Отчет по третьему заданию по практикуму

Студента 317 группы ВМК МГУ
Драпак Степана

24 ноября 2015 г.

Содержание

1 Описание задания

Задание было посвящено изучению разных вариантов реализации метода опорных векторов и их настройке их параметров. В частности были реалзованы следующие методы:

- Субградиентный спуск
- Стохастический субградиентный спуск
- Метод внутренней точки для решения прямой задачи
- Метод внутренней точки для решения двойственной задачи
- Метод используемый в Liblinear
- Метод используемый в LibSVM

2 Спецификация

Основная часть программы реализована в классе my_svm. В качестве параметров конструктора он принимает строку, которая задает нужный алгоритм решения задачи. Основные методы с которыми предстоит работать - fit и predict. fit принимает на вход помимо обучающей выборки разные настройки алгоритма, как общие, такие как число итераций, так и специфические для каждого метода, такие как кол-во элементов в пакете для пакетной реализации субградиентного спуска. Кроме того в модуле присутствуют вспомогательные функции, такие как функция визуализации, функция вычисления точности. Все эксперементы реализованы в ipython notebook файле, при желание там можно посмотреть более подробные результаты некоторых эксперементов.

Прежде чем перейти к экспериментам выведем формулу для субградиентного шага. Для удобство будем считать, что $w_0 = 0$, при этом в программной реализации добавим в матрице X еще один столбец, все значения которого будут равны единице. После сделаных выше предположений функция, которую нам нужно оптимизировать в прямой задаче принимает вид:

$$f(w) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i max\{0, 1 - (y_i \mathbf{w}\mathbf{x_i})\}$$

Дифференцируя по вектору w, получаем:

$$g(w) = \mathbf{w} + C\sum_i [y_i \mathbf{w}\mathbf{x_i} < 1] y_i \mathbf{x_i}$$

3 Эксперименты

3.1 Исследование сходимости линейного SVM

Здесь мы рассмотрим время сходимости наших методов, а также отклонение их целевых функций в зависимотси от числа объектов и размерности признакового пространства. В качестве эталона возьмем значение целевой функци от SVM из библиотеки LibSVM. Посмотрим на графики 1 - 3:

Из графиков видно:

- Метод внутренней точки сходится очень быстро, значительно быстрее чем субградиентные методы. Возможно это связано с необходимостью подбора коэффициента, отвечающего за скорость обучения, но об этом ниже.

- Стохастическсий субградиент в среднем сходится быстрее чем обычный. На графиках четко видно, когда начинается обработка нового пакета данных, но после обработки пакета значения становится весьма неплохими.

- При большом объеме выборки, почти для всех размерностей, простой градиентный метод сильно деградирует.

Теперь посмотрим на таблицу, в которой показана скорость сходимости:

	Размер обучающей матрицы	SVC	Subgradient	Batch subgradient	Внутреняя точка
	200 x 2	0.001s	0.1s	0.2s	0.2s
	1000 x 2	0.005s	0.17s	0.36s	36s
	200 x 10	0.0006s	0.12s	0.2s	0.25s
	1000 x 10	0.002s	0.26s	0.88s	37.6s

Из это таблицы бросается в глаза:

- SVC на порядки быстрее всех остальных.

- Метод внутренней точки очень сильно деградирует с ростом размерности пространства при больших выборках

3.2 SVM с RBF ядром

Здесь вновь берем за образец реализацию из LibSVM. Посмотрим на графики 4 и 5

Видно, что от кол-ва объектов характер поведения функции сильно не меняется. Понятно, что целевая функция при этом больше при больших выборках, но сходится в итоге примерно одинаково быстро.
Что касается времени, посмотрим на таблицу ниже:

	Размер выборки	SVC	Внутренняя точка
	200 x 2	0.028s	0.22s
	1000 x 2	0.26s	11s
	200 x 10	0.063s	0.21s
	1000 x 10	0.3s	8.77s

В среднем, время вышло дольше чем в прямой задаче. При этом внутренняя точка опять деградирует на больших выборках. Однако, интересно заметить, что от размеров выборки результаты почти не зависят.

3.3 Подбор коэффициента регуляризации и ширины RBF

Будем это делать на основе метода из LibSVM на кросс-валидации с 3 фолдами. На хорошо разделимой выборке(нормальное распределение с мат. ожиданием по всем параметрам 2 и 4 для класса 1 и -1 соответственно):

	Размер обучающей матрицы	best C	best gamma	best accuracy
	200 x 2	10	0.4	0.955
	1000 x 2	10	0.2	0.936
	200 x 5	0.5	0.9	0.997
	1000 x 5	0.1	0.1	0.987

На плохо разделимой выборке (мат. ожидание 2 и 3):

	Размер обучающей матрицы	best C	best gamma	best accuracy
	200 x 2	1	0.1	0.79
	1000 x 2	1	0.2	0.79
	200 x 5	0.5	0	0.88
	1000 x 5	0.1	0	0.87

По этим таблицам можно заметить, что оптимальное значение C при увеличение размерности уменьшается.

3.4 Исследование стратегий выбора градиентного шага в градиентном спуске

Будем выбирать размер градиентного шага в виде $\frac{\alpha}{t^\beta}$. t — Номер итерации, α, β - Константы. Выборка - плохо разделимая. Результаты приведены в таблице:

	Размер обучающей матрицы	best α	best β	best accuracy
	200 x 2	100	0	0.79
	1000 x 2	1	5	0.80
	200 x 10	1	0.5	0.80
	1000 x 10	0.1	2	0.79

Далее посмотрим за временем работы. Подробные результаты, то есть данные по каждой паре α, β можно посмотреть в ipython, я же здесь приведу график для α при фиксированном β и наоборот. Все подобные закономерности похожи. Обратимся теперь к графику 6.

В целом, понятно, что идея с динамическим выбором шага правильная, к тому же есть необходимое условие сходимости градиентного спуска, которое говорит, что шаг должен стремиться к 0. Из графиков видно, что уменьшение шага не всегда ведет к увеличению времени сходимости, за счет того, что итераций требуется значительно меньше.

3.5 Исследование влияния размера выборки на оптимальный размер пакета в пакетном градиентном спуске

Посмотрим на график 7. График по времени убывает, но после 500 объектов в пакете стабилизируется. Что касается точности, при маленьких пакетах точность маленькая, если вспомнить графики из первого пункта, ничего удивительного, каждые

новые 10 объектов тянут объектную функцию к своему лоакльному минимуму и в итоге все плохо. При больших же пакетах все достаточно стаблильно. Таким образом, для больших выборок целесообразно использовать пакетный алгоритм, чтобы избежать проблем с оперативной памятью.

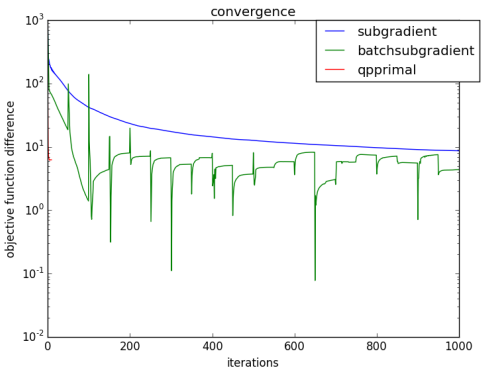
3.6 Визуализация

Визуализация представлена на рисунке 8. В основу функции для визуализации была взят пример с сайта sklearn http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html.

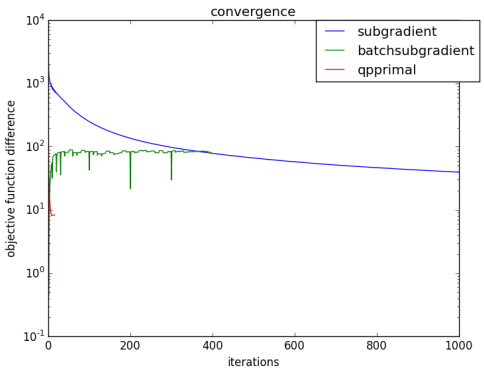
С точки зрения теории все понятно. Опорные вектора, отмеченные двумя кружками лежат ближе всех к разделяющей поверхности. Можно отметить, что RBF ядро достаточно не тривиально определяет область, где лежат точки из "коричневого" класса. Если в тестовой выборке будут выбросы, которые лежат далеко за всеми остальными точками, то алгоритм с RBF ядром будет не верно их классифицировать. Об этом следует помнить, строя модели, особенно в пространствах с большими размерностями, так как там "поводов" оказаться за пределами правильной области значительно больше.

4 Выводы

И того, видно, что SVM достаточно сильный алгоритм классификации, однако он требует внимательно настройки очень большого числа параметров, которые, зависят и от обучающей выборки и от значения других параметров. Все это делает настройку достаточно не тривиальным делом.

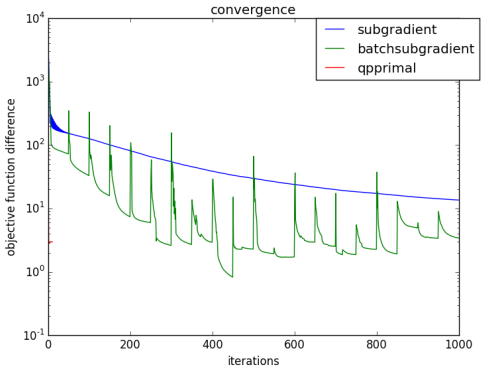


(a)

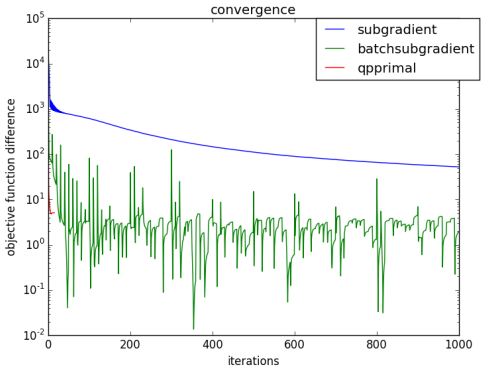


(b)

Рис. 1: Размерность пространства = 2. (a) - 200 объектов (b) - 1000

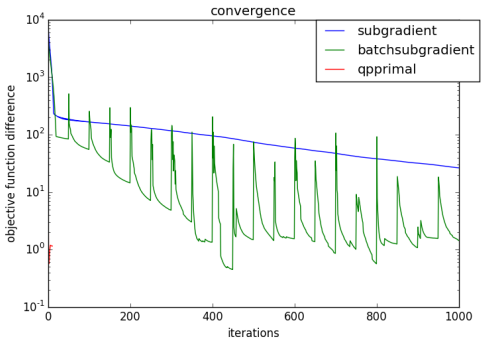


(a)

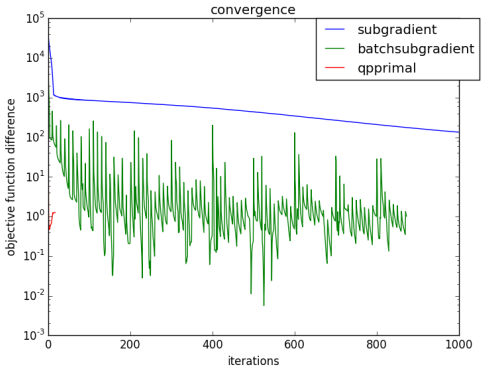


(b)

Рис. 2: Размерность пространства = 5. (a) - 200 объектов (b) - 1000

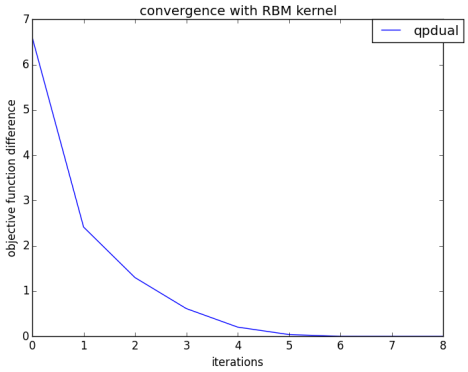


(a)

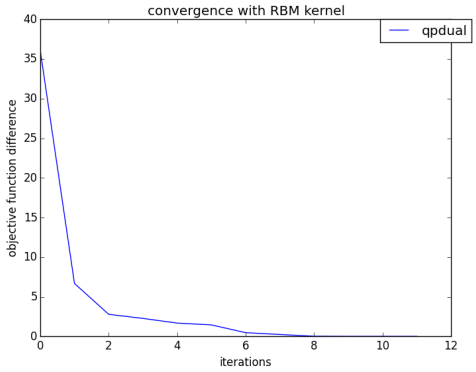


(b)

Рис. 3: Размерность пространства = 10. (a) - 200 объектов (b) - 1000

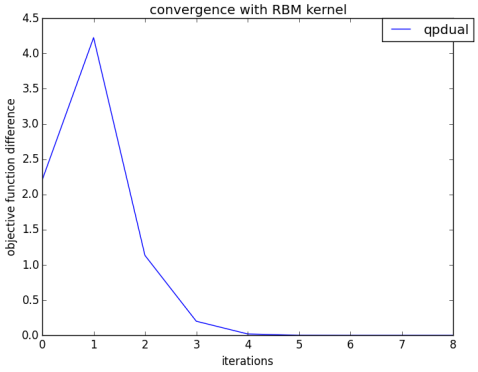


(a)

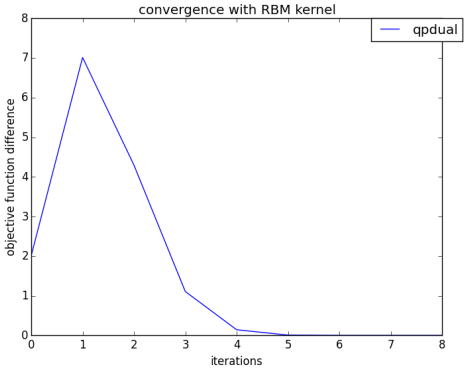


(b)

Рис. 4: Размерность пространства = 2. (a) - 200 объектов (b) - 1000

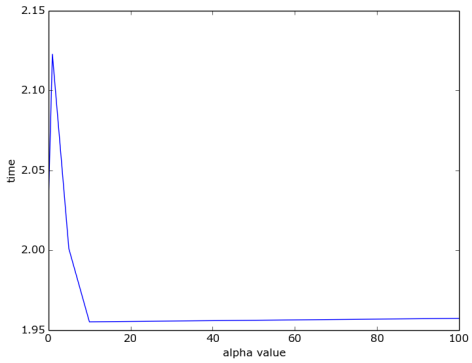


(a)

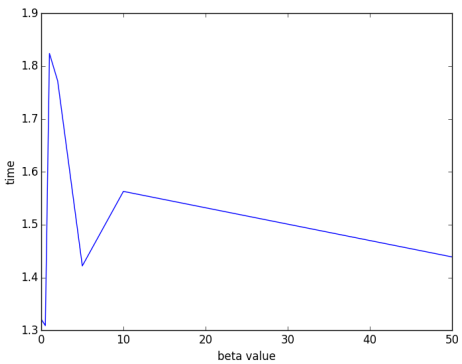


(b)

Рис. 5: Размерность пространства = 10. (a) - 200 объектов (b) - 1000

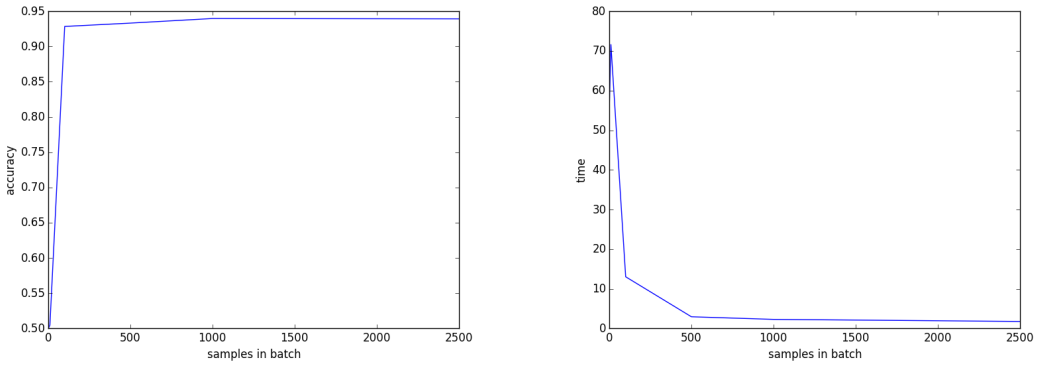


(a)



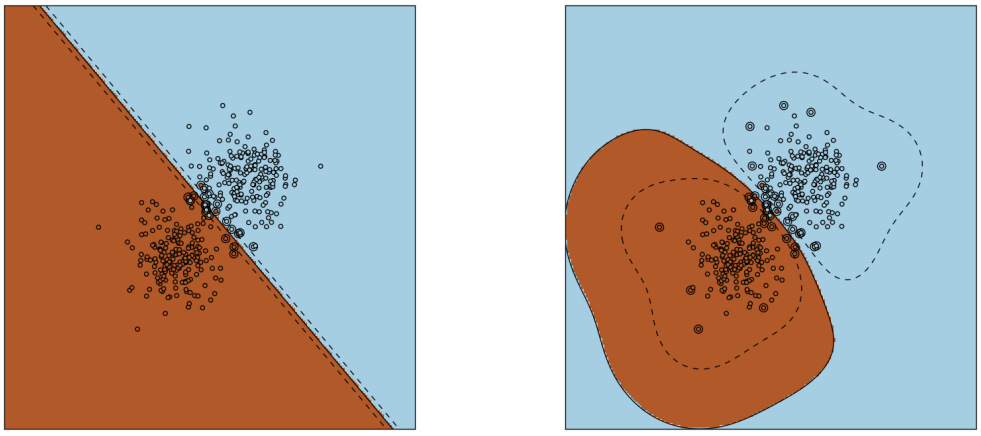
(b)

Рис. 6: Зависимость времени работы субградиентного спуска от (a) — α (b) — β



(a) (b)

Рис. 7: Зависимоть (a) - точности (b) - времени работы от числа объектов в пакек для стохастического субградиентного спуска



(a) (b)

Рис. 8: Визуализация результата работы метода опорных векторов (a) — с линейным (b) — с RBF ядром