Assignment 01 - Data Preparation

Due Date	@November 10, 2022
Course	Data analysis for decision making
Master	MCSBT
Professor	Jesús García San Luis

Data Preparation

Introduction

Data preparation, also known as ‘preprocessing’ or ‘formatting’, consists on the act of cleaning and consolidating the raw data before submitting it to analysis. Maybe it is not the most appraised task, but performing a preparation of the data is a thorough key component for a correct data analysis.

Carrying out the process of validating, cleaning and augmenting the raw data is the foundation to obtain precise and significant insights from them. The validity and power of a data analysis depends on the efficiency of the previous preparation made.

The aim of this assignment is to practice some of the data preparation techniques through a hands-on exercise. The next variations and questions have to be answered.

1. I have dropped columns ‘nameOrig’ and ‘nameDest’. Can you explain why?

After importing the necessary libraries and modules into our code, and after setting up the data, the first step we must take is understanding what feature each column represents. This is crucial because failing in the understanding of these features could lead to mistakes in our analysis and modelling. After examining what every column represents, one of the most usual operations in data cleaning is **narrowing down the columns**. One of the reasons one could drop a column from the data set is because it does not provide useful and significant information for the analysis to succeed and the model to produce accurate results.

In this case, we are dropping variable columns **‘nameOrig’** and **‘nameDest’** for this previous reason. These columns represent the origin and the destinations of the transaction, these values are of **type string** and there is a unique combination of characters for each origin and destination (it is likely but not necessarily). Since the model trainer uses numeric values for the evaluation of the data, type strings should be encoded (or a factorisation should be done) for the trainer to be able to analyse the data.

It is important to know this to determine whether or not these columns are significant for modelling, and as we are trying to predict if a financial transaction is fraudulent, we come to the conclusion that that

origin and destination are not relevant for this prediction.

2. The program uses an arbitrary encoding for the column ‘type’. Replace it by one hot encoding and check if the model performs better

Machine learning algorithms are able to handle or support categorical values without the need of manipulating the data. Categorical values are those that can only take a fixed, limited amount of values. There are several types of encoding but the ones used in the exercise include:

- **Replacing values**, where we interchange the categories with numbers. This is the most simple way of encoding and implies converting the values to of the categories to an integer.
- The other type of encoding we use is **one hot encoding**, where each category is transformed into its own column. This implies assigning a true or false value, 1 or 0, if the observation belongs to that category or not.

→ In the next table I show the results for the accuracy and ACU for the arbitrary encoding and one hot encoding scenarios:

	Balanced	Unbalanced
Arbitrary encoding	0.9308600337268128 0.9301987119250074	0.9987455736339019 0.7042735128305146
One-hot encoding	0.9308600337268128 0.9301987119250074	0.9987455736339019 0.7042735128305146

As we can see, the results are identical, I have found two possible ways to explain this behaviour:

- Firstly, We can argue the model is already making a good interpretation of the type of transaction (**category/categories** columns), so when you change the encoding method, it makes the same interpretation of the encoding.
- The second explanation is that the solution or accuracy of the model does not vary because the type column might not be representative to determine if a transaction is fraudulent or not, in other words, it is not significant.

3. The program does not standardise the data. Insert the code needed to standardise the values and check the model performance.

The standardisation is a data preparation procedure that is executed with the aim of rescaling the columns so that the mean and standard deviation are 0 and 1 respectively. Usually, the standardisation of the data implies the improvement of results, because a standardisation is made with the aim of preventing that the higher value variables do not mask the effect of those with smaller values. For that we use the next formula:

$$\frac{d[column] - d[column].mean()}{d[column].std()}$$

→ By implementing the standardisation, we obtain the next results:

	Balanced	Unbalanced
Arbitrary encoding	0.5109612141652614 0.5134537370218423	0.9985511606915546 0.5
One-hot encoding	0.5109612141652614 0.5134537370218423	0.9985511606915546 0.5

As we can see in the results, the performance of the model decreases significantly when we standardise the data. We can argue that this procedure is not useful in this example.

> 💡 One thing I noticed, is that the code you gave us, at the time of standardising the data did not take into account the column 'newbalaceDest'. So I took the initiative to try the code including this column in the standardisation.

→When I did this, I noticed that the accuracy and ACU also decreases compared to the non-standardised data frame, but not as much:

	Balanced	Unbalanced
Arbitrary encoding	0.8819561551433389 0.8834143155096147	0.9989677598537274 0.6613163572581716
One-hot encoding	0.8819561551433389 0.8834143155096147	0.9989770176128868 0.658130740270884

```
# Insert here the code to standardize the dataest values and check
# if the model performance improves
for column in ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest']:
    d[column] = (d[column] - d[column].mean()) / d[column].std()
d
```

```
# Insert here the code to standardize the dataest values and check
# if the model performance improves
for column in ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']:
    d[column] = (d[column] - d[column].mean()) / d[column].std()
d
```

---

4. I have balanced the dataset because it is very unbalanced (it has many more valid transactions than fraud transactions). Please, try the program with the unbalanced dataset and check the result (you only need to comment one line)

A balanced dataset is one in which the same number of input samples are used to represent each output class (or target class). Oversampling or undersampling are some ways that can be used to achieve balancing. One of the non-written rules of machine learning is that the data set ought to be balanced, or at least near to being balanced. The major justification for this is to give each class equal priority. Data balancing is important when predicting variables with low probability of taking place, just like in our example, where fraudulent transactions considerably exceed in number the non-fraudulent ones:

1482 fraudulent vs. 1078691 non fraudulent, a ratio of 0.00137.

→ Taking a look at the non standardised scenario:

	Accuracy score	AUC (Area Under the Curve)
Non Balanced	0.9987455736339019	0.7042735128305146
Balanced	0.9308600337268128	0.9301987119250074

I have already done this in previous questions. Nevertheless it is important to highlight how the model performs when we execute the code with the unbalanced data frame vs a balanced dat frame: the output is a higher accuracy, however, we observe a much lower AUC with the unbalanced set.

In this case what we have to take into account is the AUC score, since what we are trying to predict is the fraudulent transactions the AUC score is far better to measure the accuracy of the predictions. Taking this into account, the BALANCED data set is far better.

5. Why do you think the accuracy is better with the unbalanced dataset but the AUC (the right score to use) is worse?

To answer this question we have to look at the way each result is calculated.

- On the hand, the accuracy is calculated as the amount of well predicted observations overall, this is taking into account all the observations in the data frame, fraudulent and non-fraudulent ones. So if we look at this result more in detail, we come to the conclusion that since there are way more non-fraudulent observations and the model does a great job predicting these cases, the accuracy is very high with this result.
- However, on the other hand, the ACU (Area Under the Curve) takes into account for the calculation of the result the true and false negatives for the fraudulent and non fraudulent cases.

→ So taking a look at the confusion matrix, as we can see, the UNBALANCED data sets produces a lot of false negatives for the non-fraudulent observations, whereas with the BALANCED data set, this number reduces considerably:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(testY, predY)
cm
```

UNBALANCED CONFUSION MATRIX	0	1	BALANCED CONFUSION MATRIX	0	1
0	215636	86	0	244	27
1	185	128	1	43	279