In this answer, three models have been build by using given data to predict hourly entries in NYC subways .

The first model is a simple linear regression model.

The second one is a two-level regression model.

The third one is a nonparametric model.

This article is divided into 5 parts.

1, Some observations and thoughts from data.

2, General principles of models.

3, Three models.

4, Summary.

5, Possible further improvements.

Some observtions and thoughts from data

The dataset is about ridership of NYC subways. Here is a typical record in the dataset.

In [5]:

```
from linear_models import load_data
dataframe = load_data()
print dataframe.iloc[0]
```

```
Unnamed: 0                  0
UNIT                     R001
DATEn                2011-05-01
TIMEn                  01:00:00
Hour                          1
DESCn                   REGULAR
ENTRIESn_hourly               0
EXITSn_hourly                 0
maxpressurei              30.31
maxdewpti                    42
mindewpti                    35
minpressurei              30.23
meandewpti                   39
meanpressurei             30.27
fog                           0
rain                          0
meanwindspdi                  5
mintempi                     50
meantempi                    60
maxtempi                     69

precipi                       0
thunder                       0
Name: 0, dtype: object
```

Following is the meaning of various data fields in this record.

Unnamed: 0: Meaningless.

UNIT: Remote unit that collects turnstile information. Can collect from multiple banks of turnstiles. Large subway stations can have more than one unit.

DATEn: Date in “yyyy-mm-dd” (2011-05-21) format.

TIMEn: Time in “hh:mm:ss” (08:05:02) format.

hour: Hour of the timestamp from TIMEn. Truncated rather than rounded.

DEScn: Represent the "REGULAR" scheduled audit event (occurs every 4 hours).

ENTRIESn_hourly: Difference in ENTRIES from the previous REGULAR reading.

EXITSn_hourly: Difference in EXITS from the previous REGULAR reading.

maxpressurei: Maximum pressure in inHg.

maxdewpti: Maximum dewpoint in F.

mindewpti: Minimum dewpoint in F.

minpressurei: Minimum pressure in inHg.

meandewpti: Mean dewpoint in F.

meanpressurei: Mean pressure in inHg.

fog: Indication of appearance of fog.

rain: Indication of appearance of rain.

meanwindspdi: Mean windspeed in mph.

mintempi: Minimum temperature in F.

meantempi: Mean temperature in F.

maxtempi: Maximum temperature in F.

precipi: Precipitation in inches.

thunder: Indication of appearance of thunder.

A model should be build based on the dataset. In this model the date item "ENTRIESn_hourly" is the value to be predicted. This is the output of the model. The other items which represent date, time and weather conditions are the values used to predict "ENTRIESn_hourly". These are inputs of the model. These inputs are also called features. The center task of model is using features to calculate output.

Since this task is to predict numerical values. The simplest model is by unifying some features into a simple linear models. The simple linear model have following form.

$$y = \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_k * x_k + \epsilon$$

y is the output. $x_1, x_2 \ldots x_k$ are the values of features. $\beta_1, \beta_2 \ldots \beta_k$ are so called weights or coefficients which will be determined by training process. ϵ is the estimated error.

Assumptions are made by utlizing this simple linear model. One important assumption is independence of features. That is the change of one feature will not effect other features. However, some observations may indicate that the features may be dependent on each other. Here is the form that show some correlation between feature "rain" and fearure "hour" in a particular turnstile. The values in the form is the average "hourly_entries" given the "rain" and "hour".

In [6]:

```
from observations import rain_correlated_time
unit_number = 1
time_number = 5
print rain_correlated_time(unit_number, time_number)
```

```
              no_rain    rain
17:00:00  5734.750000  5259.2
13:00:00  6423.850000  6538.6
09:00:00  4340.736842  4554.5
01:00:00  1990.263158  1711.6
21:00:00  4644.157895  4900.4
```

As we can see, for some "hour", the average "hourly_entries" is higher when "rain" is 1. And for other, the average "hourly_entries" is lower. This observation shows that the feature "rain" is dependent on the feature "hour". This observation is intuitive. When it is raining(corresponding to feature "rain") in mid-night(corresponding to feature "hour"), the ridership of subway may not be affected by rain. And if it was raining in rush hour, more people may take subway to work because the traffic on the road may become terrible in rainy days. These thoughts on causation may not be accurate. The center idea is features are not independent. A model with consideration of correlation should be build. Here is a typical formula of two-level regression.

$$y = \beta_1(u_1, u_2 \ldots u_l) * x_1 + \beta_2(u_1, u_2 \ldots u_l) * x_2 + \ldots + \beta_k(u_1, u_2 \ldots u_l) * x_k + \epsilon$$
$$\beta_1(u_1, u_2 \ldots u_l) = f_1(u_1, u_2 \ldots u_l)$$
$$\beta_2(u_1, u_2 \ldots u_l) = f_2(u_1, u_2 \ldots u_l)$$
$$\ldots$$
$$\beta_k(u_1, u_2 \ldots u_l) = f_k(u_1, u_2 \ldots u_l)$$

y is the output. $x_1, x_2 \ldots x_k$ are the values of level-two features.
$\beta_1(u_1, u_2 \ldots u_l), \beta_2(u_1, u_2 \ldots u_l) \ldots \beta_k(u_1, u_2 \ldots u_l)$ are so called coefficients functions which will be determined by model and training process. ϵ is the estimated error. $u_1, u_2 \ldots u_l$ are values of level-one features.

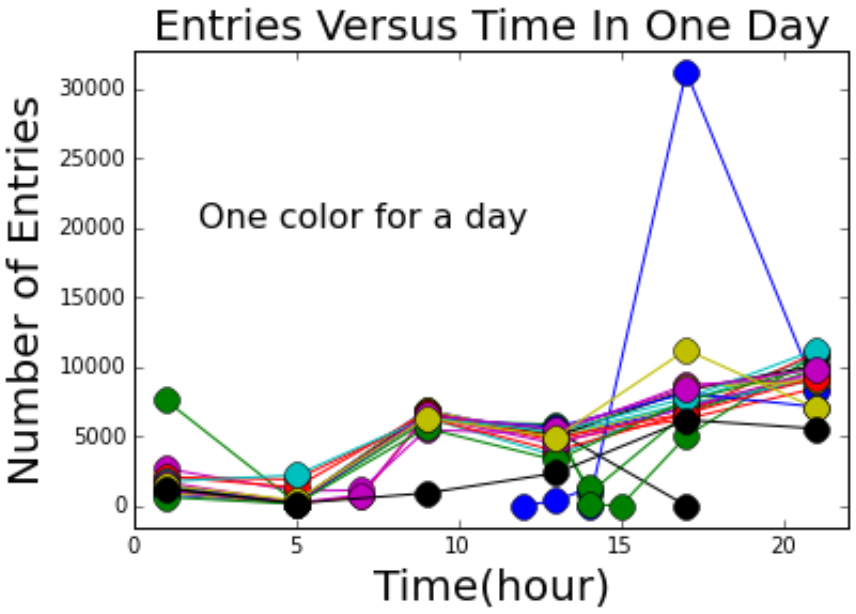
Furthermore, when people talked about crowdedness of the subways. They may say, there are a lot people in this station 5:00 pm yestoday and a lot people will be in this station at the same time. Following figures show ridership in different days at and in one particular station.

In [7]:

```
%pylab inline
from observations import plot_daily_data
import observations
unit_number = 1
plot_daily_data(unit_number)
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['rando
m']
`%matplotlib` prevents importing * from pylab and numpy
```



Out[7]:

```
<module 'matplotlib.pyplot' from '/home/adenguo/anaconda/lib/p
ython2.7/site-packages/matplotlib/pyplot.pyc'>
```

As the picture reveals, the ridership is periodical. The similar amount of hourly entries will appear repeatedly based on same time of different days. This provide the philosophy of non-parametric model. The general formula of non-parametric model is as following.

$$y = f(D_1, D_2, \dots D_n)$$

y is the output. $D_1, D_2, \dots D_n$ are records from dataset. $f()$ is a general form of function.

General principles of model constructions

First, the features are the same for three models. I select following features as input of models.

'unit'(I translated this variable into a dummy variable indicating if a record belongs to a specific turnstile in linear model and two-level regression model)

'hour'(I translate this variable to 'dec_time')

'meantempi'

'rain'

'meanwindspdi'

'precipi'

'DATEn'(I translated this variable to 'weeksday' which represent Sunday, Monday and etc. in a numerical way and 'is_weekend' which using bool value represent if it is a weekend.)

However, these features play different roles in three models.

I also translate the output 'ENTRIESn_hourly' into 'hourly_entries' in implementation of three models.

Second, the whole dataset is split into training and test dataset. Three model are all trained based on same training data and evaluated on test data. Two measurements for models are used which are sum squared error(SSE) and R-square(R2) value. The split of dataset were done for ten times. And each model is trained and evaluated for ten times.

Three models

Simple linear model

Features : the model was built as a conventional linear model. Intercept is included in this model. These features of the model are dummy variables for 'unit'(total 465 variables), 'dec_time', 'meantempi', 'rain', 'meanwindspdi', 'precipi' and 'is_weekend'.

Training: the model is trained using ordinary least squares(OLS) implemented in statsmodels package.

Predicting: Values of the record are directly used to evaluate the number of hourly entries in the linear regression function. And a little non-linearity is done on the predicted value. If the predicted value is small than zero. It is set to zero.

Following is the training and evaluating process.

In [8]:

```
from linear_models import *

#Set time_intervals for a simple linear model

time_intervals = [(0,23.999,weeks_day)]

#features will be used in linear models
features_list = ['rain','meantempi', 'dec_time','meanwindspdi','precipi',
'is_weekend']

#Phase 1, preprocessing and save result.
dataframe = load_data()
new_dataframe = create_new_dateframe_0(dataframe)
new_dataframe = add_datetime_str(new_dataframe)
new_dataframe = add_dec_time(new_dataframe)
new_dataframe = add_weeksday(new_dataframe)
new_dataframe = add_is_weekend(new_dataframe)
new_dataframe, dummy_unit_list = add_dummy_unit(new_dataframe)
features_list.extend(dummy_unit_list)
write_features_to_file(features_list, '../features_list')
new_dataframe.to_csv('../new_weather_turnstile.csv')
dataframe = pandas.read_csv('../new_weather_turnstile.csv')
from sklearn.cross_validation import train_test_split
test_propotion = 0.1
train_new_weather_turnstile, test_new_weather_turnstile = \
train_test_split(dataframe, test_size = test_propotion)
train_new_weather_turnstile.to_csv('../train_new_weather_turnstile.csv')
test_new_weather_turnstile.to_csv('../test_new_weather_turnstile.csv')

#Phase 2, building model and save result
features_list = read_features_from_file('../features_list')
dataframe = pandas.read_csv('../train_new_weather_turnstile.csv')
coeff_matrix = create_coeff_matrix(features_list, time_intervals, datafram
e)
coeff_matrix.to_csv('../coeff_matrix.csv')

#Phase 3, Test and evaluating
features_list = read_features_from_file('../features_list')
coeff_matrix = pandas.read_csv('../coeff_matrix.csv')
test_dataframe = pandas.read_csv('../test_new_weather_turnstile.csv')
predictions = make_predictions_dataframe(test_dataframe, time_intervals,\
                                         features_list, coeff_matrix)
sum_square_complex = compute_sum_square(test_dataframe['hourly_entries'],
predictions)
r_square_complex = compute_r_squared(test_dataframe['hourly_entries'], pre
dictions)

print "SSE is "+ str(sum_square_complex)
print "R2 is "+ str(r_square_complex)
```

```
SSE is 40161141139.4
R2 is 0.492777531908
```

On this calculation, the SSE is 38615772513.4 and R2 is 0.467863803062.

Complexity: computational complexity is evaluated on predicting process. The time of predicting a record is O(1) which is constant time for evaluating.

Two level regression model

Features : the features are grouped into two level. Coefficients of level two features are functions of level one features (see the formula above). Level two features are 'unit'(total 465 variables), 'dec_time', 'meantempi', 'rain', 'meanwindspdi' and 'precipi'. Level one features are 'dec_time' and 'weeksday'. It is worth noting that 'dec_time' is both level one and level two feature. I think that this variable is correlated with itselves.

Training: the level-one features are divided into intervals such as 00:00 to 05:00 in weeksday and 13:00 to 16:00 in weekend. And the training data are filtered by these intervals. Each interval will be corresponding to a subset of data. Then a linear regression model with level two features was training based on each of these subsets. every model is trained using ordinary least squares(OLS) implemented in statsmodels package.

Predicting: The predicting is a two-step process. First step is choosing which group of coefficients to use based on level one features. Second step is computing number of hourly entries based on chosen coefficients and level two features. Also, a little non-linearity is done on the predicted value. If the predicted value is small than zero. It is set to zero.

Following is the training and evaluating process.

In [9]:

```
from linear_models import *

#Set time_intervals for a simple linear model
time_intervals = [(0,4,works_day), (4,8,works_day), (8,12,works_day), \
                  (12,16,works_day), (16,20,works_day), (20,23.999,works_d
ay),\
                  (0,4,weekend), (4,8,weekend), (8,12,weekend), \
                  (12,16,weekend), (16,20,weekend), (20,23.999,weekend)]

#features will be used in linear models
features_list = ['rain','meantempi', 'dec_time','meanwindspdi','precipi']

#Phase 1, preprocessing and save result.
dataframe = load_data()
new_dataframe = create_new_dateframe_0(dataframe)
new_dataframe = add_datetime_str(new_dataframe)
new_dataframe = add_dec_time(new_dataframe)
new_dataframe = add_weeksday(new_dataframe)
new_dataframe = add_is_weekend(new_dataframe)
new_dataframe, dummy_unit_list = add_dummy_unit(new_dataframe)
features_list.extend(dummy_unit_list)
write_features_to_file(features_list, '../features_list')
new_dataframe.to_csv('../new_weather_turnstile.csv')
dataframe = pandas.read_csv('../new_weather_turnstile.csv')
from sklearn.cross_validation import train_test_split

test_propotion = 0.1
train_new_weather_turnstile, test_new_weather_turnstile = \
train_test_split(dataframe, test_size = test_propotion)
train_new_weather_turnstile.to_csv('../train_new_weather_turnstile.csv')
test_new_weather_turnstile.to_csv('../test_new_weather_turnstile.csv')

#Phase 2, building model and save result
features_list = read_features_from_file('../features_list')
dataframe = pandas.read_csv('../train_new_weather_turnstile.csv')
coeff_matrix = create_coeff_matrix(features_list, time_intervals, datafram
e)
coeff_matrix.to_csv('../coeff_matrix.csv')

#Phase 3, Test and evaluating
features_list = read_features_from_file('../features_list')
coeff_matrix = pandas.read_csv('../coeff_matrix.csv')
test_dataframe = pandas.read_csv('../test_new_weather_turnstile.csv')
predictions = make_predictions_dataframe(test_dataframe, time_intervals,\
                                         features_list, coeff_matrix)
sum_square_complex = compute_sum_square(test_dataframe['hourly_entries'],
predictions)
r_square_complex = compute_r_squared(test_dataframe['hourly_entries'], pre
dictions)

print "SSE is "+ str(sum_square_complex)
print "R2 is "+ str(r_square_complex)
```

```
SSE is 12216177511.5
R2 is 0.837844779438
```

On this calculation, the SSE is 11897041343.3 and R2 is 0.837420573143.

Complexity: computational complexity is evaluated on predicting process. The time of predicting a record is O(1) which is constant. This constant time is a little longer than that of simple linear model. But I think the difference can be safely neglected.

Non-parametric model

Features: Features for non-parametric model are 'unit', 'dec_time', 'meantempi', 'rain', 'meanwindspdi', 'precipi' and 'is_weekend'.

Training and predicting: Features are divided into two groups. One is categorical features and the other is numerical features. A subset of data is created by selecting records whose categorical feature are identical to the record(unknown record) which is under predicting. The weighted euclidean distances between unknown record and every record in the subset are calculated based on numerical features. k records with smallest distance are selected. The number of hourly entries are average of hourly entries of these k record.

Following is the training and evaluating process.

In [10]:

```
from nonparametric_model import *


#Set features for nonparametric model
numeric_features_list_with_weight = [('meantempi',1), ('dec_time',2), \
                                     ('meanwindspdi',1),('precipi',1)]
categorical_features = ['unit', 'rain', 'is_weekend']
delta = 2
k = 10

#Phase 1, preprocessing and save result.
dataframe = load_data()
new_dataframe = create_new_dateframe_0(dataframe)
new_dataframe = add_datetime_str(new_dataframe)
new_dataframe = add_dec_time(new_dataframe)
new_dataframe = add_weeksday(new_dataframe)
new_dataframe = add_is_weekend(new_dataframe)
write_features_with_weight_to_file(numeric_features_list_with_weight, \
                                   '../numeric_features_list_weight')
write_features_to_file(categorical_features, '../categorical_features')
new_dataframe.to_csv('../new_weather_turnstile.csv')
dataframe = pandas.read_csv('../new_weather_turnstile.csv')
from sklearn.cross_validation import train_test_split
test_propotion = 0.1
train_new_weather_turnstile, test_new_weather_turnstile = \
train_test_split(dataframe, test_size = test_propotion)
train_new_weather_turnstile.to_csv('../train_new_weather_turnstile.csv')
test_new_weather_turnstile.to_csv('../test_new_weather_turnstile.csv')

#Phase 2, predicting and evaluating

numeric_features_list_with_weight = read_features_with_weight_from_file \
                                    ('../numeric_features_list_weight')
categorical_features = read_features_from_file('../categorical_features')
dataframe = pandas.read_csv('../train_new_weather_turnstile.csv')
test_dataframe = pandas.read_csv('../test_new_weather_turnstile.csv')

predictions = prediction_dataframe(test_dataframe, dataframe,\
                                   categorical_features, \
                                   numeric_features_list_with_weight,\
                                   k, delta)
sum_square_complex = compute_sum_square(test_dataframe['hourly_entries'],
predictions)
r_square_complex = compute_r_squared(test_dataframe['hourly_entries'], pre
dictions)

print "SSE is "+ str(sum_square_complex)
print "R2 is "+ str(r_square_complex)
```

```
SSE is 19987489701.5
R2 is 0.704486292628
```

On this calculation, the SSE is 19062584953.9 and R2 is 0.712359529701.

Complexity: computational complexity is evaluated on predicting process. The time of predicting a record is O(n). n is total number of records.

Summary

Three models were built using some features to predict hourly entries in NYC subway. They are simple linear model, two-level regression model and non-parametric model. Their evaluation of ten time running are as following.

In [11]:

```
import summary

#uncomment following line to get 10 splitting results
#summary.run_for(10)

summary.load_result()
```

Out[11]:

	R2 linear	R2 nonparametric	R2 two level	SSE linear	SSE nonparametric	SSE leve
Split 1	0.449409	0.700920	0.817157	4.034806e+10	2.191702e+10	1.33
Split 2	0.460348	0.696894	0.820953	4.171109e+10	2.342783e+10	1.38
Split 3	0.461814	0.708022	0.816511	3.805069e+10	2.064336e+10	1.29
Split 4	0.476292	0.737757	0.830483	3.684264e+10	1.844867e+10	1.19
Split 5	0.475035	0.693565	0.803224	3.509954e+10	2.048846e+10	1.31
Split 6	0.478278	0.705190	0.808760	3.723477e+10	2.104030e+10	1.36
Split 7	0.476797	0.713586	0.820871	3.654771e+10	2.000709e+10	1.25
Split 8	0.461099	0.726785	0.826439	3.802906e+10	1.928017e+10	1.22
Split 9	0.471692	0.727459	0.835426	3.532065e+10	1.822107e+10	1.10
Split 10	0.476226	0.701547	0.797347	3.865352e+10	2.202527e+10	1.49

As we can see from the result, two-level regression model performs better than other two models. And computational complexity of nonparametric model is greater than other two models.

Possible further improvements of models

1, The features and parameters of all three model can be tuned to achieve better performance.

2, The two-level regression model only consider time as level one features. I think the performance will be better if the locations are considered in the model not only as a level indicator but also as a numerical variable.

3, The output of nonparametric model should be calculated in a more sophiscated way instead of simple average. Such ways include fitting a curve or training a small linear model based on k records.

4, More non-linearity would bring better performance. Since the hourly_entries is a number related to some measure of physical world. There will be some restrictions on this value. And predicting the value may consider these resitrictions as well.