pgd_lasso-question

March 6, 2021

[14]:
```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
```

0.0.1 Utility functions

[15]:
```
def obj(w):
    ## calculates the obj functions
    r = X*w-y;
    return np.sum(np.multiply(r,r))/2 +  lamda * np.sum(np.abs(w))
```

0.1 Data

[16]:
```
np.random.seed(50)

N = 100
dim = 30
lamda = 1/np.sqrt(N);

w = np.zeros(dim)
n_nonzero = 15
w[np.random.choice(range(dim), n_nonzero, False)] = np.random.randn(n_nonzero)
w = np.matrix(w.reshape(-1, 1))

X = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim), size = N))
y = X*w
```

Our objective function of interest is:

$$\frac{1}{2}\|Xw - y\|^2 + \lambda|w|_1$$

In the cell above, the variables X, y, w and lamda corresponds to X, y, w and λ in the equation above.

[17]:
```
opt = obj(w)
print('Optimal Objective Function Value: ', opt)
```

```
Optimal Objective Function Value:  1.3043384900597284
```

0.2 Optimal Value using SKLearn

[18]:
```
from sklearn import linear_model
clf = linear_model.Lasso(alpha=lamda / N, fit_intercept = False)
clf.fit(X, y)
```

[18]:
```
Lasso(alpha=0.001, fit_intercept=False)
```

[19]:
```
print('SKLearn obj val: ', obj(clf.coef_.reshape(-1, 1)) )
```

```
SKLearn obj val:  1.303641803846212
```

0.3 Proximal Gradient

[20]:
```
max_iter = 100 # max number of iterations of proximal gradient method
```

[21]:
```
alpha_array = np.logspace(-5, -2, num = 10, base = 10.0) #range over which you␣
↪search hyperparam
```

[22]:
```
def g_prox(x, step_size):
    """
    L1 regularization
    """
    return np.fmax(x - step_size * alpha, 0) - np.fmax(- x - step_size * alpha,␣
↪0)
```

[23]:
```
def f_prime(x, b, y):
    return x.T @ (x @ b - y)
```

[24]:
```
## Proximal Gadient

obj_pg = {} #stores obj function value as a function of iteration for each alpha
w_pg = {} #stores the final weight vector learned for each alpha

tol = 0.01

for alpha in alpha_array:
    print('Alpha: ', alpha)

    w_pg[alpha] = np.matrix([0.0]*dim).T
    obj_pg[alpha] = []


    for t in range(0, max_iter):
        obj_val = obj(w_pg[alpha])
        obj_pg[alpha].append(obj_val.item())

        ## fill in your code
```

```python
        ## be sure to include your stopping condition
        grad_fk = f_prime(X, w_pg[alpha], y)
        wk_grad = w_pg[alpha] - alpha * grad_fk
        prx = g_prox(wk_grad, alpha)
        w_pg[alpha] = prx

        sign = np.zeros(shape=(len(prx), 1))
        for elem in range(len(prx)):
            if prx[elem] > 0:
                sign[elem] = 1
            elif prx[elem] < 0:
                sign[elem] = -1
        if np.add(grad_fk, lamda * sign).all() < tol:
            print("Achieved relative tolerance at iteration %s" % t)
            break

        if (t%10==0):
            print('iter= {},\tobjective= {:3f}'.format(t, obj_val.item()))
```

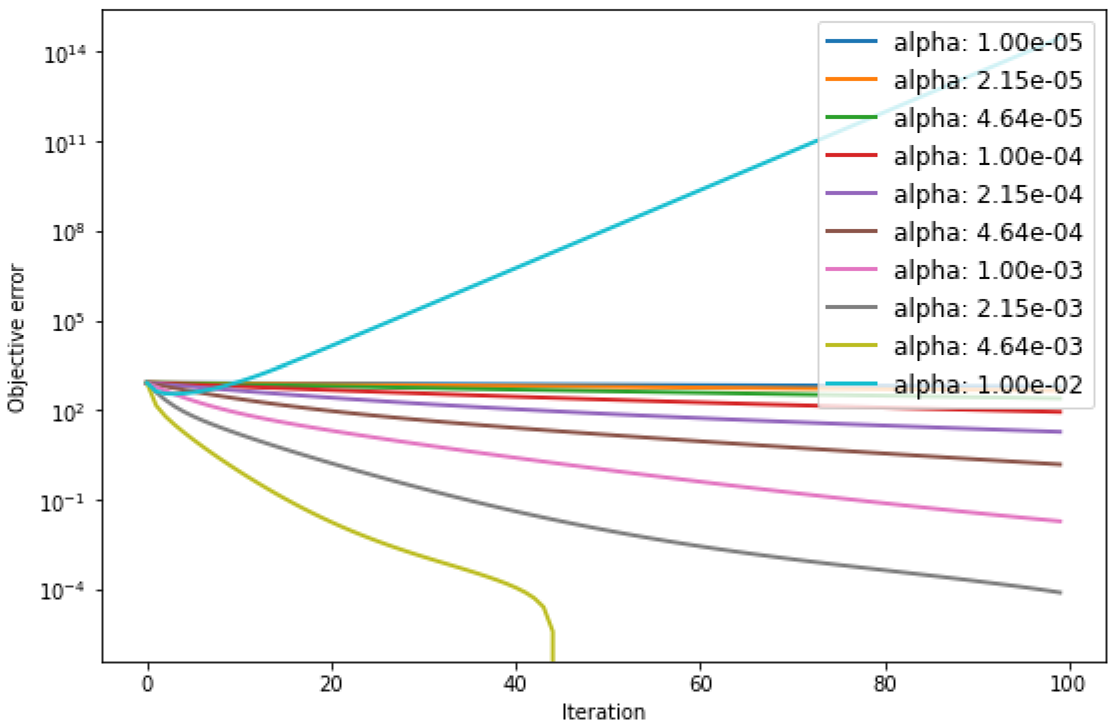
```
Alpha:  1e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 807.827723
iter= 20,       objective= 784.858060
iter= 30,       objective= 762.639041
iter= 40,       objective= 741.144378
iter= 50,       objective= 720.348736
iter= 60,       objective= 700.227700
iter= 70,       objective= 680.757744
iter= 80,       objective= 661.916192
iter= 90,       objective= 643.681189
Alpha:  2.1544346900318823e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 781.331858
iter= 20,       objective= 734.561707
iter= 30,       objective= 691.008037
iter= 40,       objective= 650.433717
iter= 50,       objective= 612.619760
iter= 60,       objective= 577.363910
iter= 70,       objective= 544.479326
iter= 80,       objective= 513.793383
iter= 90,       objective= 485.146564
Alpha:  4.641588833612782e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 727.433455
iter= 20,       objective= 638.133069
iter= 30,       objective= 561.416201
iter= 40,       objective= 495.380805
iter= 50,       objective= 438.423409
```

```
iter= 60,       objective= 389.191302
iter= 70,       objective= 346.542337
iter= 80,       objective= 309.510795
iter= 90,       objective= 277.279950
Alpha:  0.0001
iter= 0,        objective= 831.575313
iter= 10,       objective= 624.748521
iter= 20,       objective= 475.808645
iter= 30,       objective= 367.531386
iter= 40,       objective= 287.993498
iter= 50,       objective= 228.906114
iter= 60,       objective= 184.481708
iter= 70,       objective= 150.658630
iter= 80,       objective= 124.570515
iter= 90,       objective= 104.182049
Alpha:  0.00021544346900318823
iter= 0,        objective= 831.575313
iter= 10,       objective= 454.378222
iter= 20,       objective= 265.577994
iter= 30,       objective= 165.989038
iter= 40,       objective= 110.232907
iter= 50,       objective= 77.004301
iter= 60,       objective= 55.974355
iter= 70,       objective= 41.937197
iter= 80,       objective= 32.146807
iter= 90,       objective= 25.079156
Alpha:  0.00046415888336127773
iter= 0,        objective= 831.575313
iter= 10,       objective= 241.094828
iter= 20,       objective= 95.958464
iter= 30,       objective= 47.808779
iter= 40,       objective= 27.010517
iter= 50,       objective= 16.390989
iter= 60,       objective= 10.457647
iter= 70,       objective= 6.977412
iter= 80,       objective= 4.876632
iter= 90,       objective= 3.583209
Alpha:  0.001
iter= 0,        objective= 831.575313
iter= 10,       objective= 81.019724
iter= 20,       objective= 22.099363
iter= 30,       objective= 8.286735
iter= 40,       objective= 3.865040
iter= 50,       objective= 2.299016
iter= 60,       objective= 1.709656
iter= 70,       objective= 1.476937
iter= 80,       objective= 1.381248
iter= 90,       objective= 1.340408
```

```
Alpha:  0.0021544346900318822
iter= 0,        objective= 831.575313
iter= 10,       objective= 17.158520
iter= 20,       objective= 2.982339
iter= 30,       objective= 1.538040
iter= 40,       objective= 1.345306
iter= 50,       objective= 1.313666
iter= 60,       objective= 1.307106
iter= 70,       objective= 1.305362
iter= 80,       objective= 1.304776
iter= 90,       objective= 1.304527
Alpha:  0.004641588833612777
iter= 0,        objective= 831.575313
iter= 10,       objective= 2.203964
iter= 20,       objective= 1.322278
iter= 30,       objective= 1.305567
iter= 40,       objective= 1.304452
iter= 50,       objective= 1.304277
iter= 60,       objective= 1.304275
iter= 70,       objective= 1.304275
iter= 80,       objective= 1.304275
iter= 90,       objective= 1.304275
Alpha:  0.01
iter= 0,        objective= 831.575313
iter= 10,       objective= 904.886576
iter= 20,       objective= 13936.982306
iter= 30,       objective= 278153.963288
iter= 40,       objective= 5630707.199543
iter= 50,       objective= 114069346.474656
iter= 60,       objective= 2310992861.402610
iter= 70,       objective= 46819976227.000832
iter= 80,       objective= 948558929396.402710
iter= 90,       objective= 19217529660282.031250
```

[25]:
```
## Plot objective error vs. iteration (log scale)

fig, ax = plt.subplots(figsize = (9, 6))

for alpha in alpha_array:
    plt.semilogy(np.array(obj_pg[alpha])-opt,  linewidth = 2, label = 'alpha:␣
 ↪'+'{:.2e}'.format(alpha) )
plt.legend(prop={'size':12})
plt.xlabel('Iteration')
plt.ylabel('Objective error')
```

[25]:
```
Text(0, 0.5, 'Objective error')
```



From this plot, we can see as alpha remains at a very small value, the algorithm converges slowly and the objective error can hardly decrease as the iteration number increases. When alpha/step size becomes larger, the error decreases faster or the algorithm converges faster as iteration number increases. However, if we have too large a step size, i.e. in this case when alpha=0.01, the algorithm diverges.

0.4 Visualize Coefficients

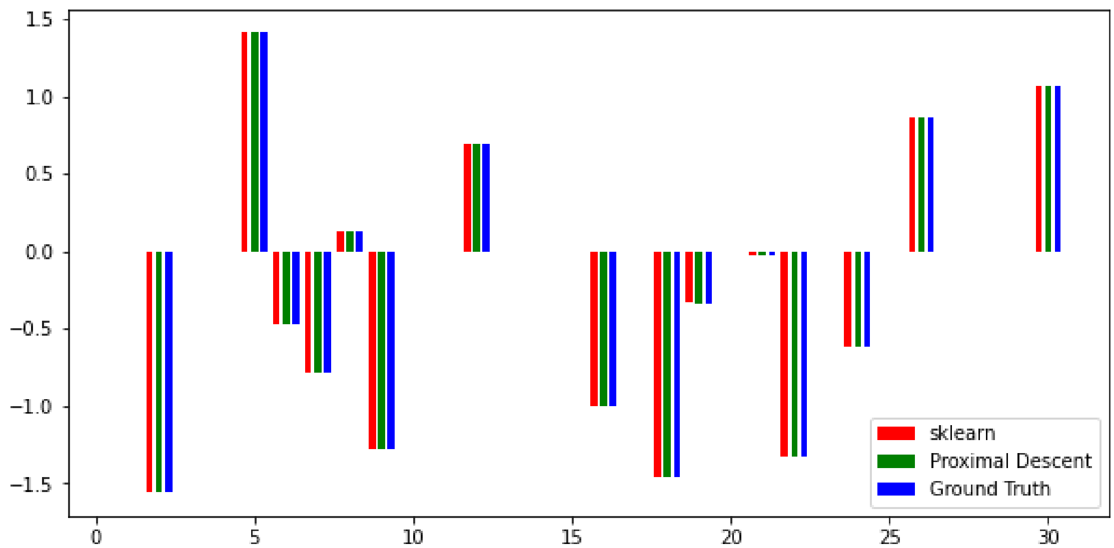
pick the coefficient corresponding to alpha value with the minimum objective function value

[34]:
```
min_obj= np.inf
min_alpha = None
```

[35]:
```
for alpha in alpha_array:
    if obj_pg[alpha][-1] < min_obj:
        min_alpha = alpha
        min_obj = obj_pg[alpha][-1]
```

[36]:
```
plt.figure(figsize = (10, 5))

ax = plt.subplot(111)

x = np.arange(1, dim+1)
```

```
ax.bar(x-0.3, clf.coef_, width=0.2, color='r', align='center', label =␣
 ↪'sklearn')
ax.bar(x, np.ravel(np.array(w_pg[min_alpha])), width=0.2, color='g',␣
 ↪align='center', label = 'Proximal Descent')
ax.bar(x+0.3, np.ravel(np.array(w)), width=0.2, color='b', align='center',␣
 ↪label = 'Ground Truth')

plt.legend()

plt.show()
```



This plot shows the comparison of coefficients from 3 different methods. Proximal descent result of the smallest obj value and sklearn both give close result to the ground truth. Notably, because we stop the proximal descent algorithm when the subgradient is close to 0 within an 1% error, the minimal objective value here corresponds to the best estimated weight.