Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет к третьему заданию практикума на ЭВМ: Метод опорных векторов

Студент 3 курса ВМК (317 группа):
Оспанов А.М.

Москва, 2014

Содержание

1 Введение

Данный отчет написан к третьему заданию практикума на ЭВМ 317 группы. Тема задания: Метод опорных векторов. Отчет написан студентом 317 группы – Оспановым Аятом.

В данной работе были реализованы разные методы решения задачи SVM. Были проведены исследования по оценке времени работы методов и исследования по подбору констант и прочие исследования. Были проведены тесты применимости данных методов в реальных задачах на примере классификации MNIST.

2 Основная часть

2.1 Предварительный подсчет данных для решения задач

Вывод формулы субградиента для функционала в прямой задаче SVM
Надо найти следующий субдифференциал

$$\partial(\frac{1}{2}\left\|x\right\|^2 + C\sum_{n=1}^{n} max\{0, 1 - y_n w^T x_n\})$$

Для начала найдем субградиент фунцкии

$$F(x) = max\{f_1(x), f_2(x)\}$$

Субградиент для данной функции определяется следующим образом:
Если $f_1(x) > f_2(x)$, то $\partial F = \{\nabla f_1(x)\}$
Если $f_2(x) > f_1(x)$, то $\partial F = \{\nabla f_2(x)\}$
Если $f_1(x) = f_2(x)$, то $\partial F = [min\{\nabla f_1(x), \nabla f_2(x)\}, max\{\nabla f_1(x), \nabla f_2(x)\}]$
Учитывая это, получаем

$$\partial(max\{0, 1 - y_n w^T x_n\}) = \begin{cases} \{0\},\ y_n w^T x_n > 1 \\ \{-y_n x_n\},\ y_n w^T x_n < 1 \\ [min\{0, -y_n x_n\}, max\{0, -y_n x_n\}],\ y_n w^T x_n = 1 \end{cases}$$

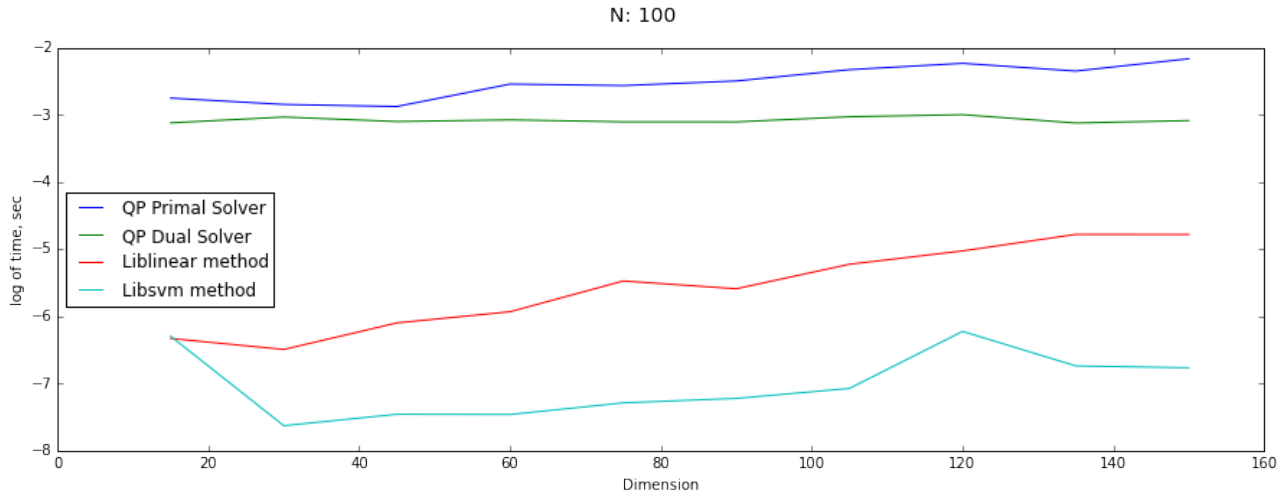
В итоге получим следующий субдифференциал

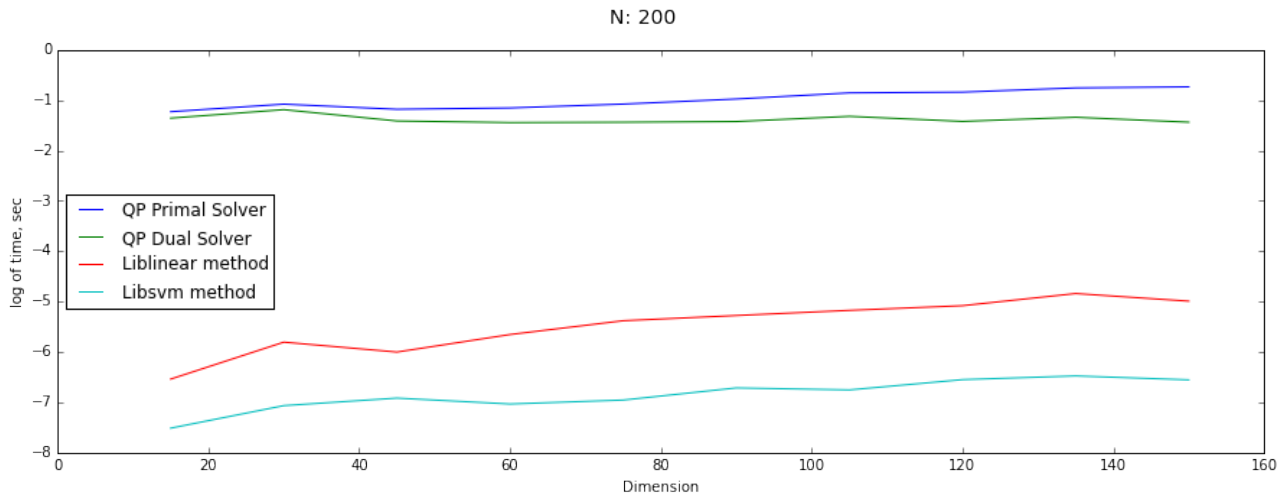
$$\partial\left(\frac{1}{2}\left\|x\right\|^2 + C\sum_{n=1}^{n} max\{0, 1 - y_n w^T x_n\}\right) = w + C\sum_{n=1}^{n} \partial(max\{0, 1 - y_n w^T x_n\})$$

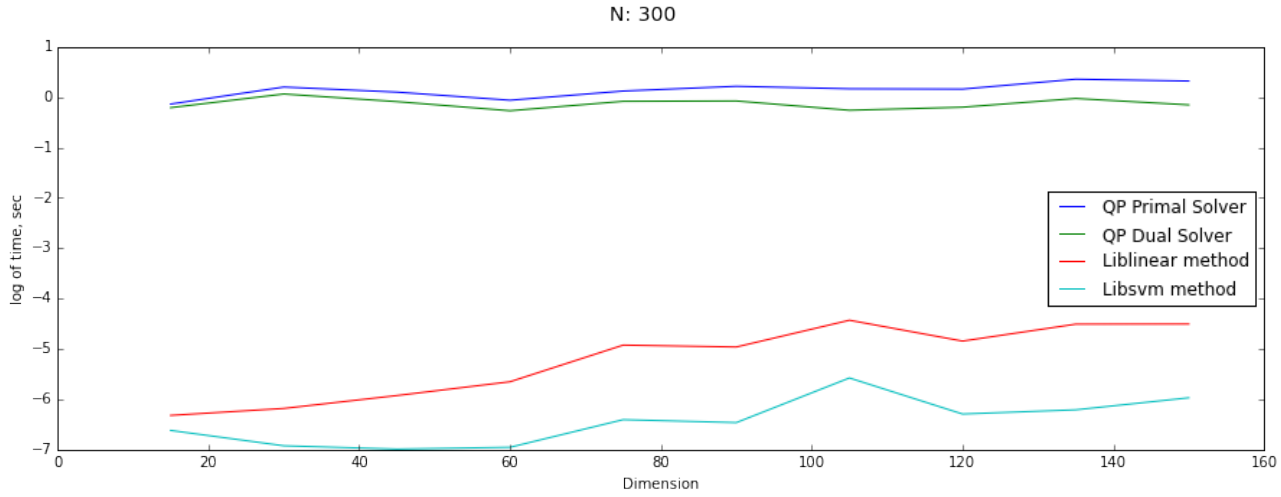
2.2 Пункт 1

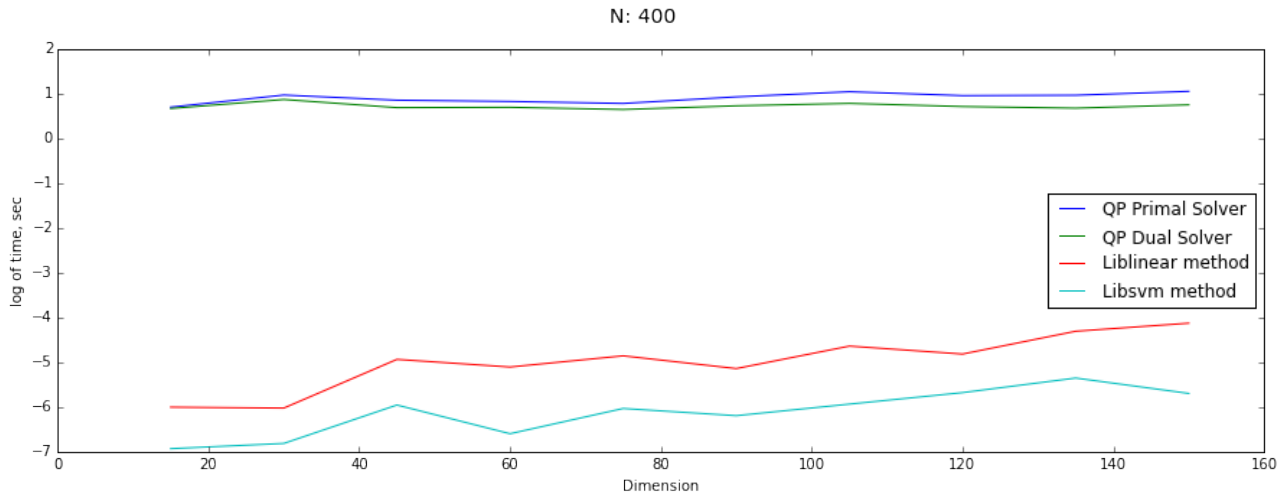
Исследуем зависимость времени работы методов от размерности пространства для каждого размера выборки в методах с линейным ядром.

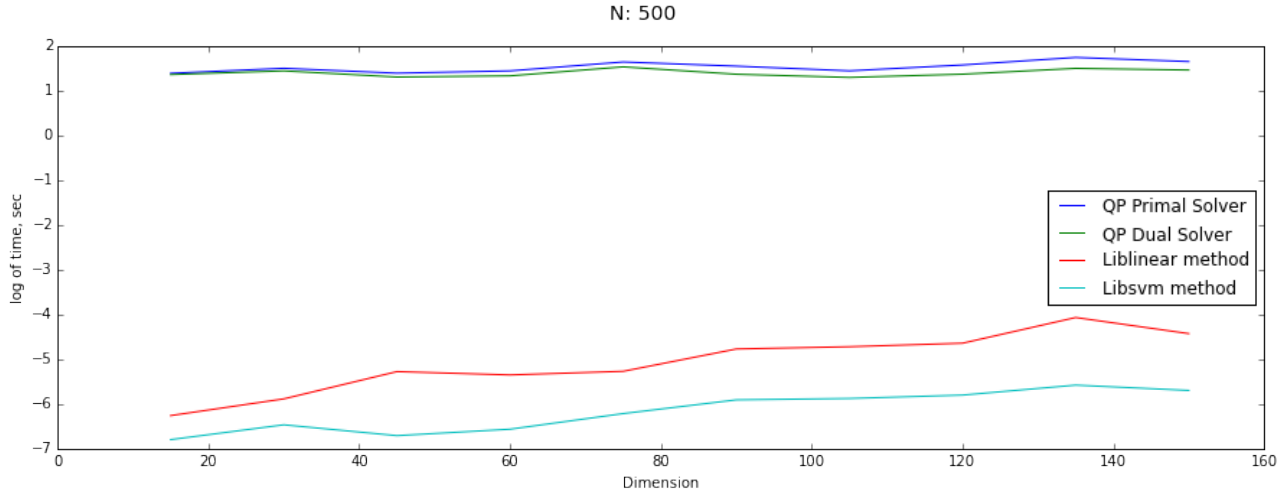
На слелующих фото представлены графики зависимости времени работы от размерности пространства. На них можно увидеть, что самый быстрый метод libsvm, далее liblinear

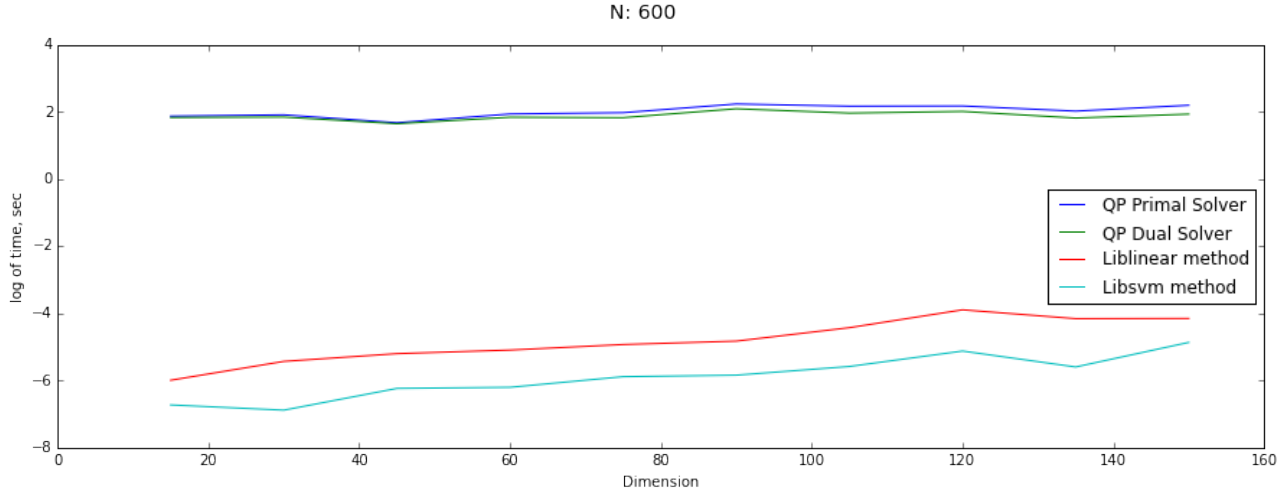


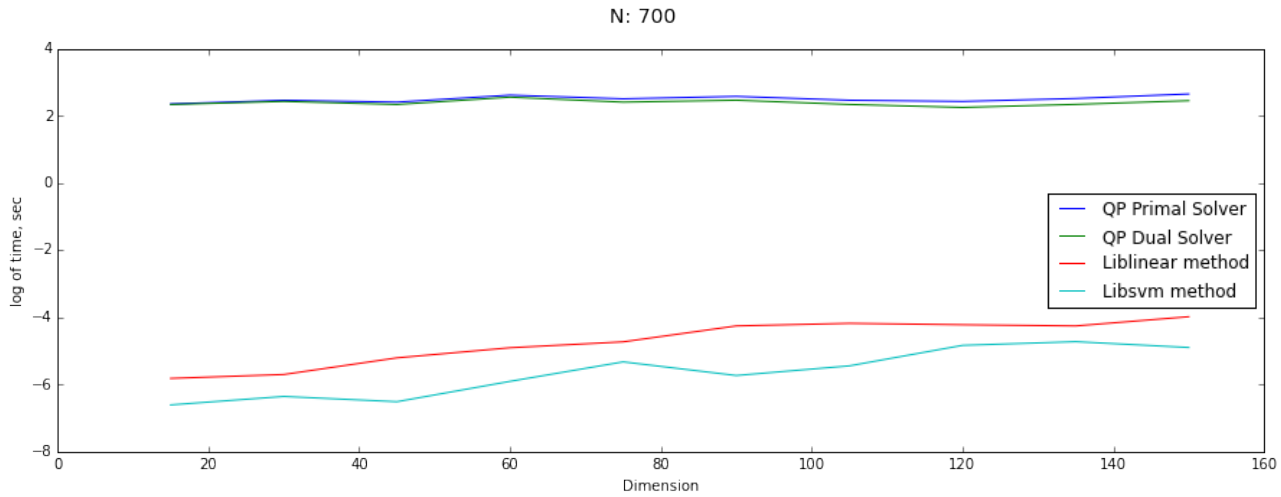


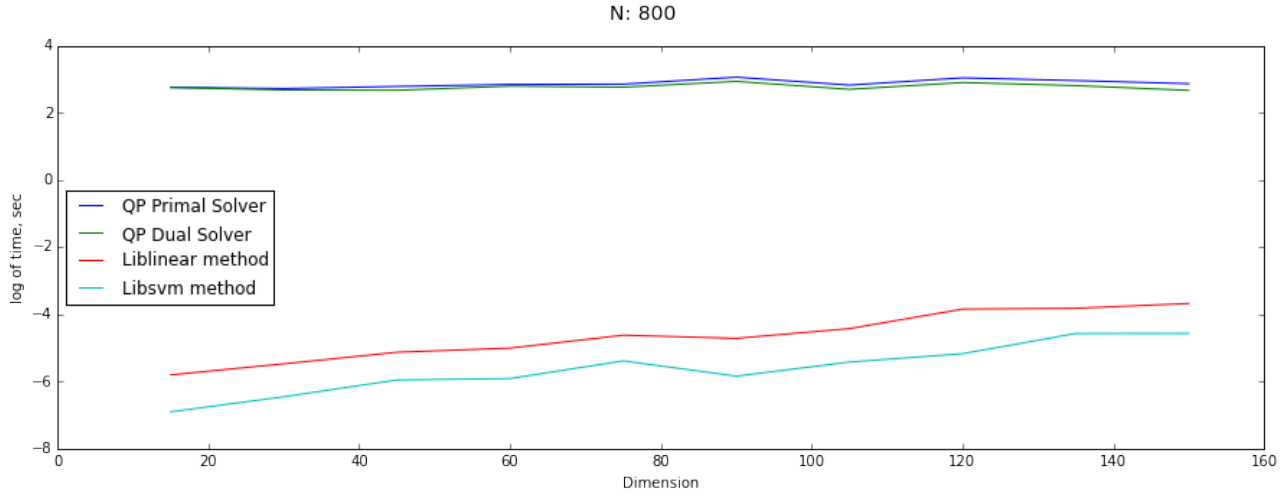


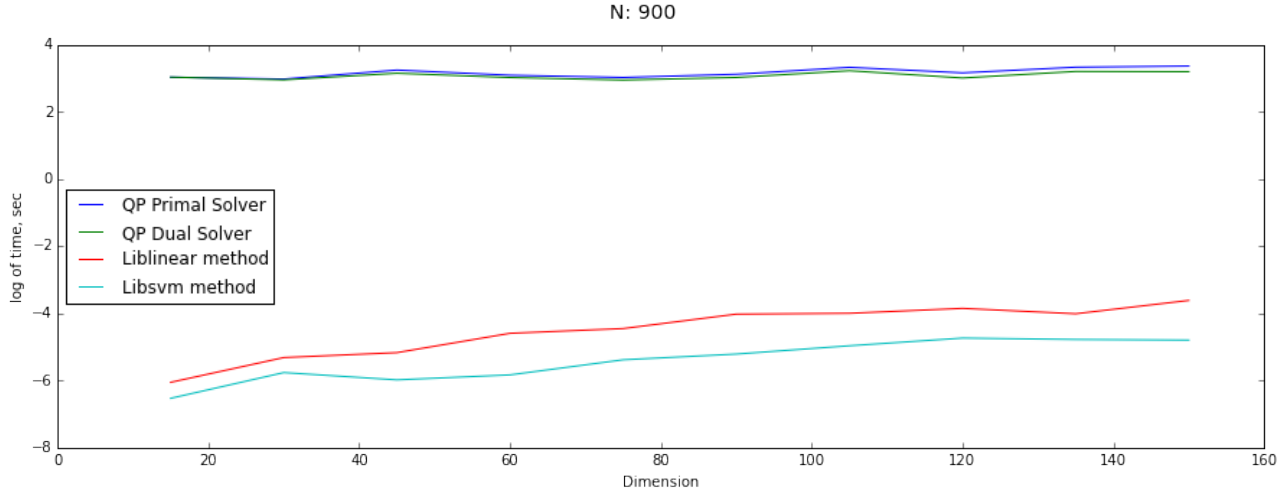

N: 400
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method

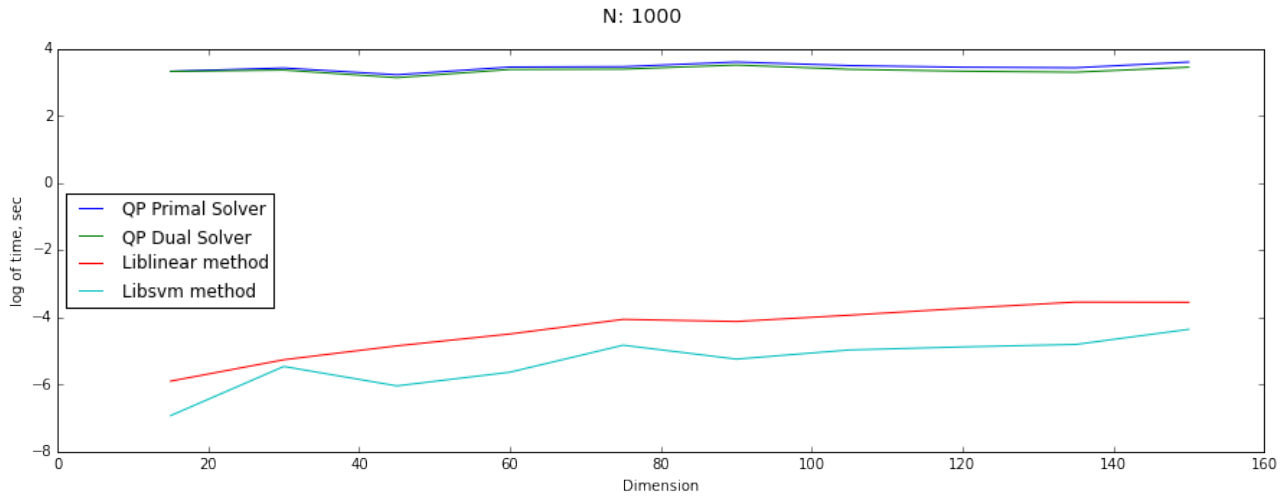

N: 500
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method


N: 600
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method


N: 700
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method

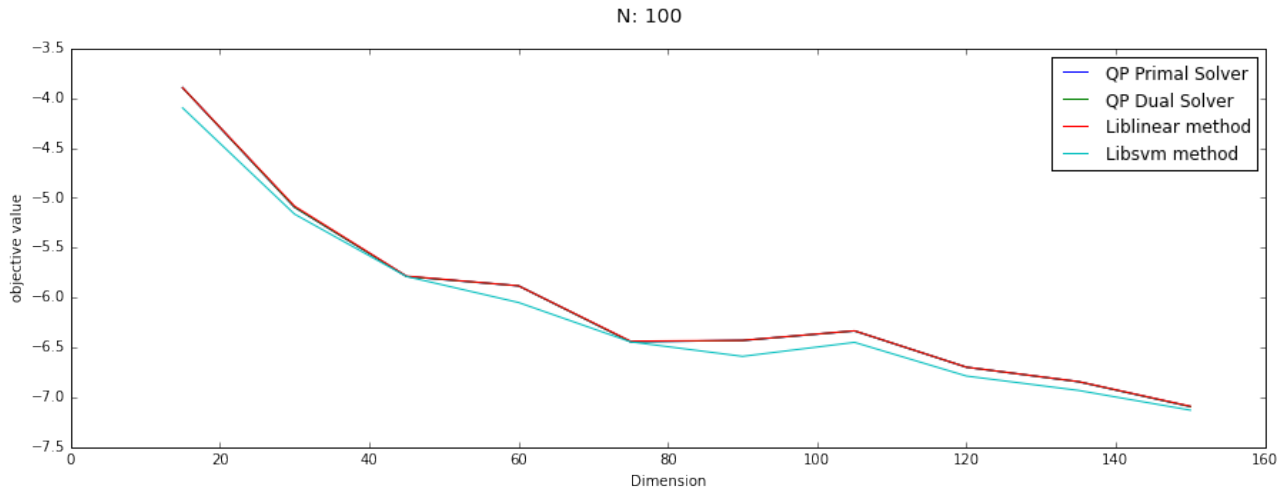

N: 800
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method

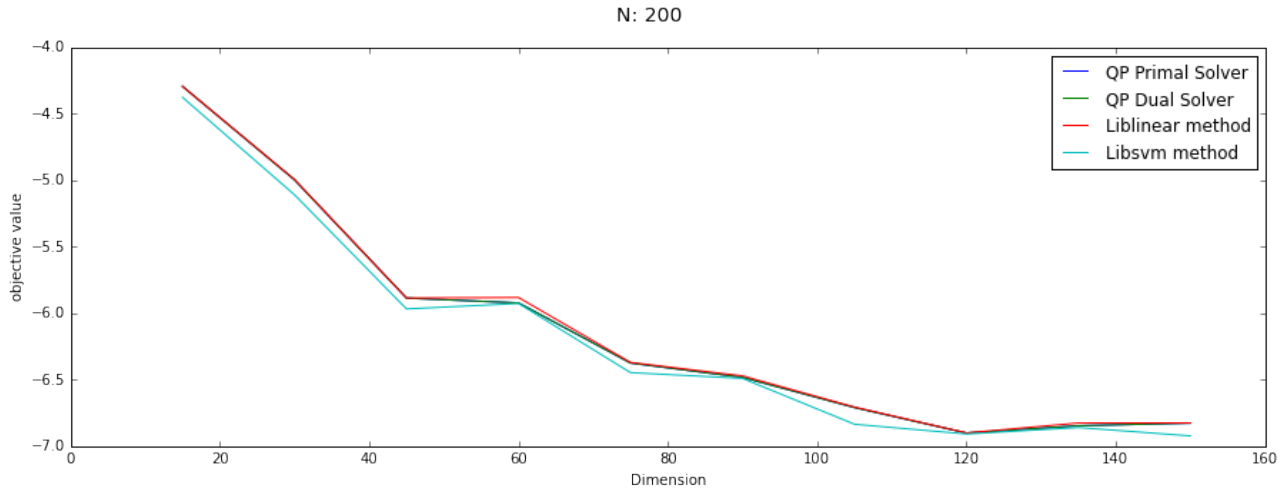

N: 900
log of time, sec
Dimension
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method

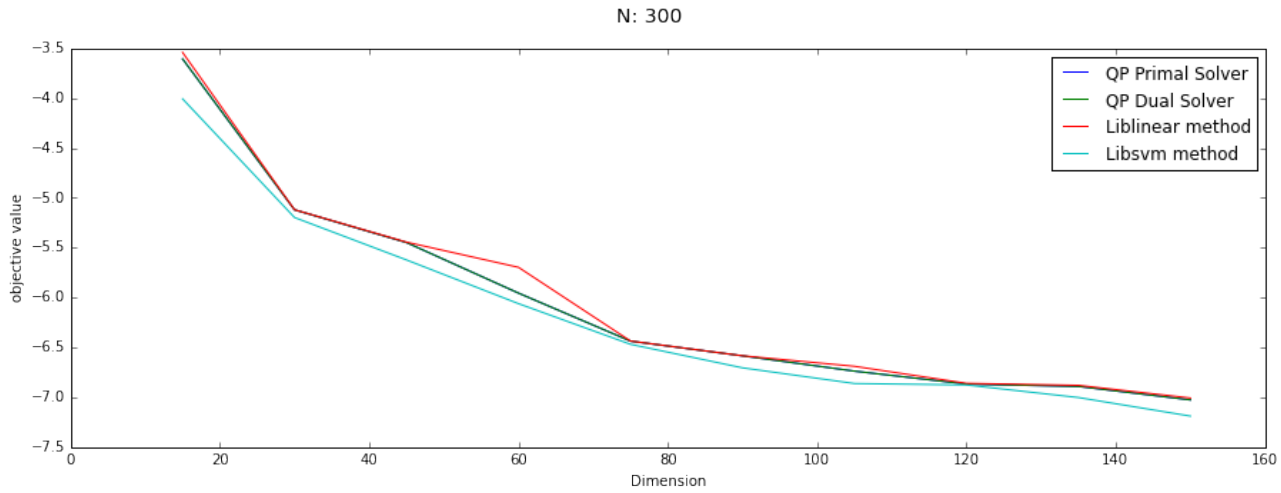


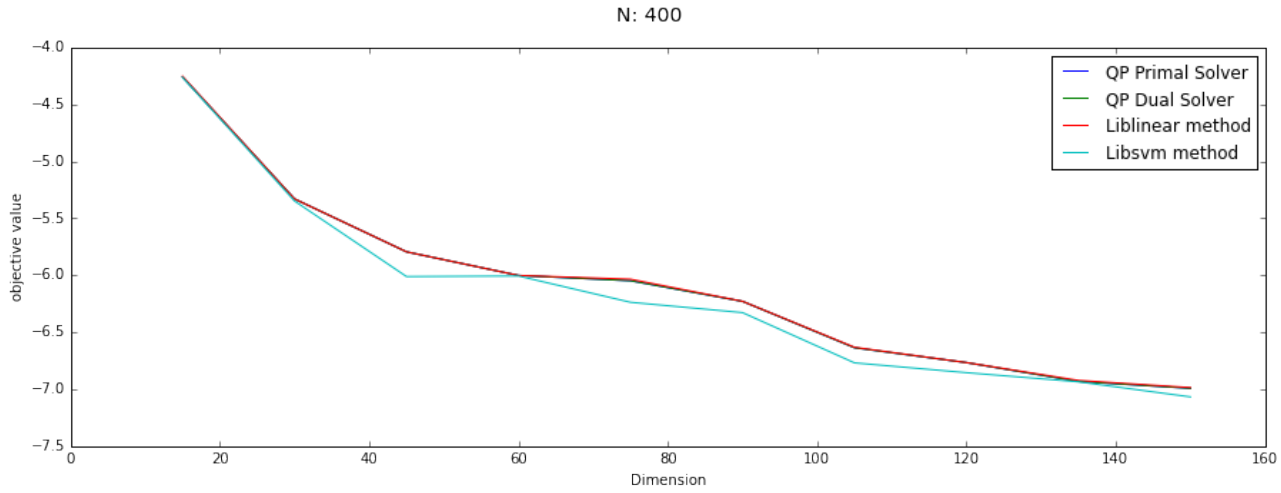
На следующих графиках представлены значения функционалов в зависимости от размерности пространства в методах с линейным ядром.

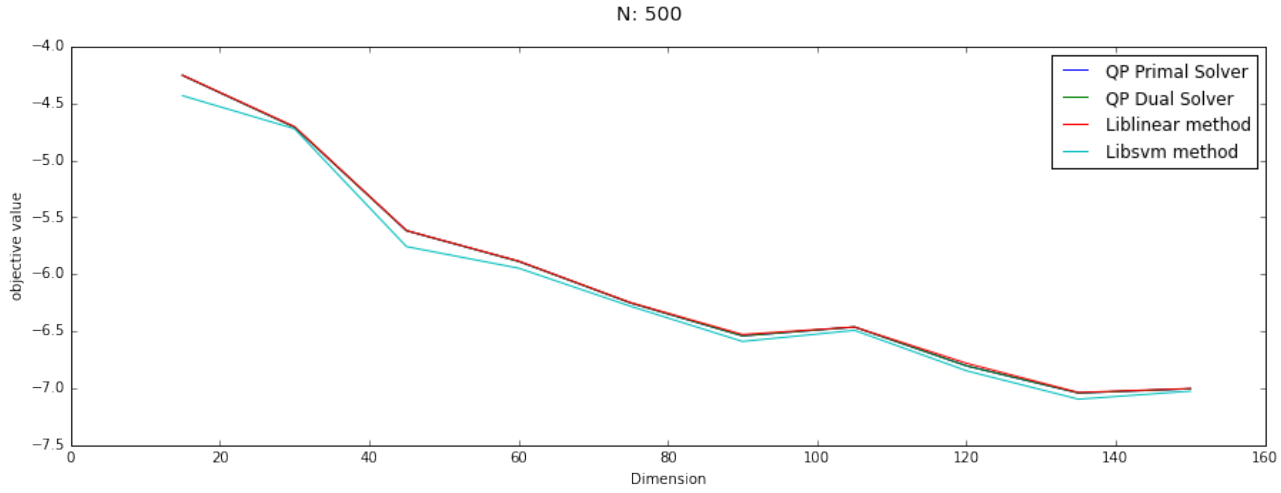
На них можно заметить, что в целом значения для всех методов похожи (одинаковы с некоторой погрешностью).

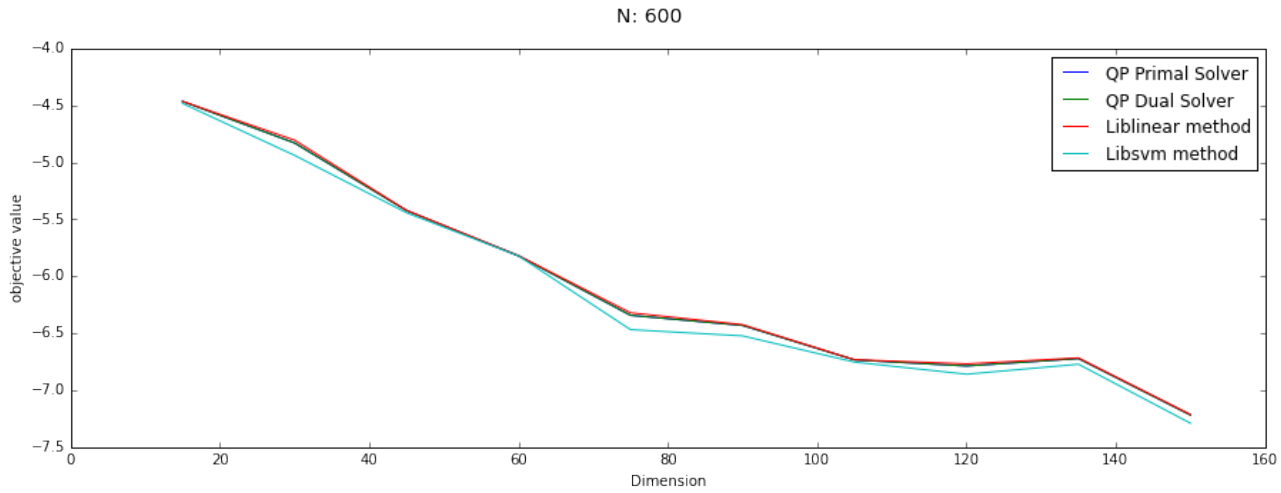


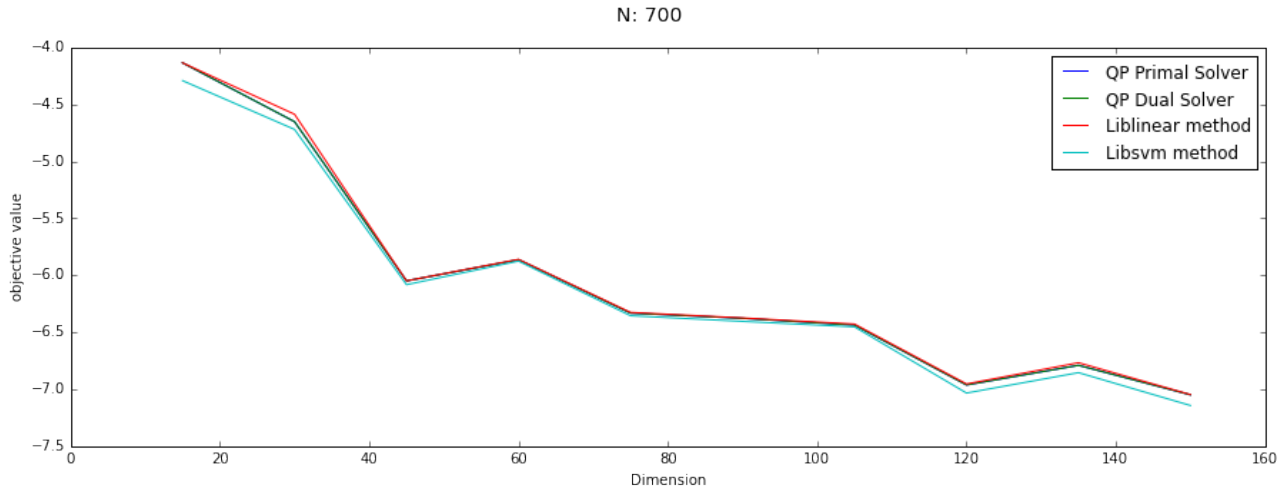


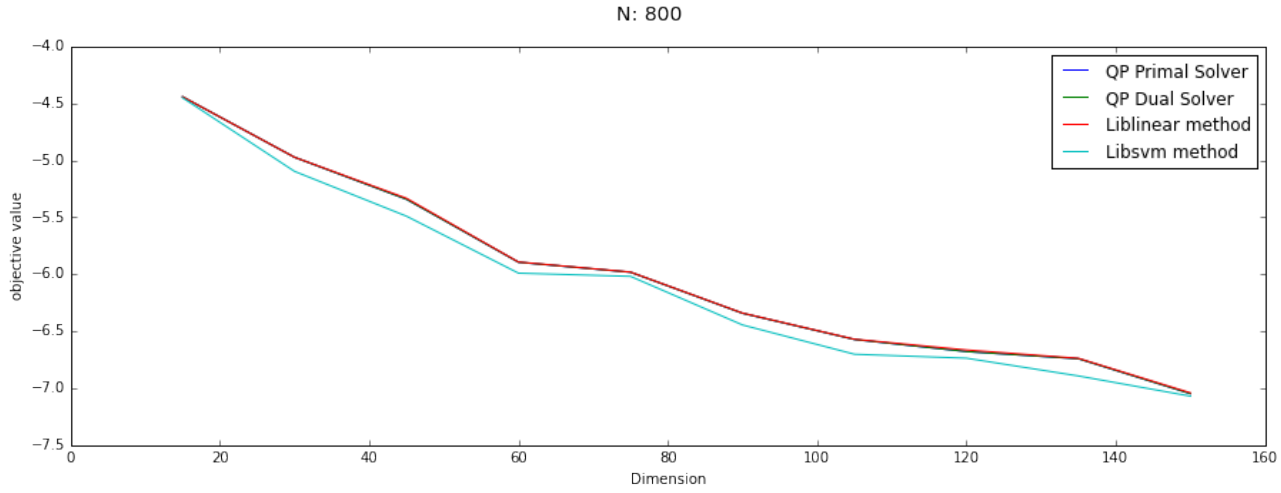

N: 300
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−3.5
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160

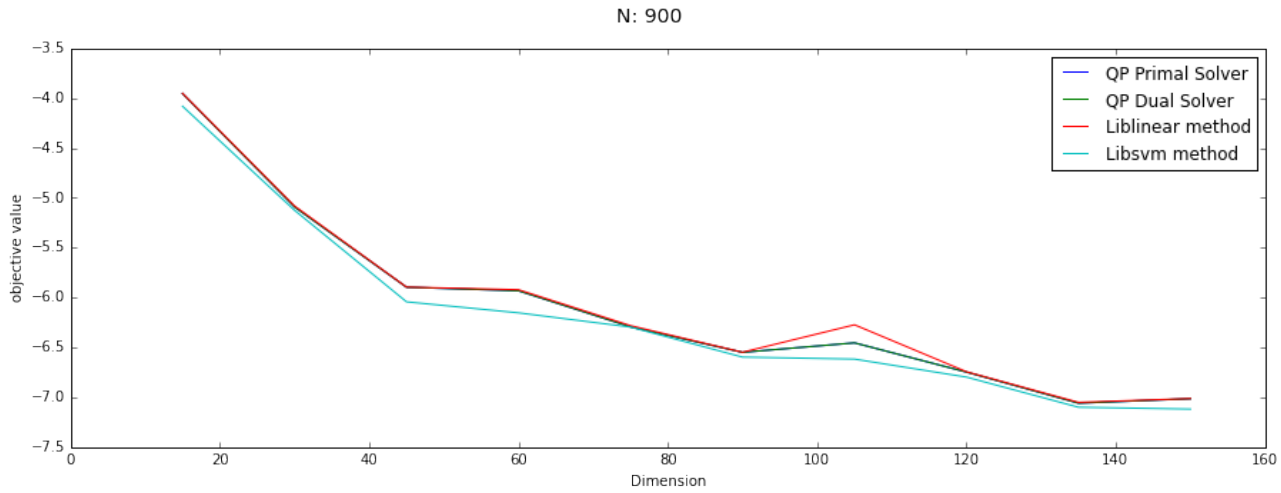

N: 400
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160

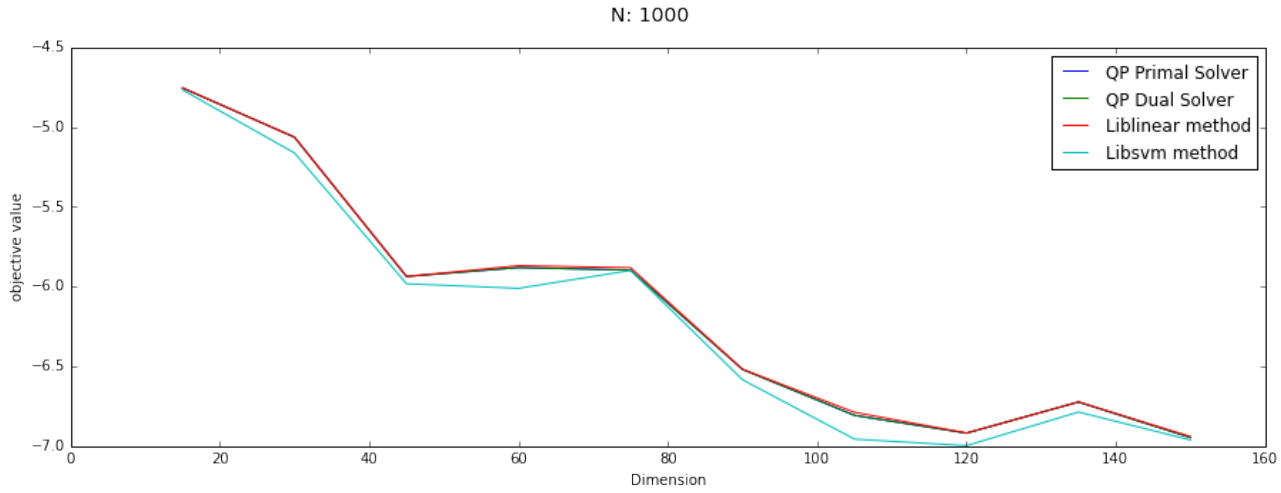

N: 500
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160


N: 600
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160


N: 700
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160


N: 800
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160

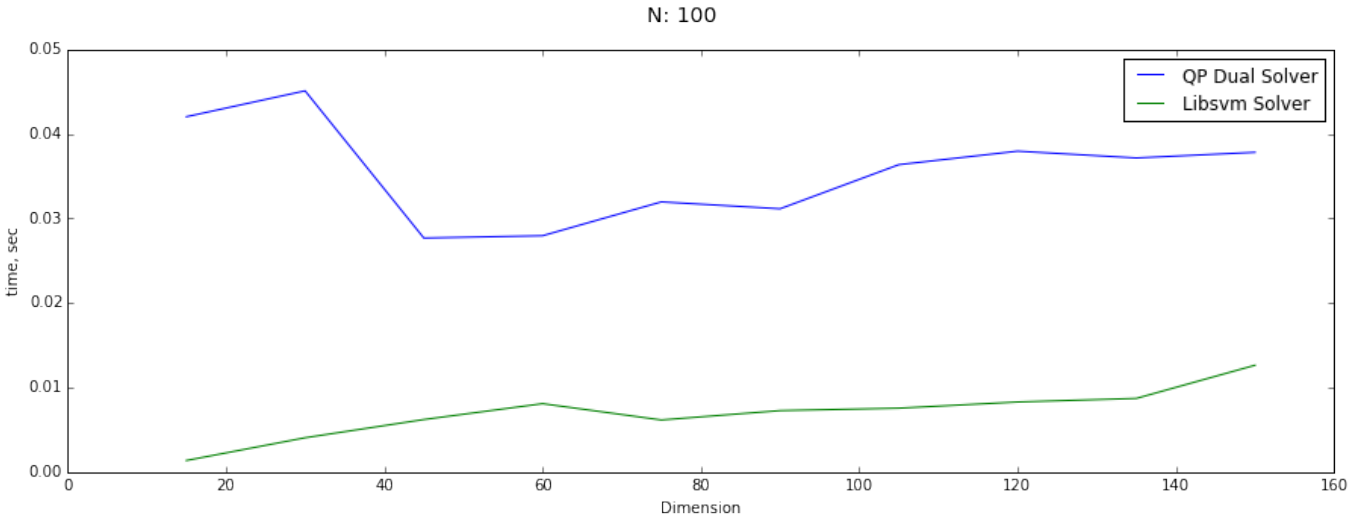

N: 900
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−3.5
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
0
20
40
60
80
100
120
140
160

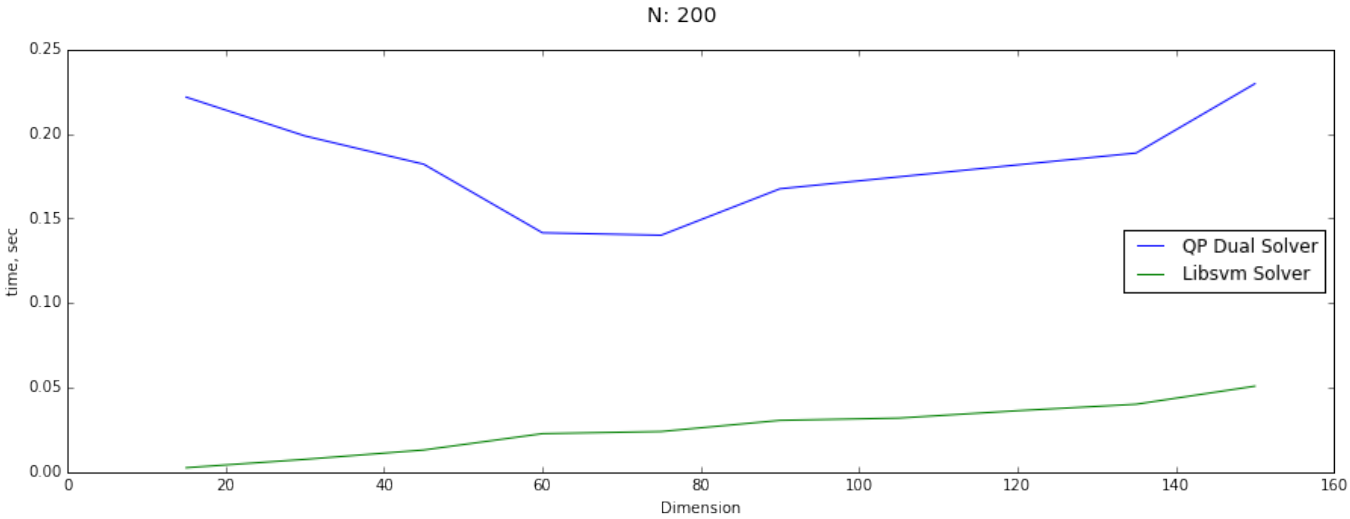

N: 1000
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
−4.5
−5.0
−5.5
−6.0
−6.5
−7.0
0
20
40
60
80
100
120
140
160

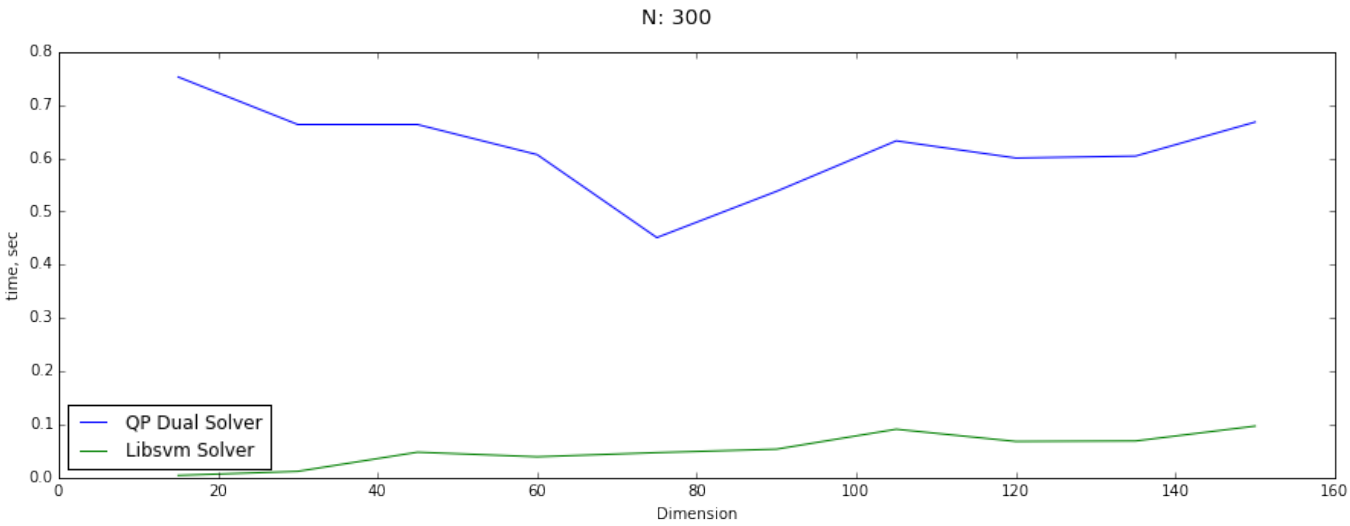
2.3 Пункт 2

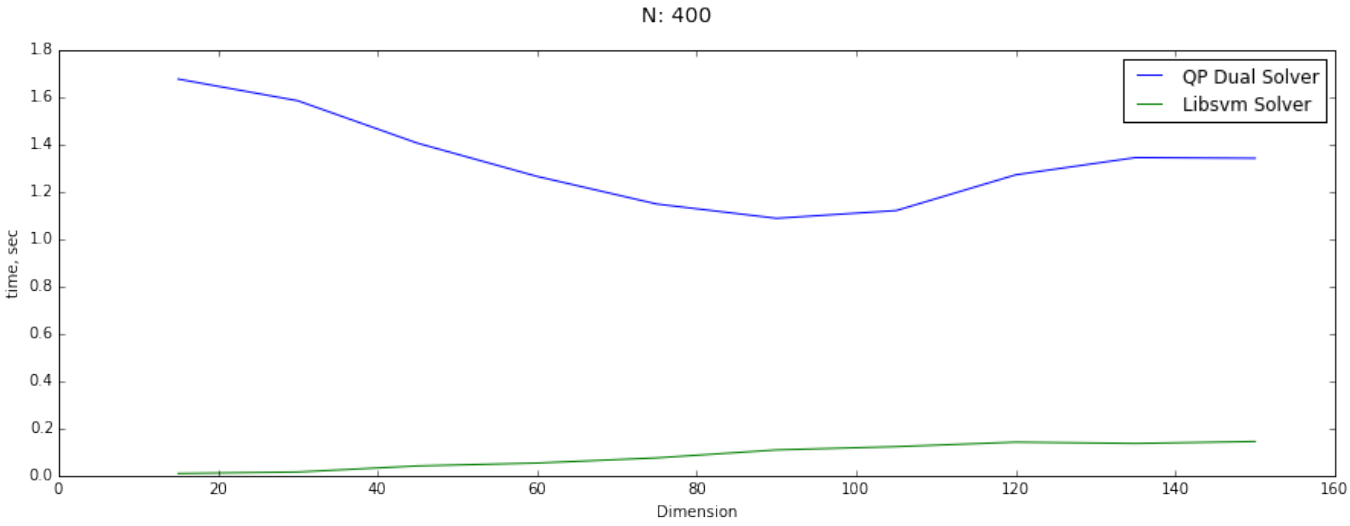
На следующих графиках представлены зависимости времени от размерности пространства для каждого размера выборки в методах с RBF ядром.

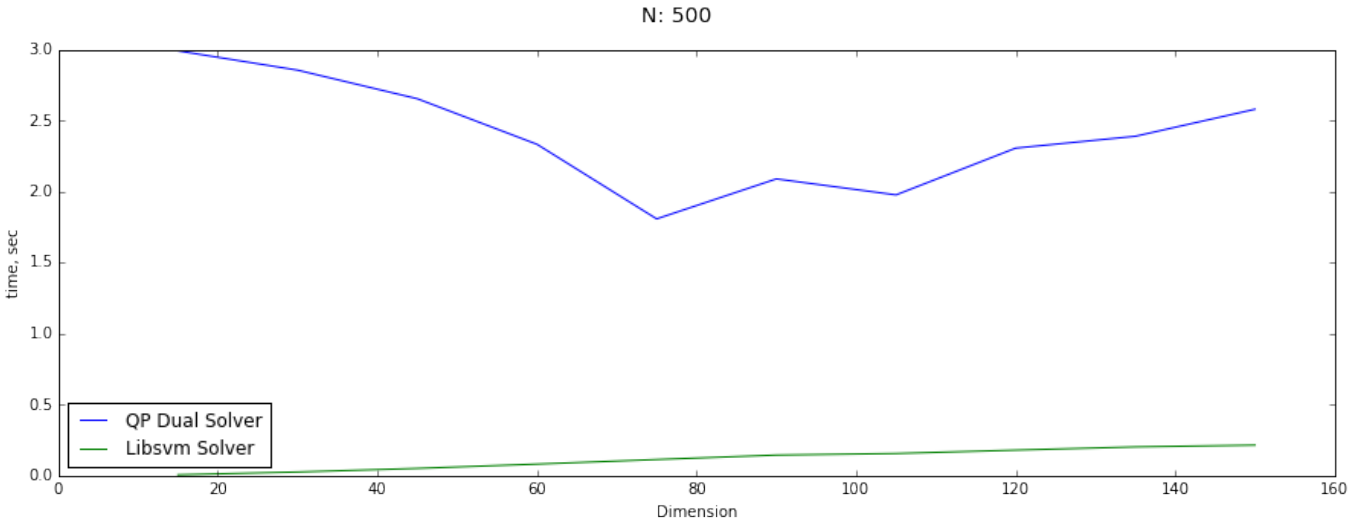
По времени работы быстрым является метод libsvm

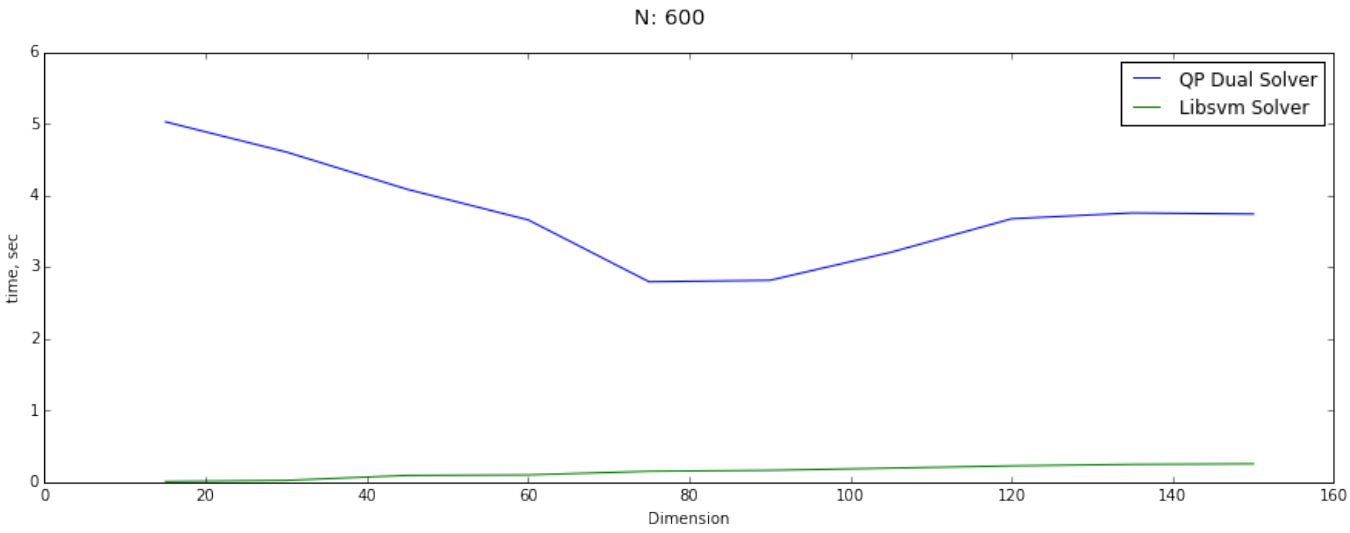


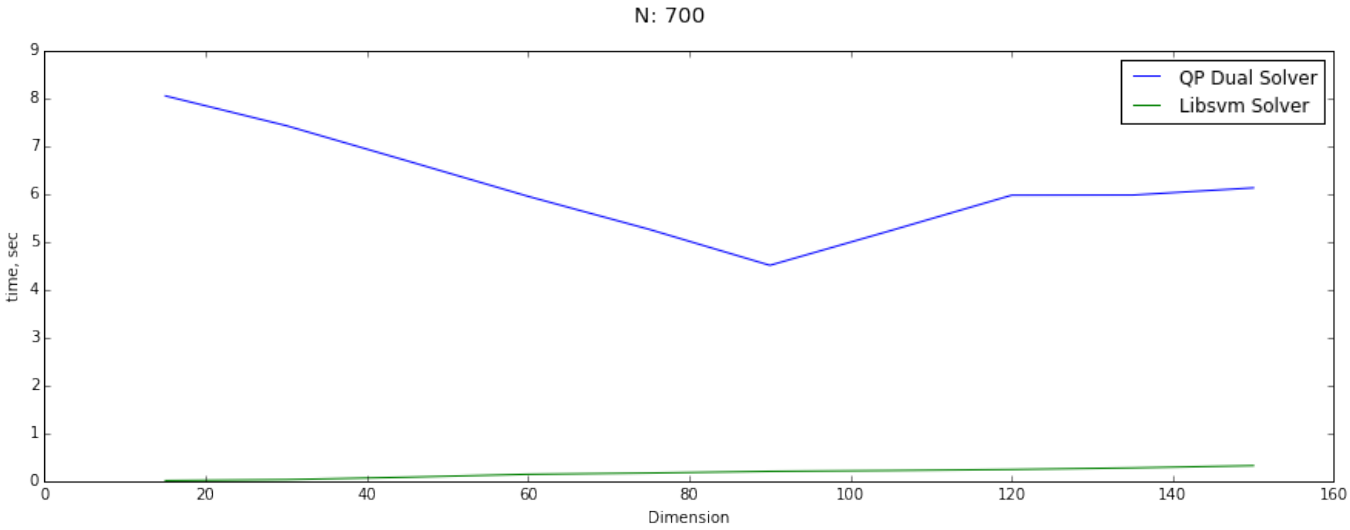


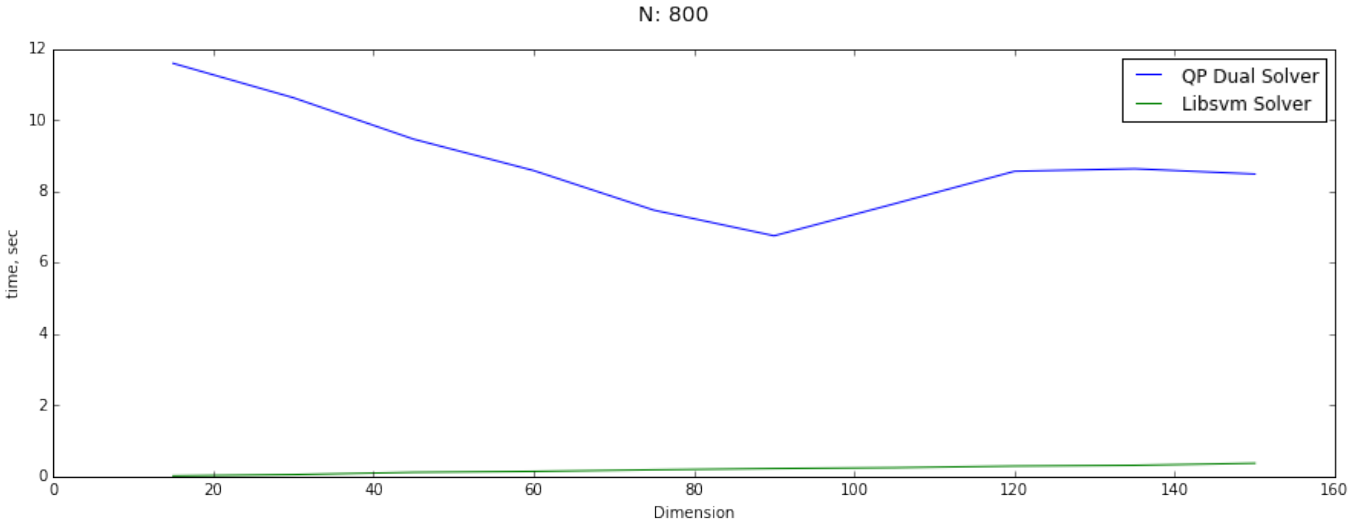

N: 300
time, sec
Dimension
QP Dual Solver
Libsvm Solver

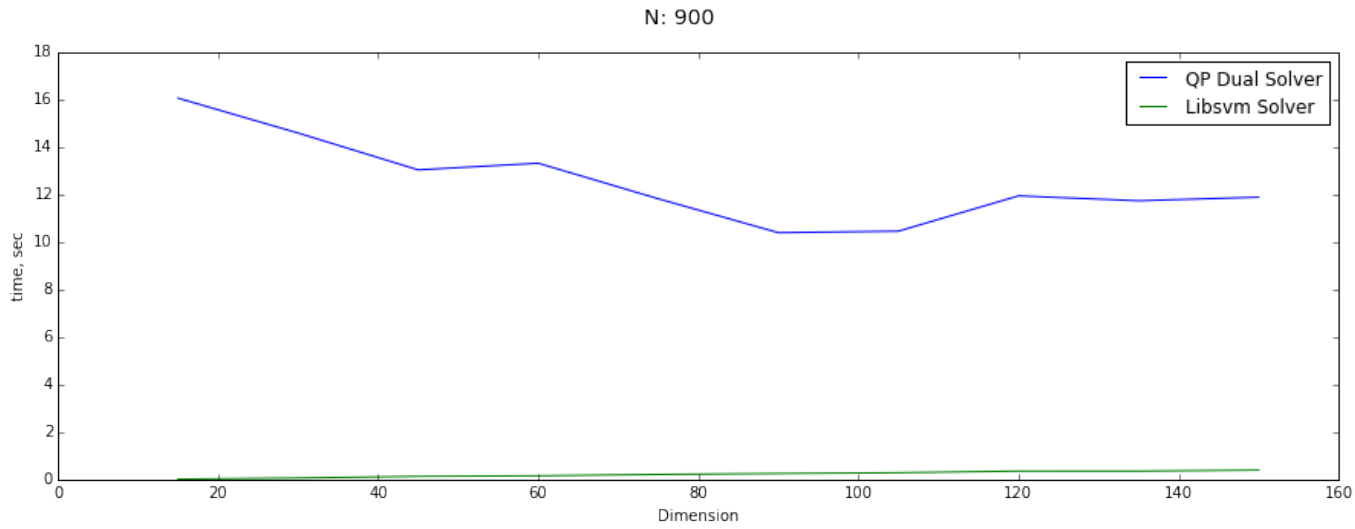

N: 400
time, sec
Dimension
QP Dual Solver
Libsvm Solver

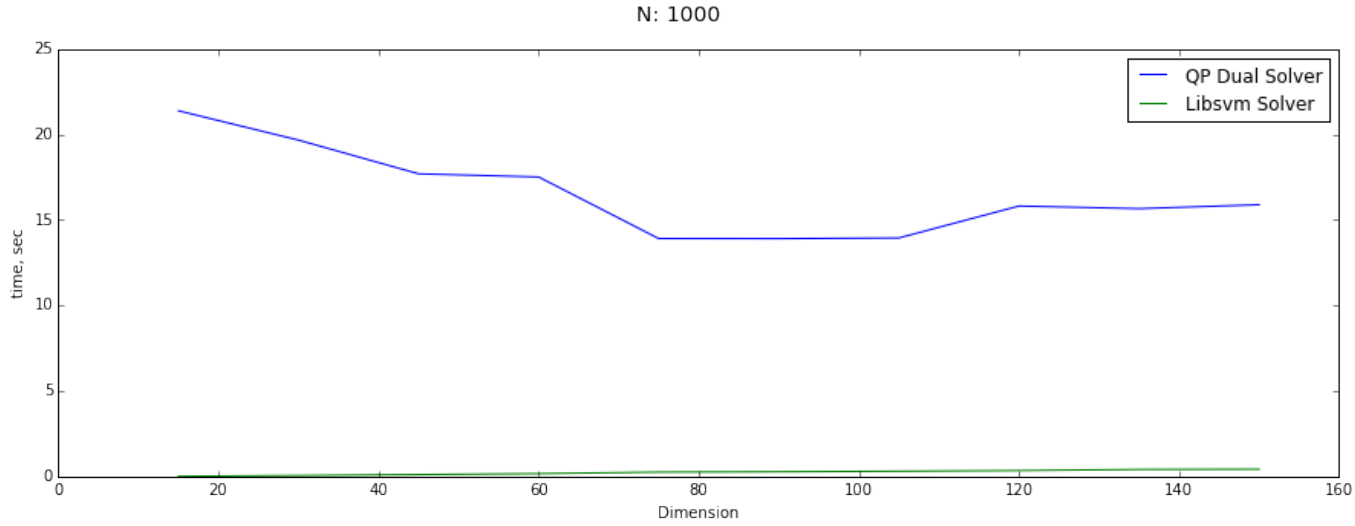

N: 500
time, sec
Dimension
QP Dual Solver
Libsvm Solver


N: 600
QP Dual Solver
Libsvm Solver
time, sec
Dimension
0
1
2
3
4
5
6
0
20
40
60
80
100
120
140
160

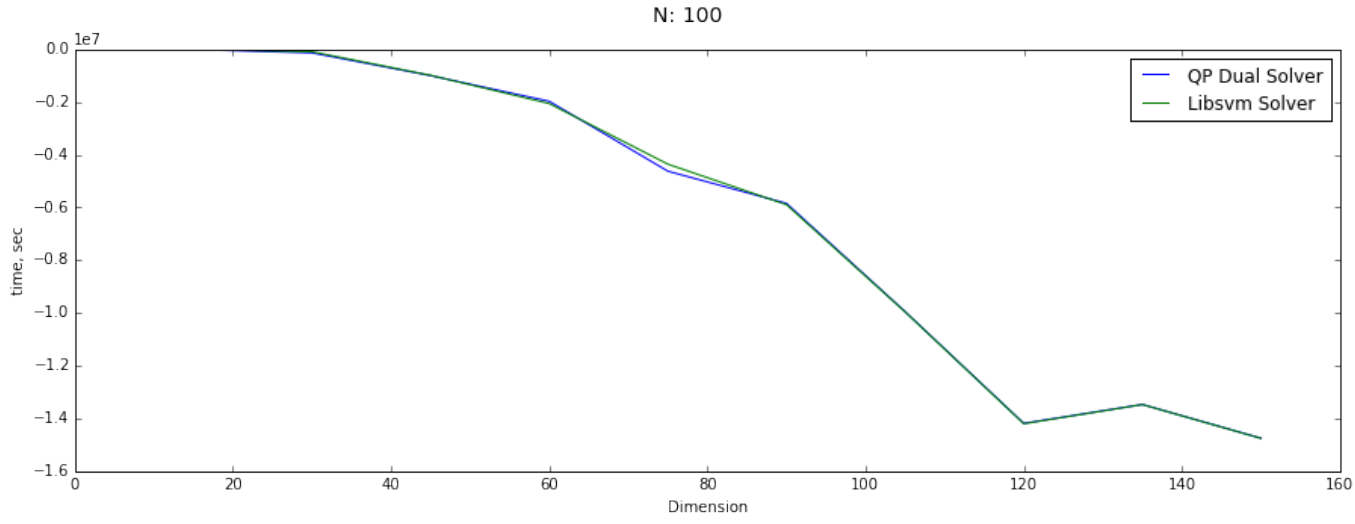

N: 700
QP Dual Solver
Libsvm Solver
time, sec
Dimension
0
1
2
3
4
5
6
7
8
9
0
20
40
60
80
100
120
140
160

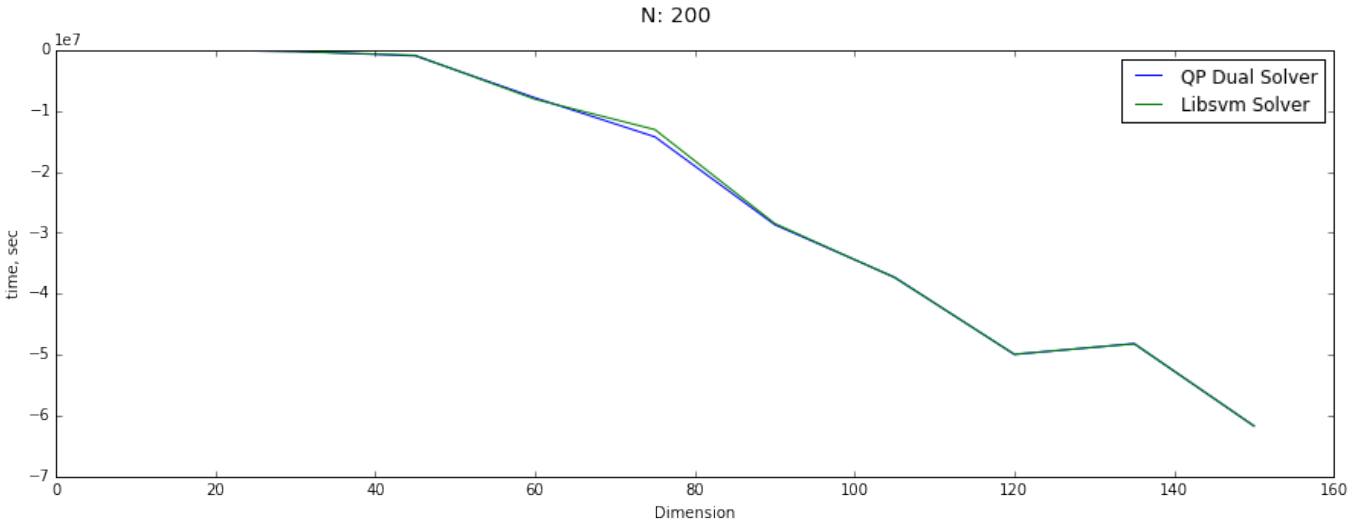

N: 800
QP Dual Solver
Libsvm Solver
time, sec
Dimension
0
2
4
6
8
10
12
0
20
40
60
80
100
120
140
160

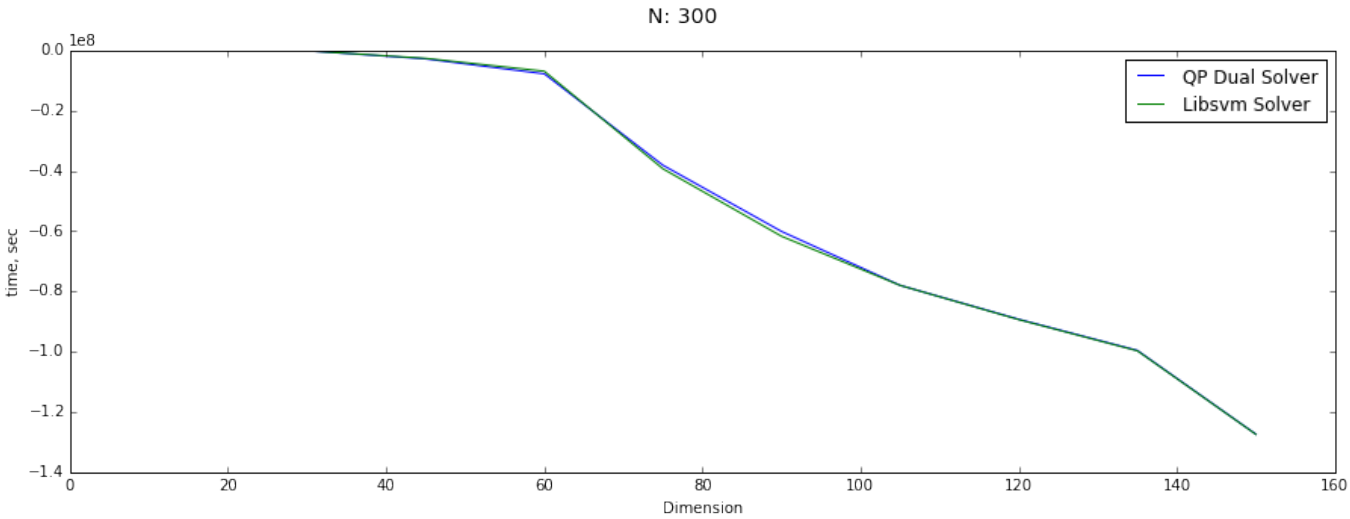


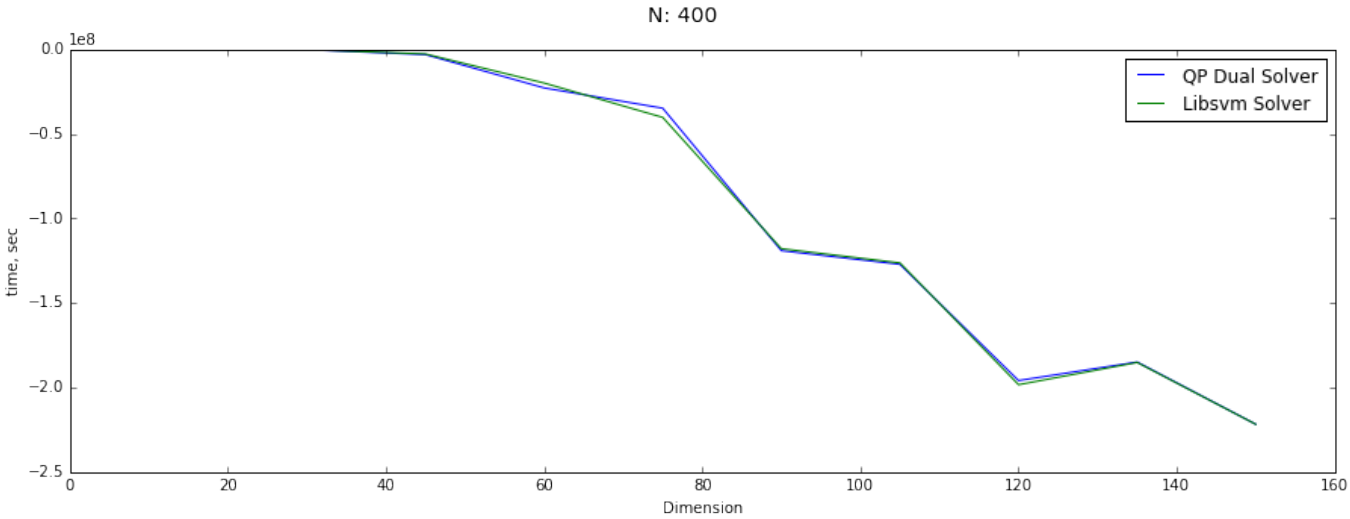


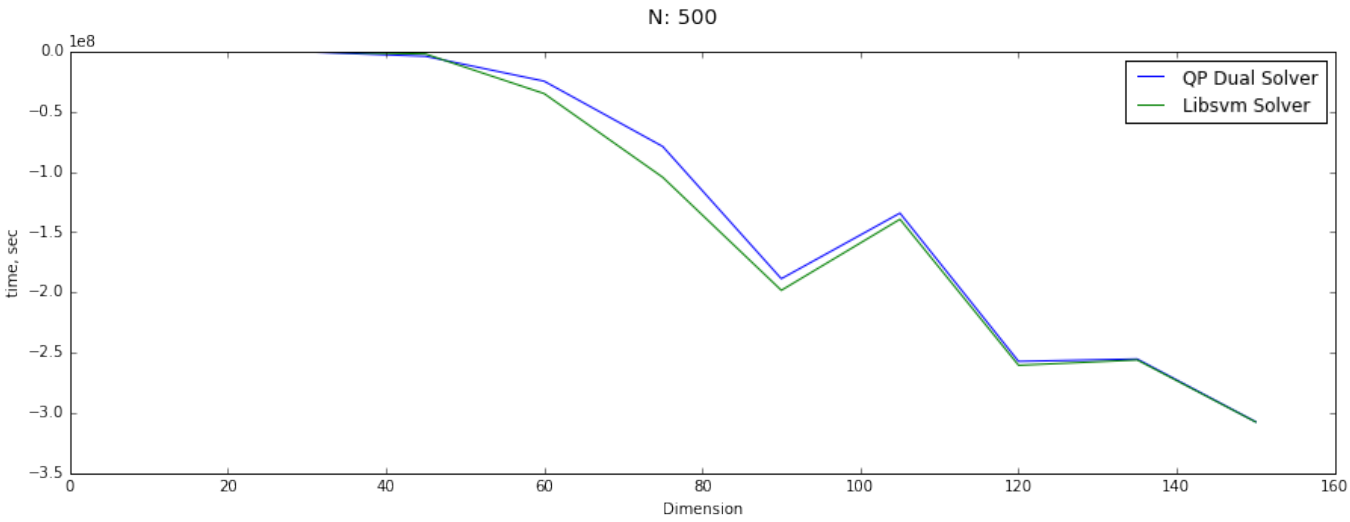
Далее представлены графики значений функционала для методов с RBF ядром. Для обоих методов они одинаковые с некоторой погрешностью при небольших выборках, но при больших выборках, значения начинают раличаться

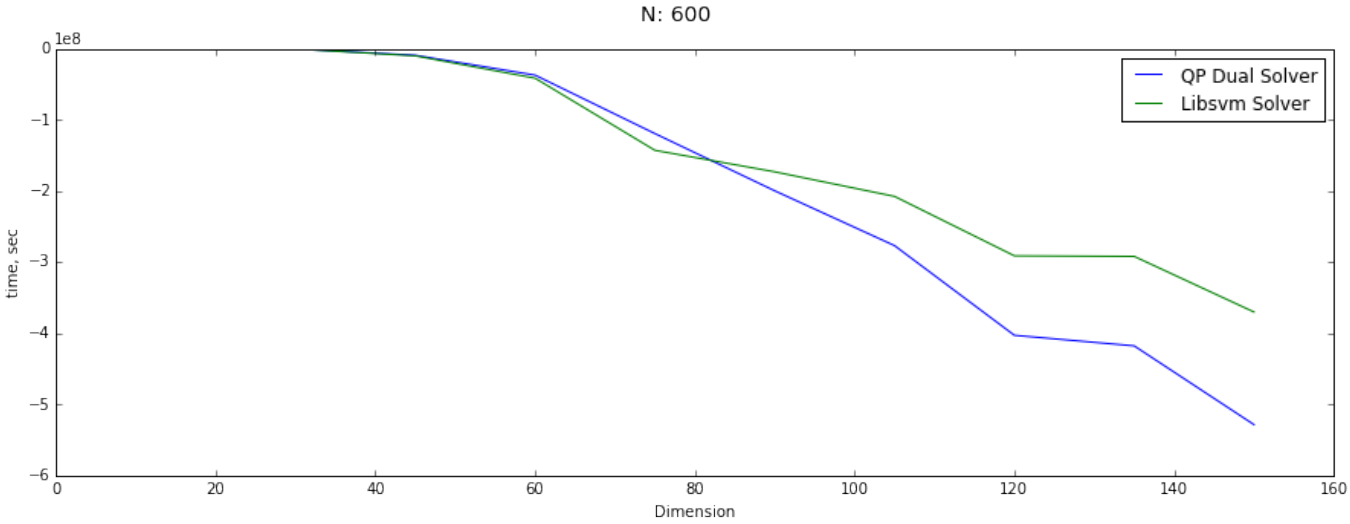


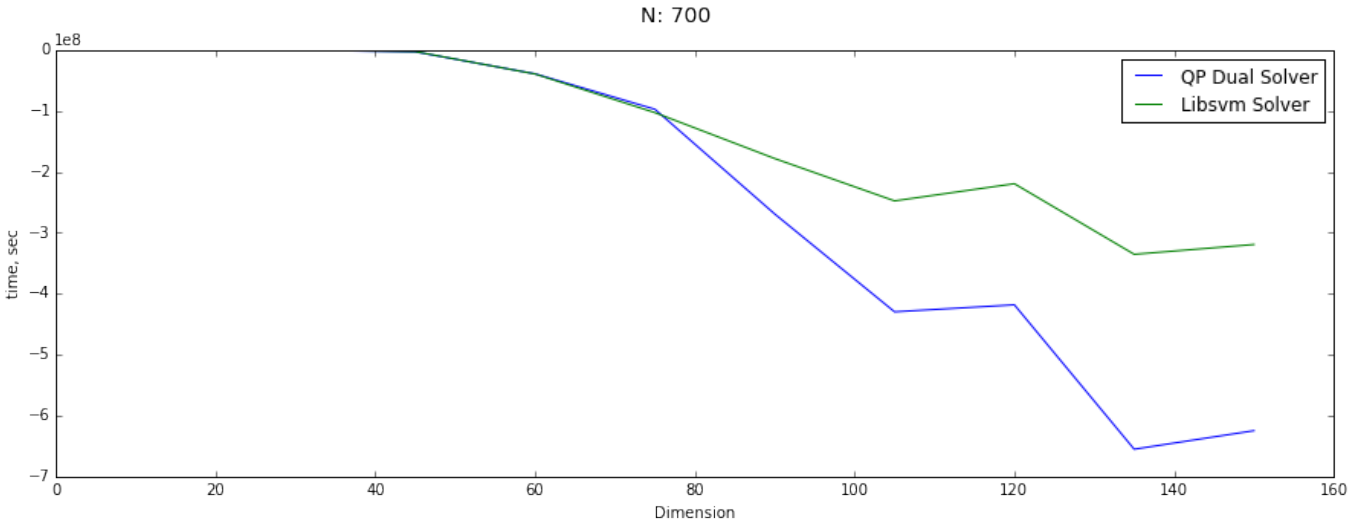

N: 200
1e7
0
−1
−2
−3
−4
−5
−6
−7
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver

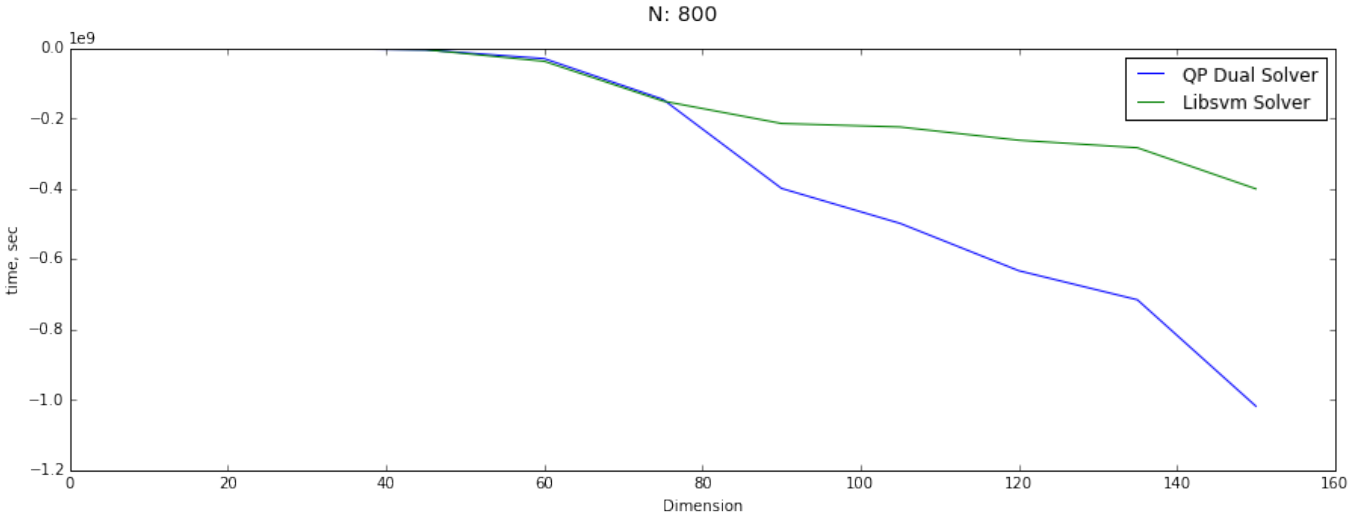

N: 300
1e8
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver

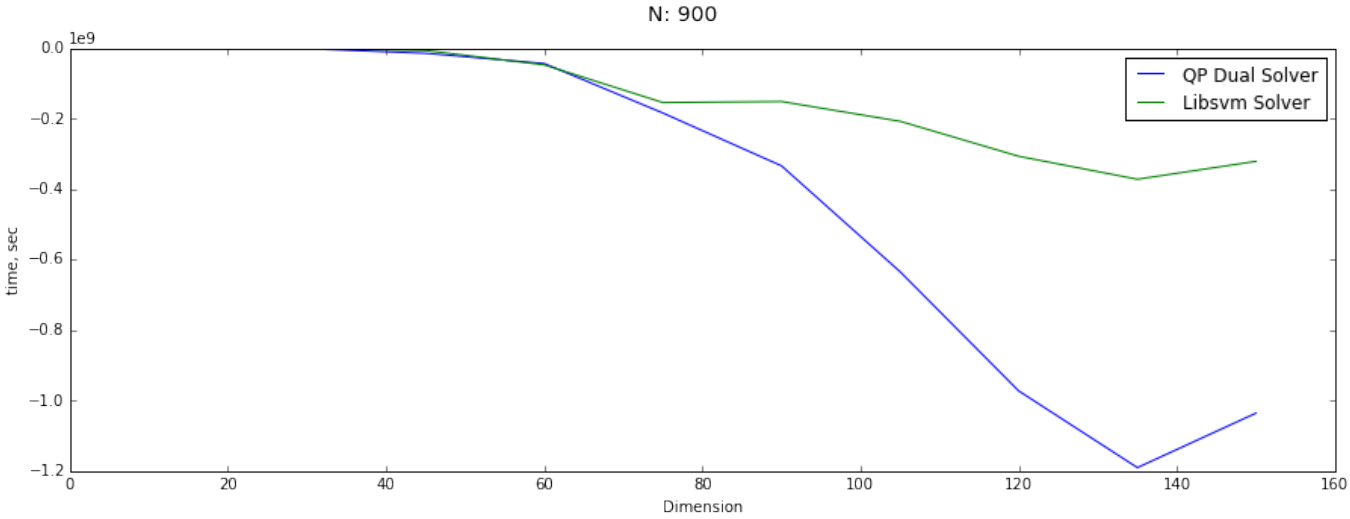

N: 400
1e8
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver

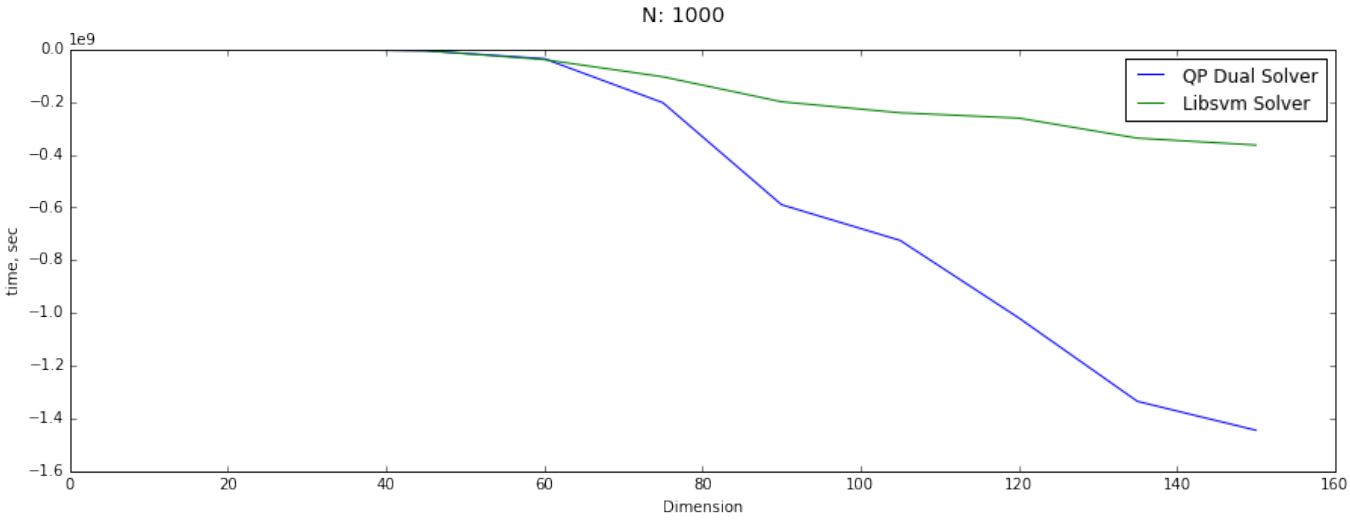

N: 500
1e8
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver


N: 600
1e8
0
−1
−2
−3
−4
−5
−6
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver


N: 700
1e8
0
−1
−2
−3
−4
−5
−6
−7
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver


N: 800
1e9
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver


N: 900
1e9
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver

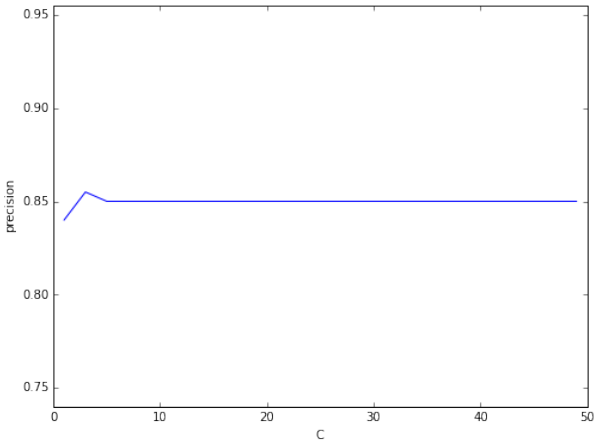

N: 1000
1e9
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−1.6
time, sec
0
20
40
60
80
100
120
140
160
Dimension
QP Dual Solver
Libsvm Solver

2.4 Пункт 3

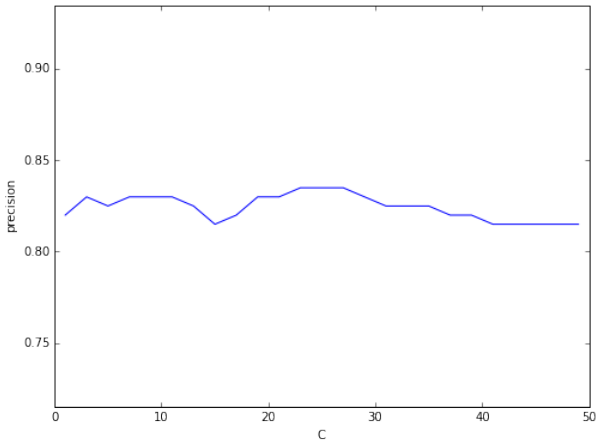
Кросс-валидация реализована с помощью библиотеки sklearn на методе libsvm.
Исследуем точность классификации в зависимости от параметра C

Плохо разделимая выборка

Линейное ядро

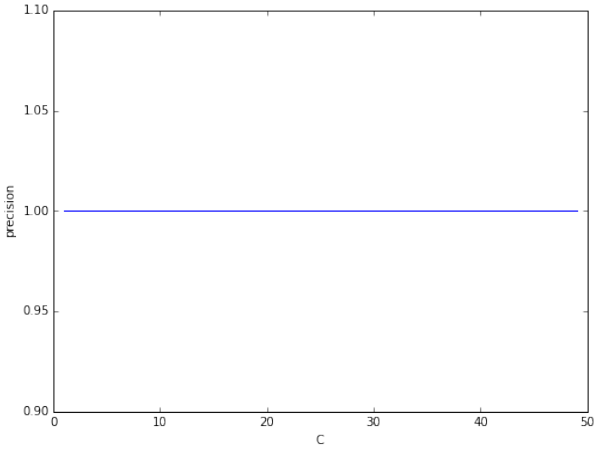


RBF ядро

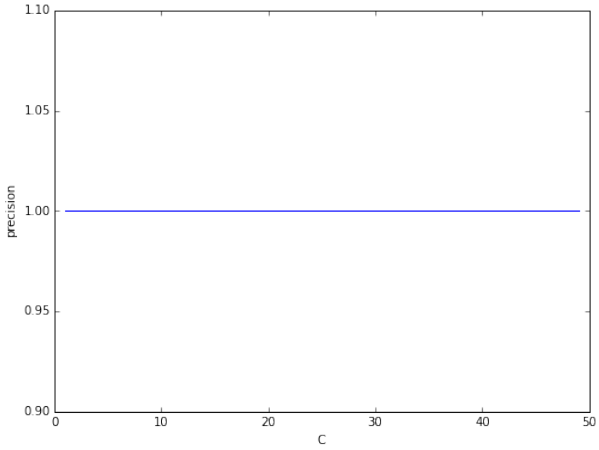


Хорошо разделимая выборка

Линейное ядро

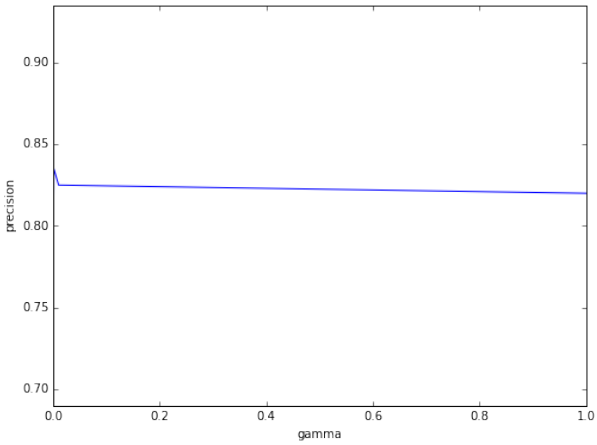


RBF ядро

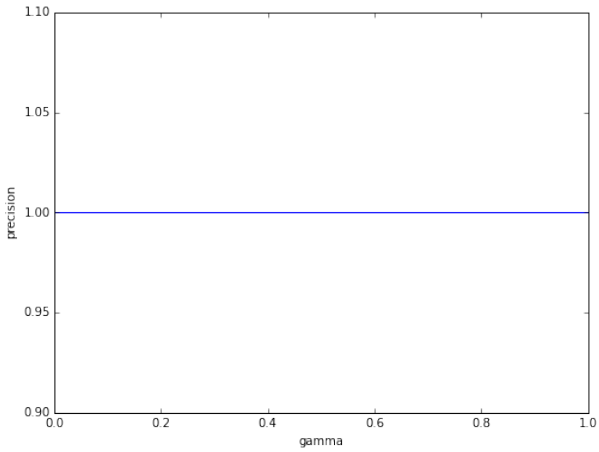


Исследуем точность классификации в зависимости от параметра γ

Плохо разделимая выборка



Хорошо разделимая выборка

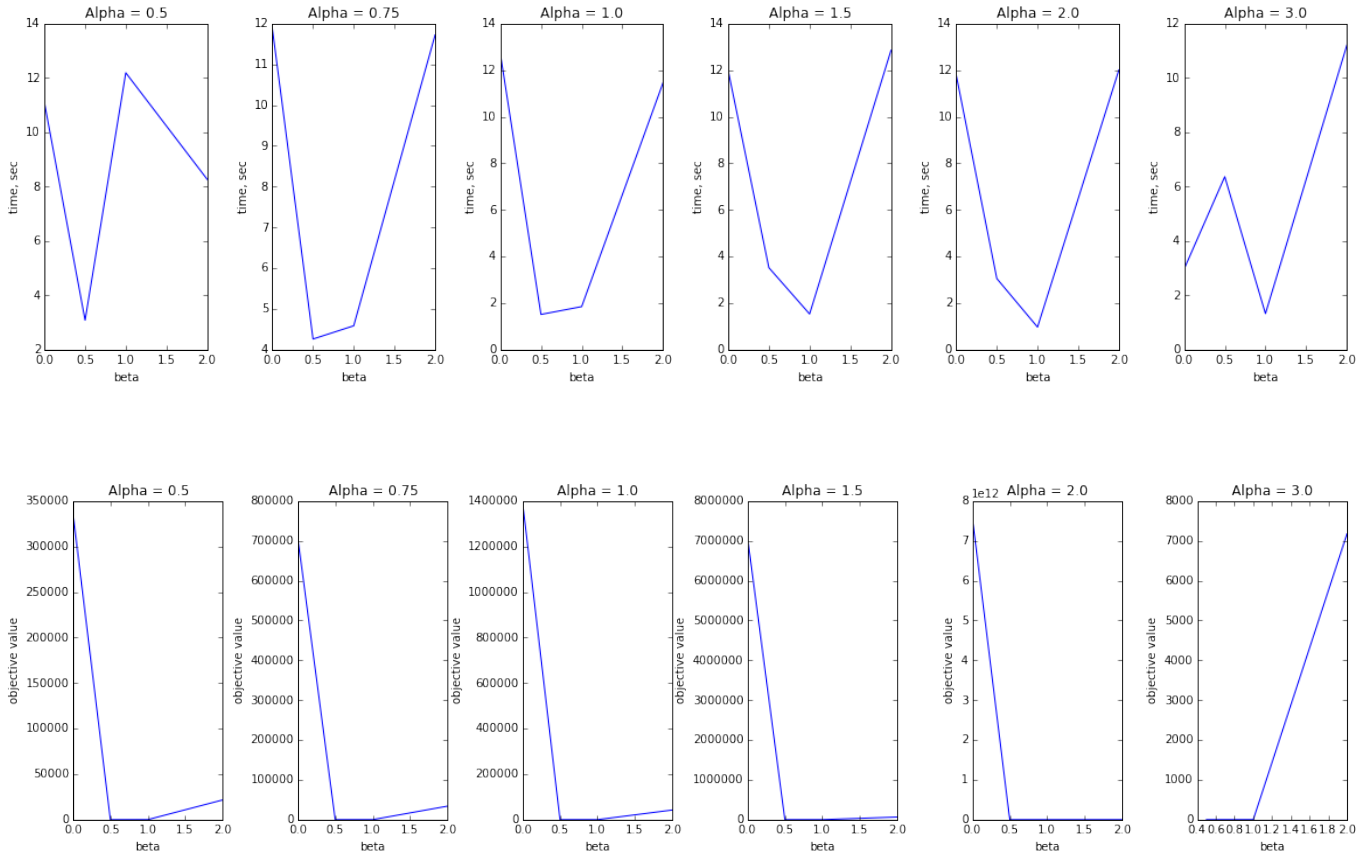


В плохо разделимой выборке видим, что максимальная точность в случае лмнейного ядра достигается при C = 3, а в случае RBF ядра максимум достигается при C = 25. А точность в зависимости от γ убывает. Т.е. классификатор более точен при гамма близком к нулю.

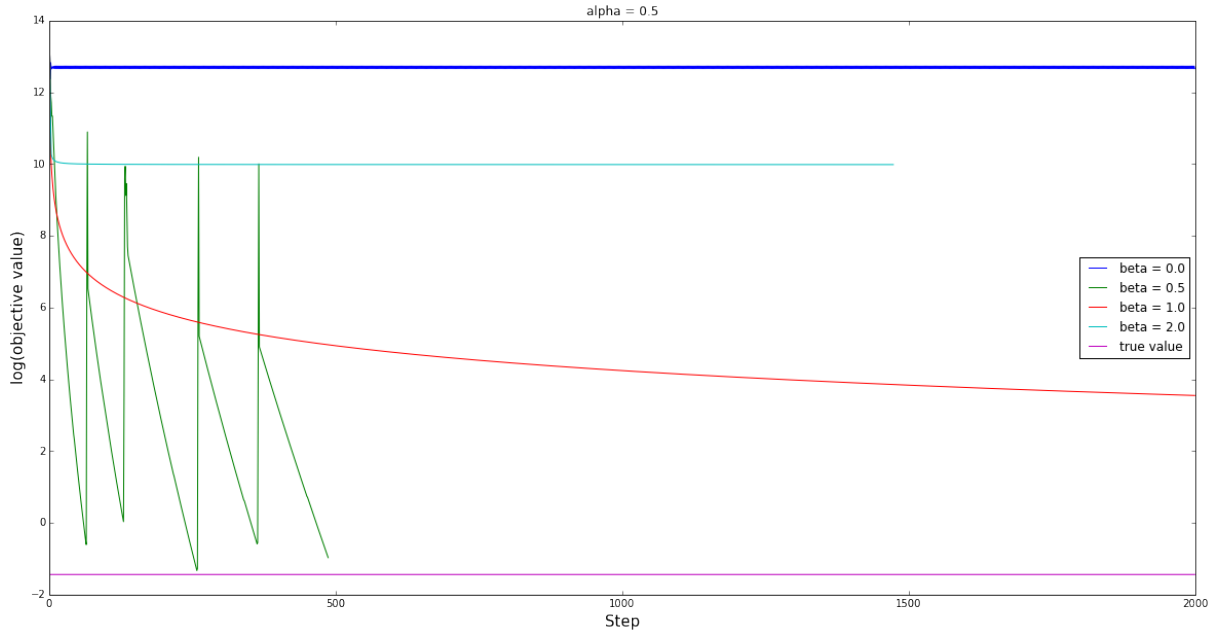
В хорошо разделимой выборке для любого C и в случае линейного ядра, и в случае RBF ядра, точность остается постоянной и равной 1. А точность от гамма также не зависит.

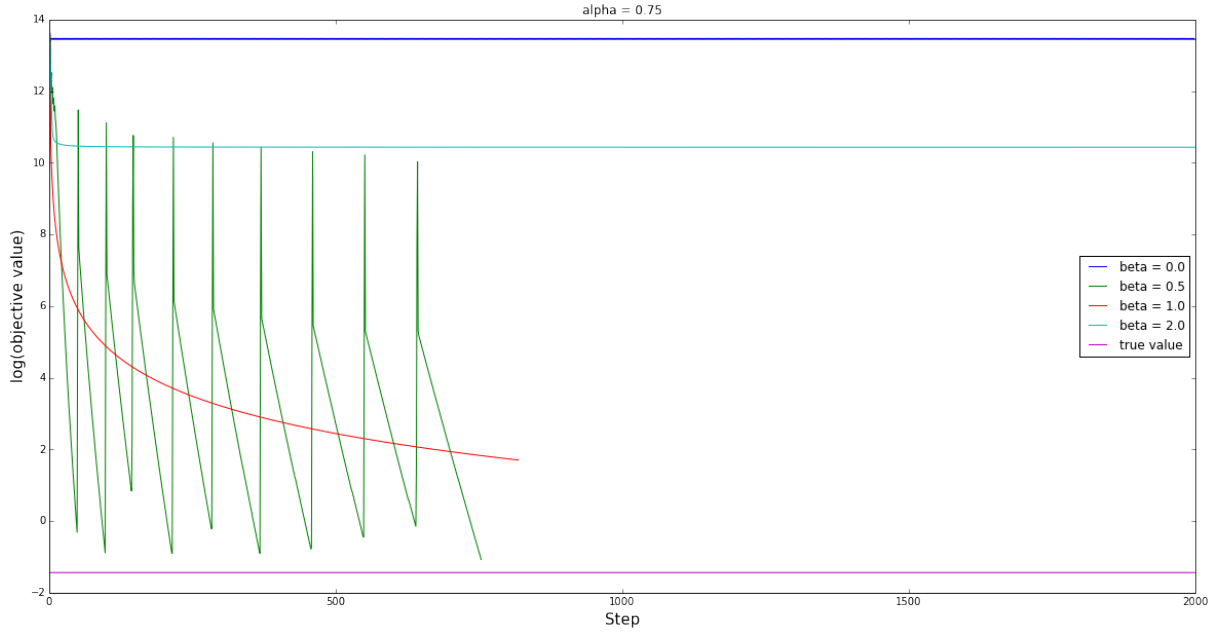
2.5 Пункт 4

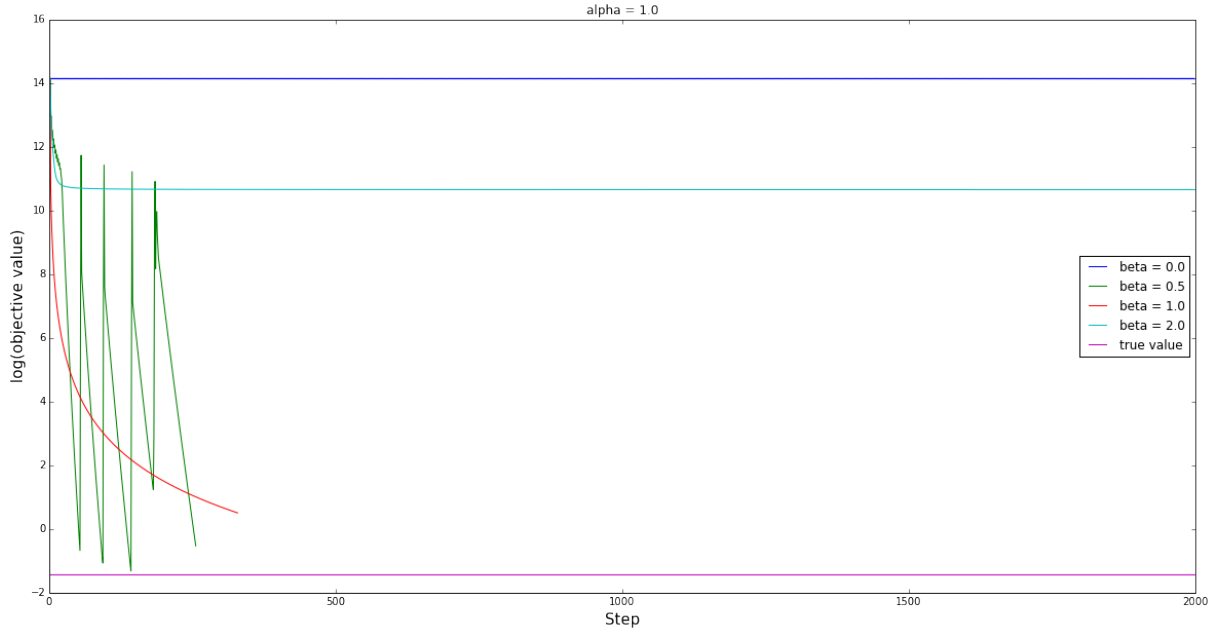
На следующих графиках показаны зависимости времени работы и значений функционала от β при разных α для метода субградиентного спуска. По ним можно сделать вывод, что при $\beta = 1$ и $\alpha = 2$ метод работает сравнительно медлоенно, но выдает более точное значение функционала

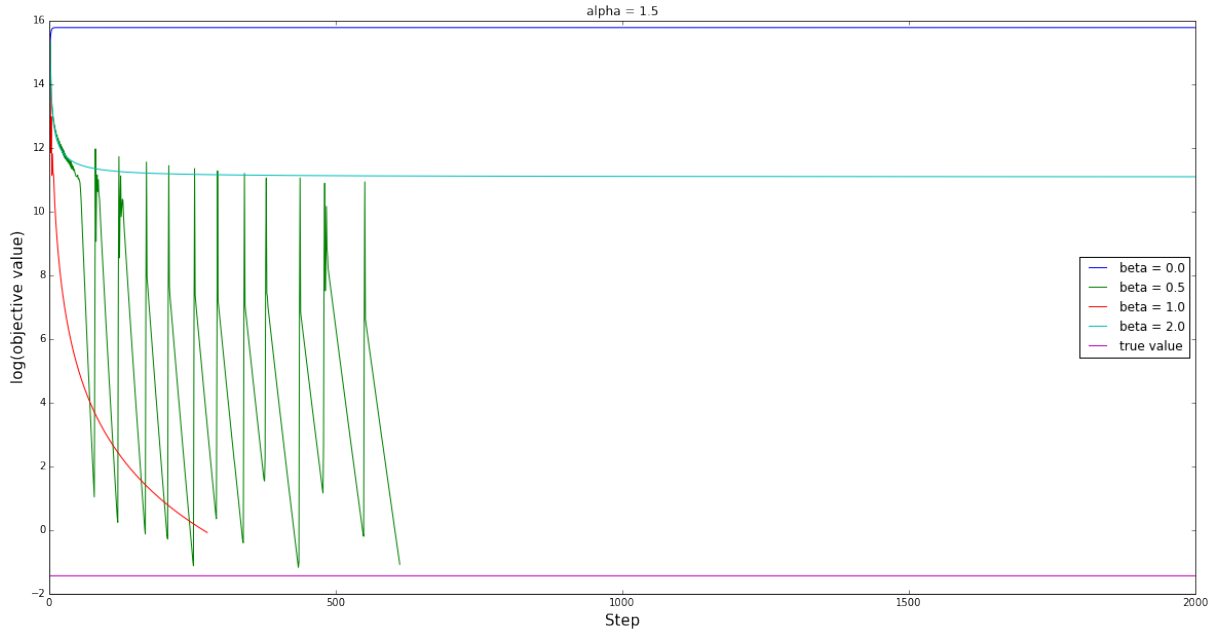


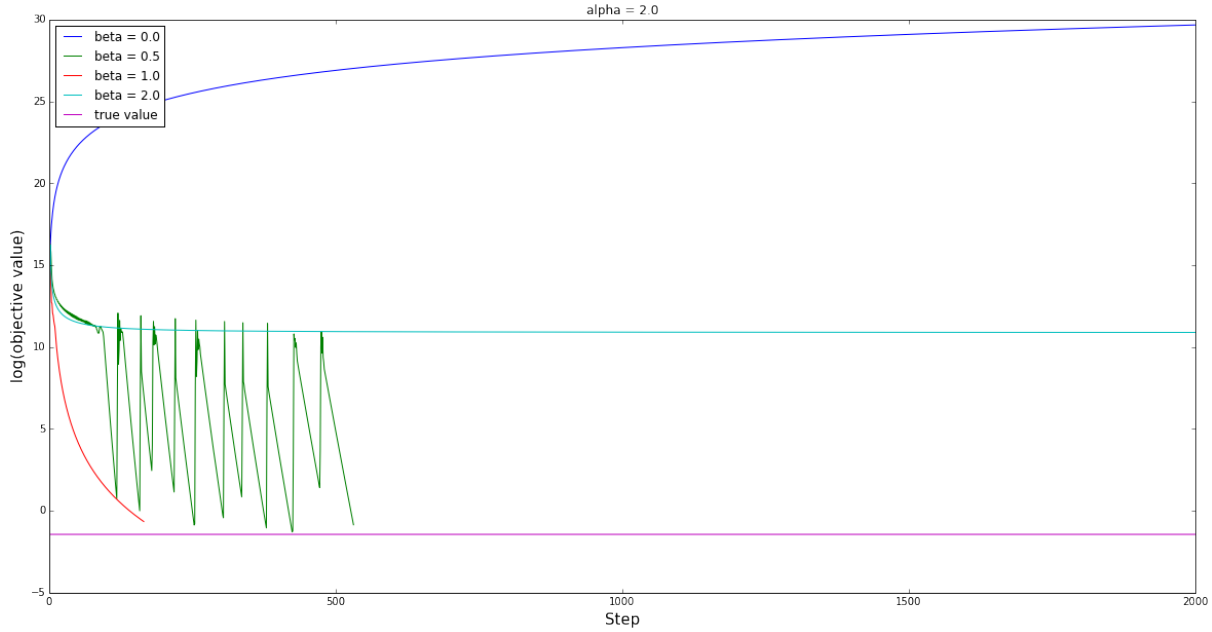
Далее приведем графики сходимости субградиентного спуска при разных значения α и β

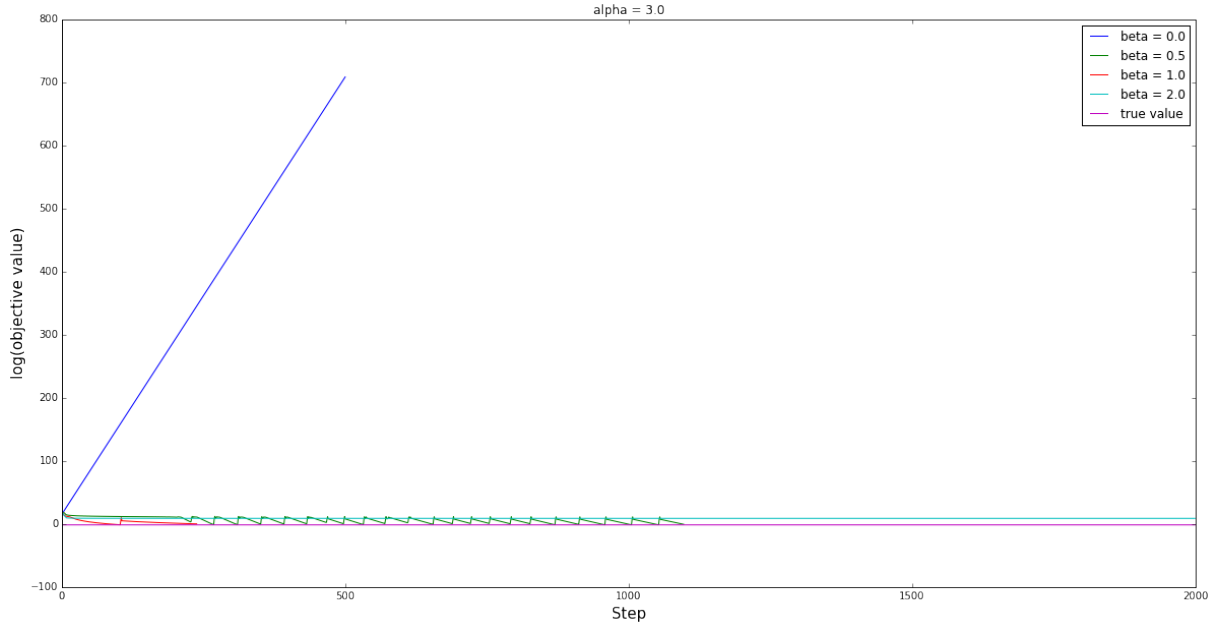



alpha = 0.75
log(objective value)
Step
beta = 0.0
beta = 0.5
beta = 1.0
beta = 2.0
true value


alpha = 1.0
log(objective value)
Step
beta = 0.0
beta = 0.5
beta = 1.0
beta = 2.0
true value


alpha = 1.5
log(objective value)
Step
beta = 0.0
beta = 0.5
beta = 1.0
beta = 2.0
true value

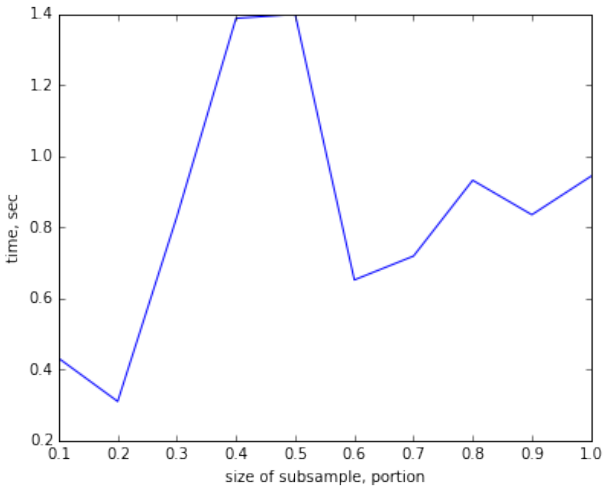


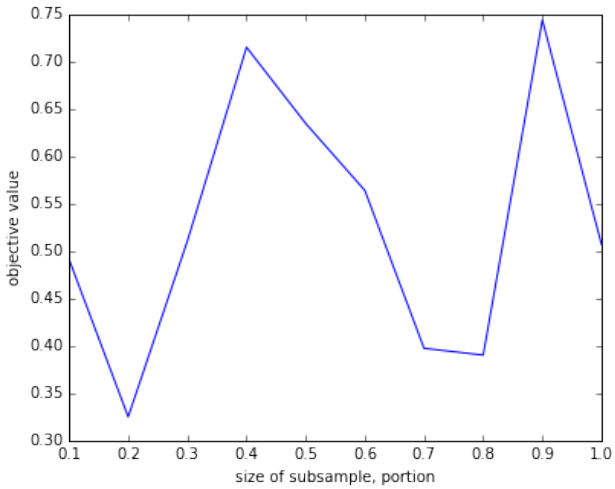


Из этих графиков тоже видно, что метод сходится быстро при $\beta = 1$ и $\alpha = 2$

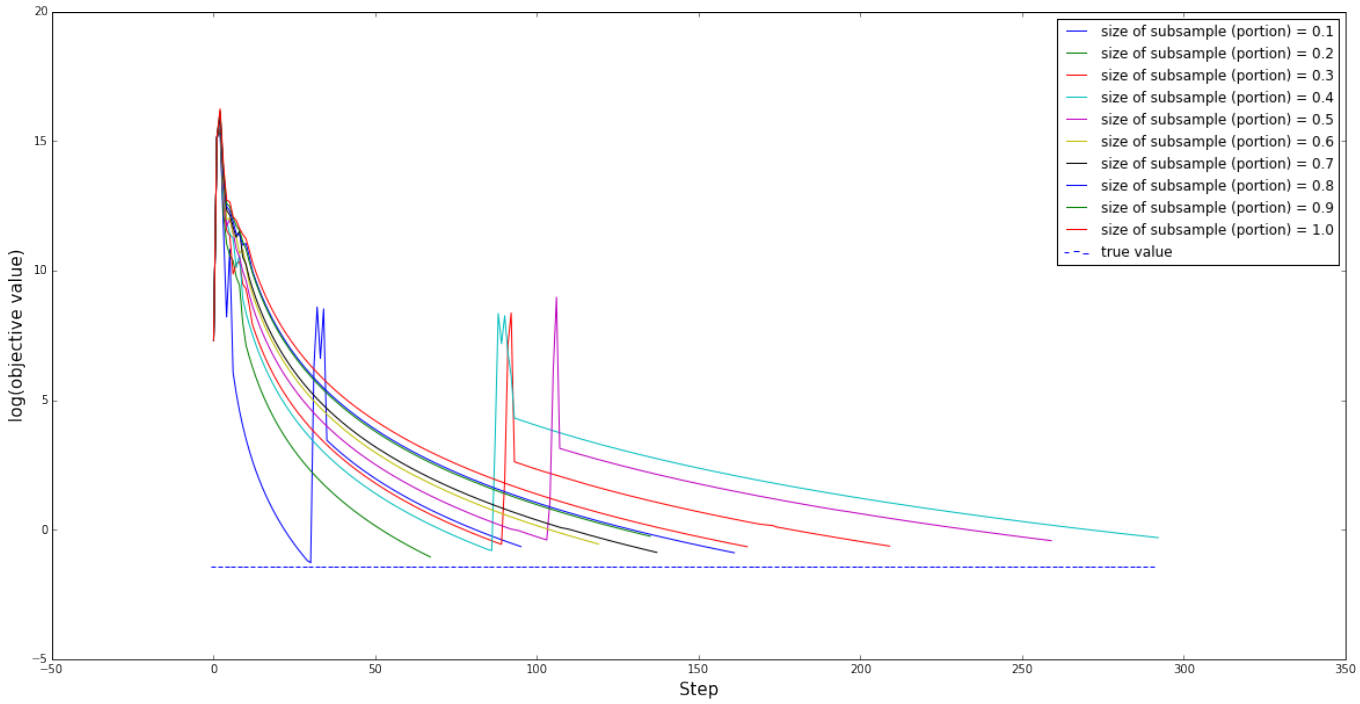
2.6 Пункт 5

На следующих графиках показаны зависимости времени работы и значений функционала от доли размера подвыборки в методе стохастического субградиентного спуска. На них видно, что более близкое к реальному значение функционала получается при размере 90% от размера выборки. Время работы сравнительно среднее при данном размере подвыборки.





На следующем графике показаны скорости сходимости метода в зависимости от доли размера подвыборки

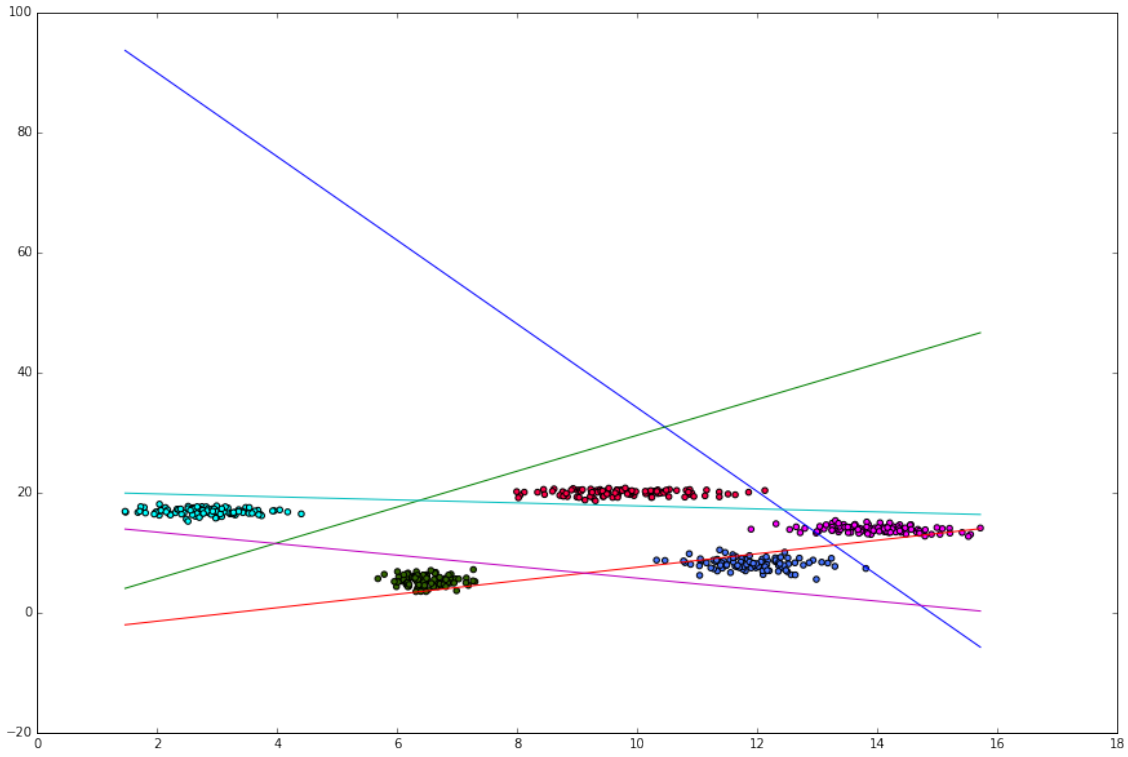


Из него также видно, что близкое к реальному значение функционала получается при 90% от размера выборки, а также видно, что при этом значении размера, метод сходится за меньшее число шагов.

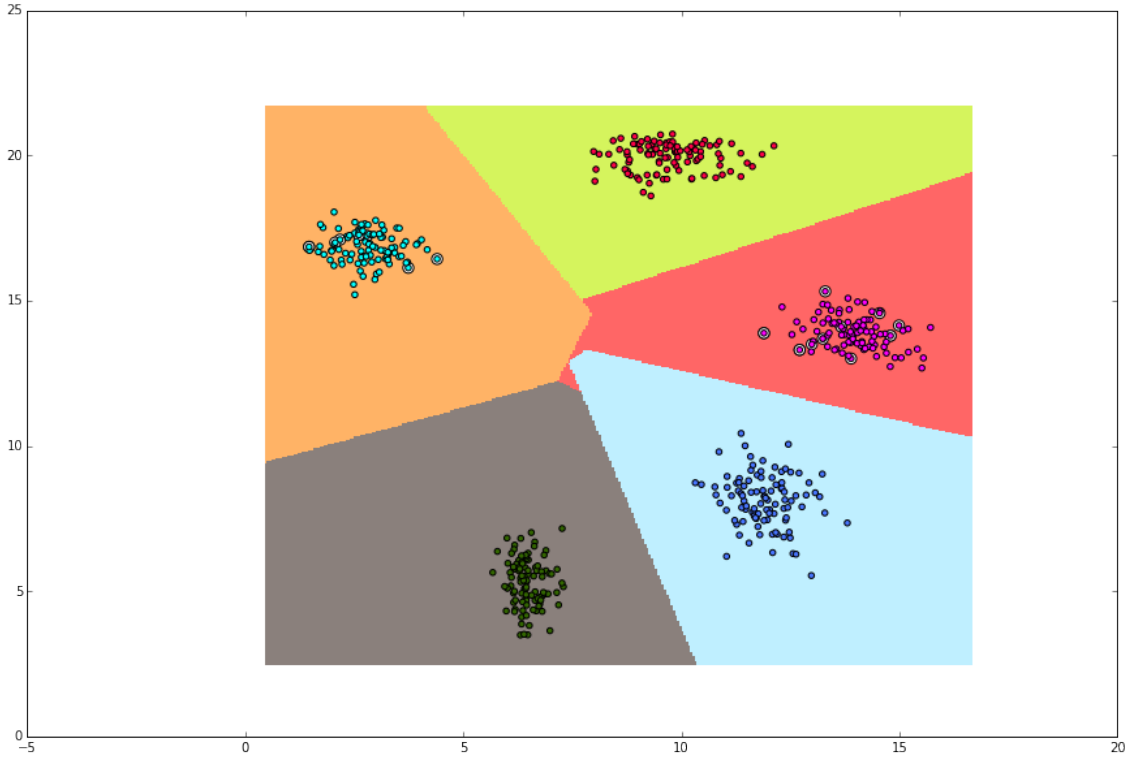
2.7 Пункт 6

Сравним методы OneVsOne и OneVsAll

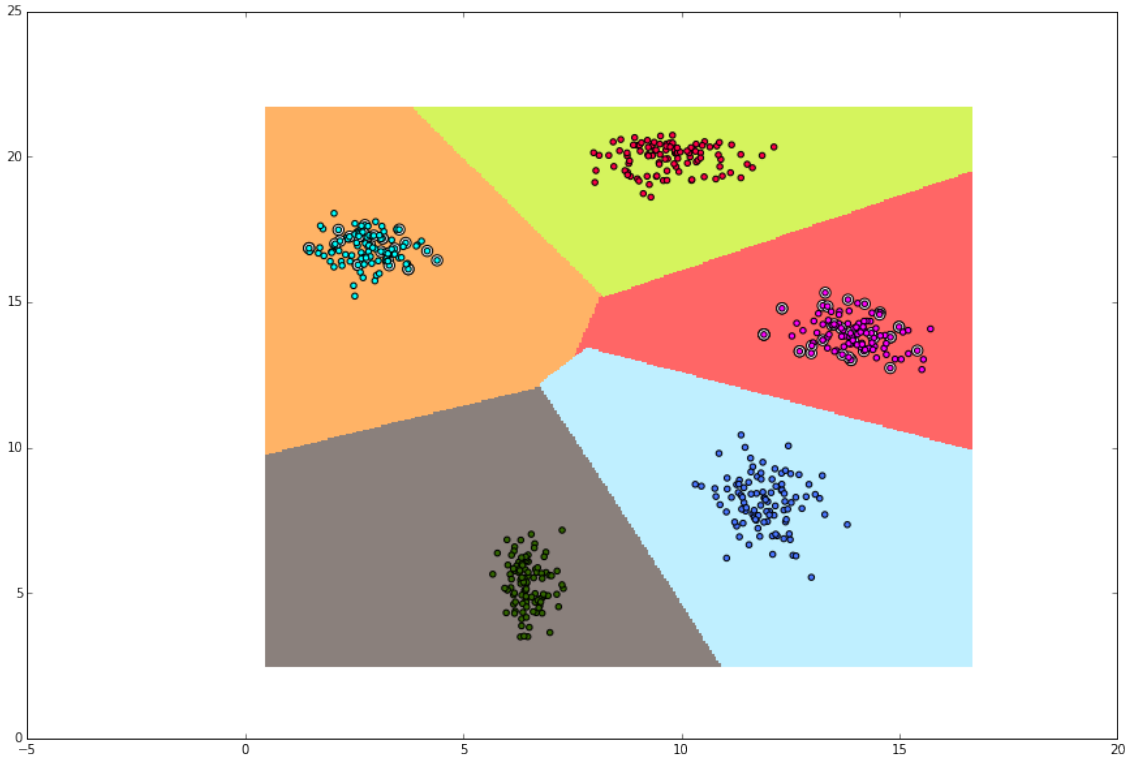
На первом графике представлен метод OneVsAll на примере liblinear. Видно, что разделяющие поверхности не точно разделяют выборки между собой, хотя данная выборка является линейно разделимой



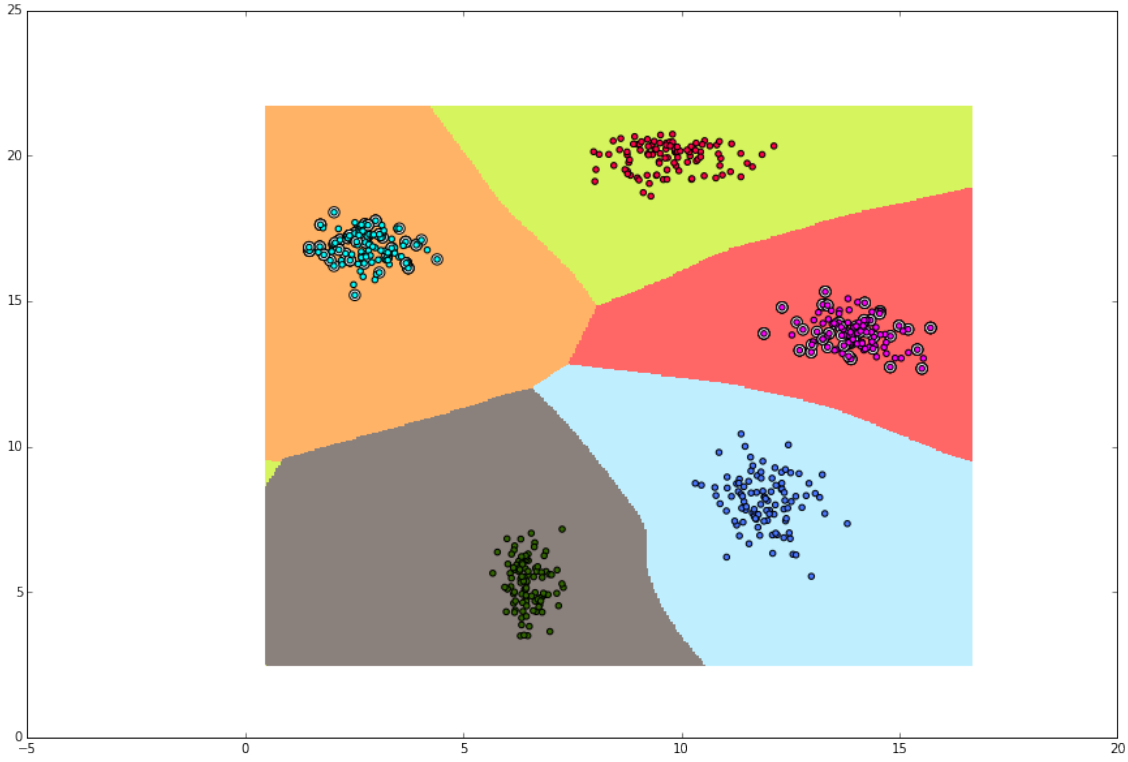
На следующем графике видно, что классификатор сработал хорошо. В данном графике представлен метод OneVsOne на примере libsvm



Далее представлен метод OneVsOne на примере libsvm с rbf ядром 0.01

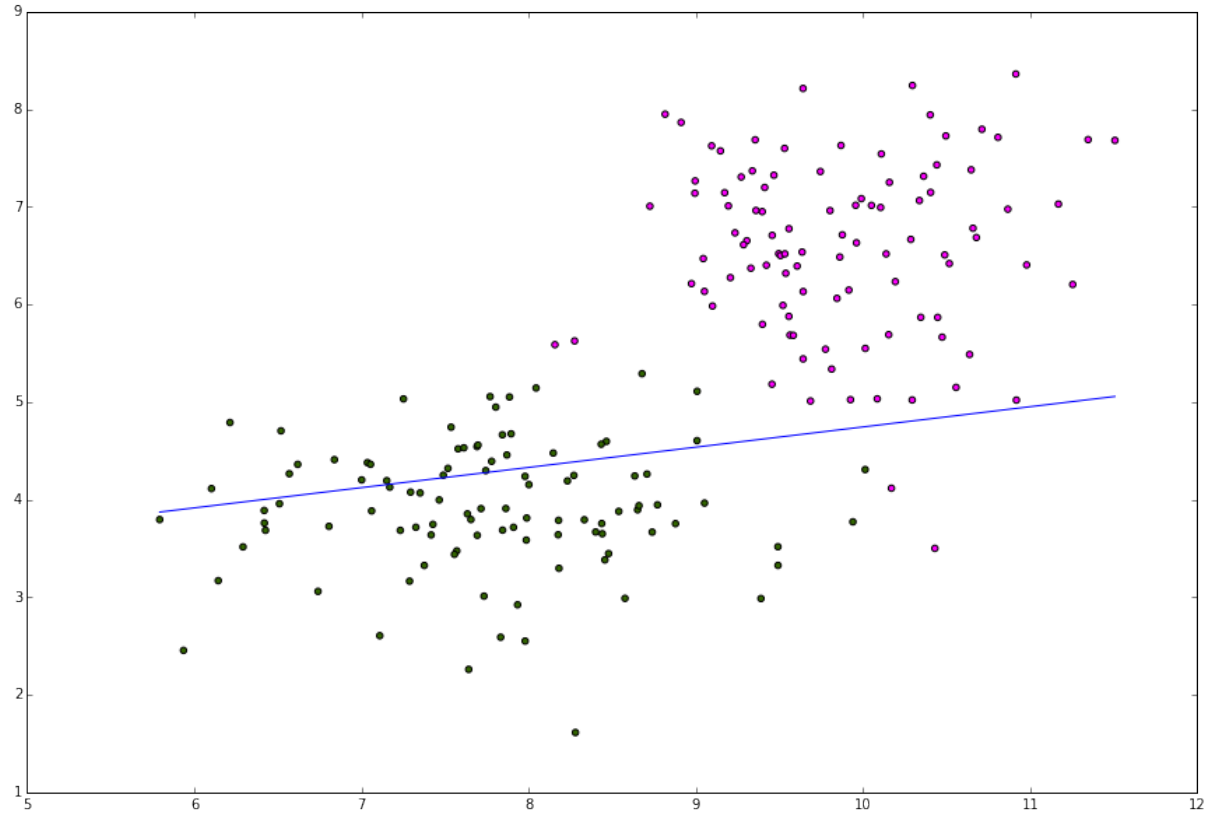


На этом графике представлен метод OneVsOne на примере libsvm с rbf ядром 1

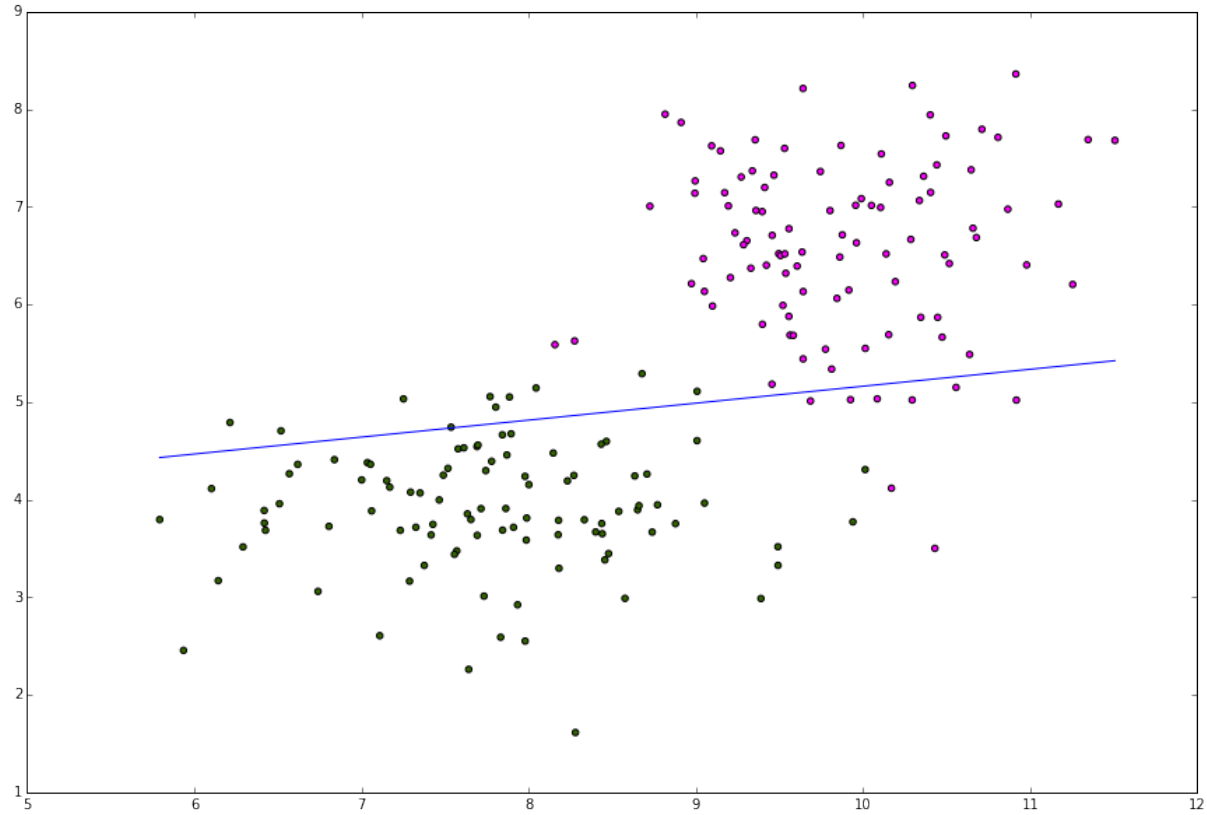


2.8 Пункт 7

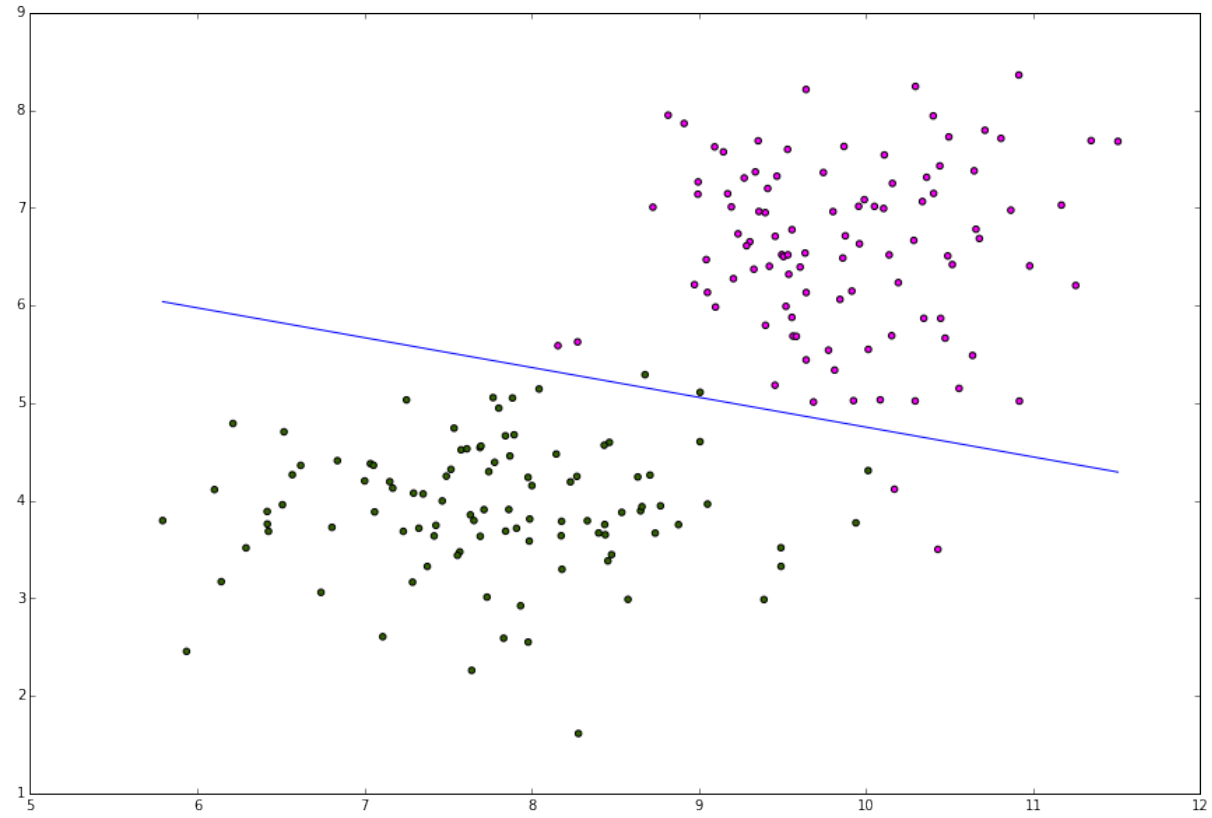
Метод субградиентного спуска



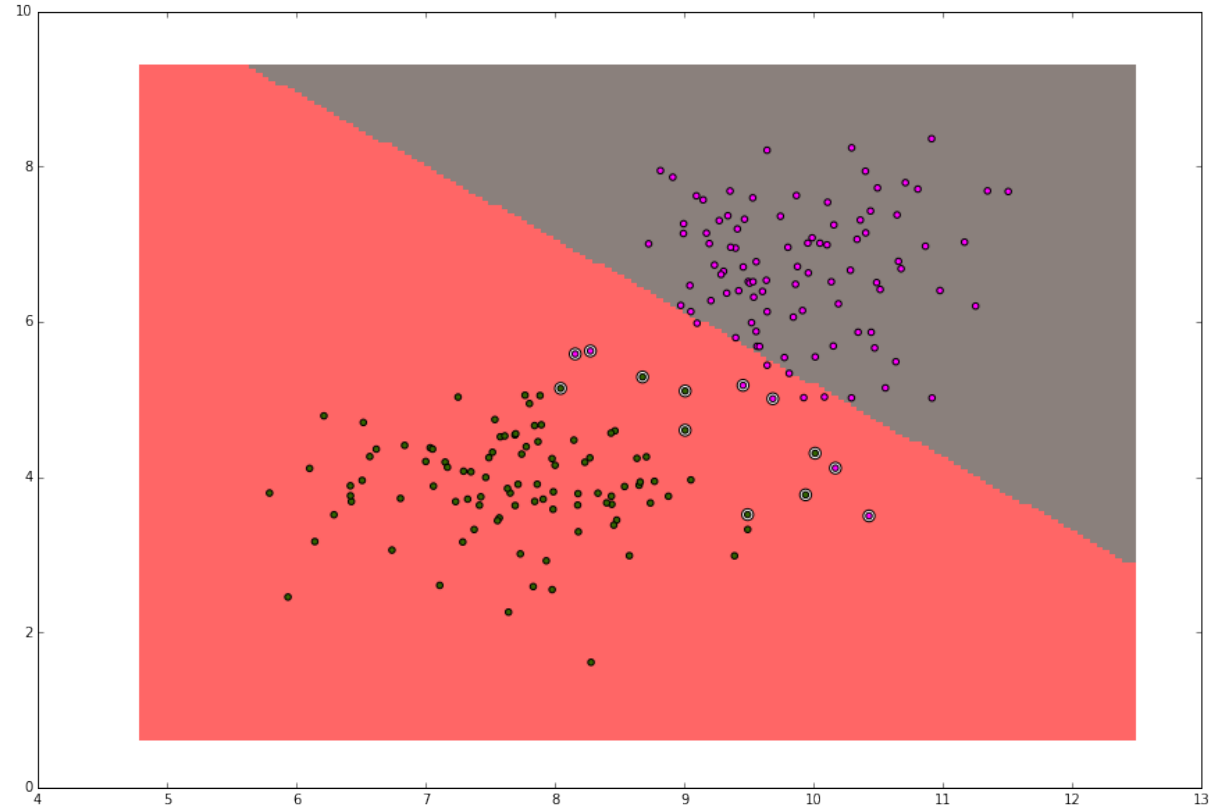
Метод стохастического субградиентного спуска



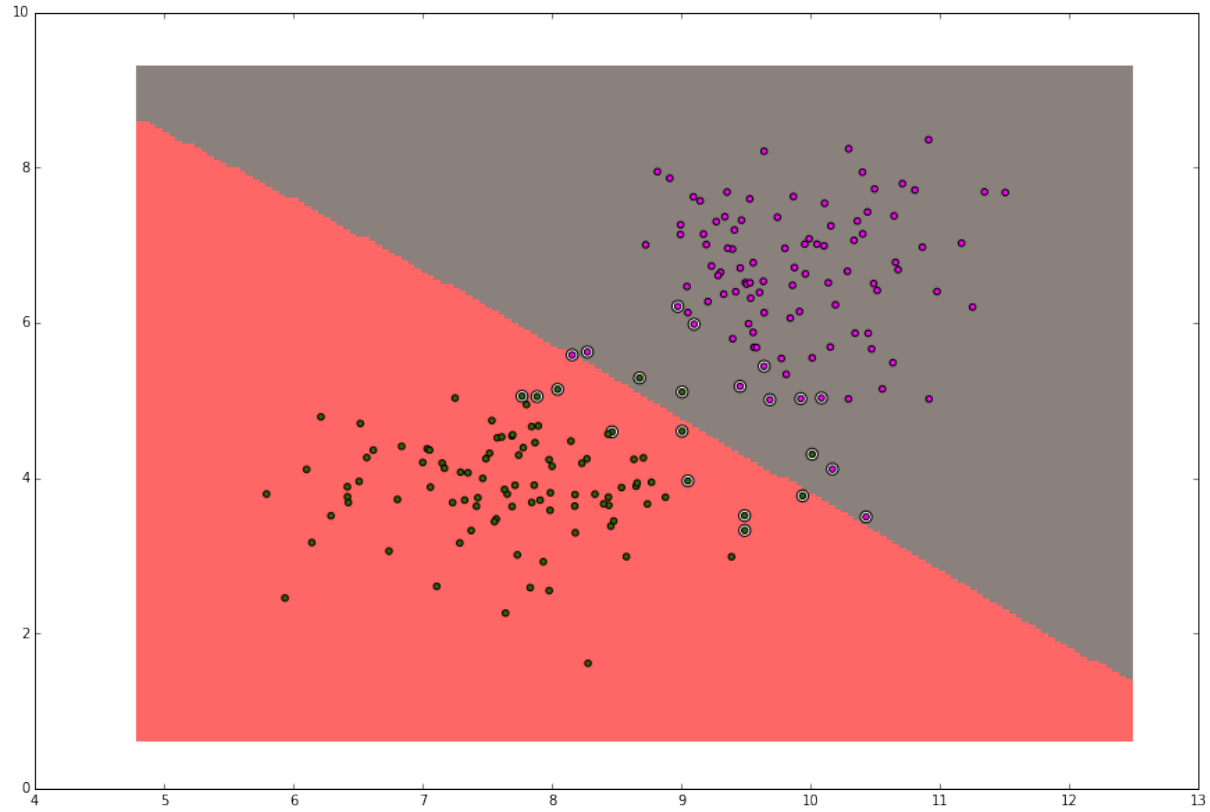
Метод primal solver



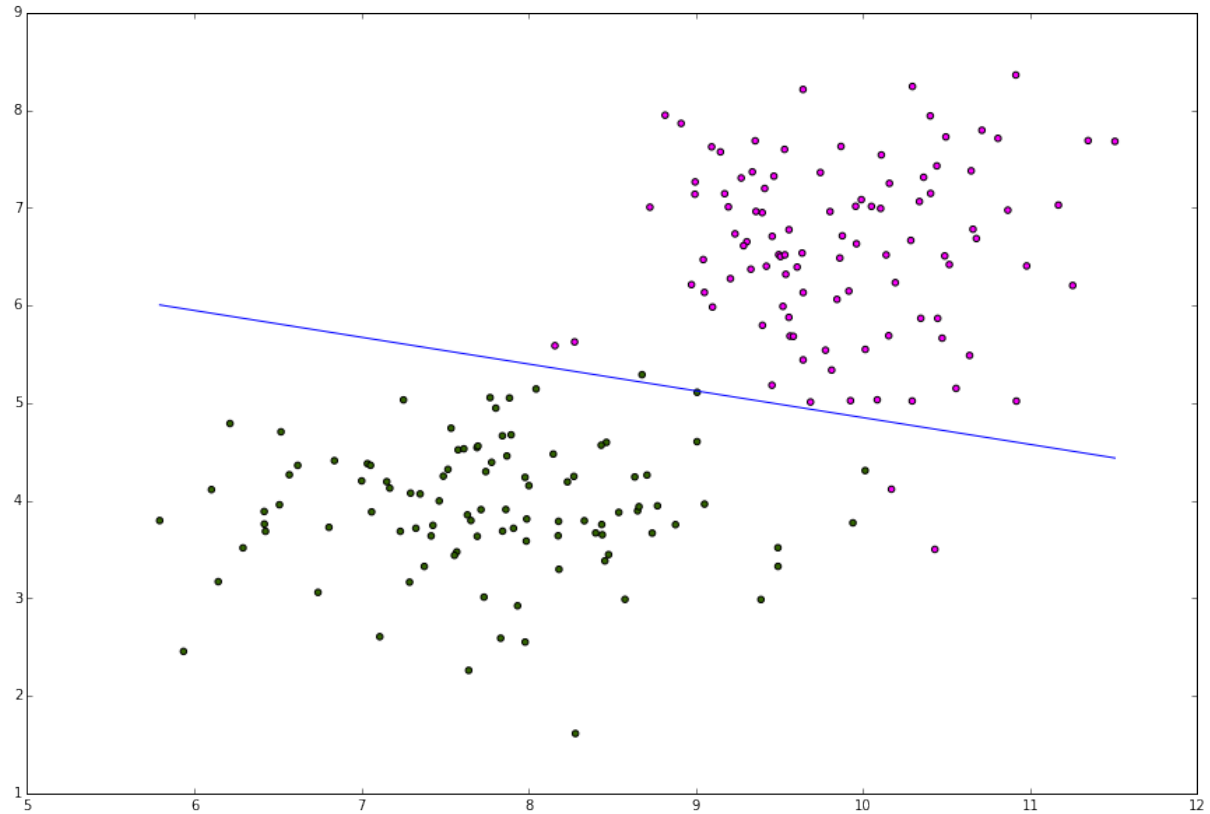
Метод dual solver



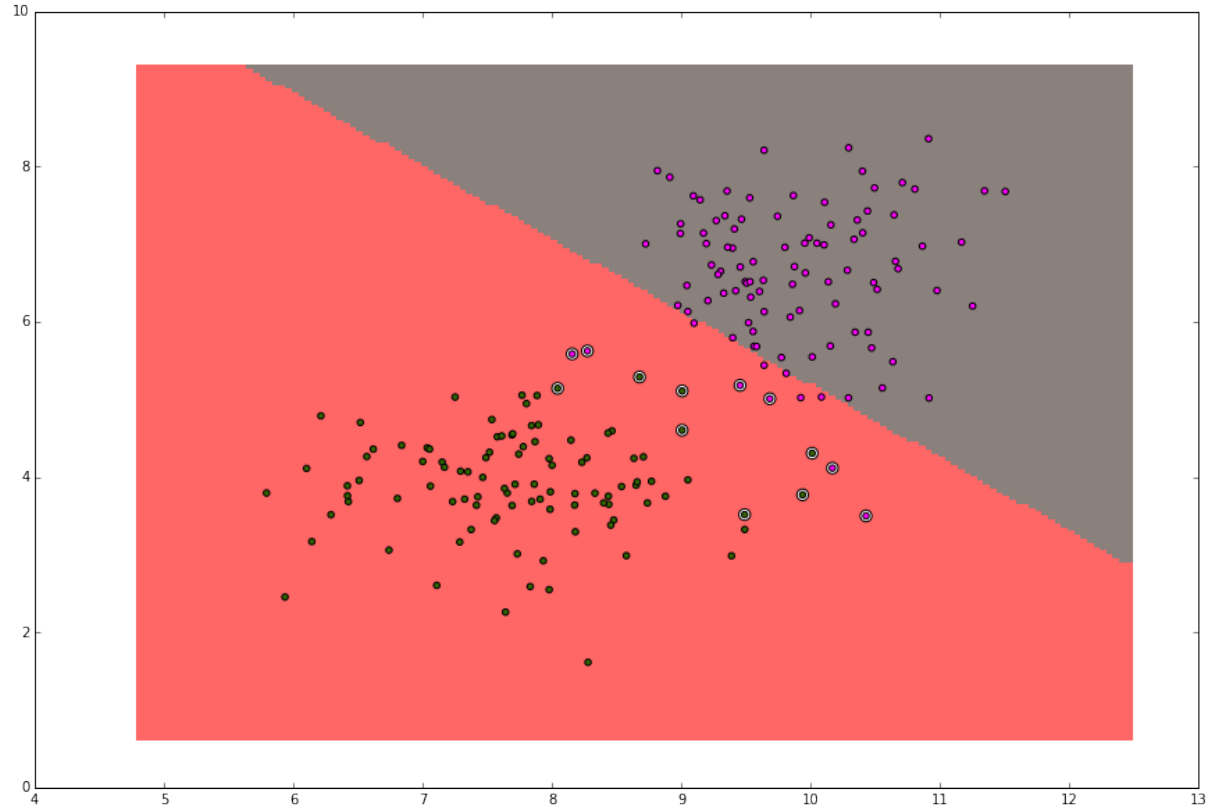
Метод primal solver с rbf ядром



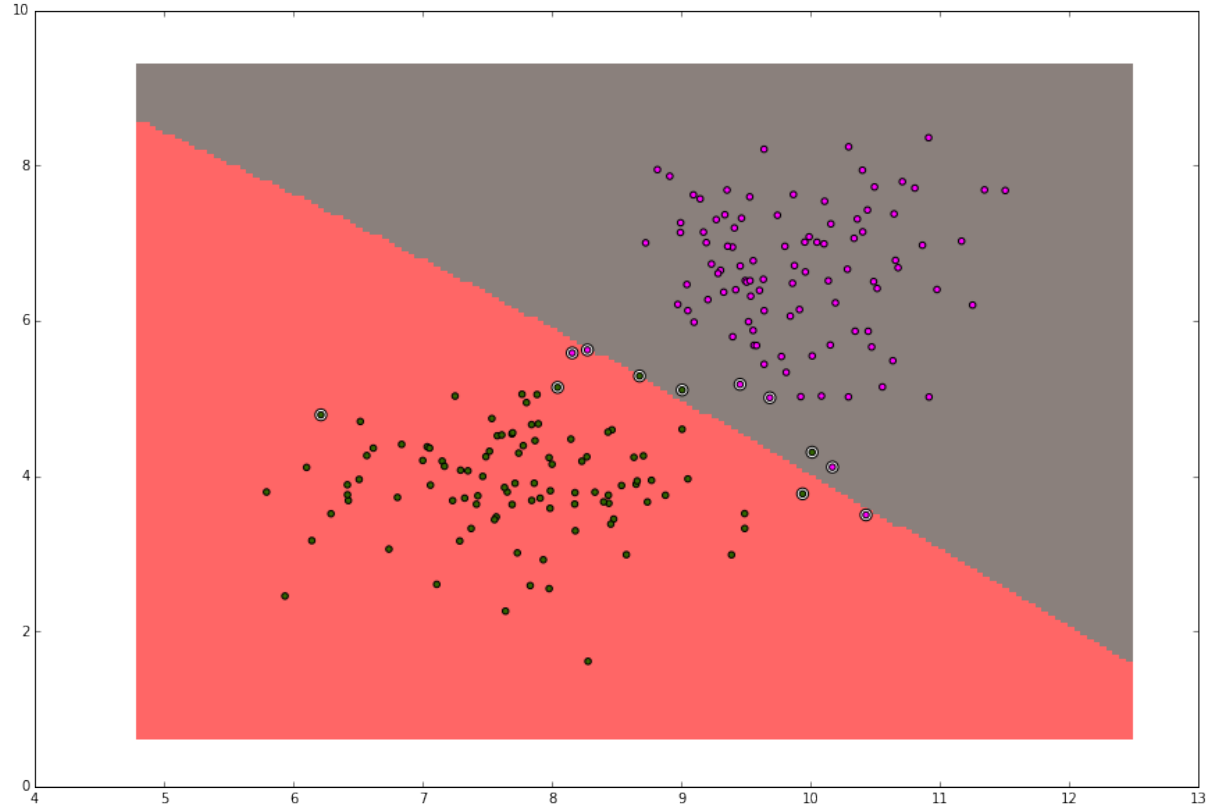
Метод liblinear



Метод libsvm



Метод libsvm с rbf ядром



2.9 Пункт 8

Для тестирования MNIST использовалась подвыборка из 200 элементов на каждый класс и метод OneVsOne. Выборка генерировалась случайным образом: данные для обучения выбирались из первых 60000 объектов (встроенная разбивка на обучение-контроль), а данные для тестирования выбирались из последних 10000 объектов. При этом количество объектов каждого класса одинаковые (в данном случае по 200 объектов)

Метод	Время работы	Точность
Primal solver	48.5 сек	0.90749
Dual solver	5.7 сек	0.90600
Dual solver с rbf ядром	5.15 сек	0.81299
Liblinear	0.616 сек	0.90749
Libsvm	1.52 сек	0.90649
Libsvm с rbf ядром	8.92 сек	0.76500

По этим данным можно сделать вывод, что самый быстрый и самый точный метод для классификации MNIST - это метод liblinear

3 Заключение

По результатам исследований выяснилось, что каждый метод решения одного и того же задания работает не похоже друг на друга. У каждого метода разное время работы и разные точности и для каждой задачи нужно выбирать метод с умом. В нашем случае, тесты показали, что для классификации MNIST надо выбирать метод liblinear решения задачи SVM.

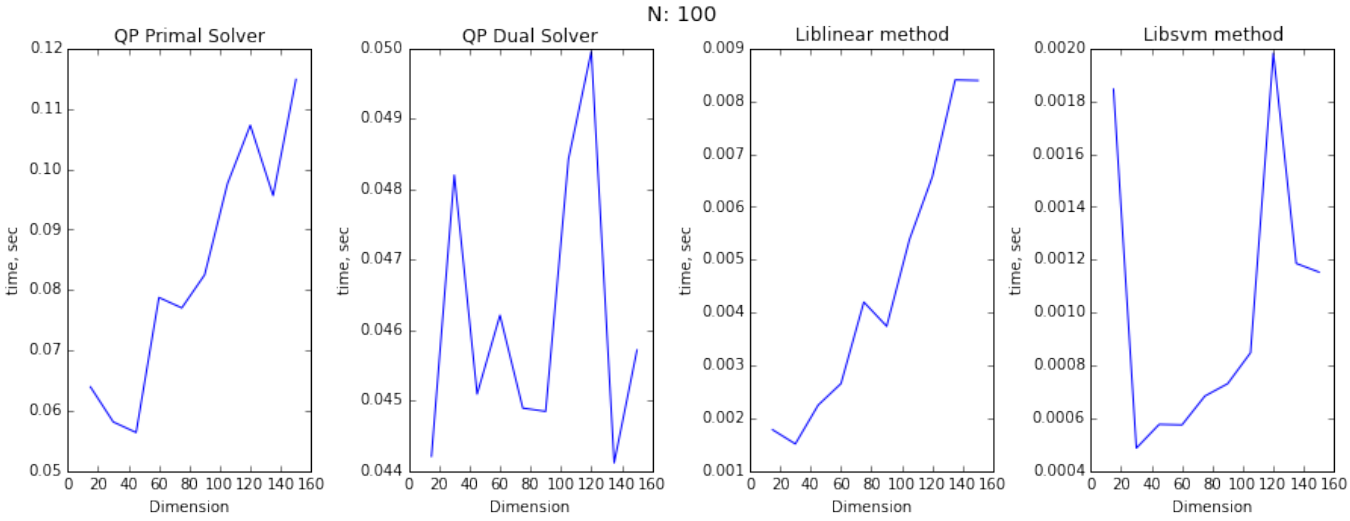
P.S. Для полного MNIST я не смог дождаться завершения обучения на liblinear (слабый процессор), но оставил код, чтобы попробовали запустить MNIST у себя на компьютере.

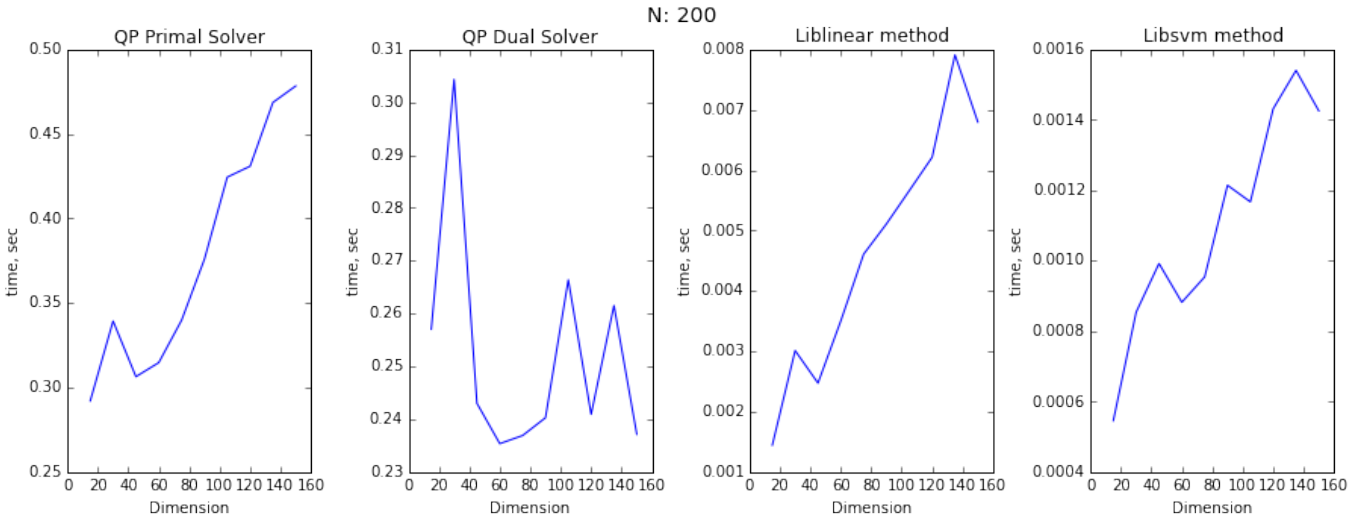
4 Приложение

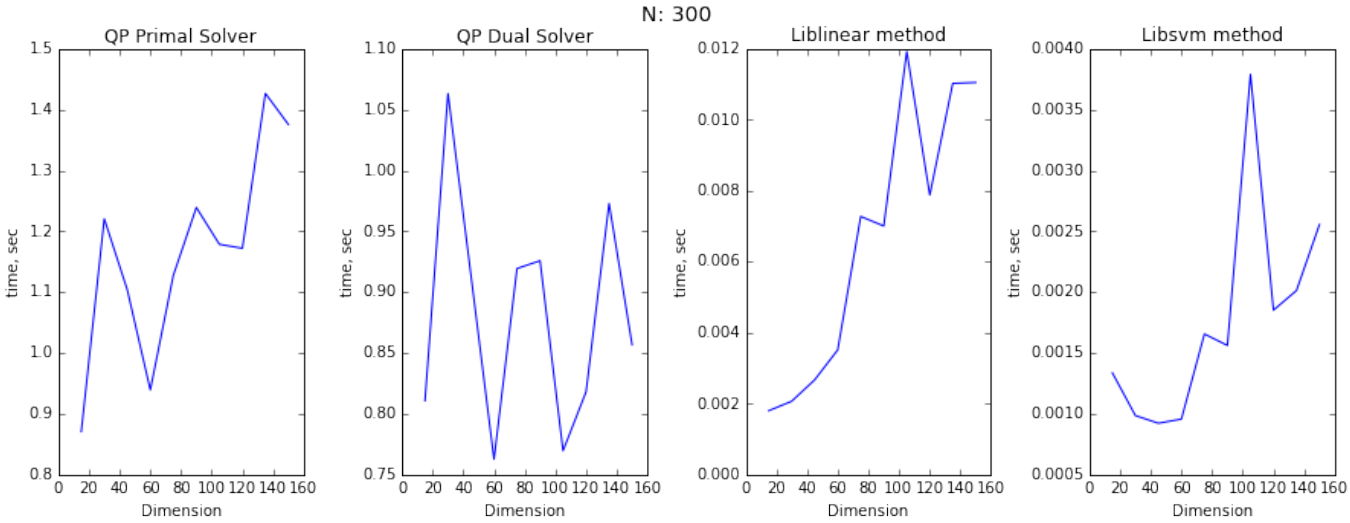
4.1 Отдельные графики 1-го пункта

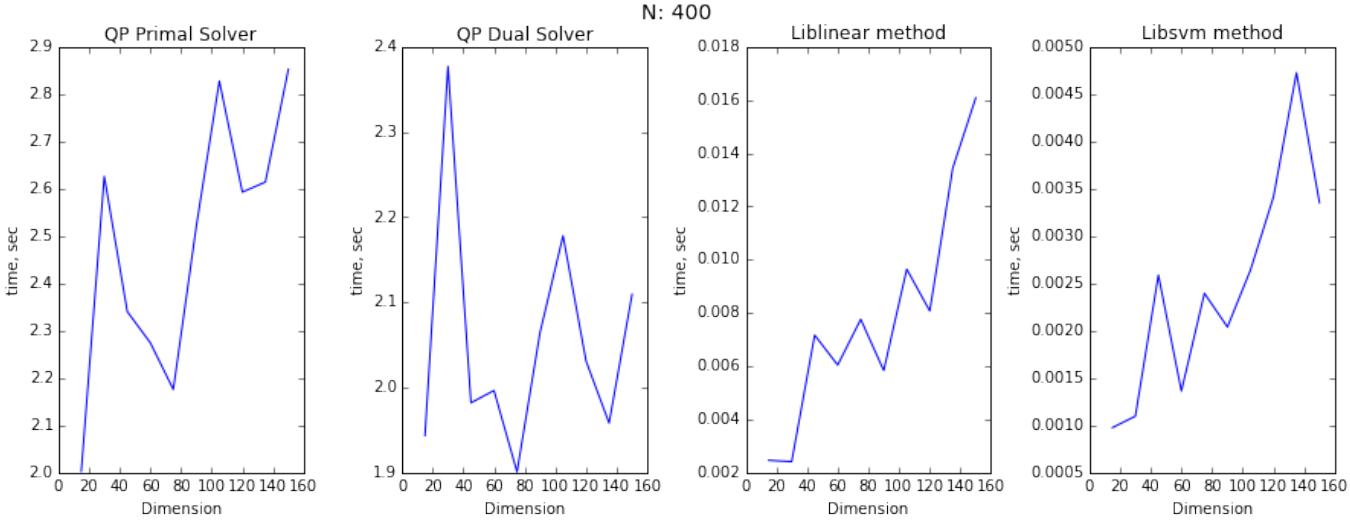
Исследуем зависимость времени работы методов от размерности пространства для каждого размера выборки.

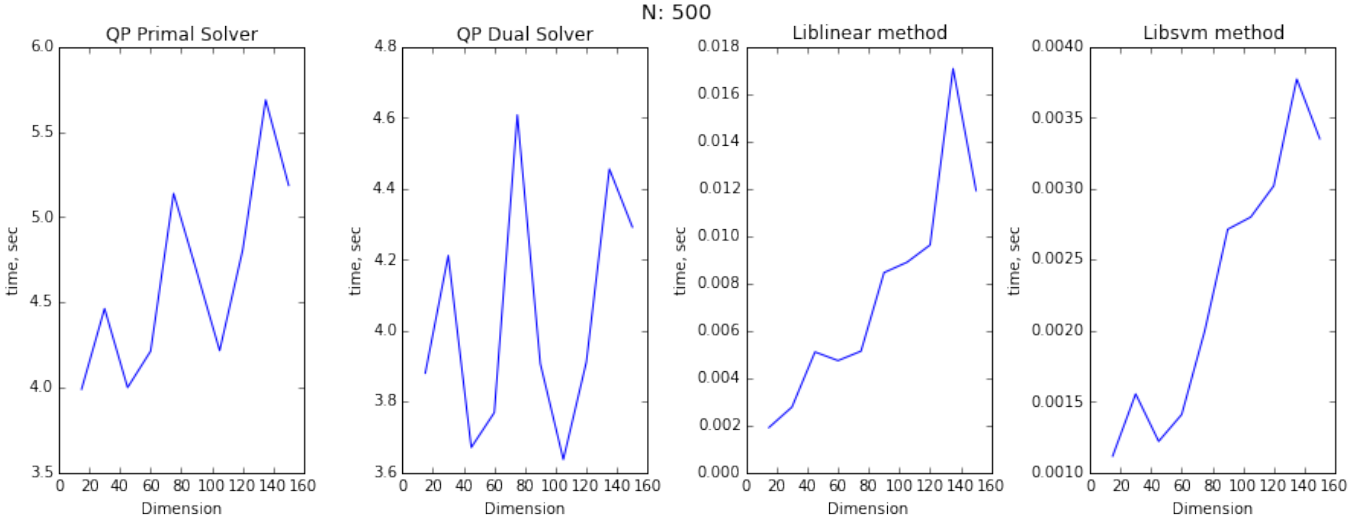
На слелующих фото представлены графики зависимости времени работы от размерности пространства. На них можно увидеть, что самый быстрый метод libsvm, далее liblinear

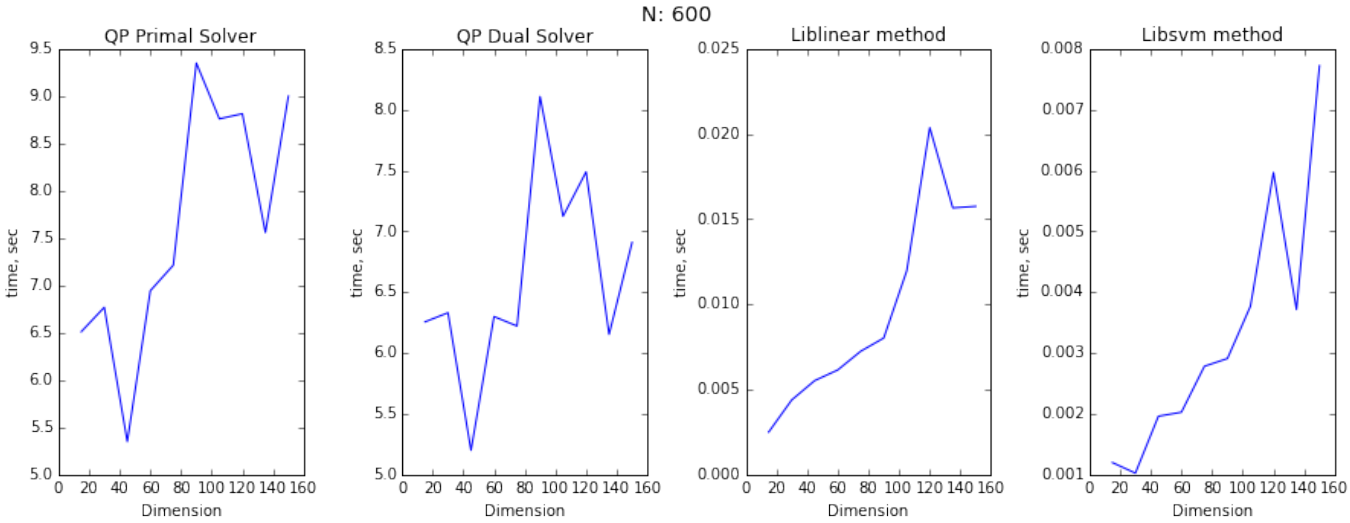


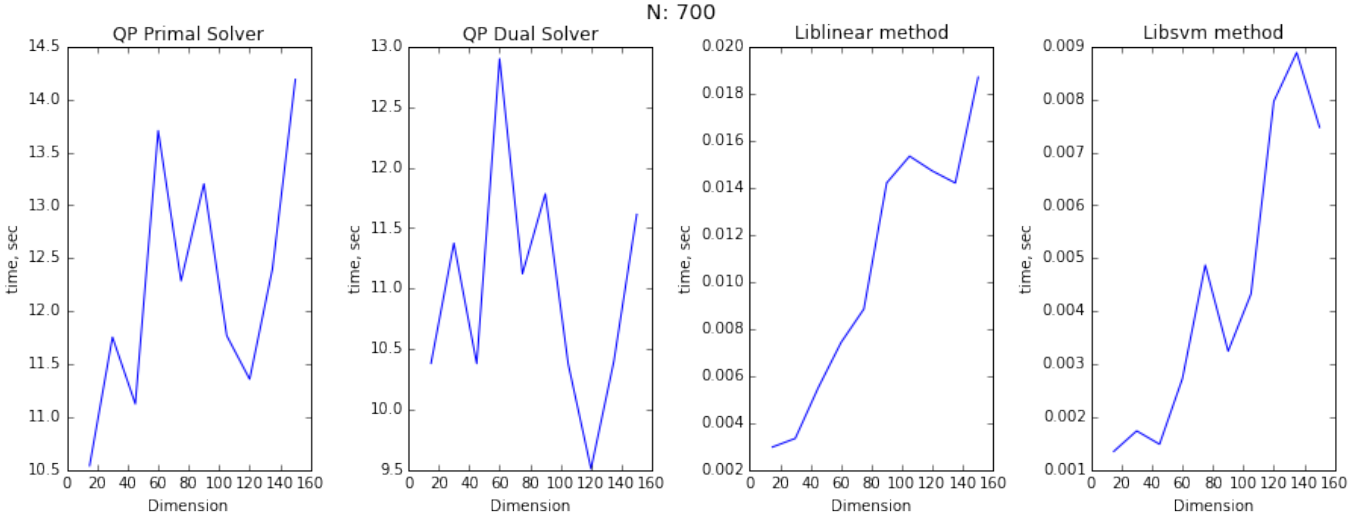


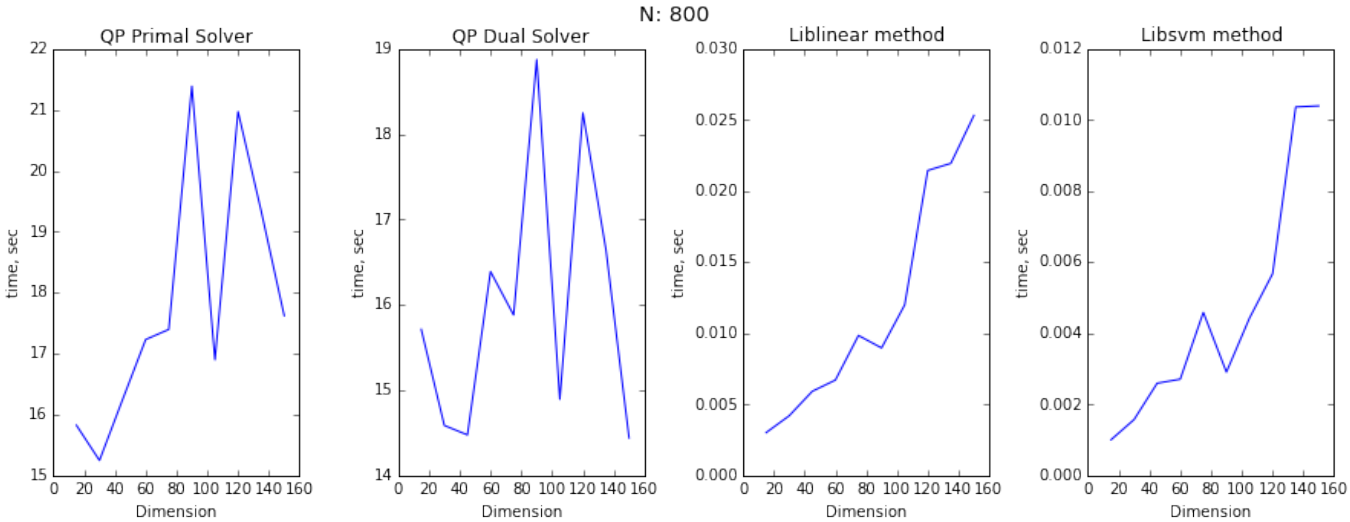

N: 300
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension

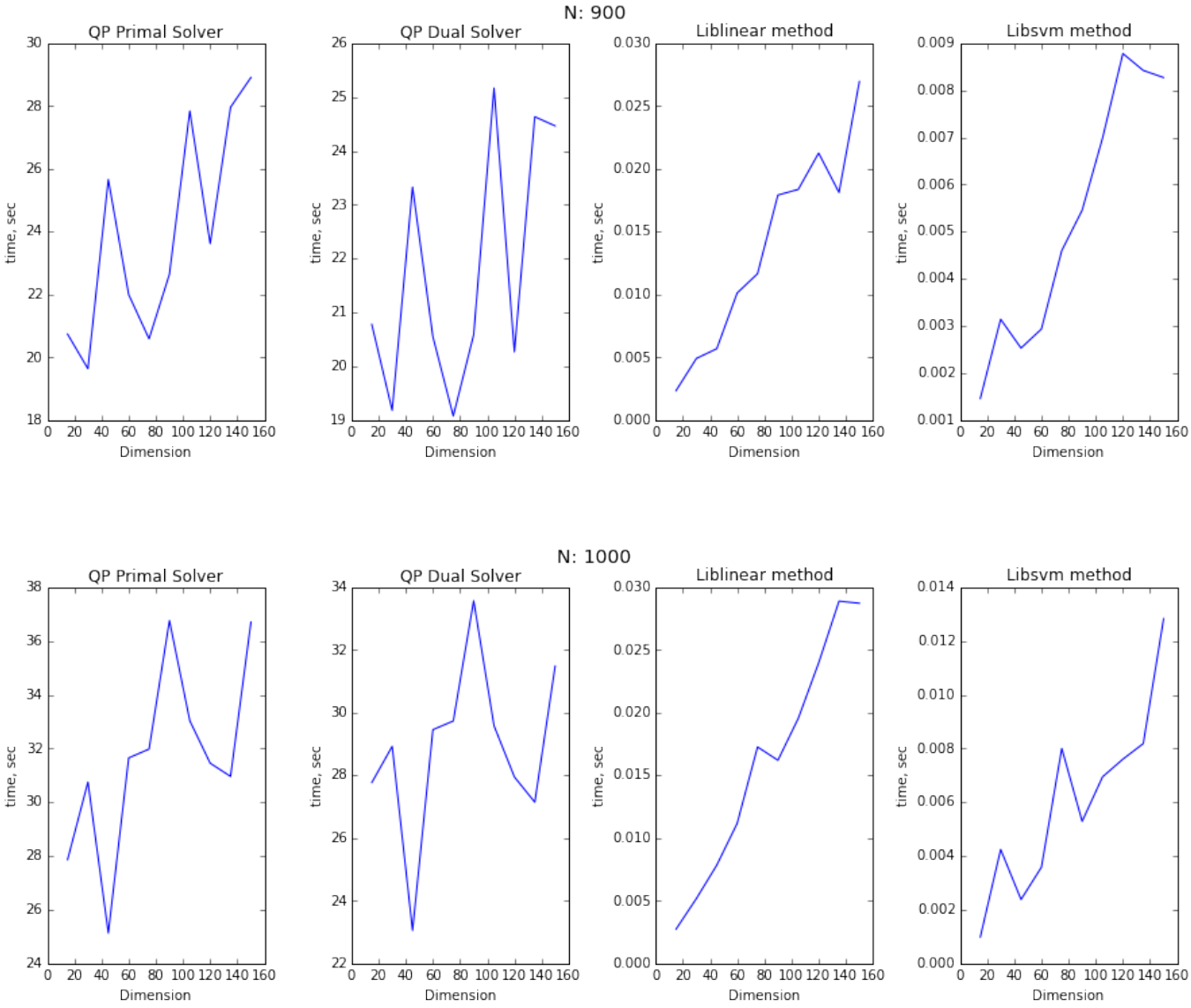

N: 400
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension


N: 500
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension


N: 600
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension

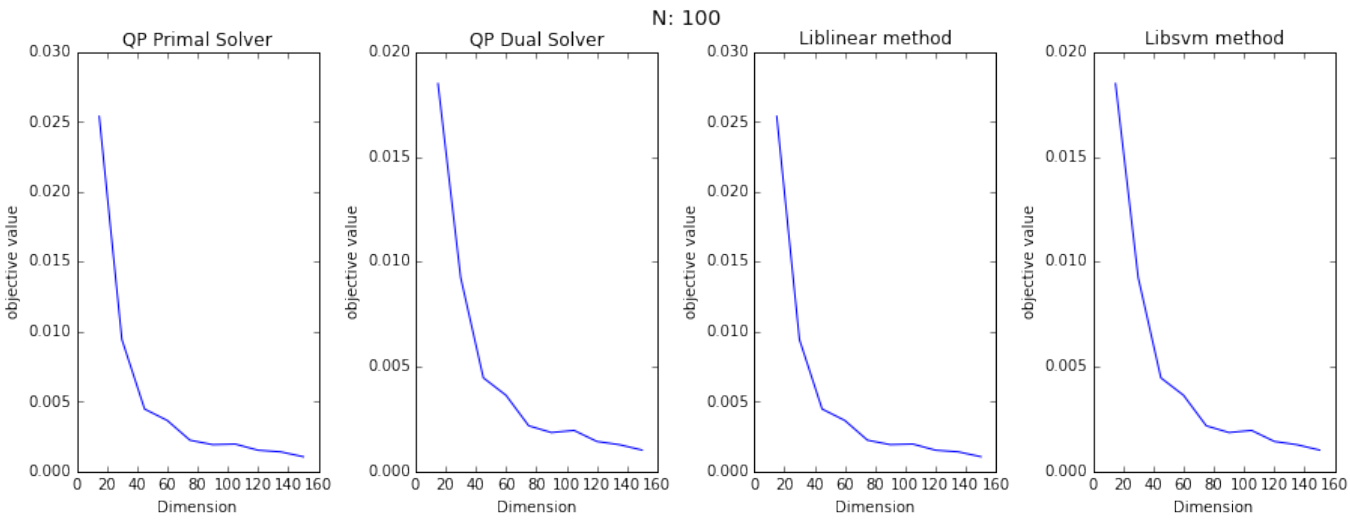

N: 700
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension

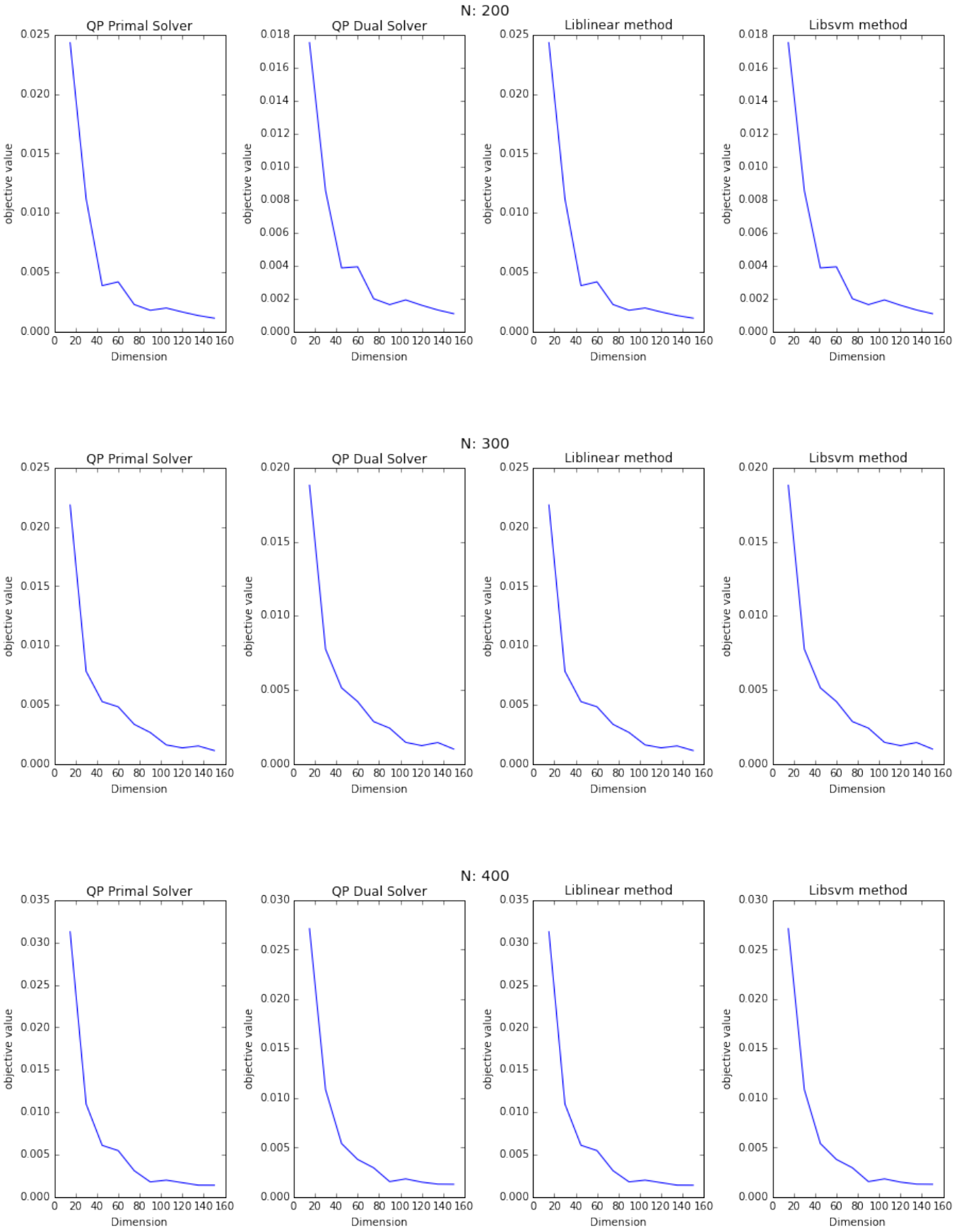

N: 800
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
time, sec
Dimension

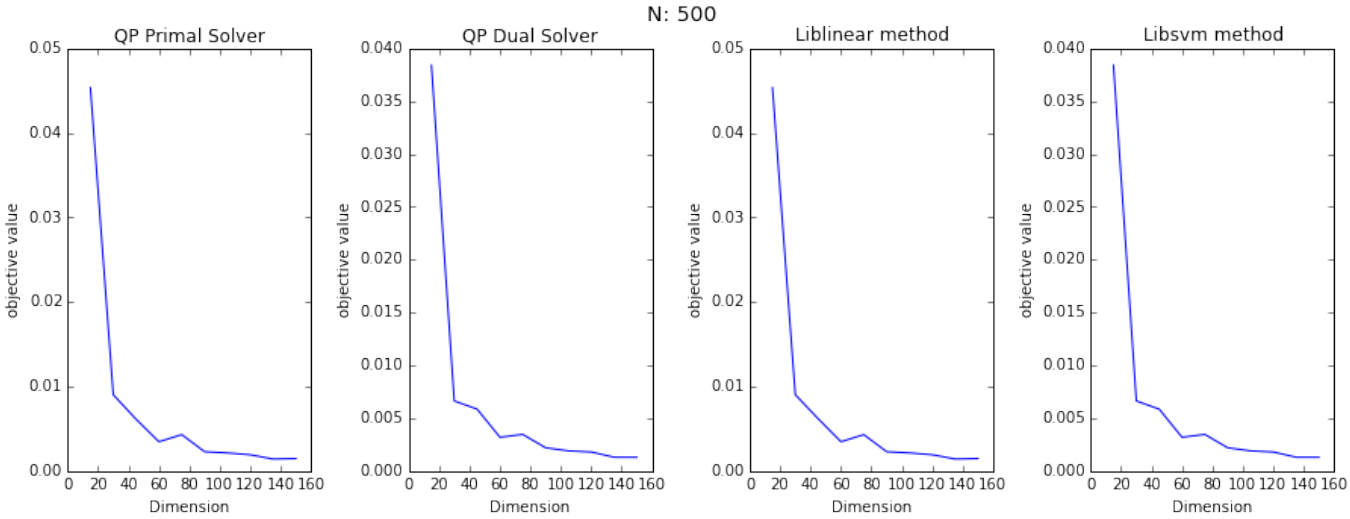


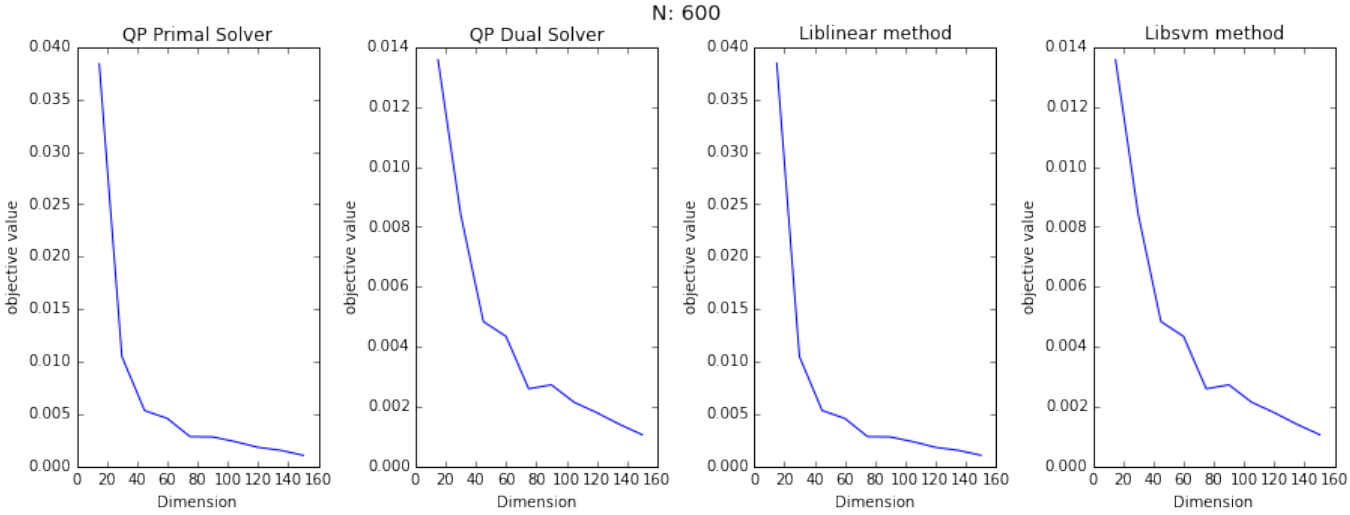
На следующих графиках представлены значения функционалов в зависимости от размерности пространства.

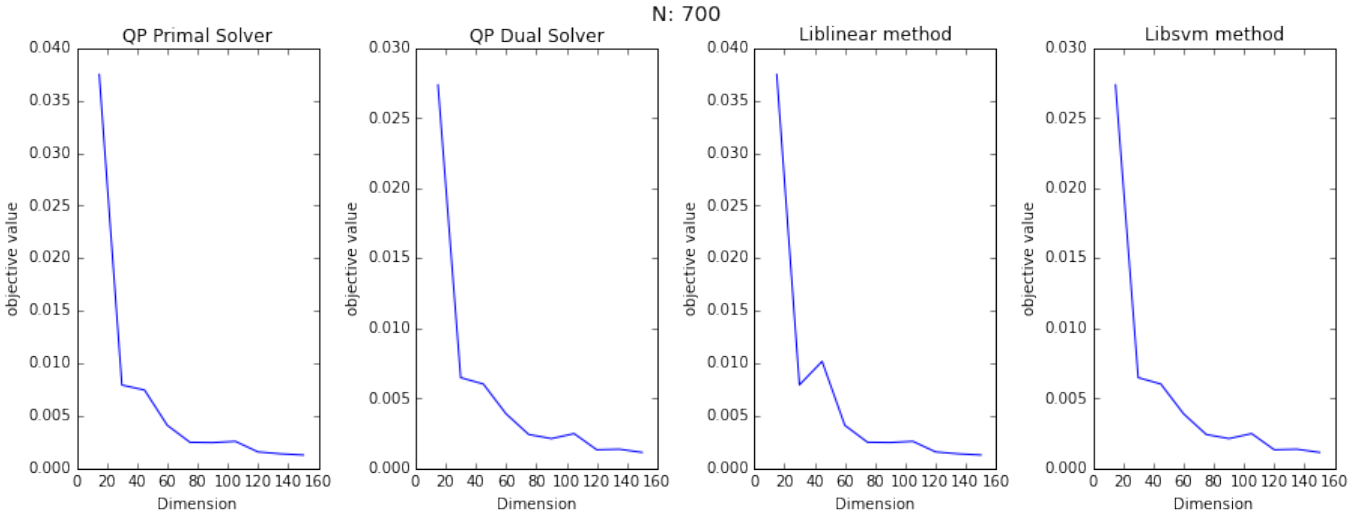
На них можно заметить, что значения одинаковые для методов primal и liblinear, а также для методов dual и libsvm. В целом значения для всех методов похожи.

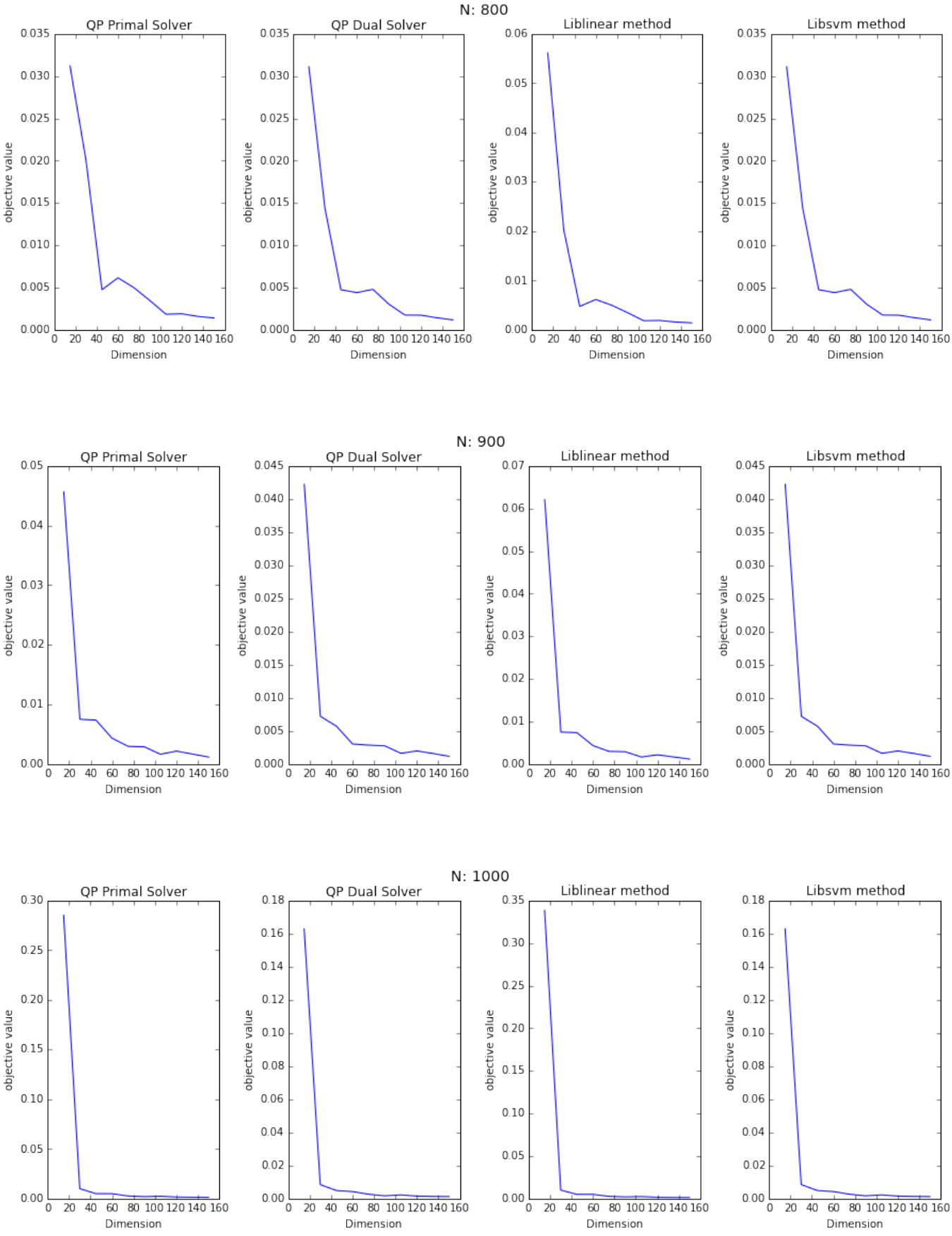



N: 200
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
N: 300
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
N: 400
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension


N: 500
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension


N: 600
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension


N: 700
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension

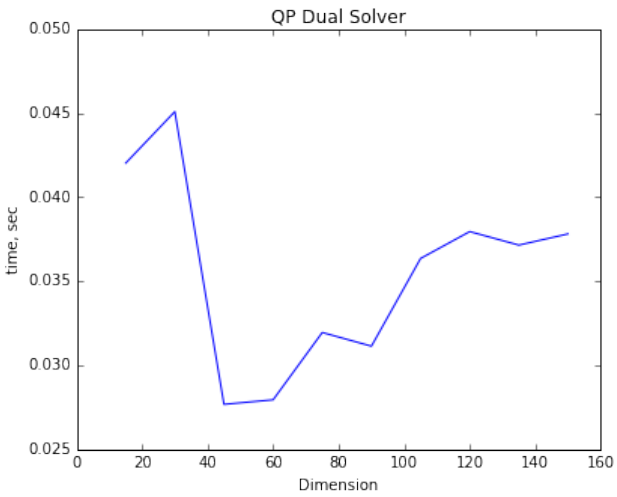

N: 800
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
N: 900
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension
N: 1000
QP Primal Solver
QP Dual Solver
Liblinear method
Libsvm method
objective value
Dimension

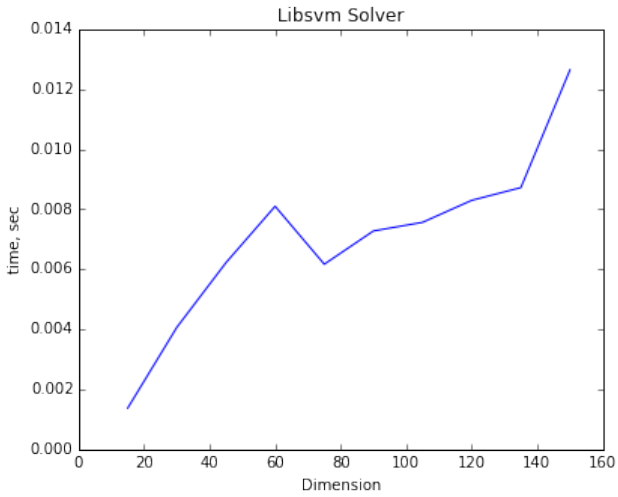
4.2 Отдельные графики 2-го пункта

На следующих графиках представлены зависимости времени от размерности пространства для каждого размера выборки.

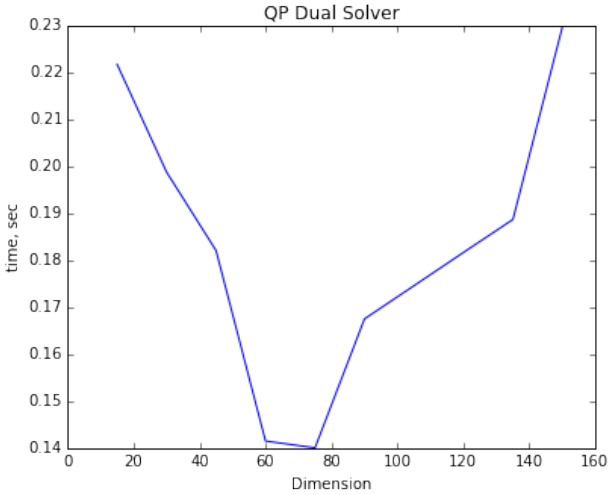
По времени работы быстрым является метод libsvm

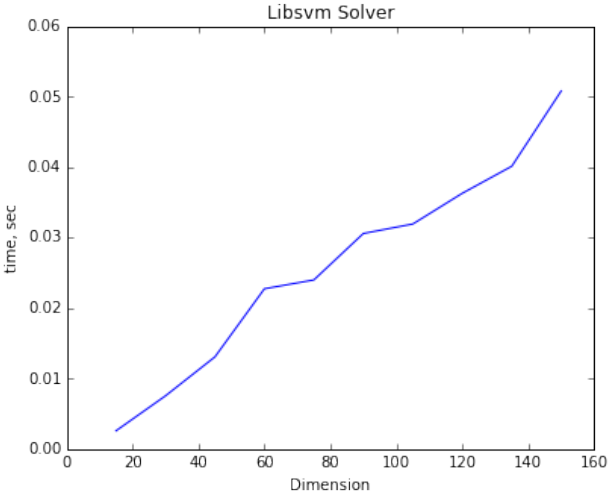
N: 100



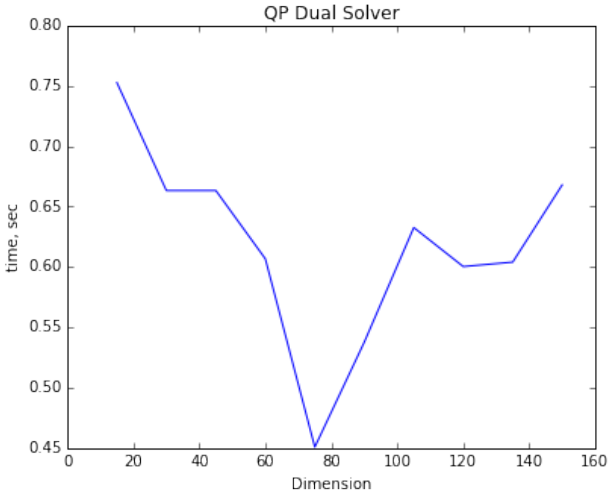


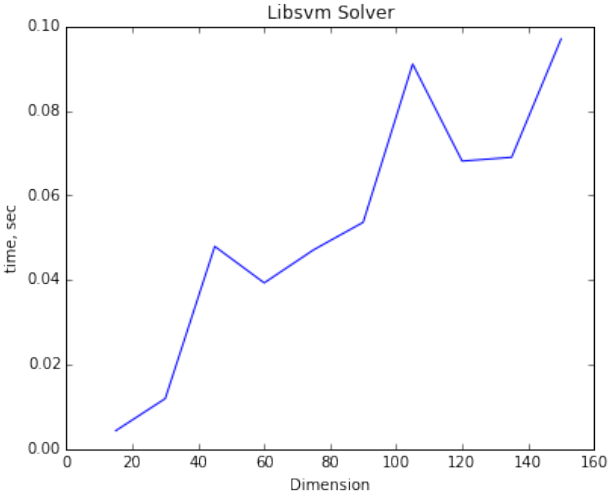
N: 200



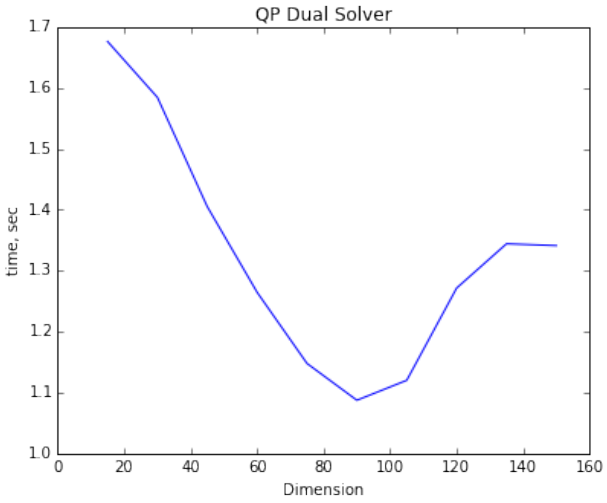


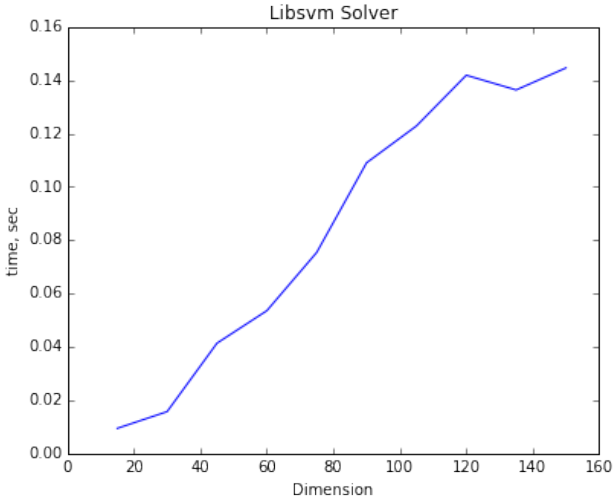
N: 300



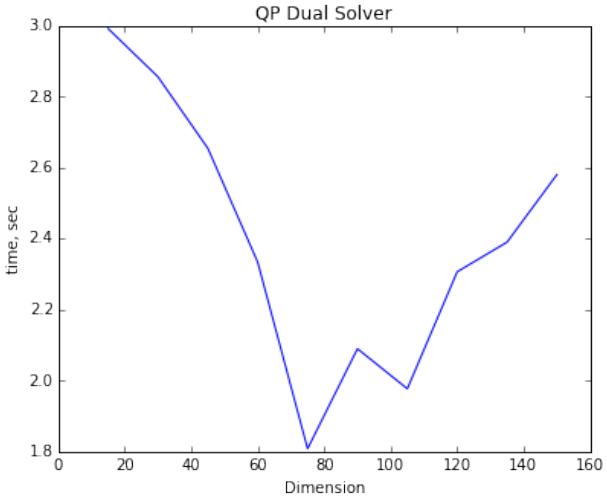


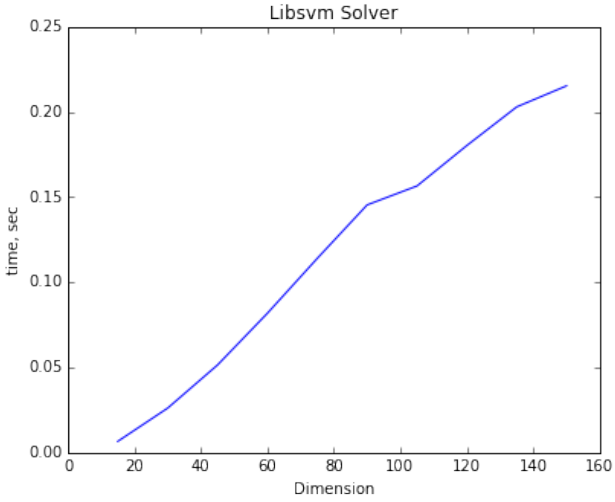
N: 400



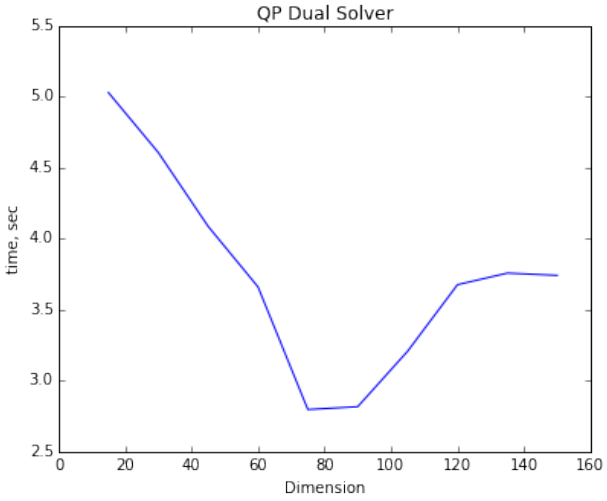


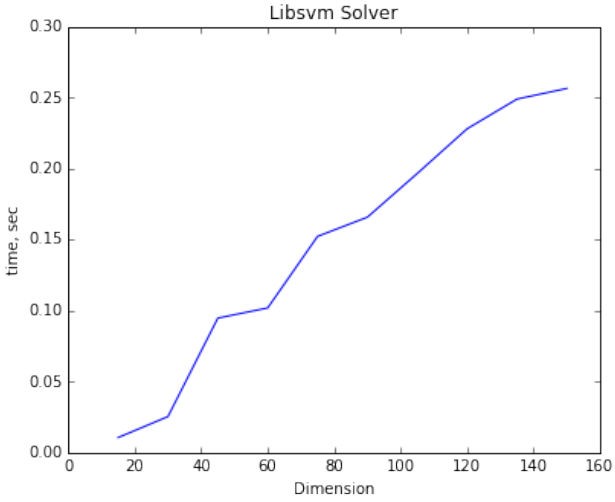
N: 500



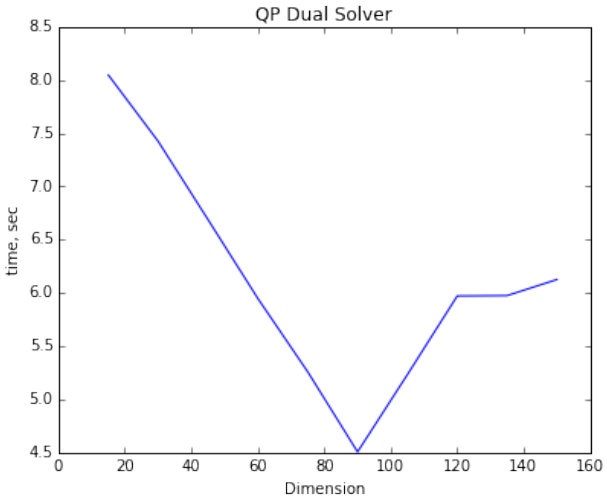


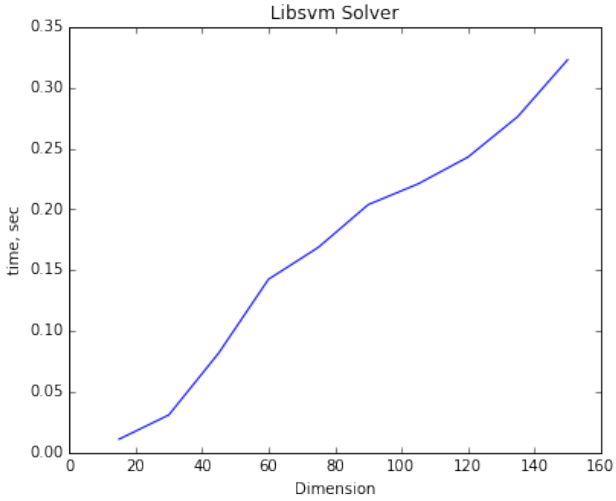
N: 600



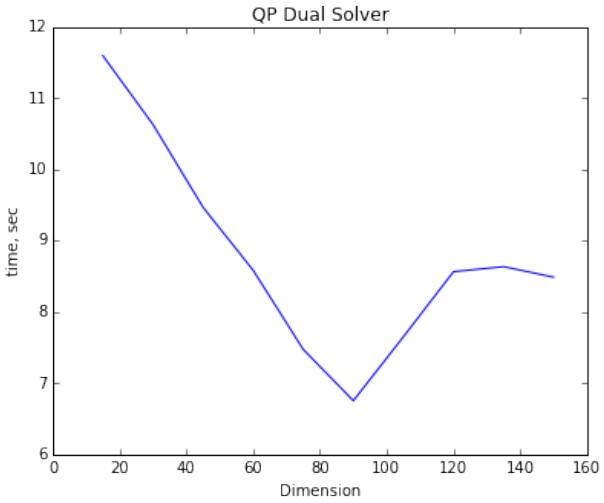


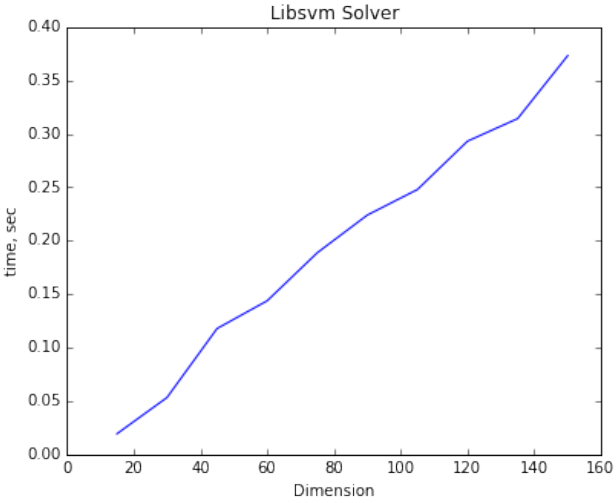
N: 700



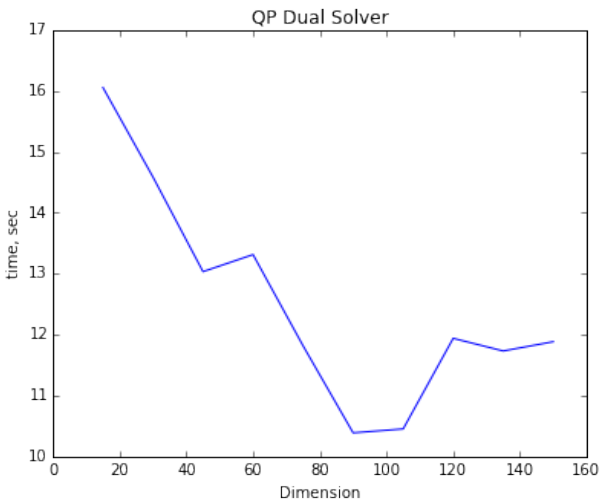


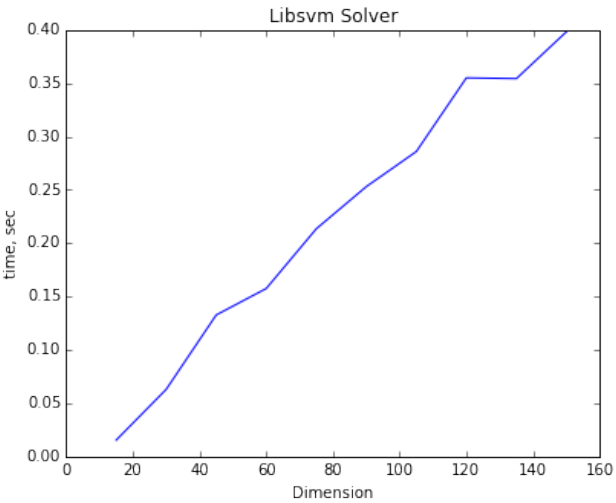
N: 800



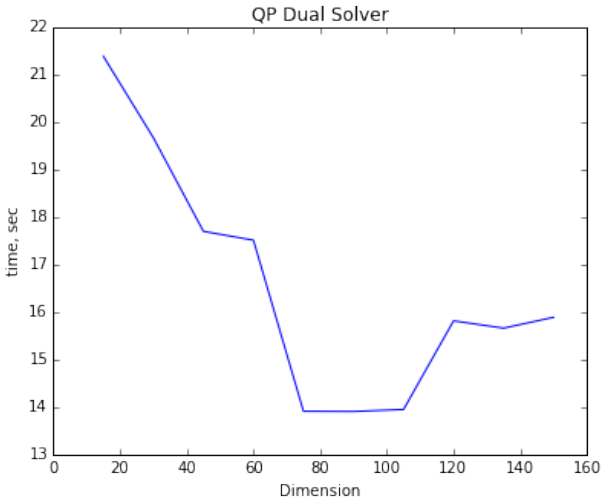


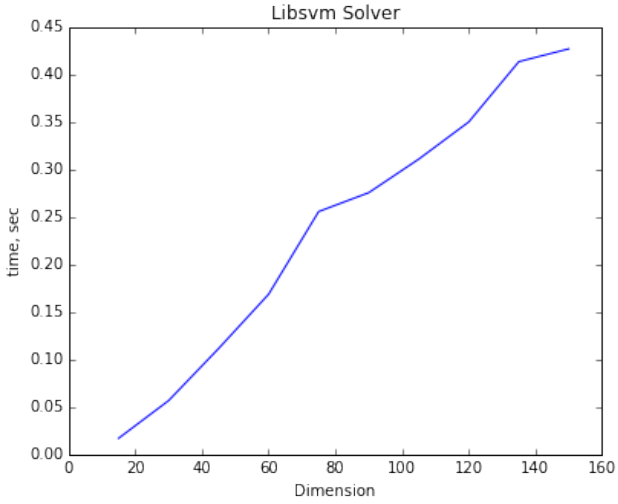
N: 900





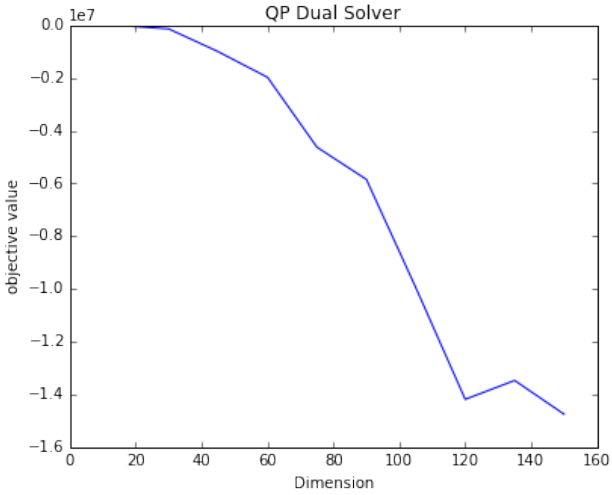
N: 1000

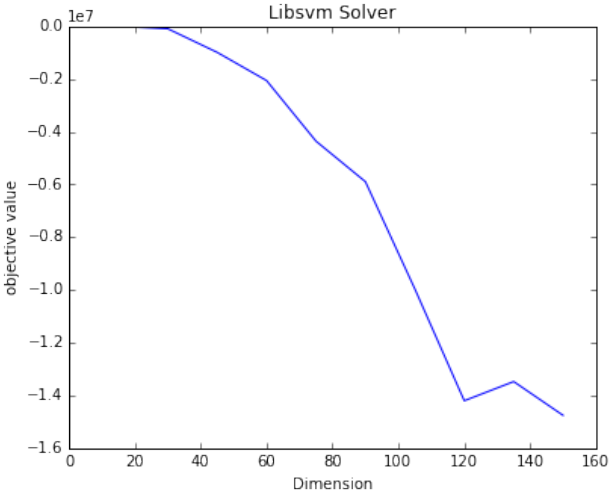




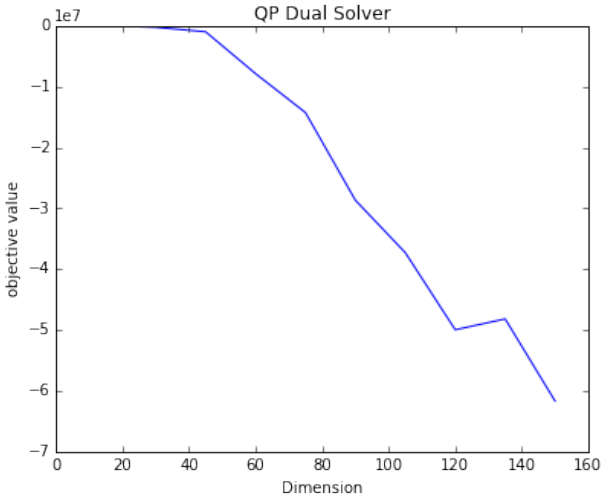
Далее представлены графики значений функционала. Для обоих методов они одинаковые

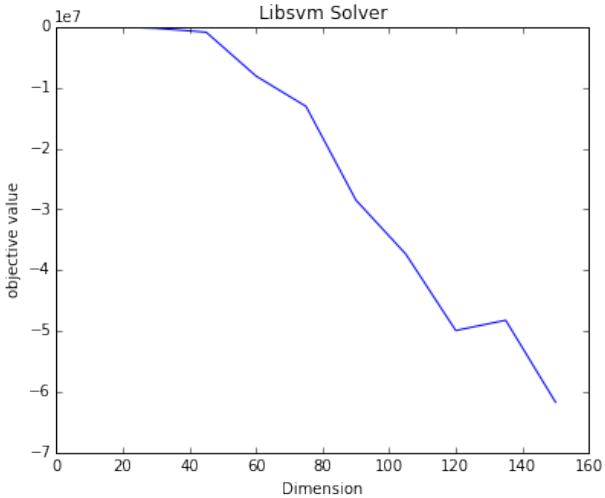
N: 100



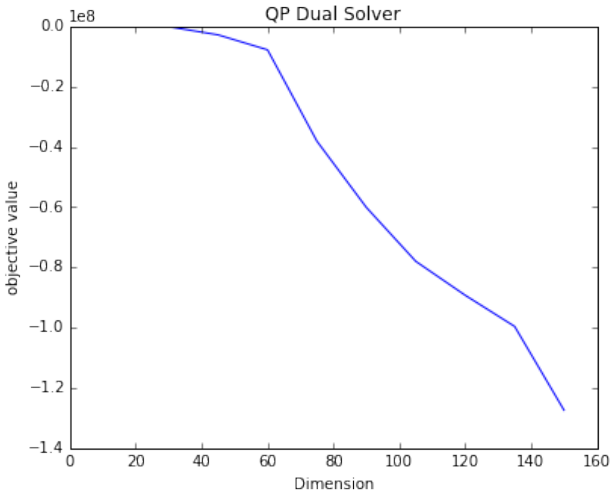


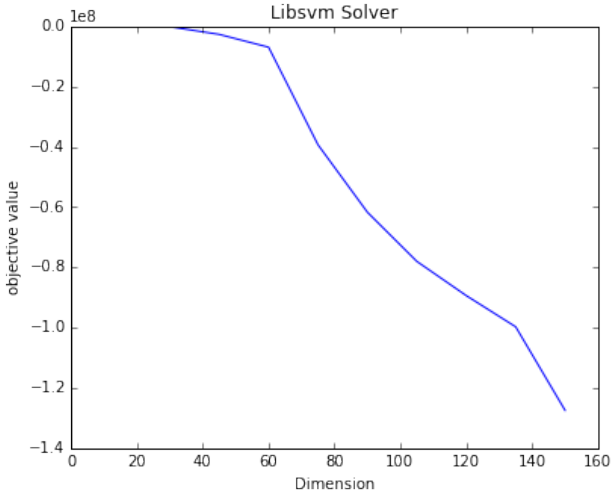
N: 200



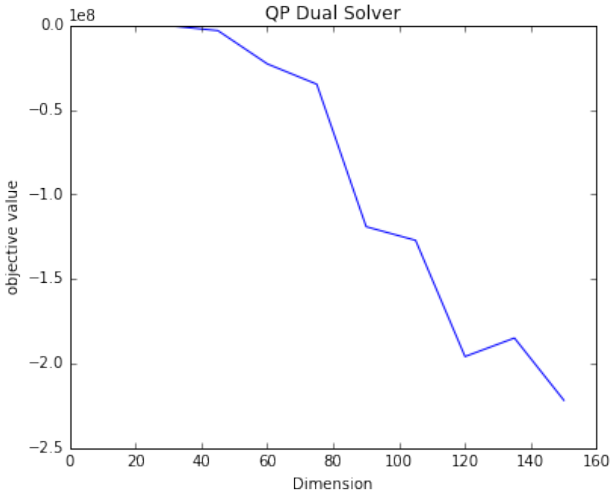


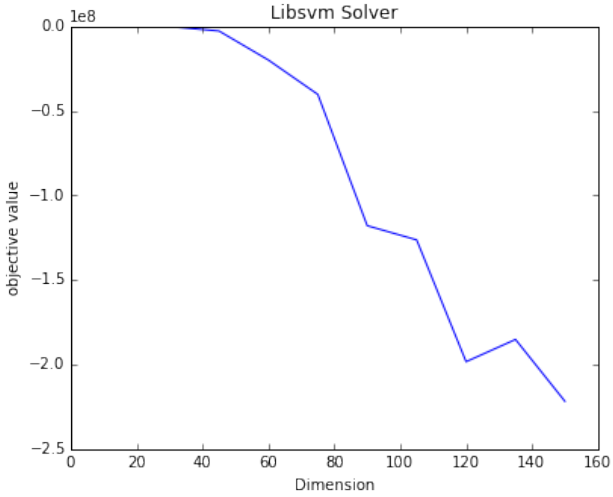
N: 300



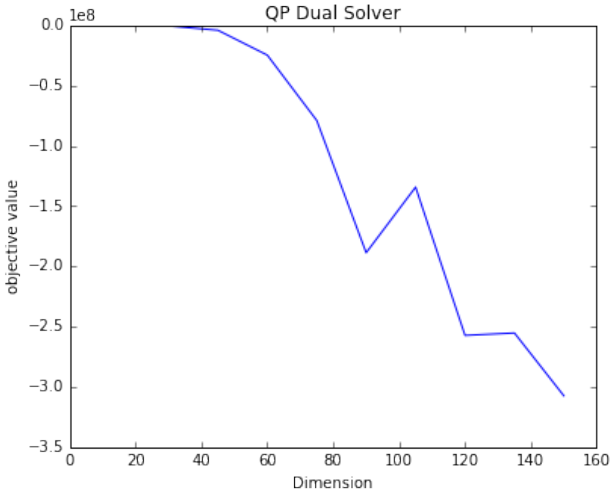


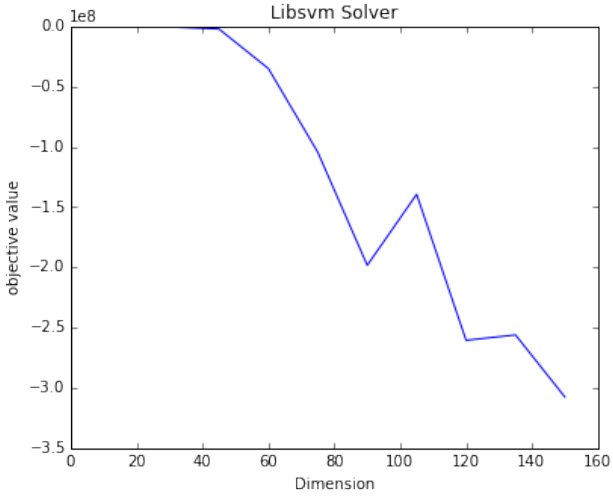
N: 400



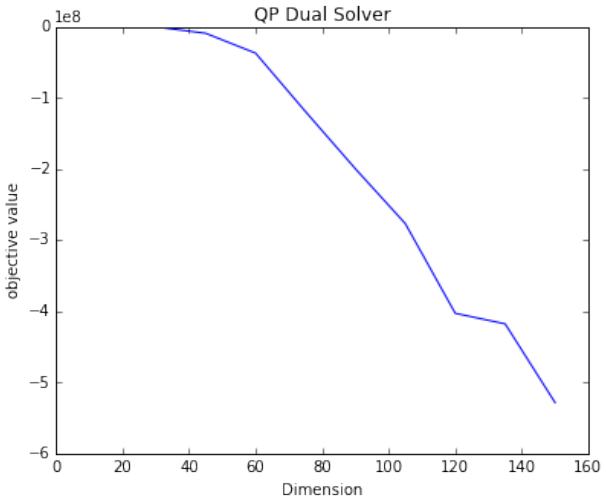


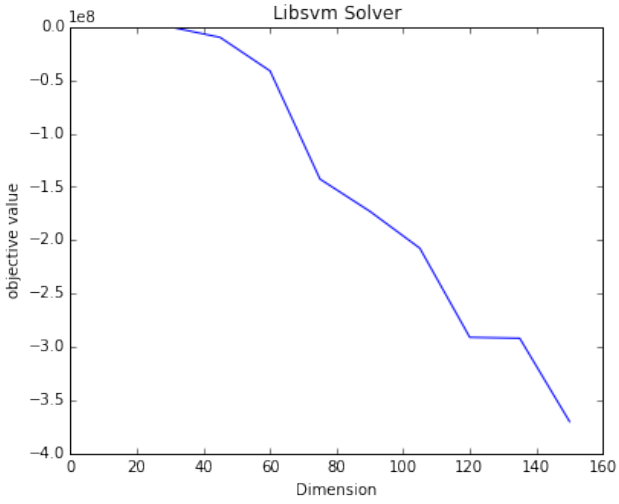
N: 500



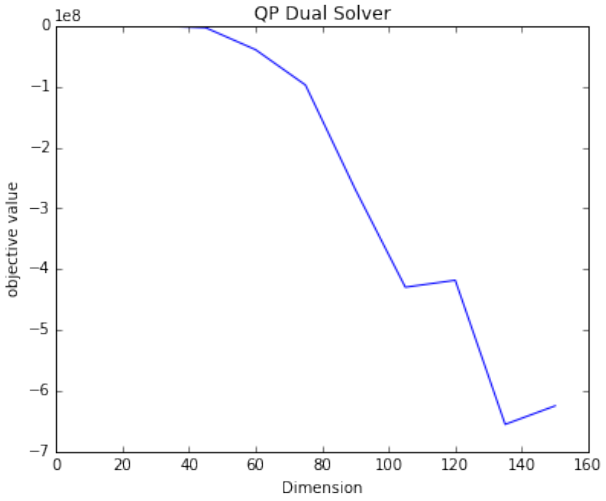


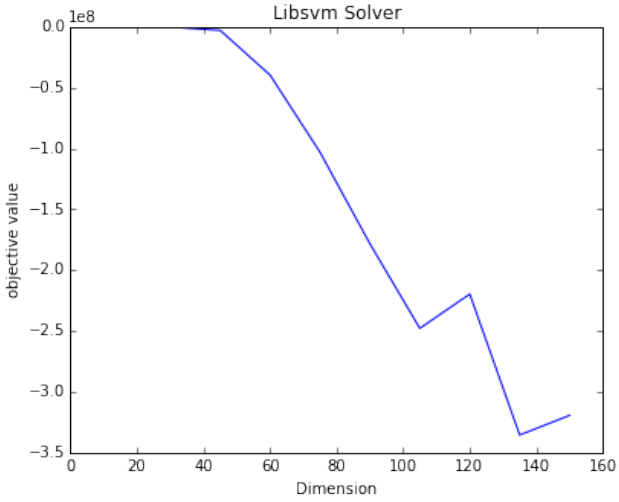
N: 600



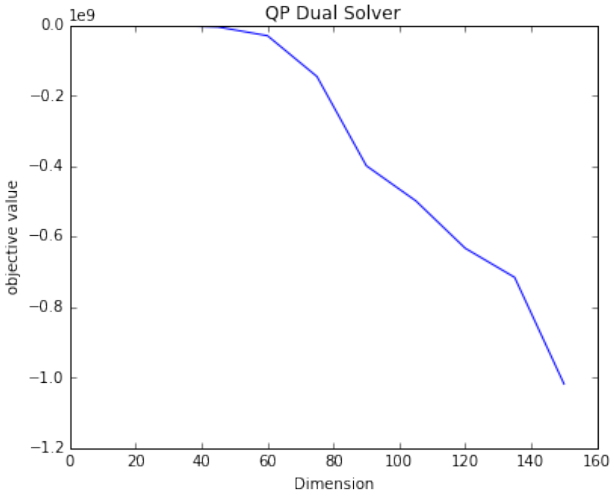


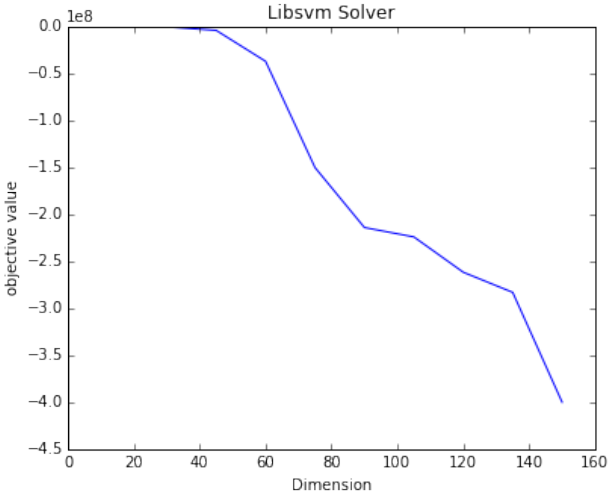
N: 700



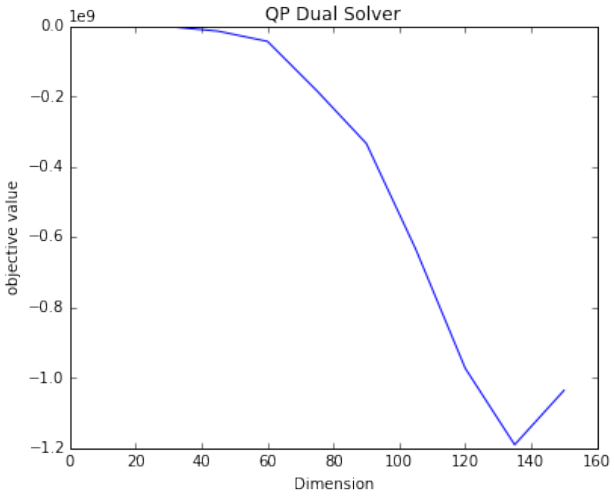


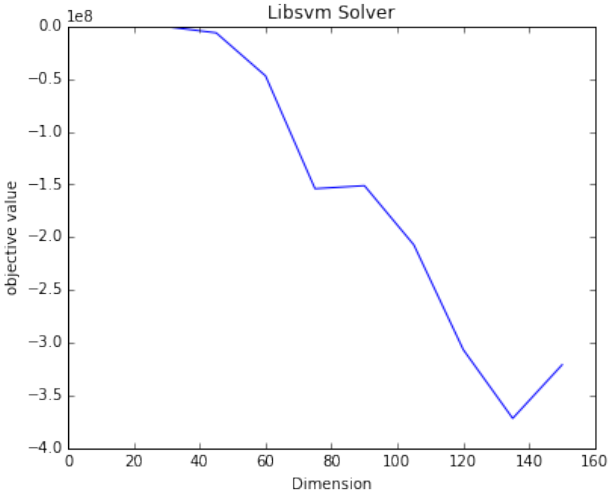
N: 800



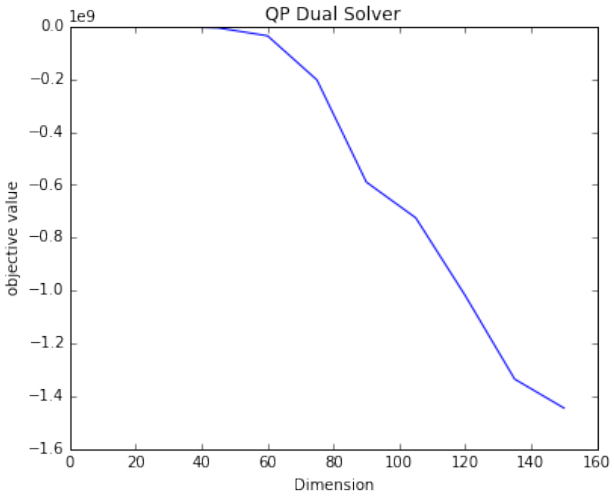


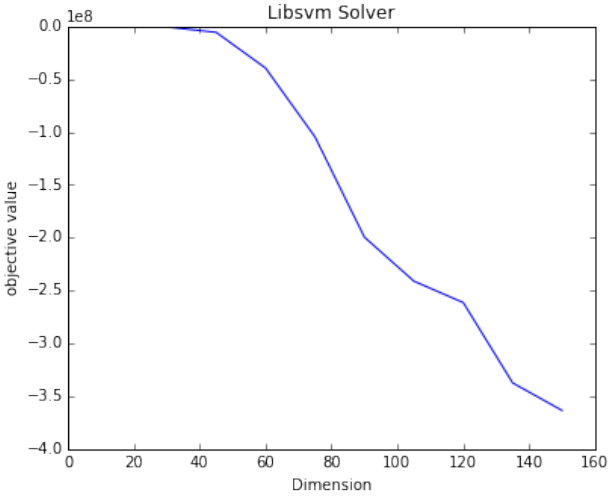
N: 900


QP Dual Solver
1e9
objective value
Dimension


Libsvm Solver
1e8
objective value
Dimension

N: 1000


QP Dual Solver
1e9
objective value
Dimension


Libsvm Solver
1e8
objective value
Dimension