CSC 591 Assigned Project 1: Missing Data Imputation

Author(s)
Chengyuan Liu: cliu32
Rachit Shah : rshah25
Sourabh Sandanshi : ssandan

1 Background And Introduction

In Data Mining tasks, there are large amounts of incomplete, inconsistent, abnormal, and biased items in the original data. These issues will affect the efficiency of data mining execution, and affect the execution result. Therefore, Data Preprocessing is indispensable, where imputation is one of the most critical work to handle the missing value problem instead of simple deletion of records.

There are a number of ways to handle missing data. The simplest way is to delete the records. This introduces bias in our data and also reduces our data. Another way is to substitute them with mean/median. This method doesn't deal with weird/irregular data and doesn't capture the variance of ground truth data. Various other methods have been explored to handle missing data. Some of the common methods include: regression imputation, nearest neighbour imputation, matrix factorization methods, iterative regression imputations (MICE), etc.

We will explore some of these methods on a 3-dimensional patient health records dataset with missing values in this report.

2 Method

In this task, We use Fancyimpute (2) package for data imputation, which is a third-party package for Python that provides implementations of various matrix calculations and imputation algorithms.

2.1 KNN

KNN classified the data by measuring the distance between different eigenvalues. The idea is that if the majority of the samples of the k most similar in the feature space (ie, the nearest neighbor in the feature space) belong to a certain category, then the new sample also belongs to this category, where K is usually not greater than 20. In the KNN algorithm, the selected neighbors are all objects that have been correctly classified. In the categorization decision, the method determines the category according to only the category of the nearest one or several samples.

Nearest neighbor imputations is one of the methods in Fancyimpute(2) which brought an excellent performance on the datasets after our experiment, the idea is to weight samples using the mean squared difference on features for which two rows both have observed data. We thought this method would perform better as it looks for data rows with similar values of other analytes within a patient's records.

2.2 Softimpute

Softimpute is another algorithm in the Fancyimpute(2) package which we figured out to be very useful for some features, the idea of it is to fit a low-rank matrix approximation to a matrix with missing values via nuclear-norm regularization. The algorithm works like EM, filling in the missing values with the current guess, and then solving the optimization problem on the complete matrix using a soft-thresholded SVD. Special sparse-matrix classes available for very large matrices. As 3D-MICE also included a EM - like iteration, we decided to go w=forward with SoftImpute as a possible algorithm

2.3 Datawig

DataWig(3) is a deep learning based algorithm for data imputation made by AWS labs, we tried to use SimpleImputer method iterativly on our case column by column, For most use cases, the SimpleImputer class is

always the best starting point. DataWig expects to provide the column name of the column you would like to impute values for and some column names that contain values that you deem useful for imputation. We used this method it was a simple way to implement a neural network based imputer.

3 Experiment Setup

3.1 Dataset Description

We are provided with a training dataset of 6,000 patients. Each patient's history contains a varying number of records stored in each row, and the values for 13 clinical lab tests are stored in columns. Each record is associated with a time stamp (minutes from arrival/admission) and time interval between two consecutive records are irregular. There are two versions of this dataset. One with ground truth values along with some missing values and the other with masked missing values on top of the already imputed values. We also have a file indicating indices of masked value per analyte and per patient. We used this file to create 13 columns indicating the indices where each analyte is masked. This was used to compute the nRMSD value later. The test consisted on data on 2267 patients following a similar masking and distribution pattern as the training set.

Missing data falls under three categories: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR). Our dataset consisted of examples of all three of them.

3.2 Data Preprocessing and Feature Engineering

In order to find out patterns in the data and determine which features may be useful, We decided to perform Exploratory Data Analysis on the data. This would also help us create features which better fit our problem domain. Below are the plots of Distribution of values for each analyte. While all of them follow some kind of normal distribution, We can see that some distributions like $X9, X11, X12$ and $X13$ are heavily right tailed with an abnormal max value. This makes normalization over the entire dataset an issue as it will be skewed. Thus, we went ahead with normalizing each analyte per patient with the maximum and minimum value of that analyte for that patient.

For Feature Engineering, we wanted to add the temporal aspect of EHR data into our model. The records per patient had irregular differences in time. This means that for a given patient some recordings might have been taken at a small interval different like 5 minutes, while others might have been at an interval of few days or months. To account for this, we decided to add new features based around the time entry. The features added were as follows -

- $time\_diff\_1$ - Difference between consecutive time entries for a patient. It is 0 for first record of a patient.
- $time\_diff\_5$ - Difference between $p_i and p_{i-5}$ time entries for a patient. It is 0 for first 5 records per patient
- $time\_diff\_median$ - Absolute difference of $time$ value of a record from the median value per patient.

All these features were normalized based on their respective maximum and minimum values.

3.3 Model Selection

We explore a variety of models as described in the Methods section. We had a baseline which is based on the 3D MICE method(1) which used a weighted combination of Gaussian Process method and MICE method. We performed model selection akin to hyper parameter tuning where instead of parameters of model we find the best model for a particular feature. We try the following models and/or parameters:

- Iterative DataWig by AWSLabs (Deep Learning)
- With and Without Normalizing
- With and Without Feature Engineering
- Different values of k for KNN imputation model
- SimpleFill (imputing with mean)
- SoftImpute (Matrix Factorization with SVD)
- BiScaler (Low-Rank SVD via Fast Alternating Least Squares)

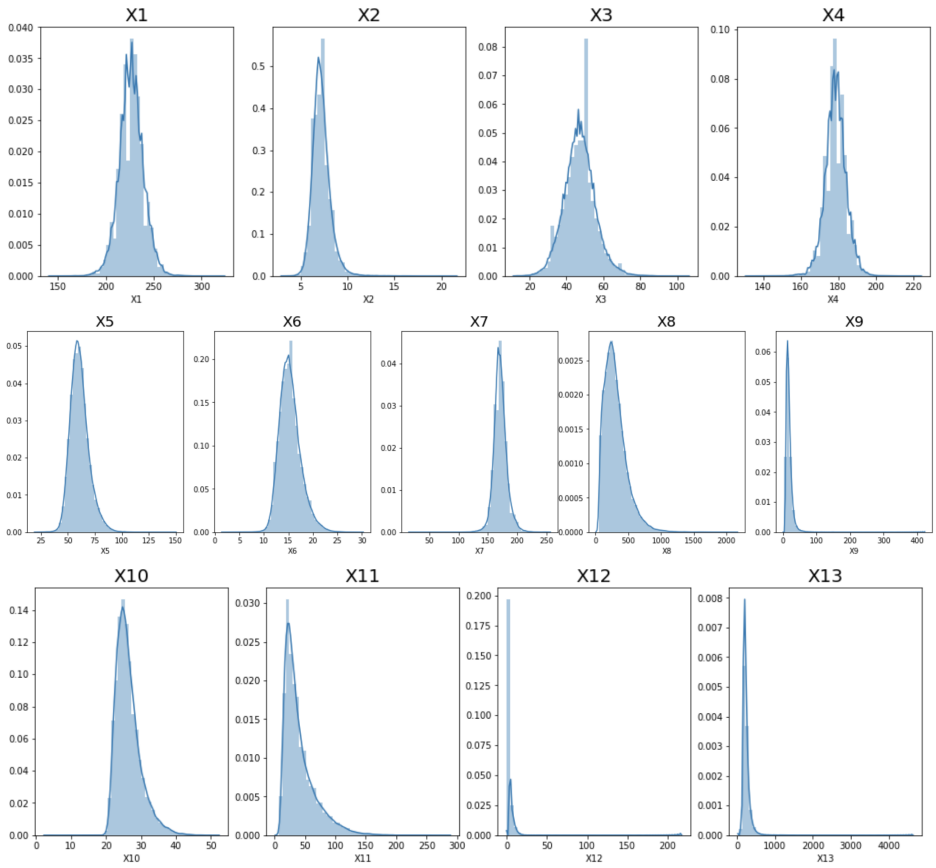

Figure 1: Distribution of values per Analyte

You can see the results of Iterative DataWig in figure 2, the results with and without normalizing for KNN (k=3) in figure 3, all models without feature engineering in figure 4 and all models with feature engineering in figure 5 (Colored with green means we beat 3D-MICE, darker means better, colored with red means we did not beat, darker means worse).

4 Results

Based on our Model Selection we can see that feature engineering and Normalizing helped a lot with the performance of models allowing KNN with a single k to beat 3D mice on training data albeit barely. We can see that the performance of each model is different for different analytes. Analytes 4, 5 and 6 performed better with SoftImpute instead of KNN and Analyte 5 & 6 were the only analytes that performed worse with feature engineering. Hence, we use a combination of models for different analytes where we choose the model that performed best on the particular analyte as our best model. The final model performance on training data along with which model we chose for each analyte is shown in figure 6. You can see that we beat 3D-MICE on 10 analytes but failed to beat it on 3 analytes (Analyte 5,6,11). Our final nRMSD value on training data is 0.2141 which beat 3D-MICE's 0.2246 by a considerable margin.

Our group ranked 5th on the leaderboard for this exercise with a test nRMSD of 0.216491 which beat 3D-MICE test nRMSD of 0.225524.

5 Conclusion

We tried a variety of ways to impute data from simple mean imputation, classic machine learning models and deep neural networks. After reading the 3D-MICE paper which used a complex iterative method of regression we assumed that simple imputation methods like regression won't perform well or won't be able to beat the baseline. Hence, we jumped right to deep learning on our first try using DataWig and an iterative method of

Iterative DataWig														
Iteration	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	Average
1	0.25566585	0.32331697	0.34096575	0.257673547	0.22075012	0.202611258	1.41933543	0.88574967	0.97945776	1.43351929	0.77385521	1.20143702	0.42738602	0.670901838
2	0.25540234	0.33827473	0.33629041	0.258157464	0.210359664	0.199837289	1.4126848	0.91743394	0.93258761	1.40675565	0.8058601	1.23936927	0.44542609	0.673726105
3	0.25692051	0.32833047	0.33791628	0.259728835	0.208588537	0.198936182	1.41250568	0.87679152	1.01066634	1.40316286	0.78407889	1.33896801	0.46521933	0.683216417
4	0.25800777	0.34261039	0.33637884	0.259843304	0.210268054	0.199091369	1.41351293	0.90204647	0.97609817	1.40353293	0.79731927	1.31469859	0.44259496	0.681231005
3D-MICE	0.2024073	0.2621022	0.2369696	0.2177181	0.1467821	0.1455925	0.2713614	0.2310303	0.2592127	0.2482796	0.1897457	0.2324037	0.2770005	0.224661977

Figure 2: Iterative DataWig

KNN (k=3) with and without Normalization														
Method	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	Average
KNN-Non Normalized	0.27960307	0.2902492	0.29353683	0.27469321	0.28376459	0.27518737	0.31097095	0.29540317	0.29331831	0.29457503	0.30634758	0.28596169	0.2939531	0.290581854
KNN Normalized	0.2042565	0.25273571	0.23363042	0.22340957	0.20689583	0.20284526	0.2601069	0.2243889	0.24129703	0.23276591	0.21757831	0.23331296	0.27341403	0.231279794

Figure 3: KNN-3 with and without Normalizing

Model Selection without Feature Engineering														
Method	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	Average
KNN-2	0.21250313	0.26240395	0.240713	0.23113893	0.2111386	0.20719764	0.26958653	0.23119035	0.24800636	0.24100597	0.22263966	0.2410718	0.28368785	0.238637213
KNN-3	0.2042565	0.25273571	0.23363042	0.22340957	0.20689583	0.20284526	0.2601069	0.2243889	0.24129703	0.23276591	0.21757831	0.23331296	0.27341403	0.231279794
KNN-4	0.2043892	0.24851688	0.23183658	0.22075209	0.20671175	0.20402649	0.25654338	0.22583057	0.24035984	0.23183188	0.21636584	0.22994378	0.26832838	0.229648975
KNN-5	0.20576643	0.2458101	0.23211014	0.22097746	0.20902269	0.20710011	0.25628277	0.22824435	0.24002402	0.23291984	0.21768412	0.22988174	0.26531233	0.230087392
KNN-6	0.20770044	0.24487704	0.23338586	0.2220414	0.21181723	0.21007254	0.25780424	0.23101954	0.24191566	0.23600037	0.22025293	0.23111136	0.26396108	0.231689206
KNN-7	0.20985951	0.24464005	0.23502889	0.22365673	0.21446513	0.21310967	0.25863401	0.23450918	0.24427246	0.23859235	0.22327825	0.23310678	0.26349529	0.233588331
KNN-8	0.21293113	0.2448223	0.23673911	0.22551584	0.21748662	0.21661381	0.25977729	0.23798753	0.24581159	0.24103564	0.22610403	0.23523351	0.26292808	0.235614344
KNN-9	0.21548326	0.24506742	0.23871161	0.22742326	0.22046304	0.22005127	0.2607119	0.24147942	0.24778736	0.24381	0.2288465	0.23680984	0.26278387	0.237648364
KNN-10	0.21801852	0.2456336	0.24078003	0.22945262	0.22321914	0.22313835	0.26177635	0.24428397	0.24905897	0.2459415	0.23177702	0.23851588	0.26264218	0.239556778
SoftImpute	0.22726493	0.26361833	0.2815325	0.21704635	0.16875306	0.16975303	0.28269752	0.24706712	0.26172465	0.26665547	0.22794458	0.24385008	0.29000646	0.242147237
SimpleFill	0.29357665	0.27594999	0.30146439	0.29446788	0.28730753	0.29065423	0.30900862	0.32015577	0.30088458	0.31547805	0.31202509	0.30658351	0.28072393	0.299098479
BiScaler + SoftImpute	0.64960086	0.61010349	0.65676997	0.65806543	0.66187955	0.6629553	0.65266778	0.59193415	0.57327272	0.62743747	0.59046025	0.59588009	0.56784553	0.622990198
BEST	0.2042565	0.24464005	0.23183658	0.21704635	0.16875306	0.16975303	0.25628277	0.2243889	0.24002402	0.23183188	0.21636584	0.22988174	0.26264218	0.222900223
3D-MICE	0.2024073	0.2621022	0.2369696	0.2177181	0.1467821	0.1455925	0.2713614	0.2310303	0.2592127	0.2482796	0.1897457	0.2324037	0.2770005	0.224661977
Best Model	3	7	4	Softimpute	Softimpute	Softimpute	5	3	5	4	4	5	10	5

Figure 4: Model Selection Without Feature Engineering

Model Selection with Feature Engineering														
Method	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	Average
KNN-2	0.20345519	0.25419728	0.22767715	0.22528364	0.20898483	0.20981768	0.26261367	0.20518687	0.23667651	0.22394295	0.19656036	0.225076	0.27815704	0.227509934
KNN-3	0.19909607	0.24612142	0.22222734	0.21859777	0.20473196	0.20541945	0.25388921	0.20111355	0.2316412	0.2189211	0.19174865	0.22042613	0.26725335	0.221629786
KNN-4	0.19887236	0.24229195	0.22113225	0.21675056	0.20558248	0.20569569	0.25186633	0.20415173	0.23201286	0.21763875	0.19377472	0.21885231	0.2629146	0.22088743
KNN-5	0.19966129	0.24012453	0.2221406	0.21674871	0.20759789	0.20861776	0.25208685	0.20838632	0.23304223	0.22069028	0.1966304	0.21924852	0.26087032	0.22198813
KNN-6	0.20239134	0.23981178	0.2233003	0.21796715	0.20984929	0.21116453	0.25265957	0.21348294	0.23429598	0.22423182	0.20067199	0.22138985	0.25983946	0.223927385
KNN-7	0.20532246	0.23994346	0.22573996	0.21955497	0.21299146	0.21400219	0.25350796	0.21875373	0.23688165	0.22738839	0.20512665	0.22383642	0.25907297	0.226317097
KNN-8	0.20835204	0.24021376	0.22816289	0.22194649	0.21612526	0.21690923	0.2545822	0.22319782	0.23918161	0.23088208	0.20937371	0.22589131	0.25896412	0.228752502
KNN-9	0.21110231	0.24072148	0.23080876	0.22453669	0.21928312	0.22059308	0.2561755	0.22772117	0.24161832	0.23379861	0.21354219	0.22845663	0.25919338	0.231350096
KNN-10	0.21365605	0.24164862	0.23268133	0.22662986	0.22207832	0.22322435	0.2577937	0.23136537	0.24353092	0.23650207	0.21707331	0.23101776	0.25900076	0.233554031
SoftImpute	0.21403574	0.25703156	0.25324471	0.2131297	0.17624691	0.17789686	0.27229169	0.21791626	0.25213999	0.24568428	0.20880853	0.23237349	0.28369113	0.231114681
BiScaler + SoftImpute	0.24327662	0.26120431	0.27286944	0.23093787	0.18391183	0.18169378	0.27430115	0.23517756	0.26889424	0.26729545	0.23222242	0.2470515	0.29040868	0.245326527
BEST	0.19887236	0.23981178	0.22113225	0.2131297	0.17624691	0.17789686	0.25186633	0.20111355	0.2316412	0.21763875	0.19174865	0.21885231	0.25896412	0.215301136
3D-MICE	0.2024073	0.2621022	0.2369696	0.2177181	0.1467821	0.1455925	0.2713614	0.2310303	0.2592127	0.2482796	0.1897457	0.2324037	0.2770005	0.224661977
Best Model	4	6	4	softimpute	softimputeB	softimputeB	4	3	3	4	3	4	8	

Figure 5: Model Selection With Feature Engineering

Final Model														
-	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	Average
Model	KNN-4	KNN-6	KNN-4	SoftImputeFE	SoftImputeNoFE	SoftImputeNoFE	KNN-4	KNN-3	KNN-3	KNN-4	KNN-3	KNN-4	KNN-8	
FINAL NRMSD	0.19887	0.23981	0.22113	0.213129696	0.168753059	0.169753032	0.25187	0.20111	0.23164	0.21764	0.19175	0.21885	0.25896	0.2140982
3D-MICE	0.20241	0.2621	0.23697	0.2177181	0.1467821	0.1455925	0.27136	0.23103	0.25921	0.24828	0.18975	0.2324	0.277	0.224662

Figure 6: Final Model Results

optimizing imputations similar to MICE using DataWig. This didn't perform well and we tried simpler methods like mean imputation and KNN imputation which performed surprisingly well and even beat 3D-MICE with some feature engineering help later on. This gave us a lesson of starting with simpler methods first before exploring complex algorithms (follow Occam's Razor).

You can find our code at https://github.ncsu.edu/rshah25/MLAP1.

References

[1] Luo, Yuan, et al. "3D-MICE: integration of cross-sectional and longitudinal imputation for multi-analyte longitudinal clinical data." Journal of the American Medical Informatics Association 25.6 (2017): 645-653.

[2] S. Feldman, "iskandr/fancyimpute", GitHub, 2019. [Online]. Available: https://github.com/iskandr/fancyimpute. [Accessed: 28- Sep- 2019].

[3] "awslabs/datawig", GitHub, 2019. [Online]. Available: https://github.com/awslabs/datawig. [Accessed: 28- Sep- 2019].