Alison Sun
CSE 40
Prof. Lise Getoor
4 December 2023

Hands-On Assignment 5: Report

0 Introduction

My given metadata is a 1598 row x 15 col table of `object` values, with one column for an integer label. The main columns contain integers and floats, as well as strings of city names (`Bakersfield, San Francisco, Fremont, sAN jOSE`) and sports (`boxing, tennis, track & field`, etc.). Additionally, several columns contain mixed integer and string values that presumably represent weight (`1009 kg, 436 lb`, etc.). I predict that the metadata is some sort of university athlete statistics chart; however, the column names (`col_00, col_01, col_02`…) offer no indication as to the true meaning of these values.

	label
count	1598.000000
mean	2.507509
std	1.702591
min	0.000000
25%	1.000000
50%	2.000000
75%	4.000000
max	5.000000

Numeric information yielded by `unique_data.describe()`

At a glance, both linear and non-linear models alike would be suitable for this project. We can use logistic regression, decision trees, and *k*-nearest neighbors.

1 Data Cleaning

The data cleaning component of this project was split into four helper functions, which were then called by one function, `clean_data`.

In order to deal with missing or empty values in the table, I created a function, `drop_sparse_columns`, that takes in a pandas DataFrame and a float `sparsity_threshold` and drops the columns in the frame that have greater than sparsity_threshold of its

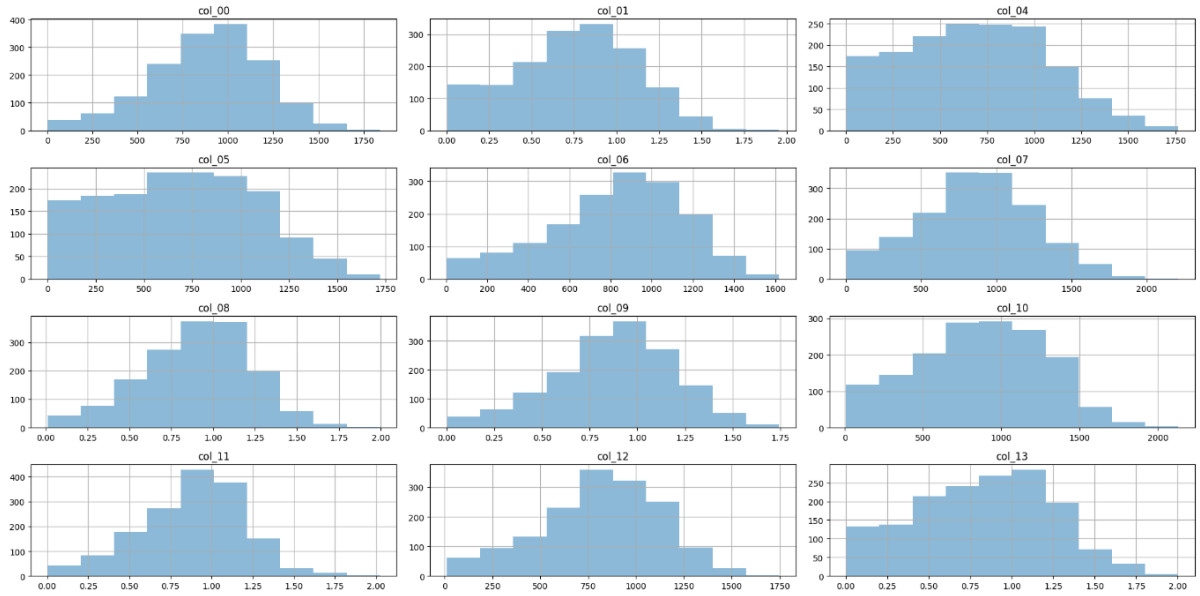
values empty. For instance, `drop_sparse_columns(frame, 0.50)` would drop columns with more than half empty values.

Another issue that arises in datasets happens when values in numeric columns contain more than just numbers; for instance, several cells in my metadata contained values like `52 kg` or `436 lb`. To standardize these columns so that they only contained numbers, I wrote another function, `extract_numbers`, that takes in a frame and a list of columns to ignore. `extract_numbers` searches through the columns not in the list, and uses regular expressions to extract non-numeric values and cast the numbers to integers or floats. For instance, after my data was cleaned, the value `219 kg` became `219.0`.

My metadata also contained a column of comma-separated string values, specifically, the names of cities; to deal with this, I implemented a one-hot encoding function that creates a column for each possible city and fills in each value with `0` or `1` depending on whether that value is present.

All columns of the uncleaned data had the type `object`, rather one that actually reflected the values in each column. To address this, my function `guess_types` iterated through each column and assigned dtype `int`, `str`, or `float` to each one accordingly.

2 Data Visualization



Histogram representations of numeric columns from `unique_data`.

These histograms of the numeric (non-one-hot encoded) columns seem to follow a Gaussian distribution, signaling that a linear model would work well to represent this data. Specifically, I will be using logistic regression to do so.

3 Modeling

Overall, logistic regression was the most accurate model to use — it gave accuracy values of up to 1.0, without having to tweak any parameters. Furthermore, I had originally intended to use decision trees as another way to model and predict my data, but switched to using the random forest model, which proved to be more accurate. Similarly to logistic regression, the random forest yielded extremely high accuracy values (about 0.997), without having to change anything in the parameters.

I also experimented with different values of `n_neighbors` for the KNN model. I initially ran the model with 4 neighbors, which yielded accuracy values between 0.975 and 0.9875. changing the number of neighbors to 9 gave me a highest accuracy value of 0.9905956112852664. Beyond

9 neighbors, the accuracy level actually ended up decreasing again, signaling that 9 was the ideal number.

Model	Mean Accuracy	Standard Deviation of Accuracy
Logistic Regression	0.999375	0.00125
Random Forest	0.993113244	0.0046098644931428
k-Nearest Neighbors	0.985607366	0.0037559467299064

4 Analysis

My accuracy rates were suspiciously high, which calls the possibility of overfitting into question. I believe that accuracy was a good evaluation metric for this project. Standard deviation was as well, in order to gauge reliability and fitting of the models; but, this measurement must be considered in context of the accuracy values as well. Standard deviations can be low, i.e. well-fitted or even overfitted, regardless of whether accuracy values are 0.01 or 0.99.

5 Conclusion

This project aimed to analyze an unmarked dataset (i.e., no column names to signal what the data meant) and predict the label column using various ML models. Moreover, statistical significance was found between all models.

Future improvements could focus on exploring feature selection techniques, as well as different classification and data visualization methods. Additionally, I'd like to spend time considering ways to avoid overfitting, due to my suspiciously high accuracy levels. Another approach to this assignment taken by some of my peers involved skipping one-hot encoding altogether, and rather just dropping non-numeric columns. I also wonder what this project would look like if I hadn't encoded non-numeric columns; this was convenient for standardizing data types in the metadata, but added a lot of redundant columns consisting mostly of zeroes.

Overall, I was interested in approaching this machine learning assignment as a problem with “no right answers,” especially because all my past work in computer science classes have involved problems with clear answers and not a lot of creativity in exploring ways to solve them.