Bike Sharing Demand

■ **Huizhe Wang**

ABSTRACT

Bike Sharing Demand[1] Kaggle Competition took place for the Capital Bikeshare program in the city of Washington, D.C three years ago. This project chose this competition as topic to predict the count the total count of bikes rented during each hour covered by the test set, using only information available prior to the rental period. This project used basic features provided by dataset to propose new features, then used two machine learning models - Random Forest, Gradient Boosting and the combination of the two models, one deep learning model - Long short-term memory model to predict count. The solution is evaluated with RMSLE. The best result is of 0.36618.

Keywords: Bike Sharing Demand, Random Forest, Gradient Boosting, LSTM, RMSLE

1 Introduction

1.1 Background

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis.

Currently, there are over 500 bike-sharing programs around the world. There are many benefits of Bike Sharing systems: environmental benefit, extends range of transit system, increase viability of other modes, and benefit local business, etc. For example, the more peoples use bikes as their trip mode, the less car used in daily trips. It has a great effect on greenhouse gas emissions and reduce pollution.

Some researchers have categorized the evolution of bike share systems into four "generations" (Parkes, Marsden, Shaheen, & Cohen, 2013). The first generation is the White Bikes program in Amsterdam launched in 1965, which characterized by no payment or security functions (Fishman et al., 2013). Compared with the first generation, the second generation added a coin deposit "system" which was similar to trolleys at a supermarket. The first two generation's common problem is security. The third generation of bike share systems are characterized by dedicated docking stations (in which bicycles are picked up and returned), as well as automated credit card payment and other technologies to allow the tracking of the bicycles (Shaheen, Cohen, & Martin, 2013). The features of fourth-generation systems are not quite so clear, but are said to potentially include dock-less systems, easier installation, power assistance and transit smart card integration (Parkes et al., 2013). [2]

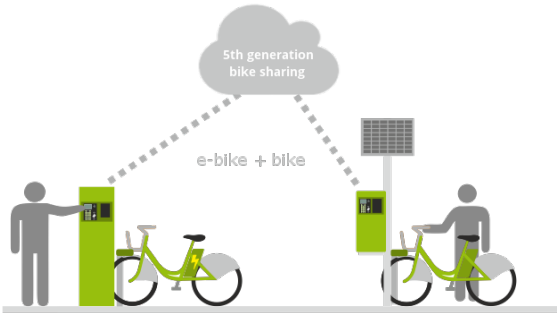

fig 1. the old generations with fixed location

fig 2. the newest generations without fixed location

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

With the popular of bike sharing systems. There are new issues should be solved. There is little bikes to be used in snow days. Bikes always park unorderly to affect the appearance of cities. Put over much bikes in the sparsely populated regions is a waste of resource. With predict the number of bikes using in a given condition: bike renting company would be able to provide a better service to customer, the traffic management teams would be able to plan it better.

fig 3. the unorder packing

1.2 Given Datasets

For the competition, two datasets are provided with the information about hourly rental data spanning two years, where the training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month. The training dataset contains the following features:

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather -

1. Clear, Few clouds, Partly cloudy, Partly cloudy
2. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3. Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4. Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed

casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

1.3 Submission and Evaluation

For this competition, and in order to participate in it and get a scored value of the submission, a CSV file has to be generated containing the respective timestamps present in the test set along with the correspondent total number of rentals prediction.

Kaggle evaluates the submissions using the Root Mean Squared Logarithmic Error (RMSLE). The RMSLE is calculated as:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

where n is the number of hours in the test set, pi is your predicted count, ai is the actual count and log(x) is the natural logarithm.

For consistency, the same measure of error in the predictions is going to be used for the purpose of this work.

1.4 Competition Notebook

Before start my own project, I did some surveys about what kagglers did. The github link is: https://github.com/huizhewang/INFO7390/blob/master/Kaggle%20Notebook.ipynb.

The summaries are as follows:
By the end date of this competition, 3252 teams from all over the world provided their solutions. There are about 90% teams using Random Forest and Gradient Boost model to do prediction, with little team using other models, for example, Neural Network. There are two reasons for this phenomenon: First, the competition was held in between 2014 and 2015. Neural Network was not much popular as now, most teams like choose Neural Network as their model; Second, the result of Neural Network was not good, once these teams found a better model, they will replace Neural Network.

There is a good conclusion in Wang Wen's project [2], here will using his conclusion to show the works these teams did. The Table is shown as Table 1.

Table 1 Summary of projects in Kaggle competition

Reference	Focus	Modeling Methodology	Evaluation Methodology	Conclusion	Details
Yin et al.	Optimize predictive model with smallest RMSLE	(1)Ridge Regression (2)Support Vector Regression (3)Random Forest (4)Gradient Boosted Tree	RMSLE by 10-fold crossvalidation	Random Forest is found to perform the best which is 0.31.	Good RMSLE Data Preprocessing: (1) Digitization (2) Periodic Function
Du et al.	Optimize predictive model with	(1) Basic Linear Regression (2) Generalized Linear Models with Elastic Net	RMSLE by 10-fold crossvalidation	Two of the Tree based models is found to perform	Relative big RMSLE compared with

	smallest RMSLE	Regularization (3) Generalized Boosted Models (4) Principal Component Regression (5) Support Vector Regression (6) Random Forest (7) Conditional Inference Trees		the best: CTree: 0.46 Random Forest: 0.50	Yin et al. group. Data Preprocessing: (1) Digitization (2) Add some features (3) Remove some feature
Lee et al.	Optimize predictive model with smallest RMSLE	(1) Neural Network (2) Poisson Regression (3) Markov Model (4) Mean Value Benchmark	RMSLE	Neural Network is found to perform the best which is 0.49. (// Not building random forest)	Relative big RMSLE compared with Yin et al. group.
Patil et al.	Using R code to build predictive models using Random Forest	(1) Random Forest (2) Enhancing RM model using TuneRF (3) Conditional Inference Trees (4) Generalized Boosted Models	RMSLE	Random Forest with TuneRF performance is 0.49	Relative big RMSLE compared with Yin. Good data exploration
Jing et al.	What factors affect the bike sharing system operation	(1) Multiple linear regression (2) Poisson and Topology method	P values	Variable atemp(temperature people feels) is the most important variable	Good visualization: 3D graphs
Ray	Using R code to build predictive models using Random Forest	(1) Random Forest	RMSLE	Random forest model with RMSLE 0.38675 on Kaggle leader board top 5%.	Good Score Three methods to improve model Using data visualization and Decision Tree to help create new variables
Wayne Liu	Using R code to build predictive models using Regression and Generalized Boosted Models	(1) Regression (2) Generalized Boosted Models	RMSLE	Score 1.38378 on Kaggle leader board (// Error rate too high)	Not as good as Ray Good data exploration

Based on the case study of modelling algorithms in similar project of Kaggle competition, four of six cases found random forest perform best, Du et al.② found tree based models show best performance, the error rate of which are Ctree: 0.46 and random forest 0.50. Lee et al. ③ found Neural Network performs the best with 0.49 error rate, but she did not build random forest model. Beside them,

Ray's④ random forest model gets top 5% in Kaggle Competition leaderboard.

These surveys from kaggles literatures gave some good points:

1. Feature temp and atemp have high correlation, I need only analysis one feature in my own project to avoid multi-collinearity.
2. The datetime column can be used to extract data like the month, day, hour which can be used in model for making better predictions.
3. Random Forest and Gradient Boost model could get good result.
4. Dummy variables and scaling variables are good ideas.
5. Remember handle the output, let them equal or bigger than 0.
6. Choose RMSLE which kaggle provided as the only standard, it will be easy to compare the predict result.

There will be some extention in my own project:

1. Except analysis the basic features, this project will create additional features in data exploration to give more categories for the project.
2. Most kagglers predict count directly. However, Adding the value of registered and casual will equal to count. this project will use two ways to predict count: predict count directly, predict registered ad casual separately and then get the value of count.
3. Even though Random Forest and GBoost Model really predict good than Neural Network. However, this project is a final project in course INFO 7390, the propose is learn how to use both machine learning and deep learning. In this project, this project still uses one kind of Neural Network to predict count as professor suggest.

2 Data Cleaning and Exploration

2.1 Data Cleaning

The training set is composed of the first 19 days of each month and is provided with the response variables, it has then 10,886 labeled observations. The testing set is the complement of the training set, that is, it is composed of the days from the 20th to the end of the month for each month. It has then 6,493 unlabeled observations.

After get the two dataset, this project obtain some simple descriptive statistics from it.

```
: train_df.shape
: (10886, 12)
: test_df.shape
: (6493, 9)
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.026364	0.685284	1.418427	20.23086	23.655084	61.886460	12.799395
std	1.116174	0.160223	0.464424	0.633839	7.79159	8.474601	19.245033	8.164537
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900

fig 4. descriptive statistics of original dataset

From these statistics it is important to notice some things:

1. In average, there the number of registered rentals is more than 4 times the number of casual rentals.
2. The "feel like" temperature is in average three degrees greater than the temperature given; nevertheless both measures are distributed fairly similar.
3. The seasons appear to be equally distributed on its four values. This is an indicator that we have a complete sample for the 2 years.

Based on the features given, the following step is doing data cleaning:
From the timedate feature included in the given dataset, This index spans from 2011-01-01 00:00:00 to 2012-12-19 23:00:00 with 147 missing observations. Moreover, in order to use this data more conveniently, the following features were derived:
• hour: Hour of the day. An integer between 0 and 24.
• day: The day of the week. An integer between 0 and 6 where 0 is Monday and 6 is Sunday.
• month: The month of the year. An integer between 1 and 12.
• DOW: The specific day in the timedate. An integer between 1 and 31. • month: The specific month of in the timedate. An integer between 1 and 12.
• year: The specific year. An integer with values in [2011, 2012].

After the 'datetime' feature, there are some mistakes in 'workingday' and 'holiday' features. For example, tax day is holiday, however, users need to work at that day.

2.2 Data Exploration/Visualization

Based on the features given, the following feature were thought to be useful in the current work. It is important to mention that all the basic features considered are being presented below:

Basic Features
For this part, this project will analysis the relationship between target feature and the features dataset provided one by one.

2.2.1 the relationship between season feature and count

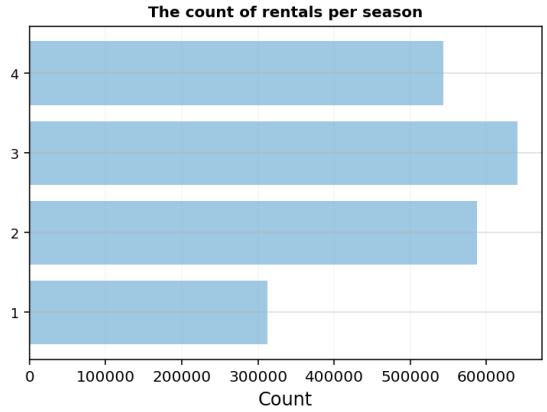

fig 5. relationship between season and count

As dataset introduction said, season - 1 = spring, 2 = summer, 3 = fall, 4 = winter. This figure shows that user prefer to use bike sharing systems in summer and fall. The reason could be the weather feature, there are least rental in spring because the spring is cold Washington, D.C.

2.2.2 the relationship between weather feature and count

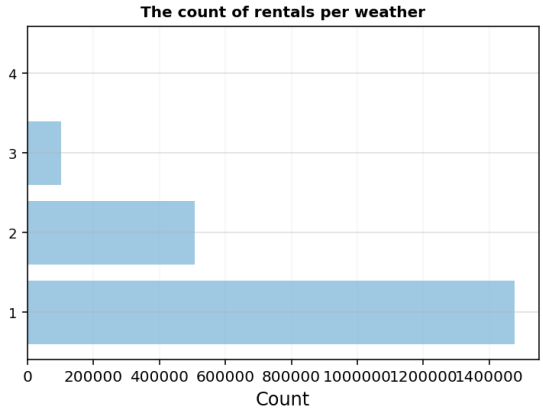

fig 6. relationship between weather and count

As dataset introduction said,
1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

This figure shows that users choose bike sharing systems as their transportation when the weather is Clear, Few clouds, Partly cloudy, Partly cloudy. The weather feature has very important influence of count.

2.2.3 the relationship between hour/weekingday feature and count

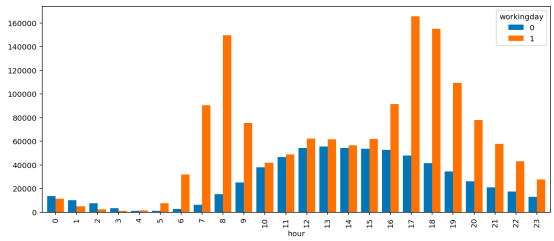

fig 7. relationship between hour/weekingday and count

This figure shows the following information:

1. riders prefer to using bike sharing systems on working day.
2. On working day, the peak of using bike sharing systems are 5:00 - 6:00 am and 4:00 - 8:00 pm, because they are the rush hour.
3. On no-working day, the peak of using bike sharing systems are 11:00 am - 5:00 pm.

2.2.4 the relationship between temp feature and count

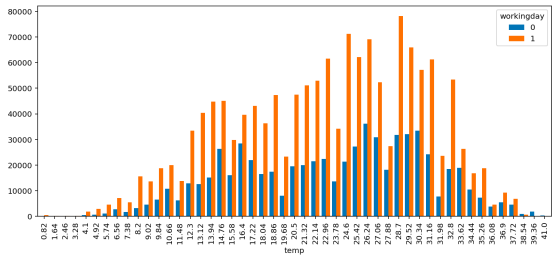

fig 8. relationship between temp and count

This figure shows that if the value of temp is between 28 Celsius and 31 Celsius, users prefer to use bike sharing system.

2.2.5 the relationship between wind speed feature and count

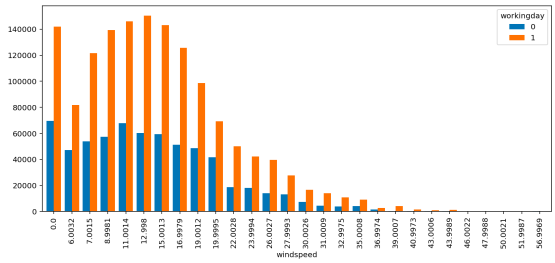

fig 9. relationship between speed and count

This figure shows that if the value of windspeed < 20, users prefer to use bike sharing system.

2.2.6 the relationship between humidity feature and count

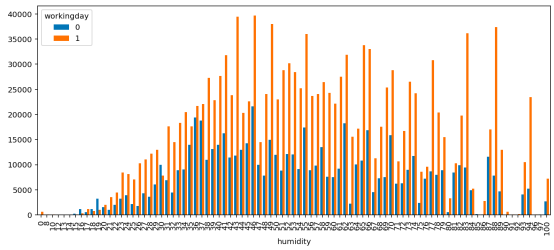

fig 10. relationship between humidity and count

This figure shows that if the value of 43< humidity < 60, users prefer to use bike sharing system.

Additional Features

As mentioned above, the basic features are not enough for this project. this project creates additional features to make my models work better. I will create 3 features which will based on features hour, workingday, temp, windspeed, and humidity.

2.2.7 create rush hour feature as peak

choose the basic features hour and workingday to create a new feature. As mentioned above: when on workingday, if at 8:00 or at 12:00 or between 17:00 - 18:00, set the new feature as 1, otherwise set it as 0. Moreover, when on the weekend, if between 11:00 - 17:00, set the new feature as 1, otherwise set it as 0.

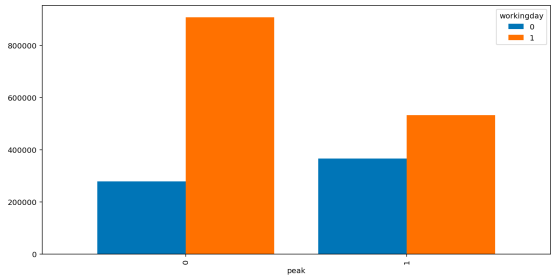

fig 11. relationship between rush hour and count

This figure shows that the new feature has an important effect on count. It means the feature could be used in input values.

2.2.8 create ideal environment feature as ideal

choose the basic features temp and windspeed to create a new feature. As mentioned above, when on the workingday, if the value of temp between 28 Celsius and 31 Celsius and windspeed less than 20, set the new feature ideal as 1, otherwise set it as 0.

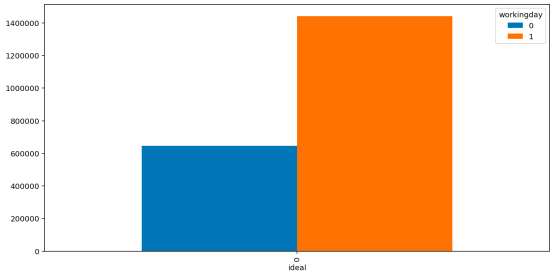

fig 12. relationship between ideal and count

This figure shows that the new feature has an important effect on count. It means the feature could be used in input values.

2.2.9 create sticky humidity feature as sticky

choose the basic features humidity and workingday to create a new feature. As mentioned above, when on the workingday, if the value of humidity stickier than 60, set the new feature ideal as 1, otherwise set it as 0. If it is not workingday, most people choose stay at home or other way to go out.

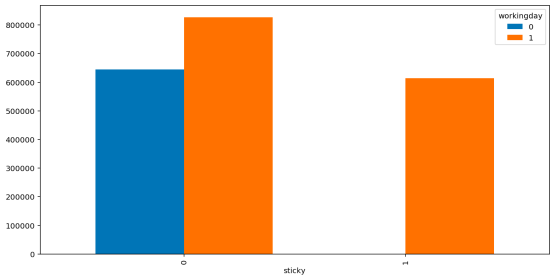

fig 13. relationship between sticky and count

This figure shows that the new feature has an important effect on count. It means the feature could be used in input values.

3 Model Selection and Algorithm

As mentioned above, this project used two machine learning model - Random Forest, Gradient Boosting, and the combination of the two models, one deep learning model - Long short-term Memory. This part will talk about their algorithms and realization one by one.

Before building predictive models, it is important to find out the relationship of all features with target variable (count). The correlation plot below shows that count has a strong relationship with temperature, and relative strong relationship with hour, month, humidity, weather type. When building models, these features will get more attention.

3.1 Data Preparation for Model

3.1.1 Data Split for model

In data mining, source dataset is split to two sub-set for different use. The training set is used to build the predictive models, the test set is unseen data when building models, which is used to evaluate the models performance.

This project split train set and test set with day. Normally, the dataset is from the first day to 19th day. Set the dataset when before 15th day, and the test set between 15th day and 19th day.

3.1.2 Target features for model

As mentioned above, when predict count, there are two way to be used. First, set count as target feature directly. Second, set casual and registered as target feature, and then add them together, it will be equal to the value of count per row.

casual	registered	count
3	13	16
8	32	40
5	27	32
3	10	13
0	1	1

fig 14. count equals to add registered and casual

3.2 Machine Learning

3.2.1 Random Forest

Random forests or random decision forests 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. [4]

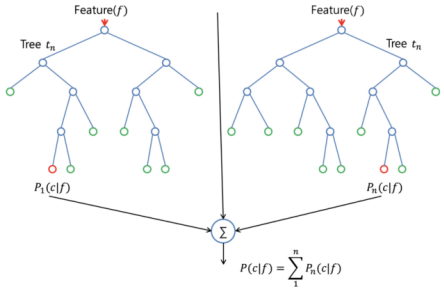

fig 15. work flow of random forest

Advantages of using Random Forest technique:

- Handles higher dimensionality data very well.
- Handles missing values and maintains accuracy for missing data.

Disadvantages of using Random Forest technique:

- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the regression model.

This project used RandomForestRegressor, the meaning of parameters is as following:
n_estimators :
The number of trees in the forest.

max_depth :
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split :
The minimum number of samples required to split an internal node.

min_samples_leaf :
The minimum number of samples required to be at a leaf node.

n_jobs :
The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.

3.2.2 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. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. [8]

The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean. The latter two papers introduced the view of boosting algorithms as iterative functional gradient descent algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification. [5]

Algorithm 1: Gradient_Boost

1 $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2 For $m = 1$ to M do:
3 $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4 $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5 $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6 $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7 endFor
end Algorithm

fig 16. algorithm of Gradient Boost

Advantages of using Gradient Boosting technique:

- Supports different loss function.
- Works well with interactions.

Disadvantages of using Gradient Boosting technique:

- Prone to over-fitting.
- Requires careful tuning of different hyper-parameters

Here used GradientBoostingRegressor, the meaning of parameters is as following:
loss :
loss function to be optimized. ‘ls’ refers to least squares regression. ‘lad’ (least absolute deviation) is a highly robust loss function solely based on order information of the input variables. ‘huber’ is a combination of the two. ‘quantile’ allows quantile regression (use alpha to specify the quantile).

n_estimators :
The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

learning_rate :
learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.

max_depth :
maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

min_samples_leaf :
The minimum number of samples required to be at a leaf node

3.2.3 Realization of two models

Random Forest and Gradient Boosting all belong to machine learning. Here using Scikit-learn to build machine learning models. Scikit-learn is a library that provides a variety of both supervised and unsupervised machine learning techniques as well as utilities for common tasks such as model selection, feature extraction, and feature selection.

Scikit-learn provides an object-oriented interface centered around the concept of an Estimator. According to the scikit-learn tutorial "An estimator is any object that learns from data; it may be a classification, regression or clustering algorithm or a transformer that extracts/filters useful features from raw data." The API of an estimator looks roughly as follows:

```
class Estimator(object):

def fit(self, X, y=None):
"""Fits estimator to data. """
# set state of ``self``
return self

def predict(self, X):
"""Predict response of ``X``. """
# compute predictions ``pred``
return pred
```

fig 17. the API of an estimator

The Estimator.fit method sets the state of the estimator based on the training data. Usually, the data is comprised of a two-dimensional numpy array X of shape (n_samples, n_predictors) that holds the so-called feature matrix and a one-dimensional numpy array y that holds the responses (either class labels or regression values).

Estimators that can generate predictions provide an Estimator.predict method. In

the case of regression, Estimator.predict will return the predicted regression values; it will return the corresponding class labels in the case of classification. Classifiers that can predict the probability of class membership have a method Estimator.predict_proba that returns a two-dimensional numpy array of shape (n_samples, n_classes) where the classes are lexicographically ordered. [7]

In this project, the set of parameters for the two models as follows:

1. RandomForestRegressor:
    a. 'n_estimators': 1000
    b. 'max_depth': 15
    c. 'random_state': 0
    d. 'min_samples_split' : 5
    e. 'n_jobs': -1
2. GradientBoostingRegressor:
    a. 'n_estimators': 150
    b. 'max_depth': 5
    c. 'random_state': 0
    d. 'min_samples_leaf' : 10
    e. 'learning_rate': 0.1
    f. 'subsample': 0.7
    g. 'loss': 'ls'

The steps for the two model as follows:

1. import packages from library
2. set the parameter and input feature for this model
3. using model.fit() to get the model_registered, model_casual, model_count
4. predict the values: y_pred_count, y_pred_total using model.predict().
5. set the values smaller than 0 to 0. Then, using get_rmsle(predict, actual) method to get the RMSLE result. The smaller, the better.
6. change parameter and input features to improve model
7. predict test data with model
8. submit the result to Kaggle to get the RMSLE result.

3.2.4 Combine the two models

```
y_p = np.round(.2*rf_p_total + .8*gbm_p_total)
print get_rmsle(y_p, rf_t_total)
```

```
0.3184189300346792
```

```
rf_pred = predict_on_test_set(rf_model, rf_cols)
gbm_pred = predict_on_test_set(gbm_model, gbm_cols)
y_pred5 = np.round(.2*rf_pred[0] + .8*gbm_pred[0])
# output predictions for submission
pred = pd.DataFrame({'datetime': df_test['datetime'],'count': y_pred5})
pred = pred[['datetime','count']]
pred.to_csv('submission3-1.csv', index=False)
```

```
plt.plot(y_pred5)
plt.xlabel('data row',fontsize = 16)
plt.ylabel('count',fontsize = 16)
```

fig 18. the code of combinative models

Even though there is no validation dataset for combinative models. I also create it two check whether it will get a better result or not for test dataset.

3.3 Deep Learning

3.3.1 Long short-term Memory

Long Short Term Memory networks are a type of Recurrent Neural Networks (RNN) capable of learning long-term dependencies in the data. They are explicitly designed to avoid the problem of vanishing gradient. All recurrent neural networks have the form of a chain of repeating modules of neural network. LSTMs also have this chain like structure, but the repeating module has a different structure. LSTM cells have three gates, each gate has an activation function which allows the cell to read, write and forget information. [6]

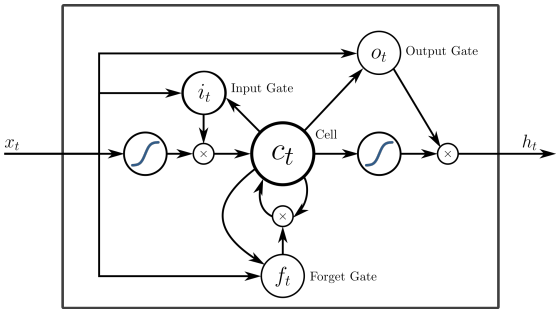

fig 19. work flow of LSTM

The meaning of parameters used in this project are as following:

Activations:
Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers.

Available activations:
softmax, elu, selu, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, linear

Optimizer:
This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the Optimizer class. See: optimizers.

Available optimizer:
SGD, RMSprop(This optimizer is usually a good choice for recurrent neural networks), Adagrad, Adadelta, Adam, Adamax, Nadam, TFOptimizer

Loss Function:
This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse), or it can be an objective function.

Available loss functions:
mean_squared_error,
mean_absolute_error,
mean_absolute_percentage_error,
mean_squared_logarithmic_error,
squared_hinge, hinge, categorical_hinge, logcosh

Callbacks:
A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes. The relevant methods of the callbacks will then be called at each stage of the training.

Available callbacks:
BaseLogger, TerminateOnNaN,
ProgbarLogger, History,
ModelCheckpoint, EarlyStopping,
RemoteMonitor, LearningRateScheduler,
TensorBoard (TensorBoard is a visualization tool provided with TensorFlow), ReduceLROnPlateau,
CSVLogger, LambdaCallback

3.3.2 Realization of LSTM

Here using Keras to build machine learning models. Keras is a model-level library, providing high-level building blocks for developing deep learning models. It does not handle itself low-level operations such as tensor products, convolutions and so on. Instead, it relies on a specialized, well-optimized tensor manipulation library to do so, serving as the "backend engine" of Keras. Rather than picking one single tensor library and making the implementation of Keras tied to that library, Keras handles the problem in a modular way, and several different backend engines can be plugged seamlessly into Keras.

Here using the TensorFlow backend. (TensorFlow is an open-source symbolic tensor manipulation framework developed by Google. By default, Keras will use TensorFlow as its tensor manipulation library.)

The type of this model is the Sequential model, a linear stack of layers. The first layer in a Sequential model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape.

Build steps as following:

1. scaling data
2. split train dataset to train and validation data.
3. import packages from library
4. set the parameter and input feature for this model
5. stack Layers Using .add()
6. configure learning Process using .compile()
7. training data using .fit()
8. evaluate performance
9. get the result of RMSLE and improve model
10. predict test data with model
11. submit the result to Kaggle to get the RMSLE result.

In my model, I stack 9 LSTM layers on top of each other, making the model capable of learning higher-level temporal representations.

The first 8 LSTMs return the full output sequences, but the last one only returns the last step in its output sequence, thus dropping the temporal dimension (i.e. converting the input sequence into a single vector). The model shows in the table 2.

Table 2: LSTM Model Summary

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 1, 512)	1083392
lstm_2 (LSTM)	(None, 1, 256)	787456
lstm_3 (LSTM)	(None, 1, 128)	197120
lstm_4 (LSTM)	(None, 1, 64)	49408
lstm_5 (LSTM)	(None, 1, 32)	12416
lstm_6 (LSTM)	(None, 1, 16)	3136
lstm_7 (LSTM)	(None, 1, 8)	800
lstm_8 (LSTM)	(None, 1, 4)	208
lstm_9 (LSTM)	(None,, 2)	56
dense_1 (Dense)	(None, 1)	3

4 Conclusion

4.1 Random Forest:

the result of validation set is as following:

- count = 0.44353268319380657
- total = 0.44125110350369806

where, count means predict count directly, total means predict count by adding registered and casual. This project will analysis the result one by one.

4.1.1 For predict count directly:

the result of validation set is as following:

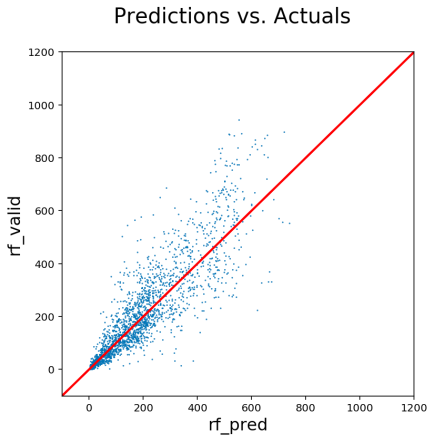

fig 20. predictions vs. actuals for predict count directly

the more point close to the red line, the better the result is.

the result of predict the test dataset:

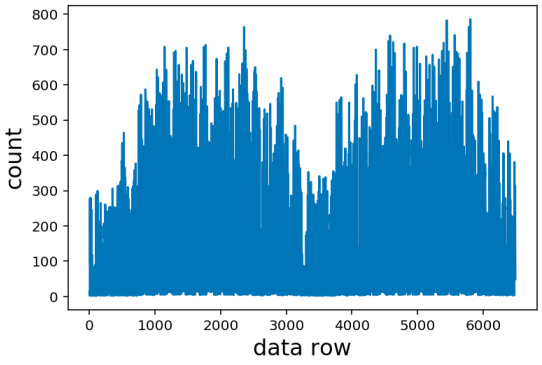

fig 21. the result of test dataset (predict count directly)

It is the result of test dataset. It means the count value per row.

4.1.2 For predict count by registered and casual:

the result of validation set is as following:

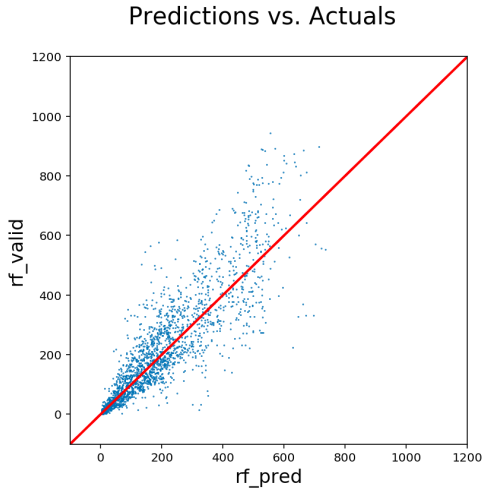

fig 22. predictions vs. actuals for predict count = registered + casual

the more point close to the red line, the better the result is.

the result of predict the test dataset:

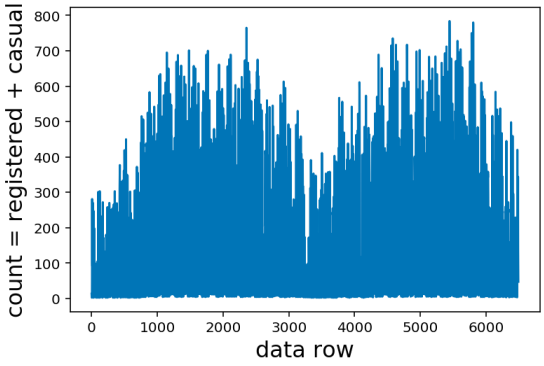

fig 23. the result of test dataset (predict count = registered + casual)

It is the result of test dataset. It means the count value per row.

4.1.3 Compare the two predict ways:

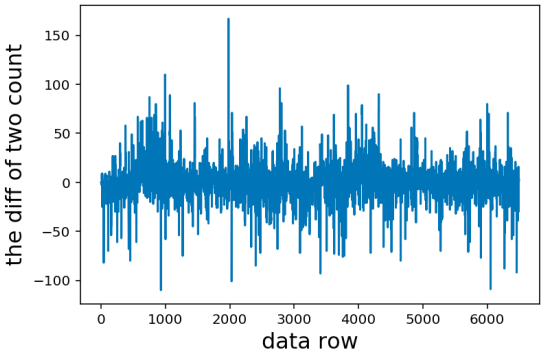

fig 24.the difference between two count

If the point close to 0, it means that at the one hour, the prediction results for the two ways are same. Otherwise, they are different. The graph shows that most point are in -50 and 50.

4.1.4 Get RMSLE of test dataset

After predict the counts of test dataset, I submit the two results to Kaggle, and get RMSLE of the test dataset, which are shown as follows:

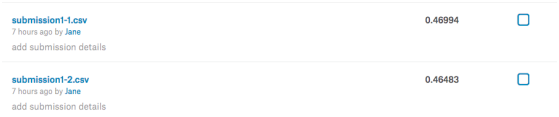

fig 25. the RMSLE between two count

Here can see that the result is above 0.46. The predict model Random Forest is not a good one for my project. Moreover, comparing the two score, we can see that predict count by adding registered and casual has a better score than predict directly as the lower, the better.

4.2 Gradient Boosting:

the result of validation set is as following:

- count = 0.315007370141069
- total = 0.3147256174226919

where, count means predict count directly, total means predict count by adding registered and casual. This project will analysis the result one by one.

4.2.1 For predict count directly:

the result of validation set is as following:

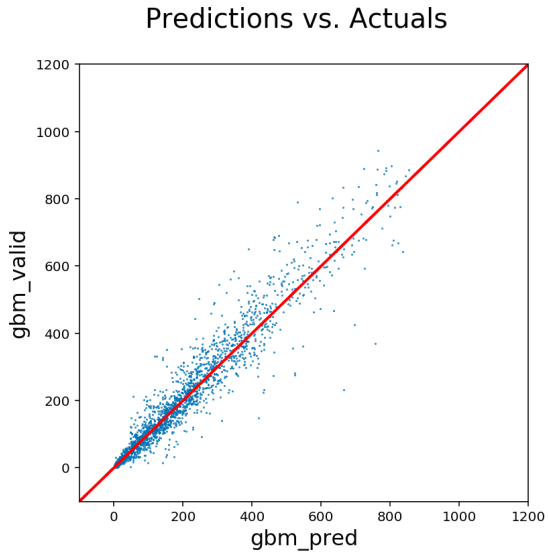

fig 26. predictions vs. actuals for predict count directly

the more point close to the red line, the better the result is.

the result of predict the test dataset:

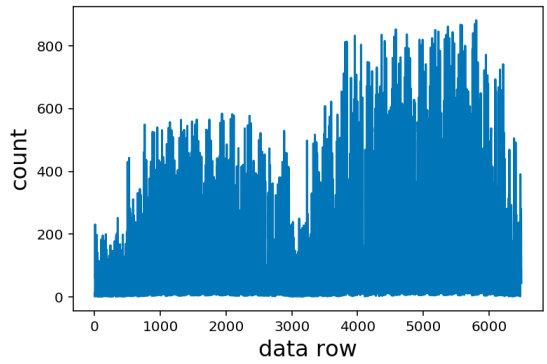

fig 27. the result of test dataset (predict count directly)

It is the result of test dataset. It means the count value per row.

4.2.2 For predict count by registered and casual:

the result of validation set is as following:

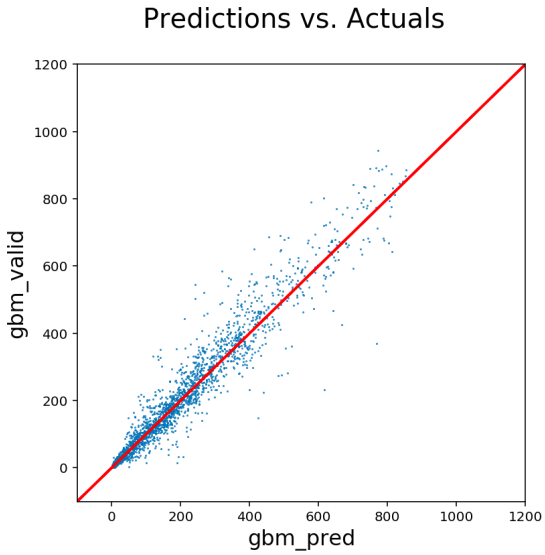

fig 28. predictions vs. actuals for predict count = registered + casual

the more point close to the red line, the better the result is.

the result of predict the test dataset:

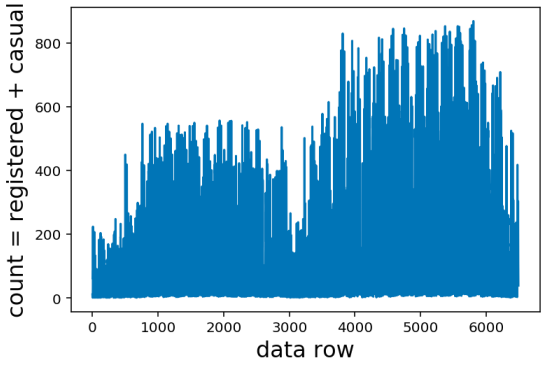

fig 29. the result of test dataset (predict count = registered + casual)

It is the result of test dataset. It means the count value per row.

4.2.3 Compare the two predict ways:

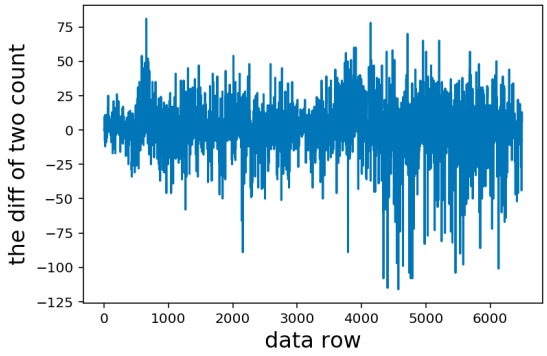

fig 30.the difference between two count

If the point close to 0, it means that at the one hour, the prediction results for the two ways are same. Otherwise, they are different. The graph shows that most point are in -25 and 25.

4.2.4 Get RMSLE of test dataset

After predict the counts of test set, I submit the two results to Kaggle, and get RMSLE of the test dataset, which are shown as follows:

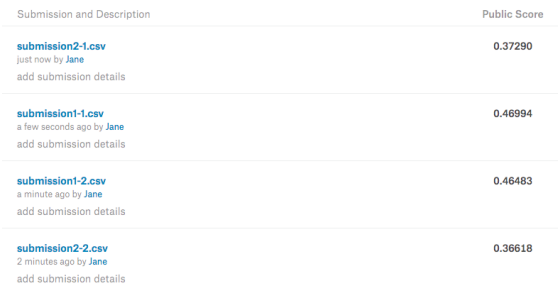

fig 31. the RMSLE between two count

Here can see that the result is above 0.366 and 0.37. The predict model Gradient Boosting is better than Random Forest for my project. Moreover, comparing the two score, we can see that predict count by adding registered and casual has a better score than predict directly as the lower, the better.

4.3 Combinative Models

For this part, this project used random forest and gradient boosting together. Set the model 0.2*rf_p_total and 0.8*gbm_p_total; then use the combinative model to predict count of test dataset. The results are as following:

4.3.1 For predict count directly:

the result of predict the test dataset:

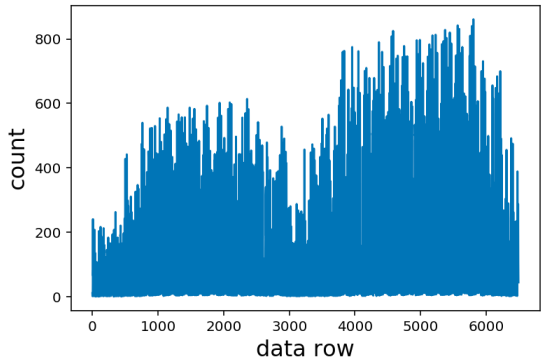

fig 32. the result of test dataset (predict count directly)

It is the result of test dataset. It means the count value per row.

4.3.2 For predict count by registered and casual:

the result of predict the test dataset:

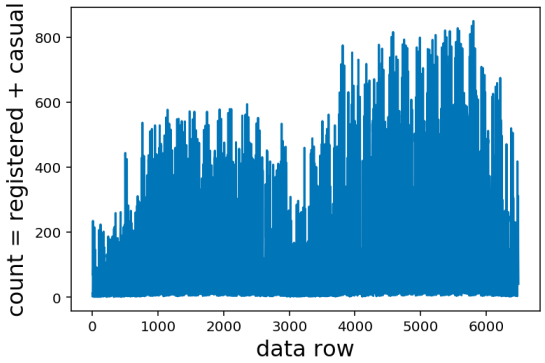

fig 33. the result of test dataset (predict count = registered + casual)

It is the result of test dataset. It means the count value per row.

4.3.3 Get RMSLE of test dataset

After predict the counts of test set, I submit the two results to Kaggle, and get RMSLE of the test dataset, which are shown as follows:

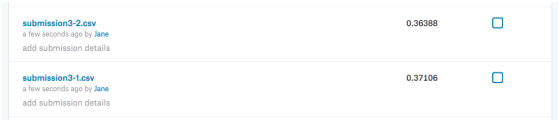

fig 34. the RMSLE between two count

Here can see that the result is 0.363 and 0.37. The predict combinative model is better than Gradient Boosting and Random Forest for my project. Moreover, comparing the two score, we can see that predict count by adding registered and casual has a better score than predict directly as the lower, the better.

4.4 Model LSTM:

the result of validation set is as following:

- count = 0.43323475
- causal = 0.67320853
- registered = 0.4203577

where, count means predict count directly, registered means the riders who has registered and casual means the riders who is non-registered. This project will analysis the result one by one.

4.4.1 For predict count directly:

the result of predict the test dataset:

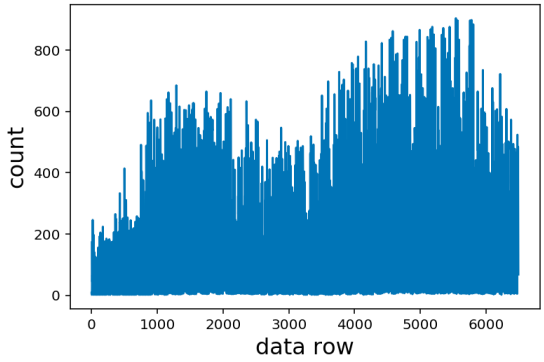

fig 35. the result of test dataset (predict count directly)

It is the result of test dataset. It means the count value per row.

4.4.2 For predict count by registered and casual:

the result of predict the test dataset:

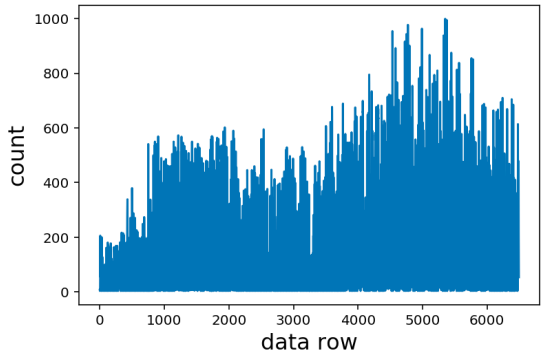

fig 36. the result of test dataset (predict count = registered + casual)

It is the result of test dataset. It means the count value per row.

4.4.3 Compare the two predict ways:

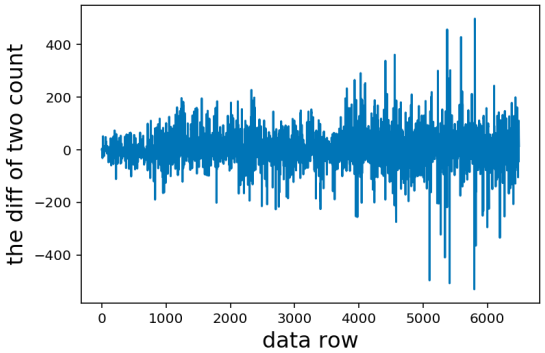

fig 37. the difference between two count

If the point close to 0, it means that at the one hour, the prediction results for the two ways are same. Otherwise, they are different. The graph shows that most point are in -200 and 200.

4.4.4 Get RMSLE of test dataset

After predict the counts of test set, I submit the two results to Kaggle, and get RMSLE of the test dataset, which are shown as follows:

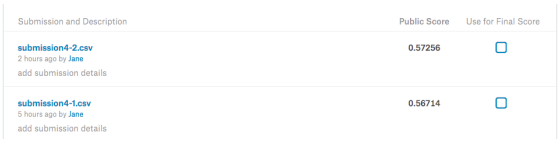

fig 38. the RMSLE between two count

Here can see that the result is above 0.56. The predict model LSTM is not a good one for my project. The result is same to the prediction before starting project. Moreover, comparing the two score, we can see that predict count by adding registered and casual has a little worse score than predict directly as the lower, the better. The result is different to the machine learning models I gave in this project.

5 Discussion

5.1 Conclusion

The project used three models to predict count of bike rental. From the result, there are some conclusion can be get:

1. It is clear that the features provide more information for the regression models, allowing them to make better predictions, as is the case of the features rush hours, ideal temperature and humidity, along with the indicators for sticky weather. It is same to the prediction before the project.

2. The result of using registered and causal to predict count is different with using count directly. The most same count of the two ways is using the combinative mode with Random Forest and Gradient Boosting. The best RMSLE result is 0.63,

which in Top 15. The most difference is using LSTM. It is clear that predict count using registered and causal separately is better than predict count directly for two machine learning model, and have the opposite result for the one deep learning model.

3. With the implemented solution it was clear that the best model is Gradient Boosting, the worst model is LSTM. It is same to the prediction before the project.

4. No predictive model is perfect, there will be always have better one.

5.2 Evaluation & Limitations

The aim of this project is to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C. However, it is impossible to consider all the factors in one research. There are too many factors will affect bike users' behavior, for example, an event of bicycle racing or marathon, a short term government policy related to traffic, a tornado or typhoon. Then, as one user who joined this competition on Kaggle said, there are some error in temp feature collection. Moreover, the train dataset the competition gave is not enough to get a good result as there are only about 1w data.

5.3 Future Work

Without think about the factor of this competition, there are many additional area of research worthy to do in the future. For example, add more independence variables by other factors which will affect users' rental behavior, likes pollution, government policy, and geographic location, etc. Moreover, getting more data from other cities or more years is a good way to improve the predict result. Trying other enhancing techniques to improve the model's performance is another good choice as no one model is perfect.

References:

[1] Kaggle Inc. Bike sharing demand. https://www.kaggle.com/c/bike-sharing-demand/, 2018.
[2] Wang, W. *Forecasting Bike Rental Demand Using New York Citi Bike Data*, Dublin Institute of Technology, 2016.
[3] Breiman, L. (2001). *Random forests*. Machine Learning, 45(1), 5–32.
[4] Scikit-learn Documentation: sklearn.ensemble.RandomForestClassifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, v0.19.1.
[5] Scikit-learn Documentation: sklearn.ensemble.GradientBoostingClassifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html, v0.19.1.
[6] Fishman, E. (2015). Bikeshare: A Review of Recent Literature. Transport Reviews, (ahead-of-print), 1–22.
[7] Keras Inc. Keras Documentation. https://keras.io/, 2018.
[8] Lipton, C. Z., Berkowitz J., & Elkan C.(2015) *A Critical Review of Recurrent Neural Networks for Sequence Learning*, arXiv:1506.00019v4 [cs.LG] 17 Oct 2015.
[9] Ordóñez J. F., & Roggen D.(2015) *Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition*, Wearable Technologies, Sensor Technology Research Centre, University of Sussex, Brighton BN1 9RH, UK.
[10] Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., & Feuston, B. P. (2003). Random forest: a classification and regression tool for compound classification and QSAR modeling. Journal of Chemical Information and Computer Sciences, 43(6), 1947–1958
[11] Lin M., *Application of Machine Learning Techniques to Forecast Bike Rental Demand in the Capital Bikeshare Program in Washington, D.C*. DOI: 10.13140/RG.2.1.1433.7766, 2015-08-26 T 17:24:18 UTC.

Appendix:

Github Link:
https://github.com/huizhewang/INFO7390
Kaggle Notebook:
https://github.com/huizhewang/INFO7390/blob/master/Kaggle%20Notebook.ipynb
Code with Document:
https://github.com/huizhewang/INFO7390/blob/master/bike%20sharing%20demand.ipynb