BIKE DEMAND PREDICTION

1) Introduction

I am given a dataset of hourly bike rental volume and some associated variables. My task is to develop a prediction model of the hourly bike demand. In this note, I first discuss the features I use for the prediction. Then I explore the performances of several models and the results of the best performing one. Last, I mention about some considerations on scalability and then conclude. The Python code that I use in the analysis can be found in the Appendix.

2) Feature Analysis

The dataset includes the following variables:

- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month (1 to 12)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit :
    - 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 : Normalized temperature in Celsius. The values are divided to 41 (max)
- atemp: Normalized feeling temperature in Celsius. The values are divided to 50 (max)
- hum: Normalized humidity. The values are divided to 100 (max)
- windspeed: Normalized wind speed. The values are divided to 67 (max)
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

These variables are either categorical or continuous. Fortunately, all categorical variables are enumarated in a suitable manner and all continuous variables are already normalized. In addition, there is also no missing values in any of the variables. So, I can use these variables as is. I also define a

new variable called “isWorking” which takes value 1 if the day is a working day and not a holiday. However, it turns out that this variable is very highly correlated with “workingday” variable.

The variable that I want to forecast is “cnt”. Moreover, it is the sum of “casual” and “registered”. So, I will also consider forecasting “casual” and “registered” separately and combining the resulting forecasts to obtain “cnt”.

In Figure 1 below, I depict the average bike rental counts (“cnt”) with respect to the categorical features in the dataset. As can be seen, means of “cnt” differs between the different values of almost all features. However, some features are less promising with respect to the information it contains. For example, “holiday”, “workingday” and “isWorking” show little variability in terms of average “cnt”. But, still one can see some level of difference. So, I consider to put all these features into the prediction model except the “isWorking” as it is almost the same with “workingday”. Though I don’t report here, I also considered a subset of these features excluding “holiday” and “workingday, but the results remain almost the same for all models I will consider. Hence, I pursue the analysis using all features available. Similarly, Figure 2 gives the scatter plots of the continuous variables with respect to the “cnt”. As can be seen, there seems to be some variability in the features. However, “temp” and “atemp” seem to be highly correlated. So, in my feature set I only include “atemp” to avoid multicollinearity.

As a result, my feature set include “season”, “holiday”, “workingday”, “weathersit”, “atemp”, “hum”, “windspeed”, “yr”, “mnth”, “weekday” and “hr”.

Another consideration is that some variables might have different effects on “casual” and “registered”. So, it might be a good idea to exploit this by training different models for each of these cases and combine the results for “cnt”. As Figure 3 and Figure 4 show, casual users rent more bike on holidays and weekends and registered users rent more bike on weekdays and less on holidays. Moreover, casual users rent less relative to registered users on winter.

Figure 1: Average bike rental count with respect to the features

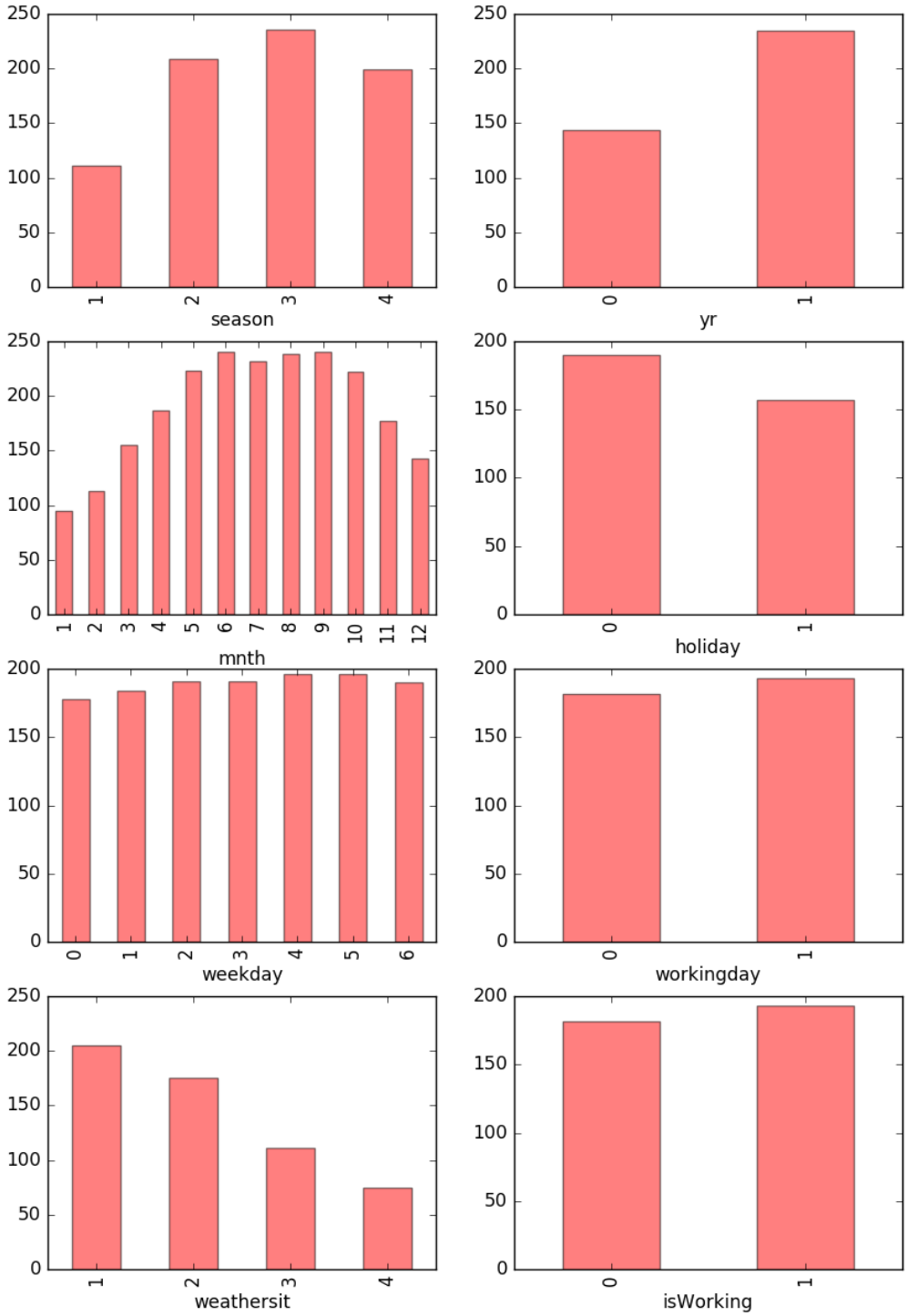


Figure 2: Scatter plots of continuous variables with respect to the bike rental count

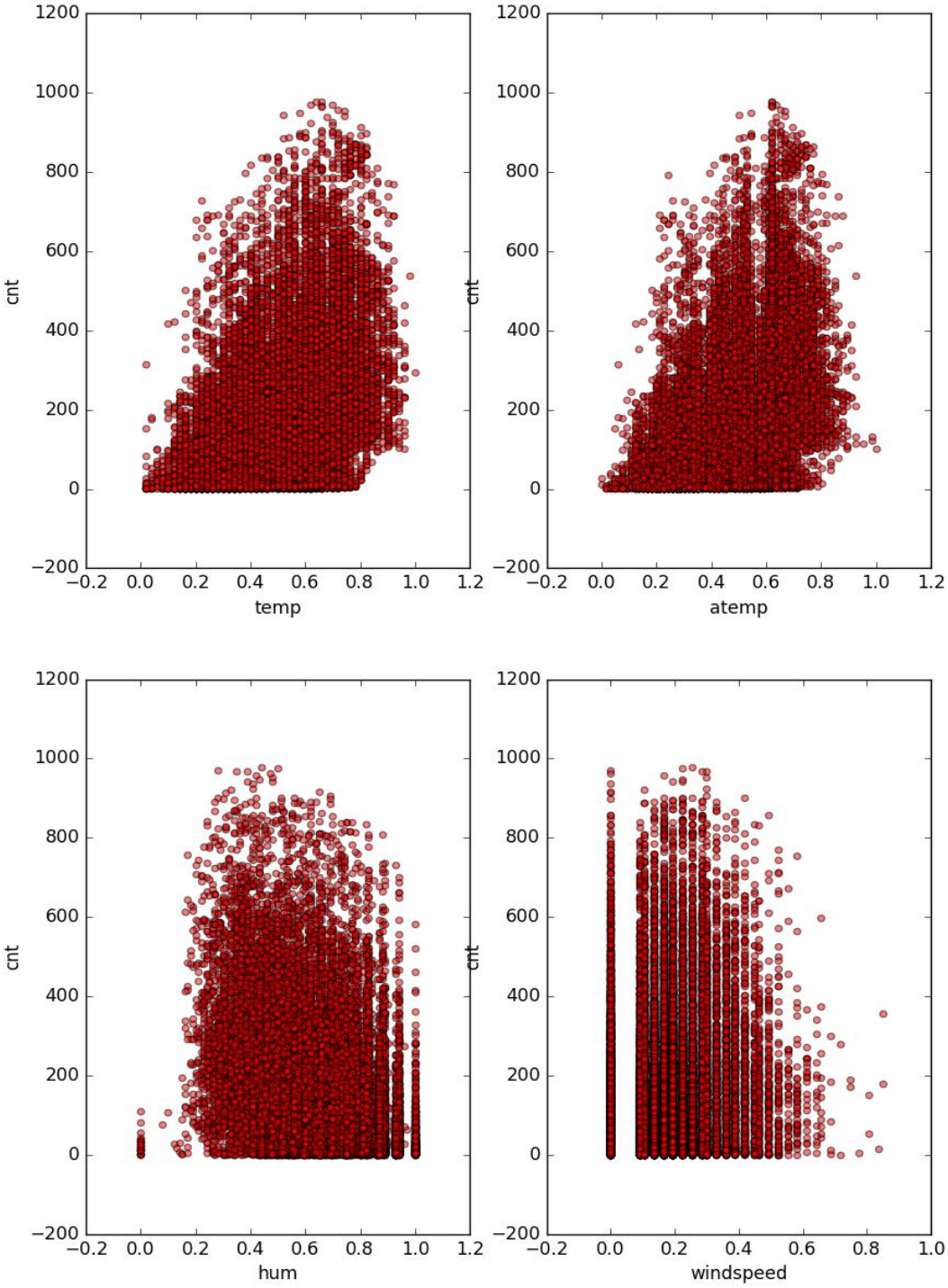


Figure 3: Average bike rental count for casual users with respect to the features

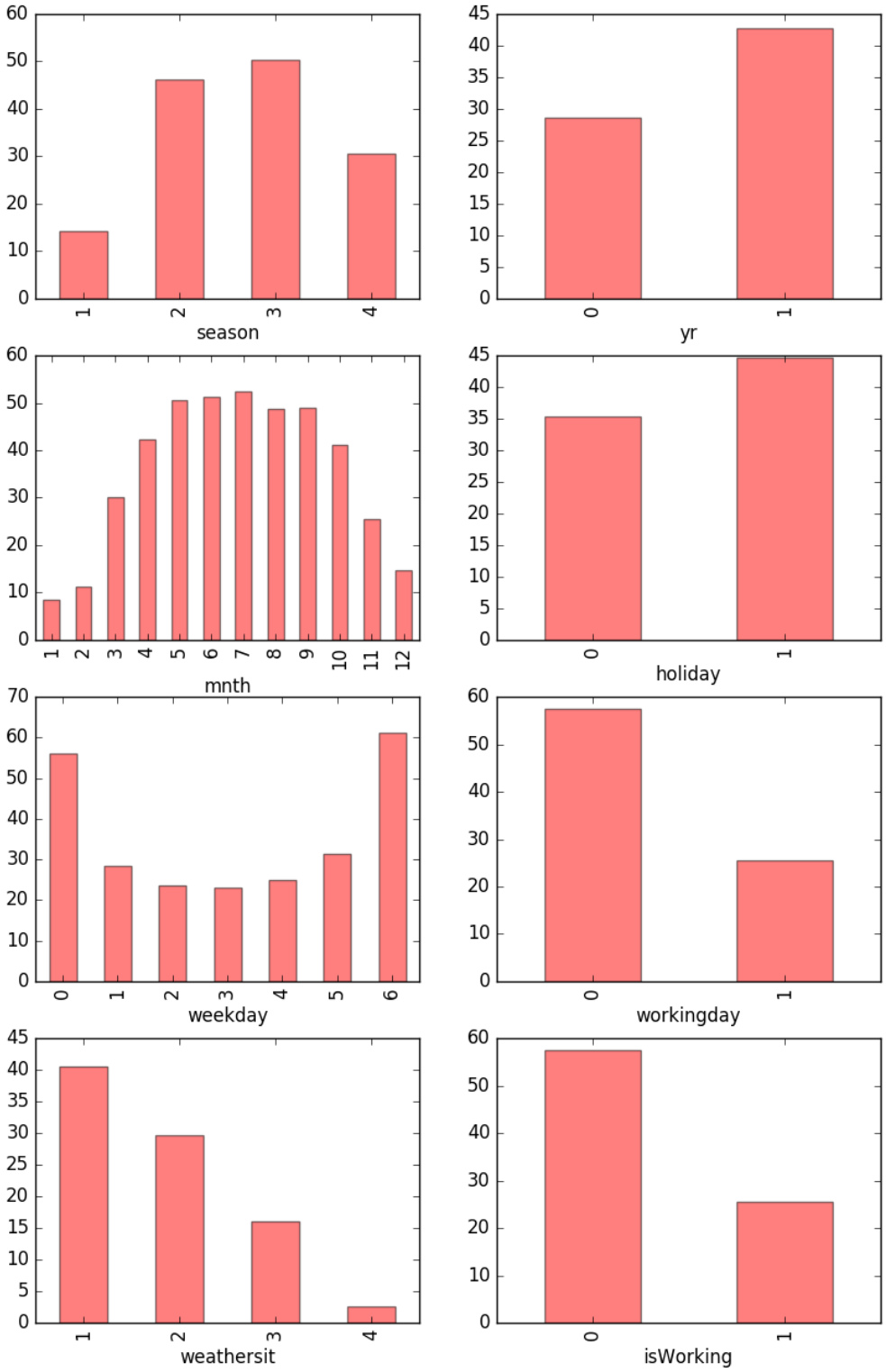
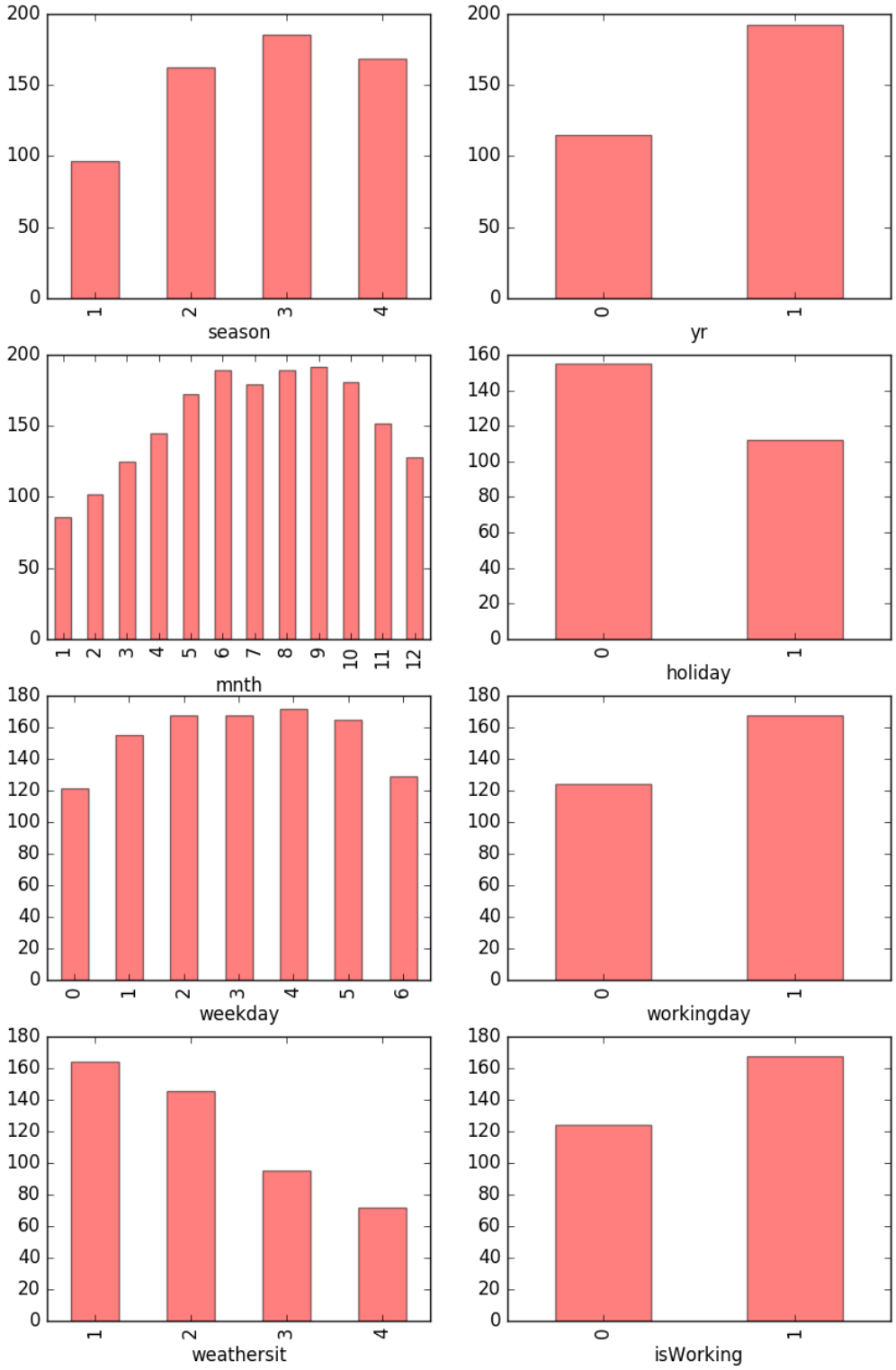


Figure 4: Average bike rental count for registered users with respect to the features



3) Model Selection and Results

My approach for the prediction is to try several methods and choose the best performing one. To this end, I randomly split the data into training and test sets such that the test set comprises the %20 of the whole sample. I train all the models in the training sample and report the Root Mean Squared Error (RMSE) on the test sample. All results are given in Table 1 below.

First, I train a simple linear regression model where the dependent variable is “cnt”. RMSE of this model in the test sample is 141.95. Then, I train the model separately for dependent variables “casual” and “registered”. The RMSEs are 36.46 and 123.33 respectively. Moreover, I combine the predictions of the models for “casual” and “registered”. The RMSE of this combined model in the test sample is 141.95. So, the combined model’s performance is the same as the initial model.

Second, I train a set of Ridge regression models in the training sample. Different than the simple linear regression model, Ridge regression has one hyperparameter to tune called “alpha” which is the penalty for the model complexity (or regularization term in the loss function). I try several values for “alpha” to choose the best performing one using grid search. The evaluation is made under a cross-validation set randomly chosen from the training set. However, the results of the Ridge regressions are almost the same as the simple linear regression models for “cnt”, “casual”, “registered” and combination. Though I do not report here, I also tried Lasso regressions and the results are almost identical with the results of Ridge regressions.

Third, I try an ensemble method. To this end, I train Gradient Boosting Regression (GBR) models. This model has four hyperparameters. To tune these hyperparameters, I use grid search. Evaluation is made under a ramdomly chosen cross-validation set from the training set. After selecting best hyperparameters, I evaluate the final model in the test set. I first train the model for the dependent variable “cnt”. The best hyperparameters are {'n_estimators': 300, 'min_samples_leaf': 11, 'max_depth': 10, 'learning_rate': 0.05}. In GBR models, the training performance of models always increase as the number of estimators increase. As the left panel of Figure 5 shows, the model actually performs almost similar for 80 estimators and 300 estimators. So, to avoid overfitting, one might choose the model with 80 estimators. In fact, it seems that after 50 estimators the performance of the model seems to be entered a phase of diminishing returns. In that sense, one can also choose 50 in very large smaples. However, for simplicity and expositional purposes I continue with the 300 estimators throughout the analysis. The RMSE of this model is 39.71 which is way better than the linear regression and Ridge regression. The right panel of Figure 5 displays the relative importances of the features for this model. It seems that “hr”, “atemp”, “hum”, “windspeed”, “weekday”, “mnt” and “yr” are the most important variables for the prediction of “cnt”.

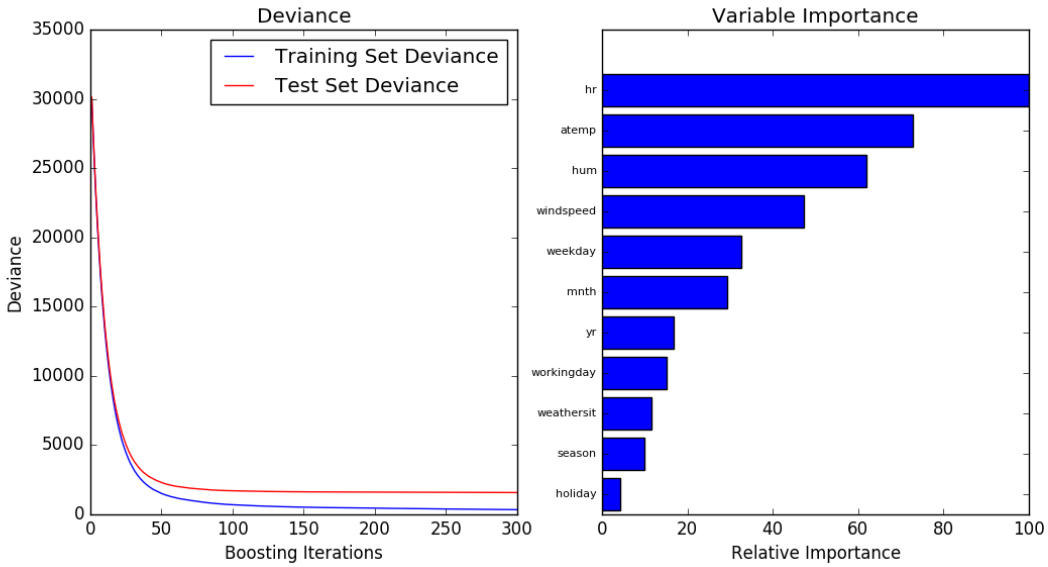
I also train the model for “casual” and “registered” as dependent variables separately. The RMSEs of these models on the test set are 14.20 and 33.34 respectively. These are also better than the RMSEs of the linear and Ridge regressions. I also combined the predictors of “casual” and “registered”. The RMSE of this combined model is 38.90 which beats the model for “cnt”.

As a result, my best performing model is GBR models trained separately for “casual” and “registered” and then combined together.

Table 1: Regression results

	Method	Dependent Variable	Best Hyperparameters	RMSE
1	Linear regression	cnt		141.95
2	Linear regression	casual		36.46
3	Linear regression	registered		123.33
4	Linear regression (casual + registered)			141.95
5	Ridge ression	cnt	{'alpha': 1}	141.95
6	Ridge ression	casual	{'alpha': 1}	36.47
7	Ridge ression	registered	{'alpha': 1}	123.33
8	Ridge ression (casual + registered)			141.95
9	Gradient Boosting Regression	cnt	{'n_estimators': 300, 'min_samples_leaf': 11, 'max_depth': 10, 'learning_rate': 0.05}	39.71
10	Gradient Boosting Regression	casual	{'n_estimators': 300, 'learning_rate': 0.05, 'max_depth': 10, 'min_samples_leaf': 11}	14.20
11	Gradient Boosting Regression	registered	{'n_estimators': 300, 'learning_rate': 0.05, 'max_depth': 10, 'min_samples_leaf': 11}	33.34
12	Gradient Boosting Regression (casual + registered)			38.90

Figure 5: GBR number of estimators evaluation and feature importances



4) Considerations on Scalability

One last consideration about the scalability: What happens if the sample size of the problem discussed here reaches very large amounts?

Fortunately, the linear regression problem suits well to parallelization. The solution to the linear regression problem has an analytical solution which involves matrix transpose, matrix multiplication and matrix inversion essentially. Matrix multiplication can be paralleled by exploiting the fact that QR or Cholesky decomposition can be distributed to several machines and the results can be aggregated in a single machine by the summation of different outer products of simple vectors. This can reduce the time and space complexity of the model almost one polynomial degree.

There are several available tools to implement this already. For example, Apache Spark provides parallelization of linear regression problems by using the above mentioned method.

5) Conclusion

To be able to predict bike demand, I tested the performances of several regression models including an ensemble method. It turned out that, the best performing model for “cnt” is the method that combines separate GBR prediction models of “casual” and “registered”.

Appendix – Python Code

```
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.linear_model import Ridge
from sklearn import ensemble
from sklearn.cross_validation import train_test_split
from sklearn.grid_search import GridSearchCV
import matplotlib.pyplot as plt
import math
```

```
if __name__ == '__main__': #This is for the sake of parallelization

    #take the original data
    data_path = "C:/Users/YEB/Desktop/bike/"

    data = pd.read_csv(data_path+"hour.csv", header=0, delimiter=",")

    #load the data on a data frame
    df = pd.DataFrame(data)

    #create a variable called isWorking
    work_day = df['workingday'].as_matrix()
    holiday = df['holiday'].as_matrix()
    df['isWorking'] = np.where(np.logical_and(work_day == 1, holiday == 0), 1, 0)

    #prepare some lists of features for the visualization and feature engineering
    #bar is a list of categorical variables
    bar =
["season","yr","mnth","holiday","weekday","workingday","weathersit","isWorking"]
    #scat is a list of continuous variables
    scat = ["temp","atemp","hum","windspeed"]

    #bar plots and scatter plots
    i=0
    fig, axes= plt.subplots(nrows=4,ncols=2)
    for col in bar:
        means = df.groupby(col).cnt.mean()
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'bar', x = 0, y = 1, ax = ax, alpha =
0.5,color='r',fontsize=12,figsize=(10,15))

    fig.savefig('bar_cnt.png')

    i=0
    fig, axes= plt.subplots(nrows=2,ncols=2)
    for col in scat:
        means = df[[col,"cnt"]]
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'scatter', x = 0, y = 1, ax = ax, alpha = 0.5,color='r',
fontsize=12,figsize=(10,15))

    fig.savefig('scatter_cnt.png')

    i=0
    fig, axes= plt.subplots(nrows=4,ncols=2)
    for col in bar:
        means = df.groupby(col).casual.mean()
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'bar', x = 0, y = 1, ax = ax, alpha =
0.5,color='r',fontsize=12,figsize=(10,15))
```

```
    fig.savefig('bar_casual.png')

    i=0
    fig, axes= plt.subplots(nrows=2,ncols=2)
    for col in scat:
        means = df[[col,"casual"]]
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'scatter', x = 0, y = 1, ax = ax, alpha = 0.5,color='r',
fontsize=12,figsize=(10,15))

    fig.savefig('scatter_casual.png')


    i=0
    fig, axes= plt.subplots(nrows=4,ncols=2)
    for col in bar:
        means = df.groupby(col).registered.mean()
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'bar', x = 0, y = 1, ax = ax, alpha =
0.5,color='r',fontsize=12,figsize=(10,15))

    fig.savefig('bar_reg.png')

    i=0
    fig, axes= plt.subplots(nrows=2,ncols=2)
    for col in scat:
        means = df[[col,"registered"]]
        colInd = i%2
        rowInd = math.floor(i/2)
        i=i+1
        ax = axes[rowInd,colInd]
        means.plot(kind = 'scatter', x = 0, y = 1, ax = ax, alpha = 0.5,color='r',
fontsize=12,figsize=(10,15))

    fig.savefig('scatter_reg.png')

    #split the original data randomly to training and test set.
    train, test = train_test_split(data, test_size=0.20, random_state=98745)

    #use the features below for prediction
    features1 = ['season', 'holiday', 'workingday', 'weathersit',
            'atemp', 'hum', 'windspeed', 'yr',
            'mnth', 'weekday', 'hr']


    #fit a linear regression model with features1
    regr = linear_model.LinearRegression()
    regr.fit(train[features1], train["cnt"])
    resultRegr1 = regr.predict(test[features1])

    #print some metrics
    print("Linear regression on count - Root Mean Squared Error: %.2f"
```

```
          % np.sqrt(np.mean((resultRegr1 - test["cnt"]) ** 2)))


    #fit a linear regression model with features1 on casual
    regr.fit(train[features1], train["casual"])
    resultRegr3 = regr.predict(test[features1])

    #print some metrics
    print("Linear regression on casual - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultRegr3 - test["casual"]) ** 2)))


    #fit a linear regression model with features1 on registered
    regr.fit(train[features1], train["registered"])
    resultRegr4 = regr.predict(test[features1])

    #print some metrics
    print("Linear regression on registered - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultRegr4 - test["registered"]) ** 2)))

    print("Linear regression on casual and registered combined - Root Mean Squared
Error: %.2f"
          % np.sqrt(np.mean((resultRegr3 + resultRegr4 - test["cnt"]) ** 2)))


    #This includes several values for alpha to tune
    param_grid = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 50,
100, 1000]}

    #fit a Ridge regression model with features1
    rig = Ridge()
    rigBest = GridSearchCV(rig, param_grid, n_jobs=-1).fit(train[features1],
train['cnt'])

    # print the best hyperparameters
    print("Best alpha of the ridge regression on count: " +
str(rigBest.best_params_))
    resultRidge = rigBest.predict(test[features1])

    #print some metrics
    print("Ridge ression on count - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultRidge - test["cnt"]) ** 2)))


    #fit a Ridge regression model with features1 on casual
    rigBest = GridSearchCV(rig, param_grid, n_jobs=-1).fit(train[features1],
train['casual'])

    # print the best hyperparameters
    print("Best alpha of the ridge regression on casual: " +
str(rigBest.best_params_))
    resultRidgeCas = rigBest.predict(test[features1])

    #print some metrics
    print("Ridge ression on casual - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultRidgeCas - test["casual"]) ** 2)))


    #fit a linear regression model with features1 on registered
```

```
    rigBest = GridSearchCV(rig, param_grid, n_jobs=-1).fit(train[features1],
train['registered'])

    # print the best hyperparameters
    print("Best alpha of the ridge regression on registered: " +
str(rigBest.best_params_))
    resultRidgeReg = rigBest.predict(test[features1])

    #print some metrics
    print("Ridge ression on registered - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultRidgeReg - test["registered"]) ** 2)))

    print("Ridge ression on casual and registered combined - Root Mean Squared
Error: %.2f"
          % np.sqrt(np.mean((resultRidgeCas + resultRidgeReg - test["cnt"]) **
2)))


    # This includes some values for the hyperparameters of GBR for tuning
    param_grid = {'n_estimators': [100, 200,300],'learning_rate': [0.1, 0.05,
0.01], 'max_depth': [10, 20],'min_samples_leaf': [3, 7, 11]}

    #fit a GBR model on cnt
    error = 1000000
    est = ensemble.GradientBoostingRegressor(warm_start=True)
    gbr1 = GridSearchCV(est, param_grid, n_jobs=-1).fit(train[features1],
train['cnt'])

    # print the best hyperparameters
    print(gbr1.best_params_)
    resultGbr1 = gbr1.predict(test[features1])

    #print some metrics
    print("GBR (cnt) - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultGbr1 - test["cnt"]) ** 2)))

    #replicate the best model for visualization
    est =
ensemble.GradientBoostingRegressor(n_estimators=gbr1.best_params_['n_estimators'],

learning_rate=gbr1.best_params_['learning_rate'],

max_depth=gbr1.best_params_['max_depth'],

min_samples_leaf=gbr1.best_params_['min_samples_leaf'])
    est.fit(train[features1], train['cnt'])

    # compute test set deviance
    test_score = np.zeros((max(param_grid['n_estimators']),), dtype=np.float64)

    for i, y_pred in enumerate(est.staged_predict(test[features1])):
        test_score[i] = est.loss_(test["cnt"], y_pred)

    fig = plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.title('Deviance')
    plt.plot(np.arange(max(param_grid['n_estimators'])) + 1, est.train_score_, 'b-
',
             label='Training Set Deviance')
```

```
    plt.plot(np.arange(max(param_grid['n_estimators'])) + 1, test_score, 'r-',
             label='Test Set Deviance')
    plt.legend(loc='upper right')
    plt.xlabel('Boosting Iterations')
    plt.ylabel('Deviance')

    # Plot feature importance
    feature_importance = est.feature_importances_
    # make importances relative to max importance
    feature_importance = 100.0 * (feature_importance / feature_importance.max())
    sorted_idx = np.argsort(feature_importance)
    pos = np.arange(sorted_idx.shape[0]) + .5
    plt.subplot(1, 2, 2)
    plt.barh(pos, feature_importance[sorted_idx], align='center')
    plt.yticks(pos,[features1[i] for i in sorted_idx], fontsize=8)
    plt.xlabel('Relative Importance')
    plt.title('Variable Importance')
    fig.savefig("importance.png")


    #fit a GBR model on casual
    est = ensemble.GradientBoostingRegressor(warm_start=True)
    gbrCasual = GridSearchCV(est, param_grid, n_jobs=-1).fit(train[features1],
train['casual'])

    # print the best hyperparameters
    print(gbrCasual.best_params_)

    resultGbrCasual = gbrCasual.predict(test[features1])
    print("GBR (Casual) - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultGbrCasual - test["casual"]) ** 2)))


    #fit a GBR model on registered
    est = ensemble.GradientBoostingRegressor(warm_start=True)
    gbrRegistered = GridSearchCV(est, param_grid, n_jobs=-1).fit(train[features1],
train['registered'])

    # print the best hyperparameters
    print(gbrRegistered.best_params_)

    resultGbrRegistered = gbrRegistered.predict(test[features1])
    print("GBR (Registered) - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultGbrRegistered - test["registered"]) ** 2)))

    print("GBR (Casual + Registered) - Root Mean Squared Error: %.2f"
          % np.sqrt(np.mean((resultGbrCasual + resultGbrRegistered - test["cnt"])
** 2)))
```