Lecture 22: MATH 342W: Introduction to Data Science and Machine Learning

Sergio E. Garcia Tapia*

May 1st, 2025 (last updated May 2, 2025)

1 Missingness

Last time, we began discussing missingness. Recall this means there is missingness in X, not in y's. If y's are missing, then the unit would not be in $\mathbb{D}$. None of the algorithms that we have seen can handle missing data.

1.1 Listwise Deletion

The most naive (simple) approach for handling missingness is **listwise deletion**, which involves dropping all rows that have missingness. This may be alright if a "trivial" amount of data is missing. However, this there is "a lot" of missingness, listwise deletion can introduce errors in our predictive model.

The first type of error is estimation error because there is less data, which an also make us more likely to overfit. But that is not all; if there is a pattern to the missing data, then the predictive model is prone to poor extrapolation, likely due ot **sampling bias**. For example, one of our features may be age, and it could be that the units that are missing correspond to senior-aged individuals (see Figure 1). To know what we should do next, we need to discuss what is known as **missing data mechanisms**.

*Based on lectures of Dr. Adam Kapelner at Queens College. See also the course GitHub page.

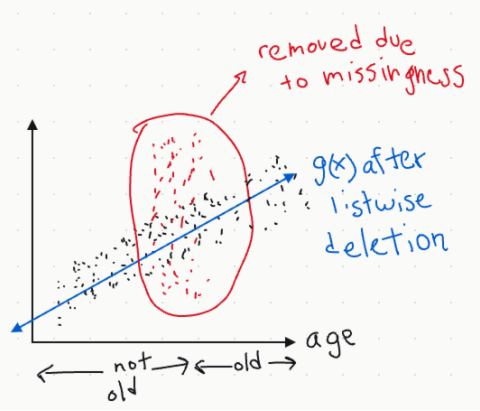


Figure 1: Computing predictive function g on $\mathbb{D}$ after performing listwise deletion.

1.2 Missing Data Mechanisms (MDMs)

Missing data mechanisms are devised in accordance to the question "why is x_{ij} missing?". We will discuss the following possibilities:

(1) Data is missing completely at random (MCAR).

(2) Data is missing at random (MAR).

(3) Data is not missing at random (NMAR).

1.2.1 Missing Completely At Random (MCAR)

Let M_{ij} be the Bernoulli random variable which equals 1 if x_{ij} is missing, meaning

$$P(M_{ij} = 1 \mid \gamma)$$

where γ is independent of all X and $\boldsymbol{y}$ (that is, $M \sim \text{Bernoulli}(\gamma)$).

Example. MCAR may be the result of data corruption (think of a solid state drive subjected to magnetism). This is rare in practice due to modernized data storage solutions. Nevertheless, we need a way to deal with the possibility.

The solution is to **impute** all x_{ij} that are missing. Let $\mathcal{M}$ be the set of feature indices that contain missingness in some unit. We *predict* values the missing x_{ij} by viewing $\boldsymbol{x}_{\cdot,j}$, the jth column (which has all n values for the jth feature) as the response; we use $\boldsymbol{x}_{\cdot,-j}$ (all other features besides the jth feature) and $\boldsymbol{y}$ as the inputs. This yields an **imputed design matrix**

$$X_{\text{imp}} = \text{Imputed}(X, \boldsymbol{y})$$

Then we apply our methodology as usual to compute a prediction function:

$$g = \mathcal{A}(\langle X_{\text{imp}}, \boldsymbol{y} \rangle, \mathcal{H})$$

Why does this work? We are assuming that there is a functional relationship between X and $\boldsymbol{y}$, which suggests that there is probably some way to recover X from $\boldsymbol{y}$.

1.2.2 Missing At Random (MAR)

This means that the probability that x_{ij} is missing is dependent not only on γ (as in MCAR), but also on all other values for the measurements on unit i besides j that are available, as well as values of the measurements on unit i that are missing:

$$P\left(M_{ij} \mid \gamma, \boldsymbol{x}_{i,-j}, \boldsymbol{x}_{i,-j_{\text{missing}}}\right)$$

For example, suppose $i = 3$, $p_{\text{raw}} = 6$, and that a certain unit is

$$\boldsymbol{x}_i = \begin{bmatrix} 17.1 & 15.6 & \texttt{NA} & \text{“Blue”} & \texttt{NA} & 7 \end{bmatrix}$$

That is, the missingness of one of the features influences the missingness of another. In MAR, we also impute.

Example. Suppose there is a survey, and a person does not answer a question. Perhaps they do not answer because the question does not apply to them, or they choose not to for personal reasons, or maybe the person is old and struggles to read a question or notice it is there to begin with.

1.2.3 Not Missing At Random (NMAR)

In this case, the probability that data is missing is given by

$$P\left(M_{ij} \mid \gamma, \boldsymbol{x}_{\cdot,-j}, \boldsymbol{x}_{\cdot,-j_{\text{missing}}}, x_{ij}, \vec{U}\right)$$

This is the most general mechanism. Notice the probability depends on x_{ij} too, the missing value itself. Here, $\vec{U}$ are the features that are unknown.

Example. Suppose you are the ith candidate filling out a survey, and on the jth question asks whether you spent time in jail, which you decide not to answer.

In this case, the data is unimputable, but we impute anyway. When imputing, it is recommended to create binary variables indicating missingness for all indices $\mathcal{M}$.

Example. Suppose $\mathcal{M} = \{3, 7, 11\}$, meaning features indexed 3, 7, and 11 have missingness in some units. Then the matrix M might look like

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 1 & 0 \end{bmatrix}$$

where the first column corresponds to the 3rd feature, the second to the 7th feature, and the third to th 11th feature.

Entry M_{ij} is 1 if x_{ij} is missing, and is zero otherwise, We treat the columns of M as features and augment matrix X to $[X \mid M]$. Then we impute on that:

$$[X_{\text{imp}} \mid M] = \text{Impute}([X \mid M], \boldsymbol{y})$$

Finally, we compute our predictive model using this augmented matrix:

$$g = \mathcal{A}(\langle [X_{\text{mp}}, M], \boldsymbol{y} \rangle, \mathcal{H})$$

The idea is that knowing a feature is missing may carry more information than the value of the feature itself had it been present.

Example. For example, if we are collecting data regarding public school shootings in Chicago and a student's GPA is missing, that may be more telling than the fact that their GPA is low.

In the real world, assume that you should use NMAR.

1.3 Imputing

How do we actually impute? An imputation really means a prediction, but the latter carries the implication that we are predicting on y. Ultimately, we impute so that we can predict y. If $\boldsymbol{x}_{\cdot,j}$ is continuous, binary, or multinomial, then we can use any algorithm suitable for continuous, binary, or multinomial responses, respectively. Note that random forests can be used for all feature types[1] and it is highly accurate.

We can apply the following algorithm to perform imputation.

Algorithm 1 (Miss Forest (2012) Imputation Algorithm)**.** Let $\mathcal{M}$ be the set of feature indices for which a value is missing in some unit. Let M be the binary matrix whose entries indicate missingness for the features encoded by $\mathcal{M}$.

(0) Initialize the missing values for $\boldsymbol{x}_{\cdot,j}$.

- If $\boldsymbol{x}_{\cdot,j}$ is continuous, use Average$[\boldsymbol{x}_{\cdot,j}]$;
- If $\boldsymbol{x}_{\cdot,j}$ is categorical, use Mode$[\boldsymbol{x}_{\cdot,j}]$.

(1) For all $j \in \mathcal{M}$:

- Let I_j be the row indices for which feature j does not have missingness. Then let $\boldsymbol{x}_{I_j,j}$ be the vector of entries of $\boldsymbol{x}_{\cdot,j}$ for which no values of the jth feature are missing. Fit

$$\boldsymbol{x}_{I_j,j} \sim [X_{I_j,-j} \mid M_{I_j,\cdot} \mid \boldsymbol{y}_{I_j,\cdot}]$$

  using random forests. That is, treat $\boldsymbol{x}_{I_j,j}$ as a response, and created an augmented matrix by concatenating the columns of $X_{I_j,-j}$, $M_{I_j,\cdot}$, and $\boldsymbol{y}_{I_j,\cdot}$ (The notation above is similar to the R notation for `lm(y ~ X)`).
- Repeat for all features.
- Predict x_{ij} for missing values.

(2) Repeat step 1 until

$$\|\boldsymbol{x}_t - \boldsymbol{x}_{t-1}\| > \|\boldsymbol{x}_{t-1} - \boldsymbol{x}_{t-2}\|$$

(that is, continue with imputaton until it cannot do better) or $T = 10$ iterations (a default). Let $\boldsymbol{x}_t$ be a vector of all imputed at iteration t.

What if you want a test set? We need a test set to compute honest metrics so that we can gauge performance. The reason we have honest metrics is that you have a $\boldsymbol{y}_{\text{test}}$, which we can compare it to $\hat{\boldsymbol{y}}_{\text{test}}$, which is like predicting the future.

Suppose we shuffle our data set and cut off a part of it to use as X_{train} and $\boldsymbol{y}_{\text{train}}$, and another to use as X_{test} and $\boldsymbol{y}_{\text{test}}$:

$$\begin{bmatrix} & * & \\ & X_{\text{train}} & \\ * & & \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{y}_{\text{train}} \\ \\ \end{bmatrix}$$

[1] Not exactly true, due to survivor features and counts, as you can find out by taking MATH 343. However, we will assume it always works here.

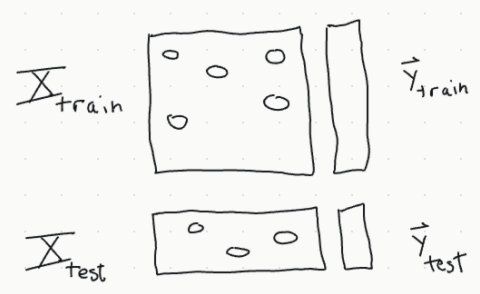


Figure 2: The usual split of $\mathbb{D}$ into train and test sets, but now we must consider missingness.

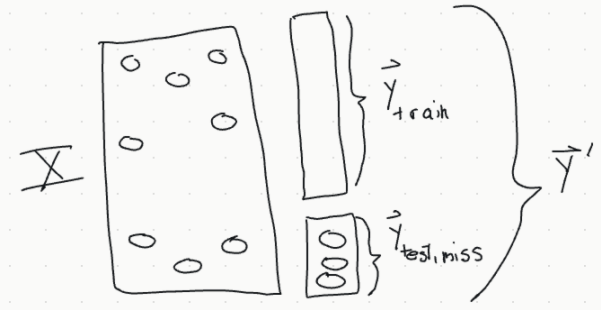


Figure 3: X concatenates the rows of X_{train} and X_{test}, and $\boldsymbol{y}'$ concatenates $\boldsymbol{y}_{\text{train}}$ with a vector of missing values whose length is that of $\boldsymbol{y}_{\text{test}}$.

$$\begin{bmatrix} * \\ & X_{\text{test}} \\ & * \end{bmatrix}$$

However, if there is missingness in X_{test}, what do we do? One thought could be to impute on both, but this can get us into trouble because the imputation process involves looking at $\boldsymbol{y}_{\text{test}}$ to use as a feature. While we can open X_{test}, we cannot open $\boldsymbol{y}_{\text{test}}$, since it has the values that we are trying to predict (see Figure 2). To get around this, after splitting, use the whole X (that is X_{train} and X_{test}), and create a $\boldsymbol{y}'$ where we keep $\boldsymbol{y}_{\text{train}}$ unchanged by make the $\boldsymbol{y}_{\text{test}}$ indices all missing (see Figure 3). Then, we will impute on this:

$$X_{\text{imp}} = \text{MissForest}(X, \boldsymbol{y}')$$

Note that we do not care what it imputes to for the indices of $\boldsymbol{y}_{\text{test}}$; we just want to use it to fill in the missing entries in X without relying on the actual values of $\boldsymbol{y}_{\text{test}}$. Indeed, we need to impute both X_{train} and X_{test}, but we do it in such a way that the imputation cannot see the future.

This concludes our discussion for missingness, allowing us to apply our algorithms.

2 R Demo

See `QC_MATH_342W_Spring_2025/practice_lectures/lec22.Rmd` (click here).

3 Asymmetric Cost Modeling for Binary Classification

In our earlier lectures when we introduced binary classification, we did not make a distinction between the kind of misclassifications that could occur:

$$\hat{y} = 0, \text{ but } y = 1 \quad (\textbf{False Negative (FN)})$$
$$\hat{y} = 1, \text{ but } y = 0 \quad (\textbf{False Positive (FP)})$$

That is, we initially only cared about counting errors, as in our originally definition of misclassification error:

$$\text{Misclassification Error} := \sum_{i=1}^{n} \mathbb{I}_{y_i \neq \hat{y}_i}$$

Suppose we denote C_{FP} to mean the cost of a false positive, and C_{FN} to be the cost of a false negative. In the setting above, $C_{FN} = C_{FP}$, so that misclassification error is a suitable function to minimize, but this may not always be the case.

Example. There are many examples of when $C_{FP} \neq C_{FN}$. Among them:

- Consider the binary classification problem of whether a person pays back a loan, with a response of 1 meaning that they will pay back. A false positive means we predict they will pay it back, so we give them the loan, but instead they up default on the loan. A false negative means we predicted they would not pay, but in fact they would have paid it back, so we deny them the loan. In this case, the false positive is much more costly, so we might have $C_{FP} > C_{FN}$.
- Suppose we create an authorization system that predicts 1 if a person is authorized to access a secured facility and 0 otherwise. A false positive means we say that they are allowed to enter but in fact they are not cleared personnel. A false negative means we say they are not allowed when in fact they have received clearance. The latter is inconvenient but not a huge problem; the former might lead to secrets revealed to outsiders. In this case, a false positive is again worse than a false negative, so $C_{FP} > C_{FN}$.
- Consider a fire alarm that goes off, but there is no fire (false positive), versus when there is a fire, but the fire alarm does not go off (false negative). Here, the false negative is most costly, since lives are at stake, so $C_{FN} > C_{FP}$.

Predictions can be summarized in a 2 by 2 matrix called a **confusion matrix**:

		$\hat{\boldsymbol{y}}$		
	0	TN	FP	N
$\boldsymbol{y}$	1	FN	TP	P
		PN	PP	n

The meaning of each symbol is as follows:

- N is the number of $y_i = 0$.

- P is the number of $y_i = 1$.
- PN is the number of predicted negatives, i.e., $\hat{y}_i = 0$.
- PP is the number of predicted positives, i.e,. $\hat{y}_i = 1$.
- TP is the number of true positives, i.e., $y_i = 1 = \hat{y}_i$.
- TN is the number of true negatives, i.e., $y_i = 0 = \hat{y}_i$.
- FP is the number of false positives, i.e., $y_i = 0$ but $\hat{y}_i = 1$.
- FN is the number of false negatives, i.e., $y_i = 1$ but $\hat{y}_i = 0$.
- n is the total number of data points. Note $N + P = n$, and $PP + PP = n$. Also $TN + FP = N$, $FN + TP = P$, $TN + FN = PN$, $FP + TP = PP$ (easier to just see the table).

3.1 Performance Metrics

We can define several performance metrics we can define based on the quantities in the confusion matrix. We begin with **misclassification error**, denoted M_E:

$$M_E := \frac{FP + FN}{n}$$

We can also define an **accuracy** metric in terms of M_E:

$$\text{Accuracy} := 1 - M_E$$

We can also define **precision**, the proportion of positive predictions that are correct:

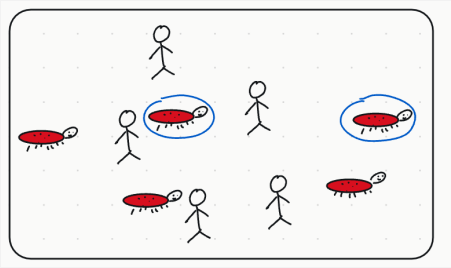
$$\text{Precision} := \frac{TP}{PP}$$

We define **recall** as the proportions of positives classified correctly:

$$\text{Recall} := \frac{TP}{P}$$

There is a tradeoff between FN and FP when using the precision and recall metrics.

Example. Suppose we are looking at images and we are trying to distinguish between a bug and a human. We will say the response $y = 1$ if it is a bug, and $y = 0$ if it is a human. Say our sample has 5 humans and 5 bugs, and our model makes two correct predictions about which should be bugs by circling them, but does not circle any other object (it thinks there are no more bugs):



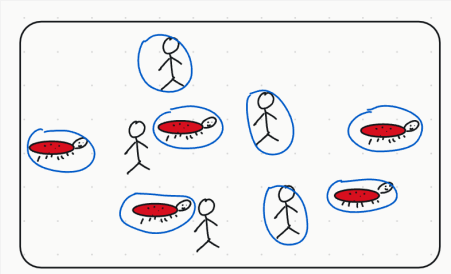
Then the precision is

$$\text{Precision} := \frac{TP}{PP} = \frac{2}{2} = 1$$

We achieved 100% precision, which sounds like we did a great job. Meanwhile, there are 5 humans in total, so the recall is

$$\text{Recall} := \frac{TP}{P} = \frac{2}{5} = 0.4$$

40% is not a good recall. Note we did not make any false positive errors. However we have three false negatives, since we did not circle three actual bugs. Thus implicitly we are saying that the cost of false positives is higher than the cost of false negatives.
Now say instead that we care more about FN. Imagine the model circled 8 objects, 5 of which are bugs and 3 of which are humans:



Then the precision is

$$\text{Precision} := \frac{5}{8}$$

The recall is

$$\text{Recall} := \frac{5}{5}$$

Here we are implicitly saying FN is most costly, because we did not make any false negative errors, in spite of making false positive errors.

We can combine precision and recall into a performance metric F_1, called the **geometric average**:

$$F_1 := \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

Another useful metric is **False Discovery Rate (FDR)**, the proportion of mistakes when $\hat{y} = 1$:

$$FDR := \frac{FP}{PP}$$

(Note FDR $= 1 -$ Precision). Related to FDR is the **False Omission Rate (FOR)**, the proportion of mistakes when $\hat{y} = 0$:

$$FOR := \frac{FN}{PN}$$

FOR and FDR also trade-off effectively, similar to how precision and recall trade-off. Say we weigh FP more heavily than FN ($C_{FP} > C_{FN}$). In that case, we are discouraged from making FP predictions, so we will be less inclined to predict $\hat{y} = 1$, and more $\hat{y} = 0$. Thus, the number of FP decreases and the number of FN increase, meaning FOR increases. If instead we decide that $C_{FN} > C_{FN}$, then we will do the opposite.

Another performance metric C is called the **total cost**:

$$C := C_{FP} \cdot FP + C_{FN} \cdot FN$$

The **average cost** AC is the cost incurred per prediction:

$$AC = \frac{C}{n}$$

We are concerned with how to tailor our algorithm to minimize the total cost, which in turn accounts for FP and FN costs.