Q2

July 14, 2020

[15]:
```
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
import cvxpy as cp
import pandas as pd
import time
from collections import defaultdict
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
ratings = loadmat('ratings.mat')['M0']
ratings_missing = loadmat('ratings_missing.mat')['M1']
```

[3]:
```
print(ratings.shape)
ratings
```

```
(200, 100)
```

[3]:
```
array([[5, 5, 5, …, 5, 5, 5],
       [5, 5, 5, …, 5, 5, 5],
       [5, 5, 5, …, 5, 5, 5],
       …,
       [3, 4, 6, …, 5, 5, 4],
       [7, 6, 4, …, 6, 5, 6],
       [4, 4, 7, …, 5, 6, 5]], dtype=uint8)
```

[4]:
```
print(ratings_missing.shape)
ratings_missing
```

```
(200, 100)
```

[4]:
```
array([[0, 5, 5, …, 5, 5, 0],
       [5, 0, 0, …, 5, 0, 5],
       [5, 5, 5, …, 5, 0, 0],
       …,
```

```
       [0, 4, 6, …, 5, 5, 0],
       [7, 6, 4, …, 0, 0, 6],
       [4, 4, 7, …, 0, 6, 0]], dtype=uint8)
```

0.1 Method 1

Solve the following optimization problem:

$$min_m ||M||_* \tag{1}$$

subject to

$$M(i,j) = M_1(i,j) \quad \forall (i,j) \ \in \ \text{observed set} \tag{2}$$

[5]:
```
def optimize(X):
    m,n = X.shape

    known_value_indices = tuple(zip(*np.argwhere(X).tolist()))
    known_values = ratings_missing[X!=0].tolist()

    M = cp.Variable((m,n))

    objective_fn = cp.normNuc(M)

    constraints = [
    M[known_value_indices] == known_values
    ]

    problem = cp.Problem(cp.Minimize(objective_fn), constraints)

    problem.solve()

    return M, problem.value
```

[6]:
```
def reconstruction_error(M, M0):
    return np.linalg.norm(M-M0, 'fro') / np.linalg.norm(M0, 'fro')
```

[7]:
```
%%timeit
optimize(ratings_missing)
```

```
6.85 s ± 22 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[8]:
```
M, v = optimize(ratings_missing)
print('Reconstruction Error: {}'.format(reconstruction_error(M.value, ratings)))
```

```
Reconstruction Error: 0.03408590631339972
```

[9]:
```
print('Optimal Value: {}'.format(round(v,2)))
```

```
Optimal Value: 965.47
```

Method 1 Results Pure optimization took about 6.84s to reach an optimal value of ~965.47 and a reconstruction error of approximately 3.41%

0.2 Method 2

[10]:
```
def svt(X, delta, tau, iters):
    Y = np.zeros_like(X)
    X0 = np.copy(X)

    err = np.zeros((iters,1))

    mask = X == 0

    for i in range(iters):
        U, S, V = np.linalg.svd(Y, full_matrices=False)
        S_t = np.diag((S-tau))
        S_t[S_t < 0] = 0
        Z = U @ S_t @ V
        P = X-Z
        P[mask] = 0
        Y0 = Y.copy()
        Y = Y0 + delta * P

    return Z
```

[11]:
```
iterations = (1_000, 2_000)
delta_taus = ((0.1, 50), (2, 50), (0.1, 500), (2,500))
```

[12]:
```
results = []
for i in iterations:
    for params in delta_taus:
        tick = time.time()
        z = svt(ratings_missing, *params, i)
        tock = time.time()

        runtime = tock-tick

        error = reconstruction_error(z, ratings)

        results.append((i, *params, error, runtime))
```

[13]:
```python
results_dict=defaultdict(list)
for tup in results:
    results_dict['iterations'].append(tup[0])
    results_dict['delta'].append(tup[1])
    results_dict['tau'].append(tup[2])
    results_dict['error'].append(round(tup[3],3))
    results_dict['runtime'].append(tup[4])
```

[14]:
```python
pd.DataFrame(results_dict).sort_values(['error','runtime'],␣
 ↪ascending=[True,True]).reset_index().drop('index',axis=1)
```

[14]:

	iterations	delta	tau	error	runtime
0	1000	2.0	500	0.037	6.534128
1	2000	2.0	500	0.037	15.875451
2	2000	0.1	500	0.039	14.890137
3	1000	0.1	500	0.046	6.574397
4	1000	2.0	50	0.339	6.729908
5	1000	0.1	50	0.339	6.751017
6	2000	0.1	50	0.339	13.479637
7	2000	2.0	50	0.339	13.888789

Method 2 Results With a delta of 2.0 and atau of 500, both 1000 and 2000 iterations approximately gave the same reconstruction error of ~3.7%. However at 2000 iterations, run time for the solution was double that of the 1000 iteration for pretty much the same error. Making the best choice for this method being row 0 above

- 1000 iters
- delta = 2.0
- tau = 500

0.3 Final Results (c)

Overall, pure optimization, method 1, proved to be superior with a smaller reconstruction error at approximately 1/10th of a second slower than the best parameters for method 2. However this is a relatively small matrix at a dimension of only (200,100). It would be interesting to see conergence time on a much larger matrix for both methods

[ ]:
```python

```