CAP4770 Assignment 2

Note: For your programming assignment, you can create a pdf file containing all your code and output. For instance, a pdf file generated from the Jupyter notebook will be great. Submit your pdf file.

Load the data set named 'car-data.csv' on Canvas. You will run regression analysis on it. The target/output will be 'Selling_Price'. You will use the rest attributes/features to predict the selling price.

Although 'Car_Name' should be quite relevant to the selling price, there is no easy way to use that attribute without getting additional information not currently in the dataset. You can ignore (drop) that attribute. If you have extra time, I encourage you to think about how that information may be used.

You will go through a similar sequence of steps as we illustrated in the lectures (in the two sets of slides about an end-to-end machine learning project). The goal is to get you some hands-on experiences about a machine learning project. You can adapt the code in Chapter 2 of the following book:
Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2nd edition, 2019.
You can find the Python code at the github page accompanying the book.

Feel free to explore on your own. For the submission, please go through at least the following sequence of steps.

Step 1: Split the data into training and testing sets.

1. Load the data as a data frame.
2. Briefly look at the dataset using DataFrame methods such as 'head()', 'info()', 'describe()', and 'value_counts()', etc.
   Question: Which attributes are numerical attributes, and which are categorical attributes?
   Question: Are there any missing data?
3. Generate histograms for the numerical attributes.
4. Split the dataset into a training set and a test set with test_size=0.2 of the dataset. Let's call the training set `strat_train_set', and call the test set 'strat_test_set'.
   Please use stratified sampling. First, we will split 'Year' into five bins. Choose the sensible boundaries for the bins. Add an attribute called 'Year_Cat', which is the categorical attribute representing which bin an input instance falls into. Then, do stratified sampling based on 'Year_Cat', using the Scikit-Learn class 'StratifiedShuffleSplit'. This gives you a training set and a test set. Put the test set aside for now. You will work with the training set for the most part.

Step 2: Experiment with the training dataset

5. Generate the correlation matrix. Display the correlations in the descending order of all the attributes with respect to the selling price. This tells you which attributes might be more important for predicting the selling price.
6. Use 'scatter_matrix()' to generate scatter plots between each pair of numerical attributes. Again, this tells you how the attributes are related. Note that the 'Kms_Driven' attribute seems to have strange relationship with other attributes.
7. Experiment with adding an attribute of kilometers driven per year. Let's call it 'Kms_Per_Year'. You may assume the current year is the maximum of the 'Year' attribute plus 1.
8. Generate the correlation matrix again. You will see that the newly added attribute is positively correlated with the selling price, which is counter intuitive.

9. Split the training set, ‘strat_train_set’, into two tables. One will be called ‘cars_inputs’, which contains the input attributes without ‘Selling_Price’; the other is called ‘cars_labels’, which contains the output labels only.
10. Experiment with one-hot encoding on the categorical attributes.

Step 3: Once you have learned about your data and how to manipulate it, you can write pipelines to do several steps of processing in one pipeline.

11. Write a transformer code to optionally add the ‘Kms_Per_Year’ attribute.
12. Write a pipeline code for the numerical attributes to do the following: call imputer to filling missing entries; call the transformer in the previous step to add (or not) the ‘Kms_Per_Year’ attribute; use the ‘StandardScalar’ class to scale the attributes’ values.
    Note: Although there may not be any missing data, it may still be a good idea to call the imputer.
13. Write a ‘ColumnTransformer’ to work on all the columns of the input data frame ‘cars_input’. It should call the pipepline for processing the numerical attributes in the previous step, and it calls OneHotEncoder on the categorical attributes. Let’s call the resulting array ‘cars_prepared’, which contains the processed input attributes.

Step 4: Run regression

14. Try linear regression. Print the prediction error, i.e., the rmse of the learned estimator when it applies to the entire training set.
15. Repeat the above with a decision tree. You should see that it overfits dramatically.
16. Perform 10-fold cross-validation with the decision tree, using the ‘cross_val_score()’ method. Print out the cross-validation scores.
17. Repeat the cross validation with linear regression and print out the cross-validation scores.
18. Repeat the cross validation with RandomForestRegressor and print out the cross-validation scores. Then, train it on the entire training set. You should see it still overfits somewhat.
19. Write code to do grid search on the best hyper parameters for the random forest regressor. Print out the best hyperparameters. Print out the feature importance under the best estimator.
20. Use the best estimator/model in the previous step to do prediction on the training data and print out the rmse.
    The result here is fragile. Different runs of the grid search will likely give you different estimators. Some may still overfit drastically. You can check this out by running your grid search code multiple times.

Step 5: Prediction on the test set and save the model

21. Prepare your test data. Use your best model in the previous step and make predictions on the test data. Print out rmse. Also, print out the 95% confidence interval (for the true rmse).
22. Calculate the coefficient of determination (R^2) of the final result.
23. Save your trained model using ‘joblib’.