Used Cars Price Prediction in Parallelism

Quanwei Hao, Ruixiao Duan, Wangbo Jia

I. Introduction

CARS are of vital importance in US families. A number of families choose to purchase used cars, therefore it is inevitable to have a platform which allows users to get the estimated price which is more accurate to price in the car selling market. Thus we are going to implement a model predicting price given a car's features and characteristics.

Since many modern datasets have grown drastically in both data volume and data dimensionality. Large data volume and high data dimensionality bring both resource and computational challenges to machine learning algorithms. For example, a social networking site such as Facebook consists of tens of millions of records, each is composed of hundreds of attributes. Similar situation in this work, the number of used cars are in millions and it is really a huge project to process such a big data, so we evaluate approaches to scale up our model in a big-data setting.

In the platform aspect, we use Spark [1], which promotes the efficiency of iterative algorithms. In the parallel algorithm aspect, we pick stochastic gradient descent, which can obtain optimal generalization guarantees with a small number of passes over the data. The algorithm can easily be parallelized on Spark, and it can also be used in an online setting, at which a data instance is seen only once in a streaming fashion. In this paper, we will discuss implementation of the model but more on parallel computation.

In advance, we would like to list what we finished in this project.

- Took overview and analyzed row dataset.
- Preprocessed row dataset includes text preprocessing, binarization and normalization in parallel.
- Training linear regression model in `pyspark.ml` package.
- Tuned hyperparameters to improve the performance of model.
- Evaluated model by k folds cross-validation.
- Supposed some advice to boost model.

II. Related work

Generally, the problem is to train a prediction model. There are multiple methods for prediction. Stock prediction is a popular topic and lots of models . Linear regression [2], logistic regression [3] and neural network [4] models have already show their ability to solve this problem. Besides, some algorithms are applied in prediction problem, like K-Nearest Neighbor (KNN) [5] and Support Vector Machine (SVM) [6].

On the other hand, Hadoop and Spark are both efficient platform helping with parallel computing. These two distributed platforms have their own advantages in the implementation of machine learning algorithms. The Hadoop [7] platform allows for distributed processing of large datasets across clusters of computers using simple programming models. It utilizes MapReduce [8] as its computational paradigm, which is easily parallelized. Additionally, Hadoop provides a Distributed File System (HDFS).

Spark is a large-scale data processing engine that aims to make data analysis fast. It supports in-memory, fast, expressive cluster computing. It can both improves efficiency through in-memory computing primitives and general computing graphs and usability through rich APIs in Java, Scala and Python and interactive shell. A job can load data into memory and query it repeatedly by creating and caching resilient distributed datasets (RDDs). Moreover, RDDs achieve fault tolerance through lineage: information about how RDDs are derived from other RDDs is stored in reliable storage, thus making RDDs easy to be rebuilt if a certain partition is lost. Spark can execute up to be two orders faster than Hadoop for iterative algorithms.

III. Project Perspectives

A. Dataset Overview

Before talking about perspectives, let's take a quick overview on dataset. The test dataset has more than 6000 records each with 12 features including brand, model, location where the car is available, year of the model, total kilometers driven in the car, fuel type, owner type, engine, power, price of new car, etc and price which we want to predict in Fig. 1.

Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

Fig. 1: Samples of the dataset. For each sample, there are 12 features and 1 label. Name, Location, Year, Fuel_Type, Transmission and Owner_Type are categorical, while Kilometers_Driven, Mileage, Engine, Power, Seats and New_Price are numerical.

	Price
count	6019.000000
mean	9.479468
std	11.187917
min	0.440000
25%	3.500000
50%	5.640000
75%	9.950000
max	160.000000

Fig. 2: The distribution of true label Price. There are totally 6019 counts, and the mean value is 9.48, while standard deviation is 11.19. In the other hand, all values fall in the range $[0.44, 160]$.

The true label in this problem is feature Prices. Fig. 2 shows the distribution of Prices. There are totally 6019 counts, and the mean value is 9.48, while standard deviation is 11.19. In the other hand, all values fall in the range $[0.44, 160]$. We will use this information to evaluate our model.

B. Linear regression with parallelism

Since we are supposed to focus on parallel computing rather than hyperparameters tuning, we are going to implement this prediction in linear regression. In training process, parallelism with Spark can be applied to calculate objective function such as Mean Square Error (MSE) in each iteration. In validation process, prediction can be done in parallel to evaluate our model.

Remember the features in dataset. Some are numbers, which can directly used as a value of one dimension. Other are text, also called natural language that can not used as a value. Thus, a natural language processing is needed.

Natural Language Toolkit (NLTK) can be used in Spark which is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength Nature Language Processing (NLP) libraries, and an active discussion forum.

IV. Dataset Analysis and Preprocessing

Before going through the prediction model, we would like to look at some details of dataset shown in Tab. I. When we count the samples of data, there are totally 6019 samples in entire dataset. However, there are some null-objects in multiple cells, which are not friendly to our training and testing. Besides, some features in the dataset are not numerical, which needs some preprocessing before sending them to model.

Feature	Details
Name	6019 non-null object
Location	6019 non-null object
Year	6019 non-null int64
Kilometers_Driven	6019 non-null int64
Fuel_Type	6019 non-null object
Transmission	6019 non-null object
Owner_Type	6019 non-null object
Mileage	6017 non-null object
Engine	5983 non-null object
Power	5983 non-null object
Seats	5977 non-null float64
New_Price	824 non-null object
Price	6019 non-null float64

TABLE I: Details of entire dataset.

A. Dataset Clean

Remember that there are totally 6019 samples in the dataset, however, the number of non-null object for every feature is not always 6019. Some are a little less, and others are really small in hundreds. Thus, for these features, it is necessary to apply different strategies to do with. We drop New_Price feature, and then drop samples with null value in any other features. After this preprocessing, there are 5975 complete data samples with 11 features left. We will cover these drops in the following.

1) Features Drop: Now take a look at feature New_Price, there are only 824 non-null object in this column, in other words, for the rest samples, we have no idea about what they are roughly. If we drop samples with null in New_Price, we will loss more than 85% samples. This is really a bad idea. Thus, considered there are totally 12 features in this dataset, we would like to drop New_Price feature. Although it is obvious to know that the price of used cars depends on the new price. But here we can not find a better way.

2) Samples Drop: For the features that are not completely filled with non-null value, Mileage, Engine, Power and Seats, since there are only less than 1% samples missing, we would like to drop the incomplete samples.

B. Categorical Features

Among all features, Name, Fuel_Type, Transmission and Owner_Type are categorical. Normally, it is easy to come up with binarization to process categorical features. This strategy still works for most features, but for those categorical features with thousands of different value, binarization looks not good since it expands dimension of features to number of different values. Sometimes it increases cost to train such a high dimensional dataset.

1) Name: For feature Name, there are totally 1855 different values. If we apply binarization for this feature, the dimension of dataset will increase to nearly 2000, as we explained above, it is really a bad idea. Here, we would like to introduce a technology called **word2Vec** in *pyspark.ml* package.

Word2Vec computes distributed vector representation of words. The main advantage of the distributed representations is that similar words are close in the vector space, which makes generalization to novel patterns easier and model estimation more robust. Distributed vector representation is shown to be useful in many natural language processing applications such as named entity recognition, disambiguation, parsing, tagging and machine translation.

In implementation of **Word2Vec**, skip-gram model is used. The training objective of skip-gram is to learn word vector representations that are good at predicting its context in the same sentence. Mathematically, given a sequence of training words w_1, w_2, $\ldots$, w_T, the objective of the skip-gram model is to maximize the average log-likelihood

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j=-k}^{j=k}\log p(w_{t+j}|w_i), \tag{1}$$

where k is the size of the training window.

In the skip-gram model, every word w is associated with two vectors $\vec{u_w}$ and $\vec{v_w}$ which are vector representations of w as word and context respectively. The probability of correctly predicting word w_i given word w_j is determined by the softmax model, which is

$$p(w_i|w_j) = \frac{\exp(\vec{u_{w_i}}^{\intercal}\vec{v_{w_j}})}{\sum_{l=1}^{V}\exp(\vec{u_l}^{\intercal}\vec{v_{w_j}})}, \tag{2}$$

where V is the vocabulary size.

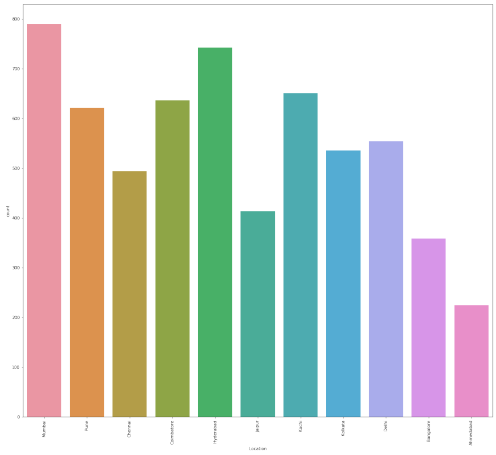
The skip-gram model with softmax is expensive because the cost of computing $logp(w_i|w_j)$ is proportional to V, which can be easily in order of millions. To speed up training of **Word2Vec**, we used hierarchical softmax, which reduced the complexity of computing of $\log p(w_i|w_j)$ to $O(\log(V))$.

In this **word2Vec**, we can set up the dimension of the output vector size, which is a hyperparameter in our model.

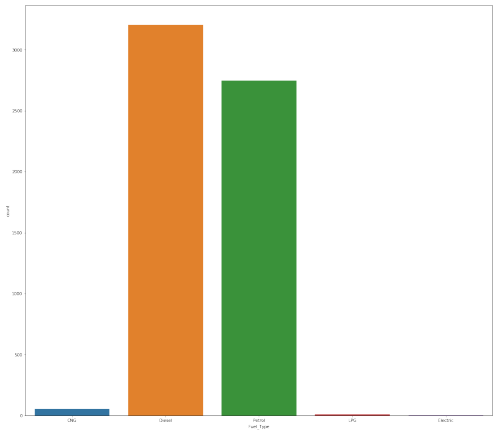
2) Location, Fuel_Type, Transmission, Owner_Type: For the rest categorical features, Fig. 3 shows the distributions, it is easy to conclude that the different values of these features are small, thus it is reasonable to apply binarization, which is also called one-hot encoding. One-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). Although one-hot encoder will cause dimension lift, it is still in a low dimension since the small number of different values.

C. Number Features: Kilometers_Driven, Mileage, Engine, Power, Seats

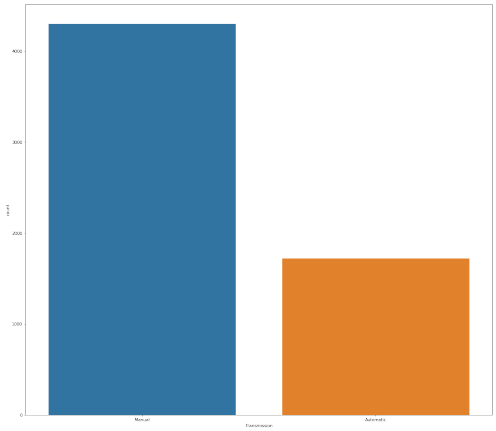
For the features with numerical values, we can directly use those value. However, things are not always like what we desire. The first problem is that some features although have numerical part, they have units and the data type is string. The second problem is that among all numerical features, the values locate in totally different ranges. Some are in thousands while others are less than 1 in Fig. 4. All these troubles need to be considered in preprocessing.



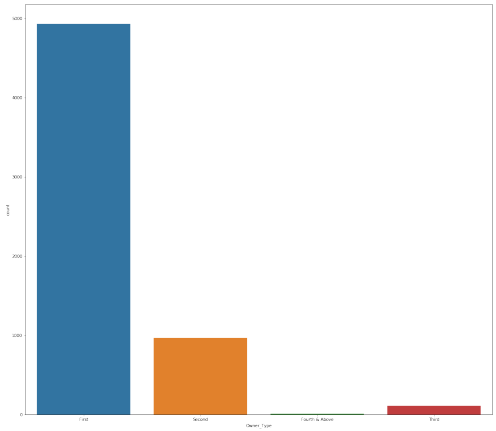
(a) Distribution of Location.



(b) Distribution of Fuel_Type.



(c) Distribution of Transmission.



(d) Distribution of Owner_Type.

Fig. 3: Distribution of Location, Fuel_Type, Transmission and Owner_Type. The exact values are not important. This figure only shows the number of distinct values is small.

1) Number Extraction: First let's focus on number extraction. Since these features are in "number + unit" format, we only need to split entire string and throw unit part.

2) Normalization: In order to find the fast path to get the lowest cost in training progress, we need to normalize dataset. The formula is given in Eq. 3. After normalization, all features are in zero-mean and one-variance.

$$x_{normalized\_i} = \frac{x_i - \mu}{\sigma}, \tag{3}$$

where μ is the mean of data and σ is the standard deviation of data.

V. Linear Regression

Now we have already converted features to a matrix $\vec{X} \in \mathbb{R}^{n \times d}$ and the label is price which is $\vec{Y} \in \mathbb{R}^n$. In the following part, we would like to find a mapping $f : \mathbb{R}^d \to \mathbb{R}$ to convert $\vec{X}$ to $\vec{Y}$ so that for a new $\vec{x} \in \mathbb{R}^d$, the mapping can give an output $\hat{y} \in \mathbb{R}$, which is the predicted used car's price.

A. Principle

Since this problem is a prediction estimation, we would like to start our model from linear regression, which is easy and has the least expectation error in linear model.

	Year	Kilometers_Driven	Seats	Mileage_upd	Engine_upd	Power_upd
count	5973.000000	5973.000000	5973.000000	5973.000000	5973.000000	5973.000000
mean	2013.387075	57588.486858	5.280261	18.342590	1621.514817	112.859955
std	3.246697	37904.170824	0.805087	4.168823	600.781324	53.680592
min	1998.000000	171.000000	2.000000	6.400000	624.000000	34.200000
25%	2012.000000	33901.000000	5.000000	15.300000	1198.000000	75.000000
50%	2014.000000	53000.000000	5.000000	18.200000	1493.000000	93.700000
75%	2016.000000	73000.000000	5.000000	21.100000	1984.000000	138.100000
max	2019.000000	775000.000000	10.000000	33.540000	5998.000000	560.000000

Fig. 4: Statistic of numerical features. Their value are in different ranges, so normalization is needed for fast optimization.

Generally, linear regression is to find a $\vec{\beta} \in \mathbb{R}^d$ to make prediction by

$$\hat{y} = \vec{x}^{\intercal}\vec{\beta}, \tag{4}$$

or written for entire dataset

$$\hat{\vec{Y}} = \vec{X}\vec{\beta}. \tag{5}$$

In order to find the best $\vec{\beta}$, we would like to minimize the loss function of prediction. Assumed that for each data sample, the loss function is $\ell(y, \hat{y})$, the total loss function is

$$L(\vec{Y}, \hat{\vec{Y}}) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i). \tag{6}$$

Besides, in case of overfitting, a regularization term is often added after loss function part, so the total loss function becomes

$$L(\vec{Y}, \hat{\vec{Y}}) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \lambda \left\| \vec{\beta} \right\|_2^2, \tag{7}$$

where λ is regularization parameter, which determines the wight of regularization term.

B. Loss Function

There are different types of loss function $\ell(y_i, \hat{y}_i)$ available. They have different formula and they are applied in different cases.

1) Mean Absolute Error (MAE) or L1 Loss: MAE is the sum of absolute differences between our target and predicted variables. So it measures the average magnitude of errors in a set of predictions, without considering their directions. (If we consider directions also, that would be called Mean Bias Error (MBE), which is a sum of residuals/errors). The range is also 0 to ∞.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} \tag{8}$$

2) Mean Square Error (MSE) or L2 Loss: Mean Square Error (MSE) is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values.

$$\text{MSE} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n} \tag{9}$$

L1 loss is more robust to outliers, but its derivatives are not continuous, making it inefficient to find the solution. L2 loss is sensitive to outliers, but gives a more stable and closed form solution (by setting its derivative to 0).

C. Optimizer

In order to find the $\vec{\beta}$ to minimize total loss function $L(\vec{Y}, \hat{\vec{Y}})$, we need to find a kind of optimizer to solve this minimize problem. There are multiple selections.

1) Batch Gradient Descent: Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. the parameters $\vec{\beta}$ for the entire training dataset:

$$\vec{\beta} = \vec{\beta} - \alpha \nabla_{\vec{\beta}} L(\vec{\beta}; \vec{Y}, \hat{\vec{Y}}). \tag{10}$$

As we need to calculate the gradients for the whole dataset to perform just **one** update, batch gradient descent can be very slow and is intractable for dataset that do not fit in memory.

2) Stochastic Gradient Descent: Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example $\vec{x_i}$ and label y_i:

$$\vec{\beta} = \vec{\beta} - \alpha \nabla_{\vec{\beta}} L(\vec{\beta}; y_i, \hat{y}_i). \tag{11}$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

3) Mini-Batch Gradient Descent: Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of k training examples:

$$\vec{\beta} = \vec{\beta} - \alpha \nabla_{\vec{\beta}} L(\vec{\beta}; y_{i:i+k}, \hat{y_{i:i+k}}). \tag{12}$$

This way, it reduces the variance of the parameter updates, which can lead to more stable convergence, and can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

4) Other Optimizers: Besides optimizer based on gradient descent, there are multiple other types optimizers, like Adagrad, Adadelta and Adam. Here we will not cover these optimizer because optimizer selection is not the main purpose of our project.

VI. Implementation

In order to implement all in parallel, we applied map-reduce during preprocessing, and used model in *pyspark.ml* package, which is a library in pyspark and provides lots of API to make machine learning easy. Besides, to avoid overfitting, we applied K folds cross-validation in training process and compute Root Mean Square Error (RMSE) both for train dataset and test dataset.

A. Preprocessing

Recall in preprocessing, we are supposed to convert feature to vector for feature Name and apply binarization for other categorical features, and extract number value for all numerical value and apply normalization.

B. Model

The linear regression model is a built-in model in *pyspark.ml* package. All its hyperparamters are given as below.

```
class pyspark.ml.regression.LinearRegression(featuresCol='features', labelCol='label',
    predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06,
    fitIntercept=True, standardization=True, solver='auto', weightCol=None,
    aggregationDepth=2, loss='squaredError', epsilon=1.35)
```

`featuresCol` is the feature matrix, which is also $\vec{X} \in \mathbb{R}^{n \times d}$, `labelCol` is the true label column, which is also $\vec{Y} \in \mathbb{R}^n$, `predictionCol` is the prediction label, which is also $\hat{\vec{Y}} \in \mathbb{R}^m$, `maxIter` and `tol` determines when the training process ends, `elasticNetParam` decides what kind of regularization is applied, `regParam` determines the weight of regularization term in objective function, and `loss` decided what kind of loss function is applied.

C. Hyperparameter Tuning

In this project, we only tune two hyperparameters to find the best model: `vectSize` and `regParam`.

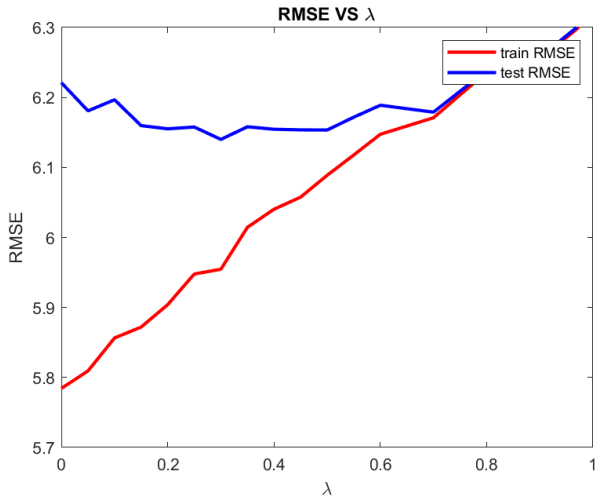
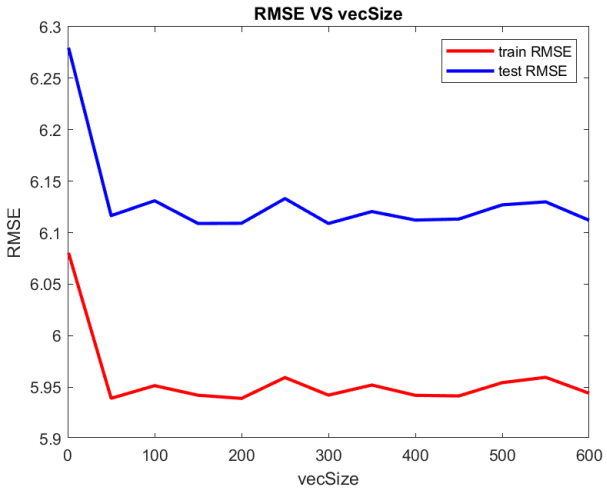


Fig. 5: Train and test RMSEs in different λs. There is an optimal λ for test RMSE but is monotonous for train RMSE.

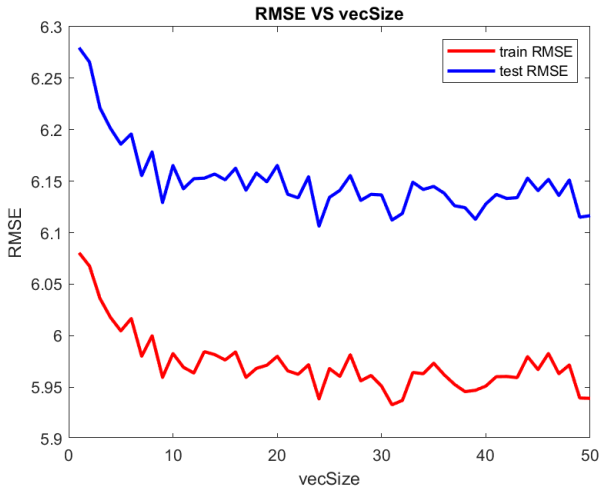
1) Regularization Parameter: In Fig. 5, we compute train and test RMSEs in different λs. It is easy to conclude that train RMSE will increase as λ increases, however, the test RMSE is not monotonous, there looks like a λ that minimizes test RMSE, and that λ is what we look for.

Here we would like to explain a little bit more about this figure. Remember that λ determines the weight of regularization term. For training dataset, if $\lambda = 0$, that means regularization is not considered, so the train RMSE should be minimum but overfitting, and test RMSE is not minimum. Then as λ increasing, more regularization will be considered, train RMSE will be larger, but test RMSE will decrease because overfitting is vanishing. But there is an optimal λ for test RMSE, as λ keeps increasing, the weight of regularization is too large, the objective function is mainly to minimize the norm of $\vec{\beta}$, which will cause error.

2) word2Vect Size: Fig. 6 shows how `vecSize` effects the training process. In conclusion, when `vectSize` is large enough, larger than 5, both RMSEs of train and test datasets are almost invariant. Now considering that cost of computation, a less `vecSize` will decrease the cost.



(a) Train and test RMSEs in different vecSizes when vecSize $\in [0:600]$.



(b) Train and test RMSEs in different vecSizes when vecSize $\in [1:50]$.

Fig. 6: Train and test RMSEs when vecSize are in different values. Fig. 6a shows the relationship when vecSize varies in a big range, it can be concluded that when vecSize is larger than 50, the RMSEs of both train and test dataset decrease, and after that, even vecSize keep decreasing, the RMSEs keeps constant. Fig. 6b shows the same thing when vecSize varies in $[1, 50]$. Both train and test RMSEs decrease rapidly when vecSize increases and then keep constant.

VII. Result Analysis

After we found the optimal λ and `vecSize`, we would like to evaluate our model. We uses these hyperparameters with optimal values to train our model and evaluate by k folds cross-validation.

features	Price	prediction
[-0.1509497420933...	65.81	41.61584208566152
[-0.1509497420933...	4.99	8.85005894208614
[-0.1509497420933...	5.7	8.057604772921088
[0.79704692696778...	11.99	15.750293335484848
[0.79704692696778...	29.99	23.77273806173353
[1.42904470634186...	9.5	10.803636311675545
[0.16504914759371...	5.5	5.6519938715992435
[0.79704692696778...	23.0	23.76070496991681
[-0.1509497420933...	2.95	2.1339389987670785
[0.48104803728075...	4.9	3.9517958511333964

Fig. 7: Validation result samples. Features column are the vectors that are converted from original data. Price column contains the true value of prices, while Prediction column is the predicted price by our model.

Fig. 7 shows some predictions and compares with true labels. Generally, the predicted prices differ from true prices in a small range. In order to evaluate our model in numbers, we would like to record the RMSEs of train dataset and test dataset in Tab. II, and analysis these RMSEs.

k-th fold	0	1	2	3	4
Train RMSE	5.9476	6.0138	5.9784	5.8962	6.0063
Test RMSE	6.3960	5.6885	6.0662	8.0781	5.7908
k-th fold	5	6	7	8	9
Train RMSE	6.0389	5.8647	6.0054	6.0655	5.9692
Test RMSE	5.4194	6.9873	5.8075	5.2517	6.0612

TABLE II: RMSEs of train and test dataset.

A. Bias vs Variance

Remember that the price is in thousand and the distribution of Price is displayed in Fig. 2. From Tab. II, it is easy to find that the bias is around 6 and the variance is too larger than bias except 3-th fold. Considered that the standard deviation of Price is around 11, so the bias which is about 6 is not big, it is acceptable. Besides, the variance is small. The training is pretty good, we almost found the best hyperparameters of this linear regression model.

B. Boosting

The bias means the predicted value has around 6k dollars away from the true prices. It is not large consider the standard deviation. However, normal used cars are usually sold at tens of thousand, so the model with difference 6k is really hard to say can be a valid model for prediction.

We implement this prediction model in linear regression, but the performance is not that good. There are several methods to improve out model.

1) Model Selection: Since we almost find the best hyperparameters of the linear regression model, but it can not give a performance to maintain test RMSE under 1k, we come up with a fact there is a better model for this prediction. Maybe the true mapping from all given features to used cars' prices are not linear. In consequence, we could never find a perfect linear model to fit this prediction.

Naturally, if linear is not working well, quadratic model comes out. We could combine any two features to a new feature which is quadratic term. In this way, the model is lifted to a quadratic model, which may have better performance than linear model.

If quadratic model still can not satisfy our needs, we could try Deep Neural Network (DNN). Compared with linear regression, the DNN is the combination of linear regression plus non-linearity. Each layer in DNN is a linear regression followed by a activation function as

$$\vec{Z^{[l]}} = \vec{A^{[l-1]}}\vec{W^{[l]}} + \vec{b^{[l]}}, \tag{13}$$

$$\vec{A^{[l]}} = g^{[l]}(\vec{Z^{[l]}}), \tag{14}$$

where $\vec{A^{[l]}}$ is the output of layer l, $\vec{W^{[l]}}$ is the weight of layer l, $\vec{b^{[l]}}$ is the bias of layer l, $\vec{Z^{[l]}}$ is the linear result of layer l and $g^{[l]}(\cdot)$ is the non-linear function of layer l.

2) Large Dataset: Remember the dataset we used only has 6019 samples, while some are non-valid. This is not a large dataset, and with such dataset, it is hard to train a model with good performance. In machine learning, there is a saying that the model training with more data will have better performance. Thus, if we would like to boost the performance of our model, we need more data.

VIII. Parallelism

Since this project is based on course EECE5645 Parallel Processing Data Analytics. We implemented our project in parallelism. In this part, we would like to list where we apply parallelism.

A. Preprocessing

Before preprocessing, convert data into RDDs and send them to workers.

1) Tokenization: Tokenization is a process to split sentences into words. This happens in workers, meanwhile in parallelism.

2) Stopwords Removal: After tokenizing sentences, some tokens have less meaningfulness. In order to reduce computation, we would like to remove these words called stopwords. This removal also happens in parallel.

3) word2Vec Conversion: After we construct the dictionary of words mapping to vectors, we would apply this dictionary to convert a sentence to a vector. This preprocessing can be finished in workers.

B. Training

During model training, there are some processing we use parallelism.

1) Model Fitting: Since we use *pyspark.ml* package, which is a built-in library API for *pyspark*, all these are implemented in map-reduce.

2) Test RMSE Computing: After we train model for each fold in k folds cross-validation, we would like to compute test RMSE to evaluate our model. When computing, we used RDDs to get that value.

IX. Roles and Responsibilities

Generally, it is a incredible time to work in this group. All of us discussed with this project frequently and everyone finished his own work in good quality and in a quick time.

A. Quanwei Hao

- Contributed to write proposal.
- Proposed idea to preprocessing and training in RDDs.
- Analyzed row dataset.
- Constructed main structure of scripts and finished coding in preprocessing and training in RDDs.
- Contributed to write presentation.
- Contributed to write final report.

B. Ruixiao Duan

- Contributed to write proposal.
- Analyzed row data and plotted figures fro data visualization.
- Finished scripts and finished coding in preprocessing and training in dataframes.
- Tuned hyperparamters and analyze performance of model.
- Contributed to write presentation.
- Contributed to write final report.

C. Wangbo Jia

- Contributed to write proposal.
- Proposed idea to preprocessing and training in dataframes.
- Constructed main structure of scripts and finished coding in preprocessing and training in dataframes.
- Tuned hyperparamters and analyze performance of model.
- Contributed to write presentation.
- Contributed to write final report.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[2] A. S. Goldberger, "Best linear unbiased prediction in the generalized linear regression model," *Journal of the American Statistical Association*, vol. 57, no. 298, pp. 369–375, 1962.

[3] J. Gong and S. Sun, "A new approach of stock price prediction based on logistic regression model," in *2009 International Conference on New Trends in Information and Service Science*. IEEE, 2009, pp. 1366–1371.

[4] E. Schöneburg, "Stock price prediction using neural networks: A project report," *Neurocomputing*, vol. 2, no. 1, pp. 17–27, 1990.

[5] K. Alkhatib, H. Najadat, I. Hmeidi, and M. K. A. Shatnawi, "Stock price prediction using k-nearest neighbor (knn) algorithm," *International Journal of Business, Humanities and Technology*, vol. 3, no. 3, pp. 32–44, 2013.

[6] W. Fenghua, X. Jihong, H. Zhifang, and G. Xu, "Stock price prediction based on ssa and svm," *Procedia Computer Science*, vol. 31, pp. 625–631, 2014.

[7] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.