Bike Renting - Report

Kartik Mehboobani

13 September 2018

Contents

1 Introduction

1.1 Problem Statement

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings using machine learning models.

1.2 Data

Our task is to build machine learning models to predict the count of the rental bikes on a particular day based on the enviromental and seasonal parameters. Given below is the sample of the data which we will use to build our models.

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	01-01-2011	1	0	1	0	6	0	2
2	02-01-2011	1	0	1	0	0	0	2
3	03-01-2011	1	0	1	0	1	1	1
4	04-01-2011	1	0	1	0	2	1	1
5	05-01-2011	1	0	1	0	3	1	1

Table 1: Training Data (Columns 1-9)

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600

Table 2: Training Data (Columns 10-16)

The features available to predict the count of rental bikes on a particular day are:

S.No.	Features
0	instant
1	dteday
2	season
3	yr
4	mnth
5	holiday
6	weekday
7	workingday
8	weathersit
9	temp
10	atemp
11	hum
12	windspeed
13	casual
14	registered
15	cnt

Table 3: Available Features

2 Methodology

2.1 Pre Processing

Any predictive modeling requires that we look at the data before we start modeling. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this we will first separate the variables in continous and categorical types. Then we will look at the distribution of the continous variables. Most analysis require the data to be normally distributed. We can visualise this by glancing at the distribution plots of the variables.

2.1.1 Exploratory Data Analysis

In Fig. 1, 2, 3, 4 we have plotted the distribution plots of the continous variables we have in the data and we can clearly see that all of them are pretty much uniformly distributed.

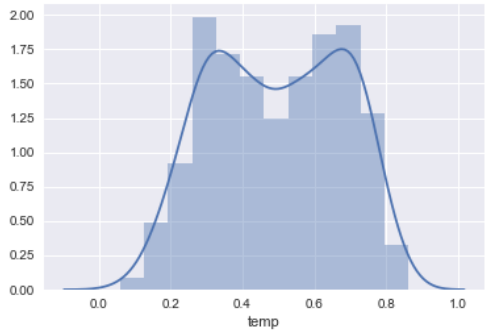

Figure 1: Temperature Distribution Plot

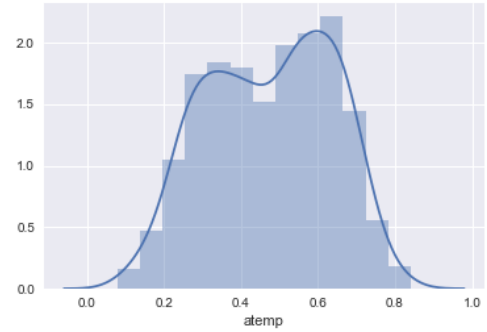

Figure 2: Feels like Temperature Distribution Plot

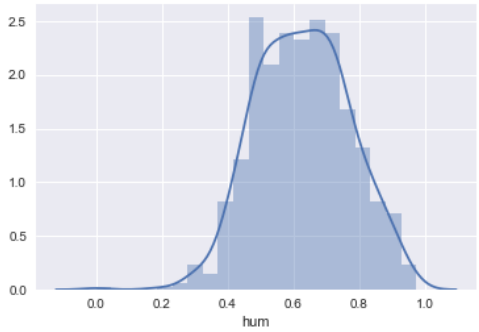

Figure 3: Humidity Distribution Plot

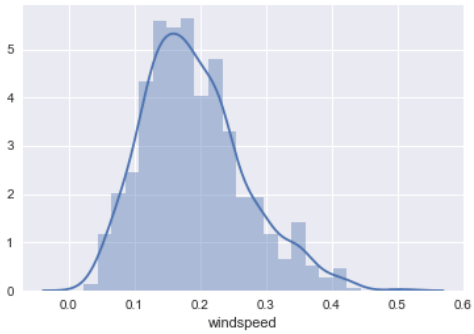

Figure 4: Windspeed Distribution Plot

We also look at the target variable i.e. **cnt** in Fig. 5

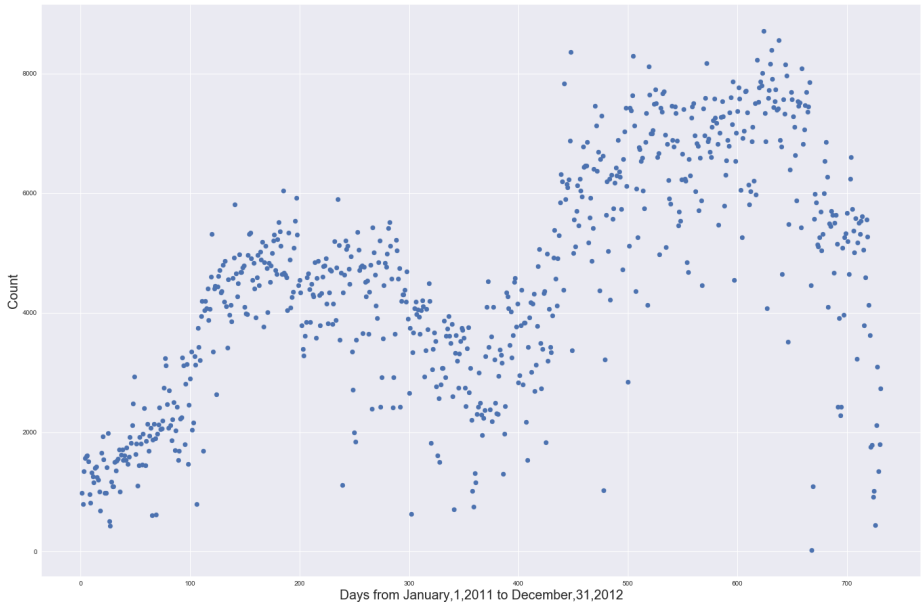

Figure 5: Count scatter Plot

2.1.2 Feature Selection

Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. There are several methods of doing that but here categorical variables are about completely different property of the day which can't be dependent on each other. And we should also create dummy variables for non-binary categorical variables.

We also have to check whether if any of the variables are corelated as to avoid multicollinearity. We can do so by making a corelation matrix of the continous variables as shown in Table 2.1.2. We can clearly see that the variable **temp** and textbfatemp, so we should drop one of these from the training data.

	temp	atemp	hum	windspeed
temp	1	0.991702	0.126963	-0.15794
atemp	0.991702	1	0.139988	-0.18364
hum	0.126963	0.139988	1	-0.24849
windspeed	-0.15794	-0.18364	-0.24849	1

Table 4: Corelation Matrix

All the other variables have different significance as we can say from practical knowledge but the variables **instant** and **dteday** are just ID variables so we should drop them. Also the variables **casual** and **registered** are also not required for model training as the target variable is just the sum of these two, so we should drop them too.

2.2 Modelling

2.2.1 Model Selection

The dependent variable, in our case ***cnt*** is continous. So the only predictive analytics we can use is **Regression**. Now we will try building various regressors to predict our target variable and then select whichever will work best.

2.2.2 Multiple Linear regression

We can tune the hyperparameters of Linear Regressiont like *copy-X, normalize, fit-intercept*. For finding the best parameters for our data we run a grid search over various values of these paramaters based on which we found that the *copy-X* of **True**, *normalize* of **True** and *fit-intercept* of **True** works best.

2.2.3 Support Vector Regressor

For support vector machines to work good we would want our data to be scaled but here our data is already scaled. Now while building the support vector classifier we can tune some hyperparameters like *C, gamma, kernel*. So for finding the parameters which will work best for our data we have done a grid search over various values of these parameters based on which we found that the C of **1000**, *gamma* of **0.0001** and for *kernel* of **linear** works best.

2.2.4 Decision Tree Regressor

We can also tune the hyperparameters of Decision Tree like *max-depth, min-samples-leaf, max-features*. For finding the best parameters for our data we run a grid search over various values of these paramaters based on which we found that the *max-depth* of **12**, *min-samples-leaf* of **10** and *max-features* of **auto** works best.

2.2.5 Random Forest Regressor

We can also tune the hyperparameters of Random Forest like *max-depth, min-samples-leaf, max-features, n-estimators, oob-score*. For finding the best parameters for our data we run a grid search over various values of these paramaters based on which we found that the *max-depth* of **15**, *min-samples-leaf* of **2**, *max-features* of **auto**, *n-estimators* of **600** and *oob-score* of **True** works best.

3 Conclusion

3.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive performance
2. Interpretability
3. Computational Efficiency

In our case, the latter two, Interpretability and Computation Efficiency, do not hold much significance. Therefore we will use Predictive performance as the criteria to compare and evaluate models.

Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

3.1.1 Mean Absolute percentage Error (MAPE)

Mean Absolute percentage Error can be calculated by calculating the percentage error term for ecah point and taking the mean of all those errors.

3.1.2 Root Mean Square Error (RMSE)

Root Mean Square Error can be calculated by taking the square root of the sum of the squares of the residuals for each point.

3.2 Model Selection

Now we have created various models for our data, we will select which will have the least MAPE and RMSE.

Comparing the MAPE and RMSE of all models for both training as well as test set we can see that on test set the Suppport vector Regressor works best but for training set the MAPE and RMSE value of the Support vector Regressor is pretty high while the Random Forest Regressor have MAPE and RMSE low value for both training as well as test set. So we will choose Random Forest Regressor for our predictions.

Model	Training MAPE	Training RMSE	Test MAPE	Test RMSE
Multiple Linear Regressor	44.10548	756.0834	18.26439	780.3431
Support Vector Regressor	46.51463	774.8872	18.16402	762.1265
Decision Tree Regressor	51.58942	683.3185	23.27345	829.3728
Random Forest Regressor	26.71262	353.5704	19.44906	684.6186

Table 5: results

A Python Code

```
# coding: utf-8

# In[ ]:


# Importing Libraries

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cross_validation import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from math import sqrt
from scipy.stats import chi2_contingency


from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

get_ipython().run_line_magic('matplotlib', 'inline')


# In[ ]:


# Setting Working Directory

os.chdir("D:\Data_Science\edWisor\Project_2\data")


# In[ ]:


# Loading Data

data = pd.read_csv('day.csv')
```

```
# In[ ]:


############ Exploratory data Analysis ##########


# In[ ]:


data.head()


# In[ ]:


features = pd.DataFrame(data.columns)
#features.to_csv('features.csv')


# In[ ]:


# Continous variables

cnames = ['temp','atemp','hum','windspeed']
cat_names = ['season', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']


# In[ ]:


# Corelation between continous variables

corr = data[cnames].corr()
corr
#corr.to_csv('Correlations.csv')


# In[ ]:


sns.distplot(data['temp'])
#plt.savefig('temp.png')


# In[ ]:
```

```
sns.distplot(data['atemp'])
#plt.savefig('atemp.png')


# In[ ]:


sns.distplot(data['hum'])
#plt.savefig('hum.png')


# In[ ]:


sns.distplot(data['windspeed'])
#plt.savefig('windspeed.png')


# In[ ]:


plt.figure(figsize=(24,16))
plt.scatter(data['instant'], data['cnt'])
plt.xlabel('Days from January,1,2011 to December,31,2012', fontsize = 20)
plt.ylabel('Count', fontsize =20)
#plt.savefig('RentCount.png')


# In[ ]:


# Creating Dummy Variables for non-binary categorical variables

for i in ['season','mnth','weekday','weathersit']:
    temp = pd.get_dummies(data[i], prefix = i)
    data = data.join(temp)
    data.drop(i, axis =1,inplace = True)


# In[ ]:


data.drop(['instant','dteday'],axis=1,inplace = True)
```

```
# In[ ]:


data.head()


# In[ ]:


# Splitting the data into train and test sets

train,test = train_test_split(data,test_size =0.2, random_state =0)


# In[ ]:


# Preparing Data for modelling

X_train = train.drop(['casual','registered','cnt','temp'],axis=1)
X_test = test.drop(['casual','registered','cnt','temp'],axis=1)
y_casual = train['casual']
y_registered = train['registered']
y_cnt = train['cnt']


# In[ ]:


# Evaluation Functions

def MAPE(y_true, y_pred):
    mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
    return mape
#Calculate MAPE

def RMSE(y_true, y_pred):
    rms = sqrt(mean_squared_error(y_true, y_pred))
    return rms
#Calculate RMSE


# In[ ]:
```

```
############ Regression Models ##########

# In[ ]:

############## Multiple linear Regression ###############

# In[ ]:

from sklearn.linear_model import LinearRegression

# In[ ]:

# Grid Search for best Parameters

reg_lm = LinearRegression()
params_lm = [{'copy_X':[True, False],
              'fit_intercept':[True,False],
              'normalize':[True, False]}]
grid_search_lm = GridSearchCV(reg_lm, param_grid = params_lm, cv =10, n_jobs =-1)
grid_search_lm = grid_search_lm.fit(X_train,y_cnt)

# In[ ]:

grid_search_lm.best_score_

# In[ ]:

grid_search_lm.best_params_

# In[ ]:

# Training with best paramaeters

reg_lm_best = LinearRegression(copy_X=True, fit_intercept=False, normalize=True)
```

```
reg_lm_best.fit(X_train,y_cnt)


# In[ ]:


# Evaluating on training set

y_pred_lm = reg_lm_best.predict(X_train)
mape1_lm = MAPE(y_cnt, y_pred_lm)
rmse1_lm = RMSE(y_cnt, y_pred_lm)
print('MAPE : {:.2f}'.format(mape1_lm))
print('RMSE : {:.2f}'.format(rmse1_lm))


# In[ ]:


# Evaluating on Test Set

y_pred_lm = reg_lm_best.predict(X_test)
mape2_lm = MAPE(test['cnt'], y_pred_lm)
rmse2_lm = RMSE(test['cnt'], y_pred_lm)
print('MAPE : {:.2f}'.format(mape2_lm))
print('RMSE : {:.2f}'.format(rmse2_lm))


# In[ ]:


############## Supoort Vector Regressor ###############


# In[ ]:


from sklearn.svm import SVR


# In[ ]:


# Grid Search for best Parameters

reg_svr = SVR()
param= [{'C':[1,10,100,1000],
```

```
        'kernel': ['rbf','linear'],
        'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10]}]
grid_search_svr = GridSearchCV(reg_svr, param_grid = param, cv =10, n_jobs =-1)
grid_search_svr = grid_search_svr.fit(X_train,y_cnt)


# In[ ]:


grid_search_svr.best_params_


# In[ ]:


grid_search_svr.best_score_


# In[ ]:


# Training with best paramaeters

reg_svr_best = SVR(C = 1000, kernel = 'linear', gamma = 0.0001)
reg_svr_best.fit(X_train,y_cnt)


# In[ ]:


# Evaluating on training set

a = reg_svr_best.predict(X_train)
mape1_svr = MAPE(y_cnt,a)
rmse1_svr = RMSE(y_cnt,a)
print('MAPE : {:.2f}'.format(mape1_svr))
print('RMSE : {:.2f}'.format(rmse1_svr))


# In[ ]:


# Evaluating on Test Set

y_pred = reg_svr_best.predict(X_test)
mape2_svr = MAPE(test['cnt'],y_pred)
```

```
rmse2_svr = RMSE(test['cnt'],y_pred)
print('MAPE : {:.2f}'.format(mape2_svr))
print('RMSE : {:.2f}'.format(rmse2_svr))


# In[ ]:


########## Decision Tree Regressor ##########


# In[ ]:


from sklearn.tree import DecisionTreeRegressor


# In[ ]:


# Grid Search for best Parameters

reg_dt = DecisionTreeRegressor(random_state = 0)
params = [{'max_depth':[2,4,6,8,10,12,15],
           'max_features':['auto','sqrt'],
           'min_samples_leaf':[2,4,6,8,10]}]
grid_search_dt = GridSearchCV(reg_dt, param_grid = params, cv =10, n_jobs =-1)
grid_search_dt = grid_search_dt.fit(X_train,y_cnt)


# In[ ]:


grid_search_dt.best_score_


# In[ ]:


grid_search_dt.best_params_


# In[ ]:


# Training with best parameters
```

```
reg_dt_best = DecisionTreeRegressor(random_state = 0, max_depth = 12,
                                    min_samples_leaf = 10, max_features = 'auto')
reg_dt_best.fit(X_train,y_cnt)


# In[ ]:


# Evaluating on training set

b = reg_dt_best.predict(X_train)
mape1_dt = MAPE(y_cnt,b)
rmse1_dt = RMSE(y_cnt,b)
print('MAPE : {:.2f}'.format(mape1_dt))
print('RMSE : {:.2f}'.format(rmse1_dt))


# In[ ]:


# Evaluating on test set

y_pred_dt = reg_dt_best.predict(X_test)
mape2_dt = MAPE(test['cnt'],y_pred_dt)
rmse2_dt = RMSE(test['cnt'],y_pred_dt)
print('MAPE : {:.2f}'.format(mape2_dt))
print('RMSE : {:.2f}'.format(rmse2_dt))


# In[ ]:


########### Random Forest Regressor ############


# In[ ]:


from sklearn.ensemble import RandomForestRegressor


# In[ ]:


# Grid Search for best Parameters
```

```
reg_rf = RandomForestRegressor(random_state = 0)
params_rf = [{'max_depth':[8,10,12,15],
              'max_features':['auto','sqrt'],
              'min_samples_leaf':[2,4,6],
              'n_estimators': [200, 500, 600],
              'oob_score':[True, False]}]
grid_search_rf = GridSearchCV(reg_rf, param_grid = params_rf, cv =10, n_jobs =-1)
grid_search_rf = grid_search_rf.fit(X_train,y_cnt)


# In[ ]:


grid_search_rf.best_score_


# In[ ]:


grid_search_rf.best_params_


# In[ ]:


reg_rf_best = RandomForestRegressor(random_state = 0, max_depth = 15,
                                    max_features = 'auto', min_samples_leaf = 2,
                                    n_estimators = 600, oob_score = True)
reg_rf_best.fit(X_train,y_cnt)


# In[ ]:


# Evaluating on training set

c = reg_rf_best.predict(X_train)
mape1_rf = MAPE(y_cnt,c)
rmse1_rf = RMSE(y_cnt,c)
print('MAPE : {:.2f}'.format(mape1_rf))
print('RMSE : {:.2f}'.format(rmse1_rf))


# In[ ]:
```

```
# Evaluating on test set

y_pred_rf = reg_rf_best.predict(X_test)
mape2_rf = MAPE(test['cnt'],y_pred_rf)
rmse2_rf = RMSE(test['cnt'],y_pred_rf)
print('MAPE : {:.2f}'.format(mape2_rf))
print('RMSE : {:.2f}'.format(rmse2_rf))


# In[ ]:


############ Result ##########


# In[ ]:


result = pd.DataFrame()
result['Model'] = ['Multiple Linear Regressor','Support Vector Regressor',
                   'Decision Tree Regressor', 'Random Forest Regressor']
result['Training MAPE'] = [mape1_lm, mape1_svr, mape1_dt, mape1_rf]
result['Training RMSE'] = [rmse1_lm, rmse1_svr, rmse1_dt, rmse1_rf]
result['Test MAPE'] = [mape2_lm, mape2_svr, mape2_dt, mape2_rf]
result['Test RMSE'] = [rmse2_lm, rmse2_svr, rmse2_dt, rmse2_rf]
result.to_csv('result.csv')


# In[ ]:


result
```