Data Integration

Final Project

Narges Fardnia

July 2024

Contents

Introduction

In this course, we explored the foundations of data integration techniques and their applications to big data. We utilized various tools, including Hadoop, Apache Spark, MapReduce, and cloud computing, to effectively manage and analyze data. This report aims to demonstrate the skills and knowledge acquired throughout the module by applying these techniques in practical scenarios.

The report is divided into two parts. The first part focuses on building a machine learning classifier using Apache Spark, showcasing the process and outcomes. The second part compares the efficiency of parallel computing with traditional computing methods, highlighting the advantages and performance differences observed.

Part 1: Classification Model with PySpark

Apache Spark™ is a multi-language engine designed for executing data engineering, data science, and machine learning tasks on single-node machines or clusters (Anon., n.d.). In this part, I will utilize this powerful engine to classify a dataset from Kaggle using one of the most popular programming languages, Python. Furthermore, I will implement this code on the Databricks platform to leverage the benefits of cloud computing.

The dataset used in this analysis is available on Kaggle, and my code can be accessed here.

EDA

The original dataset was divided into training and test sets, containing 103,904 records and 25,976 records, respectively. To improve randomness, I concatenated these datasets, resulting in a combined dataset with 129,880 rows. The target feature is "satisfaction," and the goal is to predict the conditions under which passengers are satisfied with the airline.

The dataset consists of 25 columns, one of which ("Unnamed: 0") can be removed as it serves only as an index and does not provide useful information. Another feature, "Arrival Delay," has 393 missing values, which can be imputed using the median value of all arrival delays.

Five features are categorical: Gender, Customer Type, Type of Travel, Class, and satisfaction. The distribution of these values, considering satisfaction, is illustrated in Figure 1. I converted these categorical values to numerical values by indexing.

It is important to note that the difference between the two classes of the target feature (neutral or dissatisfied, and satisfied) is about 13 percent, which indicates a relatively balanced dataset.

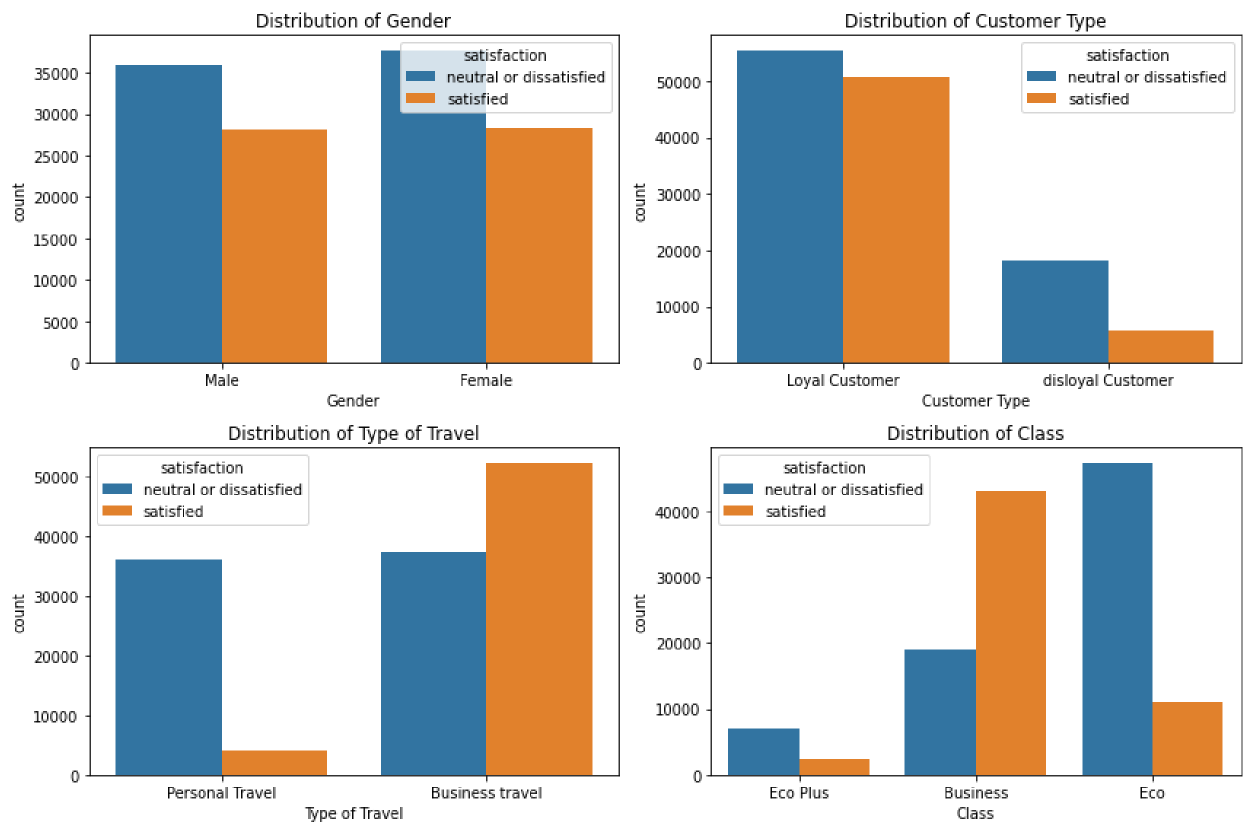

Figure 1. Distribution of categorical features.

Feature Engineering

To build a robust model, we need to select the most relevant variables from the data. To accomplish this, I used the measure of correlation and chose the columns that have the strongest correlation with the target variable. Conversely, it is advisable to omit features that have high correlation with each other to prevent multicollinearity. However, due to the limited number of features, I only removed the column with the highest correlation to others.

Figure 2 illustrates the correlation between each feature and the target variable, satisfaction. By focusing on these correlations, we can enhance the model's performance by ensuring that we use the most informative features.

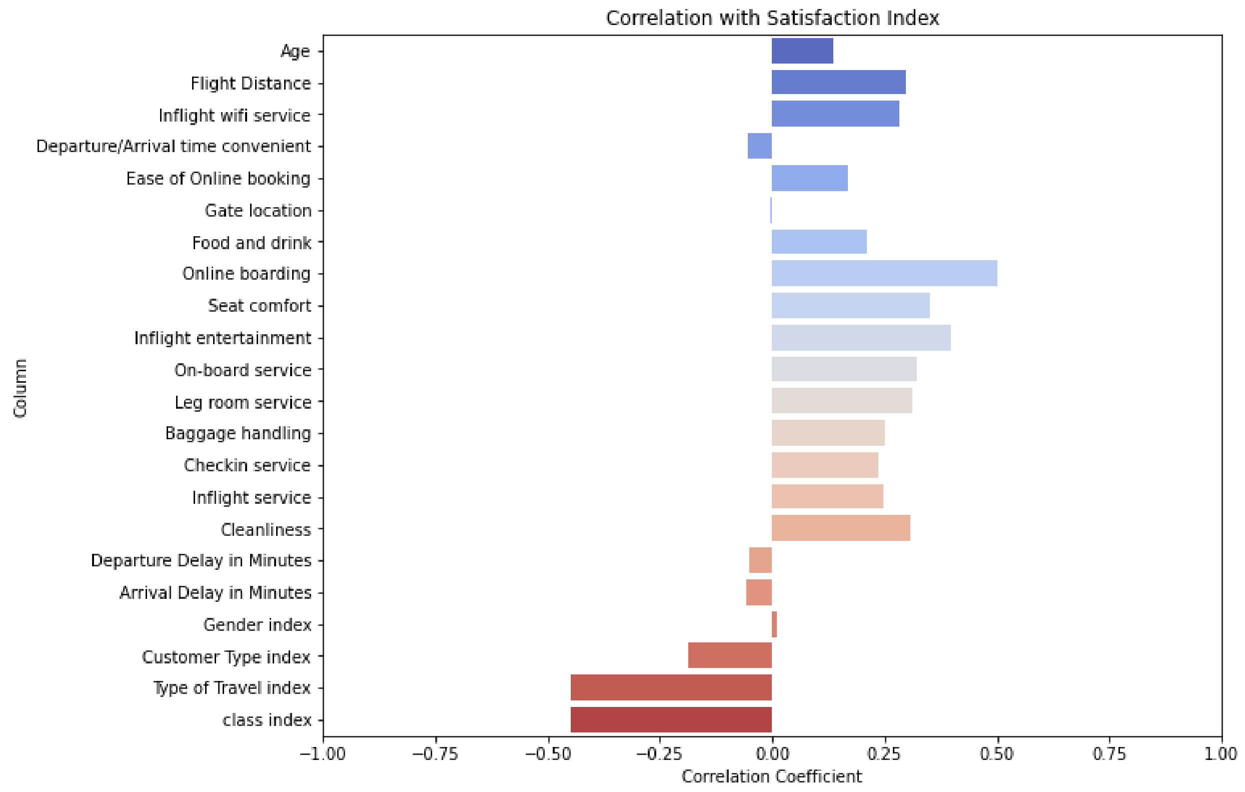

Figure 2. Correlations of features with the target.

Model

After selecting appropriate features and converting them into a vector format, I employed the Random Forest algorithm to construct a machine learning model. I allocated 70% of the data for training and reserved 30% for prediction. Surprisingly, the model achieved 100% accuracy. Upon reviewing the train-test splitting process, I discovered duplicate values present in both sets. This duplication likely explains the unusually high accuracy observed.

Part 2: Parallel Computing

In this section, I aim to compare traditional computing with parallel computing. To accomplish this, I have defined a function to count words on different websites. The code for this section can be accessed here.

I executed this code on Google Colab using its default CPU, which is an Intel Xeon CPU with 2 vCPUs. In the first part, the function `count_words_in_websites` counts the number of words on each of the specified websites. The second function, `words_list_in_websites`, provides a list of occurrences for each word on the given websites.

In the second part, I divided the process into 4 parts and performed the same word counting using `ProcessPoolExecutor` for parallel computing. The execution times of these computations are compared in Table 1 and Table 2. Although the program is not compute-intensive, as anticipated, using parallel computing accelerates the process.

Table 1. Computation time for word counting on websites

Website URL	Traditional Computing Time (s)	Parallel Computing Time (s)
https://www.berlin.de/en	-	1.58
https://en.wikipedia.org/wiki/%22Hello,_World!%22_program	-	1.58
https://cloud.google.com/learn/what-is-data-integration#:~:text=Get%20the%20report-,Data%20integration%20defined,make%20faster%20and%20better%20decisions.	-	1.58
https://www.gisma.com/faculty-and-team/	-	1.58
https://www.databricks.com	-	1.58
https://de.linkedin.com	-	1.58
https://spark.apache.org	-	1.58
Wall time	5.38	1.59

Table 2. Computation time for printing the occurrence of each word on websites

Website URL	Traditional Computing Time (s)	Parallel Computing Time (s)
https://www.berlin.de/en	-	2.2
https://en.wikipedia.org/wiki/%22Hello,_World!%22_program	-	2.3
https://cloud.google.com/learn/what-is-data-integration#:~:text=Get%20the%20report-,Data%20integration%20defined,make%20faster%20and%20better%20decisions.	-	2.3
https://www.gisma.com/	-	2.3
https://www.databricks.com	-	2.4
https://de.linkedin.com	-	2.3
https://spark.apache.org	-	2.4
Wall time	8.78	2.48

References

Anon., n.d. *Apache spark.* [Online]
Available at: https://spark.apache.org/