Real-Time Car Price Recommendation Engine built on Spark Streaming in Databricks

Michał Wójcik Marek Rewoliński

1. Introduction

In the era of Big Data, the ability to **analyze and extract insights** from large amounts of data **is increasingly valuable**. The aim of this project, using PySpark on Databricks, is to demonstrate the features of these tools to analyze vast amounts of data. Throughout the work, all the proposed functionalities were implemented, including: **RDDs**, **DataFrames**, **SparkSQL**, **Spark Machine Learning**, and **Spark Streaming**.

By leveraging these capabilities, we intend to discover insights in our data and simulate pipelines that can be applied to real-world challenges. This ensures code scalability and that all the steps taken can be reproduced in Big Data scenarios.

2. How much is your car worth? – real time price prediction using streaming

Abstract

In 2023 there were almost 100k used cars sold every single day in the US and almost **36 million times**[1] someone asked him/herself a question: **"How much is my car worth?".**

With the utilization of Big Data, Machine Learning and rich historical data we are able to answer this question in real time by analyzing the prices for similar cars with certain specifications.

Preparing distributed architecture

Big Data **distributed architecture** requires data to be stored in different **partitions** that are often not on the same machine, which makes moving the data across partitions computationally expensive. This is why initial data partitioning plays a key role. We have **separated our listings data by posting month**, which enables us to analyze listings from the same month fast and efficiently. Each month the listings are differentiated among different manufacturers, car models, etc. which gives us a possibility to create a meaningful and well generalized analysis of the current market. Ensuring an even distribution of data across partitions is crucial, and partitioning by posting month provides the best chance of achieving this, as a similar number of cars are typically listed each month.

Preparing dataset for modeling

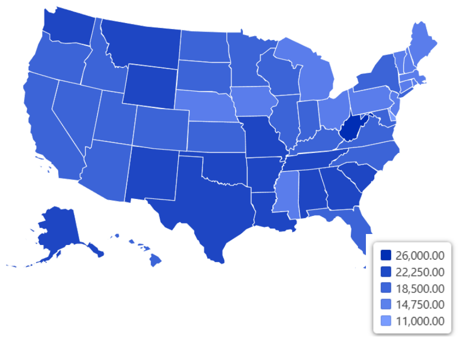
There are a handful of data and specifications that make a car its value. Utilizing Big Data capabilities we efficiently preprocessed 18 different data points about **430k car listings** from the United States from April and May 2021. We carefully selected meaningful data, by removing duplicates of the same listing and keeping the newest one, dropping empty rows or listings with errors (production year>current year) and price lower than 500 and odometer lower than 1000. We have also applied some data transformation to ensure inputs are in the numerical formats when possible.

[1] Cox Automotive. (2023, December). *Used vehicle inventory: December 2023*. Link

What dictates cars' price?

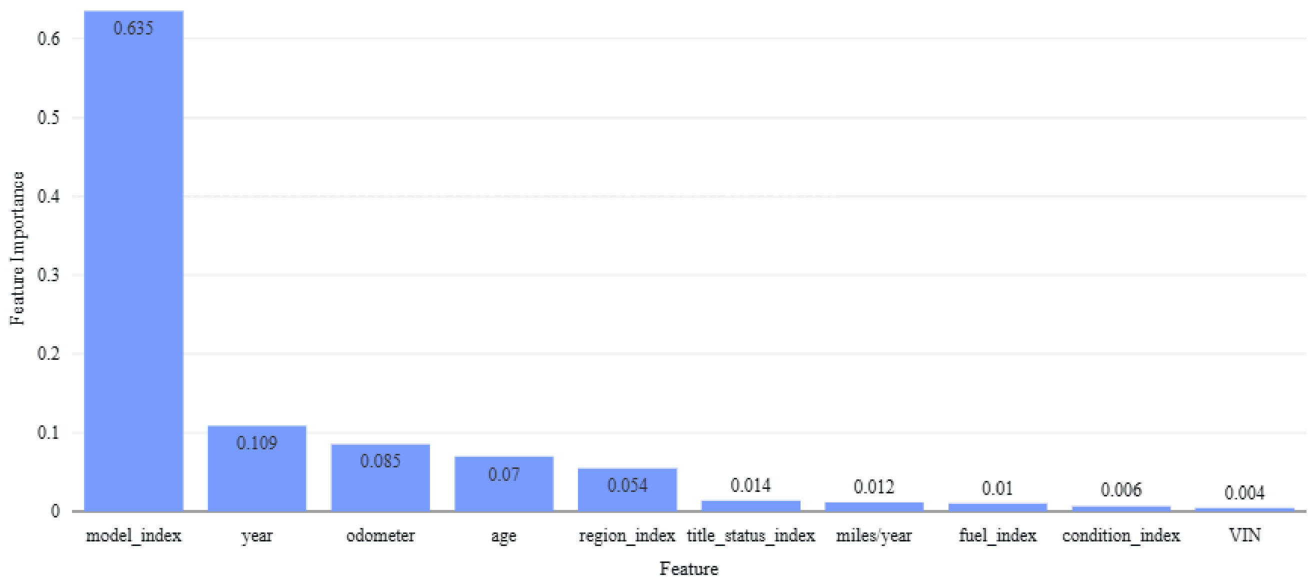
Finally our model was trained on **180k unique listings of 3,500 distinct car models** produced by the top 20 most popular car manufacturers.

The most important features (see chart below) that impact a car's price according to our model are: car model, age, odometer, miles driven per year, condition, fuel type, title status, whether VIN was added in listing and also region where the car was listed for sale.



That said, car listings in states like Virginia, Texas or Montana have higher median prices, which can be an opportunity for residents of other states to list and sell their car in selected regions for higher profit.

While not all of these features were directly provided by the sellers in the listing, they were essential for the model's performance and were derived through preprocessing steps. Specifically, car age and miles driven per year were calculated based on the available data.



Recommendations for future Model updates

While the dataset covers only two months and does not account for seasonality or the longevity of individual listings—such as cars that have been on the market for extended periods—this limitation is recognized. However, the model has been developed using Apache Spark, which allows for **efficient training on large datasets**. This scalability means the model can be easily **retrained with data spanning longer periods,** enabling future updates that incorporate seasonal trends and the age of listings.

In order to develop and maintain the tool we recommend **collecting new listing data on a monthly basis** and every **6 months retrain the model** to be able to offer accurate and valuable price recommendations.

How accurate are our recommendations?

On average, our model **misprices a car by $2350** during prediction. It **struggles** the most with cars that are **listed far below their market value**, making it a useful tool for predicting prices of cars which were not involved in incidents that would significantly lower their value like crashes or floodings.

Real time price recommendations

With the power of the **Pipeline API and Spark Streaming,** the model leverages big data capabilities to seamlessly handle and process **large volumes of data in real time**. This enables it to provide accurate and timely car valuations on a daily basis. We have prepared **100 distinct files containing each 23 new listings** that will be streamed into the model at the pace of **10 rows per second** to provide the price recommendations.

In order to **speed up the prediction** process, first we select a subset of columns from the processed observation based on feature selection done during the modelling phase. This **reduces the computational overhead** that the pipeline needs to perform lowering both time of inference as well as costs related to hardware.

Multiple observations can be processed at the same time thanks to Spark's distributed architecture making it the go-to solution for real-time applications.

How did we ensure data redundancy during model development?

When training models on large volumes of data, it is advisable to implement **checkpointing mechanisms**. At key stages of the process, temporary **copies of the current key variables** are created. This approach significantly **accelerates the iteration** process, enabling **faster experimentation** with new ideas and facilitating the search for optimal solutions, all while **minimizing computational resource usage**. This technique proved particularly valuable for this project, given the self-terminating nature of the compute clusters within the Databricks Community Edition, which served as the main platform for the project.

Distributed architecture challenges

Spark's distributed data storage presents certain limitations and challenges. Global operations often require data shuffling, which involves transferring data across nodes and is computationally expensive. This is particularly evident when detecting outliers using KMeans, as it needs to calculate global centroids for accurate outlier detection. To mitigate memory overload, it's essential to implement checkpointing and caching while carefully designing partitioning strategies.

Alternatively, analyzes can be conducted on individual partitions, offering faster and more efficient processing. However, this approach comes with the drawback of potentially misleading results, as it may not represent the global data accurately.