Demand Prediction of Bicycle Sharing Systems

Yu-Chun Yin, Chi-Shuen Lee, and Yu-Po Wong

Stanford University

Bike sharing system requires prediction of bike usage based on usage history to re-distribute bikes between stations. In this study, original data was collected around Washington D.C. area in 2011 and 2012. Original data is processed by several feature engineering approaches based on analysis and understanding of the data. Ridge linear regression, support vector regression (ϵ -SVR), random forest, and gradient boosted tree are applied to predict usage of bike sharing system in an hour based on the conditions of the hour. Test root-mean-squared logarithm error (RMSLE) is achieved at 0.82, 0.34, 0.31, and 0.31 respectively after model parameters optimization and feature selection.

Keywords—Bike-Share System, Support Vector Regression, Radom Forest, Machine Learning.

I. INTRODUCTION

A bike sharing system is a new form of public transportation systems, allowing its users to rent a bike from a location and return the bike to another location. A typical bike sharing system consists of an information technology (IT) system and bike sharing stations. The IT system keeps track of the registration database, payment, and number of available bikes in each station, while a bike sharing station has kiosk(s) connected to the IT system for user registration and bike rental/return. Bike racks are connected to kiosk and automatically release/retrieve bike to/from users. One major task for maintaining a bike sharing system is to redistribute bike to match bike demand at each station. This task requires prediction of bike demand based on the usage record and the current information about the targeting prediction time window. The full prediction problem would be predicting the rental/return counts at each of the stations. In this project, we simplifies the problem down to predict the count of total rental bikes of the entire bike sharing system only.

The dataset used for learning is provided by Fanaee-T [1] collected from Capital Bikeshare program in two years (2011 and 2012) around Washington D.C. The dataset is also hosted on UCI machine learning repository [2], which contains 17379 rows of hourly count of rental bikes (CRB) with the corresponding features including date/hour, seasons, holiday/working-day/weekday, weather, temperature, feeling-temperature, humidity, and wind speed. These features will be manipulated and used to predict the hourly CRB. This report is organized as follows: Section II describes the algorithms used for learning as well as some methods to "engineer" the features to improve the performance; the results predicted by each algorithm along with the optimization methodology is shown in Section III; Section IV discusses some issues we faced during the optimization of model parameters and future works.

II. APPROACHES

Before feeding the raw feature inputs into the learning algorithms, we first study some possible ways to engineer the raw data to potentially increase the correlation between features with target (i.e. CRB in this report).

- (1) *Digitization*: Some features are "discrete" by their nature. For instance, there are 4 seasons in any year, and they are supposed to be independent. We can thus create 4 new digital features representing the 4 seasons: x_{Spring} , x_{Summer} , x_{Fall} , and x_{Winter} . A row with the season of spring corresponds to $x_{\text{Spring}} = 1$, $x_{\text{Summer}} = x_{\text{Fall}} = x_{\text{Winter}} = 0$. The digitalization can be applied to weekday, season, and weather.
- (2) *Periodic Function*: Another trial is to transform the feature "hour" into $\cos(2\pi/24*\text{hour})$ and $\sin(2\pi/24*\text{hour})$, since hour is intuitively a periodic function.
- (3) Conditional Expectation Value Mapping (CEVM): For features with discrete values (e.g. seasons), we can transform them into their corresponding conditional expectation value: $x_i \rightarrow E[y|x_i]$, where y is used interchangeably with CRB throughout this report.

Next, the algorithms used for learning are introduced. The primal optimization problems will be first described. Then the model parameters and their optimization strategies will be discussed.

A. Ridge Linear Regression (RLR)

Primal problem:

$$\theta^* = \arg\min_{\theta} \{ \sum_{i=1}^{m} (y_i - \theta^T x_i)^2 + \lambda \sum_{i=1}^{m} \theta_i^2 \}$$
 (1)
$$\lambda \ge 0$$

$$y = \theta^{*T} x$$

where λ is a penalized coefficient to adjust the trade-off between bias and variance: the higher the λ , the lower the variance in general. The ridge function in MatLab is used to solve RLR problems. Since the computation is very fast, we optimize λ by brute force.

B. Support Vector Machine for Regression (SVR)

Primal problem:

$$\min_{w,b,\xi,\xi^{*}} \frac{1}{2} w^{T} w + C \sum_{i=1}^{l} \xi_{i} + C \sum_{i=1}^{l} \xi_{i}^{*}$$
subject to
$$\begin{cases} w^{T} \phi(x_{i}) + b - y_{i} \leq \varepsilon + \xi_{i} \\ y_{i} - b + w^{T} \phi(x_{i}) \leq \varepsilon + \xi_{i}^{*} \\ \xi_{i}, \xi_{i}^{*} \geq 0, i = 1, ..., l. \end{cases}$$
(2)

where ε controls the width of the allowed ε -insensitive zone, and C is a penalty factor, which determines the penalty for the data sitting outside the ε -insensitive zone. And the dual problem is:

$$\min_{\alpha,\alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + \varepsilon \sum_{i=1}^l (\alpha_i + \alpha_i^*) + \sum_{i=1}^l z_i (\alpha_i - \alpha_i^*)$$

subject to
$$\begin{cases} e^{T}(\alpha - \alpha^{*}) = 0\\ 0 \leq \alpha_{i}, \alpha_{i}^{*} \leq C, = 1, ..., \ell \end{cases}$$

subject to $\begin{cases} e^T(\alpha - \alpha^*) = 0\\ 0 \le \alpha_i, \alpha_i^* \le C, = 1, ..., l \end{cases}$ where $Q_{ij} = K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel function, and $e = [1, ..., 1]^T$ is the vector of all ones.

Details of the derivation of the dual optimization problem and the algorithm to solve the problem can be found in [3]. In this project, we use a publicly available package provided in [4] to solve the SVR problem. Parameter C determines the trade-off between model complexity (flatness) and the degree to which deviations larger than ε are tolerated in the optimization formulation; parameter ε controls the width of the ε -insensitive zone and affect the number of support vectors. Qualitatively, if ε is too small and/or C is too big, the model may lead to overfitting; if ε is too large and/or C is too small, the model may result in underfitting.

To tackle non-linear problems by SVR, kernels are often employed, because they introduce different nonlinearities into the SVR problem by mapping input data x implicitly into a Hilbert space via a function $\phi(x)$ where it may then be linearly separable. While there are many options for the kernels, in this project, we focuses on the Gaussian radial basis function (RBF) kernel because it gives the best predictions. The RBF kernel can be expressed as

$$K(x_i, x_j) = exp(-\gamma ||x_i - x_j||^2)$$
 (3)

where y is the bandwidth of the kernel. The RBF kernel spreads the influence of each training point to an area in the high dimensional space, and γ determines how far the influence of each training point spreads. If y is too small, the influence spreads far, which potentially leads to high bias; if γ is too large, the influence is in a narrow region, leading to loss of mapping to higher dimensional feature space and potentially high variance.

It is well-known that the selection of γ , C, and ϵ will significantly affect the final performance. Furthermore, we observed that the computation time needed to solve the optimization problem is also highly dependent on γ and C. Empirically, we found that the computation time increases rapidly with increasing C, and it can easily take hours to search the optimal parameters without a reasonable initial guess. According to [5], optimal choice of C can be derived from standard parameterization of SVM solution, resulting in $C^0 = max(|\mu_v + 3\sigma_v|, |\mu_v - 3\sigma_v|)$, where μ_v and σ_v are the mean and the standard deviation of the training targets respectively. For the selection of ε , it is well-known that the value of ε should be proportional to the input noise level. And as suggested in [5], the optimal ε can be estimated empirically by $\varepsilon^0 = 3\sigma\sqrt{\ln m/m}$, where σ is the input noise level and m is the number of training samples. While σ is usually estimated from the squared sum of the fitting error of the training data, here we estimate $\sigma \approx 0.5$ given the data is normalized. Fortunately, as will be shown in Section III, the

model performance is not sensitive to the selection of ε as long as ε is small enough. As for γ , empirically we found that the computation time is less sensitive to γ , and the optimal γ value is usually somewhere between 10⁻²~1. As a result, the optimization strategy for SVR is as follows:

- i. Given a set of features F and a small set of training data 3000~5000), estimate $(C^0, \gamma^0,$ $(\mu_v + 3\sigma_v, 0.1, 1.5\sqrt{\ln m/m}).$
- Check if ε^0 is small enough so that the test error does not change with respect to ε .
- Optimize (C, γ) on F around (C^0, γ^0) by grid search to get (C^*, γ^*) .
- iv. Plot the learning curve on $\{F, C^*, \gamma^*\}$ (i.e. training error and test error vs. the training sample size).
- v. Perform feature selection to obtain an optimal subset features F_s .
- vi. Re-optimize (C, γ) on F_s to get (C^{**}, γ^{**}) .

C. Random Forest Method for Regression (RF)

Random forest is an algorithm with ensemble learning method, which constructs multiple decision trees, and the output is the average of the output of the individual tree (regression). This method combines the idea of "bagging" (bootstrap aggregating) and the random selection of features in order to improve the variance in decision tree learning, which has low bias and very high variance. In RF, the tree formation uses classification and regression trees (CART) methodology, which minimizes (4) on the root node and recursively on the child nodes [6][7]:

$$\min_{y} \sum_{i: x_{ij} \geq S} (y - y_i)^2 + \min_{y} \sum_{i: x_{ij} \leq S} (y - y_i)^2$$
 (4)

where index i is for size of training samples in the node and index j is for candidate features in the . S is the critical value used to split samples on a node. The optimization stops with constraints on tree depth or the minimum number of data points in the leaves.

A RF first chooses bootstrap sample D_i from D, the original training data set, and then uses a modified tree learning method that splits with candidate features randomly selected from a subset of the total features. Typically, for a dataset with d features, \sqrt{d} features are used in each split [6][8]. Finally, we will have ensemble of trees and the output is the average of the output of the trees, which reduce the variance. Two key parameters are tree number and tree depth. Small tree depth will decrease the complexity of tree structure that results in high error (bias). Large tree number will improve variance of individual trees, leading to smaller error. Computation time usually linearly increases proportional to tree numbers. In this project, RF is implemented by Weka, a Java-based package available to the public [9].

Optimization strategy for random forest (RF):

- i. Given a set of data F
- ii. Optimize tree number and tree depth to get $(B^*, depth^*)$
- Feature selection with forward search on {F, B*, iii. depth* to get F_s
- Re train model with { F_s, B*, depth*} iv.

D. Gradient Boosted Regression Tree (GBT)

Gradient boosting is an ensemble method, which enhances performance of regression classifier. Each iteration fits a model to the residuals left by the previous classifier. Output is the sum of these classifiers. To prevent overfitting, shrinkage rate, α , is included that the output would be sum of the classifiers weighted by α . Smaller α can control overfitting; however, it requires a large number of iterations, which is often too time-consuming to optimize.

$$L = \sum (y^{i} - f(x)^{i})^{2}$$

$$f(x)^{i} = f(x)^{i} + \alpha(\nabla L)$$
 (5)

where $f(x)^i$ is a set of prediction; L is the error of the prediction. We uses gradient boosting machine along with random forest in our learning model.

III. RESULTS

This section shows results of applying the algorithms described in Section II on the prediction problem of rental bike demand. The performance metric used in this project is the root mean square logarithmic error (RMSLE):

RMSLE =
$$\sqrt{\frac{1}{m} \sum_{i=1}^{m} (log(p_i + 1) - log(y_i + 1))^2}$$
 (6)

where p_i is the predicted CRB. In fact, RMSLE is the root mean square of the logarithm of the ratio between the predicted values and the actual values (i.e. targets). One reason for us to choose this metric is because this metric is also used by a public competition on Kaggle [10], which allows us to compare our results against as a reference. Relative error, error divided by target, can be estimated from RMSLE, e.g. RMSLE=0.3 estimates that relative error = 34%. In the rest of this report, the 30/70 method is used to measure the generalization error for the sake of computational efficiency, unless otherwise specified.

A. Ridge Linear Regression (RLR)

Since RLR has a closed-form analytical solution and only one model parameter λ , the performance is determined by the selection of features. Fig. 1a shows the generalization error (RMSLE in short) before and after feature engineering. Various combinations of the feature engineering described in Section II (1)-(3) were tested, and the CEVM is found to give the best results. However, RMSLE=0.8 is still considered very poor. In Fig. 1b, the average with the standard deviation of the CRB vs. hour is plotted. Even though the hour feature indicates the highest correlation with

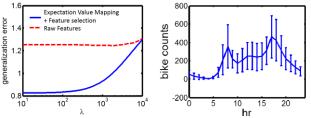


Fig. 1. (a) Generalization error of ridge regression vs. parameter λ . The blue line represents the result after mapping features to conditional expectation values as described in Section II. (3). (b) Hourly average with standard deviation of the count of rental bikes vs. hours.

the CRB among all the features, the correlation is still low (the noise is large). Thus the RLR result suggests that this CRB prediction problem is highly non-linear.

B. Support Vector Regression (SVR)

Since this CRB prediction problem is highly non-linear, we use the RBF kernel in the SVR (both polynomial and sigmoid kernels have been tested and the RBF kernel is proved to be the best). In addition, digitalization of features described in Section I.-(1) is employed to achieve the best performance. Based on the optimization strategy described in Section II.B, we first optimize C and γ around C⁰ and γ ⁰ as shown in Fig. 2a to get C* = 79 and γ * = 0.16. One can clearly see the trade-off between bias and variance on the plot. Owing to the initial estimation of C⁰ and γ ⁰, the optimization by grid search can be performed in minutes.

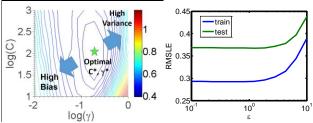


Fig. 2. (a) Optimization of C and γ in the SVR. The contour represents the generalization error given by the 30/70 method. (b) Training and test errors vs. the loss function ε , showing that the performance is insensitive to ε for small ε 's.

Next, the test and training errors are plotted vs. ε in Fig. 2b to verify that the performance is indeed insensitive to ε as long as ε is small enough. Therefore $\varepsilon = 0.1$ is used from now on. Fig. 3a shows the learning curve using (C^*, γ^*) . The minimum test error = 0.361 occurs when the full dataset is trained. From the learning curve, one can conclude that (i) acquiring more training samples and (ii) feature selection may help, since the test error is still decreasing, and there is a gap between the training and test errors. Therefore, forward and backward searches are carried out as shown in Fig. 3b. The forward search is found to achieve the better result, and the resulting selected features are (from high to low importance): hour > working day > season > year > Furthermore, we found that both forward and backward searches reach the same order of the top four selected features, indicating the importance of these four features.

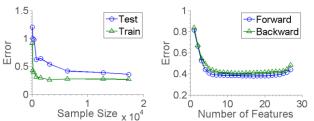


Fig. 3. (a) Training and test error vs. the sample size after the optimization of C and γ . (b) Forward and backward search for feature selection.

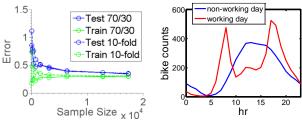


Fig. 4. (a) Learning curves with errors estimated by 30/70 method and 10-fold cross-validation, showing similar trends. (b) The mean count of rental bikes vs. hour for working days and non-working days. Two different patterns are clearly observed.

Next, we select the top 10 features (denoted by Fs) from the forward search and then re-optimize (C, γ) to obtain (C^{**}, γ^{**}) = (158,0.25). With the (C^{**} , γ^{**}), learning curve is re-plotted in Fig. 4a. Test error is improved to 0.35. K-fold crossvalidation (CV) with k=10 is also drawn on the same plot in Fig. 4a, showing that 30/70 method gives similar results to CV. In addition, the minimum generalization error given by CV is 0.34, lower than 30/70 method, due to the larger training sample size being used. To seek for additional tricks to improve the performance, we turn our attention to the top two significant features: hour and working day. Interestingly, as shown in Fig. 4b, a clear patterns are observed: during working days, the peaks occurs at 8am and 5pm, suggesting that the rental bike demand is dominated by commuting workers; while during non-working days, the peak occurs in the afternoon, implying that the demand is dominated by those who go out for fun. Based on this observation, by adding one more feature: hour*working-day into Fs, the generalization error is lowered to 0.33.

C. Random Forest Method (RF)

The maximum number of the features in a node is choen as 5. The feature engineering is applied including digitzation of features and periodic function of hour. Following the optimization strategy described in Section II, we first optimize tree number and tree depth by eveluating the test error estimated by 30/70 method. The result of the optimization of tree number and tree depth is shown in Fig. 5a. With larger tree depth, the generalization error will be smaller because the model complexity increases and the training error (bias) is reduced. On the other hand, the test error decreases with larger tree number because a large number of trees can average out the variance. The performance saturates when the tree number is increased to >20 and the depth is increased to >15. The computation time linearly increase with tree number. For the learning of 100 trees, it take less than one minute; therefore we choose tree number=100 and unlimited tree depth as our optimization parameters. In Fig. 5b, we compares the learning curves between the feature sets before and after feature selection by forward search. The RMSLE estimated by 30/70 method is reduced from 0.33 to 0.32 after feature selection, and the improvement is more pronounced for smaller training sample size. The resulting selected features are (from high importance to low): hour > working-day > season > year > weather > ..., which is consistent with the result from the SVR.

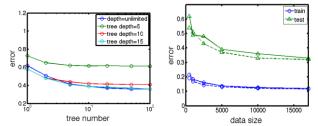


Fig. 5. (a) Optimization of tree number and tree depth. (b) Learning curve by 30/70 on optimized tree number and tree depth. Green line and the dash line are the test error of the given dataset before and after feature selection.

D. Gradient Boosted Tree (GBT)

We tried our learning model with 20 iterations and shrinkage rate of 0.5 along with RF with tree number of 100 and unlimited tree depth. The training error is about 0.0006. The test error for the full dataset is 0.31, which is similar to the result of RF. GBT provides several knobs to control overfitting including tree structure and shrinkage rate α . However, our model is not improved a lot with different tree structure. Optimization of shrinkage rate α needs larger iteration number, which takes more than 2hr for shrinkage rate α of 0.1 and iteration number of 1000.

IV. DISSCUSION & FUTURE WORK

In this section, we discuss the issues we faced when implementing the learning algorithms and optimization, which leads us to the future works.

- From the RLR, we conclude that this prediction problem is highly non-linear, and the resulting RMSLE is high; by employing SVR with RBF kernels, RMSLE is reduced significantly. Interestingly, while applying proper feature engineering techniques (Section II) can greatly improve RLR performance, the same technique does not help that much (RMSLE from SVR is reduced from 0.38 to 0.33 after applying feature engineering). Since RBF kernels already map the features to a high dimensional space, preprocessing the features may not be as intuitive as the case of linear regression, and requires more experience and prior knowledge in the problem.
- While further improving the SVR performance is possible by optimizing the model parameters and more feature preprocessing (e.g. adding hour*working-day), the difficulty arises from the fact that re-optimization of the model parameters is necessary but the optimization is time-consuming (e.g. several minutes) even with a judicious initial guess. What makes it worse is that the performance can be severely affected by some arbitrarily preprocessed features thrown into the program for learning if they are not correlated to the targets, which is hard to know in advance. Therefore, feature selection is often needed to reach the optimal solution. In short, more sophisticated strategies to select model parameters are required for rapid testing on adding new features.
- Although SVR performs well in this problem, the random forest method seems to give a better result.

What makes RF even better is that the optimization of the model (i.e. tree numbers and depth) is straightforward and less computationally expensive. Therefore, without applying sophisticated techniques, we found that RF outperforms SVR.

For future works, we plan to work on two aspects:

- Data: we have identified several critical features, including hour, working day, year, etc. One commonly used trick for improvement is to mix these features by some functions (e.g. x*y, sqrt(x*y), etc.). Another trick we will try is to utilize the time-dependence between the data points. For example, the CRB today may have something to do with the CRB yesterday at the same time.
- Theory: we have found that RF is the most promising candidate to deliver best performance. Hence it is worthwhile to dive into RF and stretch it to the limit. For SVR, we have not fully explored the model selection so far. By the fact that a linear combination of kernels is still a kernel, we can combine various kernels to test the limit of SVR.

V. CONCLUSION

Table I. summarizes the results of the learning models that we have tried. The final performance is measured by 10-fold cross-validation on the full 17,379 data points. The random forest method is found to perform the best, in terms of both prediction accuracy and training time. While each model has been optimized as much as possible, the optimization is limited by training time and our understanding of the algorithms, not necessary to be the true limit of each one. Further study is needed to test the limits, as discussed in Section VI.

TABLE III SUMMARY OF RESULTS

Method	Performance (RMSLE by 10-fold CV)	Computation Time for Training
Ridge Regression	0.82	<1 sec
Support Vector Regression	0.33	~10's secs
Random Forest	0.31	~10's secs
Gradient Boosted Tree	0.30 (not optimized)	~10's mins

Reference

[1] Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

[2]

https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

- [3] A. J. Smola and B. Scholkopf, "A Tutorial on Support Vector Regression," Journal of Statistics and Computing, vol. 14, pp. 199-222, 2004
- [4] Chih-Chung Chang and Chih-Jen Lin, LIBSVM: a library for support vector machines. [online]: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

- [5] V. Cherkassky and Y. Ma, "Practical Selection of SVM Parameters and Noise Estimation for SVM Regression," Neural Networks, vol. 17, pp.113-126, 2004
- [6] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
- [7] Witten, Ian H., and Eibe Frank. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2005.
- [8] Ho, Tin (1998). "The Random Subspace Method for Constructing Decision Forests". IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8): 832–844. doi:10.1109/34.709601.
- [9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.
- [10] https://www.kaggle.com/c/bike-sharing-demand/