CS156 Week#7 Assignment - RBF & Yosemite Weather

Tomer Eldor

Minerva Schools

- Credit: Thanking Yoel Ferdman who helped me work on this assignment, as I'm not feeling well and needed help.

Yosemite Village Yearly Weather

Temperature is cyclical, not only on a 24 hour basis but also on a yearly basis. Convert the dataset into a richer format whereby the day of the year is also captured. For example the time “20150212 1605”, can be converted into (43, 965) because the 12th of February is the 43rd day of the year, and 16:05 is the 965th minute of the day.

This data covers 6 years, so split the data into a training set of the first 5 years, and a testing set of the 6th year.

Using the temperature dataset from your pre-class work (https://course-resources.minerva.kgi.edu/uploaded_files/mke/rj3Edn/yosemite-temperatures.zip (https://course-resources.minerva.kgi.edu/uploaded_files/mke/rj3Edn/yosemite-temperatures.zip))

In [2]:
```
# Setup: Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.model_selection import train_test_split
from mpl_toolkits.mplot3d import Axes3D

%matplotlib inline
```

```
/Users/tomereldor/anaconda/lib/python2.7/site-packages/matplotlib/font_
manager.py:273: UserWarning: Matplotlib is building the font cache usin
g fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. T
his may take a moment.')
```

Fetching the Data

In [3]:
```
# creating a list of filenames
years = range(2011, 2017) #specifying range of years as variable
files = ['CRNS0101-05-%d-CA_Yosemite_Village_12_W.txt' % y for y in year
s] #creating a list of file_names with the appropriate years

# loading the data from text. we are only interested in datetime,
# which is divided into date in col 1 & hour/min in col 2,
# and temperature in col 8. So we only grab columns 1,2,8.
data = [np.loadtxt(f, usecols=[1, 2, 8, 9]) for f in files]
data = np.vstack(data)

## Converting into Pandas DF
df = pd.DataFrame(data)
#Renaming columns
df.rename(columns={0:'Date',1:'HHmm',2:'Temp', 3:'Rain'}, inplace=True)
df.head() #debug
```

Out[3]:

	Date	HHmm	Temp	Rain
0	20110101.0	5.0	-6.4	0.0
1	20110101.0	10.0	-6.5	0.0
2	20110101.0	15.0	-6.5	0.0
3	20110101.0	20.0	-6.5	0.0
4	20110101.0	25.0	-6.7	0.0

In [4]:
```
# len(df.Rain[df.Rain> -9999])
#len(df.Rain[df.Rain < -9998])
```

In [5]:
```
## Data Sanity verification

## making sure data looks right, I checke for means, min , max, and valu
es of data.
# print min(df.Temp),  max(df.Temp)
# print min(df.Rain),  max(df.Rain)
# len(df.Rain[df.Rain> -9999])
# len(df.Rain[df.Rain < -9998])

# There are some rows with corrupted values:
# temperature of -9999 degress, or Rain of
# these aren't informative and corrupted so I drop them.

len(df.Rain[df.Temp < -9998])
df = df[df.Temp > -9998.0] #removes bad data

len(df.Rain[df.Rain < -9998]) # 1862 rows only,  leaving 629434 rows
df = df[df.Rain > -9998.0] #eliminating those

#df.rename(columns={0:'Date',1:'HHmm',2:'Temp'}, inplace=True)
df_copy = df.copy()
df_copy.tail()
```

Out[5]:

	Date	HHmm	Temp	Rain
631291	20161231.0	2340.0	0.3	0.0
631292	20161231.0	2345.0	0.2	0.0
631293	20161231.0	2350.0	0.0	0.0
631294	20161231.0	2355.0	-0.1	0.0
631295	20170101.0	0.0	-0.1	0.0

Enriching to day and minute numerical format.

Temperature is cyclical, not only on a 24 hour basis but also on a yearly basis. Convert the dataset into a richer format whereby the day of the year is also captured. For example the time 20150212 1605, can be converted into (43, 965) because the 12th of February is the 43rd day of the year, and 16:05 is the 965th minute of the day.

In [6]:
```
'''Create converted Minutes column: from HHmm to continuous integer coun
t of minutes from start of day.
This will convert to an int count of minutes from 0 to 1440 (num of minu
tes per day) in increments of 5 minutes.
Initially, keeping the HHmm column and Min column and compare for debugg
ing'''

df['Min'] = np.floor_divide(df['HHmm'], 100) * 60 + np.mod(df['HHmm'], 1
00)
df.tail() #debug
```

Out[6]:

	Date	HHmm	Temp	Rain	Min
631291	20161231.0	2340.0	0.3	0.0	1420.0
631292	20161231.0	2345.0	0.2	0.0	1425.0
631293	20161231.0	2350.0	0.0	0.0	1430.0
631294	20161231.0	2355.0	-0.1	0.0	1435.0
631295	20170101.0	0.0	-0.1	0.0	0.0

In [7]:
```
# converting dates to actual DateTime format from string
df['Date'] = pd.to_datetime(df['Date'], format = ('%Y%m%d'))
# from the datetime object, we can grab its components (year/month/da
y...)
df['Yr'] = df['Date'].dt.year # grabbing year
# Converting 'day' from date in date-time format to continuous integer c
ount of days passed since Jan 1 that year
df['Day'] = df['Date'].dt.strftime('%j').astype(int)

# reordering columns
df = df[['Date','Yr','Day','Min','HHmm','Temp','Rain']]
df.tail()
```

Out[7]:

	Date	Yr	Day	Min	HHmm	Temp	Rain
631291	2016-12-31	2016	366	1420.0	2340.0	0.3	0.0
631292	2016-12-31	2016	366	1425.0	2345.0	0.2	0.0
631293	2016-12-31	2016	366	1430.0	2350.0	0.0	0.0
631294	2016-12-31	2016	366	1435.0	2355.0	-0.1	0.0
631295	2017-01-01	2017	1	0.0	0.0	-0.1	0.0

This data covers 6 years, so let's split the data into a training set of the first 5 years, and a testing set of the 6th year. The data covers from 2011 to 2016 and some of 2017. I'll split it to test from 2011 to 2015, then test from 2016 onwards. I'll be working with numpy onwards, so I'll convert needed columns into numpy arrays

In [8]:
```
# train set: before 2016
train_data = df[df['Yr'] < 2016]
# test set: 2016 or after
test_data = df[df['Yr'] >= 2016]

# Transformation from Pandas DF to Numpy Arrays
train_data = train_data.values
test_data = test_data.values

train_data
```

Out[8]:
```
array([[Timestamp('2011-01-01 00:00:00'), 2011, 1, ..., 5.0, -6.4, 0.
0],
       [Timestamp('2011-01-01 00:00:00'), 2011, 1, ..., 10.0, -6.5, 0.
0],
       [Timestamp('2011-01-01 00:00:00'), 2011, 1, ..., 15.0, -6.5, 0.
0],
       ...,
       [Timestamp('2015-12-31 00:00:00'), 2015, 365, ..., 2345.0, -1.6,
0.0],
       [Timestamp('2015-12-31 00:00:00'), 2015, 365, ..., 2350.0, -1.4,
0.0],
       [Timestamp('2015-12-31 00:00:00'), 2015, 365, ..., 2355.0, -1.6,
0.0]], dtype=object)
```

In [9]:
```
# TRAINING and TEST Sets Split
# Since in Numpy I can't access columns by names, but I might change the
 DB structure whilt working on it,
# It's safer and simpler for me to make a dictionary of column names to
 their position and call using it
col_dict = {'date':0, 'year':1, 'yr':1, 'day':2, 'Day':2, 'min':3,
'Min':3, 'HHmm':4, 'temp':5}

# train sets
mins_train = np.array(train_data[:, col_dict.get('min') ])
days_train = np.array(train_data[:, col_dict.get('day')])
temp_y_train = np.array(train_data[:, col_dict.get('temp') ])

# test sets
mins_test = np.array(test_data[:, col_dict.get('min')])
days_test = np.array(test_data[:, col_dict.get('day')])
temp_y_test = np.array(test_data[:, col_dict.get('temp')])

# debug
# print test_minutes[:10], test_minutes[-10:]
print temp_y_test[:10], temp_y_test[-10:] #debug
# print test_days[:10], test_days[-10:] #debug
```

```
[-1.6 -1.9 -2.0 -2.1 -2.2 -2.4 -2.7 -2.8 -3.0 -3.1] [1.3 0.9 0.7 0.4 0.
3 0.3 0.2 0.0 -0.1 -0.1]
```

Predictions

1. Cover each input dimension with a list of radial basis functions. This turns the pair of inputs into a much richer representation, mapping (d,t) into ($\Phi_1(d)$, $\Phi_2(t)$). Experiment with different numbers of radial basis functions and different widths of the radial basis function in different dimensions.
2. Using this new representation, build a linear parameter model that captures both seasonal variations and daily variations.
3. Create two plots, one showing the time-of-day contribution, and one showing the time-of-year contribution.
4. Make a 3D plot showing temperature as a function of (day, time). Make sure to label your axes!
5. Using mean squared error, quantify how your model performs on the testing data if you:
    A. Train with just the daily component of the model
    B. Train with just the yearly component of the model
    C. Train with the full model.

RBF

In [11]:
```
# Setup, Import RBF
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.metrics import  r2_score, mean_squared_error

# We need to reshape the used np arrays (temp, min, day) to a column ins
tead of a row
temp_y_train, temp_y_test = temp_y_train.reshape(-1,1), temp_y_test.resh
ape(-1,1)
all_minutes = df['Min'].values.reshape(-1,1)
all_days = df['Day'].values.reshape(-1,1)

# Function to initialize Centers list: dividing the range of chosen data
 into equal distances to start as RBF centers
def initialize_centers_list(minimum,maximum,n_rbf):
    centers = []
    # divide the range into equal distances to start as RBF centers
    centers = [ i for i in range(minimum,maximum,maximum/(n_rbf-1)) ]
    # reshape from row to column
    centers = np.asarray((centers)).reshape(-1,1)
    return centers



def test_model_ridge(x_train,y_train,x_test,y_test,
alpha_ridge=0.0001):
    ''' Function that will fit and predict using a Ridge regression
        on the chosen data and report results of MSE and R^2 score (coli
nearity between y_true and y_predicted)'''
    # fitting the model and predicting using it
    regr = Ridge(alpha = alpha_ridge, fit_intercept=False) #choosing a c
lassifier
    regr.fit(x_train,y_train) # fitting a Ridge Regression
```

```
    y_pred = regr.predict(x_test) #predicting using the regression
    # Calculating MSE of the model's prediction vs true y
    mse = mean_squared_error(y_test, y_pred)
    # calculating r^2 of the model's prediction vs true y
    #*The same results is given by simply regr.score() for the ones I te
sted, but I'm going with R2 score to be more specific and certain.
    r2 = r2_score(y_test, y_pred)
    #print "results: ", mse, r2 debug
    return mse, r2

# print(test_model_ridge(X_train,temp_y_train)) #debug

def test_RBF(numbers_of_rbfs_to_test, widths_to_test):
    global df, all_minutes, all_days
    rbf_tests = []
    for n_rbf in numbers_of_rbfs_to_test:
        for sigma in widths_to_test:
            print("Testing for {} RBF centers, with Sigma ={}".format(n_
rbf, sigma))

            #initializing a list for the RBF centers for minutes
            center_min = initialize_centers_list(0,1399,n_rbf)
            #initializing a list for the RBF centers for days
            center_days = initialize_centers_list(0,364,n_rbf)

            ### Coverting each input dimension with a list of radial bas
is functions ###
            # The function rbf_kernel computes the radial basis function
 (RBF) kernel between two vectors X,Y: K(x, y) = exp(-gamma ||x-y||^2)
            # (from http://scikit-learn.org/stable/modules/generated/skl
earn.metrics.pairwise.rbf_kernel.html#sklearn.metrics.pairwise.rbf_kerne
l and the user guide)
            # Calculating RBF Kernels on the entier dataset, test+train
            minutes_rbf = rbf_kernel(all_minutes,center_min,gamma=1/sigm
a)
            days_rbf = rbf_kernel(all_days,center_days,gamma=1/sigma)

            ###  Split Data to Train and Test ###
            # find the row of first 2016 entry to split by
            split_id = np.where(np.logical_and(df.Date == '2016-01-01 0
0:00:00' , df.Min == 0.0))[0][0] # that's row 525479 apparently

            # Splitting to test and train sets : either minutes or days
            X_mins_train, X_mins_test = minutes_rbf[:split_id], minutes_
rbf[split_id:]
            X_days_train, X_days_test = days_rbf[:split_id], days_rbf[sp
lit_id:]
            # rebuilding X training/test set of both minutes and days to
gether, by concatenating arrays togetehr as 2 columns
            X_train = np.concatenate((minutes_rbf[:split_id],days_rbf[:s
plit_id]),axis=1)
            X_test = np.concatenate((minutes_rbf[split_id:],days_rbf[spl
it_id:]),axis=1)


            ### FITTING THE MODELS, TESTING AND REPORTING RESULTS ###
            # "Building a linear parameter model regression that capture
```

```
s both seasonal variations and daily variations"

            # Full Model (Minutes + Days): Fitting Regression, predictin
g and calculating performance MSE & R^2
            mse_full, r2_full = test_model_ridge(X_train,temp_y_train,X_
test,temp_y_test, alpha_ridge=0.0001)

            # Only Minutes Model: Fitting Regression, predicting and cal
culating performance MSE & R^2
            mse_mins, r2_mins = test_model_ridge(X_mins_train ,temp_y_tr
ain, X_mins_test, temp_y_test, alpha_ridge=0.0001)

            # Only Days Model: Fitting Regression, predicting and calcul
ating performance MSE & R^2
            mse_days, r2_days = test_model_ridge(X_days_train ,temp_y_tr
ain, X_days_test, temp_y_test, alpha_ridge=0.0001)

            # appending the results performance as a sublist into the li
st of performances
            rbf_tests.append([n_rbf, sigma, mse_full, r2_full, mse_mins,
 r2_mins, mse_days, r2_days])

    # transforming into DataFrame
    rbf_tests = pd.DataFrame(rbf_tests, columns = ["rbf
centers","sigma","Full Model: MSE","Full Model: R^2","Minutes Mode:
MSE", "Minutes Model: R^2","Days Model: MSE", "Days Model: R^2"])
    #print('{}\n'.format(rbf_tests)) #debug
    return rbf_tests, X_mins_train, X_mins_test, X_days_train, X_days_te
st


# Testing! insert lists of number of RBFs to test and of widths.
# This function currently uses Ridge Regression.
rbf_tests, X_mins_train, X_mins_test, X_days_train, X_days_test = test_R
BF(numbers_of_rbfs_to_test=[3,5,6,10,12,6*12], widths_to_test=
[0.05,0.1,0.2,0.4,0.5])
rbf_tests
```

```
Testing for 3 RBF centers, with Sigma =0.05
Testing for 3 RBF centers, with Sigma =0.1
Testing for 3 RBF centers, with Sigma =0.2
Testing for 3 RBF centers, with Sigma =0.4
Testing for 3 RBF centers, with Sigma =0.5
Testing for 5 RBF centers, with Sigma =0.05
Testing for 5 RBF centers, with Sigma =0.1
Testing for 5 RBF centers, with Sigma =0.2
Testing for 5 RBF centers, with Sigma =0.4
Testing for 5 RBF centers, with Sigma =0.5
Testing for 6 RBF centers, with Sigma =0.05
Testing for 6 RBF centers, with Sigma =0.1
Testing for 6 RBF centers, with Sigma =0.2
Testing for 6 RBF centers, with Sigma =0.4
Testing for 6 RBF centers, with Sigma =0.5
Testing for 10 RBF centers, with Sigma =0.05
Testing for 10 RBF centers, with Sigma =0.1
Testing for 10 RBF centers, with Sigma =0.2
Testing for 10 RBF centers, with Sigma =0.4
Testing for 10 RBF centers, with Sigma =0.5
Testing for 12 RBF centers, with Sigma =0.05
Testing for 12 RBF centers, with Sigma =0.1
Testing for 12 RBF centers, with Sigma =0.2
Testing for 12 RBF centers, with Sigma =0.4
Testing for 12 RBF centers, with Sigma =0.5
Testing for 72 RBF centers, with Sigma =0.05
Testing for 72 RBF centers, with Sigma =0.1
Testing for 72 RBF centers, with Sigma =0.2
Testing for 72 RBF centers, with Sigma =0.4
Testing for 72 RBF centers, with Sigma =0.5
```

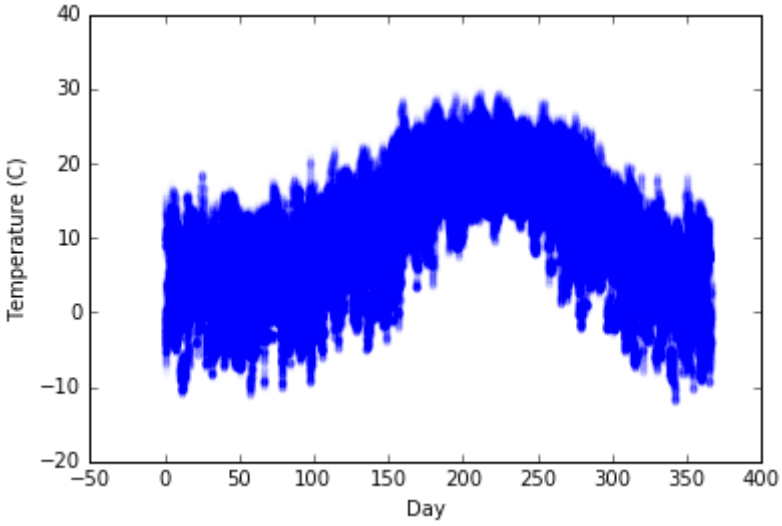
Out[11]:

	rbf centers	sigma	Full Model: MSE	Full Model: R^2	Minutes Mode: MSE	Minutes Model: R^2	Days Model: MSE	Days Model: R^2
0	3	0.05	161.737520	-1.667071	163.033295	-1.688439	162.215564	-1.674954
1	3	0.10	161.721867	-1.666813	163.015534	-1.688146	162.217405	-1.674984
2	3	0.20	161.554221	-1.664048	162.778537	-1.684238	162.283648	-1.676077
3	3	0.40	161.115916	-1.656821	162.741256	-1.683623	161.879562	-1.669413
4	3	0.50	160.404283	-1.645086	162.310194	-1.676515	161.591099	-1.664657
5	5	0.05	160.936788	-1.653867	163.033295	-1.688439	161.410298	-1.661675
6	5	0.10	160.882306	-1.652968	162.975918	-1.687492	161.411994	-1.661703
7	5	0.20	160.194363	-1.641624	162.228330	-1.675165	161.456579	-1.662438
8	5	0.40	159.499632	-1.630168	162.178411	-1.674341	160.802202	-1.651648
9	5	0.50	158.474423	-1.613262	161.595057	-1.664722	160.335095	-1.643945
10	6	0.05	161.323095	-1.660237	162.506484	-1.679751	162.320193	-1.676679
11	6	0.10	161.269908	-1.659360	162.450344	-1.678826	162.322044	-1.676710
12	6	0.20	160.607859	-1.648443	161.712470	-1.666658	162.389636	-1.677825
13	6	0.40	160.190293	-1.641557	161.676214	-1.666060	161.999151	-1.671385
14	6	0.50	159.506798	-1.630286	161.254548	-1.659107	161.718303	-1.666754
15	10	0.05	156.688165	-1.583807	159.764532	-1.634536	160.256393	-1.642647
16	10	0.10	156.689674	-1.583831	159.764532	-1.634536	160.257889	-1.642672
17	10	0.20	156.702966	-1.584051	159.764532	-1.634536	160.272953	-1.642920
18	10	0.40	155.764221	-1.568571	159.764532	-1.634536	159.279791	-1.626543
19	10	0.50	155.095766	-1.557548	159.764532	-1.634536	158.572892	-1.614886
20	12	0.05	157.600631	-1.598853	162.192420	-1.674572	158.852907	-1.619503
21	12	0.10	157.518527	-1.597499	162.103303	-1.673103	158.854178	-1.619524
22	12	0.20	156.401665	-1.579082	160.930230	-1.653759	158.836042	-1.619225
23	12	0.40	154.949795	-1.555141	160.795028	-1.651529	157.464480	-1.596608
24	12	0.50	152.577635	-1.516023	159.226768	-1.625668	156.493544	-1.580597
25	72	0.05	132.096238	-1.178283	158.037842	-1.606063	135.643966	-1.236785
26	72	0.10	131.611111	-1.170283	157.301324	-1.593918	135.641046	-1.236737
27	72	0.20	124.619975	-1.054998	147.599731	-1.433937	135.003181	-1.226219
28	72	0.40	117.085908	-0.930761	146.826665	-1.421189	126.476774	-1.085617
29	72	0.50	107.163223	-0.767134	137.857807	-1.273292	120.404136	-0.985479

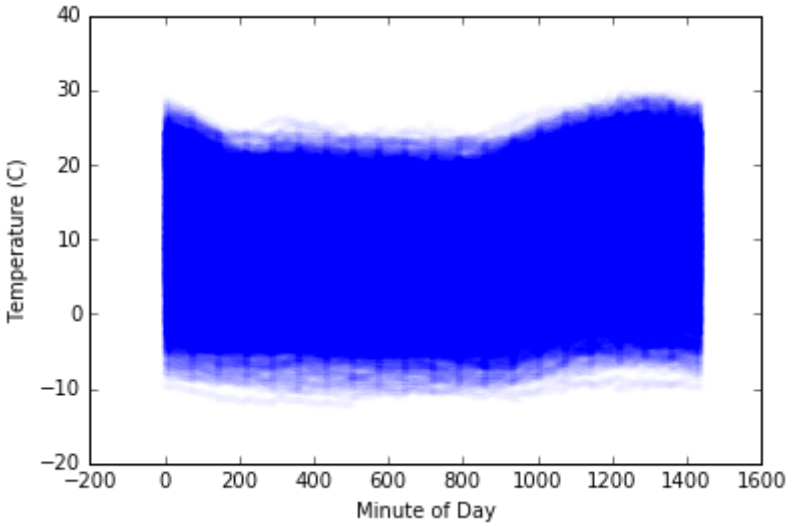
Model Results Analysis

It seems as though the more RBF centers we insert, the smaller the MSE of all models, and the less negative the R^2 colinearity between the true temperatures and our predictions. This makes sense, but not as helpful, since it might result in overfitting as we increase the RBF centers. Taking the most extreme case - we could calculate a center FOR EACH DAY/EACH MINUTE. In that case, the accuracy will be great, because we are essentially being most flexible with our model to fit the training data. But, we are not modeling a generalized reoccuring minute/daily/yearly/ pattern anymore; our model will overfit to the specific data that we have, and might be worse in predictions in the long term.

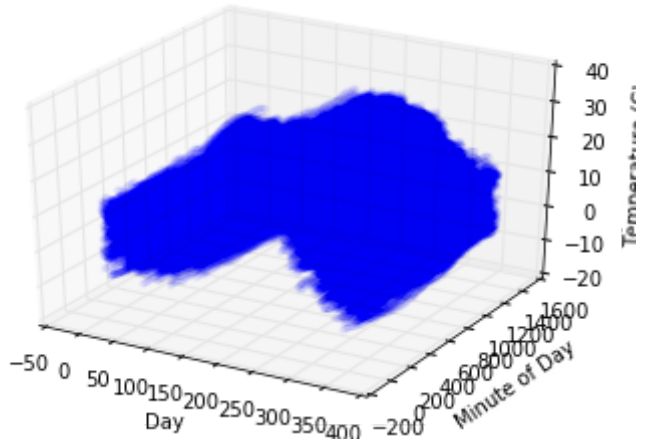
In [13]:
```
%matplotlib inline

days = np.concatenate((days_train, days_test))
minutes =
np.concatenate((mins_train.reshape(-1,1),mins_test.reshape(-1,1)))
temperatures = np.concatenate((temp_y_train, temp_y_test))


# time-of-year contribution plot
plt.scatter(days,temperatures, color='blue', s=10,alpha = 0.01)
plt.xlabel("Day")
plt.ylabel("Temperature (C)")
plt.show()
```



In [85]:
```
# time-of-day contribution
plt.scatter(minutes,temperatures, color='blue', s=10,alpha=0.01)
plt.xlabel("Minute of Day")
plt.ylabel("Temperature (C)")
plt.show()
```



In [87]:
```
#Make a 3D plot showing temperature as a function of (day, time). Make s
ure to label your axes!
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(days,minutes,temperatures,cmap='viridis', s=10,alpha=0.01)
ax.set_xlabel('Day')
ax.set_ylabel('Minute of Day')
ax.set_zlabel('Temperature (C)')
plt.show()
```



In [ ]:
```
# ColorMap 3D - doesn't load
from matplotlib import cm
#Make a 3D plot showing temperature as a function of (day, time). Make s
ure to label your axes!
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(days,minutes,temperatures,cmap='viridis', s=10,alpha=0.01)
ax.plot_surface(days, minutes,temperatures, cmap=cm.coolwarm,
linewidth=0, antialiased=False, alpha = 0.1, zorder=0)
ax.set_xlabel('Day')
ax.set_ylabel('Minute of Day')
ax.set_zlabel('Temperature (C)')
plt.show()
```

Yosemite Village Rain

There is occasionally rain in Yosemite Village. Fortunately the original dataset also includes precipitation.

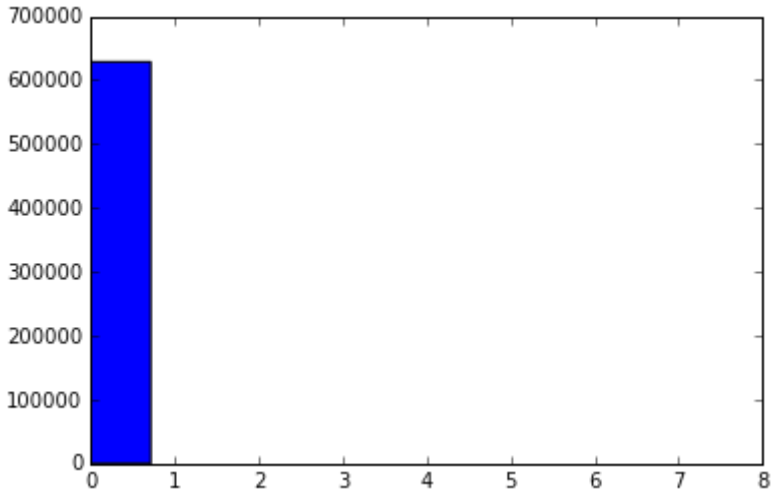
1. Using logistic regression predict the probability of rain in a given day of the year.
2. What accuracy would a classifier get if it simply predicted “No rain” all the time?
3. Describe your model and data preparation to a level of detail so that another machine learning researcher could reproduce your results.

Submit your code, and results as a pdf produced from a Python Notebook. Include relevant images or plots.

Process

I'm using logistic regression since it gives us directly "probability" resutls, as P(Y|X), for whatever we insert it.

In [50]:
```
# I'll define "Rain" as in percipitation is any value above 0 (since the
re are not that many anyway)
plt.hist(df.Rain)
plt.show()
print("There are {} rainy observations out of {} observations, percentag
e is {}".format(len(df.Rain[df.Rain>0]), len(df.Rain),
float(len(df.Rain[df.Rain>0]))/ len(df.Rain)))

print('(Q2:) Therefore, an accuracy for a classifier that always predict
s "NO" would be: {}%'.format(100.0-(float(len(df.Rain[df.Rain>0]))/
len(df.Rain))))
```




```
There are 15677 rainy observations out of 629341 observations, percenta
ge is 0.0249101838272
(Q2:) Therefore, an accuracy for a classifier that always predicts "NO"
would be: 99.9750898162%
```

In [58]:
```
## Convert Rain to binary classification
df['RainBinary'] = (df['Rain'] > 0)
df.head()
```

Out[58]:

	Date	Yr	DayInt	Min	HHmm	Temp	Rain	RainBinary
0	2011-01-01	2011	1	5.0	5.0	-6.4	0.0	False
1	2011-01-01	2011	1	10.0	10.0	-6.5	0.0	False
2	2011-01-01	2011	1	15.0	15.0	-6.5	0.0	False
3	2011-01-01	2011	1	20.0	20.0	-6.5	0.0	False
4	2011-01-01	2011	1	25.0	25.0	-6.7	0.0	False

In [63]:
```
X = df['Day'].values
y = df['RainBinary'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1
23)

X_train = X_train.reshape(-1,1)
X_test = X_test.reshape(-1,1)


logreg = LogisticRegression()
logreg.fit(X_train,y_train)
pred = logreg.predict(X_test)

print "R^2 score for test set:", logreg.score(X_test,y_test)
```

```
R^2 score for test set: 0.974525855494
```

Results

Results show supposedly great accuracy of 0.974525855494. However, as we saw before, there are only 0.02% of observations with Rain in them, and the Naive classifier that would classify everything as 'No' would have abotu the same or even marginally better accuracy - of 0.975 vs 0.974.

In [ ]:
```
df_grouped = df.groupby(['Year','Day'], as_index=False)['RainBinary'].ag
g(['id','max'])
```

Remark

I wasn't sure what was meant by predicting for each day. I took it as a simpler case of rpredicting per a given moment; however, maybe the intention was predicting per DAY. In that case, I would need to group by day, take the maximum of (Binary Rain Classification), then model per that. I tried this in many ways but it didn't work, and I'm not feeling good enough to keep trying, but hope to revisit this kind of transformaiton in the future.

For example: df_grouped = df.groupby(['Year','Day'], as_index=False)['RainBinary'].agg(['id','max'])

In [ ]:
```

```