Biomass_Regression_All_Data_two_vars

May 24, 2017

This script builds a regression model to predict biomass based on lidar metrics. Data points at all scan angles are used.

```
In [1]: %matplotlib inline
        import pandas, statsmodels, numpy, math, scipy.stats
        from matplotlib import pyplot as plt
        #from __future__ import division
        from numpy import log as ln
        import warnings
        warnings.filterwarnings('ignore')
```

Data Prep Notes: The clipped las files were imported into ArcMap and converted to point shapefiles. The point shapefiles were each joined to the plot discs to add columns for plot ID and biomass. The joined points were then exported to .csvs in the directory `CSVs_ClippedAndJoined`. The order of fields should run: x, y, z(height), angle, plotID, biomass. Points from discs with no associated biomass were dropped.

```
In [2]: firstfile = r"D:\Users\jkelly\CSVs_ClippedAndJoined\first_all_clipped_and_j
        first = pandas.read_csv(firstfile, header=0)
        lastfile = r"D:\Users\jkelly\CSVs_ClippedAndJoined\last_all_clipped_and_joi
        last = pandas.read_csv(lastfile, header=0)
        groundfile = r"D:\Users\jkelly\\CSVs_ClippedAndJoined\Grounds_clipped_and_j
        ground = pandas.read_csv(groundfile, header=0)
```

```
In [3]: first.columns=['X1','Y1','X','Y','height','angle','PlotID','biomass']
        del first['X1']
        del first['Y1']
        last.columns=['X1','Y1','X','Y','height','angle','PlotID','biomass']
        del last['X1']
        del last['Y1']
        ground.columns=['X1','Y1','X','Y','height','angle','PlotID','biomass']
        del ground['X1']
        del ground['Y1']
```

```
In [4]: first=first.sort('PlotID')
        last=last.sort('PlotID')
        ground=ground.sort('PlotID')
```

The following function produces a series of metrics summarizing the lidar height measurements on each inventory plot for first, last and ground returns: mean, coefficient of variation, 10th, 50th and 90th deciles for height and density fraction, following Gobakken et al, 2012.

```
In [5]: def heightstats(SERIES, I):
            # function to compute "cloud metrics" on a height or elevation series
            totalechoes = len(SERIES)
            MEAN = numpy.mean(SERIES)
            CV = scipy.stats.variation(SERIES)
            #coeff of var (ratio of biased standard dev to mean)
            H10 = numpy.percentile(SERIES, 10)
            H10series = SERIES[SERIES > H10]
            H10count = len(H10series)
            D10 = H10count/totalechoes
            # "canopy density" = the proportion of laser
            # hits above fraction to total # of echoes
            # described in Gobakken et al. 2012, p. 447
            H50 = numpy.percentile(SERIES, 50)
            H50series = SERIES[SERIES > H50]
            H50count = len(H50series)
            D50 = H50count/totalechoes
            H90 = numpy.percentile(SERIES, 90)
            H90series = SERIES[SERIES > H90]
            H90count = len(H90series)
            D90 = H90count/totalechoes

            outlist = [I, MEAN, CV, H10, H50, H90, D10, D50, D90]

            return(outlist)
```

The next section builds a dataframe associating each plot with its biomass, elevation, and computed summary statistics. The plot-level biomass numbers given in the data are in units of Mg/ha, which are here converted to the actual mass on a the 0.04 ha plots.

```
In [6]: heightlist = []
        for i in first.PlotID.unique():
            # subset corresponding to ith plot id
            L = last[last.PlotID == i]
            F = first[first.PlotID == i]
            G = ground[ground.PlotID == float(i)]

            # just height column from subset
            Last_heightseries = L.height
            First_heightseries = F.height
            elevseries = G.height

            # just biomass column from subset
            biomass = L.biomass*0.04
            # the 0.04 is to convert Mg/ha to Mg/400m^2 plot
```

```
    if len(Last_heightseries)>=1 and len(First_heightseries)>=1:
        # ^^^  gets rid of empty height series

        # fetch "cloud metrics"
        Last_stats = heightstats(Last_heightseries, i)
        First_stats = heightstats(First_heightseries, i)

        if len(elevseries) > 0:
            Gmean = numpy.mean(elevseries)
        else: Gmean = -1

        # build a list for next row in dataframe
        stats = Last_stats
        for item in First_stats:
            stats.append(item)
        b = numpy.mean(biomass)
        stats.append(b)
        stats.append(Gmean)
        if b>0: heightlist.append(stats) #gets rid of NaN


heightarray = numpy.array(heightlist)     # build the dataframe
df = pandas.DataFrame(heightarray)
columnlist = ['PlotID', 'L_mean', 'L_cv', 'L_h10', 'L_h50',
              'L_h90', 'L_d10', 'L_d50', 'L_d90', 'PlotID_again',
              'F_mean', 'F_cv', 'F_h10', 'F_h50', 'F_h90', 'F_d10',
              'F_d50', 'F_d90', 'Biomass', 'elev']

df.columns = columnlist
# delete duplicate columns
del df['PlotID_again']
```

In [7]: df.describe()

Out[7]:

```
           PlotID       L_mean        L_cv       L_h10       L_h50       L_h
count  154.000000  154.000000  154.000000  154.000000  154.000000  154.0000
mean    93.116883    2.882272    1.508728    0.002792    0.931591    8.7727
std     50.724934    1.713424    0.433447    0.012153    1.381876    4.3586
min      2.000000    0.206482    0.666363    0.000000    0.040000    0.2400
25%     55.250000    1.588867    1.202409    0.000000    0.140000    5.4925
50%     96.000000    2.550631    1.426635    0.000000    0.270000    8.3850
75%    135.750000    4.170365    1.715680    0.000000    0.993750   11.7165
max    178.000000    7.915523    2.857065    0.120000    6.380000   19.8040

            L_d10       L_d50       L_d90      F_mean        F_cv       F_h
count  154.000000  154.000000  154.000000  154.000000  154.000000  154.0000
mean     0.842814    0.496161    0.100046   11.446377    0.504331    3.7304
```

```
std       0.039673    0.005197    0.000534    3.564578    0.289637    4.2103
min       0.729089    0.468468    0.098137    1.220464    0.166920    0.0000
25%       0.820128    0.494567    0.099872    9.087133    0.348159    0.1017
50%       0.847703    0.498274    0.100080   10.943383    0.431127    1.9790
75%       0.873282    0.499613    0.100323   13.433398    0.584139    6.4945
max       0.899889    0.500000    0.102804   21.774059    2.199853   16.2100

             F_h50       F_h90       F_d10       F_d50       F_d90      Bioma
count   154.000000  154.000000  154.000000  154.000000  154.000000  154.0000
mean     12.367338   17.210695    0.896085    0.499204    0.099676    5.7529
std       4.041083    3.812580    0.013928    0.002669    0.000431    2.8232
min       0.070000    4.961000    0.789216    0.468215    0.097949    0.9120
25%       9.820000   14.464000    0.898350    0.499200    0.099473    3.6420
50%      12.105000   16.930000    0.899754    0.499585    0.099782    5.1960
75%      15.020000   19.537500    0.899880    0.499881    0.100000    6.9258
max      23.000000   26.640000    0.900173    0.500000    0.100654   17.4634

              elev
count   154.000000
mean      3.118651
std       1.874205
min       0.194529
25%       1.573210
50%       2.635528
75%       4.435760
max       8.226820
```

Then a regression model is developed and its statistics are displayed. Though Gobakken et al developed different models for different forest classes, in each case both the response and explanatory variables were log-transformed. However, an exhaustive trial of all one-, two- and three-variable combinations revealed the "best" two-variable model –in terms of R^2, as well as AIC compared on identical variables – to be one which log-transforms the 50th percentile cutoff for height of first returns and also includes the untransformed value for first-return mean. To rule out (multi) collinearity, models with a condition number greater than 50 were excluded from consideration.

```
In [19]: def formula_tester(VAR1, VAR2):
             try:
                 #three variables three logs (not a contender)
                 #Formula = "ln(Y) ~ 1 + ln(" + VAR1 + ") + ln(" + VAR2 + ") + ln("

                 # three variables two logs - best model so far
                 #Formula = "ln(Y) ~ 1 + ln(" + VAR1 + ") + ln(" + VAR2 + ") + " +

                 # three variables one log -- good
                 #Formula = "ln(Y) ~ 1 + ln(" + VAR1 + ") + " + VAR2 + " + " + VAR3

                 # three variables no log
                 #Formula = "ln(Y) ~ 1 + " + VAR1 + " + " + VAR2 + " + " + VAR3
```

```
                # one variable log (not a contender)
                #Formula = "ln(Y) ~ 1 + ln( " + VAR1 + ")"

                # two variables two logs (not a contender)
                #Formula = "ln(Y) ~ 1 + ln( " + VAR1 + ") + " + "ln( " + VAR2 + ")

                #two variables one log -- not bad
                Formula = "ln(Y) ~ 1 + ln(" + VAR1 + ") +" + VAR2

                result = sm.ols(formula=Formula, data=df).fit()
                if result.condition_number < 50 and result.rsquared>0.7:
                    return(result.rsquared, result.aic, VAR1, VAR2)
                    #return(result.rsquared, result.aic, VAR1, VAR2) #two variable
                else:
                    return()
            except:
                return()
```

```
In [20]: import statsmodels.formula.api as sm

         Y = df['Biomass']
         variable_list = ["df['F_mean']","df['L_mean']","df['F_cv']","df['L_cv']",
                          "df['F_h10']","df['L_h10']","df['F_h50']","df['L_h50']",
                          "df['F_h90']","df['L_h90']","df['F_d10']","df['L_d10']",
                          "df['F_d50']","df['L_d50']","df['F_d90']","df['L_d90']",
                          "df['elev']"]

         regression_parameters_list = []
         for variable1 in variable_list:
             for variable2 in variable_list:

                 regression_parameters_list.append(formula_tester(variable1, variab
             parameters = formula_tester(variable1, variable2)

             #remove empty tuples and report best r-squareds
             if len(parameters) > 1 and parameters[0]>0.7:
                 regression_parameters_list.append(parameters)
         sorted(filter(lambda a: a != (), regression_parameters_list))
         #regression_parameters_list
```

```
Out[20]: [(0.7136724104353942, 25.914568617848545, "df['L_h90']", "df['F_mean']"),
          (0.7153631014376780l, 25.002542806256713, "df['F_cv']", "df['F_mean']"),
          (0.71811908422823867, 23.504179693283731, "df['elev']", "df['F_mean']"),
          (0.71849397496043177, 23.299229322244571, "df['L_mean']", "df['F_mean']")
          (0.76808072198590027, -6.5405004686309667, "df['F_h50']", "df['F_mean']")
          (0.77657977233883646, -12.290081906506089, "df['F_h50']", "df['F_h50']")]
```

This was the second best regression model with 2 independent variables, condition number < 50.

The best model produced unacceptable overestimates on non-forested areas.

```
In [21]: var1 = df['F_h50']
         var2 = df['F_mean']


         var1str = "ln(df['F_h50'])"
         var2str = "df['F_mean']"


         Formula = "ln(Y) ~ 1 + " + var1str + " + " + var2str
         result = sm.ols(formula=Formula, data=df).fit()
         print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  ln(Y)   R-squared:                       0.768
Model:                            OLS   Adj. R-squared:                  0.765
Method:                 Least Squares   F-statistic:                     250.0
Date:                Wed, 24 May 2017   Prob (F-statistic):           1.21e-48
Time:                        10:45:24   Log-Likelihood:                 6.2703
No. Observations:                 154   AIC:                            -6.541
Df Residuals:                     151   BIC:                             2.570
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
Intercept           0.4665      0.068      6.857      0.000         0.332     0.601
ln(df['F_h50'])    -0.2049      0.034     -5.998      0.000        -0.272    -0.137
df['F_mean']        0.1449      0.007     19.755      0.000         0.130     0.159
==============================================================================
Omnibus:                       39.687   Durbin-Watson:                   1.609
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              206.187
Skew:                          -0.761   Prob(JB):                     1.69e-45
Kurtosis:                       8.460   Cond. No.                         45.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
```

```
In [22]: result.bse

Out[22]: Intercept          0.068035
         ln(df['F_h50'])    0.034156
         df['F_mean']       0.007336
         dtype: float64
```

```
In [25]: intercept = result.params[0]
         b1 = result.params[1]
         b2 = result.params[2]

         intercept, b1, b2
```

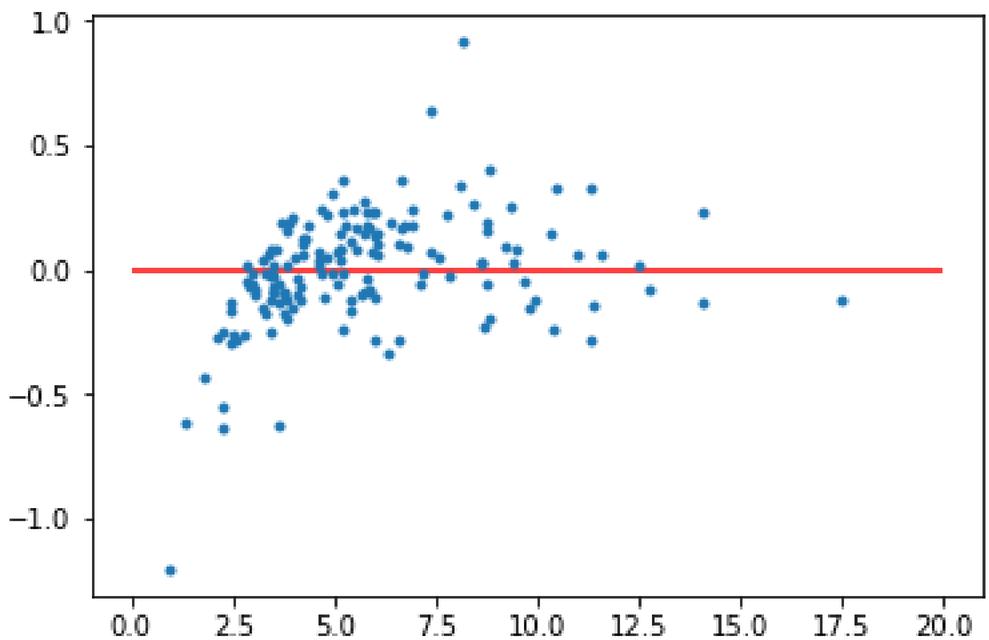
```
Out[25]: (0.46653258757917682, -0.20488443892984287, 0.14492814972141516)
```

```
In [27]: f = open('parameters_all_two.csv', 'w')
         with f:
             f.write(str(intercept)+' \n ')
             f.write(str(b1)+ ' \n ' )
             f.write(str(b2)+ '\n' )

         f.close()
```

Inspecting the data and residuals:

```
In [29]: R = result.resid
         plt.plot(Y, R, '.')
         plt.hlines(0,0, 20,colors=u'r')
         plt.show()
```




```
In [30]: df.mean()
```

```
Out[30]: PlotID      93.116883
         L_mean       2.882272
```

```
L_cv        1.508728
L_h10       0.002792
L_h50       0.931591
L_h90       8.772786
L_d10       0.842814
L_d50       0.496161
L_d90       0.100046
F_mean     11.446377
F_cv        0.504331
F_h10       3.730494
F_h50      12.367338
F_h90      17.210695
F_d10       0.896085
F_d50       0.499204
F_d90       0.099676
Biomass     5.752972
elev        3.118651
dtype: float64
```

In [31]:
```
#df.to_csv("ALL_plots_yhat")
```

In [33]:
```
df.describe()
```

Out[33]:
```
           PlotID      L_mean        L_cv       L_h10       L_h50       L_
count  154.000000  154.000000  154.000000  154.000000  154.000000  154.000
mean    93.116883    2.882272    1.508728    0.002792    0.931591    8.772
std     50.724934    1.713424    0.433447    0.012153    1.381876    4.358
min      2.000000    0.206482    0.666363    0.000000    0.040000    0.240
25%     55.250000    1.588867    1.202409    0.000000    0.140000    5.492
50%     96.000000    2.550631    1.426635    0.000000    0.270000    8.385
75%    135.750000    4.170365    1.715680    0.000000    0.993750   11.716
max    178.000000    7.915523    2.857065    0.120000    6.380000   19.804

            L_d10       L_d50       L_d90      F_mean        F_cv       F_
count  154.000000  154.000000  154.000000  154.000000  154.000000  154.000
mean     0.842814    0.496161    0.100046   11.446377    0.504331    3.730
std      0.039673    0.005197    0.000534    3.564578    0.289637    4.210
min      0.729089    0.468468    0.098137    1.220464    0.166920    0.000
25%      0.820128    0.494567    0.099872    9.087133    0.348159    0.101
50%      0.847703    0.498274    0.100080   10.943383    0.431127    1.979
75%      0.873282    0.499613    0.100323   13.433398    0.584139    6.494
max      0.899889    0.500000    0.102804   21.774059    2.199853   16.210

            F_h50       F_h90       F_d10       F_d50       F_d90     Biom
count  154.000000  154.000000  154.000000  154.000000  154.000000  154.000
mean    12.367338   17.210695    0.896085    0.499204    0.099676    5.752
std      4.041083    3.812580    0.013928    0.002669    0.000431    2.823
min      0.070000    4.961000    0.789216    0.468215    0.097949    0.912
```

```
25%       9.820000   14.464000    0.898350    0.499200    0.099473    3.642
50%      12.105000   16.930000    0.899754    0.499585    0.099782    5.196
75%      15.020000   19.537500    0.899880    0.499881    0.100000    6.925
max      23.000000   26.640000    0.900173    0.500000    0.100654   17.463

             elev
count  154.000000
mean     3.118651
std      1.874205
min      0.194529
25%      1.573210
50%      2.635528
75%      4.435760
max      8.226820
```

```
In [ ]:
```