CS535 Big Data Term Project

Team Project Final Submission

Team members: Abdul Matin, Peng Jiang, and Glade Snyder

The Project Title: Competitive Craig List Pricing

Problem Formulation

When trying to sell a car sometimes knowing what the competition in the market is more important than car value estimation. The point of this program is to help one decide how much they can sell their car for rather than how much their car value is. Each market will be independent to how much you can sell the car for compared to others. Also, when there is a lot of overpriced cars selling you will not undersell your car because you will be looking at what prices of cars are being listed at. This is a common approach in selling homes and in real estate. For people want to buy a car, this works for them the same as people who try to sell.

This program also makes sure that people do not pay overpriced cars. Users can also use this program to see if there are better deals on cars that they are not aware of that are currently in the market. Even if the price of a car is below the market value decided by Kelly Blue Book it is irrelevant if you know of another priced car is lower than the one you are looking at. This program helps to give a pulse of what the current very flamboyant market looks like.

This is interesting as a big data problem because we are approaching the problem to be tailored to anyone that wants to sell their car on craigslist. It is to a specific submarket where more for sale by owners will be. While Kelly Blue Book and NADA we assume uses more global data in figuring out the value of a car. It is also interesting because we will be using a neural network to figure out the price of the cars. That makes it exciting and interesting especially in seeing how different architectures of neural networks give different results. We feel that we can learn a lot about neural network different architectures that can be best for figuring out the price of the car. We feel that this can be a great learning experience in getting used to neural network inner workings and get exposed to using the distributed machine learning via PyTorch on the lab machines at school. We also were exposed to Spark MLib and how the infrastructure would work on training a neural network. We quickly gravitated towards PyTorch instead of Spark due to how much easier it was for us to bring up and get going.

On the users' side, we want to make a Python application for users that take inputs (features of the car) and generate the outputs (car pricing) automatically. For simplicity, the users only need to open the app, input whatever features their cars have and the app will just generate the results that the users are look for.

This Program that we have created can be used by anyone who desires to sell their own car. They can use this created tool to get an idea about how the market is in its current state. Also, it is a quite widespread practice for people to check online and see how much value their car has. Those users could also use it to get an idea about how much their car is worth in the market currently. Another third group that is often looked over are those that can grab this program and use it to train their own neural network that is better or increase the dataset to include more than just craigslist data but also other programs. They would need to simply create the Api to access the data and then manipulate the data to match the same format as we have in our pandas dataframe.

Strategies to solve the problem.

Preprocessing

The starting problem or challenge for every machine learning task is understanding the dataset that we are going to be using, trying to understand, manipulate and data mine. Understanding the important data points of a car is not too difficult to understand as we are all familiar with cars. We all know that odometer reading, make, model, year, and title status are particularly important for the price of a car. The main beginning challenge was not understanding the data, but it was cleaning up the data to be in a useable state.

The data coming from Craigslist had its own set of challenges. One of the main challenges was that each user filled in attributes to the car they were selling differently. They also tended to leave important data points out like the make, model, odometer reading and title status out of the data. While processing the data I even saw cars with a price of 0 or null which means it was not filled out. We also saw in various places where the price was set to be a word and thus we need to filter out all data points that couldn't be converted into a string. As you can imagine with thousands of user's inputs, we saw all sorts of discrepancies and corner conditions that needed to be handled as part of the preprocessing of the data. Handling of these exceptions took quite a bit of trial and error.

After cleaning the data to be better suited for our Neural Network, we discovered that many of the inputs like make and model for our neural network had strings as their value. Strings are not the best for being processed by a neural network. So, when processing the data, we had to map all the string inputs to enumerated values. Since there could be discrepancies between data inputs that were the same per example: 4 cylinder and four cylinder we had to try and catch all those same values and map them to one value. An example of this could be makes Toyota could map to 1, Chevy could map to 2, etc. Whenever there were null values where an input was not provided, they were mapped to 0 always. Each input like manufacturer, model had its own map that was saved for keeping track of to which enumeration each attribute was mapped to. Each one of these maps was then stored in a top level look up table / map to help facilitate finding an attribute's correlating enumeration map.

Things that we wanted to learn and see if other benign things like paint color, transmission, drive and other important but not critical things really impacted the prices or not. To facilitate experimentation, we created our script to be able to receive inputs from user to choose which attributes they wanted to use as part of the inputs of their Neural Network. A user could use any attribute of the many provided ones to train and create a neural network. We would grab all the corresponding attributes and load up the data into a pandas dataframe that we would then clean using the above techniques. Having this capability helped us to really adapt the neural network and experiment.

For a Neural Network to be effectively trained it needs to be fed random data and be partitioned to provide training and test data. We created a capability to shuffle all the pandas dataframes and to partition the data based upon fractions that we provided. We then used the test data to help determine if we were overtraining the data. This would lead to a growing RMSE on the test data.

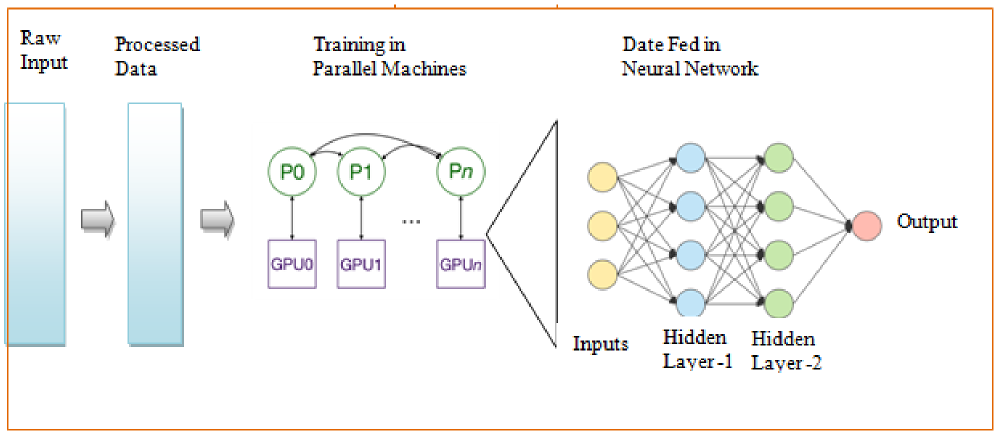


Fig. 1: The model diagram of our proposed work

Recap on Strategies Used

- **Data pre-processing and explorative analytics**
  - We use Python as programming language.
  - We use Pandas DataFrame from Python to take care of the data we want to process.
  - Our dataset is in an csv file, and it is already rearranged in a big table. We can simply insert the whole csv file into a DataFrame and clean the DataFrame by
    - Removed entries that did not have critical values like odometer readings and model year.
    - removed ones that had a price of $0. They were treated as anomalies and all of us know that it is not free.

        - Needed to find and consolidate inputs that were different in spelling but the same value Ej “4 cylinder”, “four cylinders”, and “four cylinder” needed to all be treated the same.
        - Went through and enumerated all the different inputs that were strings to help neural network to train more easily.
        - Kept track and saved the enumeration translation map for converting from string to enumerated values or vice versa.
    - Also partition the data into three sets of data. These are completely randomized, shuffled, and partitioned according to the partition fractions that we decide is the most effective for the training of our neural network.
        - Training inputs and outputs
        - Validation inputs and outputs
        - Testing inputs and outputs
    - The best neural networks for training 26k inputs are [50, 50, 50, 50, 50, 50] with 6000000 epochs, which takes about one and half hour to train.
- **Deep learning model application**
    - We decided not to use Spark and MLlib for training the Neural Network due to how much trouble we were having in getting it to work.
    - We decided to go with Pytorch with different implementations. Pytorch running on local CPU (AMD Ryzen 9 5900X), distributed Pytorch running on multiple lab machines’ CPU and Pytorch running on local GPU (NVIDIA RTX 3060).
    - We will be using many different shapes and types of neural networks to find out which one gives the best pricing results by using RMSE. When we find the neural network that does the best, we will save it using the pickle library mentioned above.
    - We will supply the capability for retraining the neural network and saving it to a different pickle file and letting the user decide which saved neural network to use.
- **Python application**
    - We will construct a Python file to build an interface or application to deal with user input and display output. The users will be able to input their car year, the make, the manufacturer, the model, the condition, the number of cylinders, fuel type, odometer mile reading, title status, transmission type, the driver type, paint color. We will pick some of the variables that are mandate for users’ input. For example, the year, the manufacturer, the model, title status and odometer mile reading. These are crucial to figure out a car’s price. Things like color, transmission type etc. We don’t believe will be particularly important for the price of the car.
    - By loading the Neural Network states (.pickle file), we will be able to use the training results we obtained from the Pytorch trainings that we performed. We can generate a curve for the results and apply the inputs that users gave. We look for the corresponding price from the inputs that users gave and the training results we just loaded. We will do a weighted balance on the results we are getting for each input we received. As mentioned, there are some features of cars that do not contribute a lot into pricing. We will find the balance there to make sure our final results are reliable.
    - Then, we get the results from it (predicted price) and display it to users for reference.

- **Frameworks**
  - Pandas: to preprocess our data
  - Apache Spark: for cluster management during distributed neural network learning
  - MLlib: to use the built-in neural network model for training up the data.
  - Pytorch:
  - Argparse: will be used for parsing command line arguments to dynamically set up our neural networks inputs, outputs, and architecture of the neural network by how many layers and hidden units are used.

- **Dataset**
  - We will be loading a csv file with the following categories.
  - The year, the manufacturer, the model, the condition, the number cylinders, fuel type, odometer mile reading, title status, transmission type, the drive type, paint color and listed price
  - We will be fine tuning and pruning which variables we use to get an accurate neural network.
  - We will be trying to predict price from the dataset.
  - We are using the used car dataset from Kaggle [1].

Training Experimentation

Spark MLlib: Based on professor and Laksheen's suggestions, we are giving up on using Spark MLlib setup since MLlib has very limited APIs for neural networks and it is just not suggested to use Spark for neural networks generally than Pytorch or Tensorflow. Since Pytorch is a better library supporting distributed data in parallel computing mode. Our team decided to use Pytorch instead.

Pytorch running on CPU: We used the neural network code in Pytorch from CS545 we all have taken before. And we are able to make it training on multiple dimension inputs work. We used a small portion of dataset (about 420 inputs, each input has 12 parameters we need to consider) for testing our software. It works perfect without any issues. We are able to get good results by running 30000 epochs with hidden list = [50, 50, 50, 50, 50, 50] in just a few minutes. We have it tested with user inputs, and it is giving fairly correct results. Of course, we are not done yet. We have created a bigger dataset for further testing, and it seems like it's running forever before getting the model completed. This is the point where we are investigating distributed Pytorch so that we could run our model in 10 different machines at the same time.

Distributed Pytorch running on CPU: We have used distributed Pytorch framework for the parallel execution of our experiments. Unfortunately, we couldn't use the CS120 lab machine's GPU due compatibility issues. That's why we had to run our experiments on the CPU of several lab machines. We have noticed that parallel execution on CPUs didn't provide enough speed in training. So, we have experimented our model for shorter version of dataset which includes around 4000 inputs. We can get expected level of prediction accuracy by running around 20000 epochs with hidden list = [30,30] in nearly half of hour.

Pytorch running on GPU: It looks like running on CPU and multiple lab machines is not fast enough to train our dataset. We do need a faster way in order to train the original dataset which is about 400k inputs in total. Since lab machines do not support running on GPU, we were changing the code we have for Pytorch running on CPU and make everything passed to GPU. The results are shockingly good. We are able to run 30000 epochs with list = [50, 50, 50, 50, 50, 50] and 26000 inputs in about a minute. Running on the original file, which has about 400k inputs takes about 1 hour. But the RMSE looks fine (around 1000, sometimes lower), so it takes much more time to get better than that.

What did we learn about the data? There are a couple of input parameters that might not play an important role in training the model. For example, pain color, fuel, transmission and drive. Some parameters like the year, manufacturer, model and odometer plays very important role. Changing in one of these parameters might change the predicted price a lot.

Result Analysis

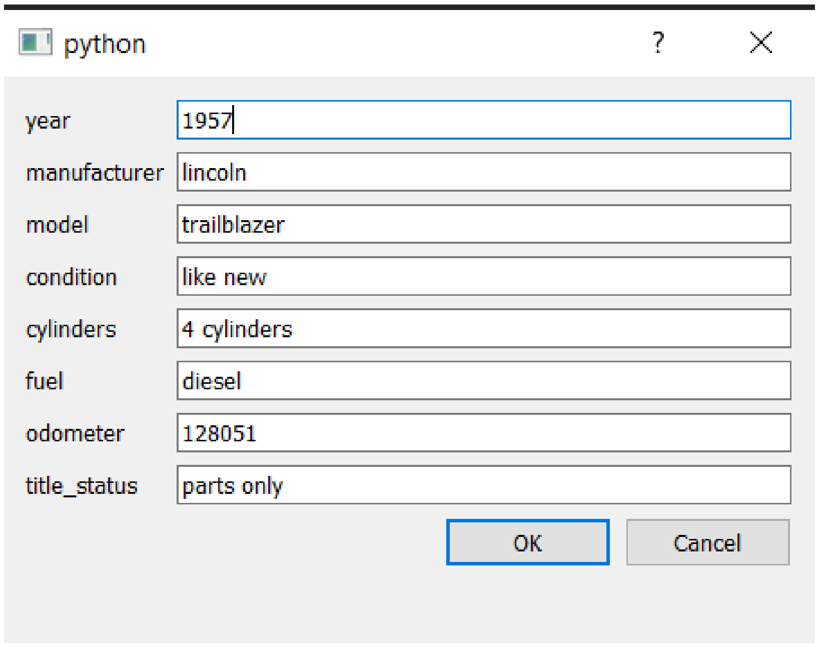
Usually, the more epochs we train, the smaller the RMSE will be, which is nicer since the error bound is lower. But we get CUDA errors if we train it too many epochs for some reason. Distributed Pytorch is too slow. Finally, we are able to train 26k inputs within an hour and get around 800 RMSE.

How was the RMSE? How well did our neural networks train?

At first when we are training with the smallest portion of the dataset, it is pretty easy to get small RMSE around 200. As we are training larger and larger portion of the dataset, the RMSE gets bigger and bigger, even we we're trying to increase the number of epochs, number of hidden units and number of hidden layers. It takes way too much time to get small RMSE for big datasets. Finally, for dataset with 26k inputs, we got around 870 RMSE after almost an hour of training on GPU. In most cases, we get CUDA error if we train it for more time.

Also, we used a lot of visual inspection on figuring out whether the prices we get make sense with the data we have been using. This will require us to parse out all similar comparable cars and see if the price looks right by averaging all the comparable just as you would do with a home appraisal. As results, the predictions are ideal and close to what we think.

Also, as part of the testing we tested the Graphical Interface that allowed the user to input variables to predict the desired price of the car that they want to put in. The User interface would only allow the user to give inputs that are or were used in the neural network. One problem or challenge with the user interface is that it fills out randomly the data fields and doesn't keep track of which models belong to which manufacturers. So, in other words, we can have a manufacturer of Ford and a mode of Forester which is an invalid combination. We didn't have time to fine tune the data that is automatically filled out inside of the user interface. The user interface will also only load up the variables that the neural network was trained on so that you have a one-to-one mapping and thus facilitating a more accurate price prediction. One can also fill out null values and they will map to zeros. We feel like the User interface is sufficient. But as for future investigation, we could make it a dropdown menu version that users can only select valid values.



year	1957
manufacturer	lincoln
model	trailblazer
condition	like new
cylinders	4 cylinders
fuel	diesel
odometer	128051
title_status	parts only

Team Contributions

We had an extremely contributive team where everyone was willing to meet often and contribute to our success.

	Abdul Matin	**Peng Jiang**	**Glade Snyder**
Preprocessing of Data			X
PySpark Trial		X	X
PyTorch Distributed Architecture	X		
Neural Network Infrastructure	X	X	X

NNET experimentation and training	X	X	
Price Prediction and Load Saved Neural Network	X	X	X

Bibliography

[1] Used Car dataset for training our neural network, available on "https://www.kaggle.com/austinreese/craigslist-carstrucks-data"