Project 2: Car Recommender System

Zarif Ali, Umer Farooqui, Yaseen Sadat, Muhammad Aneeq

April 4, 2024

Brief Problem Description and Project Question/Goal

Choosing the right car can be a daunting task for many consumers given the vast array of options available in the market; each individual has unique preferences and priorities when it comes to choosing a vehicle, with many factors like reliability, top-speed, car type, etc. each factoring in a different level of importance to the buyer. **Our project aims to simplify this decision-making process by developing a personalized car recommender system**. This system will allow users to input their preferences and weigh the importance of different car attributes according to their priorities— the goal is to provide users with a tailored list of car recommendations that best match their specified criteria to make the car selection process more efficient and user-friendly. The overarching goal of this project is to streamline the car selection process for consumers through a GUI, which would allow users to specify their preferences and prioritize different car attributes; we aim to generate customized recommendations that closely align with their individual needs and desires. For our project's data, we created our dataset, `car_data_set.csv`, which is formatted to include 10 attributes, including CAR MODEL, ENGINE, HORSEPOWER, PRICE, TORQUE, CAR TYPE, RATING, RELIABILITY, 0-60, and MAX SPEED, and hold the image path of the given car. We currently have 54 cars in the dataset we're using, but it can be expanded upon by extracting the dataset website we used, Auto Data, which contains 9 of 10 attributes, excluding reliability score. For the reliability score, we're using data from Repair Pal.

Computational Overview

Our `main.py` module runs our Pygame GUI and allows users to set preferences in two separate ways. The first is the drop-down menu which lets users set the range of attributes including horsepower, price, and torque, while the other two drop-down menus specify which car type and engine type the user is looking for. The next step of our selection process is to let users prioritize how much specific attributes like reliability, rating, max-speed and 0-60 time matter to them, and we've included sliders to numerically indicate the importance of each attribute according to their needs. While the preferences do not directly affect the decision tree, it will take the cars from the decision tree and rank them based on a pre-set algorithm in which we gather all the indicators apply a mathematical algorithm and score them. Then finally we rank them.

The result of these computations is displayed in an interactive interface, where the user is presented with the top-recommended car, complete with all its specifications, alongside the most similar car. Additionally, we provide the next four cars that closely follow the top recommendation, giving the user a range of options to consider.

In our Car Recommender System, we've developed an intuitive approach to letting users select their ideal car based on a range of attributes, including horsepower, price bracket, engine type, torque, and car type. The core of our system is a decision tree that we've constructed using the `tree.py` module, where each node in this tree represents a specific car attribute, while the leaf nodes represent individual car models. This structure enables us to efficiently navigate through the array of car models and attributes to find the best match for the user's preferences. This module closely resembles and uses many of the methods outlined in Exercise 2 to build up the decision tree. Our modifications and enhancements come from accounting for the new attributes specific to our dataset as well as implementing the `car_ranker` function. Firstly, the `encode_(attribute)` functions help convert attributes like engine type and car type into a format that can be easily processed by our program to start building the decision tree.

The `car_ranker` function is one of our most important implementations as it evaluates and ranks car models from a provided list according to user-defined preferences on metrics such as rating, reliability, acceleration (zero to sixty time), and max speed. Each preference is weighted by the user, indicating the importance of each metric. To ensure a fair comparison between cars with varying performance attributes, the function normalizes the horsepower, torque, and speed on a scale of 0 to 1 based on where they lie between the minimum and maximum values in the dataset. For acceleration, the normalization formula is inverted so that a lower zero-to-sixty time translates to a higher normalized score, aligning with the notion that faster acceleration is typically preferred. After normalization, the function calculates a performance score for each car by combining the normalized scores for each attribute, weighted according to user selection. The cars are then ranked from highest to lowest performance score, outputting a list of recommendations that best match the specified criteria.

The overall tree portion of our project works in three steps, first, we access the dataset, then we transform the data from the set into a personalized precise system, and then we build the decision tree itself using similar concepts of Exercise 2 in combination with our ranking algorithm, which ultimately allows us to rank the outcomes of the decision tree based on information provided by the user.

The `project_graphs.py` file contains a set of functions and classes that are used for the operation of the graph aspect of our project. It includes the functionality to generate a dictionary mapping each car name to its corresponding attributes, and it defines a weighted graph structure to represent the relationships between different car models based on their similarity. We're using the same classes from Exercise 4 to build up our graph system, but where we differ is our implementation of the similarity score which uses an Euclidean algorithm.

The `euc_similarity_score` function calculates the Euclidean similarity score between cars and is subsequently used to create edges in the graph. This enables us to recommend not only the best-matching car but also similar alternatives. The rationale behind using the Euclidean algorithm is its ability to quantify the similarity between various car models based on their attributes. We determine similarity by calculating the Euclidean distance between our best-matched car and the next consecutive car within our decision tree. The formula for the Euclidean distance is $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$, where x_i and y_i are the attribute values of two cars, and n is the number of attributes. A shorter distance between the best-matched car and another car indicates a higher degree of similarity.

To ensure compatibility with our graph, the `data_work.py` module plays a crucial role in pre-processing the car data, where It handles tasks such as normalization and encoding of attributes. Specifically, we normalize numerical attributes to a 0-1 scale to ensure a fair comparison between different cars, similar to how normalization works within our `car_ranker` function. We also convert categorical attributes into numerical values for use in our graph.

Our graph is structured as a complete graph, where every vertex is connected to every other vertex and the weights of the edges represent the similarity scores between cars, calculated using the Euclidean distance. In constructing a complete graph, we are able to ensure that our system can explore all possible car comparisons to generate the most relevant recommendations for the user.

Changes to Project Plan

After receiving feedback from our TA and realizing that our original project idea was not effectively utilizing the graph or tree data structures beyond just data storage, we decided to pivot our focus. Our initial plan was ambitious, aiming to investigate a broad range of educational and economic indicators across African countries. The overarching goal was to investigate trends in educational proficiency across African countries and forecast future trends over the next five years using the UNESCO database. We aimed to graph various educational metrics, including literacy rates, school enrolment ratios, and educational attainment levels to uncover patterns, disparities, and potential areas for improvement across a select number of countries within Africa. However, as we delved deeper into the project, we found it challenging to come up with a concrete plan for incorporating economic calculations in our code. The computations we had in mind for comparing certain indicators did not seem to yield any useful trend analysis.

Furthermore, our TA pointed out that our project lacked a clear computational component that leveraged the trees and graphs. As a result, we decided to shift our focus to a more manageable and computationally interesting project, which is now the Car Recommender System.

This new project allowed us to make better use of data structures like decision trees and graphs to analyze and recommend car models based on user preferences— it provided a clear framework for applying computations that made sense to us and most importantly, utilized graphs and trees in tandem to carry out our calculations to determine the best car and the consecutive similar cars, which was lacking in our original proposal. The Car Recommender System also offered a more defined scope for us to tackle and allowed us to concentrate our efforts on specific computational challenges, like implementing similarity scores obtained by our graph module and incorporating them with our decision tree.

The decision to change our project plan was driven by the desire to create a more computationally meaningful project that aligns with the learning objectives of our course; in our final iteration, we believe were able to develop a project that not only met the requirements of our assignment but also allowed us to apply computational concepts to a real-world problem that is consumer-facing.

Discussion

Our Car Recommender System aimed to simplify the car selection process for users by providing personalized recommendations based on their specific needs and how much a given attribute mattered to them. The results of our project indicate that we effectively achieve this goal. By utilizing decision trees and graphs, we were able to analyze car attributes and user preferences to recommend cars that closely match the user's desired specifications.

One of the main limitations we encountered was the availability and completeness of car datasets. While we managed to compile a dataset with a variety of car attributes, the data was not always consistent across different car models, which posed challenges in our analysis.

For further exploration, there are several avenues we could pursue, but most notably in terms of data collection. Expanding the dataset to include a wider range of car models and attributes would improve the accuracy and diversity of our recommendations, and we believe with more time, we could implement a module that could automate the data-scrapping process.

Another area of interest could be the integration of real-time data, such as current market prices and availability, to provide users with the most up-to-date information; this again would fit under a more robust data-collection process.

In conclusion, we believe our Car Recommender System demonstrates the potential of using computational techniques to assist users in making informed decisions when selecting a car. While there are areas for improvement and further exploration, the system provides a solid foundation for a more comprehensive and sophisticated car recommendation tool in the future.

References

[1] Khan Academy. (n.d.). *The Euclidean Algorithm.* Retrieved from https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm

[2] Numberphile. (2018, February 14). *The Euclidean Algorithm - Numberphile.* [Video]. YouTube. https://www.youtube.com/watch?v=AY9MnQ4x3zk

[3] Pygame. (n.d.). *Pygame Documentation.* Retrieved from https://www.pygame.org/docs/

[4] RepairPal. (n.d.). *Reliability.* Retrieved from https://repairpal.com/reliability

[5] Auto-Data.net. (n.d.). *Auto-Data.net.* Retrieved from https://www.auto-data.net/en/