Used Car Value Prediction

Yang Gao, Yifei Li, Zhuochen Liu, Yuankai Wang, Hao Yang
Nov. 9, 2018

Abstract

In our project, we tried to predict the price of used cars in Poland. We built several models for the dataset provided by user mapik88 on Kaggle (available at [1]). Our final model is a classification model which achieved roughly 80% testing accuracy.

1 Introduction

When people want to sell their cars, they want to have an idea about how much their cars worth. A natural approach is to find several similar used cars on the website and look at their price. Unfortunately, since the car price can be affected by many factors, it is time-consuming to find similar cars on the internet. Therefore, we would like to build a model for the data of used cars. Eventually, we found a proper dataset for used car price on Kaggle [1]. The dataset contains various features about the used car, including (but not limited to) the car model, car maker, mileage, and color. Since the dataset has a reasonable size (contains information for more than 200k cars) and covers a lot of features, we decide that this dataset is ideal for training our model.

2 Preprocessing

After we downloaded the data and examined it, we found that the data is not perfectly structured for the neural network to train with. It contains a lot of empty values, useless features, and meaningless outliers.

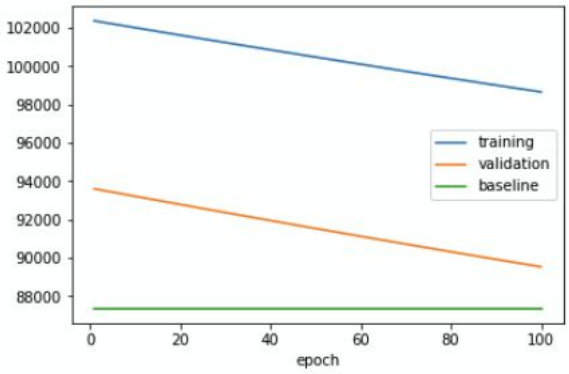
The data contained start time and end time with the same value, which is meaningless, so we deleted those columns. Some of the columns are the identification information of the car, such as the VIN number. These columns are not features of a car, so we also removed them. Some columns have the same value for all cars so they will not affect the output results. We removed these columns to speed up training process. Some of the columns have the same meaning, such as the year of production and the car age, so only one of them need to be reserved. Some of the features have both a raw classification value column and one-hot encoded columns, so we removed the columns for raw values. Some rows of data do not contain all of those information, or contain null data in other words, so we removed those rows. (The data are

started and ended at about the same time, so we delete the column related to the start and end time. The unique id is needed for a database, but a single index is sufficient for us to train using the neural network. Some columns that all the data has the same value, like the dealer name. Some of the columns contain the same meaning, like the production year and the car age. Some of the columns contain null data, like the VIN number. Some of the columns contain both its original form and its one-hot encoded form, like gearbox kind.) Some of the rows contain outliers. For example, one of the cars with a poor condition has a price of 3,000,000,000, which we suspect it involves money laundry. To get rid of those outliers, we went through the whole dataset using the filter function of excel worksheet and polished the data.

We finally got a dataset of 96582 rows and 97 columns. We uploaded the data to the server and loaded it using python. We picked out the columns that need to be standardized, which include car age, mileage, the number of parameters available, and the number of features. We then one-hot encoded the data that had not been encoded yet and finally got 1106 columns. Before we trained the model, we split the dataset into training and testing data with a ratio of 80% to 20%. For training data, we split it into training and validation sets again with a ratio of 80% to 20%.

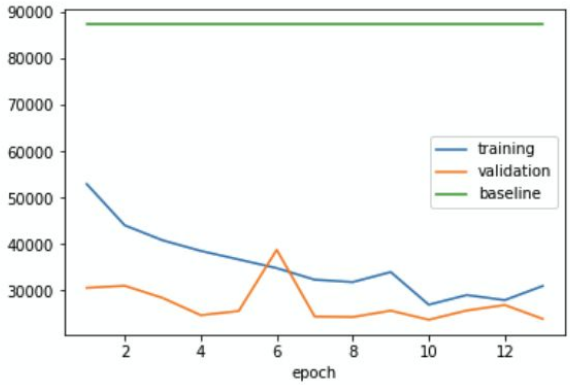
3 Regression Approach

Our first approach is to train a network that can predict the accurate prize. To have an overview of the model, we computed the standard deviation of the pricing and set it as the baseline. Then, we trained the network using a simple model. The model has one layer and applied early stopping (patience is set to 3). Initially, we wanted to set the loss of this model as the baseline. However, the training loss and the validation loss are both worse than the standard deviation (shown in picture 1). Even though the loss keeps decreasing, it is not a reasonable model since it does not get better than standard deviation even after 100 epochs. Thus, we decided to keep the standard deviation as the baseline when training our future models.



Picture 1: The training loss, validation loss and standard deviation for the simple model.

Then, we constructed a more sophisticated model. The model has five layers, and for each layer, there are 10000, 1000, 100, 10 and 1 nodes. To get rid of negative values, we applied ReLU activation to each layer. We also included early stopping for the model (with patience equals to 3). The network stopped training due to early stopping after 13 epoches. This time, we found that the training loss and validation loss are both smaller than baseline (shown in picture 2). Therefore, we kept this model as the best regression model we trained.



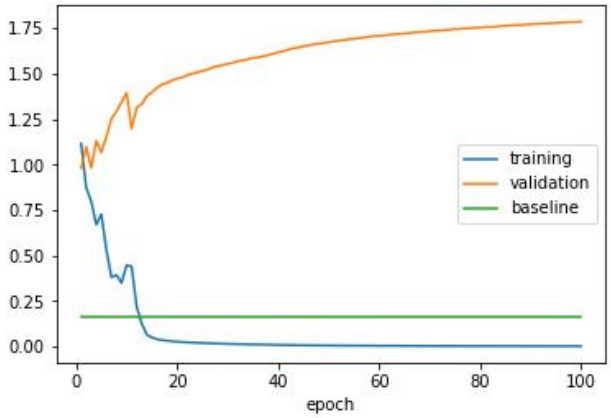
Picture 2: The training loss, validation loss and standard deviation for the sophisticated model.

4 Classification Approach

We feel that it is hard to determine the exact value of a used car, and because the price of a car is a large number, a small prediction error can lead to a large loss. Besides, the most important thing we want to know is an approximate price range of the used cars. Therefore, we decide to divide cars into six categories according to their prices and build a classification model.

The first division method we used is to divide our dataset into six parts such that each part has the same amount of data. We trained with the new classification model.

We decided to use 1% of the original dataset to test the model since we do not want to waste too much time training the huge dataset. We wanted to drive our training set error to 0 in order to deliberate overfitting. We tried several models and one of them drive the training error to 0.



Picture 3: The plot of the loss for the model with 1% dataset (see appendix for details)
The result showed that this model is correct for the problem. We decided to use this model for the whole dataset in the following step.
We got about 77% accuracy on the testing set. However, we figure out that our division method is only applied to our input data, which is not a reasonable approach.

However, we then realized that the price of a car is not uniformly distributed, so we modified our way to divide the six categories. The new categories are 0-4999, 5000-9999, 10000-19999, 20000-49999, 50000-99999, and greater or equal to 100000. We believe this kind of division system better classifies the quality of cars according to their prices.

The baseline accuracy of this approach is to find out the most common price interval and divide it by the number of all data, which is the accuracy we can achieve by taking random guesses. The baseline accuracy equals 31.76% and our model should perform at least better than the baseline.

Our first model is to simply combine several hidden layers with ReLU activation in between in order to prevent the layer from collapsing, and uses early stopping to prevent overfitting. The validation accuracy for this model is around 77%, which is good for us, but since the training stops at 8th epoch due to overfitting, we want to try to apply dropout to train more epochs and improve the result.

Our final model consists of 5 hidden layers with “ReLU” activation layers in between. The dropout regularization is applied between each pair of layers with a dropout rate of 0.3. The model achieves a validation accuracy of 81.7%, and the accuracy for predicting the test data is 80.7%, which is pretty good.

5 Results

Regression model:
For our regression model, since it is hard to compute the exact pricing of the car, we measured our regression model's accuracy based on the percentage difference between the predicted pricing and the actual pricing. After using the sophisticated model to predict the pricing for cars in the testing set, we found out that there are around 45% of the predicted values have an error less than 10% and around 74% of the predicted values have an error less than 20%. We also noticed that for some cars, the error rate can go up to 1300%. Due to the low performance of this regression model, we eventually decided to abandon this model and take the other approach.

We measure our accuracy not only throw the #corr/#total data but also through identifying #of classification within 2 stages difference(for example, classified as second categories but should be fifth categories). Here is the final result:
within a total of 19316 test point, we got
correct points: 15550
incorrect points: 3766

number of incorrect points differ by 2 stage: 108

6 Conclusion

Our best deep learning accuracy for this around 80%. We think it is an acceptable result for this problem. Having an estimation of the price of the used car help people to have a better prediction their cost on buying a car or selling their used cars, thus avoiding to be cheated by evil people or dealers. The dataset we used is from Poland. If we want to estimate the price for used cars in other regions, this model will be suitable as well. The only thing we need to do is trying to collect more accuracy and reliable data from the society.

References:

[1]. mapik88. Used Car Offers, Kaggle, 2018, www.kaggle.com/mapik88/used-car-offers.

Appendix:

Model And Result for Regression Single Layer Model with one node:
(Model):

```
epoch = 100
patience = 3
```

```
model = Sequential()
model.add(Dense(1,input_shape=(1105,)))
model.compile(loss="mean_squared_error", optimizer="Adam", metrics=['accuracy'])
hist = model.fit(X_train, P_train, epochs=epoch, validation_split=0.2, verbose=1, callbacks=[EarlyStopping(patience=patience)])
```

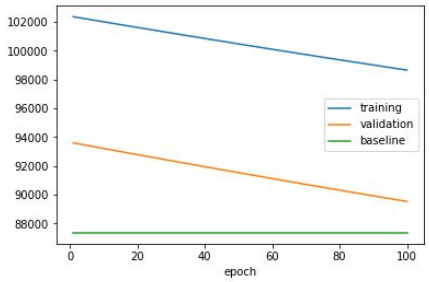
(Result Summary):

```
7]: model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 1106
=================================================================
Total params: 1,106
Trainable params: 1,106
Non-trainable params: 0
_________________________________________________________________
```

```
8]: results = pd.DataFrame()
results['epoch'] = hist.epoch
results['epoch'] = results['epoch'] + 1
results['training'] = np.sqrt(hist.history['loss'])
results['validation'] = np.sqrt(hist.history['val_loss'])
results['baseline'] = P_train.std()
ax = results.plot.line(x='epoch',y='training')
ax = results.plot.line(x='epoch',y='validation',ax=ax)
results.plot.line(x='epoch',y='baseline',ax=ax)
print("RMSE validation =",results.validation.iloc[-1])

RMSE validation = 89519.0695182425
```



Model and Result for Improved Regression Model:
(Model):

```
In [15]:
epoch = 100
patience = 3
```

```
In [ ]:
model = Sequential()
model.add(Dense(10000,input_shape=(1105,)))
model.add(Activation('relu'))
model.add(Dense(1000))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dense(1))
model.compile(loss="mean_squared_error", optimizer="Adam", metrics=['accuracy'])
hist = model.fit(X_train, P_train, epochs=epoch, validation_split=0.2, verbose=1, callbacks=[EarlyStopping(patience=patience)
```

(Result):

```
In [90]:
def rmse_test(Ph,P):
    s = 0
    for i in range(len(P)):
        s=s+(Ph[i] - P[i])**2
    s=s/(len(P))
    return s**(1/2)
```

```
In [117]:
df_comp = pd.DataFrame()
df_comp['Predicted Value'] = Ph_test
df_comp['Expected Value'] = P_test
df_comp['Difference'] = abs(Ph_test-P_test)
df_comp['Within 10%'] = df_comp['Difference']<df_comp['Expected Value']*0.1
df_comp['Within 20%'] = df_comp['Difference']<df_comp['Expected Value']*0.2
```

```
In [118]:
df_comp.head(3)
```

Out[118]:

	Predicted Value	Expected Value	Difference	Within 10%	Within 20%
0	11453.482422	8500.0	2953.482422	False	False
1	2379.913330	2300.0	79.913330	True	True
2	22866.632812	25500.0	2633.367188	False	True

```
In [123]:
sum(df_comp['Within 10%'])/len(df_comp['Within 10%'])*100
```

```
Out[123]: 45.06626630772417
```

```
In [124]:
sum(df_comp['Within 20%'])/len(df_comp['Within 20%'])*100
```

```
Out[124]: 73.94905777593705
```

Model And Result for Classification Model:

(Model):

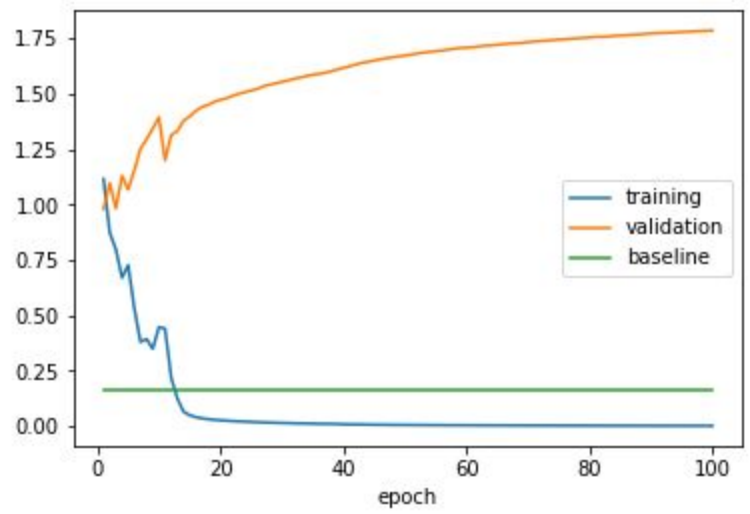
```
epoch = 100
patience = 3
```

```
model = Sequential()
model.add(Dense(1000,input_shape=(1105,)))
model.add(Activation('relu'))
model.add(Dense(500))
model.add(Activation('relu'))
model.add(Dense(250))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(6))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
hist = model.fit(X_small_train,P_small_train,epochs=epoch,validation_split =0.2,verbose=1)
# ,callbacks=[EarlyStopping(patience=patience)]
```

(Result):

```
results = pd.DataFrame()
results['epoch'] = hist.epoch
results['epoch'] = results['epoch'] + 1
results['training'] = np.sqrt(hist.history['loss'])
results['validation'] = np.sqrt(hist.history['val_loss'])
results['baseline'] = 1/6
ax = results.plot.line(x='epoch',y='training')
ax = results.plot.line(x='epoch',y='validation',ax=ax)
results.plot.line(x='epoch',y='baseline',ax=ax)
print("RMSE validation =",results.validation.iloc[-1])
```

```
RMSE validation = 1.7849863621196171
```



We measure our accuracy not only throw the #corr/#total data but also through identifying #of classification within 2 stages difference(for example, classify as second categories but should be fifth categories). Here is the final result:
within total of 19316 test point, we got
correct points: 15550
incorrect points: 3766

number of incorrect points differ by 2 stage: 108