Abstract

I built an automotive price prediction model that takes in data about a used car and estimates its local market value. I designed a model that takes in data from nationwide and local car dealers to generate the price. The software uses multiple regression analysis to generate a statistical model that predicts auto prices, and through smart caching it performs this prediction in realtime. This can help to solve a real-world problem: people will be able to use an open-source price prediction model to find out a car’s price instead of being surprised by a car dealer or a less granular price prediction model like the Kelly Blue Book.

The project also includes a user interface to browse through the database of cars. The UI has been engineered with a modern client-side software engineering stack, using open-source tools and frameworks including Backbone, Underscore, D3.js, and others. The final piece of the project is an interactive infographic that generates a map of auto prices across different geo-locations.

The project should be accessible from http://CPSC490.com:3000, or http://107.170.9.167:3000/

Deliverables

1. A working price prediction engine and user interface that allows the user to input used car data and outputs an accurate valuation based on live dealership pricing data.
2. A written report covering market insights found by the price prediction.
3. An additional user interface that allows the user to browse, filter, and sort through the entire corpus of car listing data, with all view logic performed in the web browser.

Table of Contents

Abstract
   Deliverables
   Table of Contents
1. Prediction Engine
   Multiple Regression Example - Ford Fusion#
2. Data Insights Infographic
3. Data Browser
   Software Architecture
1. Prediction Engine

The prediction engine takes in a set of data about a used car and compares it with the corpus of dealer listings in the MongoDB database. We perform a multiple regression to create a price prediction for the given car. Since the computational cost of a multiple regression for the entire database is high, we pre-compute the prediction models with Stata.

To demonstrate the efficacy of pre-computing these statistical prediction models, we perform this method on five of the most popular car models, including the Ford Fusion.

Multiple Regression Example - Ford Fusion

```
. regress price mileage age trim1 trim2 trim3

Source | SS      | df | MS      | Number of obs = 21459
Model  | 5.9693e+11| 5  | 1.1939e+11 | F( 5, 21453) =14233.61
Residual | 1.7994e+11| 21453 | 8387672.42   | Prob > F = 0.0000
         | 7.7688e+11| 21458 | 36204452.6   | R-squared = 0.7684
         |            |       |            | Adj R-squared = 0.7683
         |            |       |            | Root MSE = 2896.1

price | Coef. | Std. Err. | t     | P>|t|  | [95% Conf. Interval] |
-------|-------|-----------|-------|------|-----------------------|
mileage | -.0751323 | .0010087 | -74.48 | 0.000 | -.0771095 | -.0731551 |
age | -1469.847 | 18.2228 | -80.67 | 0.000 | -1505.56 | -1434.134 |
trim1 | -478.3864 | 53.23563 | -8.99 | 0.000 | -582.7322 | -374.0406 |
trim2 | -1010.765 | 59.88343 | -16.88 | 0.000 | -1128.141 | -893.387 |
trim3 | 7454.407 | 112.4833 | 66.32 | 0.000 | 7234.088 | 7674.726 |
_cons | 25081.66 | 54.7411 | 458.19 | 0.000 | 24974.36 | 25188.96 |
```

The regression for the Ford Fusion (as well as that of other auto models) is pre-computed on the server and the price prediction model is served in static javascript. This method, although simple, is highly effective and efficient, particularly at web scale. To prevent the same query from being wastefully re-calculated with each user request, the price prediction mechanism is cached on the frontend and is refreshed when new data is added to the database. The “refresh” of the price prediction model could be implemented through a cron job or a trigger from MongoDB.

---

1 We omit geolocation for simplicity
The static model is stored in JSON:

```javascript
var model = {
    mileage: -.07513,
    age: 1469.847,
    trim: [-478.386, -1010.765, 7454.407],
    cons: 25081.66
}
```

When the user inserts his automobile’s information into the prediction model, it simply takes in this data and applies it to the statistical model generated by the regression. The resulting price prediction is representative of the dealer data in the AutoList dataset.

Ideally, this project would have pre-computed prediction models for all the cars in the database. This demo serves as a proof-of-concept for the more general case.

2. Data Insights Infographic

This infographic shows the differences in dealer prices for new cars in different states. In many cases, the delta in sticker price is upwards of 25%. The design of this portion of the project was
inspired by Matthew Isabel's open source infographic on mortality rates (credited in the source). This takes a look at five models of automobile - the Ford Focus, Ford F-150, Ford Escape, Honda Accord, and Honda Civic. These specific models were chosen because they were the most common models in the dataset provided by AutoList\textsuperscript{2}.

Building upon Isabel's javascript, the map is drawn with HTML5 Canvas using the D3.js library. Using JSON data representing the lines that draw the states, and manipulating those lines with HTML, SVG and CSS, I made a map that shows the optimal place to buy a car of a given model.

3. Data Browser

The project also includes a user interface for browsing through the corpus of listings (over 200,000 are in the database). The left search bar allows for fine granularity queries, allowing the user to specify price and year ranges, makes, models, mileages, and states. The listing results are loaded dynamically, and the UI auto-loads additional listings when the user scrolls to the bottom of the page (a sleeker alternative to pagination).

The UI depends on Backbone.js as an MVC framework and Twitter Bootstrap as a CSS framework. The backend is a Node.js server with a MongoDB database. The database is a document-based datastore with indices on the `make` and `model` of each listing.

\textsuperscript{2} Many popular models of car were not in the provided dataset, or had relatively few compared to the U.S. auto population. This may be an inherent selection bias of the AutoList inventory.
Software Architecture

The software architecture was designed to decouple user interaction design from data retrieval - the javascript MVC model handles any dynamic UI and the JSON api is queried from the client to the server.

To handle the UI, I integrate Underscore.js templates with jQuery and Backbone.js. jQuery logic executes the calls to the API when the user scrolls to the bottom of the page or performs a search query. There is no “search button”; rather, the queries are made automatically when the user acts, like Google Instant.

4. Addendum

The entire development environment is packaged in a portable unit with Vagrant. By following the instructions in the README, you can deploy this entire application to your own server (or on your local machine) in a matter of minutes.

---

3 I choose Underscore.js template rather than React.js (the UI framework initially specified in my proposal) since Underscore templates were built to interface with Backbone.js.

4 One downside to using MongoDB to host over 200k documents in its database is that rapidly iterated queries may become slow on some servers. Depending on the hosting provider (and it’s cost / RAM / etc.), query results can arrive either instantaneously or may take multiple seconds.