Метрические алгоритмы классификации
Алгоритм классификации KNN

Григорьев Илья, 317 группа

Октябрь, 2020

Введение

Задание заключалось в написании алгоритмов классификатора KNN, кроссвалидации и подсчета матрицы попарных расстояний между векторами. Затем надо было провести ряд экспериментов с этими алгоритмами на датасете изображений цифр MNIST.

Датасет MNIST состоит 70000 изображений рукописных цифр от 0 до 9 в оттенках серого размера 28 на 28 пикселей. Выборка была поделена на обучающую (60000 изображений) и тестовую (10000 изображений). Суть классификации заключалась в определении, какая цифра изображена на картинке. То есть формально это задача 10 классовой классификации.

Эксперимент 1

Этот эксперимент заключался в исследовании, какой алгоритм поиска ближайших соседей будет работать быстрее в различных ситуациях. Для каждого объекта тестовой выборки надо было найти 5 его ближайших соседей в обучающей для евклидовой метрики. Изображения имеют 784 признака, но алгоритмы поиска соседей проверялись на изображениях с выбранным подмножеством признаков размера 10, 20 и 100. Тестировалась скорость работы следующий алгоритмов: авторская реализация („my_own“), brute-force алгоритм из библиотеки sklearn („brute“), kd-tree алгоритм из библиотеки sklearn („kd_tree“), ball-tree алгоритм из библиотеки sklearn („ball_tree“).

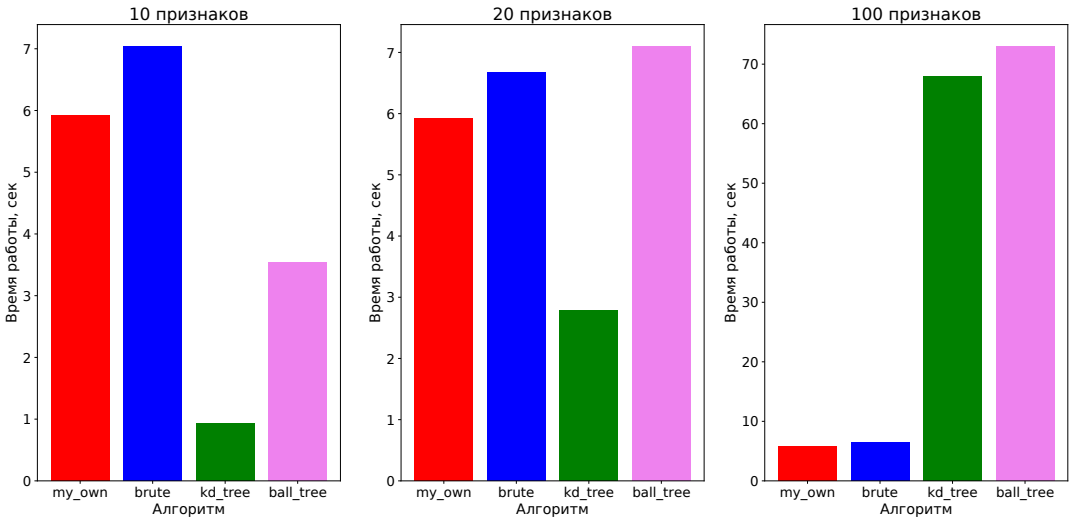

Рис. 1

Из диаграммы видно, что в 10-мерном признаковом пространстве быстрее всех оказывается алгоритм kd-tree. Алгоритм ball-tree медленнее, но все еще быстрее алгоритмов, которые не структурируют пространство. С ростом числа признаков, алгоритмы, основанные на деревьях, теряют свою эффективность. Уже в 100-мерном пространстве авторский алгоритм работает в 11 раз быстрее kd-tree и в 14 раз быстрее ball-tree. Brute-force алгоритм из sklearn чуть медленнее авторского алгоритма, но эффективнее по памяти.

Из этого эксперимента можно сделать вывод, что kd-tree и ball-tree алгоритмы поиска ближайших соседей очень эффективны при небольшом количестве признаков (до 20). С ростом числа признаков, они резко теряют эффективность. Так как изображения из датасета MNIST имеют 784 признака, везде далее использовался авторский алгоритм поиска ближайших соседей.

Эксперимент 2

Во втором эксперименте была оценена по кросс-валидации с 3 фолдами точность (доля правильно предсказанных ответов) и время работы алгоритма классификации KNN в зависимости от количества соседей (гиперпараметр k) и того, какая метрика используется: евклидова или косинусная. Деление на фолды равномерное, объекты в обучающей выборке перемешиваются, алгоритм KNN без весов.

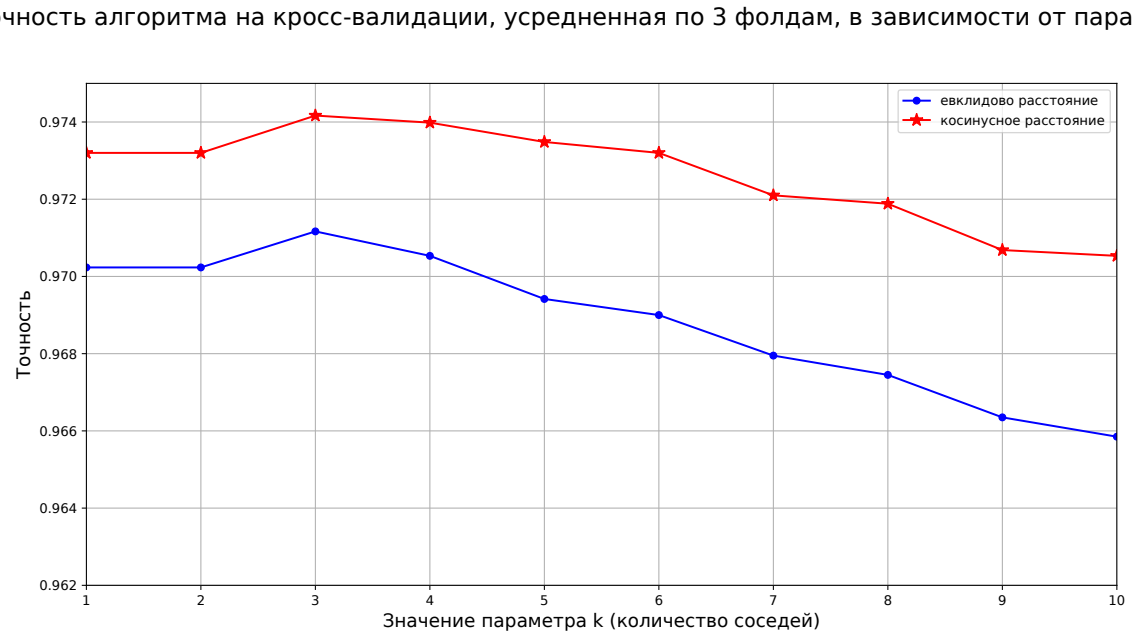

Рис. 2

Выбор евклидовой или косинусной метрики не влияет на скорость работы классификатора. Для всех значений гиперпараметра k от 1 до 10 средняя точность по фолдам при использовании косинусной метрики выше, чем при использовании евклидовой метрики. Наибольшая точность классификации достигается при $k = 3$ как для евклидовой метрики, так и для косинусной. Точность при $k = 3$ равна 0.9712 для евклидовой метрики и 0.9742 для косинусной.

Эксперимент 3

В этом эксперименте было произведено сравнение взвешенного метода KNN, где голос объекта равен $1/(distance + eps)$, где $eps = 10^{-5}$, с методом без весов на кросс-валидации с 3 фолдами. Гиперпараметр k так же подбирался в диапазоне от 1 до 10.

Скорость работы алгоритма с евклидовой метрикой и с косинусной метрикой так же не отличается. Но взвешенный алгоритм работает примерно на полминуты дольше, чем алгоритм без весов. Использование взвешенного метода KNN дает существенный прирост точности классификации по сравнению с методом без весов, особенно в сочетании с косинусной метрикой. Наилучшей точности на кросс-валидации удалось достигнуть используя взвешенный алгоритм KNN при $k = 4$ и косинусной метрики. Точность составила 0.9754.

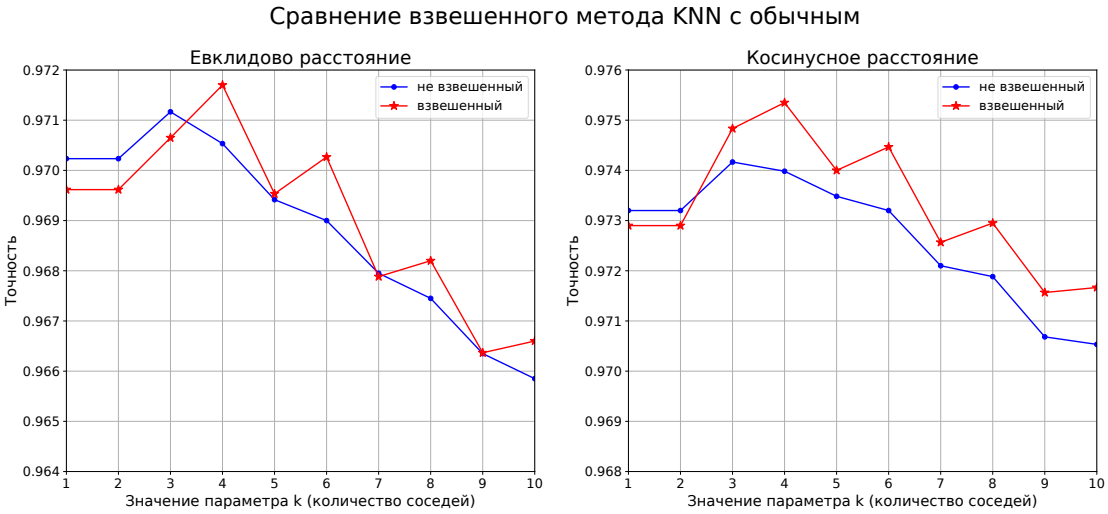

Рис. 3

Эксперимент 4

Взвешенный метод KNN с $k = 4$ и косинусной метрикой был применен к обучающей и тестовой выборке. Точность предсказаний на тестовой выборке составила 0.9752, что достаточно близко к точности на кросс-валидации с такими же параметрами, усредненной по 3 фолдам.

Лучшие алгоритмы показывают точность классификации на датасете MNIST вплоть до 0.9979, но все они получены с помощью нейросетей и, скорее всего, с аугментацией изображений.

Построим матрицу ошибок нашей классификации. Надо учитывать, что количество изображений цифр из разных классов может различаться. Например, в нашей тестовой выборке количество изображений единиц равно 1135, а количество изображений цифры пять равно 892. Лучше всего алгоритм классифицирует цифру ноль с точностью 0.9969, а хуже всего — цифру восемь с точностью 0.9610. На изображении с цифрой восемь классификатор, ошибаясь, часто выдает в качестве ответа три или ноль, что логично, так как эти цифры действительно похожи в написании. Так же алгоритм плохо работает на изображениях с цифрами девять и четыре. Если на изображениях с цифрой девять алгоритм ошибается достаточно равномерно, то на изображениях с цифрой четыре больше всего промахов было в пользу класса с цифрой девять — 25 штук.

Визуализируем часть тестовых объектов, на которых были допущены ошибки.

Можно заметить, что часть из этих цифр написаны действительно неразборчиво. Так же эти изображения в большинстве своем принадлежат описанным выше классам, на которых чаще всего происходят ошибки классификации.

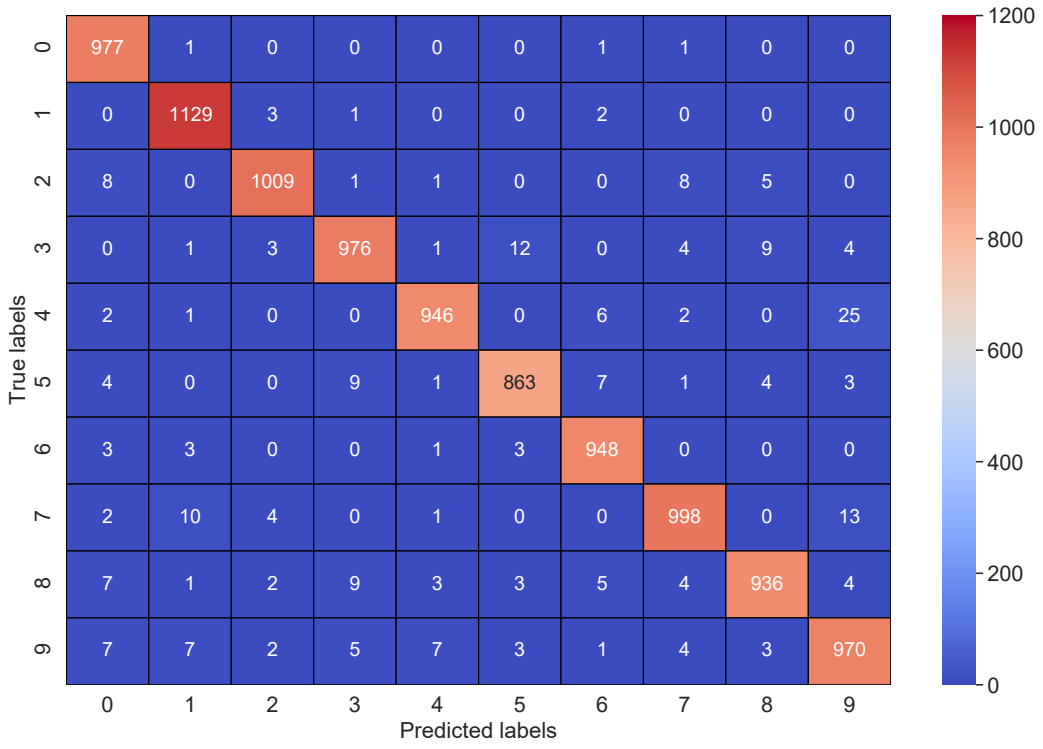

Рис. 4: Матрица ошибок для классификации без аугментации

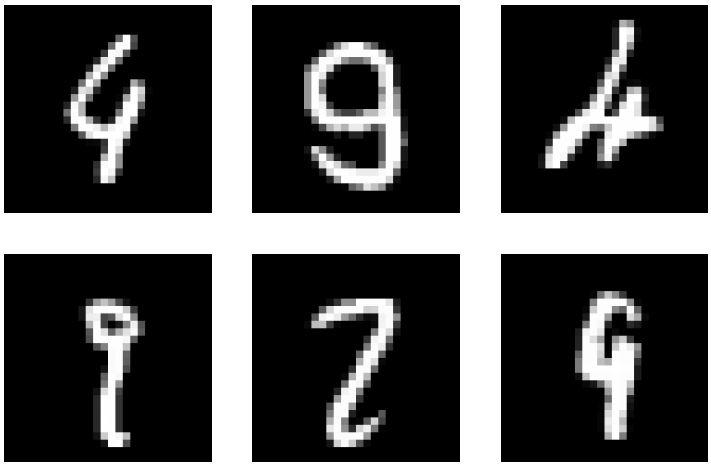

Рис. 5: Неправильно классифицированные изображения

Эксперимент 5

В этом эксперименте была сделана аугментация обучающей выборки. Для преобразования изображений применялись повороты на различные углы в разные стороны, сдвиги на разное число пикселей по каждой из двух размерностей в обе стороны и фильтр Гаусса с разными значениями дисперсии. Параметры преобразований подбирались по кросс-валидации с 3 фолдами. Наилучшим преобразованием оказалась комбинация поворотов в обе стороны на 5 градусов с сдвигами на 1 пиксель по каждой размерности. Оптимальное значение гиперпараметра k с такой аугментацией обучающей выборки равно 4, метрика косинусная, веса включены. На тестовых данных удалось поднять точность классификации до 0.9824. Точность на кросс-валидации с такими параметрами равна 0.9839, то есть немного завышена.

Рассмотрим, как изменилась матрица ошибок. Точность классификации увеличилась для всех классов. Сильно удалось поднять точность классификации цифр три и четыре, особенно алгоритм перестал слишком часто путать цифру три с цифрой пять и цифру четыре с цифрой девять. Четыре и девять алгоритм перепутал 15 раз вместо 25, как было до аугментации. Так же существенно увеличилась точность классификации цифр семь, восемь и девять. Но есть моменты, где алгоритм с аугментацией работает чуть хуже, чем без нее. Например, на изображениях с цифрой два алгоритм с аугментацией 10 раз ошибочно предсказал цифру семь, в то время как алгоритм без аугментации ошибся так только 8 раз. Но в целом, конечно, аугментация дает сильный прирост качества модели.

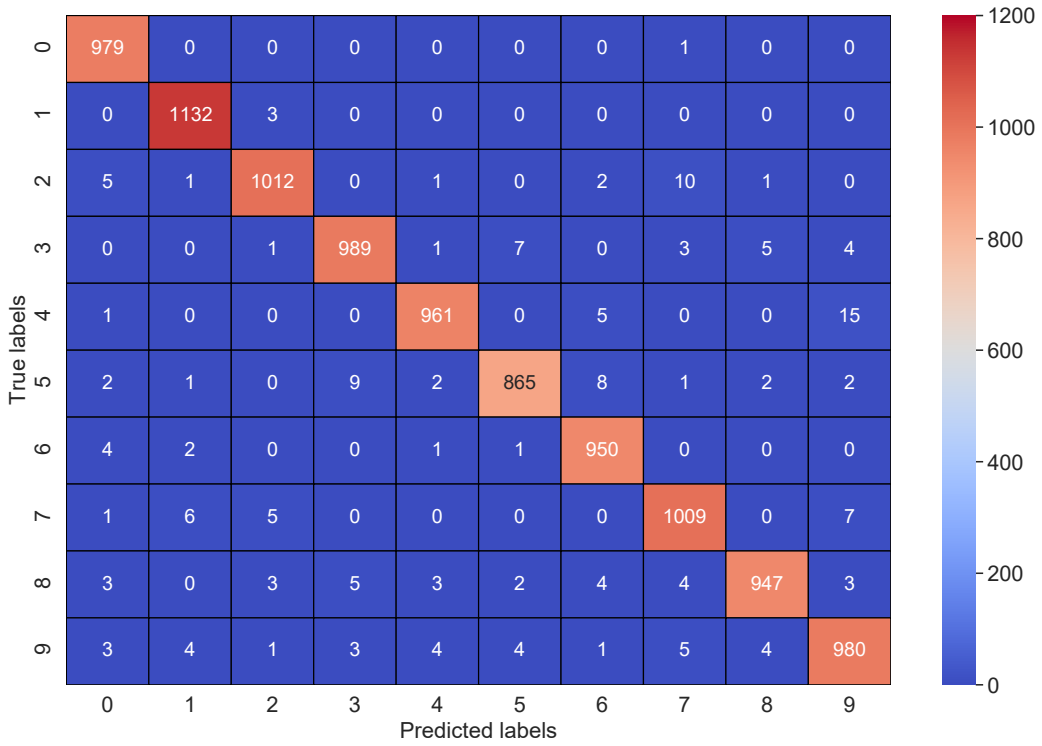

Рис. 6: Матрица ошибок для классификации с аугментацией обучающей выборки

Эксперимент 6

В последнем эксперименте надо было применить аугментацию не к обучающей выборке, а к тестовой выборке. В качестве преобразования изображений была выбрана та же комбинация из поворотов на 5 градусов и сдвигов на 1 пиксель, что и в предыдущем эксперименте. Оптимальное значение параметра k было выбрано равным 4, метрика косинусная, веса включены. Точность на тестовой выборке оказалась равна 0.9763, что выше, чем без аугментации, но сильно ниже, чем с аугментацией обучающей выборки.

По матрице ошибок видно, что улучшилась классификация цифры три, в частности алгоритм стал реже путать ее с цифрой пять. А точность классификации цифры девять наоборот ухудшилась, особенно алгоритм стал чаще путать ее с нулем. В целом, при данном подходе есть как улучшения, так и небольшие ухудшения для отдельных классов. Тем не менее, качество модели немного увеличилось с аугментацией тестовой выборки, чем без аугментации.

Лучше оказался подход с аугментацией обучающей выборки, на нем точность классификации сильно выше. Но такой подход менее эффективен по потреблению памяти. Надо постоянно хранить всю аугментированную обучающую выборку, потому что она будет нужна для каждого тестового объекта. Скорость работы алгоритма с аугментацией обучающей выборки несущественно отличается от скорости работы алгоритма с аугментацией тестовой выборки.

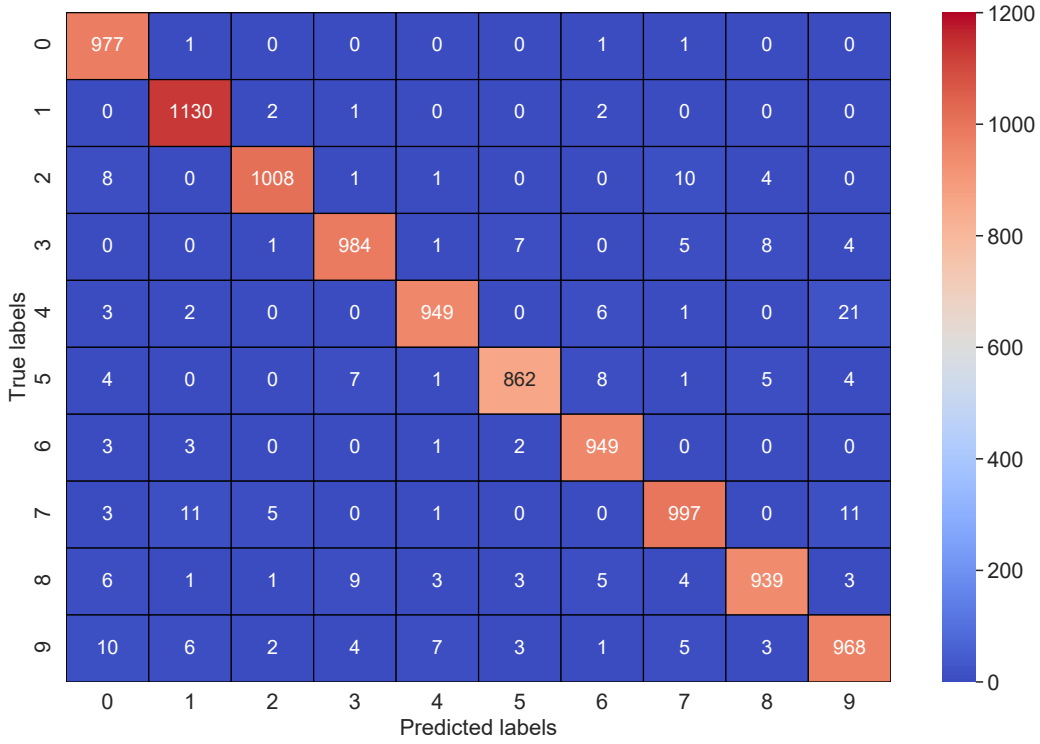

Рис. 7: Матрица ошибок для классификации с аугментацией тестовой выборки