`CompleteThat`: A python package for low-rank matrix completion

Joshua Edgerton Esteban Fajardo

January 5, 2014

Abstract

We have developed a Python package (`CompleteThat`) to perform low rank matrix completion. Given a low rank matrix with partial entries we implemented different methods to solve the matrix completion problem: First, for "small" problems we implemented two memory-based algorithms following the work by Tanner and Wei [TW14] to find, given a number r, the matrix with rank r that best fits the data using the Frobenius norm. Second, when the problem does not fit into memory, we implemented a memory-fitting algorithm (stochastic gradient descent) following the approach in Zhang [Zha04] and Bottou [Bot12]. The package is available at `https://pypi.python.org/pypi/completethat/0.1dev`.

Contents

1 Introduction

Matrix factorization has recently seen a large growth in popularity within the mathematics, statistics, and computer science communities as industry continues to apply machine learning techniques on a wide array of problems and scenarios. For our convex optimization project, we decided to take some of the more interesting and applicable topics and algorithms of our class and develop a python package to implement them. We briefly discuss the theory behind matrix factorizing, the models and approaches we choose to use and the algorithms for the optimization. Later, we walk through two case studies illustrating the usefulness and applicability in the context of image process and a recommendation system for Yahoo music and movie data.

2 Matrix Completion Problem

Consider the problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|Ax-b\|_2^2 + \lambda\|x\|_1 \\ \text{subject to} & 0 \preceq x \preceq \mathbf{1} \\ & \|x\|_2 \leq 1, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ is the optimization variable, and $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, and $\lambda > 0$ are problem data.

3 Implementation

`CompleteThat` is a python package that solves the low rank matrix completion problem. Given a low rank matrix with partial entries the package solves an optimization problem to estimate the missing entries. We allow the user to choose between several optimization algorithms including two in memory based algorithms, alternating steepest descent and scaled alternating steepest descent, and one out of memory fitting procedure using stochastic gradient descent. We use extensively the numerical libraries `spicy` and `numpy`, and the current implementation includes two classes, `MatrixCompletion` for in memory based algorithms and `MatrixCompletionBD` for the memory fitting procedure. The package is available worldwide and can be downloaded from `https://pypi.python.org/pypi/completethat/0.1dev`.

CVX

Problem (1) can be easily solved directly using CVX, a package for specifying and solving convex programs [GB14], [GB08], with the following code:

```
index = find(~isnan(M));
```

```
cvx_begin
    variable X(size(M));
    minimize norm_nuc(X)
    % s.t.
    X(index) == M(index);
cvx_end
```

Where M is a MATLAB matrix with nan on the missing entries and `X(index) == M(index)` corresponds to the $P_\Omega(X) = P_\Omega(M)$ restriction. However, the computation is very slow and for any matrix greater than 100x100 the computational time is too high. Since this is clearly unsatisfactory for practical purposes, specific algorithms were developed and used on the package.

Algorithms

Low rank matrix completion by alternating steepest descent methods

Low rank matrix completion by stochastic gradient descent

Code Example

For matrices that fit into memory use the `MatrixCompletion` module. Otherwise, use `MatrixCompletionBD` module.

MatrixCompletion

Given a `numpy` matrix M with `numpy.nan` on the missing entries, the matrix completion problem can be solved (using ASD) as:

```
>>> from completethat import MatrixCompletion
>>> problem = MatrixCompletion(M)
>>> problem.complete_it("ASD")
>>> X = problem.get_matrix() #Desired matrix
>>> out_info = problem.get_out() #Extra information
```

MatrixCompletionBD

Given a csv file with the input data, the matrix completion problem can be solved as:

```
>>> from completethat import MatrixCompletionBD
>>> problem = MatrixCompletionBD('input_data.txt')
>>> problem.train_sgd(dimension=6,init_step_size=.01,min_step=.000001,
       reltol=.001,rand_init_scale=10, maxiter=1000,
       batch_size_sgd=50000,shuffle=True)
>>> problem.validate_sgd('test_data.txt')
>>> problem.save_model()
```

4 Case Studies

Yahoo movies and music reviews database

The Yahoo music and movie data sets are publicly available datasets published by Yahoo for use to the public for recreational and research purposes. The music training data set is a small 4 mb pipe-delimited file consisting of roughly 210,000 user-movie ratings. The test set has roughly 10,000 records. There are two ratings schemas available, one on a 5-point scale and the other on a 13-point scale and we arbitrarily chose to use the 5-point scale. The Yahoo music data is a larger 2.2 gb file of 115 million user-song ratings. The ratings are on a 100 point scale.

We look to evaluate the performance of our algorithms on these two datasets, comparing the various algorithms implemented versus a naive benchmark estimate where we used the mean of the training set as our estimator for all records in the test set. Then we compared how well the algorithms performed versus the benchmark.

The benchmark for the yahoo music data is 4.088. This is the mean value of the ratings in the training set so we will compute the MSE on the test set using the benchmark which yields an RMSE of 1.10. A quick run of Mahout yields 1.17 and CompleteThat's sgd model yielded a 1.19. So our model yielded a pretty similar RMSE as the MAHOUT package. Note that for both of these instances we used a factorization of rank equal to seven and the Complete that sgd algorithm ran slightly faster than Mahout's (.537 minutes vs 1.4 minutes).

Step size	Iterations	Minutes	Train MSE	Test RMSE
1	8	0.585	1.411	1.757
2	8	0.598	1.304	1.550
3	8	0.628	1.235	1.483
4	8	0.617	1.180	1.397
5	7	0.538	1.136	1.351
6	7	0.534	1.100	1.244
7	7	0.534	1.071	1.194
8	7	0.541	1.046	1.127
9	7	0.537	1.027	1.128
10	7	0.529	1.010	1.102
15	7	0.551	0.971	1.063
20	7	0.540	0.976	1.075
30	8	0.614	1.045	1.359

Table 1: RMSE vs Step Size vs Rank

Columbia University images

Figure 1: ASD method applied three different Columbia University images

5 Future Work

We have plans for various improvements as we continue working with the `CompleteThat` python package. Overall, we would like to test the software and integrate automated testing cases into the software developing cycle. Also, documentation for the package and its functions, including examples, is high in our priority list. In addition, we hope to incorporate more rigorous error checking and make available more customization to the user.

For the memory-based algorithms it is well know that noise in the data introduces overfitting on the estimated matrix. Following the approach taken by Mazumder et al. [MHT10] where they solve the same objetive function as the problems above (using the Frobenious norm) but introducing the nuclear norm as regularizer to account for overfitting, we would introduce a regularization option into the matrix completion procedure.

For the stochastic gradient descent method, we plan to add a lambda penalty feature for combatting overfitting as well as considering options for more advanced versions of our basic latent factor model as demonstrated by Koren [Kor08] in his paper on the prize-winning Netflix models. For all algorithms, we would like to explore and research various hardware and software optimization techniques for faster code.

Finally, given the similarities between matrix completion and robust principal component analysis one further extension to the functionality of the package could be implementing the robust-PCA procedure outlined by Candés [CLMW11], et al.

References

[Bot12] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CLMW11] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[GB08] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[GB14] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[Kor08] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[MHT10] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322, 2010.

[Tre08] L. N. Trefethen. Is Gauss quadrature better than Clenshaw-Curtis? *SIAM Review*, 50:67–87, 2008.

[TW14] J. Tanner and K. Wei. Low rank matrix completion by alternating steepest descent methods. Technical report, SIAM, SIAM J. Imaging Sciences, 2014.

[Zha04] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

6 Source code

On the whole, the package directory structure looks like the Figure 2. In the following we present the source code for the package.

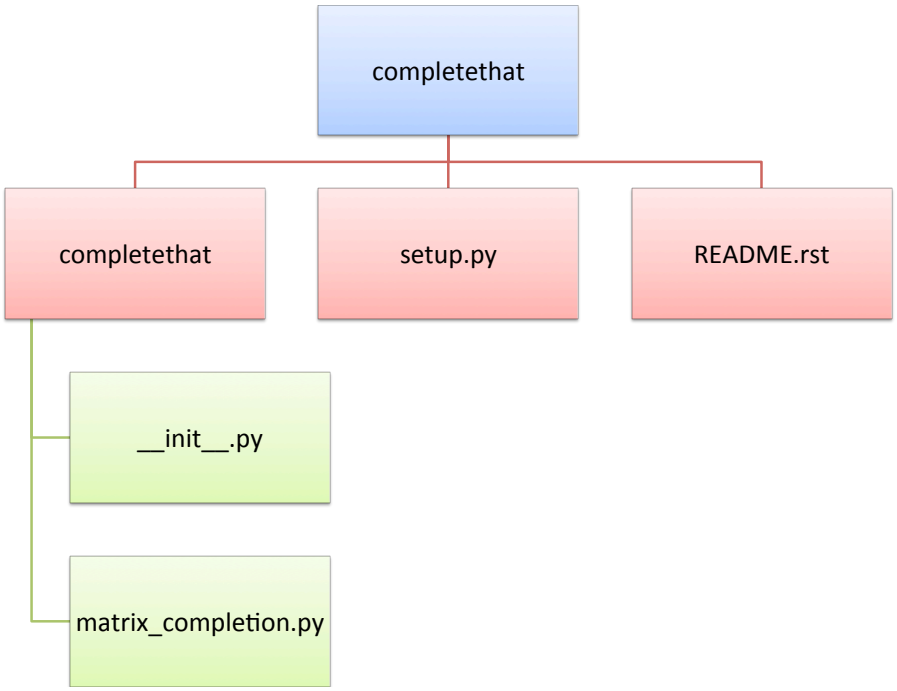

Figure 2: CompleteThat structure

---

completethat/matrix_completion.py

```
1  import numpy as np
2  from scipy.sparse import csc_matrix
3  from scipy.sparse import linalg as linalg_s
4  import os, random, time
5  from scipy import linalg
6
7  class MatrixCompletion:
8      """ A general class to represent a matrix completion problem
9
10     Data members
11     ============
12     M:= data matrix (numpy array).
13     X:= optimized data matrix (numpy array)
```

```
14      out_info:= output information for the optimization (list)
15
16
17      Class methods
18      ==================
19      complete_it():= method to complete the matrix
20      get_optimized_matrix():= method to get the solution to the problem
21      get_matrix():= method to get the original matrix
22      get_out():= method to get extra information on the optimization (iter
23      number, convergence, objective function)
24
25      """
26      def __init__(self, X,*args, **kwargs):
27          """ Constructor for the problem instance
28
29              Inputs:
30               1) X: known data matrix. Numpy array with np.nan on the unknow
        entries.
31                  example:
32                      X = np.random.randn(5, 5)
33                      X[1][3] = np.nan
34                      X[0][0] = np.nan
35                      X[4][4] = np.nan
36
37          """
38          # Initialization of the members
39          self._M = X
40          self._X = np.array(X, copy = True) #Initialize with ini data matrix
41          self._out_info = []
42
43      def get_optimized_matrix(self):
44          """ Getter function to return the optimized matrix X
45
46              Ouput:
47               1) Optimized matrix
48          """
49          return self._X
50
51      def get_matrix(self):
52          """ Getter function that returns the original matrix M
53
54              Output:
55              1) Original matrix M
56          """
57          return self._M
58
59      def get_out(self):
60          """ Getter function to return the output information
61              of the optimization
62
63              Output:
```

```
64              1) List of length 2: number of iterations and relative residual
65
66          """
67          return self._out_info
68
69
70      def _ASD(self, M, r = None, reltol=1e-5, maxiter=5000):
71          """
72          Alternating Steepest Descent (ASD)
73          Taken from Low rank matrix completion by alternating steepest descent
        methods
74          Jared Tanner and Ke Wei
75          SIAM J. IMAGING SCIENCES (2014)
76
77          We have a matrix M with incomplete entries,
78          and want to estimate the full matrix
79
80          Solves the following relaxation of the problem:
81          minimize_{X,Y} \frac{1}{2} ||P_{\Omega}(Z^0) - P_\{Omega}(XY)||_F^2
82          Where \Omega represents the set of m observed entries of the matrix M
83          and P_{\Omega}() is an operator that represents the observed data.
84
85          Inputs:
86           M := Incomplete matrix, with NaN on the unknown matrix
87           r := hypothesized rank of the matrix
88
89          Usage:
90           Just call the function _ASD(M)
91          """
92
93          # Get shape and Omega
94          m, n = M.shape
95          if r == None:
96              r = min(m, n, 50)
97
98          # Set relative error
99          Omega = ~np.isnan(M)
100         frob_norm_data = linalg.norm(M[Omega])
101         relres = reltol * frob_norm_data
102
103         # Initialize
104         I, J = np.where(Omega)
105         M_omega =  csc_matrix((M[Omega], (I, J)), shape=M.shape)
106         U, s, V = linalg_s.svds(M_omega, r)
107         S = np.diag(s)
108         X = np.dot(U, S)
109         Y = V
110         itres = np.zeros((maxiter+1, 1))
111
112         XY = np.dot(X, Y)
113         diff_on_omega = M[Omega] - XY[Omega]
```

```
114         res = linalg.norm(diff_on_omega)
115         iter = 0
116         itres[iter] = res/frob_norm_data
117 
118         while iter < maxiter and res >= relres:
119 
120             # Gradient for X
121             diff_on_omega_matrix = np.zeros((m,n))
122             diff_on_omega_matrix[Omega] = diff_on_omega
123             grad_X = np.dot(diff_on_omega_matrix, np.transpose(Y))
124 
125             # Stepsize for X
126             delta_XY = np.dot(grad_X, Y)
127             tx = linalg.norm(grad_X,'fro')**2/linalg.norm(delta_XY)**2
128 
129             # Update X
130             X = X + tx*grad_X;
131             diff_on_omega = diff_on_omega-tx*delta_XY[Omega]
132 
133             # Gradient for Y
134             diff_on_omega_matrix = np.zeros((m,n))
135             diff_on_omega_matrix[Omega] = diff_on_omega
136             Xt = np.transpose(X)
137             grad_Y = np.dot(Xt, diff_on_omega_matrix)
138 
139             # Stepsize for Y
140             delta_XY = np.dot(X, grad_Y)
141             ty = linalg.norm(grad_Y,'fro')**2/linalg.norm(delta_XY)**2
142 
143             # Update Y
144             Y = Y + ty*grad_Y
145             diff_on_omega = diff_on_omega-ty*delta_XY[Omega]
146 
147             res = linalg.norm(diff_on_omega)
148             iter = iter + 1
149             itres[iter] = res/frob_norm_data
150 
151         M_out = np.dot(X, Y)
152 
153         out_info = [iter, itres]
154 
155         return M_out, out_info
156 
157     def _sASD(self, M, r = None, reltol=1e-5, maxiter=10000):
158         """
159         Scaled Alternating Steepest Descent (ScaledASD)
160         Taken from:
161         Low rank matrix completion by alternating steepest descent methods
162         Jared Tanner and Ke Wei
163         SIAM J. IMAGING SCIENCES (2014)
164 
```

```
        We have a matrix M with incomplete entries,
        and want to estimate the full matrix

        Solves the following relaxation of the problem:
        minimize_{X,Y} \frac{1}{2} ||P_{\Omega}(Z^0) - P_\{Omega}(XY)||_F^2
        Where \Omega represents the set of m observed entries of the matrix M
        and P_{\Omega}() is an operator that represents the observed data.

        Inputs:
         M := Incomplete matrix, with NaN on the unknown matrix
         r := hypothesized rank of the matrix

        Usage:
         Just call the function _sASD(M)
        """


        # Get shape and Omega
        m, n = M.shape
        if r == None:
            r = min(m, n, 50)

        # Set relative error
        Omega = ~np.isnan(M)
        frob_norm_data = linalg.norm(M[Omega])
        relres = reltol * frob_norm_data

        # Initialize
        identity = np.identity(r);
        I, J = np.where(Omega)
        M_omega =  csc_matrix((M[Omega], (I, J)), shape=M.shape)
        U, s, V = linalg_s.svds(M_omega, r)
        S = np.diag(s)
        X = np.dot(U, S)
        Y = V
        itres = np.zeros((maxiter+1, 1))

        XY = np.dot(X, Y)
        diff_on_omega = M[Omega] - XY[Omega]
        res = linalg.norm(diff_on_omega)
        iter = 0
        itres[iter] = res/frob_norm_data

        while iter < maxiter and res >= relres:

            # Gradient for X
            diff_on_omega_matrix = np.zeros((m,n))
            diff_on_omega_matrix[Omega] = diff_on_omega
            grad_X = np.dot(diff_on_omega_matrix, np.transpose(Y))

            # Scaled gradient
```

```
216             scale = linalg.solve(np.dot(Y, np.transpose(Y)), identity)
217             dx = np.dot(grad_X, scale)
218 
219             delta_XY = np.dot(dx, Y)
220             tx = np.trace(np.dot(np.transpose(dx),grad_X))/linalg.norm(
    delta_XY[Omega])**2
221 
222             # Update X
223             X = X + tx*dx
224             diff_on_omega = diff_on_omega-tx*delta_XY[Omega]
225 
226             # Gradient for Y
227             diff_on_omega_matrix = np.zeros((m,n))
228             diff_on_omega_matrix[Omega] = diff_on_omega
229             Xt = np.transpose(X)
230             grad_Y = np.dot(Xt, diff_on_omega_matrix)
231 
232             # Scaled gradient
233             scale = linalg.solve(np.dot(Xt, X), identity)
234             dy = np.dot(scale, grad_Y)
235 
236             # Stepsize for Y
237             delta_XY = np.dot(X, dy)
238             ty = np.trace(np.dot(dy,np.transpose(grad_Y)))/linalg.norm(
    delta_XY[Omega])**2
239 
240             # Update Y
241             Y = Y + ty*dy
242             diff_on_omega = diff_on_omega-ty*delta_XY[Omega]
243 
244             # Update iteration information
245             res = linalg.norm(diff_on_omega)
246             iter = iter + 1
247             itres[iter] = res/frob_norm_data
248 
249         M_out = np.dot(X, Y)
250 
251         out_info = [iter, itres]
252 
253         return M_out, out_info
254 
255     def complete_it(self, algo_name, r = None, reltol=1e-5, maxiter=5000):
256 
257         """ Function to solve the optimization with the choosen algorithm
258 
259             Input:
260              1) algo_name: Algorithm name (ASD, sASD, ect)
261              2) r: rank of the matrix if performing alternating algorithm
262         """
263         if algo_name == "ASD":
264             self._X, self._out_info = self._ASD(self._M, r, reltol, maxiter)
```

```
265             elif algo_name == "sASD":
266                 self._X, self._out_info = self._sASD(self._M, r, reltol, maxiter)
267             else:
268                 raise NameError("Algorithm name not recognized")
269 
270 class MatrixCompletionBD:
271     """
272     A general class for matrix factorization via stochastic gradient descent
273 
274     Class members
275     ==================
276     file: three column file of user, item, and value to build models
277 
278 
279     Class methods
280     ==================
281     train_sgd():= method to complete the matrix via sgd
282     shuffle_file():= method to 'psuedo' shuffle input file in chunks
283     file_split():= method to split input file into training and test set
284     save_model():= save user and items parameters to text file
285     validate_sgd():= validate sgd model on test set
286     build_matrix():= for smaller data build complete matrix in pandas df or
        numpy matrix?
287     """
288 
289 
290     def __init__(self,file_path,delimitter='\t',*args, **kwargs):
291         """
292             Object constructor
293             Initialize Matrix Completion BD object
294         """
295         self._file = file_path
296         self._delimitter = '\t'
297         self._users = dict()
298         self._items = dict()
299 
300     def shuffle_file(self,batch_size=50000):
301         """
302 
303         Shuffle line of file for sgd method, improves performance/convergence
304 
305         """
306         data = open(self._file)
307         temp_file=open('temp_shuffled.txt','w')
308         try:
309             temp=open('backup_data_file.txt')
310             temp.close()
311         except:
312             os.system('cp ' +self._file + ' backup_data_file.txt')
313 
314         temp_array=[]
```

```
315         counter=0
316         for line in data:
317             counter+=1
318             temp_array.append(line)
319             if counter==batch_size :
320                 random.shuffle(temp_array)
321                 for entry in temp_array:
322                     temp_file.write(entry)
323                 temp_array=[]
324                 counter=0
325 
326         if len(temp_array)>0:
327             random.shuffle(temp_array)
328             for entry in temp_array:
329                 temp_file.write(entry)
330 
331         data.close()
332         temp_file.close()
333         system_string='mv temp_shuffled.txt ' + self._file
334         os.system(system_string)
335 
336     def file_split(self,percent_train=.80, train_file='data_train.csv',
    test_file='data_test.csv'):
337         """
338 
339         split input file randomly into training and test set for cross
    validation
340 
341         """
342         train=open(train_file,'w')
343         test=open(test_file,'w')
344         temp_file=open(self._file)
345         for line in temp_file:
346             if np.random.rand()<percent_train:
347                 train.write(line)
348             else:
349                 test.write(line)
350 
351         train.close()
352         test.close()
353         print('test file written as ' + train_file)
354         print('test file written as ' + test_file)
355         temp_file.close()
356 
357     def train_sgd(self,dimension=6,init_step_size=.01,min_step=1e-5,reltol
    =.05,rand_init_scalar=1, maxiter=100,batch_size_sgd=50000,shuffle=True,
    print_output=False):
358 
359         init_time=time.time()
360         alpha=init_step_size
361         iteration=0
```

```
362         delta_err=1
363         new_mse=reltol+10
364         counter=0
365         ratings=[]
366 
367         while iteration != maxiter and delta_err > reltol :
368 
369             data=open(self._file)
370             total_err=[0]
371             if alpha>=min_step: alpha*=.3
372             else: alpha=min_step
373 
374             for line in data:
375 
376                 record=line[0:len(line)-1].split(self._delimitter)
377                 record[2]=float(record[2])
378                 # format : user, movie,5-point-ratings
379                 ratings.append(record[2])
380                 #if record[0] in self.users and record[1] in self.items :
381                 try:
382                     # do some updating
383                     # updates
384                     error=record[2]-np.dot(self._users[record[0]],self._items[
    record[1]])
385                     self._users[record[0]]=self._users[record[0]]+alpha*2*
    error*self._items[record[1]]
386                     self._items[record[1]]=self._items[record[1]]+alpha*2*
    error*self._users[record[0]]
387                     total_err.append(error**2)
388                 except:
389                     #else:
390                     counter+=1
391                     if record[0] not in self._users:
392                         self._users[record[0]]=np.random.rand(dimension)*
    rand_init_scalar
393                     if record[1] not in self._items:
394                         self._items[record[1]]=np.random.rand(dimension)*
    rand_init_scalar
395 
396             data.close()
397             if shuffle:
398                 self.shuffle_file(batch_size=batch_size_sgd)
399             iteration+=1
400             old_mse=new_mse
401             new_mse=sum(total_err)*1.0/len(total_err)
402             delta_err=abs(old_mse-new_mse)
403             if print_output and iteration%10==0:
404                 print ('Delta Error: %f ' % delta_err)
405 
406         #Printing Final Output
407         if print_output:
```

```
408             print ('Iterations: %f ' % iteration)
409             print ('MSE: %f ' % new_mse)
410             minutes=(time.time()-init_time)/60
411             print ('Total Minutes to Run: %f' % minutes)
412
413
414     def save_model(self,user_out='user_params.txt',item_out='item_params.txt')
        :
415         """
416         save model user and item parameters to text file
417         user_key, user_vector entries
418         item_key, item_vector entries
419         """
420         users=open(user_out,'w')
421         items=open(item_out,'w')
422         for key in self._users:
423             user_string= key+ self._delimitter + self._delimitter.join(map(str
        ,list(self._users[key]))) + '\n'
424             users.write(user_string)
425
426         for key in self._items:
427             item_string=key+ self._delimitter + self._delimitter.join(map(str,
        list(self._items[key]))) + '\n'
428             items.write(item_string)
429
430         users.close()
431         items.close()
432
433     ## read saved model, particularly useful for fitting very large files!
434     def read_model(self,dimension=6,saved_user_params='user_params.txt',
        saved_item_params='item_params.txt'):
435         """
436
437         Read the saved user and item parameters from text files to the item
        and user dictionaries
438
439         """
440         #populate users:
441         user_data=open(saved_user_params)
442         for line in user_data:
443             record=line[0:len(line)-1].split(self._delimitter)
444             key=record.pop(0)
445             params=np.array(map(float,record))
446             self._users[key]=params
447
448         user_data.close()
449
450         #populate items:
451         item_data=open(saved_item_params)
452         for line in item_data:
453             record=line[0:len(line)-1].split(self._delimitter)
```

```
454                 key=record.pop(0)
455                 params=np.array(map(float,record))
456                 self._items[key]=params
457 
458         item_data.close()
459 
460     def clear_model(self):
461         """
462 
463         clear the user and item parameters
464 
465         """
466         del self._items, self._users
467         self._items=dict()
468         self._users=dict()
469 
470     def validate_sgd(self,test_file_path):
471         """
472 
473         run model on test/validation set, returns MSE
474 
475         """
476         mse=[]
477         counter=0
478         test_set=open(test_file_path)
479         for line in test_set:
480             record=line[0:len(line)-1].split(self._delimitter)
481             record[2]=float(record[2])
482             try:
483                 error=record[2]-np.dot(self._users[record[0]],self._items[
    record[1]])
484                 mse.append(error**2)
485             except:
486                 counter+=1
487 
488         if counter>0: print('Items/Users Key Errors: %f ' % counter )
489         # returns Mean Squared Error
490         return sum(mse)/len(mse)
491 
492     def build_matrix(self):
493         pass
```

---

completethat/matrix__init__.py

```
1 """ CompleteThat is a python package that solves the low rank matrix
    completion
2 problem. Given a low rank matrix with partial entries the package solves an
3 optimization problem to estimate the missing entries.
```

```
4
5  Mathematically, the package solves a relaxation (using the nuclear norm or the
6  Frobenius norm of the objective matrix) of the following problem:
7      minimize_{X} ||X||
8      st. X(i,j) = M(i,j) \forall (i,j)\in \Omega,
9
10 Where, M represents the data matrix and \Omega represents the set of p
       observed entries of M
11
12     Usage:
13     # MatrixCompletion
14     >>> from completethat import MatrixCompletion
15     >>> problem = MatrixCompletion(M)
16     >>> problem.complete_it(algo_name)
17     >>> X = problem.get_matrix()
18     >>> out_info = problem.get_out() #Extra info (iterations, ect)
19
20     # MatrixCompletionBD
21     >>> from completethat import MatrixCompletionBD
22     >>> temp=MatrixCompletionBD('input_data.txt')
23     >>> temp.train_sgd(dimension=6,init_step_size=.01,min_step=.000001, reltol
       =.001,rand_init_scale=10,    maxiter=1000,batch_size_sgd=50000,shuffle=True
       ):
24     >>> temp.validate_sgd('test_data.txt')
25     >>> temp.save_model()
26
27 """
28 from matrix_completion import MatrixCompletion
29 from matrix_completion import MatrixCompletionBD
```

---

completethat/matrix__init__.py

```
1  from setuptools import setup
2
3  def readme():
4      with open('README.rst') as f:
5          return f.read()
6
7  setup(
8        name='completethat',
9        version='0.1dev',
10       description='A package to solve low rank matrix completion problems',
11       long_description=readme(),
12       author='Joshua Edgerton, Esteban Fajardo',
13       author_email='ef2451@columbia.edu, jae2154@columbia.edu',
14       license='BSD',
15       packages=['completethat'],
16       install_requires=[
```

```
17              'scipy', 'numpy'
18          ],
19          classifiers=[
20              'Development Status :: 3 - Alpha',
21              'License :: OSI Approved :: BSD License',
22              'Programming Language :: Python :: 2.7',
23              'Topic :: Scientific/Engineering :: Mathematics',
24              'Topic :: Utilities'
25          ],
26          include_package_data=True,
27          zip_safe=False
28  )
```

Punctuation in equations. An equation is part of a sentence, and you may need to include a comma or a period at the end of an equation as a result, whether or not you are using inline or display math style. For example:

We next discuss how to solve the problem

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2,$$

where $x \in \mathbf{R}^n$ is the optimization variable.

Don't start a sentence with a symbol. This hurts readability:

Bad: f is smooth.
Good: The function f is smooth.

Bad: $x^n - a$ has n distinct zeros.
Good: The polynomial $x^n - a$ has n distinct zeros.

Similarly, don't start a sentence with a reference.

Use words to separate symbols in different formulas. If it might confuse the reader visually or in the actual meaning of the sentence, use words to break apart formulas:

Bad: The sequences $x_1, x_2, \ldots, y_1, y_2, \ldots$ are Cauchy.
OK: The sequences $x_1, x_2, \ldots,$ and $y_1, y_2, \ldots,$ are Cauchy.
Good: The sequences (x_i) and (y_i) are Cauchy.

OK: The image of S under f, $f(S) = \{x \mid x \in S\}$, is convex.
Good: The image of S under f, given by $f(S) = \{x \mid x \in S\}$, is convex.

Do not insert superfluous words if the meaning is clear.

Good: Consider the function $f + g + h$, where $f : \mathbf{R}^n \to \mathbf{R}$, $g : \mathbf{R}^m \to \mathbf{R}$, and $h : \mathbf{R}^p \to \mathbf{S}^n$ are closed proper convex.

References. For internal references, use the `label` and `ref` commands. *Never* number or refer to an entity using a specific number, as in "Table 3." Refer to "Table **??**", "(2)" (an equation), and so on. Only number equations that are important and should be emphasized or that you specifically refer to elsewhere. Do not simply number all equations, and do not number them randomly. A numbered equation draws the reader's attention, so it should be used relatively sparingly. Use `\S` for section references, as in §6.

For external references, you must use BibTEX. Use references like nouns or footnotes:

Good: One useful reference is [BV04].
Good: Interested readers may refer to Boyd and Vandenberghe [BV04].

Your BibTEX source must be correct. Unfortunately, many systems that export BibTEX entries export very poor ones. BibTEX ignores the `*.bib` file's capitalization for articles (but not books). To force capitalization, you must wrap the letter with curly braces. The hyphen (-), en dash (–), and em dash (—) are distinct punctuation marks that serve different purposes. (And the minus sign is different from all three.) When specifying a page range in references, use the en dash. For an example of escaping and using the en dash, look at our BibTEX entry for [Tre08].

Do not italicize English in math mode. Mathematical symbols should be typeset in math mode: write $Ax = b$, not Ax=b. This said, subscripts or superscripts that derive from English (or any human language) should not be italicized. For example, write f_{best}, not f_{best}. The exception is subscripts based on a single letter: refer to a point that is the center of some set as x_c, not x_{c}. Similarly, use commands for special functions: use $\sin(x)$, $\log(x)$, and $\exp(x)$, not $sin(x)$, $log(x)$, or $exp(x)$.

A really heinous example would be the following:

> Consider the problem
> $$minimize \quad f(Ax - b)$$
> where x is the optimization variable and A and b are problem data.

Spacing. A blank line ends a paragraph. You shouldn't leave a blank line between an equation and the following text unless you intend the equation to end the paragraph. Write:

```
The image of $S$ under $f$,
\[
    f(S) = \{ f(x) \mid x \in S \},
\]
is convex.
```

Inserting extra blank lines before `\[` or after `\]` will result in bad typesetting. However, the following is fine, since a new paragraph is called for:

```
The image of $S$ under $f$ is defined as
\[
    f(S) = \{ f(x) \mid x \in S \}.
\]

We now turn to a different topic.
```

Using a tab before `f(S)` inside the equation environment is optional, but helps readability in the source. Similar rules should be followed for other environments like `quote`; see the source of this document.

Use of notation and jargon. The correct and appropriate use of notation and jargon takes time to master. The following should give you a sense of what to think about:

- Don't use the same notation for two different things. Conversely, be consistent about notation for the same thing mentioned in two places: don't say "A_j for $1 \leq j \leq n$" in one place and "A_i for $i = 1, \ldots, n$" in another, or use two different indices in summations over the same ranges. Formally, the i and j are dummy variables with only that phrase as scope, so these are technically correct, but it will confuse the reader.

- It can be useful to choose names for indices so, for example, i always varies from 1 to m and j always varies from 1 to n (when referencing something like the rows and columns of an $m \times n$ matrix).

- Define all symbols before or near to where you use them. A good rule is that when you first use something, say, X, you should either define it immediately, or within the paragraph. There are a few cases where it is acceptable to define it later, but you must say this explicitly, as in "X is the matrix of activation levels, which we define below."

- A symbol like f refers to a function, while $f(x)$ refers to a function evaluated at a given point. Avoid sloppy writing like "The function $f(x)$ is convex." So-called 'anonymous' functions defined inline are an exception to this rule, as in "the function $x^2 \cos x$ is a counterexample," though this should be used only for simple functions.

- Use typographic naming conventions like capital letters for sets, Greek letters for dual variables, *etc.*

- Try to use mnemonic notation, so x_c for a center point, c for a cost vector, S for a generic set, C for a convex set, B for a ball, and so on. Note that there are conventions about the use of different letters in mathematics that you should not ignore: i, j, and k are usually used for indices, x through z for variables, a through d for constants, *etc.*

- Don't use symbols like $\forall$, $\exists$, and $\Rightarrow$; use the corresponding words. These symbols are usually appropriate only in formal logic.

- Don't overuse subscripts. If you do not need to assign a subscript to something, don't. For example, if you begin by defining a set as $X = \{x_1, \ldots, x_n\}$, then it is going to get annoying to refer to subsets of X, since you will need double subscripts. If possible, it would be better to not name the elements of X and only refer to them when necessary, just as, say, x and y.

  Beware especially of any topic in graph theory when it comes to this rule; there are often *many* ways to refer to relevant objects, and finding reasonable notation can often get one half the way to getting a handle on the problem itself.

- Don't assign symbols to concepts that you never refer to, or can easily refer to without:

Bad: Let X be a compact subset of a space Y. If f is a continuous real-valued function over X, it has a minimum over X.
Good: A continuous real-valued function has a minimum over a compact set.

Similarly, do not say "The solution $x^\star$ is unique" if you never need to refer to $x^\star$ again; simply say that the solution is unique. When you say "The solution $x^\star$ is unique" you are both stating a fact *and* entering the symbol $x^\star$ into the paper's symbol table and the reader's working memory.

Sections. Use structures like `section`, `subsection`, `subsubsection`, and `paragraph` to organize the exposition; never insert manual line breaks or page breaks for this purpose. In particular, do not use line breaks to separate different topics. Do not use symbols in article titles and ideally not even in section headers.

Use capitalization consistently. For section headers, capitalize the first letter, proper nouns, and lowercase everything else, as in this document. For references, write Figure 1, Algorithm 3, Theorem 4, and so on.

Use the right commands. There are certain special commands in LaTeX for notation that you otherwise might attempt to write in an ad-hoc manner. If you do the latter, the typesetting will be inferior. A few common examples:

- Norms are written $\|x\|$, not $||x||$.
- For set-builder notation, use $\{x \in \mathbf{R} \mid x \geq 0\}$, not $\{x \in \mathbf{R} | x \geq 0\}$.
- Make sure to use `\left` and `\right` when wrapping taller expressions with parentheses, curly braces, brackets, and so on.

  Bad: Let
  $$x = \operatorname*{argmin}_u (f(u) + \frac{1}{2}\|u - z\|_2^2).$$
  Good: Let
  $$x = \operatorname*{argmin}_u \left(f(u) + \frac{1}{2}\|u - z\|_2^2 \right).$$

- Use `\ldots` (lower dots) when the dots are surrounded by commas and `\cdots` (center dots) when surrounded by other objects that have full height, as in $x_1, x_2, \ldots, x_n$ and $x_1 + x_2 + \cdots + x_n$.

Similarly, make sure to use our conventions for standard mathematical objects for this class; for example, use $\mathbf{R}$, not $\mathbb{R}$, to denote the set of real numbers. Use the `\argmin` and `\argmax` commands (as shown above); do not write 'arg min', since 'argmin' is a single mathematical operator. These and other definitions are provided in `defs.tex`.

Writing optimization problems. In this class, optimization problems can be introduced in the sentence as nouns. For example, write as follows:

> Consider the problem
>
> $$\begin{array}{ll} \text{minimize} & (1/2)\|Ax-b\|_2^2 + \lambda\|x\|_1 \\ \text{subject to} & 0 \preceq x \preceq \mathbf{1} \\ & \|x\|_2 \leq 1, \end{array} \tag{2}$$
>
> where $x \in \mathbf{R}^n$ is the optimization variable, and $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, and $\lambda > 0$ are problem data.

It is important to state which symbols refer to variables and which to problem data.

Be sure to carefully distinguish an optimization problem, an algorithm for solving it, and its optimal value. You *solve* a problem (possibly, using a particular algorithm); an optimization problem *has* an optimal value; and an optimization problem *is* convex, or infeasible. It does not make sense to talk about whether or not a problem converges.