Harsh Dadhich
CSE 40 - Fall ‘23

Hands-On Assignment 5

★ Introduction

I am given a dataset of **13 columns** of which there is 1 label and 12 data-specific columns. The **label** is a response variable, while `col_00` to `col_11` are features. The data types are a mix of numeric (integer and float) and categorical values and data. There are some columns for which you are able to discern the unit of data in use. For the shape, the data set is exactly 1547 rows x 13 columns.

We can split up the numeric and categorical features into the following:

1) Numeric Features
    a) Unit-wary
        col_00 - ± (less neg), `lbs`, a weight value
        col_07 - ± (less neg), `m/s`, a velocity value
    b) Floats
        col_01, 06, 10 - ± (few neg), few >1
    c) Integers
        col_05, 09, 11 - ± (few neg), few >1
2) Categorical Features
    col_02 - Name
    col_03 - Location(s)
    col_04 - Animal (pet?)
    col_08 - Major

Proposed Models

a) Logistic Regression
    This one is a suitable choice for this dataset for binary classification tasks, as it can help predict categorical outcomes. Given `label` column, logistic regression could provide more factors that influence the classification.
b) Random Forest Classifier

This one is known for being able to handle both numeric and categorical data. It can capture the complex relationships within the dataset and helps identify what features are important, and what aren't

c) Support Vector Classifier
This one is suitable for both classification and regression asks, as it can handle, again both numeric and categorical features. However, as opposed to the RFC, this captures complex decision boundaries, making it versatile for this dataset.

Understanding & Goals

The exploration of this dataset involves deciphering the meaning and significance of each of the columns. Through patterns or anomalies, we would classify this data for analysis. The goal is to take this information to take this data to build models that can predict the `label` column based on the features (rest of the columns). The following sections will delve into the steps taken for data exploration, cleaning, and visualization paired with the modeling and analysis.

★ Data Cleaning

For the data cleaning, I have imported many of the functions from my Hands-On Assignment 3 submission. In order to process this data through a machine learning algorithm, we would need to turn it into a format that is discernable to Python. For any values that contain digits, this would be obviously converting them to their corresponding integer or float forms. As for the categorical data, since these are objects, we would need to specify them by boolean values corresponding to what labels contain what categorical data out of the larger set of possible values.

Here are the functions we've used, and an explanation for their functionality:

1) `drop_sparse_columns`
    a) **Why**: When you have columns that have a high percentage of missing data, it may end up providing little to no meaningful information to the model. Removing these columns reduces the "noise" (irrelevant data) in the dataset, and improves computational efficiency by decreasing the size.

   b) **How**: We’ve defined a function to identify columns with a sparsity threshold that exceeds `0.5` and drop them from the dataset.

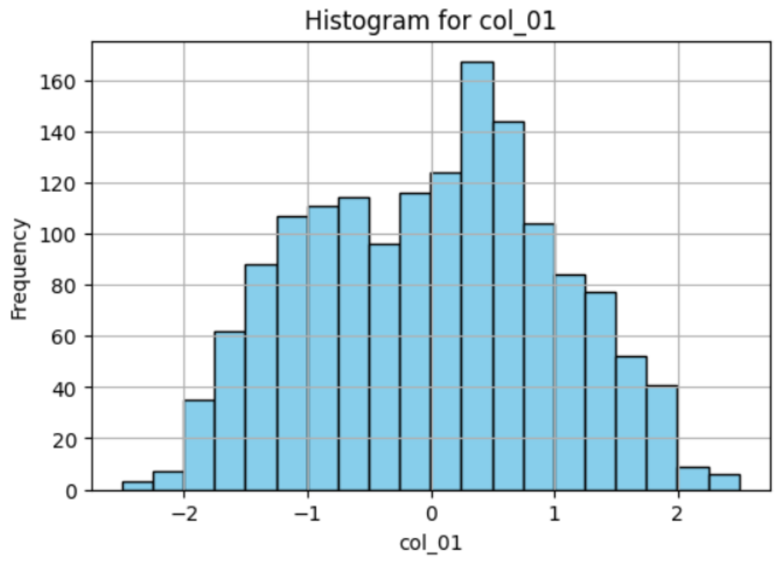
2) `extract_numbers`
   a) **Why**: All the data is read in as strings as to include the units (lb, m/s) and NaN (?) values. Extracting the numbers for those that have extractable numbers ensures a consistent representation for all data types.
   b) **How**: Created a function to capture integers and floats through a regex compilation pattern. We’ve handled different representations including units, decimal points, and negative signs.
3) `one_hot`
   a) **Why**: The words on our computer screens are all combinations of numerical digits under the roof. This is why we need to convert our categorical variables into encoded binary representation, so it is readable by our machine learning algorithm.
   b) **How**: We’ve utilized a function to perform one-hot encoding on selected categorical columns we’ve passed through a function parameter. There are binary columns for each unique feature within each column. This clearly heavily expands the output dataset in its size. However, these are simply True or False values laid out in 1’s and 0’s.
   c) **Other**: We have missing data in both numerical and categorical data. For numerical data, it is easy to leave the field blank to indicate a missing value. We could do the same by having missing values indicated by 0’s across all unique features within each column. However, a column that indicates if the value is missing or not increases computing efficiency. There is a `colXX_nan` column for each category.
4) `guess_types` - We need to ensure the correct data types for each column for model compatibility. This is why columns are casted to appropriate data types based on the predominant one in each column.
5) `int_casts` - All numeric values are forcibly converted to floats prior to calling this method. This method is called to ensure features meant to be integer values are changed to their intended data type.

★ Data Visualization

Visualization 1: Histogram for `col_01` after Feature Scaling

I created a histogram for this column by using the `StandardScaler` pre-processor from the scikit-learn library. Feature scaling is important to standardize the numeric features and ensure that the mean of the set is 0, and the standard division is 1. Doing this is particularly important for machine learning algorithms that may be sensitive to the range of input features, as it prevents any specific one from dominating the learning process.

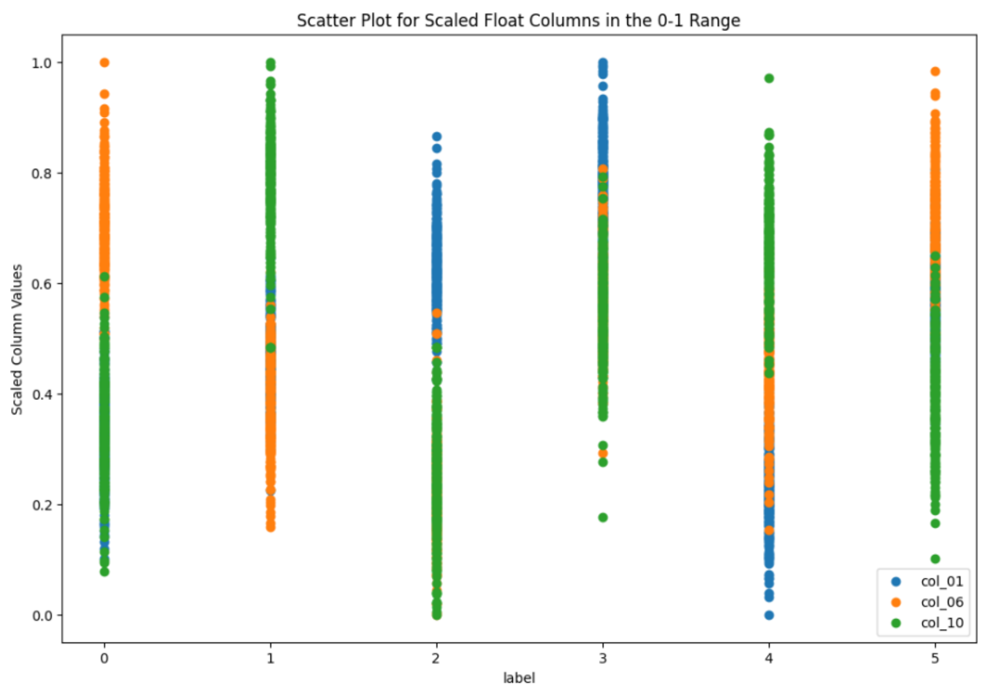
1) **Preparation**: For preparing the data, I’ve scaled the numeric features to standardize their values. I want to see how well my data confirms the standard normal distribution after being scaled. Since `clean_data` leaves an empty spot for missing values, I’ve made sure to account for that by using the median.
2) **Creation**: For creating the visualization, I’ve made use of the `matplotlib.pyplot.plt` function. There is an in-house way to create a histogram by setting axes. The y-axis is set to the frequency of the occurrences, while the x-axis displays the scaled values.
3) **Insights**: The distribution shows the spread and concentration of data points, Although from a glance it seems that there are largely more positive values, we see that there are considerable amounts of negative ones.



Visualization 2: Scatter Plot for Scaled Float Columns

I created a scatter plot to explore the relationships between the scaled float columns (`col_01`, `col_06`, and `col_10`) to see if there are any patterns or trends between these features and `label`. This plot is effective in revealing potential correlations and understanding how variables interact.

1) **Preparation**: To prepare the data for the scatter plot, I used the `MinMaxScaler` from scikit-learn to scale the values of the selected float columns to the range [0, 1]. Additionally, I handled missing values like before by filling them with the median. Now the data is not influenced by outliers or extreme values.
2) **Creation**: For creating the visualization, I’ve made use of the `matplotlib.pyplot.scatter` function. Each point on the plot represents an observation, with the x-axis denoting the label and the y-axis representing the scaled values of the float columns. There is a legend for the different columns.
3) **Insights**: The scatter plot allows us to visually assess if there are any discernible patterns. It appears that there is an inversion pattern, particularly between `col_01` and `col_06` features. Further modeling can be used to explore this.



★ Modeling

I used three classifiers to explore and evaluate their performance on the dataset. The chosen classifiers were Logistic Regression, Random Forest Classifier, Support Vector Classifier (SVC).

1) `sklearn.linear_model.LinearRegression()`
    a) **Description**: This is a model often used for binary classification problems. It models probabilities that a value belongs to a particular class.
    b) **Parameters**: No special parameters were used, and the warning related to convergence was observed. This could probably be modified by adjusting the `max_iter` param, increasing iterations.
2) `sklearn.ensemble.RandomForestClassifier()`
    a) **Description**: This is a model often used for decision trees that output the mode of the classes as the prediction.
    b) **Parameters**: No special parameters were used, and it resulted in perfect accuracy in the k-fold cross-validation.
3) `sklearn.svm.SVC()`
    a) **Description**: This is a powerful classifier that checks off both binary and multiclass classification. It finds the decision boundaries between the classes, specifically a hyperplane, covered in M8.
    b) **Parameters**: No special parameters were used, however, the performance was relatively lower compared to the other classifiers.

Model	**Mean Accuracy**	**Standard Deviation of Accuracy**
Logistic Regression	0.836	0.347
RandomForestClassifier	1.000	0.000
Support Vector Class.	0.375	0.034

Table 1

1) **LR vs RFC (False)**
   The p-val > 0.1, indicates that there’s not enough evidence to reject the null hypothesis. This makes it not statistically significant.
2) **LR vs SVC (True)**
   The p-val < 0.1, suggesting enough evidence to reject the null hypothesis. This makes it statistically significant at the 0.10 significance level.
3) **RFC vs SVC (True)**
   Again, following the previous comparison, the p-val < 0.1, indicates enough evidence to reject the null hypothesis. Therefore, it’s statistically significant at the 0.10 significance level.

Summary

The significance test implies that there are significant differences in accuracy between LR and RFC/SVC. SVC needs to be further explored and tuned to enhance its predictive capabilities. The chosen classifiers have their specific strengths as per their descriptions above. RFC stands out with the highest accuracy, however, there is likely an issue regarding the overfit of training data. As for standard deviations, we get insights into the variability of accuracy scores. The lower values indicate more stable and consistent model performance. LRC has a high accuracy but has higher variability. SVC is lower but seems to be a more stable model.

★ Analysis

Classifier Performance

The results suggest different variations in the performance of each of the classifiers. As we’ve already discussed, one of them has higher accuracy and low variability; another is vice-versa, and the final one has full accuracy and no variability. The superior performance is probably noted by its capture of complex relationships in the data, and handling it in non-linear patterns. The reasons for the likely misrepresented accuracy may be due to overfitting, data leakage, or parameter tuning. Overfitting occurs when it learns random irrelevant data in the set, therefore performing very well. Parameter tuning, such as adjusting the maximum depth of trees or minimum samples can mitigate the first issue of overfitting data.

Evaluation Metrics

Depending on the context, other metrics like precision, recall, or the mentioned F1 score could provide more insight, especially for our dataset, since it may appear imbalanced. However, if there are more instances of one class than the others, accuracy on its own can be inaccurate.

Data Issues with Cleaning

The high standard deviation we received in Logistic Regression could indicate outliers or features with high variance. This may indicate that the sparsity threshold may need to be modified individually for each column. Further investigation into which features matter more or the distribution could help us tune the arguments to the helper functions for `clean_data`.

Statistical Significance

The statistical significance aligns with the expectations from each of the classifiers. We've mentioned this in the evaluation of the modeling section.

Classifier Tuning

Parameters like regularization strength in Logistic Regression or the choice of kernel in SVC could be modified to improve our performance. We'd have to select the trade-off between training the data well and preventing overfitting for LR. In SVM, we'd choose between polynomial, linear, radial, etc. as parameters to see which one provides better accuracy. Grid Search can be used by searching through all the possible combinations of parameters for each model. Another one can be Random Search, as you can guess by the name, this creates random configurations. This can often perform better in spaces with high-dimensional datasets, specifically like ours since we have hot-encoded values for 4 categorical variables.

★ Conclusion

In conclusion, this project successfully explores and models the dataset and provides a foundation for making predictions. Each section of the project is a pipeline into the next. The cleaned data allows for modification and scaling for us to visualize the data successfully through a histogram and scatter plot. We were able to visualize the float columns, offering insights into the data distribution and correlations between inverse relationships between those similar data types. The analysis emphasized the strengths and weaknesses of the classifiers we employed and prompted further investigation into the individual parameters and techniques to increase accuracy.

★ References

[1] McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[3] van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering, 13(2), 22-30.