DATA11001 Introduction to Data Science, autumn 2023

Technical report of a group project

€€€€€€€€€€€€€€€€€€€€€€€€€

importEVorNot®

Trust the algorithm, find the deal!

€€€€€€€€€€€€€€€€€€€€€€€€€

Janne Tuukkanen, Patrik Keinonen, Toni Rämö

29. lokak. 2023

Content

1 General description

In this chapter, we introduce the problem to be solved and a brief overview of our state-of-the-art solution. A more detailed description of the various technical aspects of the product is covered in subsequent chapters.

1.1 Problem

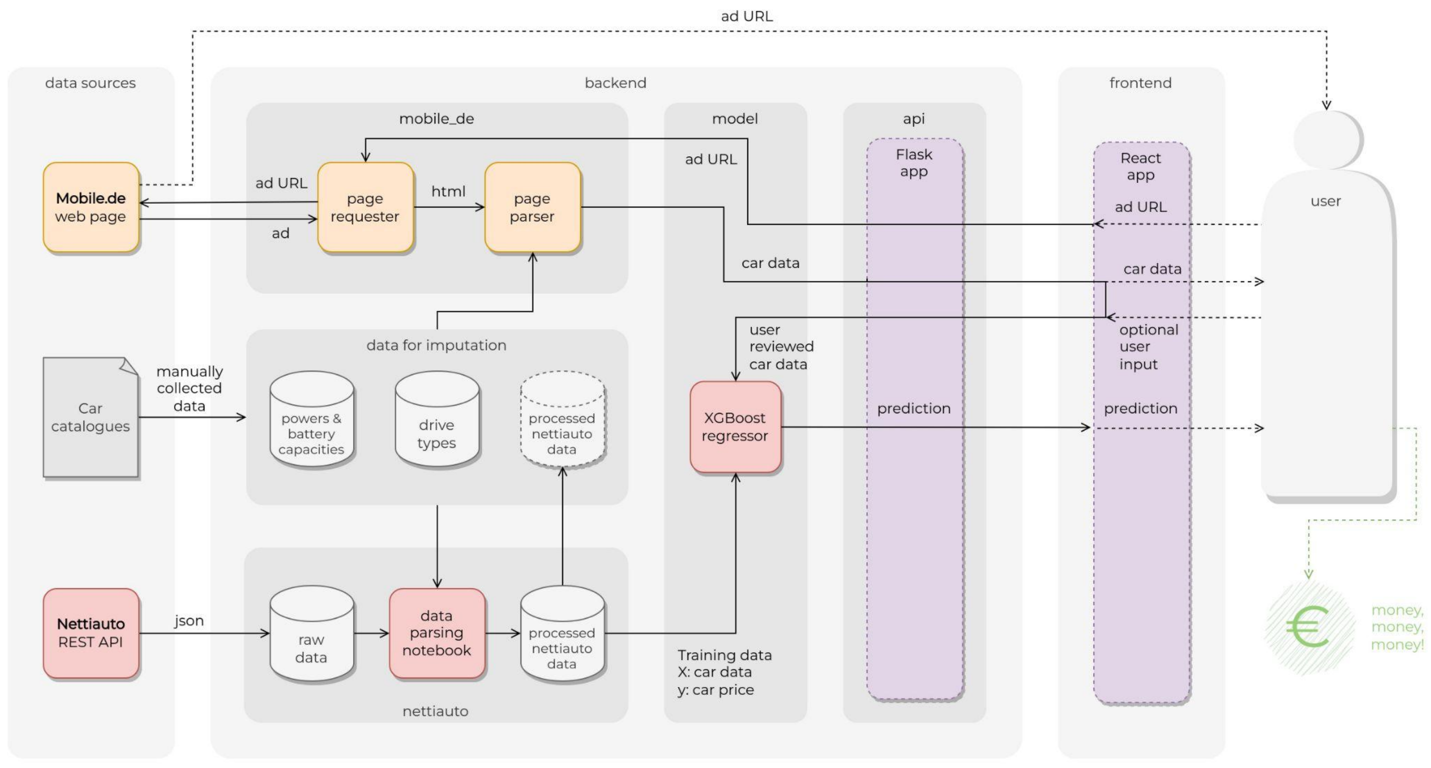
Prices of used electric vehicles (EV) vary between Finland and Germany. For this reason, there is a huge potential to make money by buying a cheaper car from the German market and then selling it with profit in the Finnish market. However, it requires extensive knowledge and manual investigation to estimate if the available deal is profitable or not.

1.2 Solution

By collecting enough actual market data on used EVs from Finland, we are able to build a regression model that can predict the expected selling price of the car in Finland. The model is made available with an interactive web application that allows users to specify car ads to be analyzed. Simple and efficient! Our application is called **importEV**orNot®. The source code of the application is available at https://github.com/jatufin/importEVorNot. and the deployed production version can be accessed via https://import-ev-or-not.fly.dev/.

1.3 Architecture of the application

importEVorNot®, like any other web application, consists of a backend and a frontend. The processing of the data and predicting takes place at the backend while a simple web user interface (UI) functions as the frontend. Additionally, the application uses multiple external sources of data. The figure below describes the architecture in more detail. The actual file names and folder structure may vary in the code base but the logic is in line with the diagram.



2 Data sources

importEVorNot® uses three main sources of data:

1. Data from **Nettiauto**, a Finnish used car marketplace (https://www.nettiauto.com/en)
2. Data from **Mobile.de** a German used car marketplace (https://www.mobile.de/?lang=en)
3. **Car catalogues** and other sources that are based on catalogue data such as EV database (https://ev-database.org/)

Car ads from Nettiauto are used as the source of training data for the regression model. After the model is trained, predictions will be made for chosen Mobile.de ads. However, both training data and prediction data may be deficient. Therefore, car catalogues are needed to help in the imputation of the data.

2.1 Obtaining data from Nettiauto

The service offers an API for data requests and registration is required. After logging in to the service and receiving a user token, this token can be used to request an API token, which is valid for one hour.

We had issues with the authentication of the data request script, so data was obtained by running API requests manually through an API testing web page. As the data is used only to train the predictive model, the lack of real-time data set is not an issue. However, for our service to run online in production, at least once a day update and retraining of the model should be implemented. In that phase of the project, the authentication problem should be solved and the process automated.

The data query process is as follows:

1. Sign-in to `Nettauto.com` as a registered user
2. Copy a cookie string: `nettix-token-user`
3. Open in a browser: `api.nettix.fi/docs/car/`
4. Authorize using the cookie string as `X-Access-Token`
5. Run desired API requests using forms on the page

As a single search result array contains 100 items at maximum, the search had to be run several times. Each time the search was run, the result page number was specified starting from one, and the received JSON file was saved in `./nettiauto/query_results` directory.

2.2 Obtaining data from Mobile.de

Acquiring data from Mobile.de, in comparison to Nettiauto, is markedly more intricate. While Nettiauto provides a REST API with an OpenAPI specification, Mobile.de is designed to be entirely server-side rendered, thus precluding any analogous shortcuts that are available in Nettiauto. To transform the car data into a functional format, Beautiful Soup and regular expressions are employed to parse the contents of the HTML document.

Mobile.de incorporates multiple tiers of security measures to inhibit automated data collection. For more information please refer to the source code.

3 Data processing

Generally, the data from both sources is somewhat incomplete. Various methods, ranging from supplementing missing details using car manufacturer catalogs to employing statistical techniques, were utilized to improve the dataset's quality.

3.1 A unified data model

The process of unifying the data model commences by comprehending the data from both sources and identifying their commonalities. Exploratory analysis resulted in a data model that encompasses basic car information along with a selection of additional features that influence the asking price of a vehicle.

3.2 Nettiauto data processing

The process of adapting the Nettiauto data to the unified data model is relatively straightforward: one must simply extract the basic information directly and convert the additional features into boolean key-value pairs.

However, the data cleansing and imputation phases for the Nettiauto dataset are more intricate. Notably, the attributes relating to battery capacities and engine power are missing in nearly 50% of the entries. Through the application of regular expression (regex) patterns on unstructured text fields and model descriptions, we managed to discern battery capacities, effectively reducing the deficit of such information to below 25%. A challenge surfaces in the heterogeneity of the battery capacity and power data, with certain entries indicating net capacity, some denoting gross capacity, and a few being patently erroneous. To tackle this variability, we employed regex patterns tailored to detect specific model variants. By doing so, the power and battery capacity details were systematically corrected using manufacturer catalog references. This rigorous methodology ensured data consistency for over 95% of the entries. Entries that couldn't be rectified, accounting for the residual 5%, were systematically discarded from the dataset.

3.3 Mobile.de data processing

The data from Mobile.de is extracted from HTML documents utilizing both Beautiful Soup and regular expressions. Parsing the HTML is a nuanced task; for a more granular understanding of this procedure, we advise the reader to consult the source code.

Mapping the majority of base information onto the unified data model is rather direct. However, certain fields necessitated inferential approaches or extraction from additional features. Given that the terminology used in these extra features differs from that in Nettiauto, we introduced a mapping table to bridge the disparity.

In the advertisements from Mobile.de, the engine power data appears to be accurately presented in most instances. Conversely, Mobile.de does not provide a specific field for battery capacity, and this information is notably absent in the unstructured textual descriptions of a vast majority of the ads. To impute the missing battery capacities, we leveraged the catalogue mapping technique used with the Nettiauto dataset.

Moreover, the Mobile.de dataset omits a specific field for the drive type. Our analysis of the supplementary features revealed the presence of “four-wheel drive.” Using this information, we formulated heuristics to deduce the drive type, drawing insights from the Nettiauto dataset.

4 Model

Prediction model we used is a gradient boosting algorithm implemented in the XGBoost library. As other gradient boosting methods, XGBoost (eXtreme Gradient Boost) uses ensembles of decision trees to produce predictions and is easily applied to almost any data set. We used a configuration, which builds random forests for gradient boost estimators. The XGBRegressor model was set to build 100 estimators in the ensemble model (`n_estimators=100,` this is the same argument as `num_boost_round`) and use a 5 tree random forest for each estimator (`num_random_tree=5`). Setting both of these values to greater than one sets the model to use a combination of gradient boosting and random forest. Maximum depth of the trees was set to 8.

4.1 Categorical values

Before training or prediction, categorical values of the input vectors are converted to exclusive boolean values. For instance car manufacturer string values (Volkswagen, Toyota, etc) are converted to new columns, (make_volkswagen, make_toyota, etc) where only the corresponding column value is set to TRUE, all others remaining FALSE. This is done with the `get_dummies` method of the Pandas library.

4.2 Training the model

The dataset was split to training and testing sets with the `model_selection.train_test_split` method of the `sklearn` library. The size of the test set was 20% of the total dataset.
Training and testing was run once with this dataset split.

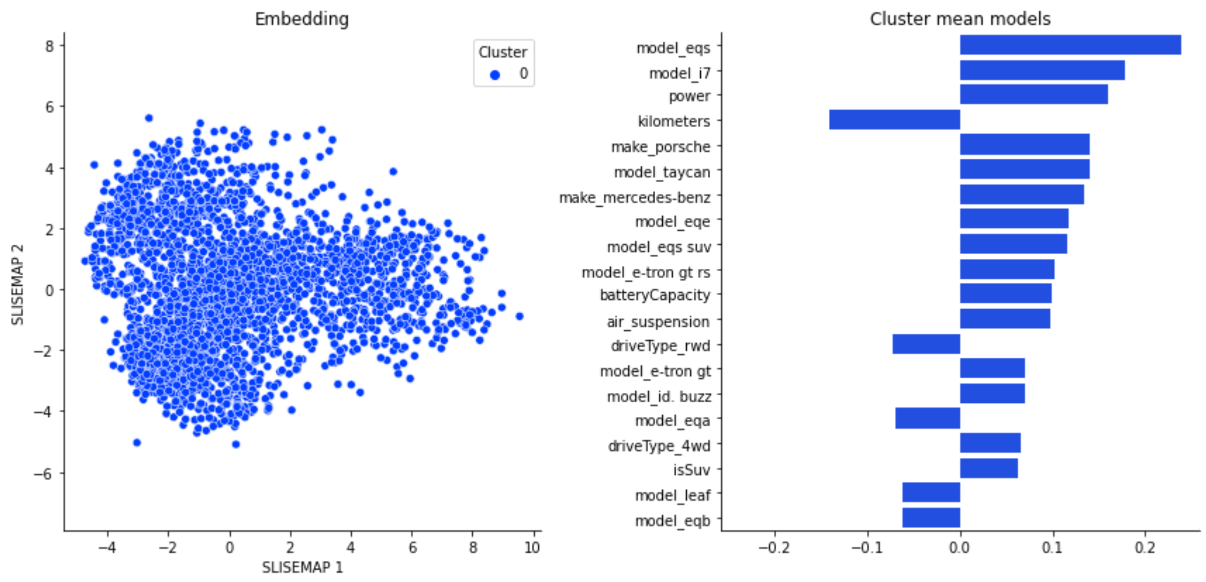
4.3 Accuracy of the model

The R^2 score for the test set was 0.958.

4.4 Interpreting the model

We performed an exploratory analysis of the model using an explainable AI method called `SLISEMAP` (see https://github.com/edahelsinki/slisemap). It builds local proxy models of the studied black box model and forms a chosen number of clusters by clustering similar local models into one. Interestingly, despite the number of clusters, the result includes only one cluster. We presume that this indicates that all of the derived local models are similar. This, in turn, suggests that a linear model could be used as a competitive alternative.

The result of the analysis is summarized in the Figure below. Analysis was made with normalized data. The left side of the figure shows derived clusters (now only one) while the right side shows top-20 the most impactful coefficients of the linear model for each cluster. To sum it up, certain models and manufacturers alone can significantly boost the expected prices. Furthermore, as expected, features that are important for EVs such as power and battery capacity are the main factors explaining the predicted price. The actual weights of the features vary depending the used hyperparameters. This run was made with defaults.



5 User interaction

Interaction with the application is simple. First, the user is expected to have chosen an ad for a used EV from Mobile.de. Then, she will paste its URL to a dedicated field (which has a default URL) in the UI of the **importEV**orNot® app and press “fetch”. This initiates a request of the page by the backend which processes the received content as described in Chapter 3.3. The user is shown the retrieved data. This allows her to ensure that all entries are valid and to adjust certain attributes such as mileage or included accessories if desired.

After validating the data, the user is expected to click “get prediction” which feeds the shown car data to the model that provides a predicted price in the Finnish market to the user. Additionally, UI shows a profit/loss statement and informs the user if the deal is a potential moneymaker or something to avoid. Since the importation of vehicles from Germany to Finland involves other expenses than just the purchase alone, we have included a modifiable field to include those into the analysis. The default value of 3000€ is used for other expenses.

6 Reflection

Overall, the project turned out to be much in line with the project canvas that was designed at the beginning of the project. This indicates that the initial idea was decent and the planning sufficient. On the other hand, the goal was ambitious, and perhaps more time was spent on the development than anticipated by the course staff. Namely, most of the effort went into processing, mapping, and validating the data. For instance, searching car model-specific data for imputation required tedious labor, and even more time could have been spent on that since not all configurations are yet covered. Additionally, retrieving the data from both Mobile.de and Nettiauto caused an unexpected delay due to scraping protections and uncooperative API, respectively. Then again, we remained determined and never let setbacks discourage us. Instead, those made us stubborn to find workarounds.

Initially, we had two alternative ideas about the prediction phase: make a prediction of a particular EV by request or provide predictions of a batch of EVs and indicate the most profitable ones to the user. Obviously, the latter would have required more development time and would not have been feasible to implement with the combination of used methods and available resources. Collections of Mobile.de ads would have been difficult to maintain up to date. For these reasons, by following the idea of a minimum viable product, we decided to go with the first option, which turned out to be a wise decision.

We worked mostly remotely and naturally part-time due to other ongoing courses, which presumably hindered the efficiency of the development somewhat. On the other hand, we were able to adapt to these restrictions and find fruitful ways of working and communicating such as performing remote pair-coding, messaging, and having ad hoc huddles via Slack. Weekly meetings happened in person, though.

APPENDIX

Known issues

- For some reason, the application is not able to fetch a sponsored ad of Mobile.de. Such ads are usually on top of the search results. One should avoid using these.
- The application is not able to fetch suggested ads of Mobile.de (at the bottom of the ad). This is due to missing "Id" field in the URL.
- Issue with obtaining data via Nettiauto API using request script (see Chapter 2.1)
- Accuracy of predictions varies between brands and models considerably.
- /schema returns encoded columns used by the model instead of columns used by the API.
- CORS should be restricted to the frontend server.

Improvement ideas

- Improve the training data set and data used for imputation: The larger the set, the better coverage of each model. Higher coverage of imputation data, better predictions.
- Batch analysis of Mobile.de ads and finder of the most profitable deals.
- Regular updations of the training data and the model.
- Monitoring and including the volatility of the ads at Nettiauto as a feature.