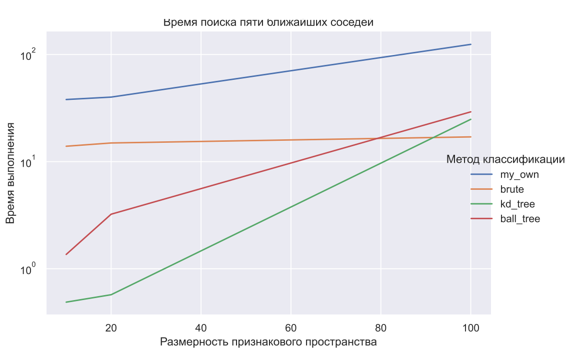
Рис. 1: MNIST

3.1 Скорость работы алгоритмов

Первым делом исследуем скорость работы алгоритмов при разных размерностях признакового пространства, то есть при случайно выбранных 10, 20 и 100 признаках.
Используем параметры «по умолчанию», то есть пока они не имеют никакого смысла и выбраны произвольным образом. Положим $k = 5$, будем использовать евклидову метрику и не взвешивать голоса.
Рассмотрим алгоритмы:

- `my own` - вариация «грубой» реализации, написанная автором
- `brute`, `kd tree`, `ball tree` - модели реализованные в `sklearn.neighbors.NearestNeighbors` с авторским методом `predict`

,



Вывод: самое долгое время работы показала собственная реализация «грубого» поиска. Аналогичный алгоритм из sklearn работает быстрее, что свидетельствует о его неплохой оптимизации внутри, также он показал некоторую устойчивость к числу признаков. Лучшие результаты показали алгоритмы, в реализации которых используются деревья. При малых размерах признакового пространства `kd tree` работает существенно быстрее конкурентов, но с ростом d начинает замедляться и при $d = 100$ практически сравнивается с `ball tree`, что подтверждает тезис, что `ball tree` работает быстрее при больших d.

3.2 Подбор гипермараметров

Гиперпараметром называют параметр модели, которому она не обучается сама, иными словами его необходимо указывать вручную.
В нашем случае гиперпараметрами выступят число ближайших соседей k и расстояние ρ.

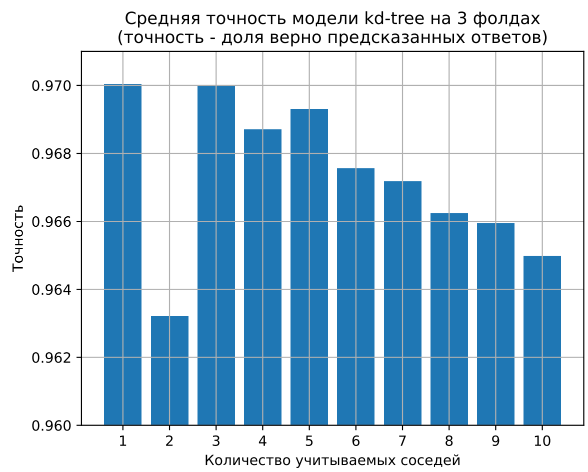
[1] MNIST (сокращение от «Modified National Institute of Standards and Technology») — объёмная база данных образцов рукописного написания цифр (источник - wiki).

Для поиска гиперматаметров будем использовать кросс-валидацию с 3 фолдами.
Для оценки качества модели введем метрику, которую назовем **точностью (accuracy)** - отношение правильно предсказанных классов к общему числу предсказаний.

3.2.1 Подбор оптимального числа соседей k

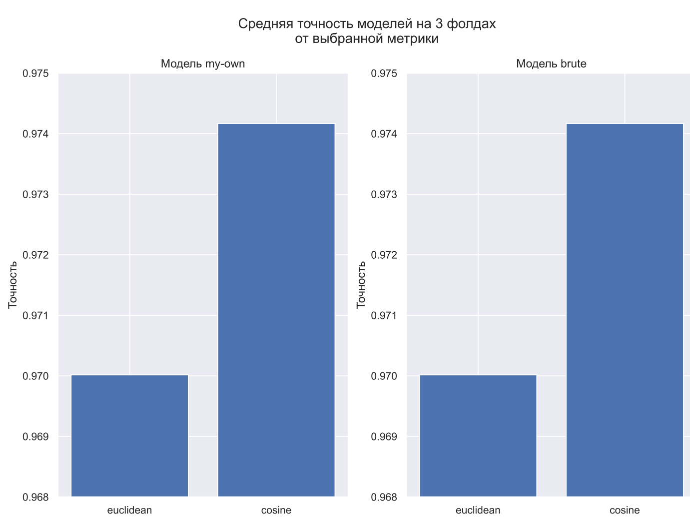
Переберем $k = 1, 2, \ldots, 10$, используя кросс-валидацию.
Основываясь на результатах из 3.1, выберем `kd tree` для оптимизации времени ожидания результатов. Все параметры оставляем так же «по умолчанию».
Для лучшей интерпретируемости усредним точность по фолдам.

Вывод: лучшие результаты показали значения $k = 1, 3$. В дальнейшей будем использовать $k = 3$. На $k = 4, 5, \ldots, 10$ модель начинает переобучаться — ее качество падает с каждым шагом. Можно заметить «провалы» при значениях $k = 2, 4$, причем при $k = 4$ ухудшение качества не такое резкое, как в случае $k = 2$. Это может быть связано с неопределенностью предсказаний классификатора, то есть в случае, когда ближайшие соседи из разных классов, то модель «подбрасывает монетку» в надежде угадать правильную метку. Вероятность такого события увеличивается при четных k и достигает максимума при $k = 2$, что и поясняет сложившуюся ситуацию.
Время выполнения алгоритма - **14мин 22с**.



3.2.2 Исследование расстояний

В методах `kd tree` и `ball tree` нет возможности использовать косинусное расстояние, поэтому исследуем на «грубых» алгоритмах.
Фиксируем $k = 3$ из 3.2.1, используем кросс-валидацию, остальные параметры - «по умолчанию».
Так же усредним точность по фолдам.



Вывод: косинусная метрика показала результаты на 4 тысячные лучше евклидовой, что на тестовой выборке дает на 40 больше верных предсказаний. Чтобы разобраться, почему так происходит, стоит понимать «смысл» используемых метрик.
Если с евклидовой еще интуитивно понятно, что это длина вектора, соединяющего две точки пространства (только не совсем понятно, что такое векторы, полученные вытягиванием картинки в строчку в 784 мерном пространстве), то косинусное расстояние показывает схожесть векторов по углу между ними и не учитывает нормы

векторов — это следует напрямую из определения косинусного расстояния. Возможно, в этом отличии и есть ключевая разница: норма картинки в нашем случае тем больше, чем больше в ней ненулевых значений, иными словами больше закрашенных пикселей, и чем пиксель темнее, тем норма больше. То есть жирно обведенная цифра будет отличаться по норме от тонко написанной той же, что не хорошо. Поэтому в данной задаче мы не хотим учитывать нормы картинок, а для этого лучше подходит косинусное расстояние, нежели евклидово.

Среднее время выполнения алгоритмов - **21мин 2с** и **3мин 4с** (`my own` и `brute` соответственно).

3.3 Как проводить голосование

Интуитивно понятно, что добавление весов к голосам должно положительно сказаться на точности модели. Исследуем эту зависимость.
Пусть:

- $k = 3$
- Расстояние — косинусное
- Алгоритм - `brute`
- Кросс-валидация на 10 фолдах
- Усредняем точность по фолдам

Вывод:
Интуитивное предположение подтвердилось, но существует и фолд, на котором невзвешенный метод показал себя лучше, чем взвешенный, то есть вообще говоря не всегда взвешенный метод показывает себя лучше.
Среднее время выполнения алгоритмов - **4мин 22с**

3.4 Собираем модель

Подведем итоги прошлых разделов, проанализировав полученную в итоге модель.

- $k = 3$ (см. 3.2.1)
- Расстояние — косинусное (см. 3.2.2)
- Алгоритм — `brute` (см. 3.1)
- Взвешенное голосование (см. 3.3)

Для наглядности используем «матрицу ошибок». По оси OX откладываются предсказания модели, по оси OY — верные ответы, на пересечениях — количество объектов, попавших под данное описание.

Вывод:
Модель верно предсказывает 9.742 значения из 10.000, что можно считать неплохим результатом. Похожая точность получалась при подборе параметров на кросс-валидации, то есть модель не переобучилась под обучающую выборку и имеет хорошую обобщающую способность.
Для сравнения рассмотрим точности приведенные на сайте, с которого была взята выборка.
Можно видеть, что качество модели может быть улучшены на целые проценты!

Чтобы разобраться подробнее, рассмотрим примеры ошибок модели.

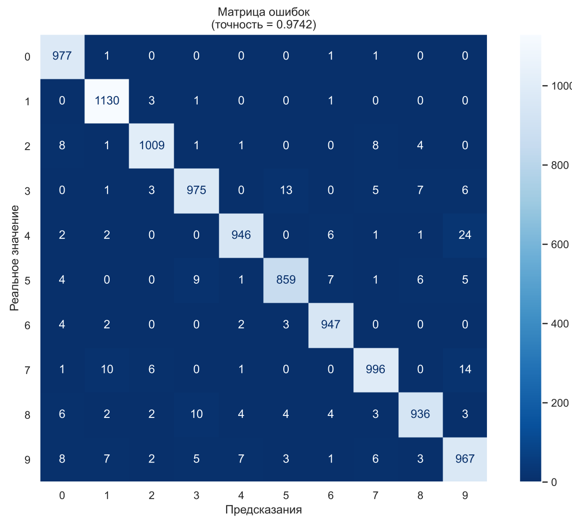


Рис. 2: Примеры, на которых модель ошибается

Ошибки происходят из-за того, что цифры написаны под неестественным углом, сдвинуты относительно центра и разной толщины. Идея: можно попробовать научить модель отличать «плохие» цифры, искусственно добавив новые «плохие» объекты в датасет.

Время предсказания 10 тыс. объектов - **52с**.

3.5 Аугментация обучающей выборки

Исследуем такие методы аугментации как поворот на фиксированный угол, сдвиг на некоторое количество пикселей, «замыливание» («блюр») и применение морфологических операций — эрозии, дилатации, открытия и закрытия.
Лучшие параметры угла поворота, сдвига и силы размытия определим на «лучшей» модели (см. 3.4) и кросс-валидации с 3 фолдами. Также для того, чтобы не было проблем с оперативной памятью будем использовать только половину обучающей выборки.

Будем перебирать параметры:

- Сдвиг (по 4 направлениям): 1, 2, 3 пикселя
- Угол (в две стороны): 5, 10, 15 градусов
- Коэффициента размытия: 0.5, 1, 1.5 для ядра размером 3×3
- Морфологические преобразования

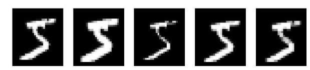
Убедимся, что преобразования работают корректно:

Сдвиг

Поворот

Замыливание

Морф.



Применяем кросс-валидацию для перебора описанных выше параметров. Усредняем точность по фолдам и умножаем на 10.000. Итого:

Таблица 1: Среднее число верных предсказаний на 10.000 объектах

	1	2	3
Сдвиг	9575	9454	9434

	5	10	15
Поворот	9728	9644	9568

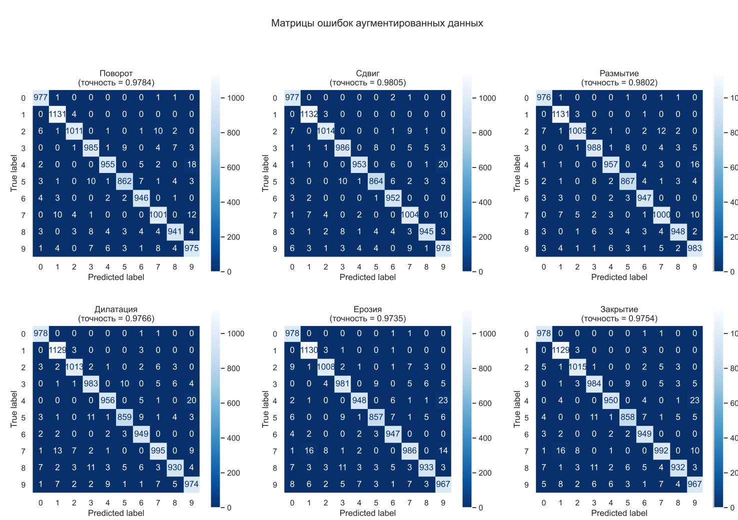
	0.5	1	1.5
Размытие	9730	9727	9732

	Дилатация	Эрозия	Закрытие	Открытие
Морф.	9643	9561	9592	9582

Фиксируем лучшие параметры:

- Сдвиг: 1 пиксель
- Угол: 5 градусов
- Коэффициента размытия: 1.5
- Морфологические преобразования: оставляем все

Для каждого оптимального параметра строим новую обучающую выборку — для сдвига выборка расширяется в 5 раз, для угла - в 3 раза и т.д. На каждой из новых выборок обучаем «лучшую» модель и составляем матрицы ошибок.
не



Как можно видеть сдвиг, поворот и размытие существенно повышают точность модели — практически на 6 тысячных в каждом из случаев.

Рассмотрим есть ли какие-то заметные изменения для конкретных цифр.

- Отличить 4 от 9 и 8 от 3 позволяет размытие.
- Отличить 9 от 0 помогают дилатация и поворот.
- Отличить 2 от 0 и 7 от 9 помогает дилатация.
- Отличить 9 от 1 помогают поворот и сдвиг.
- Все морфологические преобразования ухудшают различие между 7 и 1

Время обучения и предсказывания всех моделей — **16мин 43с**.

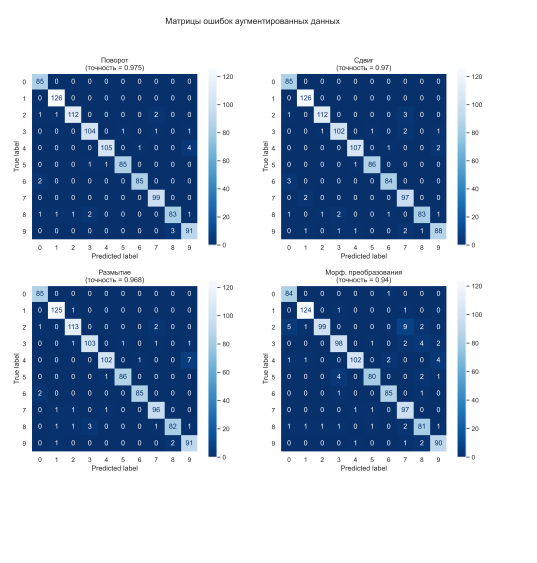
Вывод: аугментация обучающей выборки — полезный инструмент, позволяющий сильно расширить выборку и тем самым увеличить точность модели, пожертвовав временем на обучение/предсказание. Если дополнительно добавить комбинации, основанные, к примеру, на случайном выборе применяемого метода из вышеперечисленного списка с оптимальными параметрами, то, возможно, еще немного удастся поднять качество.

3.6 Аугментация тестовой выборки

Противоположная идея: будем расширять множество тестируемых объектов, далее среди искусственно созданных объектов путем голосования определим к какому классу относился начальный объект.
В ходе этого эксперимента возникли серьезные проблемы с количеством необходимой оперативной памяти, в связи с этим будем использовать тестовую выборку в размере 1 тыс. объектов.
Для того чтобы голосование имело смысл необходимо увеличить число голосующих, потому как, к примеру, с морфологическими преобразованиями получатся двое голосующих, и в случае разных ответов получается подбрасывание монетки. Для решения этой проблемы объединим все перечисленные в п. 3.5 параметры и на них будем проводить голосование.
Таким образом для поворота будет $1+2*3=7$, для сдвига $1+3*4=13$ и для морфологических преобразований $1+1*4=5$ голосующих.
Важно отметить, что в соответствие голосующим будет увеличивать тестовая выборка в то же число раз.
Снова обучим «лучшую» модель и построим матрицы ошибок.

Время обучения и предсказывания на всех выборках — **2мин 19с**.

Вывод: качество модели в среднем только ухудшается. Это можно объяснить тем, что в обучающей выборке большая часть данных «чистая» и наша модель не могла выучить необходимые «загрязнения» в объектах, поэтому она стала чаще ошибаться при применении данного метода. Но, к примеру, если соединить в себе оба вида аугментации, как тестовой так и тренировочной выборок, то модель научиться различать «грязные» данные за счет их появления в обучающей выборке, и в конечном итоге голосование по новым данным станет существенно полезнее и обоснованнее.



4 Заключение

В ходе данного исследования были установлены следующие результаты:

1. Скорости работы алгоритмов `kd tree` и `ball tree` превосходят «наивную» реализацию метода ближайших соседей.
2. С ростом числа ближайших соседей качество модели уменьшается, т.е. зачастую k не стоит брать слишком большим.
3. Метрику стоит подбирать под определенные свойства и требования, в нашем случае таким свойством было отсутствие зависимости близости объектов от их нормы.
4. Взвешенное голосование дает лучшую точность.
5. При работе с изображениями сильным способом улучшения модели является аугментация, позволяющая существенно улучшить качество модели.

Итог: метрический метод классификации поиска ближайших соседей показывает достаточно высокую точность при решении задачи классификации рукописных цифр на основе датасета MNIST.