Практикум на ЭВМ

Практическое задание №1
«Метрические алгоритмы классификации»

Георгий Демин
студент 317 группы ВМК МГУ,

30 октября 2018 г.
Москва

1 Введение

В данном отчете представлены результаты выполнения практического задания № “Метрические алгоритмы классификации” по курсу “Практикум на ЭВМ” кафедры ММП факультета ВМК МГУ. В задании изучался алгоритм k-ближайших соседей (*k-nearest neighbors*), а также некоторые методы работы с изображениями на основе базы данных рукописных цифр “MNIST”. Были проведены все необходимые эксперименты (кроме последнего) и по результатам сделаны выводы о реализациях данного алгоритма, а также быстродействии и точности в зависимости от параметров (метрики, наличия весов и k)

2 Эксперименты

2.1 Время работы в зависимости от стратегии и количества признаков

В первом эксперименте необходимо исследовать, как будет меняться время работы различных алгоритмов поиска ближайших соседей при увеличении признакового пространства. Для каждого алгоритма выбираем 10, 20, 100 признаков и вычисляем время их работы (см Рис. 1) Мы видим, что время работы у алгоритмов библиотеки *sklearn* намного меньше, чем у алгоритма, реализованного самостоятельно, оно и понятно, ведь эта библиотека специально написана и оптимизирована для сложных математических вычислений. Кроме того, мы видим, что алгоритм *brute* работает в десятки раз быстрее даже при 100 признаках и в десятки раз быстрее даже, чем *kd_tree* и *ball_tree* (масштаб столбчатой диаграммы (bar plot) *brute* специально был оставлен таким же, как и у других алгоритмов для наглядности). Связано это с тем, что алгоритмы поиска по деревьям становятся неэффективны при больших размерностях признакового пространства (долгое построение дерева из-за проклятия размерности (curse of dimensionality))

Исходя из данного эксперимента, будем использовать далее алгоритм *brute*.

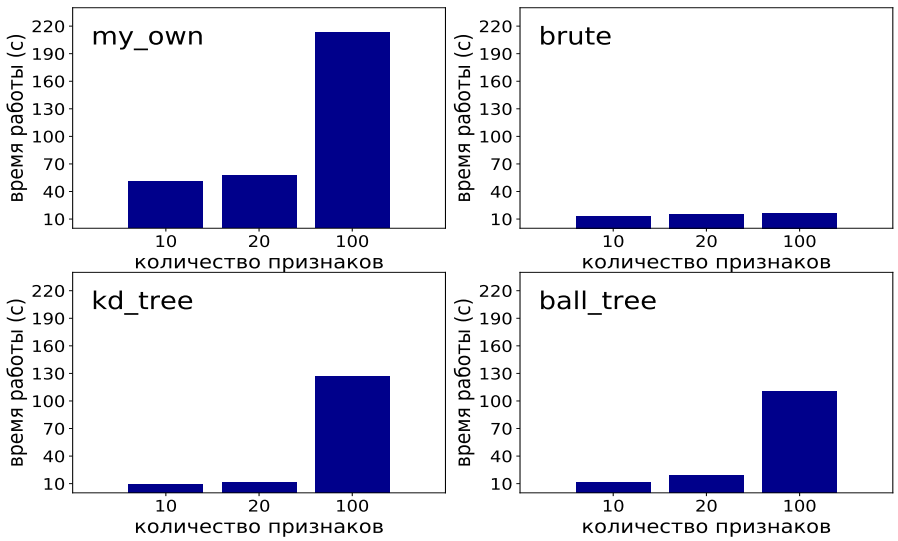

Рис. 1: Время работы алгоритмов при разном количестве признаков

2.2 Точность и время работы в зависимости от параметров

Во втором эксперименте будем исследовать точность и время работы алгоритма с помощью кросс-валидации(все представленные значения являются средним по трем фолдам (folds)) в зависимости от:

- взвешенный метод или нет
- евклидова (euclidean) или косинусная (cosine) метрика
- k от 1 до 10 (только точность)

На Рис. 2 представлена зависимость точности от количества соседей. Из графика мы видим, что взвешенные алгоритмы работают лучше. Это логично, ведь рядом с объектом есть одинаковое число объектов различных классов, правильно будет отнести его к классу, объекты которого располагаются ближе. Косинусная метрика превосходит евклидову, для объяснения этого факта вспомним, что объекты - это изображения цифр. При подсчете евклидова расстояния большую роль играет насыщенность каждого пикселя, то есть числовое значение признака. Следовательно, цифры, совпадающие по форме, но отличные по насыщенности цвета (но это может влиять, например, цвет чернил ручки или нажим) в евклидовом расстоянии будут находится друг от друга дальше, чем в косинусном, которое учитывает лишь некоторое отношение признаков. Оптимальное число ближайших соседей для нашей задачи равно 4. Этот факт тяжело обосновать теоретически и, скорее всего, на других наборах данных это число также будет отличатся.

Отметим еще один интересный факт. При $k = 2$ невзвешенные методы сильно проседают — скорее всего, это связано с особенность операции $np.argmax()$, которая при равных значениях элементов массива возвращает меньший индекс, следовательно, объект будет отнесен просто к классу с меньшим номером. Возможно, много “9” и “6” были предсказаны “0”. И вообще невзвешенные методы при четном k показывают результат хуже, чем при нечетном.

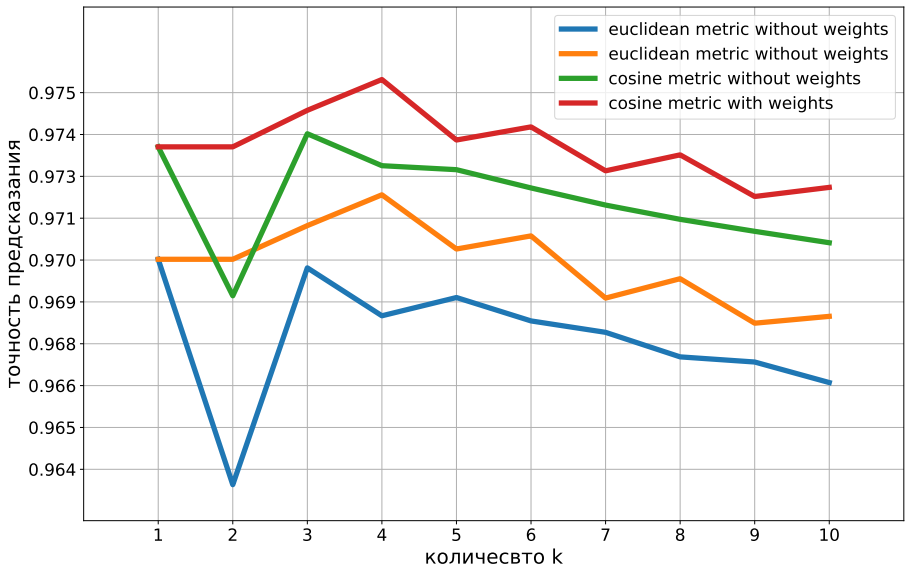

Рис. 2: Точность алгоритма на кросс-валидации при разных параметрах

Обратимся теперь ко времени работы алгоритмов в Таблице 1. Существенных различий нет, объясняется это просто: во всех 4 случаях используются возможности *numpy* для вычисления расстояний и один и тот же код на *python*, реализующий поиск соседей и кросс-валидацию, затраты на деление во взвешенном методе пренебрежимо малы.

Итак самый высокая точность у взвешенного алгоритма 4 ближайших соседей с косинусной метрикой. Далее будем использовать его.

euclidean no weights	euclidean with weights	cosine no weights	cosine with weights
52.763	51.462	52.618	52.753

Таблица 1: Время работы

2.3 Анализ лучшего алгоритма на тестовой выборке

В этом эксперименте проверим выбранные нами параметры на тестовой выборке. Точность на ней оказывается равно 0.9752, по кросс-валидации же средняя точность была 0.9753 — видим, что отличие лишь в 4 знаке после запятой. По матрице ошибок (см. Таблицу 2) видно, что "0" было предсказать проще всего и она была отнесена к другим классам лишь 3 раза. Наоборот "3", "8", "9" было предсказывать труднее всего (объяснением может служить то, что эти цифры "без острых углов", и при их написании рука человека идет более свободно, отчего, возможно, появляется небрежность). Наиболее частая ошибка алгоритма - принять "3" за "9", "7" за "9" и "3" за "5" (происходит это очевидно из-за схожести данных двух цифр в паре). Отдельно отметим, что очень много "4" было принято за "9". Это может быть объяснено тем, как мы строим расстояние между цифрами: мы не учитываем "проходит ли линия" в каком-то участке или нет, мы только считаем насыщенность цвета в каждом пикселе, поэтому легко отличимые глазом "9" и "4" (из-за линии, которая у девятки наверху есть, а у четверки нет), алгоритм принимает за одно и то же число, так как область, совпадения этих цифр зачастую намного меньше размера той самой линии, которой они различаются.

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

Таблица 2: Матрица ошибок

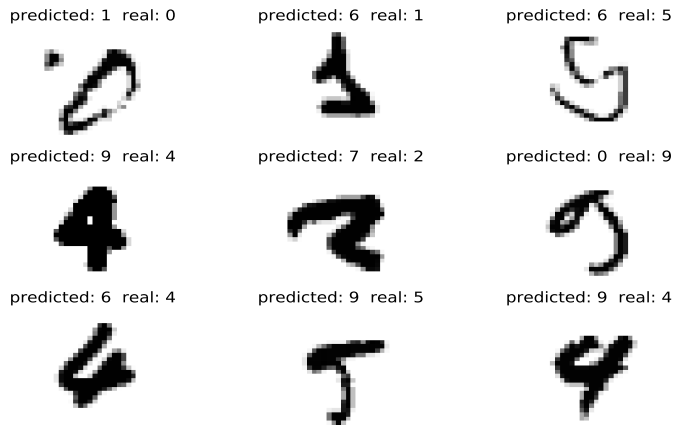

Рис. 3: Ошибочно предсказанные цифры

Посмотрим теперь на примеры изображений, на которых были допущены ошибки на Рис. 3. На 4 и 9 изображении хорошо иллюстрируется высказанное выше предположение о "4" и "9"; на 3, 5, 9 изображениях даже человеку не очень просто сразу определить цифру; на 6 и 7 изображениях видно, что

цифры сильно отличаются от “канонического вида”; а на примере 2, 4, 5, 7, 9 изображений можно увидеть подтверждение гипотезы из второго эксперимента о преимуществе косинусного расстояния из-за разной жирности написания цифр.

2.4 Размножение обучающей выборки. Подбор параметров

Цель данного эксперимента - выяснить какие изменения изображений (повороты, сдвиги, применение гауссовского фильтра и их комбинации) и с какими параметрами позволят добиться существенного улучшения точности прогноза. Мы добавляем к нашей обучающей выборке ее же элементы, но немного измененные. В начале были проверены:

- смещение по вертикали на $1, 2, 3$ пикселя (вверх и вниз)
- смещение по горизонтали на $1, 2, 3$ пикселя (вправо и влево)
- поворот по часовой и против часовой на $5°, 10°, 15°$
- фильтр Гаусса (размытие) $\sigma = 0.5, 1, 1.5$

Для смещений и размытия использовалась библиотека *OpenCV*, для поворота - *skimage* По результатам первоначального отбора было получено, что наибольшее увеличение точности для каждого типа изменения получаются при смещении изображения на 1 пиксель вниз и наверх соответственно, повороте на $10°$ против часовой стрелке и при $\sigma = 1$ в фильтре Гаусса. (см Рис. 4). Почему точность повышается при перемещении именно на *1 пиксель* и именно *вправо и вниз* (а не наверх и влево) объяснить довольно сложно, однако для поворота и размытия можно предложить интуитивное объяснени. Люди в основном правши и пишут с наклоном вправо, но с разным наклоном, поэтому при повороте против часовой стрелки все наклоненные цифры как бы нормализуются и все и становятся ближе (в терминах расстояний) к своему “эталону”. Размытие увеличивает точности по причине того, что цифры,при таком преобразовании “размываются”, интенсивность их пикселей становится меньше и (в терминах косинусного расстояния) они становятся ближе к началу координат и становятся ближайшими соседями для большего числа объектов своего класса.

Далее были проверены комбинации преобразований с лучшими параметрами. Теперь ко всей обучающей выборке добавили ее объекты, измененные с помощью одного преобразоания и с помощью другого (то есть ее объем увеличился в 3 раза). Ожидаемо лучше всего показала себя комбинация из лучших преобразований первого этап с точностью 0.9834, также отметим комбинацию размытия и смещения вправо с точностью чуть меньше 0.0983. Хорошее объяснение тут предложить сложно, но можно предположить, что такова особенность этого набора данных.

Сравним теперь матрицы ошибок алгоритма, который работал с обычной обучающей выборкой, с алгоритмом с выборкой, размноженную лучшими преобразованиями (см Таблицу 4 - она получена путем вычитания матрицы ошибок данного эксперимента из матрицы ошибок эксперимента № 3). Видно, что в основном точность увеличилась за счет правильного предсказания “8” и “9”, причем четверок, которых алгоритм отнес к девяткам стало на 34% меньше. Понятно, почему так произошло: после размытия все изображения девяток “распухли” и у этих изображений стало больше значащих пикселей (в том числе и у верхней линии, о которой говорилось в 4ом пункте и которая очень важна для отделения “4” и “4”). Отсюда становится ясно, почему размытие с дисперсией 1.5 оказалось хуже 1: изображения слишком “разбухли” и девятки с восьмерками стали больше походить на нули. Стоит отметить, что предсказания на всех классах, кроме двойки улучшились (а на двойке осталось прежним), это значит, что мы получили алгоритм абсолютно лучше первоначального на этом наборе данных.

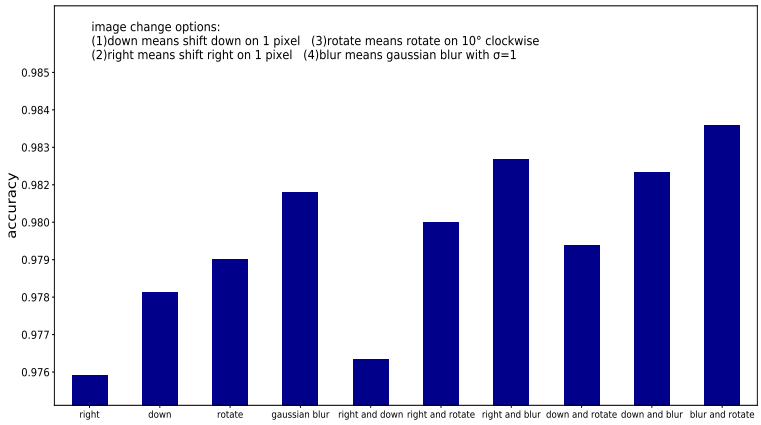

Рис. 4: Точность при изменении изображений

	0	1	2	3	4	5	6	7	8	9
0	977	1	0	0	0	0	0	2	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	6	1	1009	2	1	0	2	10	1	0
3	0	0	2	987	1	7	0	4	5	4
4	0	0	0	0	958	0	4	4	0	16
5	2	0	0	5	1	874	4	1	1	4
6	3	3	0	0	0	3	949	0	0	0
7	0	7	4	1	0	0	0	1009	0	7
8	2	0	1	2	2	4	3	4	954	2
9	2	4	0	2	5	2	1	6	3	984

Таблица 3: Матрица ошибок при добавлении к обучающей выборке повернутых и размытых элементов

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	-1	1	0	0
1	0	4	-1	-1	0	0	-2	0	0	0
2	-2	1	0	1	0	0	2	2	-4	0
3	0	-1	-1	11	0	-5	0	0	-4	0
4	-2	-1	0	0	12	0	-2	2	0	-9
5	-2	0	0	-4	0	11	-3	0	-3	1
6	0	0	0	0	-1	0	1	0	0	0
7	-2	-3	0	1	-1	0	0	11	0	-6
8	-5	-1	-1	-7	-1	1	-2	0	18	-2
9	-5	-3	-2	-3	-2	-1	0	2	0	14

Таблица 4: Отличие матрица ошибок из эксперимента 3 и матрицы ошибок эксперимента 4