Price Prediction of Cars

EE 660 Course Project

Project Type: (1) Design a system based on real-world data

Number of student authors: 1

Yijing Jiang, yijingji@usc.edu

12/06/2020

1. Abstract

This report is to solve the **regression problem** of price prediction of cars on some related features and compare the performance of different models. Use the **constant model** as a baseline and also use linear regression models , such as **MLE**, **Ridge** and **Lasso regression**, and nonlinear regression models, such as **KNN**, **CART**, **Random Forest** and **Boosting**. Nonlinear regression models perform better than linear regression models. And **Gradient boosting regression** has the best performance of RMSE on the test dataset equal to **1236.16**. Also **find the importance of features** with the price of cars.

2. Introduction

2.1. Problem Type, Statement and Goals

The Problem is to **predict the price of cars**. It's a **regression** problem and the **target output value** is the price of cars. We mainly focus on the performance of different models on this problem and to find the best model on predicting the price.

During the process, we may focus on **how those models work separately on this dataset**, and how their performance is affected by their parameters, how to choose the best parameter from comparing values of performance measures; and **comparing performance of linear regression models and nonlinear regression models**, analyze the reason of their performance and to give some suggestions on how to choose from them in real case problem like this.

We are interested in **how those models work on a real life problem with much noise** other than an ideal problem and dataset. And **how much can we improve them from setting proper parameters**.

The difficulty during training mainly comes from the **missing** and **unbalanced** dataset and also the **number of dataset** is not so sufficient. For this problem, we are also not sure about its **linearity** between features and target value.

2.2. Overview of Our Approach

I will use the constant model as the **total baseline**,and use *MLE* as the **baseline for linear regression models**, use *KNN* and *CART* as the **baseline for nonlinear regression models.**

For more linear regression models, *Ridge Regression* and *Lasso Regression* will be used to see how the regularization parameters work on the models. And how do they improve the overfitting conditions compared with the baseline model.

For more nonlinear regression models, *Random Forest* and boosting (*Gradient Boosting Regression*) for regression versions will be trained to see how they have improved from baseline models.

Use the **cross validation** method to implement the comparison of results between different models. Compute mean and variance of ***RMSE*** (Root Mean Square Error) on training and validation set in the results of all models. For some models, also record their coefficient or feature importance for further comparing.

In total, mainly focus on the comparison in these aspects: inside linear regression models; inside nonlinear regression models; between linear and nonlinear regression models.

3. Implementation

3.1. Data Set

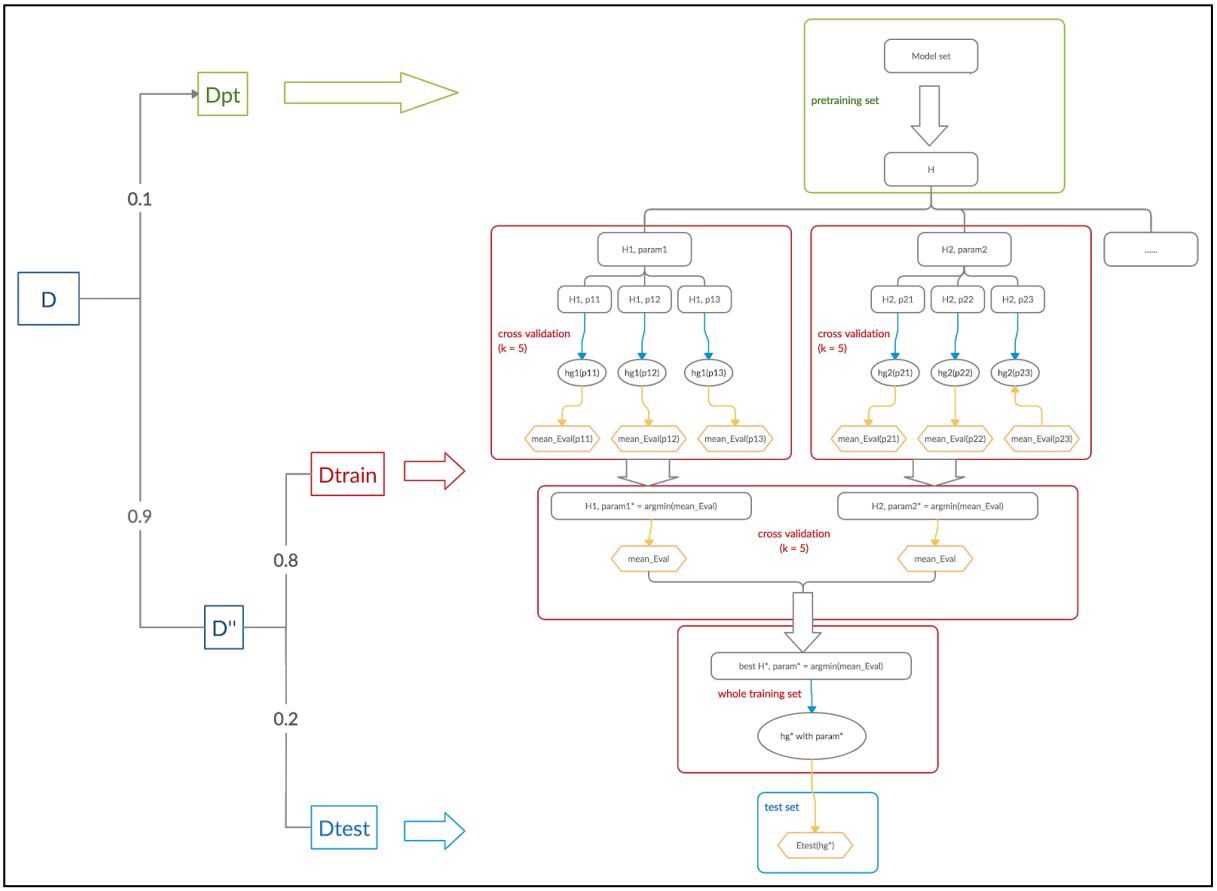
This dataset is from https://www.kaggle.com/nirmalnk/missing-cars. Basically, it is to predict the price of cars based on given features and to find the most related features in determining car price. It's a **regression** problem. Furthermore, by analyzing this data set, we can also roughly understand what kind of model is more reasonable for price prediction of this type in reality.

This dataset has totally **1436** datas which are comprised of the **price of cars** and **9 features of cars**: (the following two tables show the datas' information in file *'cars_missing.csv'*)

index	0	1	2	3	4
name	**Price**	**Age**	**KM**	**Fuel Type**	**HP**
meaning	Price of cars	Usage of car (months)	Km driven (km)	Fuel type engine used	Engine Power
type	integer	integer	integer	**categorical**	integer
range /cardinality	[4350, 32.5k]	[1, 80]	[1, 243k]	3 types (Petrol, Diesel, CNG)	[60,110]
missing data	No	No	No	No	**Yes(NA)**
unbalanced data	No	No	No	Yes(Significant)	No

index	5	6	7	8	9
name	**MetColor**	**Automatic**	**CC**	**Doors**	**Weight**
meaning	Colour of cars	Gear type of car	Engine CC	Number of doors	Weight of car (Kg)
type	binary	binary	integer	integer	integer
range /cardinality	2 types: {0,1}	2 types: {0, 1}	[1300, 2000]	4 types: {2, 3, 4, 5}	[1000, 1615]
missing data	**Yes(NA)**	No	No	No	No
unbalanced data	No	Yes(Major)	No	Yes(Significant)	No

3.2. Dataset Methodology



After dealing with the missing data, split the dataset into **Dpt** for pre training with 0.1 percent and **D''** for training and test with 0.9 percent. Split D'' into 0.8 percent **Dtrain**, and 0.2 percent **Dtest**. For **cross validation** set, use k = 5 folds validation and repeat for T = 10 times.

In the pretraining set, plot samples and pre-train a set of machine learning algorithms to decide hypothesis selection roughly.

For the training process, use cross validation to choose best parameters for each model separately and sequentially. If there are more than one parameters in the model, then the cross validation used here should be nested for those parameters. Then, use cross validation to compare the performance between different models and choose the best one as the final model. All the models are trained sequentially in the same cross validation set.

For the test set, it will only be used after choosing the best model to have the test set error on this model. So the test set is used only once in the whole process.

3.3. Preprocessing, Feature Extraction, Dimensionality Adjustment

- **Missing data and categorical features:**
  - In the features HP and MetColor, there exist missing datas with value of NA. Considering that the number of features are not so big, and each feature represents a unique information, choose to remove samples rather than to remove the whole feature. Because there is no more detailed information described in the dataset, we can't fill the missing data with reliable value. For the categorical feature, Fuel Type, assign integers to different types.
  - After processing the missing data, the number of samples decreases from the original 1436 samples ('*cars_missing.csv*') to currently 1242 samples ('*data_delete_missing.csv*'). Then split the dataset into different sets,we get:
    a. Dpt with 125 samples for pre-training ('*pretraining.csv*'),
    b. Dtrain with 893 samples for training ('*training.csv*'),
    c. Dtest with 224 samples for testing('*testing.csv*').

    The number of features has not changed.

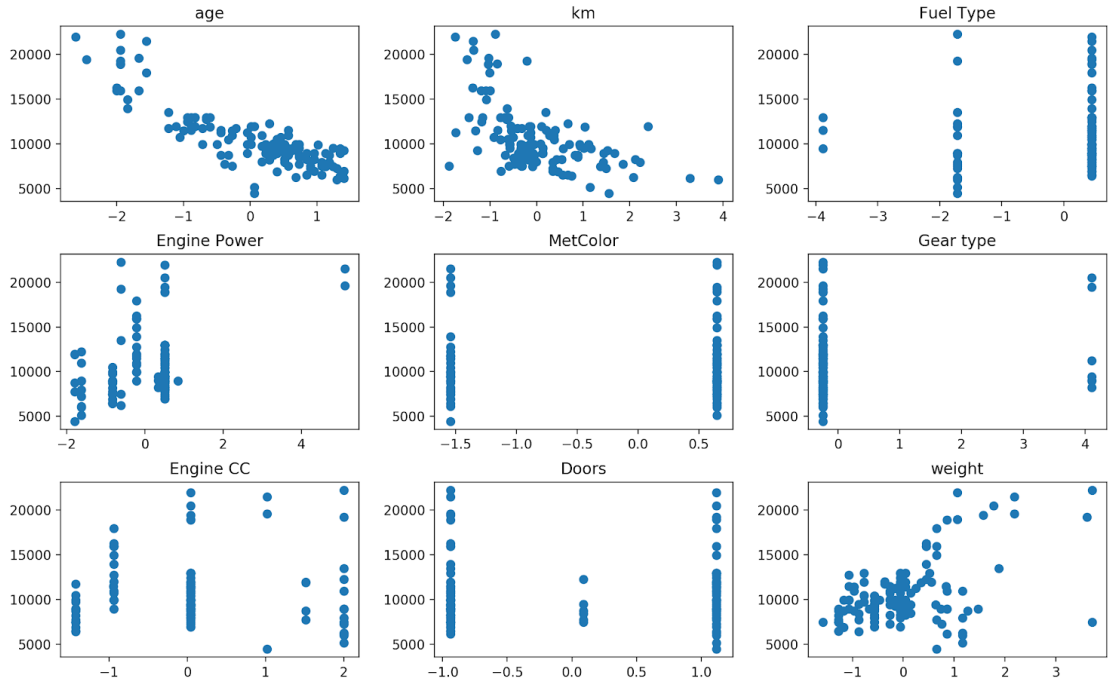
- **Pre-processing:**
  - **For linear models**, standardize the features before training the model. This is because linear regression models always make Gaussian assumptions on data. Use *sklearn.preprocessing.StandardScaler* in python. Standardization process can result in features with zero mean and one standard deviation.
  - **For nonlinear models,** normalize the features before training. This process can result in features from range [a, b] to range [0,1]. We choose to use *sklearn.preprocessing.MinMaxScaler* in python.
- **Feature extraction and dimensionality adjustment:**
  - Mainly do feature extraction and dimensionality adjustment on MLE algorithm. Choose to compare MLE on all features, MLE on one feature, MLE with polynomial features and MLE with PCA to see how features related to the target function.
  - **Polynomial features** is the method commonly called basis-set expansion in machine learning. It will change the original one-dimensional features into n-th order features to adjust the dimensionality of features.
  - **PCA** is the method to project original features to a lower dimensional space to reduce the dimensionality of features.

    Detailed implementation of methods will be described in Training Progress.

- **Pre training process**

  Look into samples by plotting features and the target value in the next page. We can see that **features with continuous values**, like feature age, km and weight, **scatter into striped shape**. And for the rest of the features, they are kind of discrete, maybe we can use the distribution on each of their values for regression. **The samples are not in a clearly linear trend in those continuous features**. We can

explore more on linearity with feature extraction and dimensionality adjustment by using the MLE model. If **too much noise** on data is one reason, we can expect improvement by using regularization methods like Ridge regression and Lasso regression. Or if **the linearity of data is not obvious**, we can expect better performance of nonlinear regression models.



Then choose the hypothesis set roughly. (split the pretraining set into training and test set)

- **MLE on all features and on one feature:** plot the E_train and E_test in *Fig1*. Features age, km, weight and Engine power have smaller error than other features, which matches the plot of samples above. We can get the assumption roughly that these four features are more related to price than the other five features.
- **MLE with PCA**: plot the E_train and E_test with different components in *Fig2*, E_train and E_test both decreases at first, but later E_test increases at components equal to 7, which indicates that for component greater than 7,the model is overfitting. So we decided to choose **components = 7**.
- **MLE with polynomial features**: try different d as follows,

	E_train	**E_test**
d = 1	1485.93	1554.14
d = 2	2361.58	2574.74
d = 3	280.66	1708397.18
d = 4	6.93e-10	96297.20

Fig1: pretraining_MLE_all features vs one features

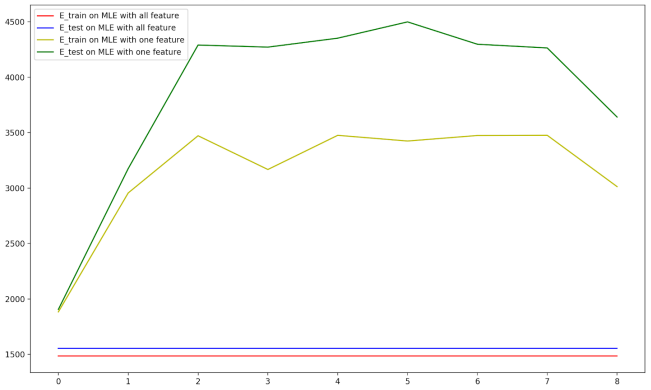


Fig2: pretraining_MLE with different components in PCA

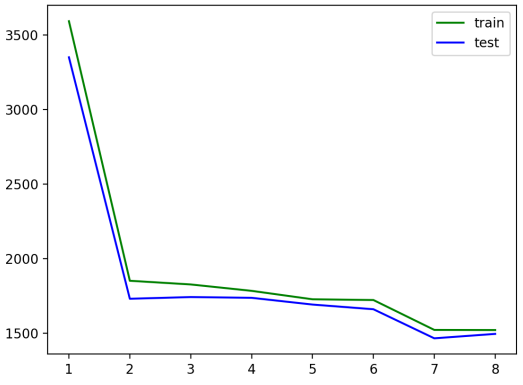


Fig3: pretraining_Ridge Regression with lambda in range [0.0001, 1000]

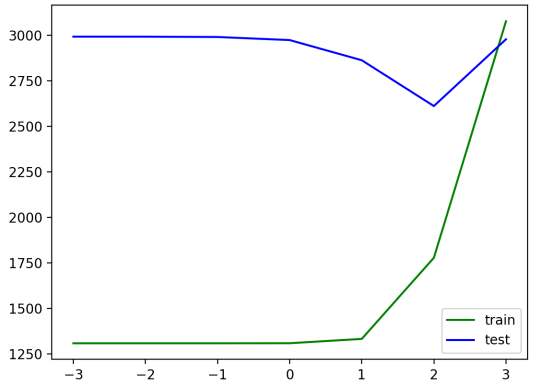


Fig4: pretraining_Ridge Regression with lambda in range[0.001, 200]

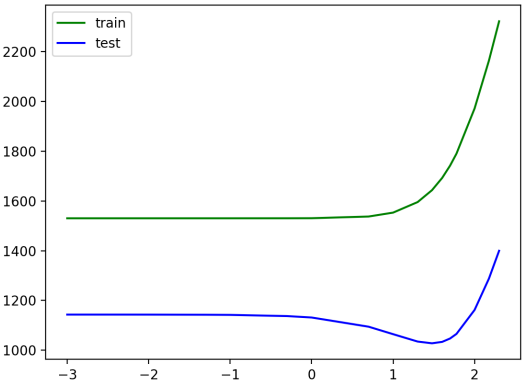


Fig5: pretraining_Lasso Regression with lambda in range[0.001, 1000]

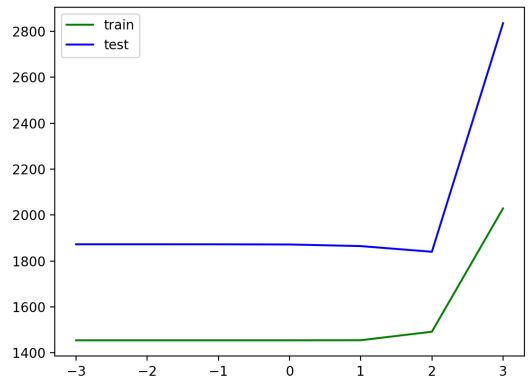


Fig6: pretraining_Lasso Regression with lambda in range[10, 1000]

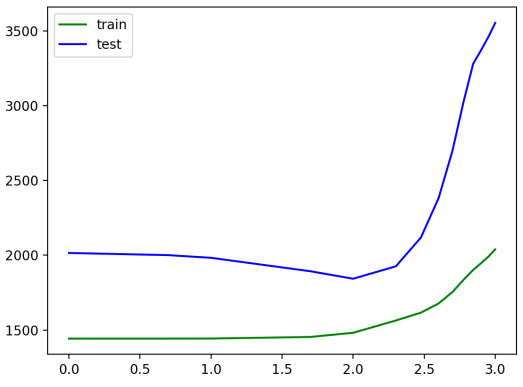
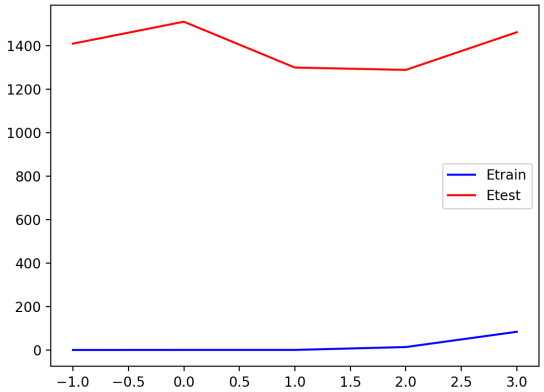


Fig7: pretraining_CART with different min_impurity_decrease



for d is greater than 2, E_train is small while E_test is large, which definitely causes overfitting. One of the reasons must be that our number of pre-training samples is around 100, but for d = 3 we will have 9^3 = 729 parameters to fit, that is our number of degrees of freedom is much larger than our samples. Considering that our training set has around 800 samples, we prefer to choose **d equals to 2** to avoid overfitting.

- **Ridge Regression**: plot the E_train and E_test with different constraints in *Fig3* and *Fig4*. To avoid overfitting, **lambda in region [0.1, 100]** may have a good performance.
- **Lasso Regression**: similar to Ridge Regression, plot the E_train and E_test with different constraints in *Fig5* and *Fig6*. Also see underfitting and overfitting regions, we choose **lambda in region [1, 100]** for further training. We expect to gain better E_val on the training process by using Ridge and Lasso regression, as they add some regularization on weights.
- **KNN**: two parameters to choose: the number of neighbors, the weights on distance. Try **k = [1, 10]** and **weights = ['uniform', 'distance']** in the training set.
- **CART**: there are many parameters that can be set as halting conditions for the decision tree in the model. Here we choose ***min_impurity_decrease*** **in range [0.001, 1000]**, as in *Fig8*. In the training process, we will first train the model with different min_impurity_decrease, and also record the number of leaves. Then to find a good fitting model with adjustment of *max_leaf_nodes* parameter.
- **Random Forest**: choose ***n_estimators*** **(parameter of number of trees) in range [10, 1000]** and ***max_samples*** **(parameter of number of sub features) in range [3, 8]** roughly.
- **Gradient Boosting Regression**: **choose *n_estimators* (parameter of number of estimators) in range [50, 1000]** and **set *max_depth* (depth of each tree) equals to 1**.

According to the distribution of samples, we expect to have a better performance of nonlinear regression models than linear regression models.

Here is a table for the chosen hypothesis that will be trained in the training set:

	preprocessing	**feature extraction**	**parameters range**
constant [baseline]	None	None	None
MLE	standardization	all features[baseline]	None
		four most related features	None
		all features with poly	d = 2
		all features with PCA	c = 7
Ridge Regression	standardization	None	lambda = [0.1, 100]

Lasso Regression	standardization	None	lambda = [1, 100]
KNN[baseline]	normalization	None	weights = ['uniform', 'distance'] k = [1, 10]
CART[baseline]	normalization	None	min_impurity_decreas e = [0.001, 1000]
Random Forest	normalization	None	n_estimators = [10, 1000] max_samples = [3, 8]
Gradient Boosting	normalization	None	n_estimators = [50, 1000] max_depth = 1

3.4. Training Process

We will train linear regression and nonlinear regression models in this part. And find the model with best performance in cross-validation as the final model.

For linear regression model, always assume output as a linear function of inputs:

$$y(x) = w^T x + \varepsilon$$

And also always assume a Gaussian distribution to the error, then the output can be estimated as follow:

$$p(y \mid x, \theta) = N(y \mid w^T x, \sigma^2)$$

our goal is to find the best θ that can estimate the distribution of y well.

3.4.1. MLE

- **Brief introduction:** chooses the parameter θ that finds the maximum likelihood:

$$\theta_{MLE} = argmax_\theta \, p(D \mid \theta) = argmax_\theta \sum_{i=1}^{N} lg \, p(y_i \mid x_i, \theta)$$

which means given the dataset (assume i.i.d) and the parameter we can have the largest probability of the labels. Apply MLE in linear regression model, we will minimize the RSS between ground and predicted results to get coefficients w in linear regression models as below:

$$J(w) = NLL(w) = \frac{1}{2} RSS = \frac{1}{2} \| Y - XW \|^2_2$$

- **Modle:**
  - Use *sklearn.linear_model.LinearRegression* to implement MLE. Will get the coefficients trained from training data and the predicted labels.
  - For MLE with polynomial, use *sklearn.preprocessing.PolynomialFeatures* to create polynomial features. If we have 9 features and do poly = 2, then we will get 9^2 = 81 features in a quadratic formula of the original 9 features.

    - For MLE with PCA, use *sklearn.decomposition.PCA* to project 9 features to lower dimensional space.
- **Reason to choose:** We choose MLE as a **baseline** for linear regression models.
- **Training:** according to the pre-training process, we have chosen four MLE models with different feature extraction methods. Implement **cross-validation** on these four models, and record the mean and variance of error:

mean variance	number of parameters	E_train	E_val
MLE on all features	**10**	**1242.55** 1196.10	**1319.16** 28302.24
MLE on four features	**5**	**1284.45** 476.47	**1309.80** 8132.26
MLE with poly = 2	**82**	**980.36** 426.06	**37719572647394.16** 2.67e+28
MLE with PCA = 7	**8**	**1287.19** 361.73	**1307.01** 7374.10

- **Analysis:**
    - **MLE with polynomial has the worst performance** with a small E_train and an extremely large E_val, which means it is overfitting. The reason may be that the complexity of 82 parameters is too large compared with our training data complexity here.
    - **MLE on four features performs better than MLE on all features** with a higher E_train but a lower E_val. The reason may be that there is more noise on the other five features, leading to a worse performance of MLE on all features.
    - **MLE with PCA equals 7 has the best performance** with a slightly larger E_train but the lowest E_val. The reason why it performs better than MLE on all features may be that the feature reduction reduces some degree of freedom, thus less overfitting. The reason why it performs better than MLE on four features may be that there is still some information on the other five features. And rather than directly ignoring those features, MLE with PCA projects features into a new space and drops some of the noise. Also it has the smallest variance on both E_train and E_val, which means **it is the most reliable model** here compared to others.

3.4.2. Ridge Regression

- **Brief introduction:** Ridge regression is actually MAP(Maximum a Posterior) with a Gaussian prior on parameters:

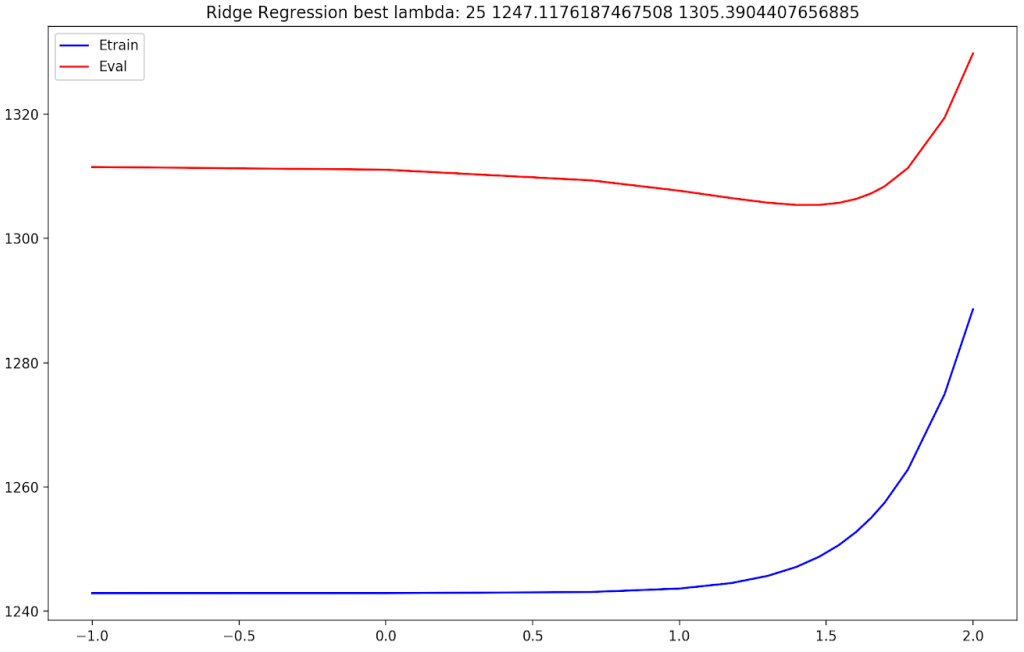
$$\theta_{MAP} = argmax_{\theta}\ p(\theta \mid D)$$
$$p(w) = N(w \mid 0, \tau^2)$$

with this Gussian prior assumption on the parameter, we will have weights gathering around 0 in the Gaussian distribution, thus bound the weights to less fit on noise.
It can also be seen in the objective function as the l_2 **regularization** part:

$$J(w) = RSS(w) + \lambda \|w\|_2^2$$

with a larger λ, the variance of Gaussian distribution will be smaller, thus more weights will have the value close to 0. We have to find a proper λ to reduce fitting on noise and avoid underfitting of the model at the same time.

- **Model:** Use *sklearn.linear_model.Ridge* to implement. It needs to input a λ, and will get the weights of the linear model after training.
- **Reason to choose:** expect that it can improve MLE model by giving some constraints on weights to less fitting on noise.
- **Training:** according to pre-training process, we choose from $\lambda = [0.1, 100]$ using **cross-validation** to see how this parameter affects E_train and E_val.
- **Analysis:**
    - Plot mean of E_train and E_val vs $log_{10}(\lambda)$ on cross-validation below:



    - In Ridge Regression model, **the overfitting and underfitting problem is determined by the parameter** λ. We can see in the figure, with larger λ, there will be stricter constraints on weights, which causes underfitting, as in the figure E_train and E_val both increase after $log_{10}(\lambda) = 1.5$. Otherwise, smaller λ gives looser constraints, that is bigger weights, as in the figure E_train decreases while E_test increases with $log_{10}(\lambda)$ decreasing before 1.5.
    - So to avoid overfitting and underfitting, we choose the best λ with the smallest E_val:
        - **best λ = 25**

- **mean of E_train = 1247.1176187467508**
- **mean of E_val = 1305.3904407656885**

3.4.3. Lasso Regression

- **Brief introduction:** Lasso regression is MAP with Laplacian assumption on prior of parameters:

$$p(w) = \prod_{j=1}^{D} Lap(w_j \mid 0, \frac{1}{\lambda})$$

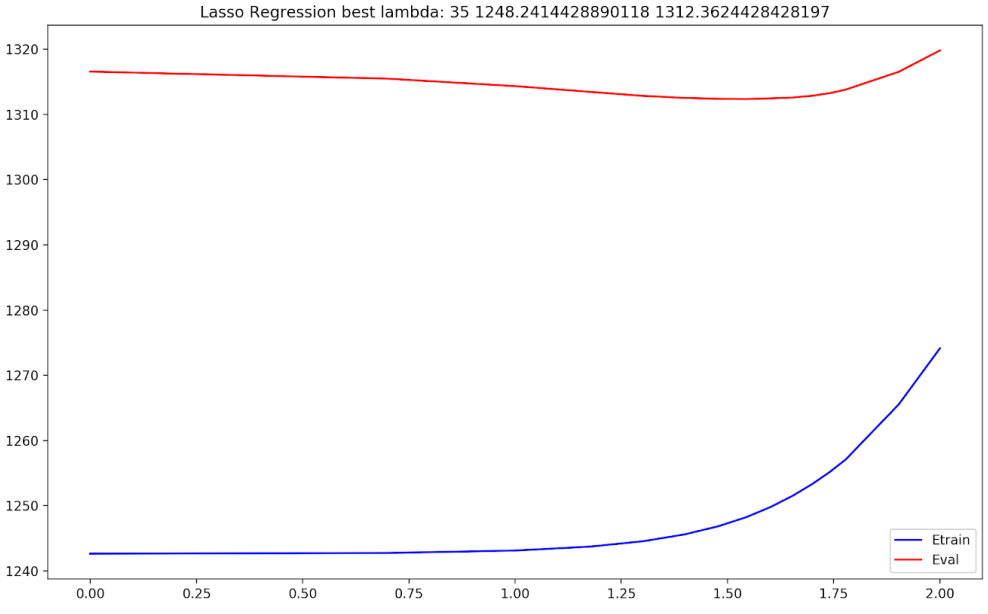
Laplacian prior favors some w_j close to 0 and little w_j far from 0, which can let not important features close to 0 to reduce its effect.

Laplacian assumption will lead to l_1 **regularization part** in objective function:

$$J(w) = RSS(w) + \lambda \|w\|$$

similar to Ridge Regression, λ here will control the degree of constraints on weights.

- **Model:** Use *sklearn.linear_model.Lasso* to implement. It also needs to input a λ, and will get the weights of the linear model after training.
- **Reason to choose:** to **compare with Ridge Regression**, to see how these two regularization methods affect the performance of linear models.
- **Training:** use **cross-validation** to choose λ from [1, 100] and plot the mean and variance of E_train and E_test vs $log_{10}(\lambda)$.



- **Analysis:**
  - E_train decreases as the λ decreases, but the E_val increases after decreasing. In the region that both E_train and E_val decrease, the model is underfitting, which means that λ is still too large to give strict constraints on weights. As λ decreases, the constraints are smaller, the model will have

larger weights and may fit noise in this condition, thus gradually leading into overfitting.

- We prefer to choose the λ when the mean of E_val is the smallest:
  - **best λ = 35**
  - **mean of E_train = 1248.2414428890118**
  - **mean of E_val = 1312.3624428428197**

Unlike linear regression models, we will not make assumptions on distribution of datas or parameters in nonlinear regression models.

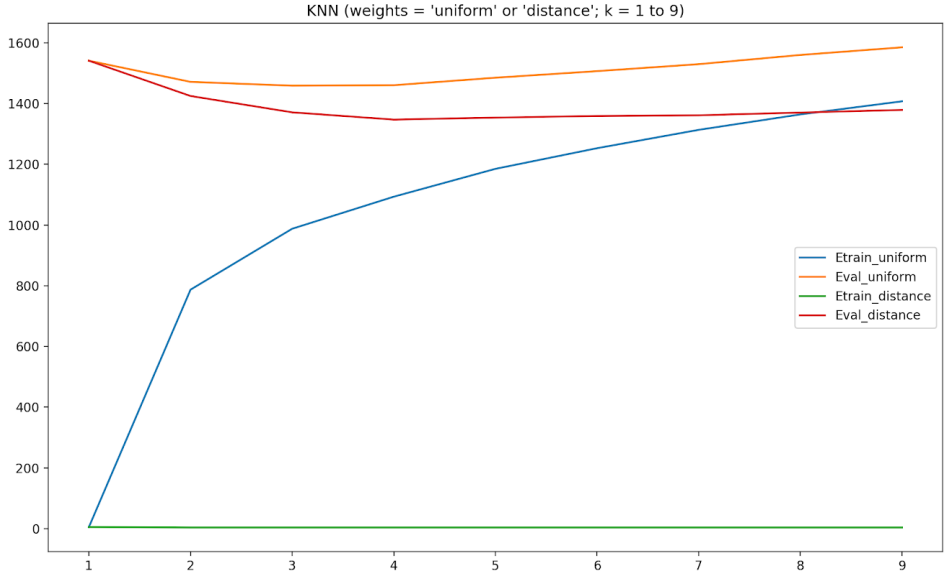
For nonlinear regression models, we mainly focus on **ABM** based models:

$$f(x) = w_0 + \sum_{m=1}^{M} w_m \Phi_m(x)$$

where f(x) is the predicted output, $\Phi_m(x)$ is some functions learned from the data. ABM is going to adapt $\Phi_m(x)$ with training dataset to decrease the loss function.

3.4.4. KNN

- **Brief introduction:** KNN calculates the distance of test data to all the training data, and chooses k neighbors with smallest distances into consideration. In regression models, KNN will get the real number prediction from the k values of those k nearest neighbors.
- **Method:** Use *sklearn.neighbors.KNeighborRegressor*. Has two parameters to specify:
  - *n_neighbors*: the number of nearest neighbors to pick.
  - *weights*: the parameter to weight the distance of different neighbors to the predicted value. It can be set as 'uniform' or 'distance'. 'uniform' means all the values from k nearest neighbors are equally weighted, while 'distance' means that the value of closer neighbors will have larger weights. In the latter one nearby points will contribute more to the regression result than faraway points [1]. So we guess that 'distance' may perform better than 'uniform'.
- **Reason to choose:** as a **baseline** to nonlinear regression model.
- **Training:** use **cross-validation** to choose from n_neighbors = [1, 10], weights = ['uniform', 'distance']. These two parameters are **nested** in the cross-validation set.
- **Analysis:**
  - **For weights = 'uniform'**, E_train is increasing with increasing of the number of neighbors, while the E_val decreases first and then increases. The reason may be that on training sets, with more data included in the computing of predicted value, the value will have more deviation from its original value, thus causing increasing E_train. So to choose the best model, we focus on the minimum mean of E_val. **Mean of E_val in cross-validation always can predict E_out well.**



- However, **for weights = 'distance'**, we find that E_train is extremely small all the time, which benefits a lot from the weights on the value of neighbors with different distances. Compared with the 'uniform' case, data here will have a large weight on close neighbors and a small weight on neighbors a little further away. Although more neighbors are taken into consideration, the predicted result around the point will still be close to its own value.
- Considering the minimum of mean of E_val, we choose:
  - when weights = 'uniform': [baseline]
  - **best k = 3**
  - **mean of E_train = 987.5386810420978**
  - **mean of E_val = 1458.7449486135433**
  - when weights = 'distance':
  - **best k = 4**
  - **mean of E_train = 3.598127118280308**
  - **mean of E_val = 1347.032450572382**

3.4.5. CART

- **Brief introduction:** CART, also called decision tree, is a method to divide the feature space into a binary tree according to the best threshold on certain features. It will result in a set of regions R_m divided from the whole feature space R_0, and each of them weighted with w_m, which is the mean of the value of samples in that region:

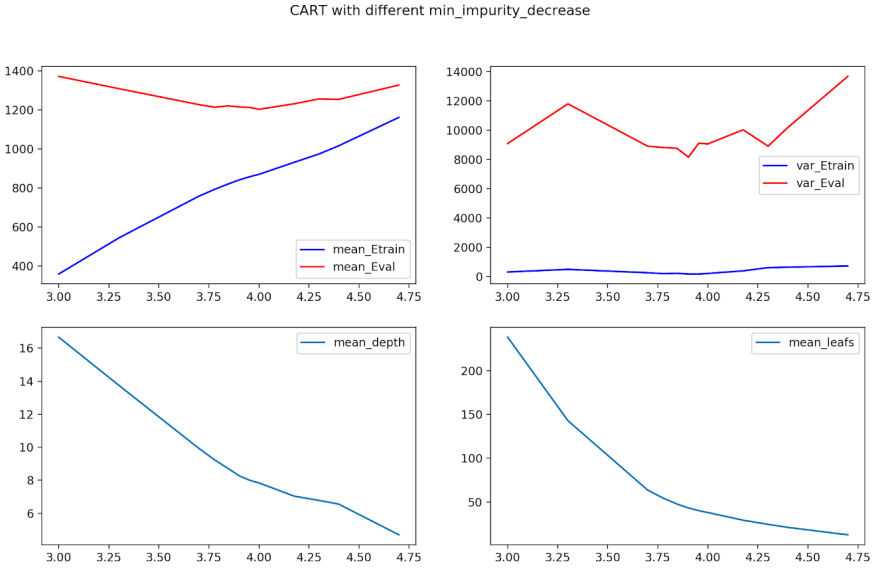
$$f(x) = \sum_{m=1}^{M} w_m \, [\, x \in R_m \,]$$

Here $\Phi_m(x)$ is the indicator function to determine whether the point is located in region R_m.

- **Methods:** Use *sklearn.tree.DecisionTreeRegressor* to implement CART. We will set basic parameters as follows:
  - *criterion = 'mse'*: use MSE to measure the loss on each node.
  - *splitter = 'best'*: in each iteration, choose the features to split with the most decrease in error.

  We mainly adjust these two parameters to control the size of the tree:
  - *min_impurity_decrease*: The error reduction caused by splitting is required to be greater than this value. It serves like a threshold to the degree of improvement.
  - *max_leaf_nodes*: the maximum number of leaves we can get at the end of the training process.

  For the prediction result of the model, we can get the feature importance, which is related to the order of splitting features. With a higher order, the feature will be splitted earlier and will have a greater feature importance in the model.
- **Reason to choose:** to see its performance on the dataset that we don't know the distribution of samples. Serve it as a **baseline** to the tree based algorithm and also will be **compared with KNN** method later.
- **Training and Analysis:**
  - Train CART with different min_impurity_decrease in range [0.001, 1000] in **cross-validation**. Plot the mean and variance of E_train and E_var, mean of depth and leaves of tree with respect to each $log_{10}(min\ impurity\ decrease)$:

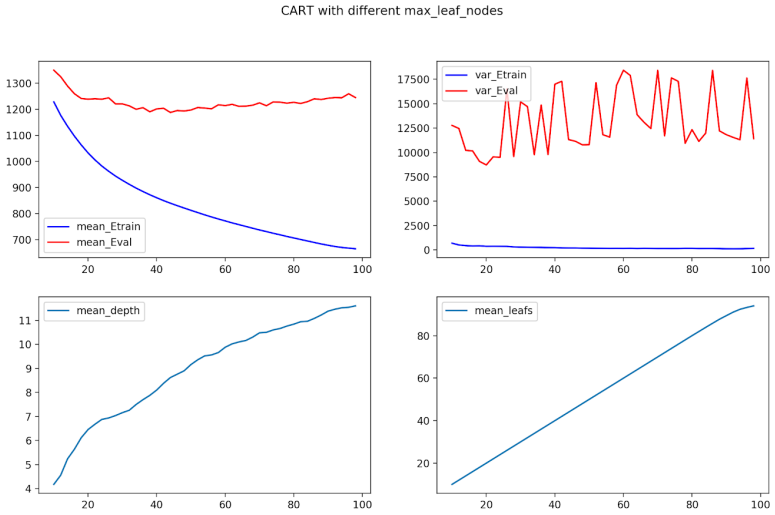


    - **For mean of E_train and E_val**, when $log_{10}(min\ impurity\ decrease)$ is less than around 3.75, the tree has a lower threshold on splitting nodes. E_train decreases while E_val increases, resulting in overfitting. Otherwise, in region that $log_{10}(min\ impurity\ decrease)$ greater than

around 4.00, E_train and E_val both decrease with decreasing of the parameter, resulting in underfitting.

- **Mean of depth and leaves decreases with increasing of the threshold of splitting nodes**, which gives the model less complexity. When the model is in the region between overfitting and underfitting, the number of leaves is in the region[10, 100].
- To find the simplest CART model with lowest number of leaves and not underfitting, **choose *min_impurity_decrease* =** $10^{3.5}$, and then try the number of leaves in range [10, 100].

- Try and plot different *max_leaf_node*s in range [10, 100] in **cross-validation** with *min_impurity_decrease* = $10^{3.5}$:



- With the increasing number of leaves in the terminal, E_train decreases while E_val decreases first and increases later (overfitting).
- Rather than directly choosing the number of leaves with minimum value of mean E_val, compute the **one standard error** bar of the minimum mean E_val, and choose the simplest model with the smallest number of leaves in that region.

- Result:
  - **max_leaf_nodes = 34,**
  - **min_impurity_decrease = 10^3.5**
  - **mean of Etrain = 897.2801627928852**
  - **mean of Eval = 1199.825184427253**

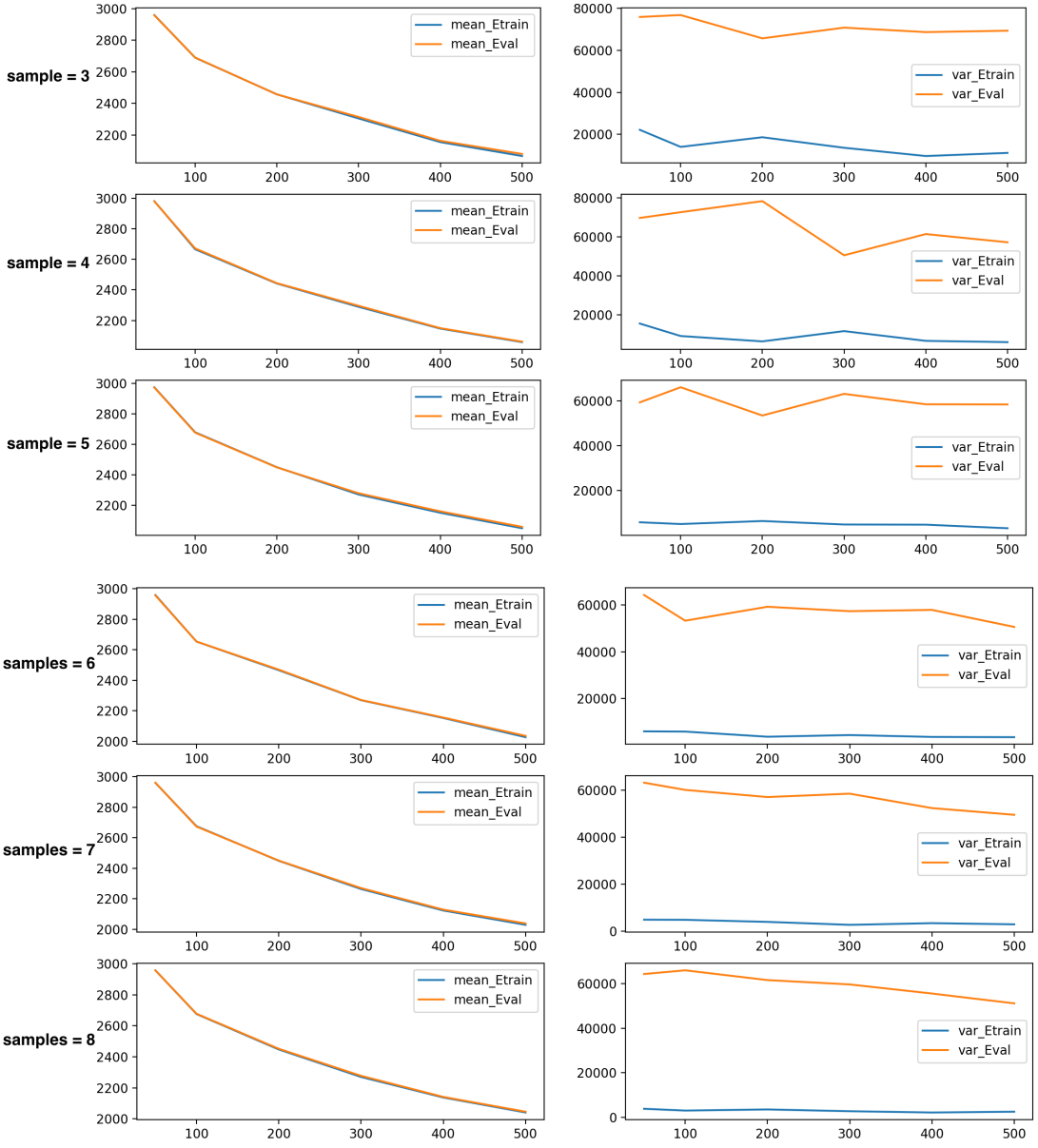
3.4.6. Random Forest

- **Brief introduction:** Random Forest does some improvement from CART by using bagging method to repeat trees several times to reduce the high variance of CART, and also select a random subset of features before splitting each region R_m to reduce the correlation between trees. The result of Random Forest tree will be computed as:

$$f(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

where B is the number of trees repeated, $f_b(x)$ is the result of one true.

- **Methods:** Use *sklearn.ensemble.RandomForestRegressor* to implement. We will use the value chosen from CART for *max_leaf_nodes* and *min_impurity_decrease* here. And to choose the subset features randomly, set *bootstrap = True*. Then adjust these two parameters to find the better performance: *n_estimators* and *max_samples*.
- **Reason to choose:** to see whether it can improve the variance of decision trees.
- **Training and analysis:**
    - Train Random forest with *n_estimators* in range [50, 500] (reduced the maximum number from 1000 to 500 because of the huge cost of running time for larger numbers) and *max_samples* in range [3, 8] in **cross-validation**, and plot the result respective to these two parameters separately:



- **When the number of samples increases from 3 to 8**, mean of E_train and E_val decrease a little (we can see in the y-axis), and variance of E_train and E_val have a significant decrease between samples = 3, 4 and the other greater samples. The reason may be that with random choice of subset, some subset may mostly have unrelated features with price and thus get a large error. With the increasing of samples, those more related features will have increasing probability to be chosen, which will improve accuracy overall.
- **When the number of trees increases**, the mean of E_train and E_val keeps decreasing, and the variance of E_train and E_val has a little decrease based on the overall trend. We can conjecture that, with a larger number of trees, the mean of E_train and E_val will keep decreasing and the variance of them will decrease more too. However, considering the runtime cost, we have no condition to try models with more trees.
- Overall, **the mean and variance of error are larger than the model before**. Maybe one more reason is that in this model, we use bootstrap on the dataset. It will input different dataset resampling from the original set for each tree, and the noise on the datas may cause larger variance.

○ Result: choose the model with minimum mean on E_var:

- ***n_estimators = 500, max_samples = 6***
- **mean of E_train = 2027.4914395771152**
- **mean of E_test = 2036.2111231866531**
- **var of E_train = 3406.0357786012664**
- **var of E_test = 50622.229154614404**

3.4.7. Gradient Boosting Regression

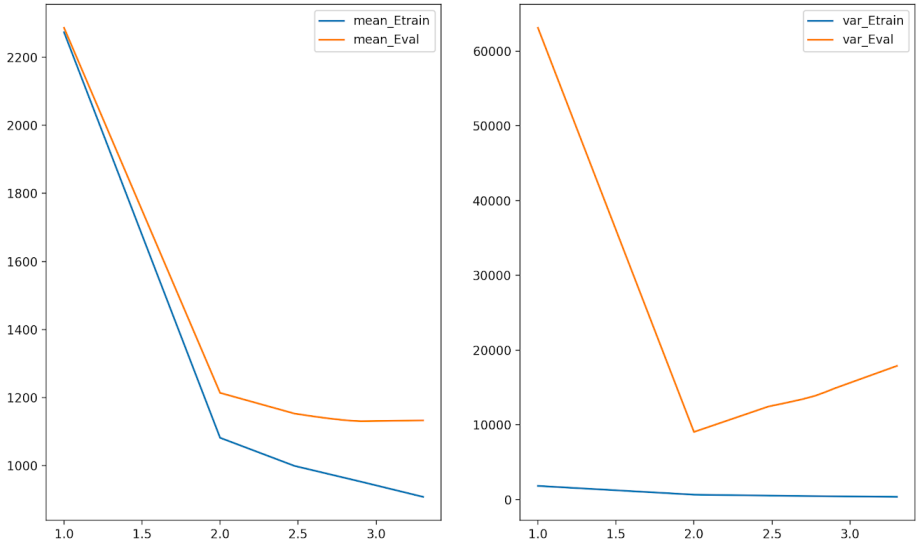
- **Brief introduction:** Boosting is a sequential of simple tree classifiers based on the previous trained model region:

$$f_M(x) = w_0 + \sum_{m=1}^{M} w_m \Phi(x; \gamma_m) = f_{M-1}(x) + \beta_M \Phi(x; \gamma_M)$$

where $\Phi(x; \gamma_m)$ is the classifier of each iteration, and has the weight w_m on each classifier; it can also be written based on the previous model $f_{M-1}(x)$ with the weight β_M. Boosting can reduce the variance of the decision tree. In the regression case, use the Gradient Boosting model, which means the loss function will be improved in the direction of negative gradient.

- **Methods:** Use sklearn.ensemble.GradientBoostingRegressor *with loss = 'ls'* (means choosing least squares as loss function), *max_depth = 1* (means we use one-stage CART as the simple classifier).
- **Reason to choose:** boosting is another method to improve the variance of CART.

- **Training:** try different parameters *n_estimators = [50, 1000]* in **cross-validation**.
- **Analysis:**
    - Plot the mean and variance of E_train and E_val vs $log_{10}(n\ estimators)$ in the next page.
    - **With the increasing number of trees**, the mean of E_train decreases and the mean of E_val increases a little after decreasing. We can conjecture that with a larger number of trees later, the model will more overfit with the increase of complexity of the model. Similar to CART, choose the simplest model to avoid overfitting. Compute the **one standard error** around the minimum mean of E_val, and choose the smallest number of trees in that region.



    - Results:
        - **simplest module with *n_estimators = 400***
        - **mean of E_train = 984.8079793557632**
        - **mean of E_val = 1143.7946063283441**
        - **var of E_train = 501.2560121171906**
        - **var of E_val = 12995.164751386594**

3.5. Model Selection and Comparison of Results

Perform model selection by using **cross-validation**, put models with the best parameters chosen above in the same cross-validation set, record the mean and variance of E_train and E_v:

	mean of E_train	**mean of E_val**	**var of E_train**	**var of E_val**
Constant [baseline]	3708.49	3702.93	4202.66	64575.19
Linear Regression Models				
MLE on all features [baseline]	1242.57	1313.76	1661.71	36752.70
MLE with PCA (c=7)	1287.13	1304.37	526.09	8326.92
Ridge Regression (lambda = 25)	1246.81	1307.01	1591.47	26787.47
Lasso Regression (lambda = 35)	1248.21	1310.16	1597.90	23722.19
Nonlinear Regression Models				
KNN with 'uniform' [baseline] (k = 3)	982.22	1453.98	2518.87	12619.16
KNN with 'distance' (k = 4)	3.38	1345.56	6.44	19377.53
CART [baseline] (leaf = 34) (decrease = 10^3.5)	897.55	1217.91	246.31	12886.31
Random Forest (estimators = 500) (samples = 6)	2278.69	2281.13	3202.17	56884.20
GD Boosting (estimators = 400)	**984.80**	**1143.47**	647.10	16551.32

- **Compare within linear regression models in the order of better performance:**

$$MLE(PCA) > Ridge > Lasso > MLE[baseline]$$

  - **MLE with PCA performs better than the others** here, which indicates that there exists some relation between features and PCA can effectively reduce unrelated composition and increase the performance of related features.
  - **Methods with regularization part, which are Ridge and Lasso Regression, work better than MLE**. List the weights below, we can see that constraints on

weights cause overall decrease of weights on Ridge and Lasso compared with MLE. Also we can find that Lasso has more sparse weights than Riege by setting the smallest absolute value of two weights in 0. Thus we can conclude that the MLE model has fitted some noise in the dataset and some constraints on weights performed by the regularization part will improve this overfitting condition.

	weights on features from [0,8]
MLE	[-2359.21, -579.91, 244.82, 411.60, 18.96, -25.21, -251.09, -96.66 , 1371.84]
Ridge	[-2265.60, -637.72, 205.08, 404.90, 24.51, -25.25, -200.93, -67.79 ,1305.19]
Lasso	[-2359.94, -600.30, 202.29, 382.94, **0.** , **-0.** , -146.02, -22.69,1224.85]

- Another comparison on λ of both Ridge and Lasso between the pretraining and training set: for Ridge, the best λ in the pretraining is around 30 and in training is 25; for Lasso, the best λ in the pretraining is around 100 and in training is 35. **There is a decrease of λ from pre-training to training set**. The reason may be that both the parameter λ of Ridge and Lasso is proportional to the variance of noise, with the increasing number of dataset, the variance of noise will decrease, thus leading to the decreasing of the best λ.

- **Compare within nonlinear regression models in the order of better performance:**

$$GD\ Boosting > CART > KNN\ 'distance' > KNN\ 'uniform' > Random\ Forest$$

  - **For KNN, the 'distance' version has better performance** according to the mean of E_train and E_val than the 'uniform' version. However it will cause a larger variance of E_val, which still makes sense because the predicted value by 'uniform' has less variation than 'distance' in the case of a slight deviation of the position of data points.
  - **For tree based algorithms, GB Boosting performs better than the other two** according to the mean of E_train and E_val. As the conjecture of overfitting conditions for Random Forest, we will not compare its performance here. However, for the variance of E_train and E_val, **both GB Boosting and Random Forest have not improved the variance any better compared with CART.** The reason for Random Forest may be that it has not trained completely with the shortage of data. For GD Boosting, the reason may be that we just use one-stage tree for each classifier.
  - We can take a short look on the feature importance by using tree-based models here. **Features with index [0,1,3,8] have a higher importance value**, which matches our expectation from the plot of the pre-training dataset.

	feature importance
CART	**[0.868, 0.029, 0. , 0.017, 0. ,0. , 0.002, 0.0039 0.079]**

Random Forest	**[0.471, 0.236, 0.008, 0.036, 0.019, 0.004, 0.027, 0.032, 0.161]**
GD Boosting	**[8.72e-01, 3.97e-02, 1.15e-03, 3.85e-02, 7.91e-05, 1.32e-03 , 3.69e-04, 2.77e-03, 4.37e-02]**

  - **CART performs better than KNN on the mean of E_train and E_val**. The reason may be that KNN can't choose the most important features when comparing distance, while CART can choose the best feature to split in each iteration. And we know that in this problem, some features are not so related to the price, so it may cause a higher error when using KNN here.
- **Compare between constant model, linear and nonlinear regression model in the order of better performance:**

$$Overall: \quad NonLR > LR > Constant$$

  - **Performance of nonlinear regression models is better than linear regression and our constant baseline model.** One of the reasons may be that the problem of predicting price is not linear, thus nonlinear regression models will work better. But there is also a case that the prediction of price has too much noise and some features are unbalanced that leads to the difficulty of fitting for linear regression models, as we can see in the figure plotted in the pretraining process that features are scattered into striped shape. So in this problem, we prefer to choose the nonlinear regression models.
- According to the analysis above, the Gradient Boosting Regression model has the minimum mean of E_val and an acceptable value of variance of E_val, so **we choose the Gradient Boosting Regression model with estimators = 400 as our final model.**

4. Final Results and Interpretation

- The final model chosen for this price prediction problem is the Gradient Boosting Regression model with *n_estimators* = 400, *max_depth* = 1.
- Get the final performance on training and test dataset as follow:
  - **RMSE on training dataset = 981.0995397762723**
  - **RMSE on test dataset = 1236.1626024897932**
  - **Features importance**: (mark the importance from high to low)

age	KM	FuelType	HP	MetColour	Automatic	CC	Doors	Weight
8.79e-01	**4.57e-02**	**4.27e-04**	**2.81e-02**	**1.81e-05**	**6.37e-04**	**3.82e-04**	**1.67e-03**	**4.38e-02**
1	2	7	4	9	6	8	5	3

  - We choose this model from 10 other models above. And we have compared its performance with the baseline models in the previous part.

    - The performance in the test set computed by RMSE is similar to the value in the last cross-validation.
    - For the features importance, we can conclude that **features age, KM, HP and weight are more related to the value of price**, while f**eature MetColour are the least related to the value of price**, which makes sense considering the real live condition, the former are the factors that will influence the degree of use of the car and the latter will be less taken into consideration when we predict the value of cars.
- In real life cases, there will be many price prediction problems like houses and cars and others. Ideally, the price of them will be in linear relation with the features like the square meter in the houses problem or the age in the cars problem. However, in the actual cases, it may not be obvious and strictly linear because **the price has a great possibility to float according to more subjective factors**, which will cause large noise on the model. This is the reason why linear regression models perform not so well here. And for nonlinear regression models, their better performance is easy to understand. Just imagine that we want to label the price of a car, we may prefer to find information about the similar cars and then decide the final price around those values. So even though there may be much noise on the data, their values will not float too far away from their usual positions. So **those models focusing more on and training based on the region of the feature space may have better performance on this real case dataset**, such as the nonlinear regression models here.

5. Summary and conclusions

- **In this report**, we trained several different models on the car price prediction problem, and found that nonlinear regression models work better here, among which the Gradient Boosting Regression model has the best performance.
- **In further research:**
    - We want to try more Random Forest models with larger numbers of trees on a larger dataset, to see how it decreases the variance of output.
    - We can also focus more on how to remove the noise from the dataset. By achieving this, we may find that the linear regression models will have a better performance on the price prediction problem. And I'm interested in which degree of noise added to an ideally linear model will cause the performance of a nonlinear regression model better than the linear regression model.
    - We can also research more on how the features importance will influence the performance of KNN and CART to determine the training model more accurately.

6. References

[1] "scikit-learn.org". [Online].
Available:https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-regression

[2] “Comparative Study on Classic Machine Learning Algorithms”. [Online]. Available: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222#:~:text=Decision%20tree%20vs%20KNN%20%3A,KNN's%20expensive%20real%20time%20execution.