Imputation and Supervised Learning on Sparse Datasets

Amir Karamlou, Anna Bair, and Ayush Sharma
Massachusetts Institute of Technology
Cambridge, MA
{annabair, ayushs, karamlou}@mit.edu

Abstract

In this work we implement, evaluate and compare several methods for dealing with sparsity and missing values in data sets. We analyze the performance of each method using the supervised training methods Support Vector Machine (SVM) and Logistic Regression (LR) for the task of binary classification. We utilize the publicly available NHANES dataset for this project, and predict smoking behavior among the respondents via our methods. We discovered that the MICE with mean imputation method gives the best performance result (89.47% *accuracy) when trained with SVM.*

1. Introduction

Real world data is often much more messy, incomplete, and complex than the standardized datasets we use in machine learning coursework. In this project, we explore the problem of a sparse real world dataset in the context of performing supervised learning for prediction. We attempt to predict whether or not a person smokes based on their responses to other health-related questions. We use several different methods of data imputation to handle our sparse dataset. Then, we evaluate the performance of our imputed dataset by using common prediction models. Ultimately, we are able to achieve almost 90% accuracy on our prediction task using imputed data.

1.1. Dataset, Problem, and Objectives

The NHANES dataset [2], is an anonymized medical dataset that includes interviews with demographic, socioeconomic, dietary, and health-related questions. As with all such optional surveys, the resulting dataset contains many missing values.

Our supervised learning task for this project was to predict smoking behavior among the survey respondents given their answers to other nutrition and health related questions.

Since our dataset was imperfect, and included issues such as missing response values, we tackled it via experimenting with different techniques for data imputation as well as collaborative filtering.

We chose to use Python for this project, since it has useful packages such as *numpy*, *sklearn* [4], *scipy*, and *matplotlib*. In addition to the standard packages, we used the package *fancyimpute* for our KNN and MICE imputation methods, and we implemented our own Collaborative Filtering Algorithm.

2. Data Pre-processing Methods

2.1. Data Formatting

Our dataset comes from a CSV file which is part of the 2013-2014 NHANES (National Health and Nutrition Examination Survey) posted on Kaggle, a popular dataset and data science website. This survey data is comprised of six main files: demographic, diet, examination, labs, medications, and questionnaire. We chose the questionnaire dataset because it

included a wide range of questions covering lots of different types of information. The original dataset contains 953 features, including information about smoking, medical conditions, mental health screens, and cardiovascular health, and 10176 rows, each representing a participant.

Our pre-processing was tailored to fit our central goal: predicting whether or not someone is a smoker. We created a target vector which reflects participants' responses to the question "Do you now smoke cigarettes?" and we normalized the responses. Then, we removed all other columns associated with smoking behavior from our data matrix. This was to ensure that our prediction is based on other features than simply different encodings of the participant's smoking behavior.

This main data matrix and target vector were modified and used in subsequent steps in a variety of ways to work with our different methods.

2.2. Sparsity

One parameter we varied throughout much of our experimentation was the density cutoff for the data. As shown in Figure 1, our dataset is far from complete.

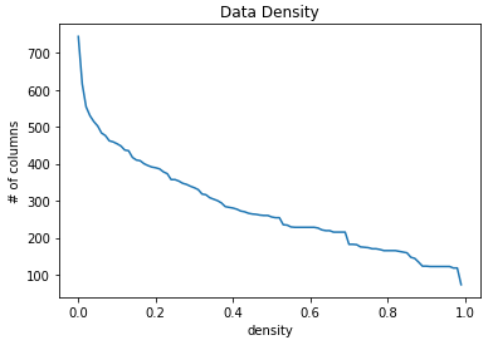

Figure 1: Proportion of the dataset with columns at or above given density values.

Our dataset is survey results from a standard questionnaire and we found that many of the questions are so specific that most participants would not have answers for them. For instance, one of the questions is "How old were you when bladder cancer was first diagnosed?" Out of 10176 survey participants, only 11 had a response to this bladder cancer question. Only 59 of the 953 columns contained a value for each respondent. Due to the sparsity of our data, we experimented with a few different methods of data imputation in order to be able to complete our original prediction task.

One of the most simple methods to deal with missing data is simply to remove the most sparse columns. Thus, before applying imputation or other methods, we would choose a sparsity level at which to limit our dataset. We chose to perform this cutoff because our ultimate goal is not to impute the entire dataset, but rather to perform a prediction. Imputation is a step in our process to perform better prediction. As well, given that some features of our dataset had remarkably few features (some only had one response), we doubted that those columns would add much to our prediction.

2.3. Evaluation

Data imputation, or the filling in of missing data values, became a central part of this project. Imputation is still an active area of research and thus filling in the missing values was a non-trivial exercise. We explored different methods of imputation and collaborative filtering, as well as some combinations and extensions in order to achieve the best results.

We used two broad categories of imputation methods: naive imputation and advanced imputation. Our naive imputation methods include zero fill, mean fill, median fill, mode fill, and KNN, all of which are described in following sections. Zero fill was trivial to implement by hand, and we used *sklearn Imputer* and *fancyimpute*'s `KNN` method for the remaining naive methods. For `KNN`, we varied the value of k from among $\{1, 5, 17, 21, 31, 33, 37\}$.

The advanced imputation methods we used were Multiple Imputation by Chained Equations (MICE) and Collaborative Filtering. We used *fancyimpute*'s *MICE* method and we implemented our own collaborative filtering. For the MICE method, we experimented with mean fill, median fill, and random fill.

For all methods except zero fill, we

tested varying density cutoffs from among $\{0.5, 0.7, 0.9\}$.

For data imputation, we evaluated the performance when running basic Support Vector Machine (SVM) and Logistic Regression (LR) implementations. For data imputation tasks in which new imputation methods are being tested, it is common to artificially remove elements, perform imputation on the now sparse dataset, and then evaluate how well the calculated imputed values match the actual values. However, since we have no ground truth data matrix containing all the values and since we are using off the shelf methods which have previously been tested, we determined that testing most of our imputation methods on artificially created datasets was outside the scope of this project.

For collaborative filtering, since we implemented the methods from scratch, we chose to test our prediction compared to ground truth values. We used our algorithm to predict a given participant's smoking patterns based on similar participants' smoking patterns. We reduced our dataset to only include the rows which contained a value for our target column (whether or not a participant is a smoker). Then, we attempted to predict the values of our target column. We evaluated our performance based on the prediction accuracy rate (whether our predicted value matched the actual value).

2.4. Naive Imputation Methods

2.4.1 Zero Fill

To begin, our naive imputation method was to fill in missing values with zeros. Although a simple method, this is a typical baseline approach used for imputation. As can be seen from our results in Table 1, we were able to achieve decently good predictions just with this simple method.

2.4.2 Mean, Median and Most Frequent Value Fill

Next, we used *sklearn Imputer* in order to fill in our data with either the mean, median, or most frequent value by column. These results are summarized in Table 2.

Method	Score	Best C	Best penalty
SVM (L2)	0.510		
SVM (L1)	0.765		
GridSearchCV SVM	0.835	0.01	L1
Default LR	0.694		
GridSearchCV LR	0.827	0.1	L1

Table 1: Prediction results on SVM and LR using naive zero fill imputation

Method	Density	SVM	LR
Mean	0.5	0.7310	0.7154
	0.7	0.7173	0.6686
	0.9	0.7014	0.7054
Median	0.5	0.8889	0.6667
	0.7	0.8889	0.7193
	0.9	0.7234	0.6933
Mode	0.5	0.8889	0.6940
	0.7	0.8889	0.7193
	0.9	0.7234	0.6894

Table 2: *sklearn Imputer* prediction performance using SVM and LR. The Method column refers to the reference method used to impute a given missing value. For instance, 'Mean' fills in missing values with the mean of all present values in its column.

2.4.3 K Nearest Neighbors

K Nearest Neighbors is another common, simple imputation method. It uses a weighted sum of the nearest neighbor entries (rows) for a given piece of missing data and imputes given these multiple observations. We chose to use the standard $k = 5$ as well as a few larger values until our prediction accuracy seemed to stop increasing. The test results are shown in Table 3.

In addition, Figure 2 summarizes our results from the above table. It samples 3 values for $k = \{5, 17, 37\}$ and compares them for both SVM and LR method. It is noteworthy that SVM peforms consistently better for our

k	Density	SVM	LR
5	0.5	0.8713	0.6939
	0.7	0.8811	0.6920
	0.9	0.7275	0.6793
11	0.5	0.8733	0.6998
	0.7	0.8811	0.6959
	0.9	0.7255	0.6813
17	0.5	0.8811	0.7076
	0.7	0.8772	0.7192
	0.9	0.7275	0.6813
31	0.5	0.8830	0.6491
	0.7	0.8831	0.7037
	0.9	0.7275	0.6934
33	0.5	0.8869	0.6998
	0.7	0.8850	0.5828
	0.9	0.7275	0.6934
37	0.	0.8850	0.6979
	0.7	0.8830	0.6452
	0.9	0.7275	0.6974

Table 3: K nearest neighbors imputation and subsequent performance on prediction task using SVM and LR

dataset. This could be attributed to the fact that SVM method is robust against bounded noise, and our dataset can be expected to behave in that fashion.

2.5. Advanced Imputation Methods

2.5.1 Multiple Imputation by Chained Equations

After these basic imputation methods, we explored more sophisticated methods. One such method is Multiple Imputation by Chained Equations (MICE) [1]. Single imputation involves filling in a given piece of missing data by referencing one other value. Multiple imputation is more robust since it computes several imputed values for each piece of missing data and then consolidates these results with their mean and variance. A key assumption used in MICE is the Missing At Random (MAR) assumption. This assumes that the missing data depends only on the available information, not on implicit features. This is not the same as the Missing Completely At Random (MCAR) assumption. MCAR is stricter than MAR because it requires that the gaps in data are completely random. MAR only requires that the missing data be correlated to features present in the data rather than latent features. It is not possible to prove that MAR holds for a given dataset; rather it is common to show that it is unlikely that MAR does not hold [6].

Assuming that MAR holds for this dataset, we used the python package *fancyimpute* to impute our dataset using MICE. We tested imputation by mean, median, and random values. As one method of evaluation, we report the results of using our imputed data to predict a given person's smoking habits using SVM and LR. The results are summarized in Table 4.

Method	Density	SVM	LR
Mean	0.5	0.8655	0.7524
	0.7	0.8947	0.7310
	0.9	0.7255	0.6794
Median	0.5	0.8791	0.6940
	0.7	0.8850	0.7037
	0.9	0.7234	0.6914
Random	0.5	0.8596	0.7173
	0.7	0.8830	0.6238
	0.9	0.7234	0.7154

Table 4: Prediction performance of Multiple Imputation by Chained Equations (MICE) using SVM and LR. The Method column refers to the reference method used to impute a given missing value.

2.6. Collaborative Filtering

In addition to the methods mentioned above, we also implemented the Collaborative Filtering (CF) method for classifying our data [5]. CF is a popular recommendation algorithm that bases its predictions for a given row on other rows that are most similar to the one of interest. In our implementation of CF we considered two approaches in predicting our target variable: we either predict the target value from the data point with the single smallest distance or use a k-nearest-neighbors (KNN) approach. In the KNN approach, we predict the target value

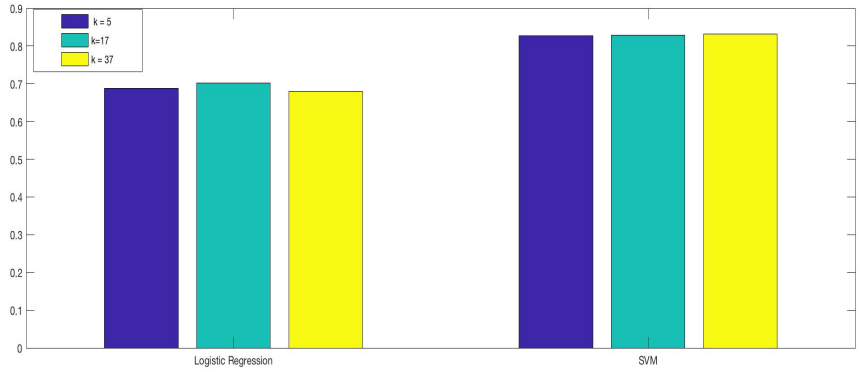

Figure 2: LR vs SVM model performance

as the mode of the smoking values of the k closest neighbors.

In the CF approach we use the training data set as the set of the known data points, and test the accuracy of our training method using the test set. We investigated two different measures for determining the distance between two points: the cosine distance and the Jaccard distance. The Jaccard distance is more robust for cases with more missing data, so we were not surprised to find that CF using the Jaccard Index had higher accuracy rates than CF using the Cosine Distance. Our results are summarized in Table 5.

k	Jaccard Accuracy	Cosine Accuracy
1	0.6705	0.5048
17	0.7777	0.5730
21	0.7835	0.5730
31	0.8012	0.5672
33	0.7953	0.5730
37	0.7895	0.5458

Table 5: Accuracy of our Jaccard Index KNN Collaborative Filtering Method on predicting a given user's smoking habits.

2.6.1 Cosine Distance

The cosine distance can be defined as:

$$d_{cos}(u, v) = 1 - \frac{u \cdot v}{|u||v|}$$

where $(\cdot)$ is the inner product between two vectors and $|x|$ denotes the norm of vector x. In our implementation, we only consider the elements of our vectors if the value at that index is present in both vectors. We discard the indices of missing values from the calculation of the inner product and vector norms.

Note that if two states completely overlap, we get $d_{cos} = 0$ and if they are completely orthogonal we get $d_{cos} = 1$ which makes d_{cos} a valid measure for the distance.

2.6.2 Jaccard Distance

The Jaccard index is a commonly used statistical indicator for measuring the pairwise similarity and can easily be extended for incomplete data [3]. We define the Jaccard distance as follows:

$$d_{\text{Jaccard}}(u, v) = 1 - \frac{|u \cap v|}{|u \cup v|} \qquad (1)$$

where $|u \cap v|$ is the number of non-missing elements that u and v have in common (the intersection), and $|u \cup v|$ is the total number of distinguishable elements that the data points share at every index (the union). Similar to the cosine distance, $d_{\text{Jaccard}} = 0$ if the two data points contain values for all the same features, and $d_{\text{Jaccard}} = 1$ if they have no overlap.

2.7. Principal Component Analysis (PCA)

PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables to a set of linearly independent variables called the principal components. This transformation is a common form of dimensionality reduction and is defined in such a way that the first principal component accounts for the largest proportion of variance in the dataset, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. Table 6 shows the explained variance of our top 5 components.

In this work we use an off-the-shelf implementation of PCA from the *sklearn* library which implements linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

Component	Explained Variance
1	3.52165322e+
2	1.51320352e+10
3	4.56051269e+09
4	3.37699782e+09
5	2.88801663e+08

Table 6: PCA: explained variance of top 5 components

3. Training Methods

In order to achieve our goal of predicting an individual's smoking habits, we used two common supervised learning methods: Support Vector Machines (SVM) and Logistic Regression (LR). For both of these models we used the *sklearn* implementations for training, and *sklearn*'s *GridSearchCV* method for parameter tuning. We used a standard 80/20 split to separate out test data, and then once test data was removed, another 80/20 split to get training and validation datasets. Thus, the test set is 20% of the total dataset, the validation set is 16%, and the training set is 64%.

3.1. Support Vector Machine (SVM)

Support Vector Machines, or SVMs are commonly used linear classfiers and tend to outperform naive perceptron-like classifiers. The key idea in an SVM is to find a maximum margin hyperplane for a set of datapoints in high dimensional space. This involves solving the following constrained optimization problem:

$$min \frac{1}{2} W^T W + C \sum_{i=1}^{n} \zeta_i$$

subject to the following constraints:

$$y_i(W^T \phi(x_i) + b) \geq 1 - \zeta^i$$

SVMs provide a couple of advantages in practice, including

- Being effective in high dimensional spaces
- Being generalizable to non-linearities via the kernel trick

For our project, we used *sklearn* package for performing SVM prediction.

3.2. Logistic Regression (LR)

Logistic regression - a bit of misnomer, is actually a model for linear classification instead. The logistic function ($\frac{e^t}{1+e^t}$) used in this method provides some nice properties to work with.

One indispensable property of logistic function is that it maps $\mathbb{R} \Rightarrow (0, 1)$. In doing so, it maps all reals to finite bounded space of $(0, 1)$ after which the output may be interpreted and treated as probabilities.

When formulated with an L2 regularization, the cost function that we optimize that the model optimizes for is the following:

$$\frac{1}{2} W^T W + C \sum \log(exp(-y(X^T W + c)) + 1)$$

and the L1 regularized version, similarly would be:

$$||W||_1 + C \sum \log(exp(-y(X^T W + c)) + 1)$$

The above version uses the L1 norm for thresholding the parameter space.

For our project, we used *sklearn* package for performing logistic regression prediction.

4. Discussion

4.1. Analysis of Results

To begin, there are a few key observations to note from our results:

1. SVM consistently outperforms LR.
2. Of the naive imputation approaches, Median and Mode Fill at 70% and 50% density all performed the best, and equally well, each with an accuracy of $.8889$.
3. Of all the imputation methods, MICE with Median Fill at 50% density performs the best, with an accuracy of $.8947$.
4. For Collaborative Filtering, the implementation using Jaccard distance consistently outperforms the one using cosine distance.
5. While most methods perform similarly at 70% and 50% density, they perform worse at 90% density.

We saw a general trend that limiting our dataset to rows with only 90% performed worse than using a sparsity cutoff of 50% or 70%. This is due to the drastically decreased size of our dataset when we only use columns with 90% of values present. The size of our dataset at different sparsity cutoffs is shown in Table 7.

Density	Rows	Cols	Total Vals
0.5	1,643	256	420,608
0.7	2,563	182	466,466
0.9	2,497	123	307,131

Table 7: Number of rows, columns, and total values (after imputation) present in our dataset at each sparsity cutoff.

We can see that, when limited to a 90% cutoff, we only have around $\frac{2}{3}$ as many data points as we do when we use a 70% cutoff. Although 90% density gives us a denser dataset, if we are using good imputation methods, it makes sense that a cutoff which provides us with more data points is more valuable for prediction methods.

We were pleased to see that our advanced imputation methods (MICE in particular) outperformed the naive imputation methods. As well, our modification to Collaborative Filtering by using the Jaccard distance outperformed our Collaborative Filtering using the standard cosine distance.

Regarding the fourth point, we expected the Jaccard distance metric to perform better than cosine distance on our dataset. Although we adapted cosine distance to only factor in features present in both vectors, Jaccard is better designed to work with sparse datasets. It is based on number of elements present rather than the values themselves and thus accounts for similarity between vectors with the same features present.

4.2. Overfitting

As in any machine learning project, we must assess how much our implementation is overfitting to the training data.

In order to counteract overfitting, we applied one of two types of regularization (L1 or L2) when performing our predictions using SVM and LR. We used a validation set to tune our parameters, including selecting the best type of regularization, and then performed our final assessment on a test set. We found that L1 regularization, or LASSO, consistently performed the best for both SVM and LR methods. This makes sense given our sparse data, since an attractive feature of L1 is that it not only performs regularization by driving the weights down, but it also performs feature selection by driving low weights to zero. This is useful for our dataset because many of the features will actually not be useful for predicting a user's smoking habits, mostly due to the incredibly low density of many of the columns. By not factoring in the contributions of low weight features to our prediction task, L1 regularization helps our methods to not overfit.

Regarding imputation, we are less likely to overfit when we use multiple samples to impute a given piece of missing data. For instance, MICE imputes the same dataset several times and then calculates the mean and variance for

stable predictions of the missing data. By accounting for the uncertainty in imputations, this method inherently protects against overfitting. As well, KNN achieves a similar effect by using k nearest neighbor rows to impute each missing value, rather than only one other value.

In our Collaborative Filtering task, prediction using a single row with the closest cosine distance is prone to overfitting, so we would prefer to use our KNN version of CF with a larger value of k that performs well on a validation set for generalizability.

4.3. Future Work

One key assumption we used in this paper was that data was Missing At Random (MAR). Since it is not possible to directly prove that data is missing at random, we chose to not include the proof in our analysis, and rather cautiously hold this assumption so that we could perform our analysis. We are aware that there often are other latent factors that contribute to the patterns of missing data, but we determined it to be outside the scope of our project.

As well, the paper we referenced when implementing the Jaccard Index for Collaborative Filtering included an outline of an algorithm which handles sparse data well by iteratively projecting the data space onto subspaces of a "feasible region". The algorithm guarantees that the projections are close positive semidefinite approximations of the original sparse matrix. The paper shows that successive projections onto the feasible region will result in an improved estimate of the (unknown) true Jaccard Index matrix. Although we did not have time to implement this algorithm, we imagine that it would yield better results than our Jaccard Index implementation of CF, given that Jaccard already outperformed the standard cosine distance version of CF.

5. Contributions

In this paper, we perform a review of current state of the art imputation methods as used for a prediction task. We demonstrate some naive methods as a baseline performance and then show performance on both packages that implement more advanced methods as well as our own implementation of a few varieties of Collaborative Filtering. In general, the advanced methods outperformed the naive methods and we achieved a prediction accuracy rate of 89.47%.

6. Division of Labor

Anna researched and implemented the naive and advanced imputation methods and wrote the Analysis section. Anna and Amir both worked on data pre-processing and collaborative filtering. Amir implemented collaborative filtering. Amir and Ayush worked on PCA. Ayush implemented the *sklearn* SVM and LR. Each member wrote the portion of the paper which they implemented.

References

[1] M. J. e. a. Azur. Multiple imputation by chained equations: What is it and how does it work? International journal of methods in psychiatric research 20.1 (2011): 4049., "2017".

[2] CDC. National health and nutrition examination survey. `https://wwwn.cdc.gov/nchs/nhanes/ContinuousNhanes/Default.aspx?BeginYear=2015`.

[3] W. Li. Estimating jaccard index with missing observations: A matrix calibration approach. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2620–2628. Curran Associates, Inc., 2015.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[6] Wikipedia. Imputation (statistics) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Imputation%20(statistics)&oldid=813636839`, 2017. [Online; accessed 11-December-2017].