Mercedes-Benz Bench Test Time Estimation

Lanjihong Ma, Kexiong Wu, Bo Xiao, Zihang Yu

***Abstract*—Bench tests are crucial for car manufacturers before their cars hit the road and come to the market. Bench test planning is challenging for each manufacturer due to robustness and complexity of the test. There are several hundreds of features affecting the bench test, which introduce a lot of uncertainty to the prediction of bench test time. In this project, we are looking for suitable machine learning algorithms to predict the bench test time. The four models we finally choose are XGBoost, random forests, extremely randomized trees and gradient boosting. We then combine them into one model by assigning a weighting factor to each of them, which can give us the optimal prediction. We evaluate each model by R-square scores, which shows how good the prediction fits our regression. With the prediction from our algorithm, car manufacturers are able to plan their bench tests in a more efficient pattern without lowering the test quality.**

INTRODUCTION

As one of the largest manufacturers of luxury cars, Daimler has always considered safety and efficiency as priorities on its production lines. To certificate the quality of each unique car configuration before they are introduced to markets, engineers have designed a robust testing system. However, it is pretty challenging to fasten the test process due to large amount of potential feature combinations included in these tests. Moreover, it is still uncertain that how each feature would affect the whole bench test time. If there was a competent algorism for predicting bench test time of each car with a given feature combination included in the test, it would be promising to manage the test time and maximize the utility that Daimler has. Our project is relatively abstract because all given features are classified by encrypting into one or two capital letters. While we were doing this project, we all just consider these features as some factors that could alter the bench test time. Thus, the input of our project is feature combinations. Each feature x_n could be either one or zero, which indicates whether a feature is included in the bench test or not. As the input from each data point, the feature combination is an array only made of one or zero. The output for our model candidates is the bench test time according to a given feature combination included in the test. We have tried multiple model candidates, among which we selected best four models, gradient boosting, random forests, extremely randomized trees and XGBoost. We then combined these four models together by assigning a weighting factor to each model. We use this combined model to output a predict bench test time with an R-square score of 0.55642.

I. DATASETS AND FEATURES

We use the data from a Mercedes-Benz Greener Manufacturing competition on the Kaggle website. There are 3367 training data sets, 842 cross-validation datasets and 4209 test datasets. This data contains an anonymized set of variables, each representing a custom feature in a Mercedes-Benz car. For example, a variable 'a' could be 4WD (4 wheel drive), while other variables might represent the headlights, added air suspension, or a head-up display. Variables with letters are categorical while variables with 0/1 are binary values, and there are 376 logistic features and 8 classification features. The ground truth it labeled 'y' and represents the time (in seconds) that the vehicles take to pass the bench test for each variable.

We divide the data into 8 classes, and Fig. 1 shows the feature distribution of class X0.

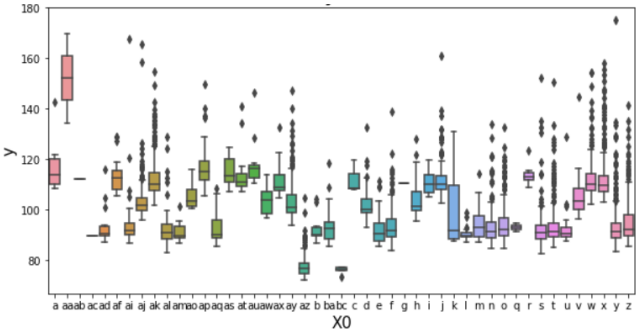

Fig. 1 Feature distribution from X0 class in histogram

We can also visualize the data in another way.

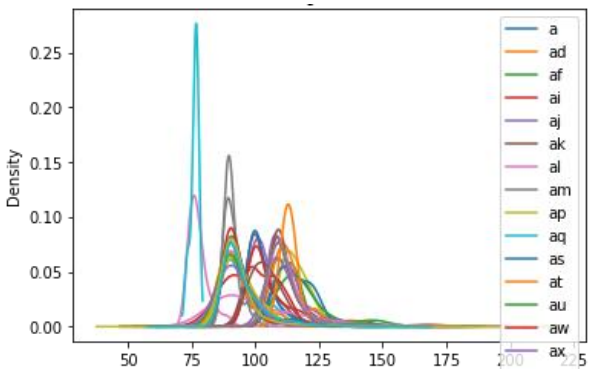

Fig. 2 Another way to display feature distribution

A. *Feature Selection*

Feature selection is a process where we should select those features that contribute most to the prediction variable or output. Having irrelevant features in the datasets can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression. In the feature selection process, we find that there is an abnormal point in the 'y' values, which is 175, as is shown in Fig. 3.

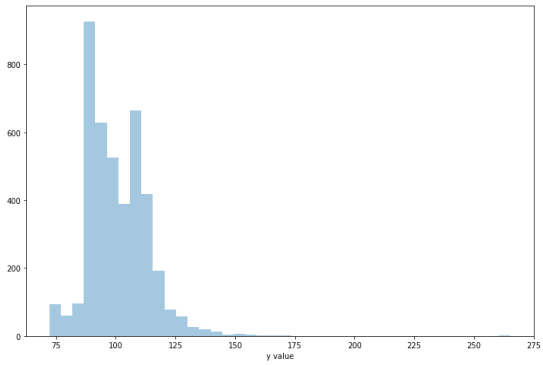

Fig. 3 Distribution of y values

We remove this abnormal point and the distribution of y values looks like a skewed Gaussian distribution, shown in Fig. 4.

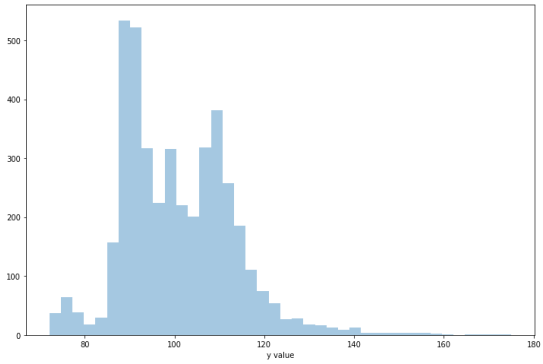

Fig.4 Distribution of processed y values

B. *Feature Extraction*

In this process, we compute the KL divergence for the classification features and set the threshold as 0.1%. The KL divergence formula is

$$D_{KL}(P \parallel Q) = \sum_i P(i)\, log\, \frac{P(i)}{Q(i)}$$

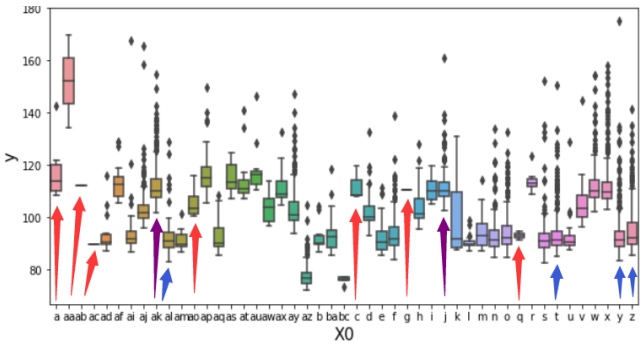

Fig. 5 Distribution before feature combination. Take class X0 as an example, the Red arrows show the sparse signals which are supposed to be filtered out, blue and purple arrows show close distributions that need to be combined.

By implementing the KL divergence, we can find where the distributions of features are too close, shown in Fig.5. And Fig. 6 shows the distribution after combination.

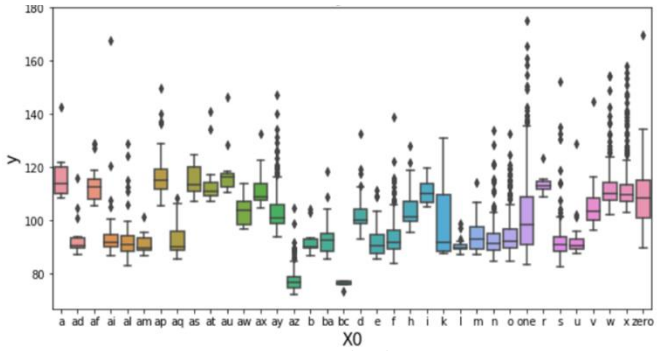

Fig. 6 Feature distribution after feature combination

II. METHODS

To make a better prediction, we choose several algorithms to build our model, the main methods we use are Gradient boosting, Random Forest, Support Vector Machine, XGBoost as well as Extremely Randomized Forest. Besides, we also use other methods such as Stochastic Gradient Decent, Linear Regression, etc. as for comparison and evaluation towards our selected algorithms. And finally, we decide to use a combination of different algorithms as an optimization to achieve the best prediction.

A. *Gradient Boosting*

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

In Gradient boosting, we need to assign a cost function $L(y, F(x))$ and the number of iterations M.

The first step is initialize the model with a constant value

$$F_0(x) = arg\min_{\gamma} \sum_{i}^{n} L(y_i, \gamma)$$

Then implement the iteration for m=1 to M and compute the residuals

$$r_{im} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial\ F(x_i)}\right]_{F(x)=F_{m-1}(x)}, \quad for\ i = 1, \ldots, n$$

Next we need to set a decision tree $h_m(x)$ for the new training set $\{(x_i\,,\ r_{im})\}_{i=1}^{n}$ and compute the following

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

The last step in the loop is to update the function

$$F_m(x) = F_{m-1}(x) + \gamma h_m(x)$$

B. *XGBoost*

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way.

In XGBoost, the objective function contains two part, the loss function and the regularization

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

- Tree Ensemble

  As for the XGBoost algorithm, a tree ensemble often contains a set of classification and regression trees. We can write y value as

$$\hat{y}_i = \sum_k^K f_k(x_i),\ f_k \in \mathcal{F}$$

  Where K is the number of trees, f is a function in the functional space F, and F is the set of all possible CARTs. Therefore, our objective to optimize can be written as:

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- Tree Boosting

  For the supervised learning models, we need to define the objective function and optimize it. Here we assume our objective function as

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i)$$

- Additive Training

  Rather than traditional training methods, in additive training, we train the trees by fixing what we have learned, then add one new tree at a time. The prediction value can be written as

$$\hat{y}_i^{(t)} = \sum_{i=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

  The objective function will be

$$obj^{(t)} = \sum_{i=1}^n \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

  Where g_i and h_i are defined as

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$$

- Regularization

  We define the regularization part as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2$$

Finally we get the objective function

$$obj^{(t)} = \sum_{j=1}^T \left[G_j w_j + \frac{1}{2}\left(H_j + \lambda\right) w_j^2 \right] + \gamma T$$

Where $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$.

For a given structure, the best solution we can get is

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$obj^* = -\frac{1}{2}\sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

III. Result and Discussion

A. Experiment

The experiment we take is to use the cross-validation dataset to evaluate our feature preprocessing part, rather than testing the model accuracy. Our cross validation test takes 4 folds by randomly picking 25% of the raw data to form the cross-validation dataset. The experiment may seem redundant since the model doesn't need to be much more robust. However, the R-square score of this robust ensemble model is 0.64, which is much higher than the best score in Kaggle competition. Then we can conclude that our work on feature preprocessing can effectively fit the model. As for the hyper parameters from the cross-validation test, we do not want them to be stuck in a local minima. Instead of choosing gradient decent, we choose the ensemble methods, Bagging and Gradient boost, which contribute to further optimization. Ensemble learning methods have another advantage in prediction for the test dataset with their model combination. The reason for that is the single model from cross-validation dataset may not be robust enough to fit the real test data.

B. Results

We choose R-square score to be the criterion of accuracy of the model we design. The R-square score, in statistics, is referred to as 'coefficient of determination', which is the proportion of the variance in the dependent variable that is predictable from the independent variables. Model with higher R-square score has the better accuracy. The most general formula is

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where total sum of squares of residuals is

$$SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i {e_i}^2$$

The total sum of squares is

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

Besides, if we divide m for both numerator and denominator, the equation will transform as

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2 / m}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{MSE(\hat{y}, y)}{Var(y)}$$

Where MSE is Mean Squared Error.

Since the project is a discrete problem rather than a continuous problem, the coefficient in discrete problem cannot be so linearly sensitive. Thus a lower R-Squared score is allowed. Before we implement this evaluation, we estimate the R-squared could be about 0.5, which is $\bar{y} = 0.5$ in the equation.

As discuss before, we have tried many algorithms in our project, and the R-square score of these models is shown in Table 1.

We can see from the table above, by comparison we can find that models with top 3 R-square scores are

- Gradient Boosting
- Random Forest
- Support Vector Machine (SVM) with linear kernel

TABLE 1 R-square score of models

Model	R-Square score
SVM(Linear Kernel)	0.5310703895479278
SVM(Polynomial Kernel)	0.4292692789452468
SVM(RBF Kernel)	0.4258513541680711
K-Nearest Neighbors(uniform)	0.4839903962740162
K-Nearest Neighbors(discrete)	0.4500349158417103
Decision Tree	0.30902602134884694
Random Forest	0.5671153256808917
Extremely Randomized Forest	0.3853554827238316
Gradient Boosting	0.6406754701863584

For training dataset and cross-validation, we have highest R-square (which is 0.6435) score using the model combination:

90% Gradient Boost + 8% Forest + 2% Support Vector Machine (linear kernel)

For test dataset, we add XGBoost and achieve highest score assessed by Kaggle. The model combination is

80% XGBoost + 10% Random Forest + 5% Extremely Randomized Forest + 5% Gradient Boosting

Comparing our results with existing results on Kaggle, we found that our R-square is the highest, which means our model can have a better prediction.

IV. CONCLUSION

The major challenge of this project is the curse of dimensionality of each data. We start the project with data processing by generating the distribution for bench test times and then remove the outlier in our dataset. We then generate distributions for eight classification features. We set a threshold to filter out sparse signal and combine close distribution. Considering the R-square scores for each model candidate, we found out the best four models are gradient boosting, random forests, extremely randomized trees and XGBoost. We then assign 5%, 10%, 5% and 80% to each model respectively to get our final model. The R-square score for our final model is 0.55647.

For the future improvements, we believe it is better to use deep neural network for data featuring. The reason for that is our data roughly has the dimensionality of 400. DNN can alleviate our workload on feature extraction by automatically doing it. Moreover, DNN could help us with finding the latent data structure by generating a feature hierarchy.

REFERENCES

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In 22nd SIGKDD Conference on Knowledge Discovery and Data Mining, 2016.

[2] Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine." February, 1999

[3] Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009). "10. Boosting and Additive Trees". The Elements of Statistical Learning (2nd ed.). New York: Springer. pp. 337–384. ISBN 0-387-84857-6. Archived from the original on 2009-11-10.

[4] Friedman, J. H. "Stochastic Gradient Boosting." (March 1999)

[5] Glantz, Stanton A.; Slinker, B. K. (1990). Primer of Applied Regression and Analysis of Variance. McGraw-Hill. ISBN 0-07-023407-8.

[6] Ben-Hur, Asa; Horn, David; Siegelmann, Hava; and Vapnik, Vladimir N.; "Support vector clustering"; (2001); Journal of Machine Learning Research, 2: 125–137

[7] Davies, Alex; Ghahramani, Zoubin (2014). "The Random Forest Kernel and other kernels for big data from random partitions". arXiv:1402.4293

[8] Davies, Alex; Ghahramani, Zoubin (2014). "The Random Forest Kernel and other kernels for big data from random partitions". arXiv:1402.4293