Integrull (Fuzzy Joins): Project Report

Jake Mainwaring, Chase Smith, Kochise Bennett, Lawrence Yan, and Cyril Tamraz

High-level Pitch (3-5 sentence pitch)

In the real world, most datasets are messy and have many missing values. Although there are workarounds – like excluding rows with empties – these methods often result in information loss and can be tedious for data scientists. Our python library, Integrull, is a tool to help you fill missing values in a more intelligent way. Each column is treated as a separate prediction problem, using the completed rows as training data. The library makes preprocessing easier so you can focus on your analysis.

Current system overview

Before diving into the details, we'd first like to discuss the tools our library is built upon. All of the code was written in python, relying heavily on pandas for data manipulation and sklearn for building our models. These tools made the most sense for our purposes, since it cleans csv files.

Otherwise, the tool is best explained by walking through its six key steps:

1. **Identify columns with null values** – this first step is fairly straightforward. Once a user imports a csv file and decides to use Integrull, any column with a 'NaN' value is marked as having empties.
2. **Label columns as 'classification' or 'regression'** – before training the models, the user inputs whether each column with nulls is a classification or regression problem.
3. **Split into training/prediction** – in each column, split the rows into rows where that value is empty and rows where it's not. For the rows where it's not, your training data becomes the array of complete values (response variable) and values from every other column (features).
4. **Train the model** – train a random forest classifier for each classification task and a ridge regression model for each regression task.
5. **Replace nulls** – update nulls with each column model's output. Every time this happens, it is added into a new data frame so as to not train the model on previous predictions.
6. **Clean up data** – before displaying the final dataset, the numeric values have to be returned back to their respective string values. Only then is the task officially complete.

Journey (where we started, what didn't work, what changed along the way)

When we initially started this project, we had a fairly different idea in mind. The project name 'fuzzy join' pointed us in the direction of approximate string matching and data cleaning. For example, if one value contained 'Guns & Roses' and another contained 'Guns N Roses', being able to detect that these two have the same intent and treating them as such.

After taking a closer look at the initial project idea, we opted for our new direction instead. This first idea seemed too focused on data cleaning and NLP, whereas our eventual idea allowed us to incorporate more from the course.

Another important point is around column labeling. Initially, we hoped to automatically label each column as a classification or regression problem. We experimented with different rule-based approaches – if the column has string variables, follows a certain distribution, etc. – but eventually decided against this. Imagine empty values in two columns, one with 1-5 Likert scale data and another with 5 regions encoded 1-5. These could be almost identical numerically. However, the Likert answers have an inherent low-to-high relationship between them and could be approached with a linear regression problem, while the 5-region question should not be. Therefore, we ended up adding a parameter where users choose 'classification' or 'regression' for each column. Our assumption is that if a user needs to clean data, presumably for a machine learning model, he or she is presumably familiar with this distinction. One other point to note is that, for classification problems where values were stored as string (e.g., 'South', 'North', 'East', 'West'), we added a function that maps these to numerical values (e.g., 0, 1, 2, 3) so that the models can handle them. At the end, we made sure to return these values back to their corresponding strings so users' data was not altered.

Future improvements

There are four key areas we would continue to build out, given the opportunity:

1. Expand functionality to SQL, R, and other languages commonly used for data analysis
2. Develop UI so analysts without coding experience can still use the product
3. Include an option for automatic detection of regression/classification
4. Improve performance and data storage limitations so our tool can be applied to larger data sets

Some of these areas would require additional expertise. That being said, we believe the library is far along enough to be helpful for those in the python community.

Team member contribution

- Jake – created all presentation materials (initial check-in, mid-semester check-in, and final presentation), wrote full report and news article, found and prepared datasets for testing, met with Ikhlaq and Alex, built the prototype from scratch (by 'prototype', this refers to all of the initial code for the program that runs on a standalone dataset, before turning it into a library)
- Chase – contributed to the upfront brainstorming and direction of our project
- Cyril – created the bulk of the code that took the prototype and turned it into a library
- Lawrence – worked on a lot of the library testing to make sure it functioned properly with various datasets

Mentor Experience
Ikhlaq was our primary mentor, and Jake was able to meet with him once during the semester. This conversation – which continued with Alex afterward – was very helpful in providing guidance and confirming we were on the right track.

All in all, this project was a great learning experience and we got a lot out of it. Thank you all for taking the time to review it!