Handling Missing Values in Machine Learning: Part 2

Georgios Drakos Follow
Jul 31, 2018 · 8 min read ★

Handling missing values is one of the worst nightmares a data analyst dreams of. Especially, if the number of missing values in the data are big enough (typically above 5%). In that case, the values cannot be dropped and a wise data scientist has to 'imputes' the missing values instead. In my previous story Handling Missing Values in Machine Learning: Part 1 I presented basic techniques of handling missing values in a dataset. The goal of today's article is to present some more advanced techniques where missing values are filled by exploring correlations.

Photo by Stephen Dawson on Unsplash

At first, it is important to understand that it doesn’t exist a good way to deal with missing data. Different solutions exist for data imputation which however depends on the kind of problem — Time series Analysis, ML, Regression etc. and it is difficult to provide a general solution.

Predictive techniques shall be used only when missing values are not observed completely at random and the variables chosen to impute such missing values have some relationship with it, else it could yield imprecise estimates.

In general, different machine learning algorithms can be used to determine the missing values. This works by turning missing features to labels themselves and now using columns without missing values to predict columns with missing values

Mean, Median in combination with groupby

In this case, we impute the missing values by not considering the global mean/median value of the observations but by investigating some dependency with other variables. The commands applied for the titanic dataset.

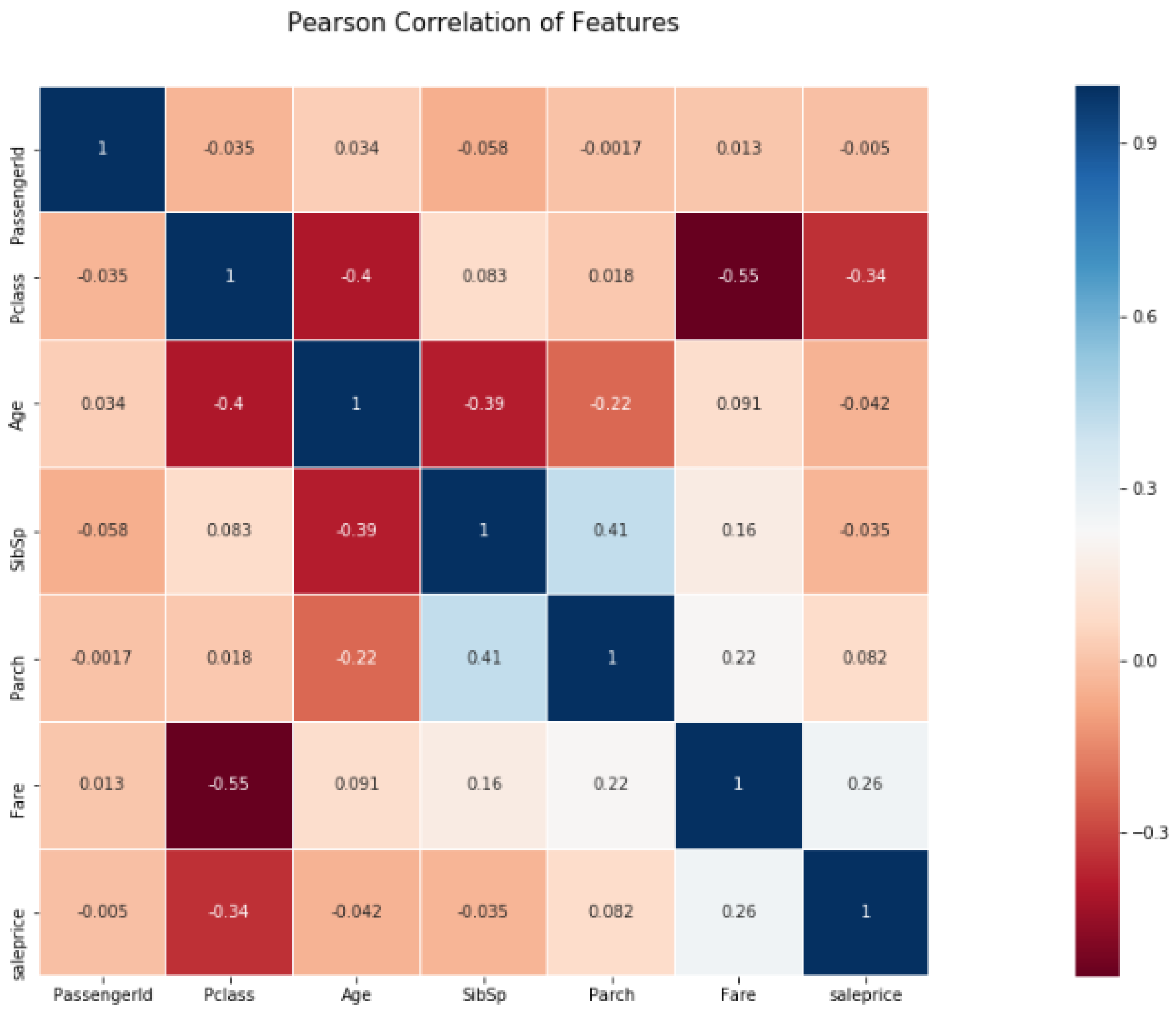
```
## Fill missing values in Age feature with each sex’s mean value of
## Age
train['Age'].fillna(train.groupby('Sex')['Age'].transform("mean"),
inplace=True)

## Fill missing values in Age feature with each sex’s median value ##
of Age
train['Age'].fillna(train.groupby('Sex')['Age'].transform("median"),
inplace=True)
```

Linear Regression

To begin, several predictors of the variable with missing values (in our case ‘Age’ feature) are identified using a correlation matrix. The best predictors are selected and used as independent variables in a regression equation. The variable with missing data is used as the target variable.

```
colormap = plt.cm.RdBu
plt.figure(figsize=(32,10))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train.corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white',
annot=True)
# Note that the categorical features have been neglected in the
# correlation matrix.
```



	PassengerId	Pclass	Age	SibSp	Parch	Fare	saleprice
PassengerId	1	-0.035	0.034	-0.058	-0.0017	0.013	-0.005
Pclass	-0.035	1	-0.4	0.083	0.018	-0.55	-0.34
Age	0.034	-0.4	1	-0.39	-0.22	0.091	-0.042
SibSp	-0.058	0.083	-0.39	1	0.41	0.16	-0.035
Parch	-0.0017	0.018	-0.22	0.41	1	0.22	0.082
Fare	0.013	-0.55	0.091	0.16	0.22	1	0.26
saleprice	-0.005	-0.34	-0.042	-0.035	0.082	0.26	1

We observe that the our target varible ‘Age’ is correlated with the features ‘Pclass’,’SibSp’,’Parch’,’Fare’ and they don’t contain missing values.

```
train.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
saleprice      891 non-null int64
dtypes: float64(2), int64(5), object(5)
memory usage: 90.5+ KB
```

Thus, we are going to fit a linear model and we will try to predict the missing values of the ‘Age’ feature. We used a Linear regression model to replace the nulls in the feature ‘Age’. In general, one can experiment with different algorithms and check which gives the best accuracy instead of sticking to a single algorithm.

```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
data = train[['Pclass','SibSp','Parch','Fare','Age']]

#Step-1: Split the dataset that contains the missing values and no
missing values are test and train respectively.

x_train = data[data['Age'].notnull()].drop(columns='Age')
y_train = data[data['Age'].notnull()]['Age']
x_test = data[data['Age'].isnull()].drop(columns='Age')
y_test = data[data['Age'].isnull()]['Age']

#Step-2: Train the machine learning algorithm

linreg.fit(x_train, y_train)

#Step-3: Predict the missing values in the attribute of the test
data.

predicted = linreg.predict(x_test)
```

```
#Step-4: Let’s obtain the complete dataset by combining with the
target attribute.

train.Age[train.Age.isnull()] = predicted
train.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
saleprice      891 non-null int64
dtypes: float64(2), int64(5), object(5)
memory usage: 90.5+ KB
```

Estimating the missing values by using a weighted least squares or generalized least squares model leads to better results (Lasso & Ridge).

The limitations of the model tend to outweigh its advantages.: They lead to an underestimation of standard errors and, thus, overestimation of test statistics. The main reason is that the replaced values are completely determined by a model applied to other variables and they tend to fit together “too well”, in other words, they contain no error. Also, one must also assume that there is a linear relationship between the variables used in the regression equation when there may not be one.

k-Neareast Neighbor (kNN) Imputation

Even though XGBoost and Random Forest could also be used for data imputation we will be discuss KNN as it is widely used. For k-Nearest Neighbor imputation, the missing values are based on a kNN algorithm. In this method, k neighbors are chosen based on some distance measure and their average is used as an imputation estimate.

The method requires the selection of the number of nearest neighbors, and a distance metric. KNN can predict both discrete (most frequent value among the k nearest neighbors) and continuous attributes (mean among the k nearest neighbors).The distance metric varies according to the type of data:

1. *Continuous Data:* The commonly used distance metrics for continuous data are Euclidean, Manhattan and Cosine
2. *Categorical Data:* Hamming distance is generally used in this case. It takes all the categorical attributes and for each, count one if the value is not the same between two points. The Hamming distance is then equal to the number of attributes for which the value was different.

One of the most attractive features of the KNN is that it is simple to understand and easy to implement. The main disadvantage of using kNN imputation is that it becomes time-consuming when analyzing large datasets. Also, the accuracy of KNN can be severely degraded with high-dimensional data because there is little difference between the nearest and farthest neighbor. Finally, the number of neighbors (k) has to be carefully selected when using kNN imputation.

```
from fancyimpute import KNN

#We use the train dataframe from Titanic dataset

#fancy impute removes column names.
train_cols = list(train)

# Use 5 nearest rows which have a feature to fill in each row's
# missing features

train = pd.DataFrame(KNN(k=5).complete(train))
train.columns = train_cols
```

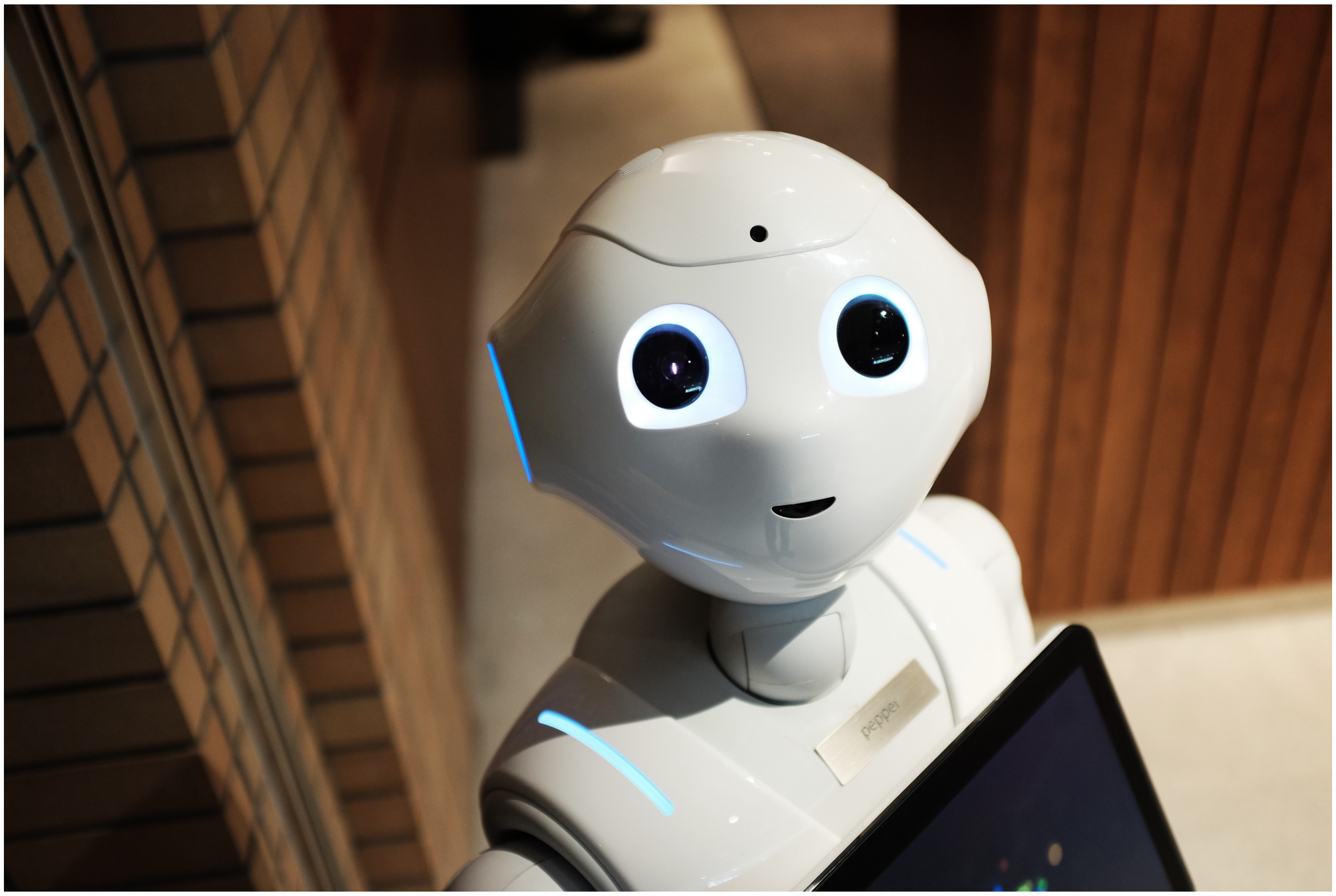
Multiple Imputation using MICE (Multiple Imputation by Chained Equations)

Unarguably one of the most advanced methodology for performing missing data imputation is Multivariate imputation by chained equations (MICE).

Creating multiple imputations, as opposed to single imputations to “complete” datasets, accounts for the statistical uncertainty in the imputations. In general, the limitation with single imputation is that because these techniques find maximally likely values, they do not generate entries which accurately reflect the distribution of the underlying data. Moreover, the chained equations approach is very flexible and can handle variables of varying types (e.g., continuous or binary).

Note that all of the techniques discussed so far are what one might call “single imputation”: each value in the dataset is filled in exactly once.

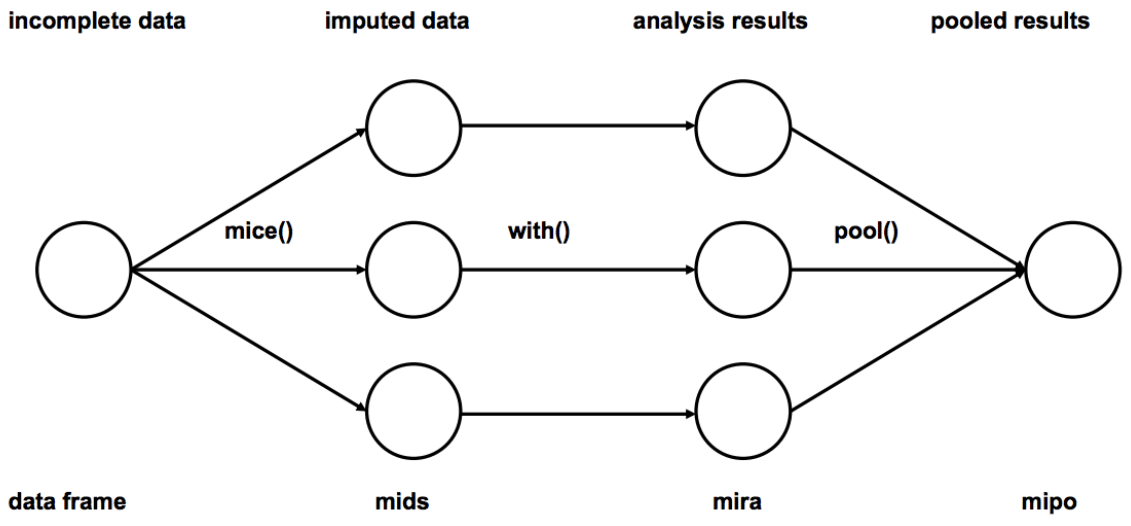
The MICE algorithm works: by running multiple regression models and each missing value is modeled conditionally depending on the observed (non-missing) values.



”A robot named Pepper holding an iPad” by Alex Knight on Unsplash

For example, let’s consider the extreme case when we impute the miisng values with the mean value. In reality, we would expect to to see some variability in it: extreme values, outliers, and records which do not completely fit the “pattern” of the data. All dataset contain noise in some extent and mean value replacement makes no attempt to represent it in its result. This leads to bias in any models, which are exposed to a trend (the presence of the mean value in the datset) which does not exist in the underlying data. This will ultimate decreases accuracy during both the train and test phases. The algorithm consists of three stages.

1. **Imputation**: Impute the missing entries of the incomplete data sets m times (m=3 in the figure). Note that imputed values are drawn from a distribution. Simulating random draws doesn't include uncertainty in model parameters. Better approach is to use Markov Chain Monte Carlo (MCMC) simulation. This step results in m complete data sets.
2. **Analysis**: Analyze each of the m completed data sets.
3. **Pooling**: Integrate the m analysis results into a final result



Source: http://www.stefvanbuuren.nl/publications/mice%20in%20r%20-%20draft.pdf

Here is the recipe for imputation using `fancyimpute.MICE` :

```
from fancyimpute import MICE

#We use the train dataframe from Titanic dataset

#fancy impute removes column names.
train_cols = list(train)

# Use MICE to fill in each row's missing features
```

```
train = pd.DataFrame(MICE(verbose=False).complete(train))
train.columns = train_cols
```

By default *fancyimpute* uses its own Bayesian ridge regression implementation.

```
model : predictor function
    A model that has fit, predict, and predict_dist methods.
    Defaults to BayesianRidgeRegression(lambda_reg=0.001).
    Note that the regularization parameter lambda_reg
    is by default scaled by np.linalg.norm(np.dot(X.T,X)).
    Sensible lambda_regs to try: 0.25, 0.1, 0.01, 0.001, 0.0001.
```

This is essentially a Bayesian implementation of `sklearn.linear_model.Ridge`. `lambda_reg` is equivalent to the `alpha` parameter. Thus a weakly penalized `Ridge` regressor is the package author's (reasonable) default.

This is by far the most preferred method for imputation for the following reasons:
- Easy to use
- No biases (if imputation model is correct)

Conclusion

The take-home message is that the advanced imputation methods design to address the problem of missing data exploit interrelationships between variables and impute multiple values rather than a single value. A suitable solution especially for big data scale data sets depends on the computational resource, as well as tolerance to errors in approximating missing values.

Among all the methods discussed above, multiple imputation and KNN are widely used, and multiple imputation being simpler is generally preferred.

P.S: Watch out for my next article which summarizes the missing values strategies.

Happy coding :)

Resources:

[1] https://www.bu.edu/sph/files/2014/05/Marina-tech-report.pdf
[2] https://arxiv.org/pdf/1710.01011.pdf

[3] https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/

[4] https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[5] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/

[6] https://github.com/iskandr/fancyimpute

Data Science Imputation Missing Values Multiple Imputation Mice Knn Imputation

About Help Legal