Bike Renting Project Explanation

This project is a simplified (basic) version of "Bike Rental" project (without any data analysis step).

The objective of the project is - using historical usage patterns and weather data, forecast(predict) bike rental demand (number of bike users (‘cnt’)) on hourly basis.

Use the provided “Bikes Rental” data set to predict the bike demand (bike users count - 'cnt') using various best possible models (ML algorithms). Also, report the model that performs best, and fine-tune the same model using one of the model fine-tuning techniques, and report the best possible combination of hyperparameters for the selected model.

Lastly, use the selected model to make final predictions and compare the predicted values with the actual values.

Acknowledgements:

Cloudxlab is using this “Bike Sharing Demand” problem for its machine learning learners for learning and practicing. This dataset was provided by Hadi Fanaee Tork using data from Capital Bikeshare. We also thank the UCI machine learning repository for hosting the dataset.

Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

Basic Description

Objective -

The objective of the project is - using historical usage patterns and weather data, forecast(predict) bike rental demand (number of bike users (‘cnt’)) on hourly basis.

Use the provided “Bikes Rental” data set to predict the bike demand (bike users count - 'cnt') using various best possible models (ML algorithms). Also, report the model that performs best, fine-tune the same model using one of the model fine-tuning techniques, and report the best possible combination of hyperparameters for the selected model. Lastly, use the selected model to make final predictions and compare the predicted values with the actual values.

Below are the details of the features list for the given Bikes data set:

instant: record index

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: whether the day is a holiday or not

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 derived via (tt_min)/(t_maxt_min), t_min=*8, t_max=+39 (only in hourly scale)

atemp: Normalized feeling temperature in Celsius. The values are derived via (tt_min)/(t_maxt_min), t_min=*16, t_max=+50 (only in hourly scale)

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 users

The "target" data set ('y') should have only one 'label' i.e. 'cnt'.

We will be following the below steps to solve this problem:

1) Importing the libraries
2) Using some pre-defined utility functions
3) Loading the data
4) Cleaning the data
5) Dividing the dataset into training and test dataset
6) using train_test_split in the ratio 70:30
7) Training several models and analyzing their performance to select a model
8) Fine-tuning the model by finding the best hyper-parameters and features
9) Evaluating selected model using test dataset

Importing the Libraries

Please import the required libraries as mentioned below:

1) Import numpy as np
2) Import pandas as pd
3) From sklearn import preprocessing
4) Please import StandardScaler from Scikit Learn - preprocessing
5) Please import mean_squared_error from Scikit Learn - metrics
6) Please import linear_model from Scikit Learn
7) Please import matplotlib's pyplot as plt
8) Import os

Loading the data

To start with, let us load the "Bikes Rental" data set (bikes.csv) and get a first hand look into the same.

The dataset contains the following parameters:

1. **instant:** record index
2. **dteday:** date
3. **season:** season (1: springer, 2: summer, 3: fall, 4: winter)
4. **yr:** year (0: 2011, 1:2012)
5. **mnth:** month (1 to 12)
6. **hr:** hour (0 to 23)
7. **holiday:** whether day is holiday or not
8. **weekday:** day of the week
9. **workingday:** if day is neither weekend nor holiday is 1, otherwise is 0.
10. **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
11. **temp:** Normalized temperature in Celsius.
12. **atemp:** Normalized feeling temperature in Celsius.
13. **hum:** Normalized humidity. The values are divided to 100 (max)

14. **windspeed:** Normalized wind speed. The values are divided to 67 (max)
15. **casual:** count of casual users
16. **registered:** count of registered users
17. **cnt:** count of total rental bikes including both casual and registered

Cleaning the Data – Dropping the unwanted features

As we can see, the bikesData data set contains some features which may not be needed for this problem e.g.

1. **instant** - This is just an index holder.
2. **casual** - It contains the count of casual bike riders, which are already included in the overall users count i.e. 'cnt', hence not needed.
3. **registered** - It contains the registered bike riders, which again, like the casual riders, are already included in the overall users count i.e. 'cnt', hence not needed.
4. **atemp** - It is the 'feel' temperature, which may not be needed, as we already have 'temp' feature which contains temperature of the area - duplicate feature, hence not needed.
5. **dteday** - It is the Date. Since, our prediction is not based on the Date, its based on hour of the day, hence Date is not needed.

Since, these features are not needed for our current problem, let us drop them from the bikesData data set.

Divide into training/ test dataset

Now, since we have cleaned the bikesData data set, let us split it into Training and Test data sets into 70:30 ratio using scikit-learn's train_test_split() function.

Also, train_test_split() function uses 'Random Sampling', hence
resulting train_set and test_set data sets have to be sorted by dayCount. Random Sampling may not be the best way to split the data.

Defining some utility functions

This function is used to calculate the basics stats of observed scores from cross-validation of models.

```
def display_scores(scores):

    print("Scores:", scores)

    print("Mean:", scores.mean())

    print("Standard deviation:", scores.std())
```

Cleaning the data – Feature Scaling

Few of the features of bikesData data set may need Scaling. For example, if we see the values of features - temp, hum and windspeed - their values are in different scales (ranges), hence, we need to apply Scaling to these features values for our ML algorithms to work fine on them.

Train and Analyze the Models - Preparing to Train the Models

In the previous step, we split the bikesData into Training data set (train_set) and Test data set (test_set), but can we use this 'Training' data set as it is, or, we need to do some more refinement before we can use it to generate ML models?

Yes, you got it right, we need to create target data set, and also, we need to create a final 'Training' data set by dropping the target label from the existing 'Training' data set (train_set).

Train and Analyze the Models - Train DecisionTree Model

Train the Decision Tree Model on the 'Training' data set using cross-validation and calculate 'mean absolute error' and 'root mean squared error' (RMSE) for this model.

Display these scores using display_scores() function.

Train and Analyze the Models - Train Linear Regression Model

Train the Linear Regression Model on the 'Training' data set using cross-validation and calculate 'mean absolute error' and 'root mean squared error' (RMSE) for this model.

Display these scores using display_scores() function.

Train and Analyze the Models - Train Random Forest Model

Train the Random Forest Model on the 'Training' data set using cross validation and calculate 'mean absolute error' and 'root mean squared error' (RMSE) for this model.

Display these scores using display_scores() function.

Fine-Tuning the Selected Model - Choosing set of hyperparameter combinations for Grid Search

Now, since we have "Trained" the models and selected the best model based on the performance measure(RMSE) values, let us apply Grid Search on this selected model to fine-tune the model (i.e. find the best hyperparameters for this model).

As first step for the same, let us import GridSearchCV and choose the set of hyperparameter combinations in the form of a 'parameter grid', which we will use to apply the Grid Search.

Fine-Tuning the Selected Model - Defining GridSearchCV

Let us instantiate the GridSearchCV class by passing to it the selected model and other required parameters.

Fine-Tuning the Selected Model - Run GridSearchCV

Let us run the GridSearchCV to get the Best Estimator and the Best Parameters

Fine-Tuning the Selected Model - Knowing Feature Importances

Now, since we have applied the Grid Search on the selected model and got the 'best estimator' and 'best parameters', let us find the importance scores of each of the features of the Training dataset.

Evaluate the model on test - Preparing to test the final model on Test dataset

Now, since, we got the best (final) model (using Grid Search) for this problem, let us use the same on the 'Test' data set to predict the 'cnt' values and then compare the predicted values to the actual values.

Evaluate the model on test - Make Predictions on the Test dataset using Final Model

Let us use the Final Model to make predictions on the 'Test' data set and calculate the RMSE. Then compare the predicted values to the actual values.