CS412: Introduction to Machine Learning
Project Report

Predicting Bike Sharing Demand

Team Members (UIN)

Abhijay Vijaykumar Patne (663324999)
Sabita Acharya (676636765)
Sai Sravith Reddy Marri (671264014)
Shaika Chowdhury (664920919)
Sreeraj Rimmalapudi (662965279)
Vinit Amitabh Kumar (650537234)

COMPUTER SCIENCE
COLLEGE OF ENGINEERING
UIC

Table of Contents

INTRODUCTION

Bike sharing systems have recently been adopted as a new means of transportation by a growing number of cities. These systems have not only automated the process of membership, rental and return; but have also provided a flexible, fast and green alternative for mobility. Users are able to rent a bike from a particular position and return back at another position. This increased flexibility poses the challenge of unpredictable demand as well as irregular flow pattern of the bikes. If we can predict the demand of bikes beforehand, we can prevent imbalance problems like unavailability of bikes or parking docks at the station. In this project, we aim to build such a model for predicting and classifying the number of bike-cycles given a set of input features.

DATA

We obtained our dataset from the UCI Machine Learning Repository [1]. The data is related to the two-year historical log corresponding to years 2011 and 2012 from Capital Bike share system, Washington D.C., USA[2], along with the corresponding seasonal and weather information[3]. It consists of 17379 instances and 16 attributes- record index, date, season, year, month, hour, holiday, weekday, working day, weather, humidity, wind speed, temperature, count of casual users, count of registered users and the total count of users. The values of all the features except the ones to be predicted are normalized.

EVALUATION CRITERIA

We have used several measures in order to evaluate the results obtained from the classification algorithms. In random forest, we look into how accurate the model is in classifying the test data. In logistic regression we measure the accuracy of the model by finding the area under the ROC curves. For support vector machines, we find out the precision, recall and f-score for each bin by using the following formula:

$$Precision = \frac{True\ positive}{True\ positive + False\ positive}$$

$$Recall = \frac{True\ positive}{True\ positive + False\ negative}$$

$$Fscore = \frac{2 * precision * recall}{Precision + Recall}$$

We can assess the performance of a regression model by measuring the difference between the predicted values and the values actually observed. This measure is known as root-mean-square error (RMSE) and it takes the following form:

$$\mathrm{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

APPROACH

Our main objective is to predict or classify the bike rental count hourly or daily based on the environmental and seasonal settings by using regression and classification.
For this purpose, we decided to treat the task as both regression and classification problem with each group member experimenting with a different algorithm for regression and classification. Here we describe some of the techniques we have experimented:

REGRESSION:

1. Linear Regression

We performed linear regression using R. Since our plan was to use linear regression to determine the baseline values for RMSE, we noted its value (RMSE=141.921) without making any changes in the dataset. Once this was done, we found out the features that did not make any effect in prediction by performing ANOVA tests on the models without and with some features removed. Several iterations of this process helped us to conclude that two features, namely "holiday" and "working day" were not significant at 5% level of significance. These features were therefore removed from the dataset and was not used for rest of the experiments.

2. Support Vector Regression (SVR)

We explored both Weka and 'e1070' package [4] available in R for performing support vector regression; but we proceeded with R because of its simplicity and ease of use. 'e1070' package supports four kernels: linear, polynomial, radial basis function and sigmoid function. We split the dataset into two groups where 80% of the data is kept for training and remaining 20% is set aside for testing purposes. The four kernels have the some parameters that need to be tuned and are shown in table 1. Apart from this, we also need to tune the value of slack penalty for the model.

Kernel	**Formula**	**Parameters**
Linear	$u^T v$	(none)
Polynomial	$\gamma(u^T v + c_0)^d$	γ, d, c_0
radial basis function	$\exp\{-\gamma\|u - v\|^2\}$	γ
Sigmoid	$\tanh\{\gamma u^T > v + c_0\}$	γ, c_0

Table 1: Types of kernels with their formula and parameters

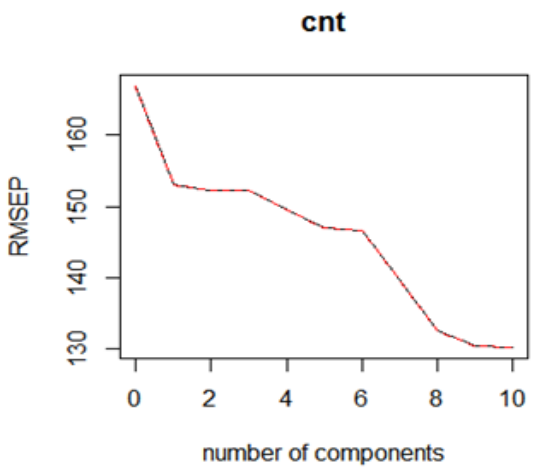
'e1070' package contains a function to tune the parameters of the kernels. But since this took a very long time for the amount of dataset we had, we switched on to perform 10 fold cross validation on the training data and tuned the parameters for each kernel. Once we got an estimate of the parameters, we provided the test data to the model and recorded the best value of the RMSE and the values of the parameters that were used to obtain the best result. The best results are summarized in table 2. It can be seen that radial basis function gives the lowest error as compared to the cases when other kernel functions are used.

Kernel	**Parameters**	**Best value of RMSE**
linear	-	147.28
polynomial	γ=0.08 , d=3, c0=0	136.08
radial basis function	γ=0.005	**95.75**
sigmoid	γ=0.05 , c0=0	63816.57

Table 2: Values of kernel parameters and the best RMSE obtained

3. Principal Component Regression (PCR)

In Weka, Principal Component Regression can be performed by first doing a Principal Component Analysis (PCA). PCA returns a number of ranked components and we can decide on the number of components to proceed with. We can then perform linear regression by using the principal components selected in the earlier step. The same functionality is provided by R through a package [5] called "pls", which allows us to decide on the number of principal components.
We split the data into 80% for training and 20% for testing. 10 fold cross validation was performed on the training data and we plotted the RMSE vs the number of principal components in order to decide on the number of components (shown in figure 1). Since there is no significant decrease in the RMSE for prediction after 8 components, we set the final number of components as 8 and obtain the RMSE for the test data.



RMSE vs number of components for training data

We obtained an error of 148.3, which is slightly higher than that of linear regression. This was expected because PCR is known to work well on high dimensional data and since we had around 10 features, its performance might not have been good enough.

4. Recurrent Neural Network

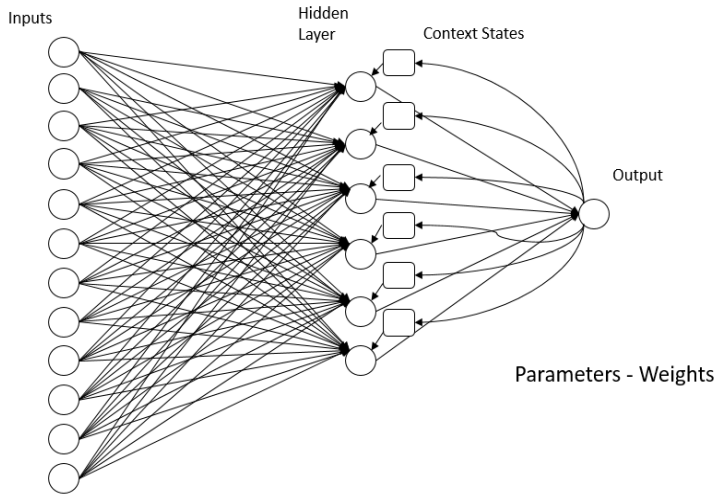
Temporality in the data:

Recurrent neural network (RNN) was used to convey the dynamic temporal behavior in the data. All of the techniques that were used previously never considered the influence of previous samples on the current sample. With this perspective in mind, we went ahead with two techniques – Conditional Random Field (CRF), which can predict with regards to neighboring samples and recurrent neural networks which can model the temporality in the data.

We tried to discretize the output and created labels for the count. This conversion would help us apply CRF to create a classifier for the data set. However, we realized that the conversion, creating a potential function and writing the indicator function would cost us a lot of time that could be used for a different technique. So, due to the time constraint, we discarded CRF and focused entirely on RNN.

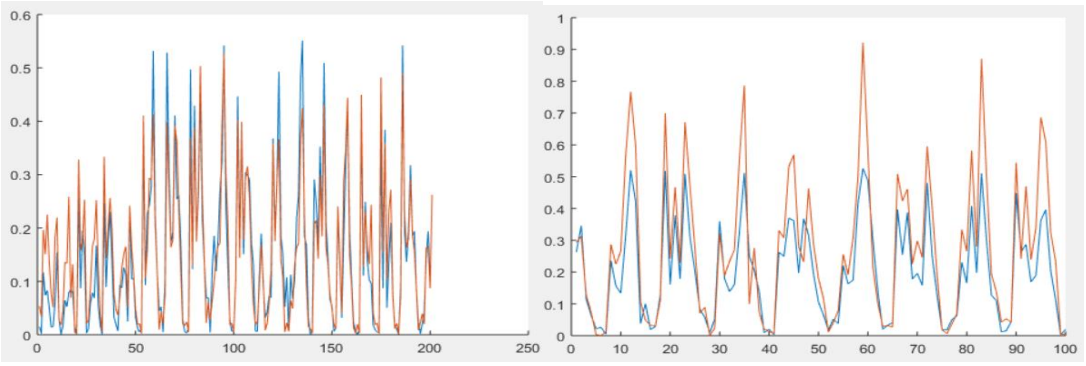
Experiments with RNN:

The plan started with implementing a fully RNN with the notion of creating a complex model, but it was dropped. We wanted to incorporate more than just previous day's condition on the prediction, but we settled on creating the least complex model that can accurately predict to a good degree. Jordan Network, a Simple Recurrent Network (SRN), was chosen to be the architecture of the network.



Jordan Network Architecture

The Jordan Network was coded from scratch to offer maximum flexibility. The data was scaled and normalized for easier computation. We used backpropagation through time with gradient descent strategy for updating weights. The RMSE that we obtained ranged from 0.03 to 0.1. The problem with gradient descent was that it got stuck in the local minima. We then tried to add an additional loop with fixed iterations that keeps generating random initial parameters and builds a model starting with those parameters. We plotted an error function for each of them. The parameters with the least error are chosen, which, if we were lucky, might be an indication of global minima. The RMSE dropped to 0.01 – 0.03 as a result. The downside though was increased time to run it. We thought about improving the speed by making the code run on GPU instead of CPU. We abandoned that since our main objective was minimizing the error. We did not focus on time complexity on any other techniques used before, but the previous process mentioned seemed rather inefficient to us.



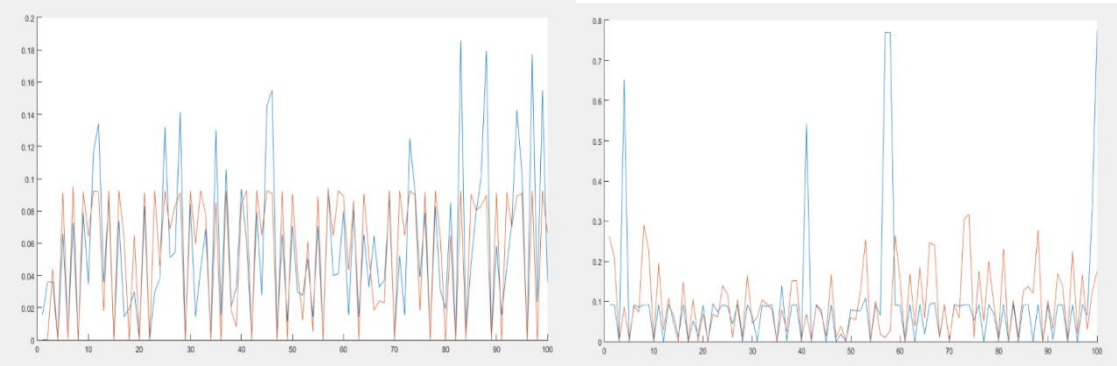
Training plot Testing plot

Gradient Descent with 10000 datasets

Genetic Algorithm (GA) was then selected to replace gradient descent. GA is a global optimizer which could potentially increase our accuracy. It took a lot of time to construct the model, though not as much as gradient descent with additional loop. The RMSE with GA decreased to 0.012. This was a good improvement.

We used the 'ga' function in Matlab. The fitness function was chosen to minimize mean squared error. The two stopping criteria were- a) 5000 Generations and b) Tolerance function – 1e-7. The population in any generation consisted of 500 individuals. The generations and individual were chosen to create a tradeoff between model creation time and accuracy.

We tried predicting a few samples' output and except for some samples, most of them were predicted with great accuracy.



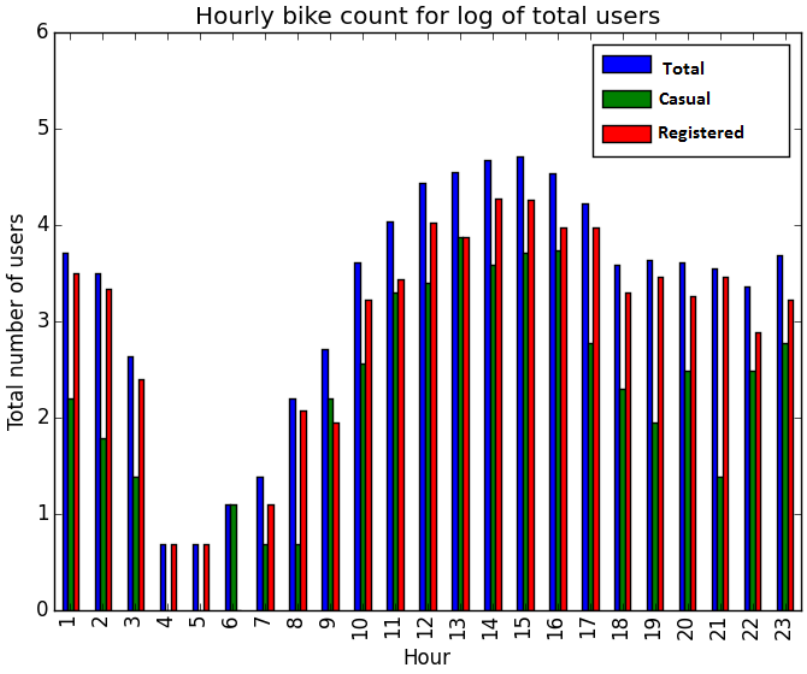
Training Testing

Genetic Algorithm with 1000 datasets

5. Least Square (OLS) Regression

OLS or linear least squares is a method for estimating the unknown parameters in a linear regression model, with the goal of minimizing the differences between the observed responses in some arbitrary dataset and the responses predicted by the linear approximation of the data (visually this is seen as the sum of the vertical distances between each data point in the set and the corresponding point on the regression line - the smaller the differences, the better the model fits the data). The resulting estimator can be expressed by a simple formula, especially in the case of a single regressor on the right-hand side.

Initially, we were predicting value for a single instance, but the accuracy was very low. After observing the dataset, we found out that grouping data according to hour will help us find a better pattern. So we formed 24 bins of data grouped according to hour. Then we calculated weighted coefficients for each bin using OLS module of python pandas.



We calculated RMSE for each of the total, casual and registered users and found out that this classifier performed really better compared to other regression model. RMSE for different outcome were as below:

- RMSE (casual users) = 25.27
- RMSE (registered users) = 67.95
- RMSE (total users) = 81.65

CLASSIFICATION:

We discretized the output and applied the underlying classification techniques.

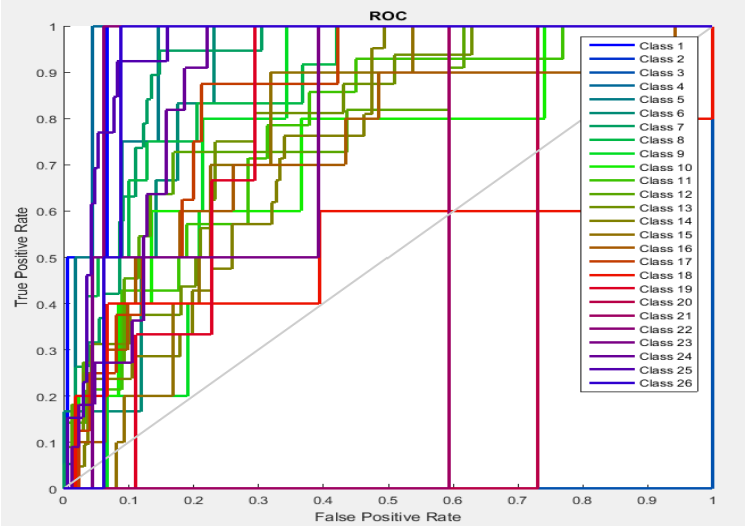
1. **Multinomial Logistic Regression**:
   We applied this technique on our dataset using MATLAB. As logistic regression takes categorical responses/y-variables and our dataset ha 1 d the response *casual* as continuous data type, so we first discretized them into bins. The discretization was done using variable width bins according to the frequency of the response variable. This way we had 26 bins/ranges for casual. We treated these bins as our labels during training which made the logistic regression task *multinomial*. Moreover, as these labels are in order, so our logistic regression problem now becomes *ordinal* and we used this as the *'model'* in MATLAB multinomial logistic regression functions. We divided our dataset into ¾ and ¼ for training and test sets respectively. Using this model, we used MATLAB *mnrfit* function to first find the p-values and coefficients of the training instances. P-value describes whether there is significant relationship between the predictors and the response. A high p-value means that with a change in the predictor, there is no relationship with a change in the response. So high p-values are considered statistically insignificant. Among our features, *holiday* and *working*

day had high p-values, so they were removed from the featureset. Then using these coefficients we predict for a new test instance using MATLAB *mnrval* function. The output of *mnrval* is an n-by-k matrix of predicted probabilities for each multinomial category, where n is the number of test instances and k is the number of class labels.

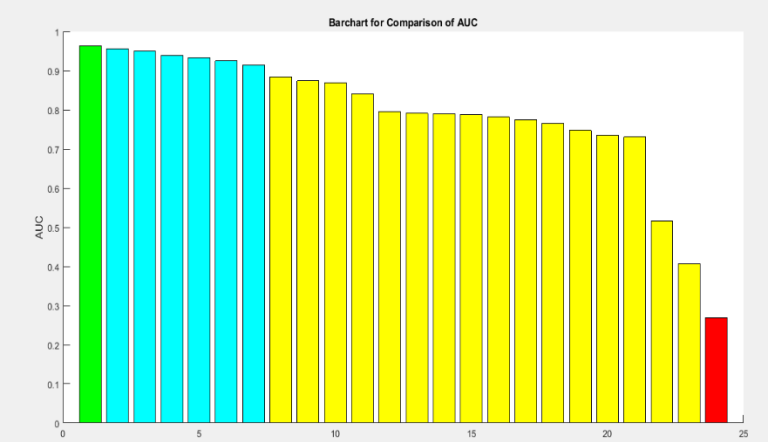
We used ROC curve to evaluate the performance of the classifier. Matlab *plotroc* function, which takes the actual/test labels and the predicted labels as input, was used to generate the ROC curves. A ROC curve is a plot of True Positive Rate against the False Positive Rate which evaluates the classification results on the positive class for binary classification. However, as our classification problem is *multinomial*, so for each class label we generate its ROC curve by considering it the positive class and all the other classes as the negative class. A diagonal line indicates random guessing, that is, it predicts positive class with a fixed probability. So the higher above the diagonal line the ROC curve lies, the better a classifier it is considered. Conversely, if a ROC curve is below the diagonal line, then its performance is worse than random guessing. The area under the ROC curve is a measure of the accuracy of the classifier. So we use the area under the curve (AUC) to compare the ROC curves of the classes. The highest accuracy we got was 0.96 for class 1(2-39), whose ROC curve was the farthest above the diagonal line. The lowest accuracy was 0.26 for class 20(1600-1699), whose ROC curve was below the diagonal line. All the other accuracies were in the range 0.4-0.9.

From the results obtained, we can say that training and testing using Multinomial Logistic Regression with variable width bins worked well on our dataset. However, when initially equal width bins were used, the accuracies were very low. A possible reason for this is the sparsity of the counts within some of the class labels.

The p-values and coefficients table along with the ROC curves and bar chart for comparison of AUC (area under the curve) are shown below.



MLR ROC Curve



Bar chart for Comparison of AUC

feature	season	year	month	holiday	week-day	working day	weather	temp	norm temp	humidity	wind speed
p values	0.0	0.025	0.0	0.78*	0.0087	0.0730*	0.0	0.0	0.0003	0.0	0.0004
coeff.	-0.33	-1.6	0.013	1.3	-0.07	3.8	0.93	19.3	-37.4	2.5	2.8

Table 3: p-values and coefficients (* represents the values above 0.05)

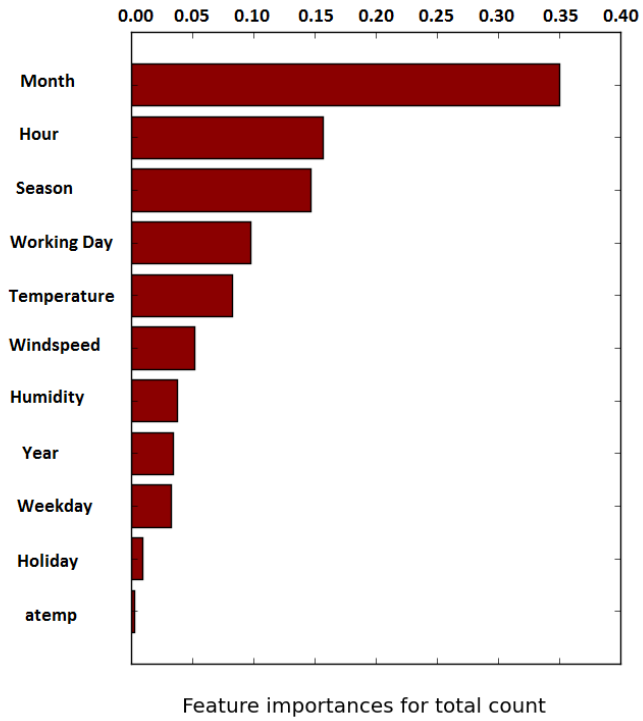
2. Random Forest Classifier

Random forests is a notion of the general technique of random decision forests[9][10] that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set

We used the same strategy used for SVM classifier to train our data with Random Forest Classifier. We have used RandomForestClassifier provided by Sci-kit learn library to train our data and number of estimators were set to 20. To measure classifier performance, we have used k-fold cross validation method with k=10. Our validation scores were as below:

- Minimum: 0.51
- Average: 0.81
- Maximum: 0.95

After training our data, we calculated feature importance out of given features and it is represented in the graph below:



Feature Important for Random Forest Classifier for total users

3. SVM Multiclass Classification

We have tried to convert the regression problem into the multiple class classification problem by discretizing the output class labels into different bins of a fixed size.

When we investigated the data, we came to know that the data was not distributed uniformly throughout, which led to the problem that when we tried to predict for the class labels which were having few counts, there recall was very low.

In order to deal with this problem, we came up with an idea that if the data could be somehow made more uniformly distributed, we could have better recall. So, in order to do that, we first sorted the data i.e. the output class label and then took cumulative frequency of the counts. We then divided the bins in the range of 1000 of cumulative frequency. This strategy enabled us to have a better uniform data distribution, which ultimately improved our precision and recall.

Steps performed

Used Grid-Search to find the best of parameters and tuning the hyper- parameters and fitting the model using SVM algorithm on a cross fold of 10.

Have considered the precision/recall/f-score of each bins rather than the overall classifier accuracy as a measure of classifier performance.

We took the following parameters:

Kernel: RBF, Linear

Gamma: [1e-3, 1e-4] for RBF and none for the linear

C: [1, 10, 100, 1000] for RBF and [1, 10, 100, 1000] for linear.

We found that the best output was through the linear kernel. We calculated this using GridSearchCV and validating the result on the cross validation of 10 folds.

Bins	Precision	Recall	F-Score	Support
0-1000	0.90	0.94	0.92	202
1000-2000	0.86	0.90	0.88	192
10000-11000	1.00	0.69	0.82	224
11000-12000	0.71	0.94	0.81	206
12000-13000	0.93	0.82	0.87	196
13000-14000	1.00	0.71	0.83	207
14000-15000	0.75	1.00	0.86	198
15000-16000	1.00	0.89	0.94	214
16000-17000	0.84	1.00	0.91	197
17000-18000	1.00	0.74	0.85	73
2000-3000	1.00	0.73	0.84	186
3000-4000	0.84	1.00	0.91	182
4000-5000	1.00	0.99	0.99	184
5000-6000	0.94	0.89	0.92	218
6000-7000	0.72	0.93	0.81	197
7000-8000	1.00	0.68	0.81	225
8000-9000	0.73	0.82	0.77	180
9000-10000	0.74	0.83	0.78	195
avg / total	0.88	0.86	0.86	3476

Precision, Recall and F-score for Support Vector Machine Classifier

EVALUATION RESULTS

Out of the methods for regression that we were able to explore, we compare the best results obtained from Linear Regression, Support Vector Regression and Principal Component Regression in table 3. The best value is obtained by Support Vector Regression with radial basis function kernel (95.75). Recurrent neural network performed comparably well with the given data. We also used Ordinary Least Square Regression in order to predict the bins in which the given data falls. RMSE of 81.65 was obtained for this method. Since the rest of the regression techniques predict the counts and not bins, we do not make a direct comparison of the result obtained from Ordinary Least Square Regression with the other techniques. Nevertheless, the low value of RMSE indicates that it is a significant technique for making predictions. Due to time limitation, we were not able evaluate the performance of other regression techniques in predicting bins.

Model	RMSE
Principal Component Regression	148.3
Linear Regression	141.92
Support Vector Regression	95.75
Ordinary Least Square Regression	**81.65**

Table 4: Comparison of the RMSE values obtained from regression

LEARNINGS

This project was a huge learning opportunity for us and we tried to explore as many techniques as we could during the allocated time. We applied different algorithms on real world data and also explored techniques for tuning parameters and measuring efficiency. Apart from this, we also learnt the following:

- We found out that data transformation plays an important role in increasing the accuracy of the model. For example, while performing classification, we grouped the data into bins based on several criteria (cumulative frequency, hour) and the noticed that the performance varies according to the chosen criteria.
- Identifying significant features not only reduces the time taken for execution but also improves the accuracy of the model.
- We do not need to restrict ourselves to classification or regression. Based on the problem definition, we can reduce our problem to an equivalent multiclass classification problem which might be easier.
- Complex methods are not always better. Even though Linear Regression is simple when compared to Principal Component Regression, it produced comparable results for our dataset. The RMSE from linear regression was also lower than that obtained by several kernel functions is SVR.
- Most of the classifiers, regressor we tried were already implemented in the respective programming languages. But the data demanded temporal classification, so we went out of our way to build RNN Classifier from the scratch using Genetic Algorithm.

REFERENCES

[1] Bike Sharing Dataset. Retrieved from UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

[2] Department of Human Resources. Retrieved from dchr.dc.gov/page/holiday-schedule

[3]The weather. Retrieved from freemeteo.com

[4]David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. e1071: Misc Functions of the Department of Statistics (e1071), TU Wien, 2014. R package version 1.6-3.

[5] Bjørn-Helge Mevik, Ron Wehrens, and Kristian Hovde Liland. pls: Partial Least Squares and Principal Component regression, 2013. R package version 2.4-3.

[6] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications. Hillsdale, NJ: Erlbaum.

[7] Design of Adaptive Filter Using Jordan/Elman Neural Network in a Typical EMG Signal Noise Removal, V. R. Mankar1 and A. A. Ghatol2

[8] Applying Genetic Algorithms to Recurrent Neural Networks for Learning Network Parameters and Architecture. O. Syed, Y. Takefuji

[9] Ho, Tin Kam (1995), Random Decision Forests, Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

[10]Ho, Tin Kam (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(8): 832–844. doi:10.1109/34.709601