Exploring Regression for problem set

September 27, 2017

1 Regression exploration

In this problem set you will explore working with regression. The first part of the notebook will define basic functions and import data that we need. Note in particular that I load `pandas` and `seaborn`. Both of these packages might have to be installed before you can run this notebook - you can modify the code below to not use them but as they are both powerful and useful I recommend you get them working.

Pandas is a library designed to handle data in a flexible way. It is derived from `R`'s data frames and Pandas is particularly nice for moderate sized datasets and dataset that mix datatypes. To learn more about `pandas` you can check out the project web page: http://pandas.pydata.org/.

Seaborn is a visualisation package particularly suited for statistical data. It builds on `matplotlib` but adds functionality and improves the default appearance (although not all defaults are that great IMHO).There is plenty of information on this on the Seaborn web page: https://seaborn.pydata.org/index.html

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from astroML.datasets import fetch_sdss_sspp
from astroML.plotting import hist
import seaborn as sns
import pandas as pd
import sys
import cPickle
from sklearn.linear_model import Ridge, Lasso

# The matplotlib inline special command must be commented out if you run this as
# a script or on the ipython/python command line.
%matplotlib inline
```

In [2]:
```
# This sets the default appearance of seaborn and its colour palette
sns.set(style="white")
sns.set_palette('colorblind')
```

In [3]:
```
def pickle_to_file(data, fname):
    """Save a variable simply to a file"""
    try:
        fh = open(fname, 'w')
        cPickle.dump(data, fh)
        fh.close()
    except:
        print "Pickling failed!", sys.exc_info()[0]

def pickle_from_file(fname):
    """Restore a variable saved with pickle_to_file"""
    try:
```

```
            fh = open(fname, 'r')
            data = cPickle.load(fh)
            fh.close()
        except:
            print "Loading pickled data failed!", sys.exc_info()[0]
            data = None

        return data
```

```
In [4]: # This reads in data on stellar sources from the SDSS.
        # If running this on a computer without the setup recommended on blackboard, it will
        # download the data from the net the first time called.
        data = fetch_sdss_sspp()
```

It can be useful to know what is in the data array! One good way is to check the documentation of fetch_sdss_spp, another is to print `data.dtype.names` which will show you the name of the columns in the data.

```
In [6]: print data.dtype.names
```

```
('ra', 'dec', 'Ar', 'upsf', 'uErr', 'gpsf', 'gErr', 'rpsf', 'rErr', 'ipsf', 'iErr', 'zpsf', 'zErr', 'pml
```

```
In [8]: # Extract some data.
        ug = data['upsf']-data['gpsf']
        gr = data['gpsf']-data['rpsf']
        ri = data['rpsf']-data['ipsf']
        iz = data['ipsf']-data['zpsf']
        T = data['Teff']
```

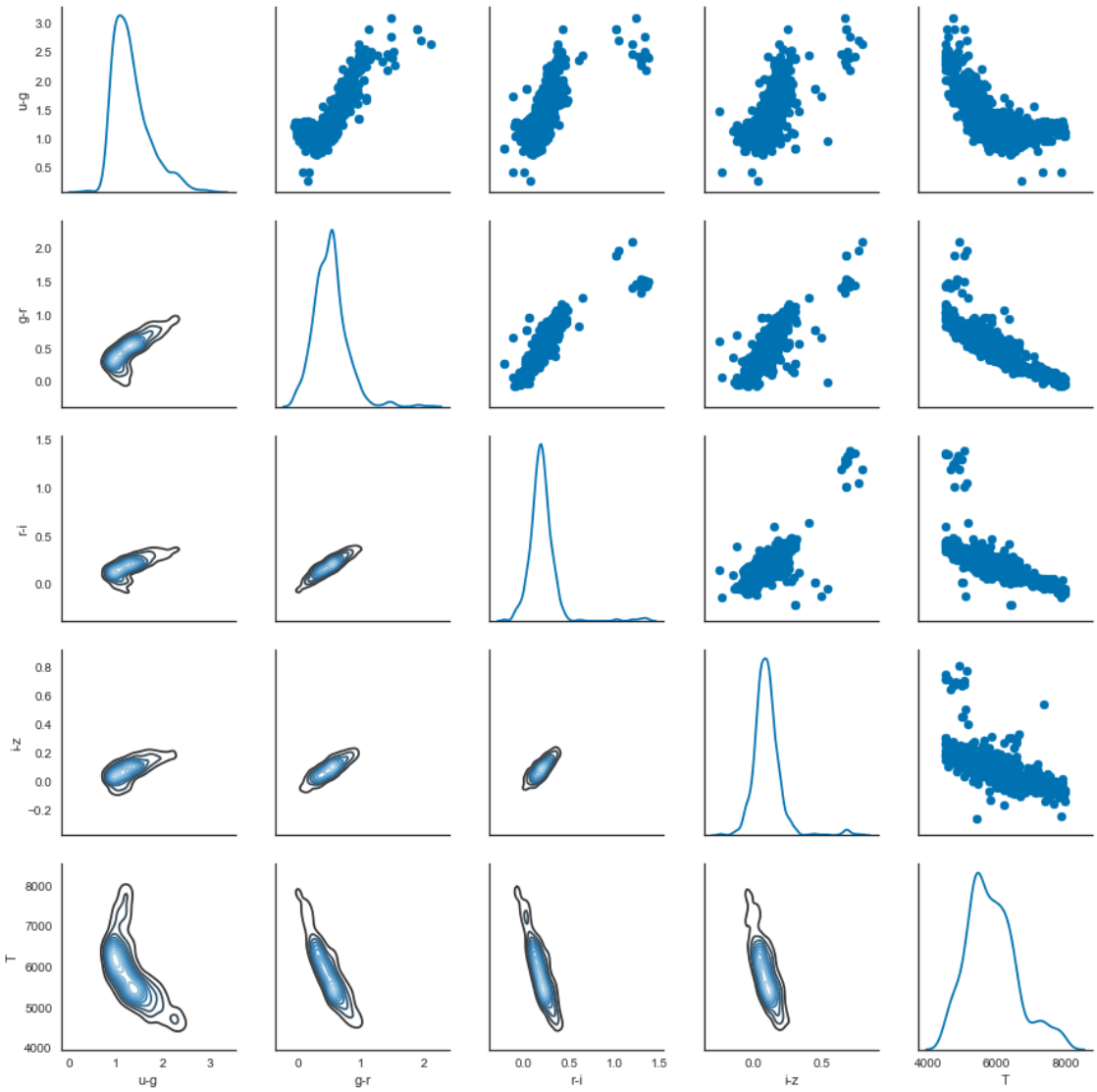
```
In [9]: # Understand what the line below does! Try without the transpose and see the shape.
        X = np.vstack((ug, gr, ri, iz, T)).T
        M = np.vstack((ug, gr, ri, iz)).T
```

1.1 Visualisation of the data

A good way to visualise data like this is to use a pair plot grid - this is provided by Seaborn in the `PairGrid` function. This allows lower and upper and diagonals to show different things. The lower half and diagonals show Kernel Density plots - we will come back to these later in the course - for now think of them as smooth versions of 2D and 1D histograms respectively. (this does take a bit to run btw! but to make it quicker I only use the first 1000 points).

```
In [12]: # Here I make a Pandas's DataFrame
         df = pd.DataFrame(X[0:1000, :], columns=['u-g', 'g-r', 'r-i', 'i-z', 'T'])
         g = sns.PairGrid(df, diag_sharey=False)
         g.map_lower(sns.kdeplot, cmap="Blues_d")
         g.map_upper(plt.scatter)
         g.map_diag(sns.kdeplot)
```

```
Out[12]: <seaborn.axisgrid.PairGrid at 0x11ef9b350>
```



There is plenty of co-linearity in this data. Let me now assume that we can write

$$T = \theta_0 + \theta_1(u-g) + \theta_2(g-r) + \theta_3(r-i) + \theta_5(i-z)$$

and we want to constrain θ.

Your task now is to do this for different types of regression. The foundation of the code is provided in the lecture notes of lecture 1 - feel free to use that!

1.2 Problem 1: Regular linear regression

a) Fit a model to the data to predict T, find θ and plot the residuals between the best fit regression line and the true values as a function of the temperature T. What do you conclude about your model?

b) Pick 8 stars at random from the sample and redo the fit. Do this twice - are the best fit lines consistent? [different people will reach different conclusions here!]

1.3 Problem 2: Ridge regression

Repeat a) & b) from problem 1 for RidgeRegression. How does the answer for b) differ from that for regular linear regression.

1.4 Problem 3: LASSO regression

Finally repeat the tasks in problem 1 for LASSO regression. Reflect on how the answer for b) changes relative to the other two questions.

In the lecture we saw that for a sufficiently large regularisation parameter, λ, the only remaining term is the one for θ_2 - can you visually explain why this is?