UNIVERSITY OF CAMBRIDGE

Department of Engineering

# Algorithms for Matrix Estimation

Author Name: **Pablo Pascual Cobo**

Supervisor: **Dr Ramji Venkataramanan**

Date: **27/05/2020**

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

*Signed* [illegible] *date* 27/05/2020

Engineering Tripos Part IIB

**Part IIB Project**

# Algorithms for Matrix Estimation

**FINAL REPORT**

**Pablo Pascual Cobo, pp423**

**Churchill College**

**Supervised by Dr Ramji Venkataramanan**

# Contents

# Technical Abstract

This project addresses the problem of recovering a low-rank matrix from a low proportion of observed entries. In other words, given a large, partially observed matrix, matrix estimation (and specifically matrix completion) achieves the task of filling in the missing entries using only the known entries and a low-rank assumption. This technique has proven to be useful in many real-world applications, such as recommender systems and inferring gene-disease relationships. In the case of an incomplete dataset of movie ratings for customers, matrix completion can be used to predict the ratings that users would give to movies they have not yet watched.

For an incomplete observation matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, the optimisation problem for matrix completion can be defined as learning a matrix $\mathbf{Z} \in \mathbb{R}^{m \times n}$ which minimises the error with respect to the known entries of $\mathbf{M}$ while being subject to a low rank constraint. It is commonly known as *spectral regularisation.*

The first part of the report compares two algorithms for matrix completion with respect to accuracy and run-time. Performance results are reported and analysed for both simulated and real data sets. However, the complexity of these algorithms scales polynomially with the matrix dimensions, and hence they cannot be used directly on large matrices.

To address this issue, the second part of the project investigates the use of side information to improve the accuracy and speed of matrix completion. We develop and implement new algorithms that can exploit perfect or imperfect knowledge of the row and column spaces of the underlying matrix. After evaluating the performance of these algorithms on simulated data, we use them to infer gene-disease associations.

# 1 Introduction

There are many real-world applications in which predictions have to be made based on observed or measured data. In most cases, the measured data can be represented as a large matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ with several missing entries and the problem becomes "completing" this matrix based on the observed entries. One such case of this matrix estimation problem is the Netflix Movie Challenge.

From a dataset of movie ratings on a scale from 1 (worst) to 5 (best) given by customers, the Netflix Movie Challenge consists of building an efficient system which recommends suitable movies to each of its customers. The ratings are laid out in an $m \times n$ matrix, where the $m$ rows are the users and the $n$ columns are the movies. However, as most users will only rate a few movies, most entries of the data matrix will be unknown. For example, for the Netflix dataset, where $m = 480,189$ and $n = 17,770$, only 1 % (100 million) of the ratings are present. In order for the system to give good recommendations to users, it is necessary to make reasonable predictions for the ratings of unrated movies.

An excerpt of this matrix representation has been shown in Figure 1 below. This matrix completion problem is ill-specified and additional constraints are required. Of all possibilities, the most common choice is to set a low rank constraint.

In the movie rating case, this can be done by applying a low-rank factor model. If there are $r$ possible genres of movies, each of these genres can be assigned a weighting (summing to a total of 1) for each user, resulting in a full user-genre matrix $\mathbf{C} \in \mathbb{R}^{m \times r}$. On the other hand, each movie can also be given a score (from 1 to 5) for each of the $r$ genres, producing another full matrix $\mathbf{G} \in \mathbb{R}^{r \times n}$. When combining these two factor matrices, a low-rank prediction of the original matrix $\mathbf{M}$ can be obtained, defined as $\hat{\mathbf{M}} = \mathbf{C}\mathbf{G}^T \in \mathbb{R}^{m \times n}$, which has dimensions $m \times n$ and rank $r$. This would result in the predictions for the unknown entries shown in Fig. 1d.

|  | Toy Story | Fargo | Johnny English | Rambo |
|---|---|---|---|---|
| User 1 | – | 5 | 1 | 3 |
| User 2 | 4 | – | – | – |
| User 3 | – | 4 | – | 3 |
| User 4 | – | 4 | – | – |
| User 5 | 2 | – | 5 | – |

(a) Excerpt of original ratings matrix **M**

|  | Action | Drama | Comedy |
|---|---|---|---|
| User 1 | 0.4 | 0.6 | 0 |
| User 2 | 0 | 0.1 | 0.9 |
| User 3 | 0.4 | 0.5 | 0.1 |
| User 4 | 0.3 | 0.3 | 0.3 |
| User 5 | 0.1 | 0.8 | 0.1 |

(b) User-genre matrix **C**

|  | Action | Drama | Comedy |
|---|---|---|---|
| Toy Story | 4 | 1 | 4 |
| Fargo | 4 | 5 | 1 |
| Johnny English | 2 | 1 | 5 |
| Rambo | 5 | 3 | 1 |

(c) Movie-genre matrix **G**

|  | Toy Story | Fargo | Johnny English | Rambo |
|---|---|---|---|---|
| User 1 | 2 | 5 | 1 | 3 |
| User 2 | 4 | 1 | 5 | 1 |
| User 3 | 3 | 4 | 2 | 3 |
| User 4 | 3 | 4 | 3 | 3 |
| User 5 | 2 | 5 | 5 | 3 |

(d) Low-rank estimate for original matrix, $\hat{\mathbf{M}}$

Figure 1: Excerpt of a ratings matrix, its low-rank factor model and estimate $\hat{\mathbf{M}} = \mathbf{C}\mathbf{G}^T$.

Practically, what this low-rank model implies is that, if *User 2* likes comedy-based movies such as *Toy Story*, he is more likely to give high ratings to similar comedy-themed movies, like *Johny English*, than to *Fargo* which is mainly a drama and action movie. The imposed low-rank assumption therefore seems a very logical approach.

It is worth noting that, in this example, a very small excerpt of the original matrix **M** has been used, as well as a very simple low-rank factor model with a rank of only 3. In practice, the factors are expected to be more elaborate producing a more realistic estimate of the original matrix.

Apart from the movie recommender system discussed above, other common applications of matrix estimation include link prediction (recommending users in social networks), pairwise distance matrices (finding distance between two cities) or ranking (predicting how likely a player is to beat another in a game).

Under the low-rank assumption and making good use of the known entries, previous research by Candès and Recht [4] has proved that matrix completion algorithms are able to achieve exact recovery of most low-rank matrices even when the proportion of known entries from the original data is very low. In addition, they demonstrated that this can be performed by solving relatively simple convex optimization problems.

As a consequence of these findings, matrix estimation has received increased interest during the last 10 years. Hastie, Tibshirani and Wainwright define the principles of matrix estimation in Chapter 7 of their book *Statistical Learning with Sparsity: The Lasso and Generalizations*, as well as including the **soft-impute** algorithm for matrix completion [7]. In addition, Cai, Candès and Shen provide theoretical insight into their alternative *Singular Value Thresholding* algorithm [2].

During the first part of this report (Sec. 2) , I will analyse the principles and performance of these two algorithms for matrix estimation, comparing them with a Python-embedded generic optimization tool. Occasionally, the known data may be corrupted with noise, so the effect of noise on the accuracy with which matrices are recovered will also be investigated following the model presented by Candès and Plan [3]. Both algorithms will be used to predict movie ratings (as in the example above) for the *Netflix* and *MovieLens 1M* datasets to confirm their success.

The input matrices on which matrix completion is applied tend to be very large and have very few known entries. Consequently, the optimization problems concerned become computationally prohibitive and efficiently recovering the original matrix becomes difficult. Using side information provides a solution to this problem. This will be addressed in the second part of this report (Sec. 3 and 4).

In some applications of matrix completion, side information is often available in addition to the observed entries. This side information is structured as two smaller side information matrices $\mathbf{A} \in \mathbb{R}^{m\times r_a}$ and $\mathbf{B} \in \mathbb{R}^{n\times r_b}$ which respectively describe the row space and column space of the original matrix. For the movie-ratings example introduced above, this side information could correspond to demographical information about the users or textual descriptions of movies, producing side information matrices **A** and **B**. Under the assumption that the full observation matrix and the side information matrices share the same latent information, the problem can be reduced to optimising a matrix $\mathbf{Z}_0 \in \mathbb{R}^{r_a\times r_b}$ of smaller size $r_a \times r_b$ (where $r_a < m$ and $r_b < n$).

Xu, Jin and Zhou investigate matrix completion with side information, including developing a rather complex algorithm [14] whereas Chiang et al. [5] propose a new model for cases where side information does not fully describe the observations. Given the complexity of existing algorithms, my focus was on developing simple algorithms for both perfect and noisy side information following the principles of **soft-impute**. This is described in Sec. 3. Finally, their performance was compared to the traditional algorithms for the prediction of gene-disease associations (Sec. 4).

Overall, the contributions of the project can be summarised as follows:

- We analyse the performance of standard matrix completion algorithms and compare them to generic optimisation tools. (Sec. 2.4)
- We investigate the effect of noisy observations on the efficiency of these algorithms and apply them to the prediction of movie ratings. (Sec. 2.5, 2.6)
- We develop novel algorithms for matrix completion using side information, with a simpler structure than those already available, using **soft-impute** as our starting point and investigating the best choice of parameters. (Sec. 3)
- We quantify the effectiveness of the developed algorithms for different proportions of observed entries and noise levels in the side information. (Sec. 3.4)
- We apply the new algorithms to predict gene-disease associations, using available side information about the genes and diseases involved. (Sec. 4)

Section 2 describes the basic theory of standard matrix completion, whereas Sections 3 and 4 describe the development and evaluation of new algorithms that use side information to perform matrix completion. Finally, Section 5 summarises the main results and conclusions drawn from the project.

# 2 Matrix Completion

## 2.1 Theory

The problem of matrix completion can be defined as follows: given an incomplete $m \times n$ observation matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ with missing entries, under low-rank assumptions, learn a matrix $\mathbf{Z} \in \mathbb{R}^{m\times n}$ which solves the following optimisation problem [4]:

$$\text{minimise rank}(\mathbf{Z}) \;\; \text{subject to } m_{ij} = z_{ij} \text{ for all } (i,j) \in \Omega, \tag{1}$$

where $\Omega \subseteq \{1, ..., n\} \times \{1, ..., m\}$ is the subset of indices of matrix $\mathbf{M}$ corresponding to observed entries. The low-rank assumption implies that the rank $r$ of matrix $\mathbf{M}$ is significantly smaller than the dimensions of matrix $\mathbf{M}$, that is, $r < m, n$.

Applying hard constraints on the known entries of the matrix will often prove to be too harsh a constraint and can cause overfitting. In addition, for some application the observed entries may be exposed to noisy. Consequently, the optimisation problem is replaced by the following soft-constrained equivalent:

$$\text{minimise rank}(\mathbf{Z}) \;\; \text{subject to } \sum_{(i,j)\in\Omega} (z_{ij} - m_{ij})^2 \leq \delta, \tag{2}$$

for some $\delta > 0$.

Unfortunately, the rank-minimisation problem is NP-hard, so there is no guarantee that a solution will be found. Instead, as suggested by Candès and Recht [4], the problem is replaced by its convex relaxation

$$\text{minimise } \|\mathbf{Z}\|_* \text{ subject to } \sum_{(i,j)\in\Omega} (z_{ij} - m_{ij})^2 \leq \delta, \tag{3}$$

where $\|\mathbf{Z}\|_*$ is the nuclear norm of $\mathbf{Z}$. The *nuclear norm* (or *trace norm*) of a matrix is equal to the sum of its singular values, and defined as

$$\|\mathbf{Z}\|_* = \text{trace}\left(\sqrt{\mathbf{Z}^\top \mathbf{Z}}\right) = \sum_{i=1}^{\min\{m,n\}} \sigma_i,$$

where $\sigma_1, \sigma_2, ...$ denote the singular values of $\mathbf{Z}$.

As a matrix with rank $r$ will have only $r$ non-zero singular values, minimising the nuclear norm proves to be an appropriate convex alternative to minimising the rank [4, 7].

In order to solve the problem in Eq.(3), it is useful to convert it into a Lagrange function, with the constraint on the observed entries of the matrix as part of the objective. The new optimisation problem, known as *spectral regularisation*, becomes

$$\min_{\mathbf{Z}} \left\{ \frac{1}{2} \sum_{(i,j)\in\Omega} (z_{ij} - m_{ij})^2 + \lambda \|\mathbf{Z}\|_* \right\}, \tag{4}$$

where $\lambda$, the Lagrange multiplier, is a tuning parameter specific to the data.

## 2.2 Algorithms

I implementred three algorithms in Python based on the existing literature on this topic. These algorithms are all based around **Singular Value Decomposition**. We recall that if a matrix $\mathbf{Z}$ has singular value decomposition $\mathbf{Z} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, a rank-$r$ approximation of $\mathbf{Z}$, $\hat{\mathbf{Z}}_r$ can be obtained by setting all but the first $r$ diagonal entries of $\mathbf{D}$ to 0, so that $\hat{\mathbf{Z}}_r = \mathbf{U}\mathbf{D}_r\mathbf{V}^T$.

### 2.2.1 `cvxpy`

As a reference to compare the algorithms being used, the optimisation problem in (4) was solved using the Python library `cvxpy` [6]. This library allows the user to set the objective function and constraints and then determines the best solution from using a broad range of solvers. Although the runtime of this method can be quite slow, it serves as an initial reference to measure the achievable accuracy by the designed algorithms.

### 2.2.2 Soft-Impute algorithm

This iterative algorithm was developed following the outline presented by Hastie et al. [7]. The adaptation of this algorithm which I used during this project is given below:

```
Soft-Impute Algorithm

1. Initialize $\mathbf{Z}^{\text{old}} = \mathbf{0}$ and $\lambda_0 = \|P_\Omega(\mathbf{M})\|_2/1.5$
2. For each $k = 1, ..., K,$ set $\lambda_k = \frac{\lambda_0}{k}$ and iterate until convergence:
     Compute $\hat{\mathbf{Z}}_k \leftarrow \mathcal{S}_\lambda\left(P_\Omega(\mathbf{M}) + P_\Omega^\perp\left(\mathbf{Z}^{\text{old}}\right)\right)$
     Update $\mathbf{Z}^{\text{old}} \leftarrow \hat{\mathbf{Z}}_k$
3. Final output is $\hat{\mathbf{Z}}_K$
```

The operator $S_\lambda$ reduces the rank of the estimate $\hat{\mathbf{Z}}_k$ by performing a soft-thresholded version of SVD. If $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$,

$$S_\lambda(A) = \mathbf{U}\mathbf{D}_\lambda\mathbf{V}^T, \quad \mathbf{D}_\lambda = diag[(d_1 - \lambda_k)_+ \cdots (d_n - \lambda_k)_+], \tag{5}$$

where $\mathbf{D}_\lambda$ is the diagonal matrix obtained by setting all entries of $\mathbf{D}$ smaller than the value of $\lambda_k$ to 0 (and subtracting $\lambda_k$ from the rest). This operation is known as *soft-thresholding*. In (5),

$$(x)_+ = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

By reducing the number of non-zero entries in the diagonal matrix, soft-thresholded SVD effectively reduces the rank and thus also the nuclear norm term in the objective function (4).

The operator $P_\Omega$ projects onto the subspace $\Omega$, i.e. onto the observed entries of the input matrix $\mathbf{M}$, with the missing entries replaced by zeros. On the other hand, the term $P_\Omega^\perp\left(\mathbf{Z}^{\text{old}}\right)$ fills the unobserved indices in $\mathbf{M}$ with entries from $\mathbf{Z}^{\text{old}}$.

In every iteration $k$, the soft-impute algorithm produces a new estimate $\hat{\mathbf{Z}}_k$ of the complete matrix. This estimate is a low-rank approximation (due to soft-thresholding) of the matrix formed by adding $P_\Omega(\mathbf{Z})$, the observed entries in the original matrix, and $P_\Omega^\perp(\mathbf{Z}^{\text{old}})$, the prediction for the missing entries from the previous estimate. In practice, this sum is equivalent to an expression which is easier to compute:

$$P_\Omega(\mathbf{Z}) + P_\Omega^\perp\left(\mathbf{Z}^{\text{old}}\right) = P_\Omega(\mathbf{Z}) + \mathbf{Z}^{\text{old}} - P_\Omega\left(\mathbf{Z}^{\text{old}}\right). \tag{7}$$

This can be expressed as the sum of a sparse matrix $P_\Omega(\mathbf{Z}) - P_\Omega\left(\mathbf{Z}^{\text{old}}\right)$ and a low-rank matrix $\mathbf{Z}^{\text{old}}$. The special structure of these two matrices allows the required matrix operations (namely SVD) to be performed efficiently even for large data matrices.

$\lambda_0$ is set to $\|P_\Omega(\mathbf{M})\|_2/1.5$, with the unknown entries of $\mathbf{M}$ set to 0. Given that $\|P_\Omega(\mathbf{M})\|_2 = \sigma_{\max}(P_\Omega(\mathbf{M}))$, this sets the initial threshold in order to reduce the rank of the matrix $\mathbf{M}$ to the number of singular values that have size comparable to the maximum singular value. This serves as a good first stage for the algorithm.

Furthermore, the parameter $\lambda$ is chosen to decrease fractionally from each iteration to the next, so that for the $k$th iteration, $\lambda_k = \frac{\lambda_0}{k}$. Consequently, the algorithm will be very aggressive (large change in estimate) in the first iterations, when $\lambda$ is large, and will gradually become more faithful in the solution achieved as $\lambda$ decreases.

Occasionally, the algorithm will converge to a solution before the $K$th iteration is reached. To avoid additional unnecessary iterations, a stopping criterion is introduced which terminates the algorithm's loop (with the current estimate as output) once it has converged. The stopping criterion for the $k$th iteration is

$$\left\|\hat{\mathbf{Z}}_k - \mathbf{Z}^{\text{old}}\right\|_F \Big/ \left\|\hat{\mathbf{Z}}_k\right\|_F < 10^{-6}, \tag{8}$$

so the algorithm will stop when the relative error between the current and the previous estimate is very small.

In the next section, where matrix completion using side information will be investigated, soft-impute will serve as the starting point for the new algorithms due to its simple and easily adaptable structure.

### 2.2.3 Singular Value Thresholding (SVT) Algorithm

SVT is another iterative algorithm, in this case introduced by Cai et al.[2]. In this project, the original SVT algorithm has been simplified to reduce the number of parameters. The adapted algorithm is listed below.

**Singular Value Thresholding (SVT) Algorithm**

```
1. Initialize $\mathbf{Y}^0 = \delta P_\Omega(\mathbf{M})$, where $\delta = 1.2/(1-p)$
2. For each $k = 1, ..., K,$ set $\lambda = 5n$ and iterate until convergence:
       Compute $\mathbf{X}^k \leftarrow \mathcal{S}_\lambda\left(\mathbf{Y}^{k-1}\right)$
       Set $\mathbf{Y}^k = \mathbf{Y}^{k-1} + \delta P_\Omega\left(\mathbf{M} - \mathbf{X}^k\right)$
3. Final output is $\mathbf{X}^K$.
```

As with soft-impute, in SVT, soft-thresholding is performed in each iteration, but with the threshold $\lambda$ remaining constant. $\lambda$ is chosen empirically and set to $\lambda = 5n$. This enforces the low rank constraint and ensures the value $\lambda\|\mathbf{Z}\|_*$ in the objective function (4) is large with respect to the squared error in the known entries.

In addition, each iteration $k$ produces two different matrices, $\mathbf{X}^k$ and $\mathbf{Y}^k$. $\mathbf{Y}^k$ is only non-zero for the matrix indices that belong to the subset $\Omega$ (corresponding to the observed entries of $\mathbf{M}$). For each iteration, $\mathbf{Y}^k$ is updated by adding some multiple of $P_\Omega\left(\mathbf{M} - \mathbf{X}^k\right)$, the error in $\mathbf{X}^k$ with respect to the original matrix. The magnitude of the correction depends on parameter $\delta$, which aims to guarantee the convergence of the algorithm. It is set to $\delta = 1.2/(1-p)$, where $p$ is the proportion of missing entries in the original matrix $\mathbf{M}$. This means that, the larger the proportion of observed entries $(1-p)$, the less aggressive the update on matrix $\mathbf{Y}^k$ will be.

As to $\mathbf{X}^k$, it is obtained by soft-thresholding $\mathbf{Y}^{k-1}$, which effectively enforces the low-rank constraint and produces an estimate $\mathbf{X}^k$ of the original matrix.

A stopping criterion is also addded to **SVT** to terminate the loop when the algorithm has converged to a solution. In this case, the stopping criterion for the $k$th iteration is

$$\left\|P_\Omega(\mathbf{X}^k - \mathbf{M})\right\|_F / \|P_\Omega(\mathbf{M})\|_F < 10^{-6}, \qquad (9)$$

i.e. the relative error of the current estimate with respect to the observed entries of $\mathbf{M}$.

Overall, the SVT Algorithm contains more parameters and generally a more complex structure than soft-impute. These can be difficult to tune, as their optimal value will depend on the properties of the input data. More details on the parameters and how they should be set are given by Cai et al. in the original paper [2].

## 2.3 Matrix Coherence

In order for these algorithms to be successful on predicting the missing entries, the input data must, as explained in the introduction, fit a low rank model. Furthermore, there is an additional requirement on the input data matrix: it must be incoherent.

*Coherence* refers to the extent to which the singular vectors of the input matrix are aligned with the standard basis, that is, not spread across all components. Most low-rank matrices can be recovered from a sampling of its entries (when a proportion of its entries are missing). However, this is not the case when the matrix is coherent, as illustrated by the following example.

Suppose there is a $m \times n$, rank 1 matrix $\mathbf{A}$, where all the entries in rows 2 to $m$ are 0:

$$\mathbf{A} = \begin{pmatrix} x_1 & x_2 & \cdots & x_{n-1} & x_n \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}, \tag{10}$$

This matrix has a singular vector $[1 \ 0 \ \cdots \ 0]^T$ . If any single element of its first row is missing, it will be impossible to recover the original matrix (even if all other elements are observed). As the singular vector is aligned with the standard basis, which makes the matrix coherent, the rank of the matrix will remain 1 no matter the value chosen for the missing top-row element. Coherence of input matrices must be avoided for exact recovery to be possible.

## 2.4 Simulations

### 2.4.1 Methodology

All three algorithms above were implemented in Python 3 from scratch and tested by simulating possible input data. The full input matrix $\mathbf{M}^{\text{full}} \in \mathbb{R}^{m \times n}$, with dimensions $m \times n$ and rank $r$, is produced as $\mathbf{M}^{\text{full}} = \mathbf{U}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$ have dimensions $m \times r$ and $n \times r$ respectively. The entries in $\mathbf{U}$ and $\mathbf{V}$ are randomly sampled from a Gaussian alphabet, i.e. $\mathbf{U}, \mathbf{V} \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$.

As to the missing entries, each entry in $\mathbf{M}^{\text{full}}$ is "removed" (and replaced by a 0) with probability $p$, producing the input matrix $\mathbf{M}$ with a proportion $p$ of entries missing. This is equivalent to Bernoulli sampling across all entries with probability $(1-p)$.

### 2.4.2 Measuring Performance

The performance of each algorithm is measured in terms of the accuracy with which they are able to recover the original, full matrix $\mathbf{M}^{\text{full}}$. This is measured as the relative RMS error between the algorithm's reconstruction $\hat{\mathbf{Z}}$ and the full matrix $\mathbf{M}^{\text{full}}$, across all entries. It will be referred to as *relative error* throughout this report:

$$\text{Relative error} = \frac{\left\|\hat{\mathbf{Z}} - \mathbf{M}^{\text{full}}\right\|_F}{\left\|\mathbf{M}^{\text{full}}\right\|_F} = \frac{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{z}_{ij} - m_{ij}^{\text{full}})^2}}{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(m_{ij}^{\text{full}\ 2})}} \tag{11}$$

As will be shown later, most real-world problems involve very large matrices and it is essential that the used algorithms are able to perform matrix completion in a manageable time. Therefore, the runtime associated to each algorithm is another important performance measure. This is simply measured as the time taken from the start of the first iteration to the end of the last one.

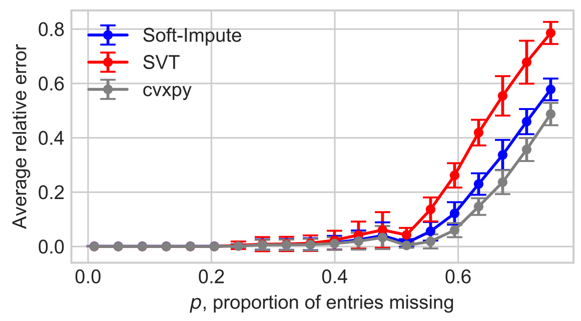
In addition, the same number of iterations ($K = 2000$) is used for each of **soft-impute** and **SVT** so that the runtime and error comparisons are fair. In order to ensure the reliability and consistency of the results achieved, all tests are averaged over 10 runs. A random number generator with set seeds is used to perform the simulations always on the same random input matrices.

### 2.4.3 Results and Discussion

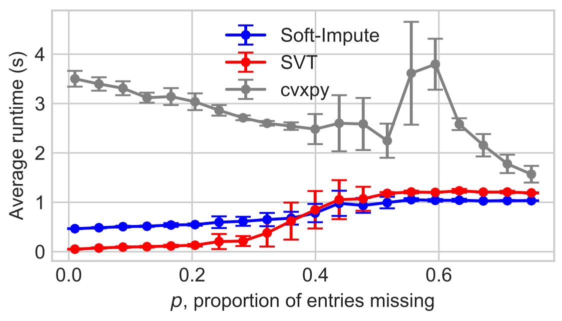
The performance of the 3 algorithms is initially compared for $50 \times 50$ noiseless matrices of rank 5 and 10 and varying proportion of missing entries $p$ in the input matrix. The results are shown in Fig. 2. The error bars represent one standard deviation, given that the graphs show the average result over 10 runs.

Both **soft-impute** and **SVT** prove to give a comparable error to the reference `cvxpy` (only slightly worse), therefore confirming they are valid algorithms for matrix completion.

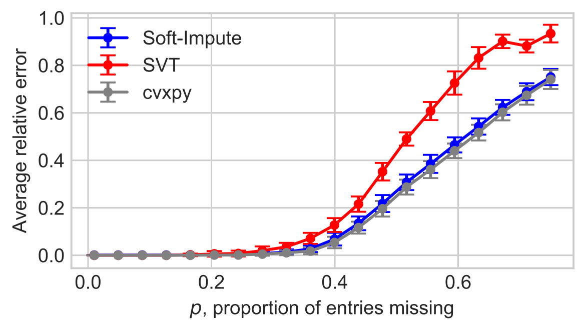
On the other hand, both have a significantly lower runtime than `cvxpy` as the latter is not problem-specific and hence it will take longer for `cvxpy` to find the best suited optimisation method for matrix completion.



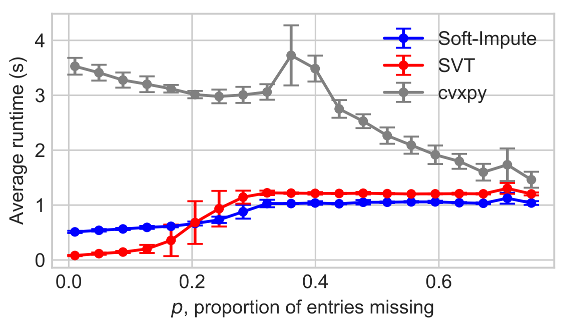
(a) Average error, rank $r = 5$



(b) Average runtime, rank $r = 5$



(c) Average error, rank $r = 10$



(d) Average runtime, rank $r = 10$

Figure 2: Performance plots for $50 \times 50$ noiseless matrices of **rank 5** (top) and **rank 10** (bottom).

As expected, the error increases with $p$, as less of the matrix entries are known and there is less information available to predict the missing entries. Comparing **SVT** and **soft-impute** for rank 5, **SVT** gives a lower relative error for $p < 0.2$ (order of $10^{-5}$ compared to $10^{-3}$). Their errors show similar trends up to $p = 0.5$. For larger values of $p$, however, **SVT** is observed to achieve significantly worse recovery of the original matrix than **soft-impute**. In terms of runtime, it is slightly less for **SVT** with $p < 0.4$ (although it is less than 1 second in both cases). For $p > 0.4$, **soft-impute** is slightly faster at performing matrix completion, although their runtimes are almost identical.

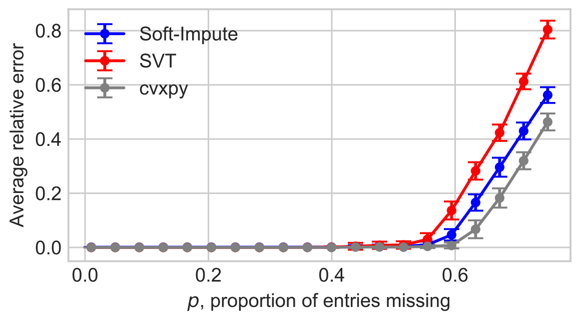
For `cvxpy`, the runtime is observed to decrease as $p$ increases. As less entries of the original matrix are known, there are less constraints and the optimization can be performed more rapidly. However, for **soft-impute** and **SVT** the runtime is observed to increase slightly with $p$ due to the slower convergence of the algorithms when there are less known entries.

For rank-10 $50 \times 50$ input matrices we observe similar trends, as shown by Figures 2c, 2d. It is worth noting that the average error increases for all three algorithms with respect to the results for rank-5 matrices. This implies that recovering higher rank matrices becomes more difficult, as there are more possible ways of enforcing the rank constraint (the number of distinct combinations of estimated entries that will result in a rank-10 matrix is higher than for a rank-5 matrix). Nevertheless, the magnitudes for the average runtime remain approximately the same.

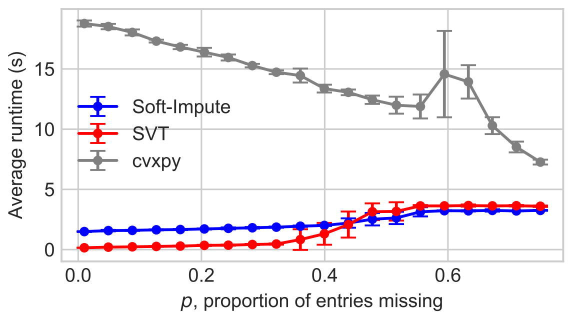
In addition, at this higher rank **soft-impute** outperforms **SVT** in runtime for $p > 0.2$ and error for $p > 0.3$, less proportion of missing entries is required for **soft-impute** to dominate than at rank $r = 5$.

Moreover, results are also collected for larger, $100 \times 100$ matrices with rank 10 (one tenth the number of columns or rows) and shown in Figure 3. Compared to those obtained for $50 \times 50$ matrices with rank 5 (same rank to columns ratio) in Fig. 2a and 2b, the plots for average relative error look pretty similar although the variance in error is perhaps lower for large values of $p$. In terms of the runtime, it increases drastically (up to almost 20 seconds) for **cvxpy** and only up to 2 or 3 seconds for **SVT** and **soft-impute**. This confirms that our algorithms are suitable for solving matrix completion in large matrices, as will be illustrated in Sec. 2.6.

Overall, the results collected suggest that **SVT** achieves better recovery of the original matrix in a lower runtime when most of the matrix entries are known ($p < 0.2$) whereas **soft-impute** is preferred when the proportion of missing entries exceeds 0.5.



(a) Average error



(b) Average runtime

Figure 3: Performance plots for $100 \times 100$ noiseless matrices of rank 10.

## 2.5 Matrix Completion with Noise

In many of the real-world applications of matrix completion, the input data might be corrupted by noise. Therefore, it is important to investigate the effect of noise on the success of the matrix completion algorithms. For example, in applications such as image or signal processing, the recovered entries or pixels may be corrupted by noise from the transmission channel. The algorithms must be able to recover the original signal (or image) by using the low-rank assumption.

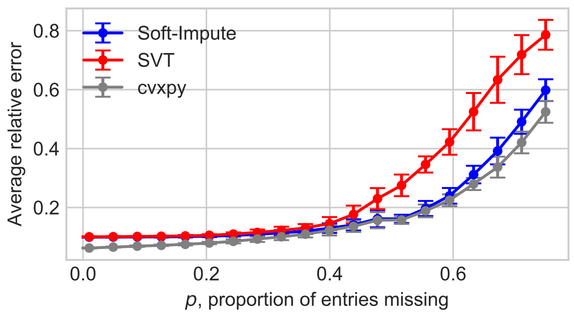
Following the methodology used by Candès and Plan [3], noise is modelled as an additive Gaussian matrix $\mathbf{E} \in \mathbb{R}^{m\times n}$ applied on the original full matrix $\mathbf{M}^{\text{full}} \in \mathbb{R}^{m\times n}$ so that the noisy matrix $\mathbf{M}^{\text{noisy}} \in \mathbb{R}^{m\times n}$ is

$$\mathbf{M}^{\text{noisy}} = \mathbf{M}^{\text{full}} + \mathbf{E}, \quad \mathbf{E} \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2), \tag{12}$$

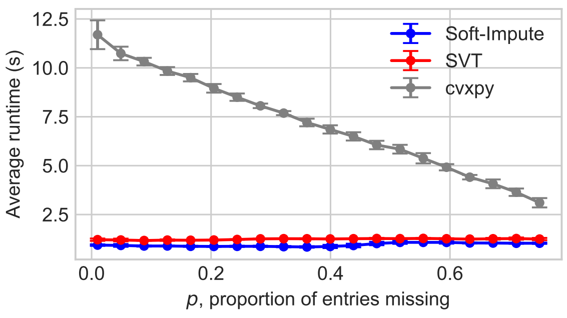
and the signal-to-noise ratio (SNR) is given by $\text{SNR} = \sqrt{\text{var}(\mathbf{Z})/\sigma^2}$.

Both the **soft-impute** and **SVT** algorithms are based on soft constraints on the known entries and thus can be used for noisy matrix completion without modifying the algorithms. On the other hand, `cvxpy` must be modified to remove the hard constraints on the known entries. This will lead to a significant increase in the computational time associated with the Python-embedded optimisation library.

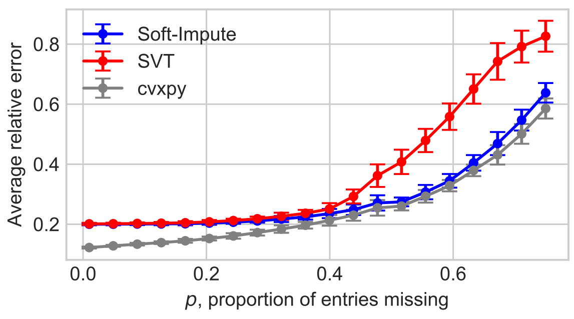
The effect of noise is investigated for $50 \times 50$ matrices of rank 5 and signal-to-noise ratios 10, 5 and 2. The SNR values are set by generating the Gaussian noise matrix $\mathbf{E}$ with appropriate values for the variance $\sigma^2$. Once the noise matrix is added, a proportion of entries $p$ is removed by Bernoulli sampling (as in the noiseless case). The resulting matrix $\mathbf{M}$ is input into the matrix completion algorithms, with the goal being to recover the noiseless original matrix $\mathbf{M}^{\text{full}}$. The results obtained are shown in Fig. 4.



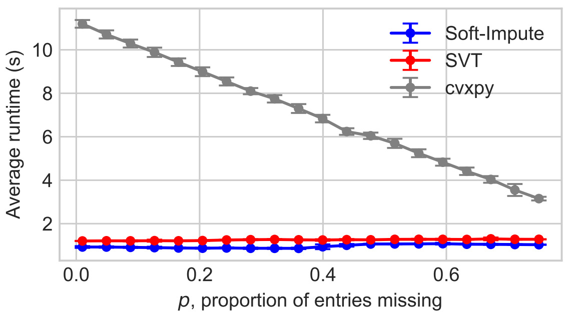
(a) Average error, SNR= 10



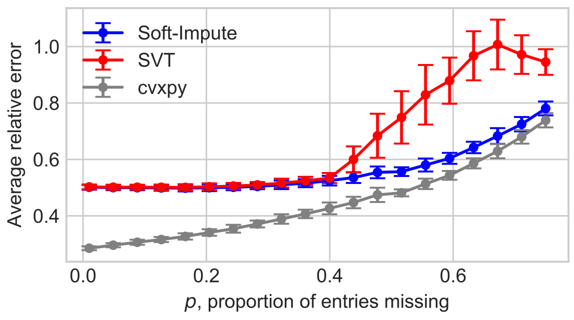
(b) Average runtime, SNR= 10



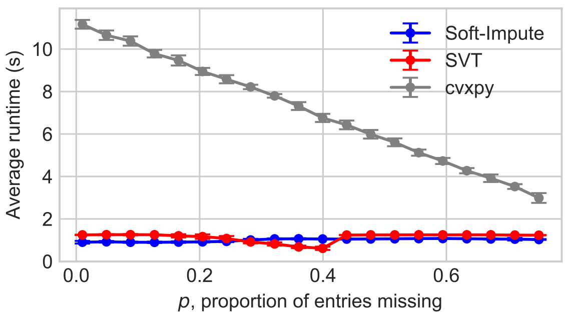
(c) Average error, SNR= 5



(d) Average runtime, SNR= 5



(e) Average error, SNR= 2



(f) Average runtime, SNR= 2

Figure 4: Performance of **soft-impute**, **IMC** and **dirtyIMC** for $50 \times 50$ noisy matrices of rank 5, with 3 different levels of noise: SNR=10 (top), SNR=5 (middle) and SNR=2 (bottom).

As expected, the accuracy with which the original matrix is recovered falls as the SNR drops and the known entries become more corrupted with noise. For SNR=10 and SNR=5, the increase in the average relative error is more noticeable at low values of $p$ (errors of 0.1 and 0.2 compared to less than $10^{-3}$ in Figure 2a). On the other hand, the error for $p = 0.75$ remains between 0.6 and 0.8 in all three graphs.

For very high levels of noise, SNR=2, which is unlikely to occur in the real-world, **soft-impute** and **SVT** are no longer able to perform accurate matrix completions and the average relative error lies above 0.5 for all values of $p$. In the presence of noise,

**soft-impute** is observed to perform better than **SVT** for all $p$, with the difference being greater when a very small proportion of entries is known.

With respect to runtime, it only increases significantly compared to the noiseless case for `cvxpy`. **Soft-impute** proved to be the faster algorithm when the observed entries are noisy.

## 2.6 Real-world Application: Predicting movie ratings

Having assessed the algorithms on simulated data, they are next tested on a real application: collaborative filtering for predicting user-movie ratings from 2 large movie rating datasets: the *MovieLens 1M* [11] and *Netflix* [1] datasets. The data from both of these sources corresponds to very large sparse matrices, with ratings only taking integer values from 1 (worst) to 5 (best):

- *MovieLens 1M* contains 1 million ratings corresponding to 4000 movies by 6000 users. This can be represented as a $6000 \times 4000$ input matrix $\mathbf{M}$, with only 4.17 % of known entries.
- The *Netflix* dataset contains 100 million ratings corresponding to 480,189 users and 17,770 movies. This can be represented by an extremely sparse $480,189 \times 17,770$ input matrix $\mathbf{M}$, where the proportion of known entries is only 1.17%.

These matrices are extremely sparse, i.e. the proportion of missing entries $p$ is very close to 1 in both cases. Therefore, it is difficult to generate the full matrix accurately. Nevertheless, as the ratings can only take 5 discrete values, this limits the possible values for the entries and therefore completion should be achievable. The data processing before and after applying the algorithms will prove essential for matrix completion to be successful.

### 2.6.1 Methodology

The same method is followed to perform matrix completion on both datasets. A cross-validation strategy is adopted: the original known entries are split into a training test set used to achieve matrix completion and a withheld test set used to evaluate the algorithm's performance. As the initial observation matrices are very large, 10 different $500 \times 500$ matrices are formed by randomly subsampling the columns and rows for the initial matrix and the results computed as an average of the 10 runs.

In addition, the mean rating for each user is subtracted from each row before applying the algorithm. This leads to the distribution of the matrix entries having approximately mean zero, closer to the Gaussian distribution with mean 0 for which the algorithm has been previously analysed.

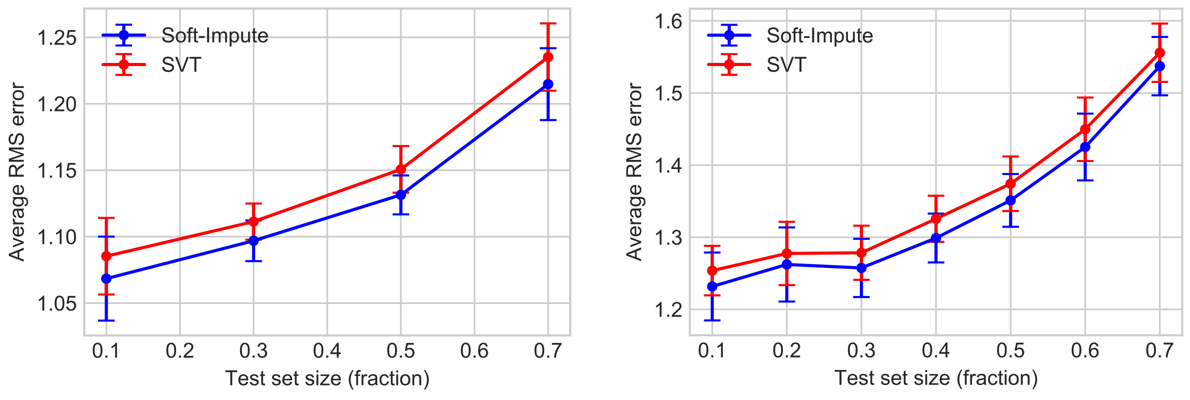
Once the output matrix from the algorithm is obtained, the mean for each user is added back to each row, all entries rounded to the nearest integer and set to 1 and 5 respectively when they are above or below the valid range of ratings. The final output is a matrix of rating predictions taking discrete values between 1 and 5. For example, User 1, who had given *Toy Story* and *James and the Giant Peach* ratings of 5 and 3 respectively, is predicted to rate the movie *Jumanji* with a 4 by both the *SVT* and *soft-impute* algorithms.

Furthermore, a new performance measure is introduced for this application which allows the results to be compared to those obtained by Li et al. [10]:

$$\text{Root-mean-square (RMS) error} = \sqrt{\frac{1}{N}\sum_{i,j\in\Omega'}(\hat{z}_{ij} - m_{ij})^2} = \frac{\left\| P_\Omega(\hat{\mathbf{Z}} - \mathbf{M}) \right\|_F}{\sqrt{N}}, \tag{13}$$

where $\mathbf{M}$ is the initial observation matrix with missing entries, $\hat{\mathbf{Z}}$ is the estimate matrix produced by the algorithm, $N$ is the total number of entries in the withheld test set and $\Omega'$ is the subset of indices corresponding to these test set entries. As with the relative error (derived in Eq. (11)) , RMS error also measures the difference between the recovered and original matrix, but without dividing by the sum of the entries of the original matrix $\mathbf{M}$ squared. Therefore, RMS error does not take into account the relative magnitudes of the input data and the values obtained are likely to be significantly larger than for previous graphs where the average relative error is used.

### 2.6.2 Results and Discussion



(a) *MovieLens 1M* (b) *Netflix*

Figure 5: Performance of the **SVT** and **soft-impute** algorithms on predicting ratings for the *MovieLens 1M* and *Netflix* datasets.

Figure 5 shows the average RMS error for the **soft-impute** and **SVT** algorithms for the prediction of movie ratings using the *MovieLens 1M* and *Netflix* datasets with different proportions of withheld test sets (from 10% to 70%).

As expected, the RMS error increases with the size of the test set, as less known entries are available to train the algorithm and thus predictions become less informed. This is equivalent to increasing the proportion of missing entries $p$ in our simulations. **Soft-Impute** outperforms **SVT**, which agrees with our previous results that conclude that **soft-impute** is a better algorithm when the proportion of known entries is low, as is the case here. Nevertheless, their error bars superpose so the difference in performance is only small.

Comparing the two datasets, the average RMS error is lower for *MovieLens 1M* throughout (by 0.15 to 0.30) due to the proportion of known entries (4.17%) being 4 times larger than for *Netflix* (1.17 %). Comparing the results to those obtained by Li et al. [10], our average RMS error was only 0.1 to 0.15 larger. This proves to be acceptable performance, given that they applied the algorithms on larger $2000 \times 2000$ matrices so are expected to obtain better results.

The average RMS errors obtained, which range from 1.05 to 1.6, would correspond to average relative errors of 0.3 to 0.5. The results mean that, on average, the predictions for movie ratings are wrong by approximately a rating of 1. Although these might seem very inaccurate predictions at first, they are acceptable considering that many users and movies in the subsampled matrices will have only 1 rating, making predictions extremely difficult.

# 3 Matrix Completion with Side Information

As seen in the previous section, standard matrix completion techniques such as SVT or soft-impute become computationally prohibitive for large matrix sizes. As data sizes are large in most applications of matrix completion, an alternative approach is required in order for matrix completion to be an effective technique.

In most real-world applications of matrix completion, in addition to the observed entries, it is possible to obtain other data, which we will call *side information*, about the matrix. In the case of the movie ratings example, this could involve demographical information about the users or textual descriptions of the movies. Similarly, when predicting links between users of a social network (link prediction), browse patterns and interactions among different users can be used as side information.

Matrix completion techniques using side information offer numerous benefits to standard matrix completion.

First, if information about the column space and row space of the matrix is available, the effective size of the low rank matrix to be recovered can be drastically reduced. As the algorithms involve updating a matrix (of the same dimensions as the low rank matrix) at each iteration, the use of side information will result in a smaller estimate matrix and thus a significant reduction in computational cost and storage requirements.

In addition, with side information, the low rank matrix may be recoverable with a much smaller number of entries. This is a key advantage, as in most applications the proportions of observed entries is very low (sometimes below 1%).

Moreover, unlike standard matrix completion, the side information approach is inductive, i.e. it allows predictions to be made for new members for which we have no observations by using the side information available. For the movie ratings application, this would correspond to predicting how a new user will rate existing movies or a new movie is expected to be rated by existing users.

Matrix completion with side information is sometimes referred to as **Inductive Matrix Completion (IMC)**. In the following section, the model and objective function for Matrix Completion with side information will be defined.

## 3.1 Theory

Using the same notation as before, let $\mathbf{M}^{\text{full}} \in \mathbb{R}^{m\times n}$ be the target matrix of rank $r$ to be recovered. Recall that $\Omega \subseteq \{1, ..., n\} \times \{1, ..., m\}$ has been defined as the subset of indices of $\mathbf{M}^{\text{full}}$ for which entries are observed.

Furthermore, following the model introduced by Xu et al.[14], define two side information matrices $\mathbf{A} \in \mathbb{R}^{m\times r_a}$ and $\mathbf{B} \in \mathbb{R}^{n\times r_b}$, where $r \leq r_a \leq n$ and $r_b \leq m$. These side information matrices $\mathbf{A}$ and $\mathbf{B}$ are also called *features* in existing literature.

The new objective function and algorithm, are based around the following assumption in order to exploit side information.

**Assumption**: The columns of $\mathbf{M}^{\text{full}}$ lie in the column space of $\mathbf{A}$ and the rows of $\mathbf{M}^{\text{full}}$ lie in the column space of $\mathbf{B}$ :

$$\text{col}(\mathbf{M}^{\text{full}}) \subseteq \text{col}(\mathbf{A}) \qquad \text{and} \qquad \text{row}(\mathbf{M}^{\text{full}}) \subseteq \text{col}(\mathbf{B}). \tag{14}$$

This assumption essentially means that all missing entries of $\mathbf{M}$ can be accurately predicted by a linear combination of feature vectors (some linear combination of the columns of the side information matrices $\mathbf{A}$ and $\mathbf{B}$). Therefore, $\mathbf{M}^{\text{full}}$ can now be expressed as $\mathbf{A}\mathbf{Z}_0\mathbf{B}^\top$ and the goal is to learn the matrix $\mathbf{Z}_0 \in \mathbb{R}^{r_a\times r_b}$. This matrix has dimensions $r_a \times r_b$ and can be significantly smaller than the original target matrix $\mathbf{M}^{\text{full}}$ (with dimensions $m \times n$). The problem is then reduced to finding a smaller optimal matrix $\mathbf{Z}_0$ of size $r_a \times r_b$, while an estimate of the original matrix is given by $\hat{\mathbf{Z}} = \mathbf{A}\mathbf{Z}_0\mathbf{B}^T$.

Applying the assumption to the initial objective function in Eq. (4), the new objective function becomes:

$$\min_{\mathbf{Z}\in\mathbb{R}^{r_a\times r_b}} \left\{ \frac{1}{2} \sum_{(i,j)\in\Omega} (m_{ij} - (\mathbf{A}\mathbf{Z}\mathbf{B}^\top)_{ij})^2 + \lambda \|\mathbf{Z}\|_* \right\}, \tag{15}$$

Unlike for standard matrix completion, the optimization problem that must be solved only concerns a $r_a \times r_b$ matrix $\mathbf{Z}_0$ and thus can be solved more efficiently (as long as $r_a \ll m$ and $r_b \ll n$).

## 3.2 Inductive Matrix Completion (IMC) Algorithm

Existing literature mentions the development and implementation of algorithms which perform matrix completion with side information efficiently, such as **Maxide** [14] or **AltMin**[8]. These algorithms, however, are difficult to understand at first sight and contain many different parameters that must be tuned specifically for each application. During this part of the project, I focused on using the theory from the previous section, together with the side information assumption and knowledge of existing algorithms, to create a simpler algorithm than those available by using the Soft-impute algorithm from Section 1 as the starting point. The algorithm I developed, which is included below, will be referred to as **IMC Algorithm** throughout this report.

```
IMC Algorithm

1. Initialize $\mathbf{Z}^{\text{old}} = \mathbf{0}$ and $\lambda_0 = \|P_\Omega(\mathbf{M})\|_2/1.5$
2. For each $k = 1, ..., K$, set $\lambda_k = \frac{\lambda_0}{k}$ and iterate until convergence:
       Compute $\hat{\mathbf{Z}}_k \leftarrow \mathcal{S}_{\lambda_k}\left(\mathbf{A}^{+}\left(P_\Omega(\mathbf{M}) + P_\Omega^{\perp}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right)\right)\mathbf{B}^{+\top}\right)$
       Update $\mathbf{Z}^{\text{old}} \leftarrow \hat{\mathbf{Z}}_k$
3. Final output is $\hat{\mathbf{Z}}_K \in \mathbb{R}^{r_a \times r_b}$, and the estimate of the original matrix is $\mathbf{A}\hat{\mathbf{Z}}_K\mathbf{B}^{\top}$
```

Note that here, $\mathbf{A}^+$ has been used to represent the Moore-Penrose pseudoinverse of matrix $\mathbf{A}$, i.e. $\mathbf{A}^+ = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top \in \mathbb{R}^{r_a \times m}$ and similarly, $\mathbf{B}^{+\top} = \left((\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}\right)^\top = \mathbf{B}(\mathbf{B}^\top\mathbf{B})^{-1} \in \mathbb{R}^{n \times r_b}$.

The algorithm follows similar steps to the soft-impute algorithm (2.2.2), with soft-thresholded singular value decomposition happening in each iteration and, as for soft-impute, the thresholding parameter $\lambda_k$ becoming gradually less aggressive.

However, due to the observation matrix $\mathbf{M}$ having different dimensions to the objective matrix $\mathbf{Z}$, the update step (first part of step 2) must include projections onto the column space and row space of side information matrices $\mathbf{A}$ and $\mathbf{B}$.

In addition, the initial parameter $\lambda_0$ (which has been set to a generally well-performing value, slightly smaller than the largest singular value of matrix $\mathbf{M}$) can be modified to suit the optimization problem better, as will be shown in Section 4.

Although the computation of step 2 seems rather complicated, in practice it can be simplified by the following result,

$$P_{\Omega}^{\perp}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right) = \mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top} - P_{\Omega}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right), \tag{16}$$

as $P_{\Omega}^{\perp}$ represents the projection onto the unobserved entries (the indices of $\mathbf{M}$ that do not belong to the subset $\Omega$). Therefore,

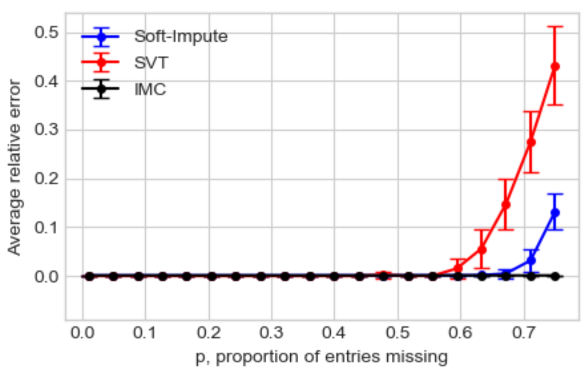
$$\begin{aligned}
&\mathbf{A}^{+}\left(P_{\Omega}(\mathbf{M}) + P_{\Omega}^{\perp}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right)\right)\mathbf{B}^{+\top} \\
&= \mathbf{A}^{+}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right)B^{+\top} \; + \; \mathbf{A}^{+}\left(P_{\Omega}(\mathbf{M}) - P_{\Omega}\left(\mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right)\right)B^{+\top} \\
&= \mathbf{Z}^{\text{old}} + \mathbf{A}^{+}\left(P_{\Omega}\left(\mathbf{M} - \mathbf{A}\mathbf{Z}^{\text{old}}\mathbf{B}^{\top}\right)\right)\mathbf{B}^{+\top}.
\end{aligned} \tag{17}$$

This last form of the result allows the update step to be run in a single line of code for each iteration.

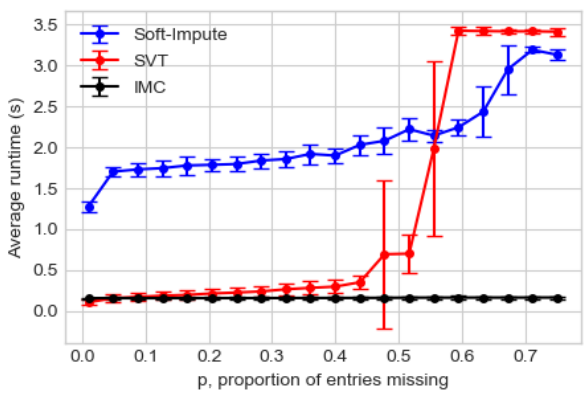
Furthermore, it is also worth noting that the algorithm above does not require the feature matrices $\mathbf{A}$ and $\mathbf{B}$ to be orthonormal. When this is the case, the projections onto the column spaces of $\mathbf{A}$ and $\mathbf{B}$ do not require computing the pseudoinverses: $\mathbf{A}^{+}$ and $\mathbf{B}^{+\top}$ can be replaced by $\mathbf{A}^{\top}$ and $\mathbf{B}$ in the equations above (for orthonormal matrices $\mathbf{A}^{\top}\mathbf{A} = \mathbf{B}^{\top}\mathbf{B} = \mathbf{I}$). However, the more general algorithm was chosen as it applies to all forms of side information matrices and the features are fixed so do not change from iteration to iteration. Computing the pseudoinverses is thus only required once at the start of the algorithm and does not lead to significant memory or runtime penalties.

### 3.2.1 Comparison with Standard Matrix Completion techniques

At this stage, some initial plots are produced to compare the performance of the new IMC algorithm with respect to the two previously introduced algorithms (soft-impute and SVT) and therefore illustrate the effect of using side information. A rank-5 ($r = 5$) observation matrix $\mathbf{M} \in \mathbb{R}^{100\times100}$ is produced whose column and row vectors lie perfectly in the subspaces spanned by the columns of non-orthogonal side information matrices $\mathbf{A} \in \mathbb{R}^{100\times12}$ and $\mathbf{B} \in \mathbb{R}^{100\times8}$. The exact details on how this is modelled will be described in 3.4.1. The performance of both traditional matrix completion algorithms (**SVT** and **soft-impute**) on predicting the matrix entries are compared to that of the **IMC** algorithm (which made use of the side information) while varying $p$, the proportion of entries missing in the observation matrix $\mathbf{M}$. For each value of $p$, 10 different matrices are 'completed' and the average relative error and average runtime (in seconds) measured as defined in 2.4.2. The results are shown in Figure 6 below.



(a) Average error



(b) Average runtime

Figure 6: Performance of **soft-impute**, **SVT** and **IMC** for $100 \times 100$ matrices of rank 5, with $100 \times 12$, $100 \times 8$ perfect side information matrices and varying $p$. Error bars representing one standard deviation.
.

Figure 6a shows that for low values of $p$, IMC achieves a similar performance to the other two algorithms. For $p < 0.6$, all three achieve excellent average error rates, of the order of $10^{-3}$ or below. More importantly, for a larger proportion of missing entries, the average error rapidly increases for **Soft-Impute** and especially **SVT** whereas it remains approximately constant at $10^{-3}$ for **IMC**. The reason for this is that, as the number of observed entries in the matrix decreases, the standard matrix completion methods have less known information they can use to estimate the missing entries, whereas **IMC** makes use of the perfect side information which limits the possible column and row vectors of the matrix estimate $\mathbf{Z}$. Therefore, unlike **soft-impute** and **SVT**, **IMC** is still able to achieve a high performance when less than 20% of the entries are known. In other words, as described at the start of this section, the use of side information allows the low-rank matrix to be recovered when a smaller number of entries are observed. In addition, the error bars for **IMC** are small throughout, illustrating that the algorithm has a very consistent performance.

Furthermore, Figure 6b shows **IMC** is also generally faster than **SVT** and **soft-impute**. **Soft-impute** has an average runtime greater than 1 second throughout, with runtime increasing with $p$ and the difficulty of estimating the matrix entries increases. Similarly, although **SVT** initially offers a very low runtime, below 0.1 seconds (caused by the algorithm converging very rapidly for a high proportion of observed entries and thus reaching the stopping criterion very early), it quickly shoots up to above 2 seconds for $p > 0.4$. **IMC**, on the other hand, remains below 0.2 seconds for alll values of $p$, that is, at least 10 times faster than **soft-impute** in every instance. This is expected, as the optimization problem which **IMC** attempts to solve (15) concerns soft-thresholding only a small $r_a \times r_b = 12 \times 8$ matrix, whereas standard matrix completion algorithms are attempting to estimate the full, $m \times n = 100 \times 100$ matrix. As every iteration involves multiple operations on the matrix (including singular value decomposition or matrix multiplications), these will be computationally cheaper (and require use of less memory) for a reduced-size matrix leading to a much lower runtime overall. In fact, as the difference between the values of $r_a, r_b$ and $m, n$ increases, the reduction in runtime achieved by using side information via **IMC** will become more significant.

## 3.3 Matrix Completion with Noisy Side Informaton

In practice, most given feature matrices $\mathbf{A}$ and $\mathbf{B}$ will not perfectly capture the column and row spaces. In other words, the assumption made in (14) will not be true. Consequently, following the model developed by Chiang et al. [5], we build a new model which balances observations and side information in order to achieve matrix completion. Essentially, the estimate $\hat{\mathbf{Z}} \in \mathbb{R}^{m\times n}$ consists of 2 parts:

- An estimate from the feature space $\mathbf{A}\hat{\mathbf{Z}}_M\mathbf{B}^\top$, where $\mathbf{Z}_M \in \mathbb{R}^{r_a \times r_b}$.
- The estimate for the noisy part outside the feature space, obtained purely from observations, $\hat{\mathbf{Z}}_N$, where $\mathbf{Z}_N \in \mathbb{R}^{m\times n}$.

By replacing the estimate matrix $\mathbf{Z}$ in the original objective function in (4) with the combination of $\mathbf{A}\mathbf{Z}_M\mathbf{B}^\top$ and $\mathbf{Z}_N$, the following objective function is obtained:

$$\min_{\mathbf{Z}_M, \mathbf{Z}_N} \left\{ \frac{1}{2} \sum_{(i,j)\in\Omega} \left( m_{ij} - (\mathbf{A}\mathbf{Z}_M\mathbf{B}^\top)_{ij} - (\mathbf{Z}_N)_{ij} \right)^2 + \lambda_M \|\mathbf{Z}_M\|_* + \lambda_N \|\mathbf{Z}_N\|_* \right\}. \tag{18}$$

### 3.3.1 Noisy Side Information Algorithm

As for matrix completion with perfect side information, there are several cases of algorithms presented in existing literature which make use of noisy side information models [5], but our intention during the project has been to develop a slightly simpler algorithm developed as an extension of the original soft-impute algorithm (Sec. 2.2.2) and the related IMC algorithm (Sec. 3.2). This noisy side information algorithm will be referred to as **dirtyIMC Algorithm**.

```
dirtyIMC Algorithm

1. Initialize $\mathbf{Z}_M^{\text{old}} = \mathbf{0}, \mathbf{Z}_N^{\text{old}} = \mathbf{0}$ and $\lambda_M = c\alpha, \lambda_N = \beta\lambda_M$
2. For each $k = 1, ..., K,$ iterate until convergence:
   Optimise $\mathbf{Z}_M \in \mathbb{R}^{r_a \times r_b}$, with $\mathbf{Z}_N$ fixed:
       Compute $\hat{\mathbf{Z}}_{M_k} \leftarrow \mathcal{S}_{\lambda_M}\left(\mathbf{A}^+\left(P_\Omega(\mathbf{M} - \mathbf{Z}_N^{\text{old}}) + P_\Omega^\perp\left(\mathbf{A}\mathbf{Z}_M^{\text{old}}\mathbf{B}^\top\right)\right)\mathbf{B}^{+^\top}\right)$
       Update $\mathbf{Z}_M^{\text{old}} \leftarrow \hat{\mathbf{Z}}_{M_k}$
   Optimise $\mathbf{Z}_N \in \mathbb{R}^{m \times n}$, with $\mathbf{Z}_M$ fixed:
       Compute $\hat{\mathbf{Z}}_{N_k} \leftarrow \mathcal{S}_{\lambda_N}\left(P_\Omega(\mathbf{M} - \mathbf{A}\mathbf{Z}_M^{\text{old}}\mathbf{B}^\top) + P_\Omega^\perp\left(\mathbf{Z}_N^{\text{old}}\right)\right)$
       Update $\mathbf{Z}_N^{\text{old}} \leftarrow \hat{\mathbf{Z}}_{N_k}$
3. Final output is $\hat{\mathbf{Z}}_K = \mathbf{A}\hat{\mathbf{Z}}_{M_K}\mathbf{B}^\top + \hat{\mathbf{Z}}_{N_K}$
```

For every iteration, the principle of this algorithm is to alternately optimise the two parts of the matrix estimate, $\mathbf{Z}_M$ and $\mathbf{Z}_N$, one at a time, while the other part remains fixed. The update operation on $\mathbf{Z}_M$ is similar to that from Sec. 3.2, except that the estimate for the part outside the feature space $\mathbf{Z}_N$ is subtracted first from the observation matrix $\mathbf{M}$. Similarly, the optimisation of $\mathbf{Z}_N$ follows the soft-impute algorithm (Sec. 2.2.2) but subtracting the feature space estimate $\mathbf{A}\mathbf{Z}_M\mathbf{B}^\top$ from the observation matrix $\mathbf{M}$.

Compared to the previous algorithms, **dirtyIMC** has more variable tuning parameters whose value will depend on the quality of the feature matrices. This, however, is not easily measured and therefore successfully tuning these parameters can be troublesome. Unlike for soft-impute and the IMC algorithm, the thresholding parameters $\lambda_M$ and $\lambda_N$ were chosen to remain constant over all iterations, as this proved to obtain a more consistent performance.

The parameter $\beta$, which determines the ratio $\lambda_N/\lambda_M$, is crucial in controlling the importance between the feature estimate $\mathbf{A}\mathbf{Z}_M\mathbf{B}^\top$ and the residual $\mathbf{Z}_N$, but very difficult to define without knowing the quality of features. For $\beta$ very large, $\mathbf{Z}_N$ will be enforced to zero and the objective function (18) will become that for **IMC** (15). On the other hand, as $\beta \to 0$, $\mathbf{Z}_M$ will be enforced to zero, side information disregarded and the objective function become that for standard matrix completion (4).

With respect to parameter $\alpha$, it depends on the proportion of entries missing, $p$. When the proportion of missing entries in $\mathbf{M}$ increases, $\lambda_M$ and $\lambda_N$ should both increase as greater importance should be given to enforcing the low rank constraint than to minimising the error in the observed entries (this error has less validity). Through experimentation, I obtained the following rule for setting $\alpha$ as a function of $p$:

$$\alpha = \begin{cases} 0.05 & p \leq 0.2 \\ \\ p^{3/2} & 0.2 < p \leq 0.7 \\ \\ 1 & p > 0.7 \end{cases} \tag{19}$$

Finallly, parameter $c$ is a constant which sets the relative magnitudes of $\lambda_M$ and $\lambda_N$ with respect to the feature matrices. $c$ is chosen to be proportional to the average of the largest singular values of side information matrices $\mathbf{A}$ and $\mathbf{B}$, $\left(\|\mathbf{A}\|_2 + \|\mathbf{B}\|_2\right)/2$, although its ideal magnitude will vary from one application to another.

## 3.4 Simulations

### 3.4.1 Methodology

In order to perform matrix completion with side information on simulated data, we produced random, unbiased observation matrices $\mathbf{M}$ whose columns and rows were contained (or partly contained in the noisy case) in the feature space of two other simulated side information matrices $\mathbf{A}$ and $\mathbf{B}$. The method followed is described below.

First, two matrices $\mathbf{Z}_A \in \mathbb{R}^{r_a \times r}$ and $\mathbf{Z}_B \in \mathbb{R}^{r_b \times r}$ are generated, with their entries randomly sampled from a normal distribution with mean 0 and standard deviation 5 to give some variance in the values of the entries,i.e. $\mathbf{Z}_A, \mathbf{Z}_B \overset{\text{iid}}{\sim} \mathcal{N}(0, 5^2)$. A reduced size, $r_a \times r_b$ matrix is produced as $\mathbf{Z}_0 = \mathbf{Z}_A \mathbf{Z}_B^\top \in \mathbb{R}^{r_a \times r_b}$. This reduced size matrix $\mathbf{Z}_0$ is essentially what we expect to produce as the output of the **IMC** algorithm when there is perfect side information.

On the other hand, side information matrices $\mathbf{A} \in \mathbb{R}^{m \times r_a}$ and $\mathbf{B} \in \mathbb{R}^{n \times r_b}$ are produced by randomly sampling a normal distribution with mean 0, standard deviation 1 ($\mathbf{A}, \mathbf{B} \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$) and non-orthogonal columns. Each column of $\mathbf{A}$ and $\mathbf{B}$ is then normalised so they would all have an $l_2$-norm of 1 (to avoid differences in magnitudes from column to column). The full size $m \times n$ observation matrix to be used for the simulations is then produced as $\mathbf{M} = \mathbf{A}\mathbf{Z}_0\mathbf{B}^\top$.

As for standard matrix completion, a proportion $p$ of the entries of $\mathbf{M}$ are removed by Bernoulli sampling with probability $(1 - p)$.

Where the noisy side information model is investigated, the columns of the side information matrices $\mathbf{A}$ and $\mathbf{B}$ must be altered so that the columns and rows of $\mathbf{M}$ no longer lie in their feature space. Each column $\mathbf{v}_i \in \mathbb{R}^m$ of matrix $\mathbf{A}$ is replaced with probability $\rho$ (referred to as noise level) by the component of a random vector $\mathbf{x} \in \mathbb{R}^m$ orthogonal to $\mathbf{A}$. The method is outlined below:

$$\begin{aligned} &\text{for } i = 1 \text{ to } r_a : \\ &\quad - \ \mathbf{x} \overset{\text{iid}}{\sim} \mathcal{N}(0,1) \\ &\quad - \ \text{With probability } \rho, \ \text{replace } \mathbf{v}_i \text{ by } \mathbf{x} - \mathbf{P}^{\parallel}_{\mathbf{A}}\mathbf{x}, \end{aligned} \tag{20}$$

where $\mathbf{P}^{\parallel}_{\mathbf{A}} = \mathbf{A}\mathbf{A}^{+} = \mathbf{A}(\mathbf{A}^{\top}\mathbf{A})^{-1}\mathbf{A}^{\top}$ is the projection matrix onto the column space of $\mathbf{A}$ and thus $\mathbf{P}^{\parallel}_{\mathbf{A}}\mathbf{x}$ is the component of $\mathbf{x}$ parallel (in column space) to $\mathbf{A}$.

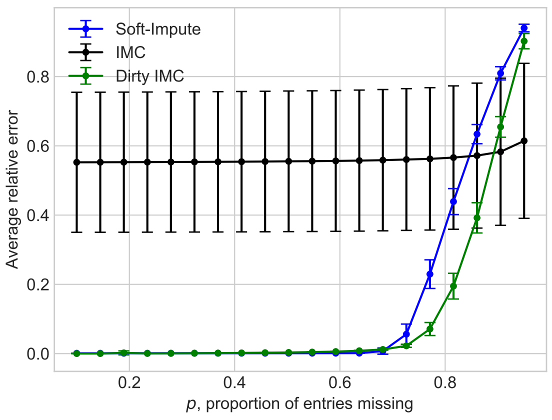
The same method of producing noisy side information applies to $\mathbf{B}$, but with dimension variables $m$ and $r_a$ replaced by $n$ and $r_b$ respectively.

### 3.4.2 Results and Discussion

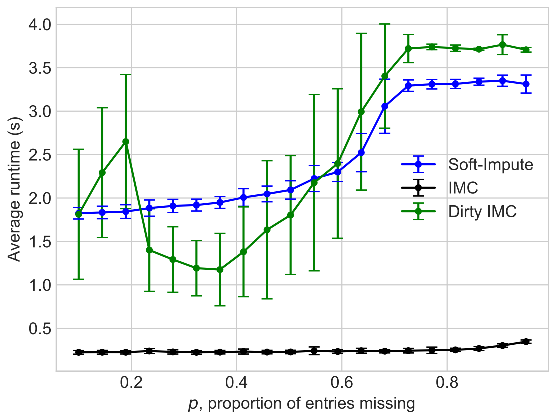
Plots are produced to compare the performance of **dirtyIMC** to that of **IMC** and the standard **soft-impute**. **dirtyIMC** is applied as defined in Sec. 3.3.1, with $\beta = \lambda_N/\lambda_M = 1$ to provide a good balance between the feature and noise estimates and $c = (\|\mathbf{A}\|_2 + \|\mathbf{B}\|_2)/3$. In each case, the observation matrices $\mathbf{M}$ are of rank 5 ($r = 5$) with dimensions $100 \times 100$, $r_a = 12$ and $r_b = 8$. Figure 7 shows the performance for the different algorithms for varying $p$ and two different feature noise levels, $\rho = 0.2, 0.6$. As in previous cases, for each value of $p$, 10 different matrices are 'completed' and the average relative error and average runtime (in seconds) measured as defined in Sec. 2.4.2.

Figures 7a and 7c show that, as expected, under noisy side information, the perfect side information model is no longer valid and hence **IMC** performs significantly worse than both **soft-impute** and **dirtyIMC**. In addition, the average error in IMC increases with noise level (from below 0.6 at $\rho = 0.2$ to above 0.9 at $\rho = 0.6$) and remains almost constant as the proportion of missing entries increases.

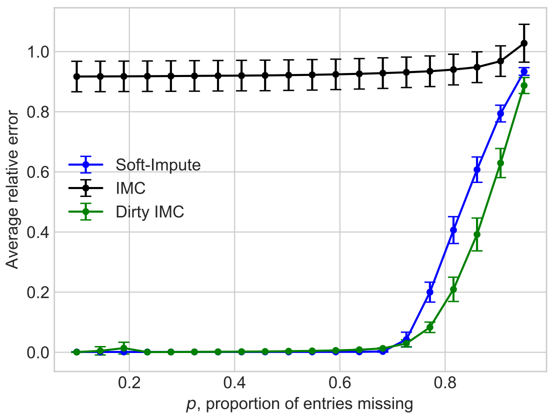
Comparing **soft-impute** and **dirtyIMC**, their performance is excellent (error of the order of $10^{-3}$) and similar for $p < 0.7$. However, for larger $p$, **dirtyIMC** is observed to perform significantly better thanks to the feature-based part of the matrix estimate, $\hat{\mathbf{Z}}_M$. In fact, for $p > 0.9$ and low feature noise level (such as $\rho = 0.2$), the perfect side information model **IMC** provides the best performance, as there are too few entries available to achieve an accurate low rank estimate by standard matrix completion.



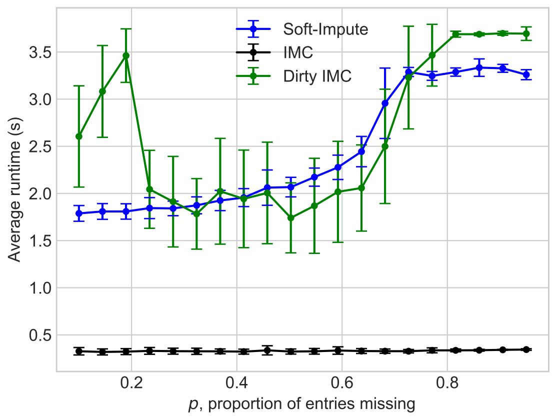
(a) Average error, $\rho = 0.2$



(b) Average runtime, $\rho = 0.2$



(c) Average error, $\rho = 0.6$

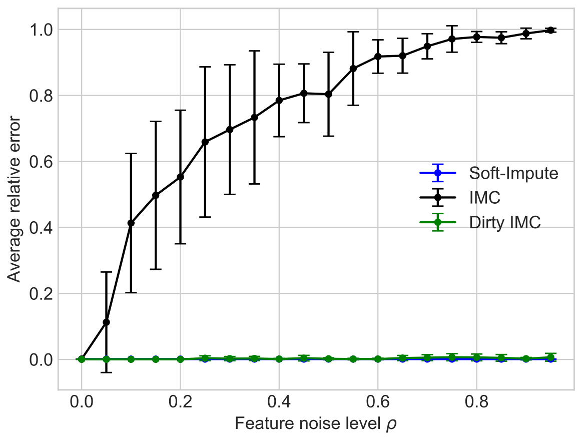


(d) Average runtime, $\rho = 0.6$

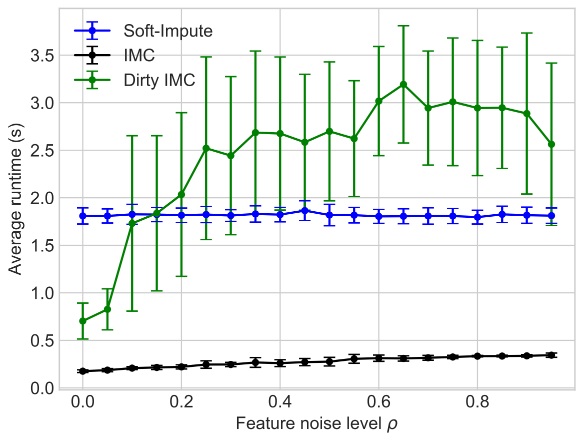
Figure 7: Performance of **soft-impute**, **IMC** and **dirtyIMC** for $100 \times 100$ matrices of rank 5, with $100 \times 12$, $100 \times 8$ perfect side information matrices, feature noise level $\rho$ and varying $p$. Error bars representing one standard deviation.

On the other hand, Figures 7b and 7d show that the runtime for **dirtyIMC** (between 1 and 4 seconds) is significantly worse than that for **IMC** (0.2 to 0.3 seconds), as a full-sized $m \times n$ matrix $\hat{\mathbf{Z}}_N$ must be learned as well as the reduced-size $r_a \times r_b$ matrix $\hat{\mathbf{Z}}_M$. However, the addditonal matrix update in every iteration compared to **soft-impute** does not seem to have a significant effect on runtime, with both plots showing similar behaviour. Generally, the runtime appears to increase with $p$ as the number of observed entries in $\mathbf{M}$ falls and the optimization problem becomes more troublesome.

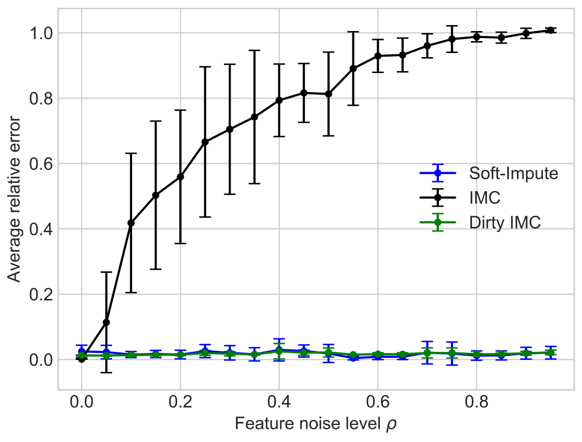
Similar simulations are performed for varying feature noise levels $\rho$ while keeping the proportion of fixed entries $p$ set to fixed values: 0.2, 0.7, 0.8. Again, the observation matrices $\mathbf{M}$ are of rank 5 ($r = 5$) with dimensions $100 \times 100$, $r_a = 12$ and $r_b = 8$. 10 trials where run for each combination of $p$ and $\rho$ and the average relative error and average runtime (in seconds). The results are shown in Fig. 8.



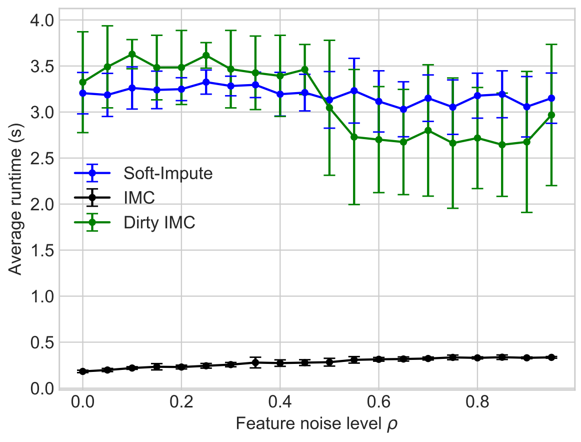
(a) Average error, $p = 0.2$



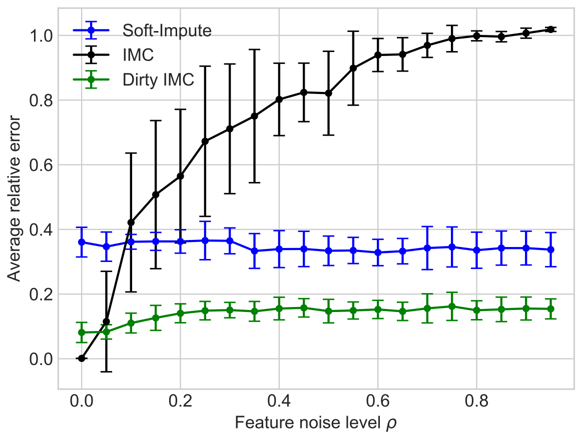
(b) Average runtime, $p = 0.2$



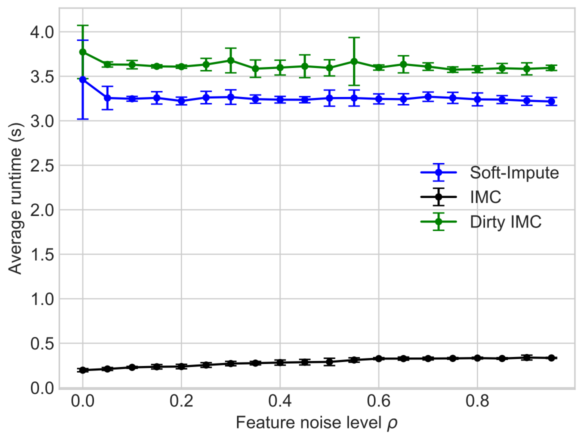
(c) Average error, $p = 0.7$



(d) Average runtime, $p = 0.7$



(e) Average error, $p = 0.8$



(f) Average runtime, $p = 0.8$

Figure 8: Performance of **soft-impute**, **IMC** and **dirtyIMC** for $100 \times 100$ matrices of rank 5, with $100 \times 12$, $100 \times 8$ perfect side information matrices, proportion of missing entries $p$ and varying feature noise level $\rho$ (as defined by Eq. (20)). Error bars represent one standard deviation.

Fig. 8 confirms that the average error for **IMC** drastically increases with feature noise level, as less of the column and row space of $\mathbf{M}$ lies within the subspace spanned by the column spaces of $\mathbf{A}$ and $\mathbf{B}$. Although **soft-impute** and **dirtyIMC** appear to have similar performance up to $p = 0.7$, we observe a significant reduction in the average error from **soft-impute** (error above 0.35) to **dirtyIMC** (error below 0.2 for all $p$) for $p = 0.8$. In addition, for the three methods, the average runtime appears to be independent of the feature noise level (except for some exceptions) and the runtime of **dirtyIMC** only becomes slightly higher than that of **soft-impute**, but always within the same order of magnitude.

Overall, the **dirtyIMC** algorithm provides a significant improvement with respect to **IMC** when the perfect side information assumption is not valid, although at the cost of an increase in runtime by a factor of 10. With respect to **soft-impute**, **dirtyIMC** provides benefits in performance when a small proportion of entries are known. This is often the case in many real-world applications, making **dirtyIMC** a very usable algorithm.

# 4 Real-world Application: Predicting gene-disease associations

In order to confirm the high performance achieved by **IMC** and **dirtyIMC**, as well as to illustrate how they can be used in the real-world, matrix completion with side information is applied to the prediction of gene-disease associations. The background information, the raw data used and general methodology used to apply matrix completion to this real-world problem follow the previous research done by Natarajan and Dhillon [12].

Predicting how likely genes are of being related to certain diseases is a crucial step in discovering causal genes and understanding genetic disorders. In addition, gene-disease associations show similarities with recommender systems, (such as the user-movie ratings example previously mentioned) in that the goal is to predict the strength of association (or rating) of a 'user' (a gene) with a given 'item' (a certain disease). Furthermore, there are multiple sources from which to extract feature information, such as biomedical literature, intrinsic gene properties or gene-phenotype (the observable characteristics of an organism) relationships. This makes Matrix Completion with side information a perfect optimization method for this application. In fact, unlike other network-based methods, inductive matrix completion allows these associations to be made for new diseases, where they are particularly useful.

## 4.1 Model

In the gene-disease matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, the rows of the matrix represent the genes ($m =$ total number of genes) and the columns represent different diseases ($n =$ total number of diseases). This observation matrix $\mathbf{M}$ was obtained from Natarajan and Dhillon's dataset, originally from the OMIM project [12], with $m = 12331$ genes and $N_d = 3209$ diseases. This is a much larger matrix than those studied during the simulations (Sec. 3.4).

For each entry $\mathbf{M}_{ij}$ of matrix $\mathbf{M}$, $\mathbf{M}_{ij} = 1$ if gene $i$ is linked to disease $j$ and 0 if either the gene and disease are not linked or the relationship is unobserved. The 0 entries therefore have a slightly different meaning to our previous examples, where they only represented unobserved associations. Due to the nature of the application, there are no negative associations, that is, cases of confirmed absence of a gene-disease connection. Also, matrix $\mathbf{M}$ is assumed to have a low rank $k$, where $k \ll m, n$,

The number of positive gene-disease associations are typically small and thus the observation matrix $\mathbf{M}$ is extremely sparse. For the 12331 genes and 3209 diseases, there are only 3954 total gene-disease associations (number of non-zero entries in the matrix). This corresponds to only 0.01% (proportion of missing entries $p = 99.99\%$) of the matrix's entries, with all columns having at least one non-zero entry (all diseases linked to at least 1 gene) but approximately 75% of the rows (genes) having no non-zero entries. This extreme sparsity increased the difficulty (compared to the simulations presented before) of achieving meaningful results and meant that side information would be essential for it, so the standard matrix completion algorithms (**SVT** and **soft-impute**) were not considered.

## 4.2 Feature matrices

The feature matrices $\mathbf{A} \in \mathbb{R}^{m\times r_a}$ and $\mathbf{B} \in \mathbb{R}^{n\times r_b}$, with $r_a = 300$ and $r_b = 200$, correspond to gene and disease side information respectively. The columns of these two matrices are extracted from various different data sources, by dimensionality reduction.

Gene feature matrix $\mathbf{A} \in \mathbb{R}^{m\times r_a}$, $m = 12331, r_a = 300$, has 300 columns which correspond to:

1. The top 100 principal components from microarray measurements data (obtained via PCA)
2. The leading 100 eigenvectors of the gene interaction network HumanNet
3. The leading 100 singular vectors of a gene-phenotype association matrix

Disease feature matrix $\mathbf{B} \in \mathbb{R}^{n\times r_b}$, $n = 3209, r_b = 200$ has 200 columns, which correspond to:

1. The leading 100 eigenvectors of the disease similarity network MinMiner
2. The top 100 principal components from a term-document OMIM diseases matrix (obtained via PCA)

In practice, extracting these features in the correct way can prove difficult and will have great consequences in the results obtained. For the gene interaction network and the disease similarity network, these are simply symmetric matrices and therefore the first 100 eigenvectors can be taken directly. Similarly, for the gene-phenotype associations matrix, Singular Value Decomposition was performed and the first 100 singular vectors extracted as columns.

For the remaining two data sources, where Principal Component Analysis (PCA) is used, the feature extraction method is more complex. In both cases, the feature vectors extracted are the first 100 eigenvectors of the covariance matrix. Before computing the covariance matrix, the columns of the original matrix must be zero-centred (by subtracting their mean). This allows us to obtain the covariance matrix simply by pre-multiplying the matrix by its transpose, i.e. if we have a centred data matrix $\mathbf{X}$ of size $n \times p$, then the symmetric $p \times p$ covariance matrix $\mathbf{C}$ will be given by $\mathbf{C} = \mathbf{X}^\top \mathbf{X}/(n-1)$. In addition, the need for standardising each column as well (so they all have standard deviation of 1) is investigated. The best results were achieved for PCA in which the columns had previously been centred but not standardised. Nevertheless, the effects of standardisation and centering will be discussed in the results below.

## 4.3 Results and Discussion

### 4.3.1 Results Evaluation

Due to the slightly different nature of this problem (with only a few entries being 1 and the rest 0) compared to the previous cases, a new way of measuring performance is introduced to replace the error measure used previously. In addition, this allows us to make a comparison between our results and those achieved by Natarajan and Dhillon [12].

The quantitative evaluation method follows a cross-validation strategy. First, the known gene-disease associations (observed entries in the matrix $\mathbf{M}$) are split into 3 equally-sized groups. The associations in one of the 3 groups are hidden while those in the other two groups are used to form the observation matrix $\mathbf{M}$ and attempt matrix completion. This is repeated 3 times, so that each group is hidden once.

Once the algorithm is run and the final output obtained ($\mathbf{A}\hat{\mathbf{Z}}_K\mathbf{B}^\top$ for **IMC**, $\hat{\mathbf{Z}}_K$ for **dirtyIMC**), we iterate through the $n$ diseases in our dataset (the columns of our matrix $\mathbf{M}$). For each disease, the genes (entries of the column) are sorted in descending order of value (from more positive to more negative), i.e. by how strongly the algorithm predicts that they are linked to the disease. Then, for every known gene-disease association $(i, j)$ in the hidden group, the rank (position in the ordered list) of gene $i$ for disease $j$ is determined. The performance is then measured as the probability of the hidden genes for each disease having a rank less than a threshold $r$. In other words, we evaluate the probability that the hidden genes are within the top $r$ positive genes more strongly linked with that disease.

### 4.3.2 Perfect Side Information Model - IMC Algorithm

The IMC algorithm was first applied to solving this problem by using matrix completion with perfect side information. From the objective function in Eq. (15), the IMC algorithm learns only a matrix of dimensions $r_a \times r_b = 300 \times 200$ and thus has a low average runtime. This matrix is assumed to have column space and row space which lie in the column spaces of side information matrices $\mathbf{A}$ and $\mathbf{B}$ respectively. Therefore, it relies on the accuracy of the feature matrices.

The algorithm was applied as defined in Sec. 3.2, with the sole difference that $\lambda_0$ was set to $\lambda_0 = \|P_\Omega(\mathbf{M})\|_2/0.08$, that is, almost 20 times larger than for previous simulations. Looking back at the objective function (15), the reason for this change is that, due to the extreme sparsity of the observation matrix $\mathbf{M}$, $\|P_\Omega(\mathbf{M})\|_2$ will be small and so this value of $\lambda_0$ ensures $\lambda$ is larger enough to enforce the rank constraint on $\mathbf{Z}$ (second term in the objective function).

The results obtained (in terms of the probability of hidden genes being among top $r$ genes) for the **IMC** algorithm are shown in Fig. 9 for varying values of rank $r$.

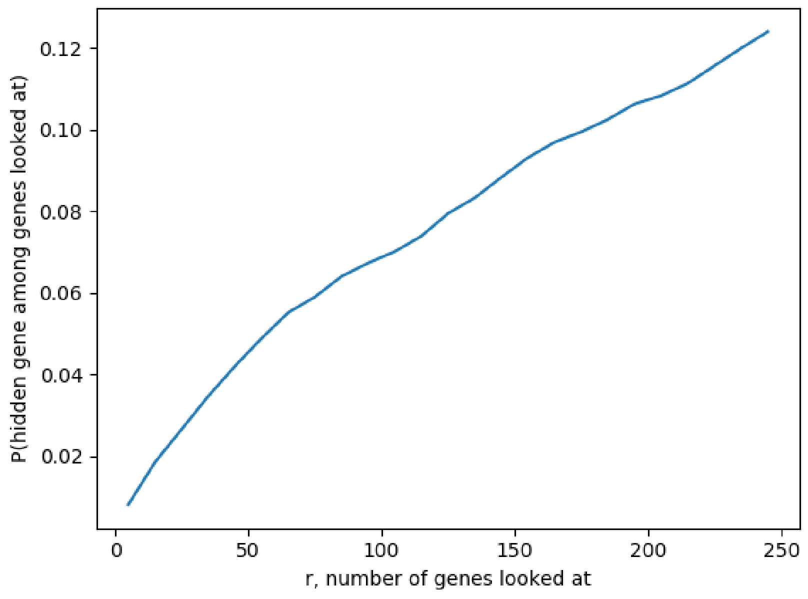


Figure 9: Performance of **IMC** algorithm for predicting gene-disease associations.

The graph shows that, for a given disease, when looking at only the top 200 of the 3209 genes, we have more than 10% chance of finding the gene that is confirmed to be linked to that disease and almost 8% when only looking at the top 50 genes. As expected, the more genes that are looked at (when $r$ increases), the higher the chance of finding the confirmed, associated gene among those looked at. The results are compared to those obtained by the various methods presented in Natarajan and Dhillon's paper [12], which result in probabilities between 6 and 24% at $r = 100$. Considering that they apply more elaborate methods to obtain the observation matrix $\mathbf{M}$ and more complex algorithms, we can conclude that **IMC** provides an acceptable performance in predicting gene-disease associations while maintaining simplicity as an algorithm. In addition, although the side information s likely to not be completely perfect, the assumption that $\mathbf{A}$ and $\mathbf{B}$ fully describe the latent feature space of $\mathbf{M}$ is valid for this application.

### 4.3.3 Investigating PCA method

In order to determine what settings would give the best results, the effect of zero-centering and standardising when performing PCA on the original data was investigated. Figure 10 shows plots of the performance of the **IMC** algorithm where PCA was performed on non-centred (not standardised either) columns (green), centred non-standardised columns (blue), and centred standardised columns (orange).

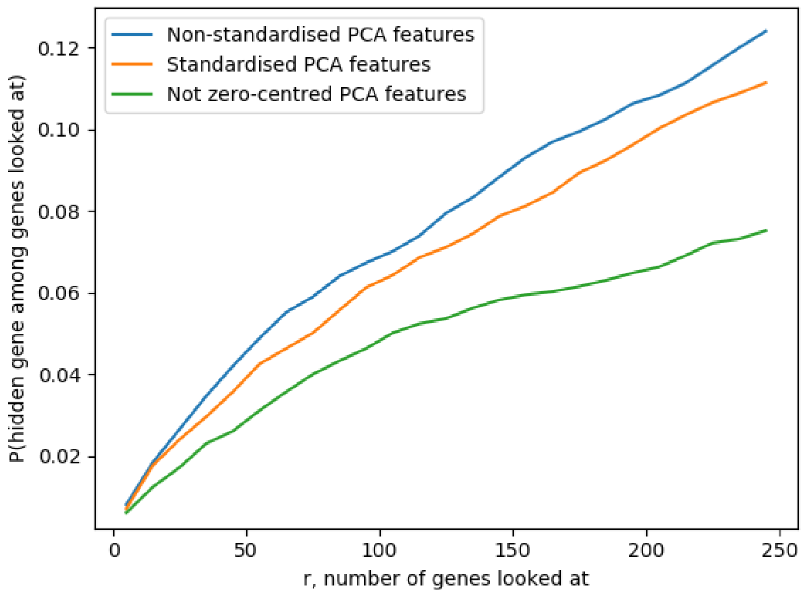


Figure 10: Plot for **IMC** performance without centering, and with and without standardising the raw data before performing PCA.

As expected, the plots for centred PCA features give significantly better performance than the plot without centering, with the difference being more than 2% at $r = 100$. Centering is necessary in order for the product of the matrix and its transpose (as explained before) to be equal to the covariance matrix. Practically, PCA essentially orthogonalises the raw data. In this sense, zero-centering ensures that one column does not dominate over the rest, as we essentially want to extract the directions of these columns and are not worried about their relative magnitudes.

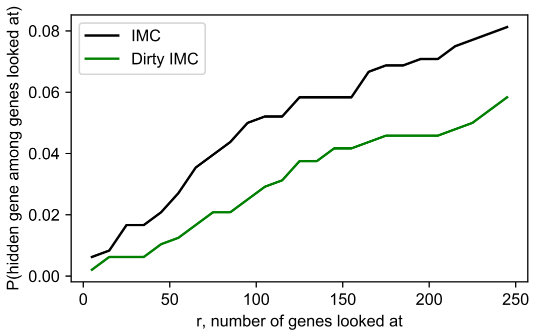
With respect to standardising, Fig. 10 shows that slightly better performance is obtained when the columns are centred but not standardised. The need for standardising is dependent on the dataset and it does not always prove to be useful (as is the case here). Dividing by the standard deviation for each column can prove useful when the datset contains information at different scales, such as one column expressed in meters and another in kilometres, in order to bring all columns to the same range of values. However, ther reason for the reduced performance achieved here is that standardising also results in some of the variance information in the original dataset being lost.

Overall, PCA was determined to suit the problem better when it is performed after zero-centering but not standardising the columns.

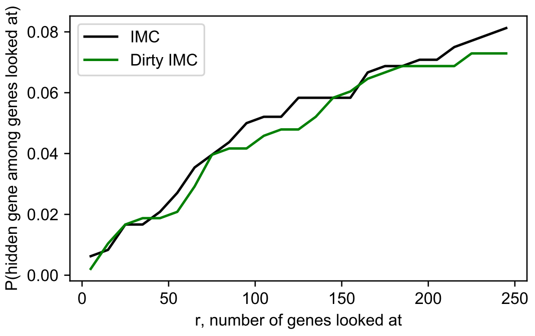
### 4.3.4 Noisy Side Information Model - dirtyIMC Algorithm

Next, the optimization problem was also attempted with **dirtyIMC** to find out if it suited a dirty side information model better. Looking back at the objective function for the side information model (18), the **dirtyIMC** algorithm learns two different matrices simultaneously: a reduced size $300 \times 200$ estimate matrix $\hat{\mathbf{Z}}_M$ corresponding to the feature space and a full size, $m \times n$ matrix $\hat{\mathbf{Z}}_N$ corresponding to the noisy part outside the feature space. As the **dirtyIMC** algorithm has to learn a much larger matrix than **IMC** (as well as the reduced size matrix), the average runtime for the algorithm will be significantly larger (up to 10 hours when the number of iterations is set to 2000). Consequently, in order to run in an acceptable runtime, the number of genes and diseases analysed in the dataset were reduced to a third of the initial numbers, that is, $m = 4111, n = 1070$. This was done by subsampling the observation matrix $\mathbf{M}$ (keeping only the first of every 3 columns and rows) and feature matrices $\mathbf{A}$ and $\mathbf{B}$ (taking the first of every 3 rows, discarding all other rows) previously produced from the raw data.

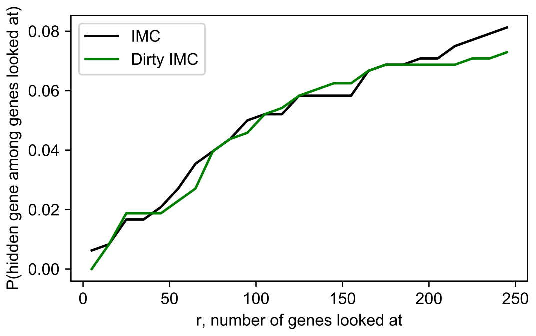
The **dirtyIMC** algorithm is applied as defined in Sec. 3.3.1, with parameter $c$ set to $\left(\|\mathbf{A}\|_2 + \|\mathbf{B}\|_2\right)/15$ giving $\lambda_M \approx 0.1 - 0.2$ and different values of $\beta = \lambda_N/\lambda_M$. Their performance is compared to that of the **IMC** algorithm and the results have been shown in Figure 11.



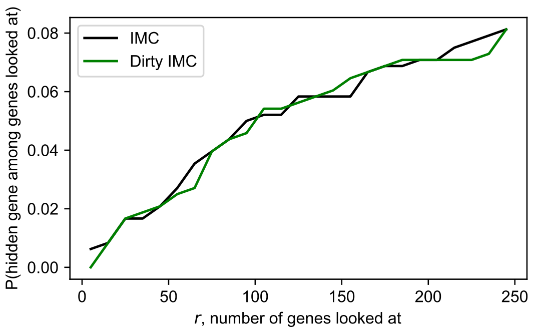
(a) $\lambda_N = \lambda_M$



(b) $\lambda_N = 5\lambda_M$



(c) $\lambda_N = 8\lambda_M$



(d) $\lambda_N = 10\lambda_M$

Figure 11: Comparison of the performance of the **dirtyIMC** algorithm, with different choices of $\beta = \lambda_N/\lambda_M$, and **IMC** in predicting the existing gene-disease associations.

First of all, it is worth noting that all 4 plots show lower probabilities for **IMC** than those originally achieved and shown in Fig. 9. Due to the subsampling, fewer observed entries are available for matrix estimation and thus the performance of **IMC** will drop.

**IMC** is observed to outperform **dirtyIMC** for all 4 settings of the ratio $\beta = \lambda_N/\lambda_M$.

For $\beta = 1$, **dirtyIMC**'s curve is significantly below that for **IMC**. Looking back at the objective function (18), with $\lambda_N = \lambda_M$ the same importance is given to imposing a low rank on $\mathbf{Z}_M$ and $\mathbf{Z}_N$. In addition, the first term in the objective function (squared error for indices in $\Omega$ corresponding to known entries) will always be small relative to the other 2 terms, as only 0.01% of the entries are known. Therefore, the objective function can be minimised by reducing $\lambda_M \|\mathbf{Z}_M\|_*$ and these will produce a very inaccurate estimate for the matrix which lies almost completely outside the feature space defined by $\mathbf{A}$ and $\mathbf{B}$

As $\beta$ increases, the performance of **dirtyIMC** becomes increasingly close to that of **IMC**. Greater importance is given on imposing the low rank constraint for $\mathbf{Z}_N$ (noisy part outside the feature space) and therefore the matrix estimates become increasingly aligned with the side information matrices and with the estimates produced by the perfect side information model. If $\beta$ were to be increased further and approach infinity, the curve for **dirtyIMC** would be identical to that for **IMC**.

However, no intermediate value of $\beta$ was found for which the probabilities obtained through **dirtyIMC** surpassed **IMC**. The nature of the algorithm is that $\mathbf{Z}_N$ attempts to estimate the part of the matrix which is not described by the features. However, in cases like this were very few entries are known, there is very little information provided to set constraints on this noise matrix $\mathbf{Z}_N$. The squared error term in the objective function will be so small no matter what the matrix $\mathbf{Z}_N$ is that the objective function can be minimised when an erroneous solution to $\mathbf{Z}_N$ is provided.

This behaviour agrees with that seen in Figure 7a, where the error for **dirtyIMC** rises above **IMC** when the proportion of entries missing is large ($p > 0.9$) and the feature noise level is relatively low (only 0.2). Therefore, we can reach the conclusion that perfect side information (even if it is not completely true) is a good assumption when the proportion of known entries in the matrix is small.

Overall, we can say that our **dirtyIMC** algorithm doesn't suit problems like this one where the number of known entries is a very small fraction of the size of the matrix. Nevertheless, it has not failed at solving the problem either. The mutability of **dirtyIMC** ensures that it will achieve a performance at least as good as that for **IMC** by setting $\beta$ large (although with a large runtime penalty).

# 5 Conclusions and Future Work

During this conclusion, we will summarise the main findings and results from this project as well as mention related areas of interest which could lead to further investigation.

The results obtained in Sections 3.4 and 3.5 proved that the average reconstruction error for **soft-impute** and **SVT** for standard matrix completion are comparable to that of the reference **cvxpy** while being significantly cheaper computationally. In fact, both algorithms achieve almost perfect reconstruction of the original matrix when the proportion of missing entries $p$ is below 30%.

**SVT** proved to perform slightly better for noiseless data where most of the matrix entries are known. On the other hand, **soft-impute** outperforms **SVT** significantly when the input data is noisy or most of the entries are unobserved. In addition, although both algorithms have very similar average runtimes, the **soft-impute** algorithm has a simpler structure, with less tuning parameters and it originated from a more logical rationale.

Applying the matrix completion algorithms to the real-world application of predicting movie ratings illustrated some of the main drawbacks of standard matrix completion. While acceptable results were still achieved, the large size of the data matrices (dimensions 480,000 $\times$ 18,000) and the lack of known ratings (4 and 1 % of all matrix entries) led to prohibitive runtimes and limited the accuracy of the predictions made.

A solution to the runtime problem is to use side information. Matrix completion with side information follows the assumption that the matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ shares the same latent information as $\mathbf{A} \in \mathbb{R}^{m\times r_a}$ and $\mathbf{B} \in \mathbb{R}^{n\times r_b}$. It reduces the optimisation problem to recovering a reduced-size matrix $\mathbf{Z}_0 \in \mathbb{R}^{r_a\times r_b}$.

The **IMC** algorithm for matrix completion with side information was developed as a simple extension of **soft-impute**, exploiting projection onto the column spaces of side information matrices **A** and **B**. Despite its simplicity, **IMC** performs better than the standard matrix completion algorithms (especially when a large proportion of the entries are unobserved) while attaining a significant drop in runtime.

Where side information is not perfect, a noisy side information model was introduced which balances an estimate for the feature space and one for the noisy part outside the feature space. **dirtyIMC** was developed, again with **soft-impute** as the starting point. Although **dirtyIMC** has a much higher average runtime than **IMC**, it was found to outperform **IMC** for noisy feature matrices. **dirtyIMC** also achieved lower relative errors than the standard **soft-impute** when the proportion of known entries in the observation matrix was small (below 30 %). By changing the value of parameter $\beta = \lambda_N/\lambda_M$, the

**dirtyIMC** is able to 'mutate' from standard **soft-impute** to **IMC**. However, its multiple parameters proved to be difficult to tune and further investigation could be carried out to deduce the optimal settings for $\alpha, \beta$ and $c$.

Matrix completion with side information was then applied to the problem of predicting gene-disease associations. Despite the extreme sparsity of the observation matrix (only 0.01 % of confirmed associations), there was a 7 % probability of finding the known, associated genes among the top 100 genes (of a total of 12331 genes) when using **IMC**. On the other hand, **dirtyIMC** was not able to outperform **IMC**, as the noisy side information model (and especifically the noise estimate $\mathbf{Z}_N$) is not suited for such a lower proportion of known entries.

Data extraction from other data sources to produce the gene and disease feature matrices proved to be extremely difficult and for techniques like PCA, data processing such as standardising and normalising can have a big impact on the success of the algorithms. Little research has been done so far on how these side information matrices should be formed.

**Future Work**

Although matrix estimation is already a broad field, there are several open questions about how to tailor the optimisation technique to suit new applications.

In many applications, such as census data, the data matrices are heterogeneous, i.e., the matrix may consist of a mixture of numerical, Boolean, categorical and ordinal data types. A key open question is: how can matrix completion be performed in such settings? Some existing work has explored the use of loss functions to embed different data types (including ordinal and Boolean) into real numbers in the context of principal component analysis (PCA) [13], but there is still room for investigation on how to optimally use side information for such heterogeneous data matrices in order to perform efficient matrix completion.

Moreover, in some applications, the data cannot be approximated well by a low-rank matrix. More powerful **latent variable models** can be used instead to capture the underlying structure of the matrix [9]. Further work could be done to explore how the techniques developed for low-rank matrix estimation can be generalized to the case of latent variable factors and therefore serve other real-world applications.

## References

[1] James Bennett, Stan Lanning, and Netflix Netflix. “The Netflix Prize”. In: *In KDD Cup and Workshop in conjunction with KDD*. 2007.

[2] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. “A Singular Value Thresholding Algorithm for Matrix Completion.” In: *SIAM Journal on Optimization* 20.4 (2010), pp. 1956–1982. URL: http://dblp.uni-trier.de/db/journals/siamjo/siamjo20.html#CaiCS10.

[3] E. J. Candes and Y. Plan. “Matrix Completion With Noise”. In: *Proceedings of the IEEE* 98.6 (June 2010), pp. 925–936. ISSN: 1558-2256. DOI: 10.1109/JPROC.2009.2035722.

[4] Emmanuel Candès and Benjamin Recht. “Exact Matrix Completion via Convex Optimization”. In: *Commun. ACM* 55.6 (June 2012), pp. 111–119. ISSN: 0001-0782. DOI: 10.1145/2184319.2184343. URL: https://doi.org/10.1145/2184319.2184343.

[5] Kai-Yang Chiang et al. “Matrix Completion with Noisy Side Information”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 3447–3455. URL: http://papers.nips.cc/paper/5940-matrix-completion-with-noisy-side-information.pdf.

[6] Steven Diamond and Stephen Boyd. “CVXPY: A Python-embedded modeling language for convex optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[7] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman  Hall/CRC, 2015. Chap. 7, pp. 167–194. ISBN: 1498712169.

[8] Prateek Jain and Inderjit S. Dhillon. “Provable Inductive Matrix Completion”. In: *ArXiv* abs/1306.0626 (2013).

[9] Christina Lee et al. “Blind Regression via Nearest Neighbors under Latent Variable Models”. In: (May 2017).

[10] Yihua Li et al. “Nearest Neighbors for Matrix Estimation Interpreted as Blind Regression for Latent Variable Model”. In: 2017.

[11] *MovieLens 1M network dataset – KONECT*. Apr. 2017. URL: http://konect.uni-koblenz.de/networks/movielens-1m.

[12] Nagarajan Natarajan and Inderjit S. Dhillon. "Inductive matrix completion for predicting gene–disease associations". In: *Bioinformatics* 30.12 (June 2014), pp. i60–i68. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu269. eprint: https://academic.oup.com/bioinformatics/article-pdf/30/12/i60/1090926/btu269.pdf. URL: https://doi.org/10.1093/bioinformatics/btu269.

[13] Madeleine Udell and S. Boyd. "PCA on a Data Frame". In: 2015.

[14] Miao Xu, Rong Jin, and Zhi-Hua Zhou. "Speedup Matrix Completion with Side Information: Application to Multi-Label Learning". In: NIPS'13 (2013), pp. 2301–2309.

# Appendices

## A Python Source Code

All the code I developed in Python during this project (including the source code for all 4 algorithms) can be accessed here: `https://bitbucket.org/PabloPascualCobo/algorithms_matrix_estimation/`

## B COVID-19 Disruption

Working from home did not have a huge impact on my project, as most of my work was computer-based and thus could be done from any location. The only significant disruption on my project was that I was no longer able to hold face-to-face meetings with my supervisor. Nevertheless, we had our weekly meetings online instead, without this causing any problems.

## C Risk Assessment

As this project was exclusively computer-based, the only possible hazards identified were eye strain and repetitive strain injury due to computer use. This was a fair reflection of the hazards encountered during the project. To avid them, regular breaks were taken after every hour of computer work.