# Задание 1. Метрические алгоритмы классификации

**Отчет**
**о выполненном задании**

студента 317 группы факультета ВМК МГУ
Воробьева Сергея Юрьевича

Октябрь, 2021

# Содержание



# 1 Введение

Задание направлено на ознакомление с метрическими методами классификации и работу с изображениями. Нужно реализовать метрический алгоритм `k` ближайших соседей, сравнить между собой различные метрики и стратегии, а также подобрать гиперпараметры с помощью кросс-валидации. В последних экспериментах предлагается поработать с преобразованиями изображений для аугментации данных.
В качестве данных выступают `70000` размеченных изображений рукописных цифр `(0-9)`. Для обучения предлагается взять `60000` изображений. Алгоритм должен предсказывать цифры на изображении. В отчете мы будем пользоваться метрикой качества `accuracy` (далее иногда будем называть ее «точность»). В данном случае это информативно, так как выборка сбалансированна по классам, и удобно, так как легко интерпретировать эту метрику. Нужно провести ряд экспериментов, направленных на подбор гиперпараметров и улучшение качества алгоритма. Результаты этих экспериментов описаны в этом отчете.

# 2 Эксперименты

## 2.1 Эксперимент 1

Предлагается сравнить 4 различные стратегии нахождения ближайших соседей: `brute`, `kd_tree`, `ball_tree`, `my_own`. Нужно исследовать, какая из них будет быстрее работать на ограниченном наборе признаков. В качестве подмножества признаков возьмем случайную подвыборку из 10, 20 и 100 признаков.

| | my_own | brute | kd_tree | ball_tree |
|---|---|---|---|---|
| 10 признаков | 61.54 | 8.57 | 3.96 | 6.56 |
| 20 признаков | 69.51 | 8.03 | 7.10 | 23.38 |
| 100 признаков | 209.11 | 10.28 | 129.33 | 119.29 |

Таблица 1: Время работы, сек

### Выводы

Можно заметить, что для любого подмножества признаков стратегия `my_own` работает ощутимо медленнее остальных. При этом лучшее качество для 100 признаков показывает стратегия `brute`. Можем заключить, что на большом числе признаков именно она работает быстрее остальных. Дальше будем использовать именно `brute`, потому что в нашей задаче у объектов `784` признака.

## 2.2 Эксперимент 2

Требуется измерить точность и время работы алгоритма по кросс-валидации с 3 фолдами в зависимости от количества соседей `(1 - 10)` и в зависимости от метрики `(cosine, euclidean)`. Для сравнения точности дважды запустим кросс-валидацию для евклидовой и косинусной метрики.
Результаты на Рис. 1.
Найдем зависимость времени от метрики и от `k`. Так как кросс-валидация считает матрицу расстояний только для максимального `k` – нет смысла использовать ее для подсчета времени. Воспользуемся интерфейсом `fit-predict` и найдем для каждой метрики и для каждого `k` время выполнения. Результаты на Рис. 2.

### Выводы

На графике видно, что косинусная метрика работает лучше евклидовой для любого `k`. Так же для нашей задачи оптимальное количество соседей - `3`. На Рис. 2 видно, что по времени алгоритмы отличаются мало для любого `k`. Следовательно, оптимальный выбор параметров для нас в этой задаче – `metric: cosine, k = 3`.

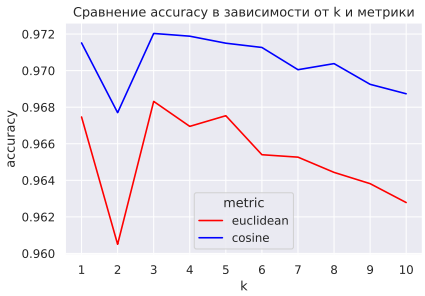


Рис. 1: Точность

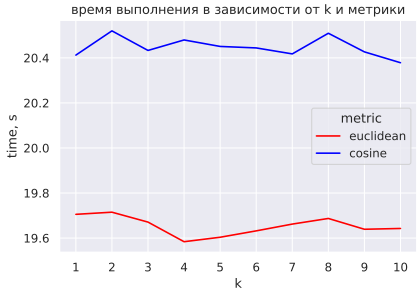


Рис. 2: Время

## 2.3 Эксперимент 3

Требуется сравнить точность алгоритма с весами и без. Запустим две кросс-валидации для алгоритма с весами и алгоритма без весов. Усредним качество по 3 фолдам и сравним полученные значения. Для этого эксперимента используем метрику `cosine` и `k = 3`.

| | С весами | Без весов |
|---|---|---|
| accuracy | 0.9732 | 0.9694 |

Таблица 2: Сравнение алгоритмов с весами и без весов

### Выводы

Можем заметить, что использование весов немного увеличивает точность прогноза.

## 2.4 Эксперимент 4

В данном эксперименте нужно обучить лучший алгоритм и построить по нему прогноз. По прогнозу нужно найти матрицу ошибок и разобрать несколько неверно-классифицированных примеров. Оценить его точность (accuracy) по fit-predict и по кросс-валидации с 3-мя фолдами и сравнить ее с лучшими алгоритмами, информация по которым есть в Интернете.

| | fit-predict | cross_val | Лучший |
|---|---|---|---|
| accuracy | 0.9730 | 0.9730 | 0.9981 |

Таблица 3: Сравнение точности

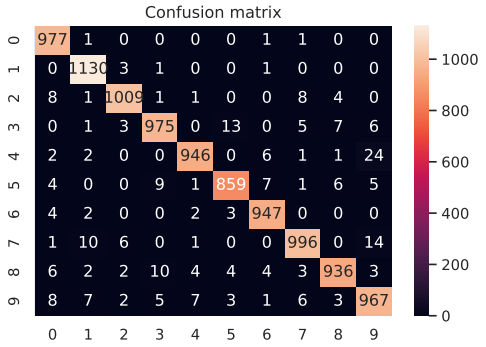


Рис. 3: Матрица ошибок

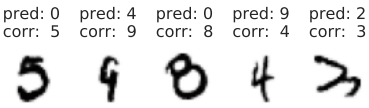


Рис. 4: Некоторые примеры неверной классификации

**Выводы**

Можем заметить, что чаще всего алгоритм ошибается на похожих по написанию цифрах: `7/1`, `3/8`, `3/5`, `9/4` и т.д. На Рис. 4 есть примеры таких ошибок. Также алгоритм ожидаемо плохо работает на плохих по качеству изображениях как на последнем примере на Рис. 4. В данном случае лучшее качество, которое получается у алгоритма - `0.9730`. Лучшее качество на этой задаче `0.9981` (Kaggle).

## 2.5 Эксперимент 5

В этом эксперименте нужно подобрать параметры преобразований изображений (смещение, повороты, размытие) для аугментации данных. Будем множить выборку в два раза для каждого параметра, проверяя результаты по кросс-валидации, а затем скомбинируем самые удачные результаты. Также посмотрим, какие ошибки удается исправить с помощью такой аугментации, построив относительную матрицу ошибок – разницу между матрицей ошибок до аугментации и после.

Таблица 4:

| **Повороты** | 5° | 10° | 15° |
|---|---|---|---|
| accuracy | 0.9980 | 0.9826 | 0.9719 |

Таблица 5:

| **Смещения** | 1px | 2px | 3px |
|---|---|---|---|
| accuracy | 0.9696 | 0.9583 | - |

Таблица 6:

| **Размытие** | $\sigma = 0.5$ | $\sigma = 1.0$ | $\sigma = 1.5$ |
|---|---|---|---|
| accuracy | 1.0 | 0.9925 | 0.9790 |

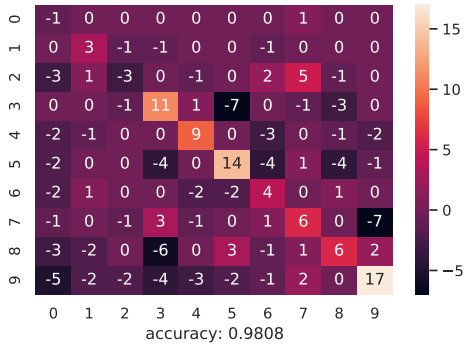


Рис. 5: Ошибки после аугментации данных

Таблица 7:

| | До аугментации | После аугментации |
|---|---|---|
| accuracy | 0.9730 | 0.9808 |

**Выводы**

Заметим некоторые особенности аугментации данных:

- Размытие с $\sigma = 0.5, 1.0$ приводит к переобучению, так как тренировочные данные почти не изменяются
- Смещения на 2px ведет в существенному ухудшению качества. Поэтому было решено не проверять 3px.
- Повороты на 5 и 10 градусов тоже ведут к переобучению, поэтому используются только повороты на 15 градусов.

Комбинируя оставшиеся преобразования, получаем выборку размером в 4 раза больше изначальной и улучшение качества на валидационной выборке почти на 1 процентный пункт (см. Таблицу 7).
На Рис. 5 можно посмотреть, какие ошибки удалось исправить аугментацией тренировочных данных. Заметим, что почти везде на диагонали удалось увеличить число правильных ответов. Исходя из этой матрицы, алгоритм начал лучше отличать пары `3/8`, `7/9`, `5/3` и еще некоторые цифры. Однако немного упало качество для пары `7/2` и для пар `5/8`, `7/4`. В целом удалось повысить точность.

## 2.6 Эксперимент 6

В этом эксперименте предлагается сделать аугментацию тестовых данных и провести голосование. Таким образом для каждого преобразования будем иметь `10000` ответов. Перед голосованием избавимся от ответов, которые имеют низкую точность (меньше 0.95). По результатам голосования построим относительную матрицу ошибок.

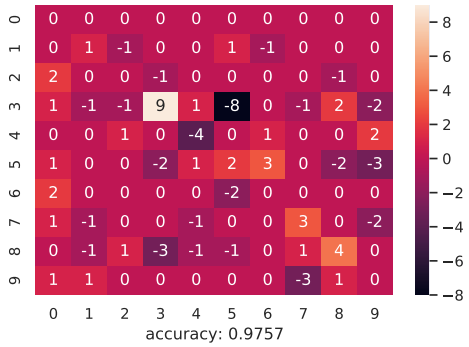


Рис. 6: Ошибки после аугментации данных

Таблица 8:

| | До аугментации | После аугментации |
|---|---|---|
| accuracy | 0.9730 | 0.9757 |

### Выводы

Заметим, что алгоритм стал лучше отличать некоторые пары цифр, например, `3/5` и `7/9`. Однако чуть ухудшилось качество на цифре `4`. В целом удалось увеличить accuracy почти на 0.3 процентных пункта. Однако в данном случае лучше работает аугментация тренировочных данных.

# 3 Вывод

Во время выполнения данного задания удалось реализовать метрический алгоритм `k` ближайших соседей и подобрать гиперпараметры по кросс-валидации. С помощью работы с изображениями и аугментации данных удалось добиться качества `98.08%` на валидационных данных.