Applied Data Science - Foundations module - Final Project

by Peter Varshavsky

The data

In this project I analyze hourly bikeshare data from Capital Bikeshare in Washington, DC. The dataset includes hourly counts of bike use by subscribers and one-time users, as well as quantitative and categorical weather variables.

Variables

- season:
  1. spring
  2. summer
  3. fall
  4. winter
- holiday: boolean
- workingday: boolean
- 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: degrees C
- atemp: 'feels like', degrees C
- humidity: relative humidity
- windspeed: wind speed
- casual: number of non-registered user rentals initiated
- registered: number of registered user rentals initiated
- count_tot: number of total rentals

Training and test sets

The data has been split into the training set containing hourly data for 1st-19th of each months, and test set containing data for the remainder of each month.

Results

I modeled hourly variation and daily variation separately assuming they were multiplicatively related. I computed hourly averages separately for working days and non working days, as there is a clear difference in use patterns. Daily variation was modeled with a linear model using several subsets of variables, with the final model including *day_num*, *season*, *temp*, *windspeed weather*.

The final model was submitted to Kaggle and received a Root Mean Squared Logarithmic Error score of 0.60530, ranking 1000th out of roughly 1500 participants. Because the test values used for scoring are not provided, it is not possible to visualize the residuals.

998	↓61	Chris H	0.60472	3	Mon, 13 Oct 2014 23:25:44
999	↓61	asda	0.60505	4	Wed, 26 Nov 2014 23:01:48 (-0.3h)
1000	new	**Peter Varshavsky**	**0.60530**	**1**	**Tue, 09 Dec 2014 21:48:31**
1001	↓62	Lotte	0.60536	3	Tue, 07 Oct 2014 14:30:19 (-11d)
1002	↓62	Simon Parker	0.60619	11	Thu, 09 Oct 2014 10:54:09

Processing and modeling

I chose to fill the missing days of every month with the average of five days before and five days after each missing period. This allowed me to work with an uninterrupted signal which made it easier to attempt harmonic analysis with FFT. I briefly attempted using FFT in Python and ARIMA in R, but, due to lack of experience, decided to fit the periodic components using least squares with available variables (*season*, *datetime*).

Filling in the missing data may also lead to unreliable P-values and R^2, since the model thinks that it has more degrees of freedom than there truly are.

Ideas for improvement of this approach

The model, as it is implemented has a number of problems. My next step would be refitting the model without interpolating the missing data, and inspecting the strangely outlying predicted counts. For example the counts predicted for January 21, 2011 are negative. For the submission I substituted all negative counts with zeros.

The weather is only used to fit daily use in this model, so the model is losing a lot of detail because hourly changes in temperature, wind, and precipitation are likely to have a lot of explanatory power.

Other approaches

After fixing the obvious problems with the current implementation, I would want to learn how to use the customary time series tools like ARMA and ARIMA models.

In [1]:

```
import patsy
import os
import sys
import csv
```

```
import timeit
from datetime import datetime
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import calendar

%matplotlib inline
plt.style.use('ggplot')
```

In [2]:

```
def readData_noParse(fileIn):
    train = pd.read_csv(fileIn, index_col = ['datetime'])
    train.index = pd.DatetimeIndex(train.index)
    return train

fileIn = "../data/train.csv"

train = readData_noParse(fileIn)
columns = list(train.columns)
columns[-1] = 'count_tot'
train.columns = columns

### Season recording is better after resample
# seasonsDict = {1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'}
# train['season'] = train.season.apply(lambda s: seasonsDict[s])
train.head(3)
```

Out[2]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	ca
2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0	3
2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0	8
2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0	5

Resample daily

In [3]:

```
train_daily = pd.DataFrame(train.resample('D', how = 'mean'))
print "Days without data (NaN):", sum(train_daily.count_tot.isnull())
```

```
Days without data (NaN): 263
```

Missing days:

Fill in missing days (20th - EOM of each month) with an average of 5 days before and 5 days after.

In [4]:
```
#print pd.concat([pd.DataFrame(train_daily.loc['2011-01-10':'2011-02-10'].mean()).transpose()]* 11, ignore_index = True)
def fill_NA_days(df, sampleStart, sampleEnd, startNA, endNA):
#     average_list = list(train_daily.loc['2011-01-15':'2011-02-5'].mean())
#     average_list_of_lists = [average_list for x in range(12)]
#     train_daily.loc['2011-01-20':'2011-01-31'] = average_list_of_lists
    ndays = int(endNA[8:]) - 19
    average_list = list(df.loc[sampleStart:sampleEnd].mean())
    average_list_of_lists = [average_list for x in range(ndays)]
    df.loc[startNA:endNA] = average_list_of_lists
```

In [5]:
```
months = [str(month).zfill(2) for month in range(1,13)] * 2
years = ['2011'] * 12 + ['2012'] * 12
sampleStarts = [years[i] + '-' + months[i] + '-' + '15' for i in range(len(months))]
sampleEnds = [years[i] + '-' + months[i] + '-' + '05' for i in range(1, len(months))]
startNAs = [years[i] + '-' + months[i] + '-' + '20' for i in range(len(months))]
monthEnds = [str(calendar.monthrange(2011, month)[1]).zfill(2) for month in range(1, 13)] + [str(calendar.monthrange(2012, month)[1]).zfill(2) for month in range(1, 13)]
endNAs = [years[i] + '-' + months[i] + '-' + monthEnds[i] for i in range(len(months))]

for i in range(len(sampleStarts) - 1):
    fill_NA_days(train_daily, sampleStarts[i], sampleEnds[i], startNAs[i], endNAs[i])

print "Number of missing values after filling in: %d" %sum(train_daily.count_tot.isnull())
```

```
Number of missing values after filling in: 0
```

In [6]:
```
### Note: can be rewritten simpler with Timestamp.month and wrapped into a function
###       to be used later on test set as well
### Warnings can be ignored
train_daily.season['2011-01':'2011-02'] = 'winter'
train_daily.season['2011-12':'2012-02'] = 'winter'
train_daily.season['2011-03':'2011-05'] = 'spring'
train_daily.season['2012-03':'2012-05'] = 'spring'
train_daily.season['2011-06':'2011-08'] = 'summer'
train_daily.season['2012-06':'2012-08'] = 'summer'
train_daily.season['2011-09':'2011-11'] = 'fall'
train_daily.season['2012-09':'2012-11'] = 'fall'
train_daily.season['2012-12'] = 'winter'
```

```
-c:5: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
```

Linear model with 1 variable *day_num*

This model explains a fair amount of daily variation with $R^2 = 0.563$, but it's obvious from the plots of the model and the residuals, that it can be improved with a seasonal component.

In [7]:

```
train_daily['day_num'] = list(range(1, train_daily.shape[0] + 1))
print train_daily.head(3)
print train_daily.columns

y, X = patsy.dmatrices('count_tot ~ day_num', data = train_daily)
mod1 = sm.OLS(y, X).fit()
```

```
            season   holiday  workingday   weather        temp       atem
```

```
p  \
2011-01-01  winter       0          0  1.583333  14.110833  18.18125
0
2011-01-02  winter       0          0  1.956522  14.902609  17.68695
7
2011-01-03  winter       0          1  1.000000   8.050909   9.47022
7

             humidity  windspeed    casual  registered  count_tot  da
y_num
2011-01-01  80.583333  10.749871  13.791667   27.250000  41.041667
    1
2011-01-02  69.608696  16.652122   5.695652   29.130435  34.826087
    2
2011-01-03  43.727273  16.636709   5.454545   55.863636  61.318182
    3
Index([u'season', u'holiday', u'workingday', u'weather', u'temp', u'at
emp', u'humidity', u'windspeed', u'casual', u'registered', u'count_tot
', u'day_num'], dtype='object')
```

In [8]:
```
print mod1.summary()
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:              count_tot   R-squared:
   0.563
Model:                            OLS   Adj. R-squared:
   0.562
Method:                 Least Squares   F-statistic:
   923.3
Date:                Tue, 09 Dec 2014   Prob (F-statistic):          5
.64e-131
Time:                        19:18:26   Log-Likelihood:
 -3818.3
No. Observations:                 719   AIC:
   7641.
Df Residuals:                     717   BIC:
   7650.
Df Model:                           1

======================================================================
========
                 coef    std err          t      P>|t|      [95.0% Con
f. Int.]
----------------------------------------------------------------------
--------
Intercept     96.3837      3.663     26.313      0.000        89.192
 103.575
day_num        0.2679      0.009     30.386      0.000         0.251
   0.285
======================================================================
========
Omnibus:                       37.563   Durbin-Watson:
```

```
   0.446
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
  22.660
Skew:                          -0.291   Prob(JB):
1.20e-05
Kurtosis:                       2.354   Cond. No.
    832.
==============================================================================
========
```

In [9]:

```
fig1, [ax1_1, ax1_2] = plt.subplots(nrows = 2, figsize = (20, 8))
ax1_1.plot(train_daily.count_tot)
ax1_1.plot(mod1.predict())
ax1_2.plot(train_daily.count_tot - mod1.predict(), '.')
ax1_1.set_ylabel("Count")
ax1_2.set_ylabel("Residuals")
ax1_2.set_xlabel("Days")
ax1_1.set_xlim([0, 719])
ax1_2.set_xlim([0, 719])
fig1.suptitle("Figure 1. Model 1: linear function of time", fontsize =
 14)
plt.show()
```

Model 2. Two variables: *day_num*, *season*

Adding season as a categorical variable improves R^2 to 0.784. As may be expected, there is more bike use in spring and summer, and less in winter than during fall (baseline).

In [10]:

```
y, X = patsy.dmatrices('count_tot ~ day_num + season', data = train_da
ily)
mod2 = sm.OLS(y, X).fit()
print mod2.summary()
```

```
                            OLS Regression Results

==============================================================================
========
```

```
Dep. Variable:                  count_tot   R-squared:
   0.784
Model:                                OLS   Adj. R-squared:
   0.783
Method:                     Least Squares   F-statistic:
   649.2
Date:                    Tue, 09 Dec 2014   Prob (F-statistic):          4
.06e-236
Time:                            19:18:26   Log-Likelihood:
 -3564.3
No. Observations:                     719   AIC:
   7139.
Df Residuals:                         714   BIC:
   7162.
Df Model:                               4

======================================================================
==============
                      coef    std err          t      P>|t|      [95.
0% Conf. Int.]
----------------------------------------------------------------------
--------------
Intercept          94.8055      4.051     23.406      0.000         86
.853   102.758
season[T.spring]   24.7885      3.811      6.505      0.000         17
.307    32.270
season[T.summer]   45.0108      3.661     12.294      0.000         37
.823    52.199
season[T.winter]  -49.8802      3.872    -12.881      0.000        -57
.483   -42.278
day_num             0.2552      0.007     38.368      0.000          0
.242     0.268
======================================================================
========
Omnibus:                       35.300   Durbin-Watson:
   0.928
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
  44.729
Skew:                          -0.464   Prob(JB):
1.94e-10
Kurtosis:                       3.794   Cond. No.
2.07e+03
======================================================================
========

Warnings:
[1] The condition number is large, 2.07e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In [11]:

```
fig2, [ax2_1, ax2_2] = plt.subplots(num = 2, nrows = 2, figsize = (20,
 8))
ax2_1.plot(train_daily.count_tot)
ax2_1.plot(mod2.predict())
```

```
ax2_2.plot(train_daily.count_tot - mod2.predict(), '.')
ax2_1.set_ylabel("Count")
ax2_2.set_ylabel("Residuals")
ax2_2.set_xlabel("Days")
ax2_1.set_xlim([0, 719])
ax2_2.set_xlim([0, 719])
fig2.suptitle("Figure 2. Model 2: linear function of time", fontsize =
 14)
plt.show()
```

Model 3. Adding quantitative weather variables:

- Variables in model: *day_num*, *season*, *temp*, *humidity*, *windspeed*

atemp is air temperature or 'feels like', which is highly correlated to temp, humidity, and windspeed. It is left out.

In [12]:
```
y, X = patsy.dmatrices('count_tot ~ day_num + season + temp + humidity
 + windspeed', data = train_daily)
mod3 = sm.OLS(y, X).fit()
print mod3.summary()
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:              count_tot   R-squared:
    0.874
Model:                            OLS   Adj. R-squared:
    0.873
Method:                 Least Squares   F-statistic:
    705.5
Date:                Tue, 09 Dec 2014   Prob (F-statistic):                5
.28e-315
Time:                        19:18:27   Log-Likelihood:
 -3370.7
No. Observations:                 719   AIC:
```

```
   6757.
Df Residuals:                     711   BIC:
   6794.
Df Model:                           7

========================================================================
==============
                         coef    std err          t      P>|t|      [95.
0% Conf. Int.]
------------------------------------------------------------------------
--------------
Intercept            120.7840      9.299     12.989      0.000       102
.528   139.040
season[T.spring]      16.1397      3.003      5.375      0.000        10
.245    22.035
season[T.summer]     -15.9060      4.106     -3.874      0.000       -23
.967    -7.845
season[T.winter]     -18.2836      3.587     -5.097      0.000       -25
.326   -11.241
day_num                0.2258      0.005     42.907      0.000         0
.215     0.236
temp                   5.2826      0.287     18.379      0.000         4
.718     5.847
humidity              -1.2782      0.094    -13.645      0.000        -1
.462    -1.094
windspeed             -2.6212      0.252    -10.414      0.000        -3
.115    -2.127
========================================================================
========
Omnibus:                      141.592   Durbin-Watson:
   1.181
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
 668.910
Skew:                          -0.808   Prob(JB):                      5
.60e-146
Kurtosis:                       7.440   Cond. No.
4.06e+03
========================================================================
========

Warnings:
[1] The condition number is large, 4.06e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In [13]:

```
fig3, [ax3_1, ax3_2] = plt.subplots(num = 3, nrows = 2, figsize = (20,
 8))
ax3_1.plot(train_daily.count_tot)
ax3_1.plot(mod3.predict())
ax3_2.plot(train_daily.count_tot - mod3.predict(), '.')
ax3_1.set_ylabel("Count")
ax3_2.set_ylabel("Residuals")
ax3_2.set_xlabel("Days")
ax3_1.set_xlim([0, 719])
ax3_2.set_xlim([0, 719])
```

```
fig3.suptitle("Figure 3. Model 3: linear function of time", fontsize =
 16)
plt.show()
```

Figure 3. Model 3: linear function of time

Model 4. Adding qualitative weather variable:

- Variables in model: *day_num*, *season*, *temp*, *windspeed*, *weather*
- Variables taken out: *humidity*

With addition of *weather*, *humidity* lost significance.

TODO: Using ANOVA, test whether leaving out remaining variables: *humidity*, *workingday*, *holiday* improves the model.

In [14]:

```
y, X = patsy.dmatrices('count_tot ~ day_num + season + temp + weather
+ windspeed', data = train_daily)
mod4 = sm.OLS(y, X).fit()
print mod4.summary()
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:              count_tot   R-squared:
   0.891
Model:                            OLS   Adj. R-squared:
   0.890
Method:                 Least Squares   F-statistic:
   826.9
Date:                Tue, 09 Dec 2014   Prob (F-statistic):
    0.00
Time:                        19:18:27   Log-Likelihood:
 -3320.3
No. Observations:                 719   AIC:
   6657.
Df Residuals:                     711   BIC:
   6693.
```

```
Df Model:                           7

==============================================================================
==============
                         coef    std err          t      P>|t|      [95.
0% Conf. Int.]
------------------------------------------------------------------------------
--------------
Intercept          106.6162      7.591     14.045      0.000          91
.713   121.520
season[T.spring]    21.5969      2.761      7.822      0.000          16
.176    27.018
season[T.summer]    -8.4998      3.689     -2.304      0.022         -15
.743    -1.256
season[T.winter]   -16.1235      3.341     -4.826      0.000         -22
.683    -9.564
day_num              0.2315      0.005     47.492      0.000           0
.222     0.241
temp                 4.5303      0.261     17.368      0.000           4
.018     5.042
weather            -47.9487      2.675    -17.922      0.000         -53
.201   -42.696
windspeed           -1.6305      0.224     -7.275      0.000          -2
.071    -1.190
==============================================================================
========
Omnibus:                       37.925   Durbin-Watson:
   1.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
 107.341
Skew:                          -0.185   Prob(JB):
4.91e-24
Kurtosis:                       4.856   Cond. No.
3.63e+03
==============================================================================
========

Warnings:
[1] The condition number is large, 3.63e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In [15]:

```python
fig4, [ax4_1, ax4_2] = plt.subplots(num = 4, nrows = 2, figsize = (20,
 8))
ax4_1.plot(train_daily.count_tot)
ax4_1.plot(mod4.predict())
ax4_2.plot(train_daily.count_tot - mod4.predict(), '.')
ax4_1.set_ylabel("Count")
ax4_2.set_ylabel("Residuals")
ax4_2.set_xlabel("Days")
ax4_1.set_xlim([0, 719])
ax4_2.set_xlim([0, 719])
fig4.suptitle("Figure 4. Model 4: linear function of time", fontsize =
 16)
plt.show()
```

Figure 4. Model 4: linear function of time

Using *mod4* on test set

Read test set

In [16]:

```
fileIn = "../data/test.csv"

test = readData_noParse(fileIn)
test.index = pd.DatetimeIndex(test.index)
print "Test shape: ", test.shape
print test.head(3)

### Resample daily
test_daily = test.resample('D', how = 'mean')
print "test_daily shape (after resampling by date):", test_daily.shape
test_daily['day_num'] = (test_daily.index - pd.Timestamp('2011-01-01')
).days + 1
### Season names
test_daily.season['2011-01':'2011-02'] = 'winter'
test_daily.season['2011-12':'2012-02'] = 'winter'
test_daily.season['2011-03':'2011-05'] = 'spring'
test_daily.season['2012-03':'2012-05'] = 'spring'
test_daily.season['2011-06':'2011-08'] = 'summer'
test_daily.season['2012-06':'2012-08'] = 'summer'
test_daily.season['2011-09':'2011-11'] = 'fall'
test_daily.season['2012-09':'2012-11'] = 'fall'
test_daily.season['2012-12'] = 'winter'
### Drop Nan
test_daily.dropna(inplace = True)
print "test_daily shape after dropping NaNs: ", test_daily.shape
```

```
-c:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
-c:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy


Test shape:  (6493, 8)
                     season  holiday  workingday  weather   temp   ate
mp  \
2011-01-20 00:00:00       1        0           1        1  10.66  11.3
65
2011-01-20 01:00:00       1        0           1        1  10.66  13.6
35
2011-01-20 02:00:00       1        0           1        1  10.66  13.6
35

                     humidity  windspeed
2011-01-20 00:00:00        56    26.0027
2011-01-20 01:00:00        56     0.0000
2011-01-20 02:00:00        56     0.0000
test_daily shape (after resampling by date): (712, 8)
test_daily shape after dropping NaNs:  (275, 9)
```

Predicting test set daily totals

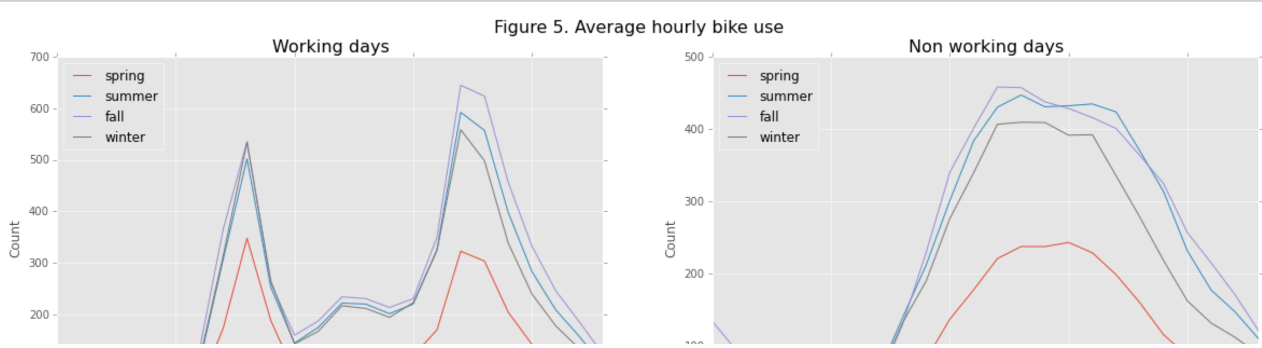
In [17]:
```
_, X = patsy.dmatrices('day_num ~ day_num + season + temp + weather + windspeed', data = test_daily)
test_daily['mod4'] = (mod4.predict(X) * 24).astype(int)
```

Within-day variation

TODO:

- Currently averages are divided by 24. This doesn't make sense

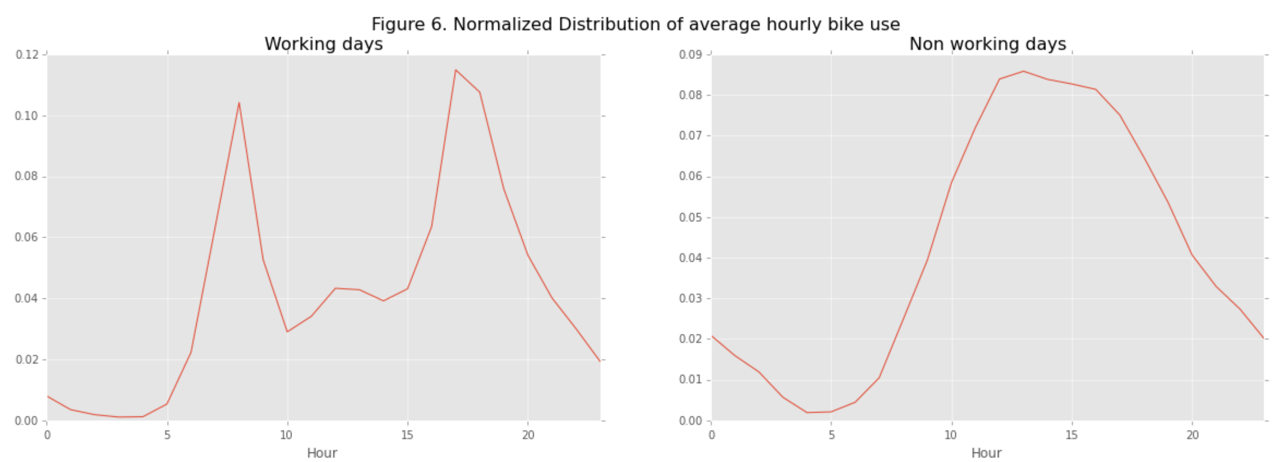
In [28]:
```
seasonsDict = {1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'}

hourlyBySW = train.groupby(['workingday', 'season', lambda x: x.hour]).count_tot.mean()

hourlyByW = train.groupby(['workingday', lambda x: x.hour]).mean()

fig5, axs5 = plt.subplots(ncols = 2, num = 5, figsize = (20,6))

for season in [1,2,3,4]:
    ### workingday == 1
    axs5[0].plot(hourlyBySW.loc[1,season,:], label = seasonsDict[season])
    ### workingday == 0
    axs5[1].plot(hourlyBySW.loc[0,season,:], label = seasonsDict[season])

for ax in axs5:
    ax.legend(loc = 2)
    ax.set_xlabel('Hour')
    ax.set_xlim((0,23))
    ax.set_ylabel('Count')

axs5[0].set_title('Working days', fontsize = 16)
axs5[1].set_title('Non working days', fontsize = 16)
fig5.suptitle("Figure 5. Average hourly bike use", fontsize = 16)


plt.show()
```



Making hourly predictions

Computing average fractions of daily use for each hour

In [29]:
```
### computing average hourly use
averageDailySum = [sum(hourlyByW.loc[workday,:].count_tot) for workday
 in [0,1]]
hourlyFractions = [hourlyByW.loc[workday,:].count_tot / averageDailySu
m[workday] for workday in [0, 1]]
hourlyFractionsDict = {workday:{i:fraction for i, fraction in hourlyFr
actions[workday].iteritems()} for workday in [0,1]}
### plotting average hourly use
fig6, axs6 = plt.subplots(num = 6, ncols = 2, figsize = (20,6))

axs6[0].plot(hourlyFractions[1])
axs6[1].plot(hourlyFractions[0])

for ax in axs6:
    ax.set_xlim((0,23))

axs6[0].set_title('Working days', fontsize = 16)
axs6[1].set_title('Non working days', fontsize = 16)

for ax in axs6:
    ax.set_xlabel('Hour')
    ax.set_xlim((0,23))

fig6.suptitle("Figure 6. Normalized Distribution of average hourly bik
e use", fontsize = 16)

plt.show()
```



Resampling back to hourly resolution

- Resample mod4 predictions by hour and fill NA with daily counts

- Note: mod4_hourly picks up all dates, not just the 20th-EOM of each month these dates will be eliminated in joining with original test file

In [30]:
```
mod4_hourly = test_daily[['mod4']].resample('H', how = 'sum').fillna(m
ethod = 'ffill')
### Join mod4_hourly with raw test data
#print "test rows: %d\nmod4 rows: %d" %(test.shape[0], mod4_hourly.sha
pe[0])
test_prediction = pd.merge(test, mod4_hourly, how = 'inner', left_inde
x = True, right_index = True)
#print "Merged shape: %d" %test_prediction.shape[0]
### Hour column
test_prediction['hour'] = test_prediction.index.hour
test_prediction['workdayPred'] = test_prediction.hour.replace(hourlyFr
actionsDict[1], inplace=False) * test_prediction.workingday * test_pre
diction.mod4
test_prediction['nonworkdayPred'] = test_prediction.hour.replace(hourl
yFractionsDict[0], inplace=False) * (1 - test_prediction.workingday) *
 test_prediction.mod4
test_prediction['hourlyPred'] = test_prediction.workdayPred + test_pre
diction.nonworkdayPred
test_prediction.head(3)
```

Out[30]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	m
2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027	11
2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000	11
2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000	11

Cleanup of submission

- Setting negative values to 0 (need to investigate negative values)
- Loading sample submissions into a data frame and joining with predictions

In [31]:
```
### Final submission series
final_predictions = pd.DataFrame(test_prediction.hourlyPred.astype(int
))
final_predictions.columns = ['count']
final_predictions[final_predictions < 0] = 0

### Load sample file
fname = "sampleSubmission.csv"
fIn = os.path.join("..", "data", fname)
sample = pd.read_csv(fIn, parse_dates = [0])
sample.index = sample.datetime
```

```
sample.columns
sample.drop('datetime', 1, inplace = True)
final_out = pd.merge(final_predictions, sample, how = 'right', left_in
dex = True, right_index = True)
final_out.drop(u'count_y', 1, inplace = True)
final_out = pd.Series(final_out.count_x)
final_out.name = 'count'
final_out[final_out.isnull()] = 0
```

Write hourly predictions to file

In [32]:

```
outputFolder = "output"
time = datetime.strftime(datetime.now(), "%Y-%m-%d_%H-%M-%S")
outputFileName = "prediction_file_" + time + ".csv"
fOut = os.path.join("..", outputFolder, outputFileName)
final_out.to_csv(fOut, header = True, index_label = "datetime")
```

Quick fixes to February 26, 2011 (negative values) and last week of Dec, 2012 (not in my predictions due to a coding error)

To do.

In [33]:

```
twoPrior = np.array(final_out['2011-1-24':'2011-1-25'].resample('H', h
ow = 'mean'))
twoAfter = np.array(final_out['2011-1-27':'2011-1-28'].resample('H', h
ow = 'mean'))

# print 'a', twoPrior.shape
# print 'a', twoAfter.shape

quickFix26 = np.zeros(24)
# for date in ['24', '25', '26' ,'27', '28']:
#     #quickFix26 += np.array(final_out['2011-01-'+date])
#     print final_out['2011-1-' + date].shape
final_out['2011-1-26']
```

Out[33]:

```
datetime
2011-01-26 00:00:00    0
2011-01-26 01:00:00    0
2011-01-26 02:00:00    0
2011-01-26 05:00:00    0
2011-01-26 06:00:00    0
2011-01-26 07:00:00    0
2011-01-26 08:00:00    0
2011-01-26 09:00:00    0
2011-01-26 10:00:00    0
2011-01-26 11:00:00    0
2011-01-26 12:00:00    0
2011-01-26 13:00:00    0
2011-01-26 14:00:00    0
2011-01-26 15:00:00    0
2011-01-26 16:00:00    0
2011-01-26 17:00:00    0
```

```
Name: count, dtype: float64
```

In [ ]: