08_regression_bayesian_stan_ca

May 7, 2025

[ ]:
```python
import os
user = os.getenv('USER')
os.chdir(f'/scratch/cd82/{user}/notebooks')
```

0.1 Linear Regression - Bayesian Multiple Regression using the Stan library

`Stan` is a library that implements a Bayesian sampling Markov Chain Monte Carlo algorithm to predict the coeffiecients in a model. It has an advantage that parameters from complex, heirachical models can be estimated.

Set up `cmdstan` We have a pre-installed version of `cmdstan` on the scratch/cd82 filesystem, so we just need to tell `cmdstanpy` where it is.

[1]:
```python
import numpy as np
import pandas as pd
import json
import cmdstanpy
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

from sklearn.datasets import fetch_california_housing

# Load the dataset
housing = fetch_california_housing()
```

[3]:
```python
import cmdstanpy
import os

# install_dir = '/scratch/cd82/regression_cmdstan' # for NCI installation
install_dir = os.getenv('HOME')

# If we need to install cmdstan
cmdstanpy.install_cmdstan(overwrite=True, dir=install_dir)

# Pass the installation directory to cmdstanpy
cmdstanpy.set_cmdstan_path(install_dir+'/cmdstan-2.36.0/')
```

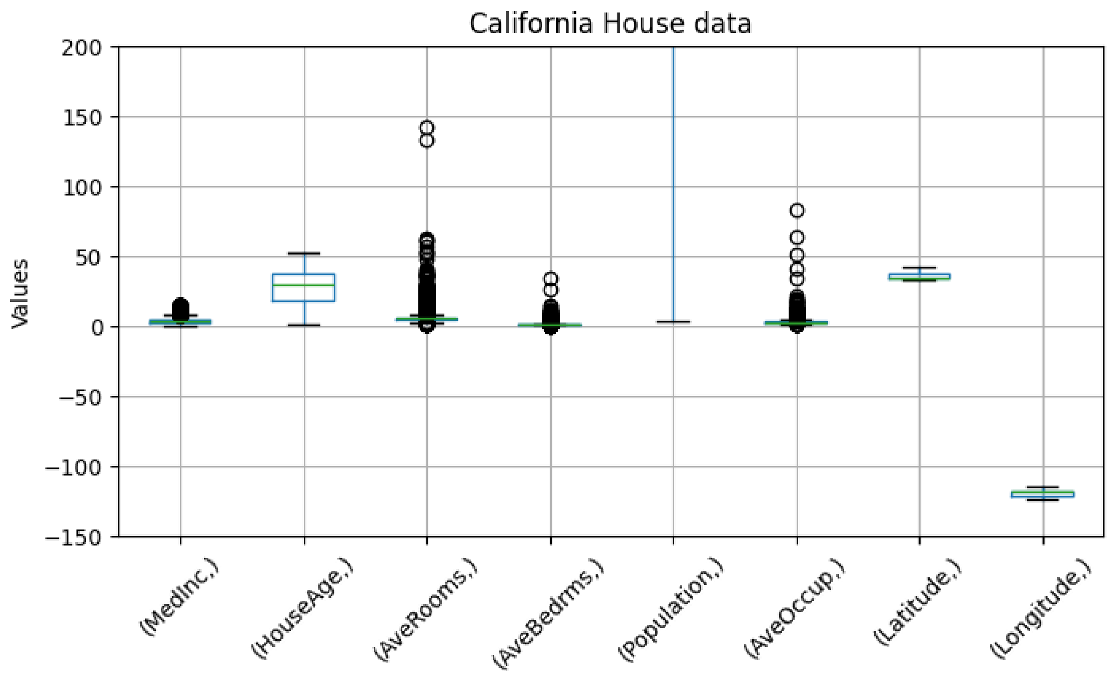
```
print(cmdstanpy.cmdstan_path())
```

```
CmdStan install directory: /home/jbowden
Installing CmdStan version: 2.36.0
Downloading CmdStan version 2.36.0
Download successful, file: /tmp/tmpf27dn7ay
Extracting distribution
Unpacked download as cmdstan-2.36.0
Building version cmdstan-2.36.0, may take several minutes, depending on your
system.
Installed cmdstan-2.36.0
Test model compilation
/home/jbowden/cmdstan-2.36.0
```

Set up our data and visualise

[4]:
```
from sklearn.datasets import make_regression

X = housing.data
y = housing.target

print(X.shape)

plt.figure(figsize=(8, 4))
# Create a box and whisker plot for each feature
X_df = pd.DataFrame(X, columns=[housing.feature_names])
#y_df = pd.DataFrame(y, columns=['y'])

# Create a box and whisker plot for each feature
X_df.boxplot()
plt.title('California House data')
plt.xticks(rotation=45)

plt.ylim(-150, 200)

plt.ylabel('Values')
plt.grid(True)
plt.show()

# Add a constant to the model (intercept)
X_int = sm.add_constant(X)
```

```
(20640, 8)
```



[5]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X_int, y,
    test_size=0.2,
    random_state=42)


N = X_train.shape[0]
K = X_train.shape[1]
print(X_train.shape)

X_train_np = X_train
print(type(X_train))
```

```
(16512, 9)
<class 'numpy.ndarray'>
```

Output data for cmdstan The `cmdstan` program requires datasets to be saved to a filesystem as `JSON` dictionaries.

[6]:
```
# N.B. Convert matrix and vector data to Numpy arrays
# and then add them to the dictionary as lists
stan_data = {'N': N, 'K': K, 'X': X_train_np.tolist(), 'y':y_train.tolist()}

N2 = X_test.shape[0]
```

```
K2 = X_test.shape[1]
stan_data_test = {'N': N2, 'K': K2, 'X': X_test.tolist(), 'y':y_test.tolist()}

# install_dir = os.getenv('HOME')
# point to the file on disk
data_file = os.path.join(install_dir, 'stan_data.json')
print("data_file", data_file)

# Save out dataset
with open(data_file, 'w') as file:
    json.dump(stan_data, file, indent=4)


data_file_test = os.path.join(install_dir, 'stan_data_test.json')
print("data_file_test", data_file_test)

# save our test data
with open(data_file_test, 'w') as file:
    json.dump(stan_data_test, file, indent=4)
```

```
data_file /home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_backup_pos
t_nci_v8/stan_data.json
data_file_test /home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_backu
p_post_nci_v8/stan_data_test.json
```

[7]:
```
from cmdstanpy import CmdStanModel
import time
import os
# Stan model code
stan_model_code = """

// This describes the input data
// Names must match what was saved to JSON data files
data {
    int<lower=0> N;
    int<lower=0> K;
    matrix[N, K] X;
    vector[N] y;
}

// These are what are being modelled
parameters {
    // a constant has been added to the input X data
    // so we do not need to model the intercept seperately
    // real intercept;
    vector[K] beta;
    real<lower=0> sigma;
```

```
}

// This is the 'Prior' definition of our model
model {
    beta ~ normal(0,1);           // The prior for our beta terms
    sigma ~ normal(0,1);          // The prior for the error term
    // intercept ~ normal(0, 1); // Not needed due X augmentation with a column␣
  ↪of 1's
    y ~ normal(X * beta, sigma);
}
"""

stan_file = os.path.join(install_dir, '/stan_model_code.stan')
with open(stan_file, 'w') as file:
    file.write(stan_model_code)

time.sleep(3)

print("stan_file:", stan_file)

model = cmdstanpy.CmdStanModel(stan_file=stan_file)
```

```
11:58:04 - cmdstanpy - INFO - compiling stan file /home/jbowden/machine_learing_
qcif/ml_gitlab_fork/notebooks_backup_post_nci_v8/stan_model_code.stan to exe
file /home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_backup_post_nci
_v8/stan_model_code
```

```
stan_file: /home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_backup_po
st_nci_v8/stan_model_code.stan
```

```
11:58:53 - cmdstanpy - INFO - compiled model executable: /home/jbowden/machine_l
earing_qcif/ml_gitlab_fork/notebooks_backup_post_nci_v8/stan_model_code
```

Select the model and print info

[8]:
```
print(model)
print(model.exe_info())
```

```
CmdStanModel: name=stan_model_code
         stan_file=/home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_b
ackup_post_nci_v8/stan_model_code.stan
         exe_file=/home/jbowden/machine_learing_qcif/ml_gitlab_fork/notebooks_ba
ckup_post_nci_v8/stan_model_code
         compiler_options=stanc_options={}, cpp_options={}
{'stan_version_major': '2', 'stan_version_minor': '36', 'stan_version_patch':
'0', 'STAN_THREADS': 'false', 'STAN_MPI': 'false', 'STAN_OPENCL': 'false',
'STAN_NO_RANGE_CHECKS': 'false', 'STAN_CPP_OPTIMS': 'false'}
```

[9]:
```
# fit the model
fit2 = model.sample(data=data_file,  iter_sampling=500, chains=4,␣
 ↪parallel_chains=2, max_treedepth=15)
```

```
12:01:54 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

16:04:01 - cmdstanpy - INFO - CmdStan done processing.
```

[ ]:
```
# file_path = '/tmp/tmp0w8evyvi/stan_model_code9dqjz_0u/
 ↪stan_model_code-20250307123500_0-stdout.txt'
# file_path = '/tmp/tmp0w8evyvi/stan_model_code9dqjz_0u/
 ↪stan_model_code-20250307123500_1.csv'
# with open(file_path, 'r') as file:
#     content = file.read()
#     print(content)

# mle = model.optimize(data=data_file)
# print(mle.column_names)
# print(mle.optimized_params_dict)
```

[10]:
```
fit2
```

[10]:
```
CmdStanMCMC: model=stan_model_code chains=4['method=sample', 'num_samples=500',
'algorithm=hmc', 'engine=nuts', 'max_depth=15', 'adapt', 'engaged=1']
 csv_files:
/tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-20250507120154_1.csv
/tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-20250507120154_2.csv
/tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-20250507120154_3.csv
/tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-20250507120154_4.csv
 output_files:
        /tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-
20250507120154_0-stdout.txt
        /tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-
20250507120154_1-stdout.txt
        /tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-
20250507120154_2-stdout.txt
        /tmp/tmpvun2w0lj/stan_model_codec1413l09/stan_model_code-
20250507120154_3-stdout.txt
```

[11]:
```
summary = fit2.summary()
print(summary)
```

```
                Mean          MCSE    StdDev       MAD           5%  \
lp__    -3276.870000  8.839670e-02  2.254180  2.031160 -3281.250000
beta[1]   -24.052700  2.664080e-02  0.611594  0.583477   -25.112200
beta[2]     0.474599  1.407550e-04  0.004558  0.004577     0.467185
beta[3]     0.011874  1.295560e-05  0.000513  0.000507     0.011023
beta[4]    -0.148245  2.162720e-04  0.006323  0.006236    -0.159060
beta[5]     0.851229  1.068620e-03  0.031934  0.030698     0.798993
beta[6]     0.000002  1.246620e-07  0.000005  0.000005    -0.000007
beta[7]    -0.003825  1.414620e-05  0.000492  0.000499    -0.004628
beta[8]    -0.294446  2.701840e-04  0.006938  0.006718    -0.306104
beta[9]    -0.287000  3.025790e-04  0.006993  0.006585    -0.299234
sigma       0.726775  9.035900e-05  0.003889  0.003906     0.720501

                 50%          95%  ESS_bulk  ESS_tail     R_hat
lp__    -3276.560000 -3273.770000   682.689  1011.100  1.006350
beta[1]   -24.047500   -23.019000   541.784   646.683  1.008040
beta[2]     0.474487     0.482347  1063.710  1379.430  1.001330
beta[3]     0.011864     0.012726  1598.130  1240.620  1.000600
beta[4]    -0.148348    -0.137877   861.386  1018.540  1.003150
beta[5]     0.850386     0.904865   896.406   994.332  1.003580
beta[6]     0.000002     0.000011  1852.550  1329.080  0.999953
beta[7]    -0.003827    -0.003015  1223.760   979.394  1.002500
beta[8]    -0.294482    -0.283162   673.155   874.594  1.002890
beta[9]    -0.287022    -0.275530   548.173   659.516  1.006800
sigma       0.726727     0.733217  1817.130  1548.510  1.001340
```

[12]:
```
print(fit2.diagnose())
```

```
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

Rank-normalized split R-hat values satisfactory for all parameters.

Processing complete, no problems detected.
```

0.1.1 Compare result with statsmodels OLS

[13]:
```
import statsmodels.api as sm
import pandas as pd
import numpy as np

# Fit the linear regression model
model_ols = sm.OLS(y_train,X_train)
results_ols = model_ols.fit()

# Get the R-squared value
r_squared = results_ols.rsquared
print('R sqrd (extracted):', r_squared)

# Print the summary of the model
print(results_ols.summary())

# Make predictions - If we had some other data
y_pred_sm_ols = results_ols.predict(X_test)
```

```
R sqrd (extracted): 0.6125511913966952
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.613
Model:                            OLS   Adj. R-squared:                  0.612
Method:                 Least Squares   F-statistic:                     3261.
Date:                Wed, 07 May 2025   Prob (F-statistic):               0.00
Time:                        16:04:52   Log-Likelihood:                -17998.
No. Observations:               16512   AIC:                         3.601e+04
Df Residuals:                   16503   BIC:                         3.608e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -37.0233      0.728    -50.835      0.000     -38.451     -35.596
x1             0.4487      0.005     95.697      0.000       0.439       0.458
x2             0.0097      0.000     19.665      0.000       0.009       0.011
x3            -0.1233      0.007    -18.677      0.000      -0.136      -0.110
x4             0.7831      0.033     23.556      0.000       0.718       0.848
x5          -2.03e-06   5.25e-06     -0.387      0.699   -1.23e-05    8.26e-06
x6            -0.0035      0.000     -7.253      0.000      -0.004      -0.003
x7            -0.4198      0.008    -52.767      0.000      -0.435      -0.404
x8            -0.4337      0.008    -52.117      0.000      -0.450      -0.417
==============================================================================
Omnibus:                     3333.187   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9371.466
Skew:                           1.071   Prob(JB):                         0.00
```

```
Kurtosis:                       6.006   Cond. No.                   2.38e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.38e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Generate predictions with a stan model

[14]:
```
# Generate predictions
# predictions = model.generate_quantities(data=data_file_test,␣
↪previous_fit=fit2)

beta_samples = fit2.stan_variable('beta')

print(type(fit2))
print('Column names: ', fit2.column_names)
print(type(beta_samples))
print(beta_samples.shape)
betas_best = beta_samples.mean(axis=0)
betas_stdev = beta_samples.std(axis=0)
```

```
<class 'cmdstanpy.stanfit.mcmc.CmdStanMCMC'>
Column names:  ('lp__', 'accept_stat__', 'stepsize__', 'treedepth__',
'n_leapfrog__', 'divergent__', 'energy__', 'beta[1]', 'beta[2]', 'beta[3]',
'beta[4]', 'beta[5]', 'beta[6]', 'beta[7]', 'beta[8]', 'beta[9]', 'sigma')
<class 'numpy.ndarray'>
(2000, 9)
```

[15]:
```
def predict(fit: cmdstanpy.stanfit.mcmc.CmdStanMCMC, paramname: str, data: np.
↪ndarray):
    mc_samples = fit.stan_variable(paramname)
    ave_vect = mc_samples.mean(axis=0)
    predictions = np.dot(data,ave_vect)
    return predictions
```

[16]:
```
# Use our beta estimates to predict y from the test data
y_pred_mc = predict(fit2, 'beta', X_test)
```

[17]:
```
# Evaluate the Stan derived model
mse = mean_squared_error(y_test, y_pred_mc)
print(f"Mean Squared Error: {mse}")
r2 = r2_score(y_test, y_pred_mc)
print(f"R² Score: {r2}")
```

```
Mean Squared Error: 0.5641395737472837
R² Score: 0.5694935069014959
```

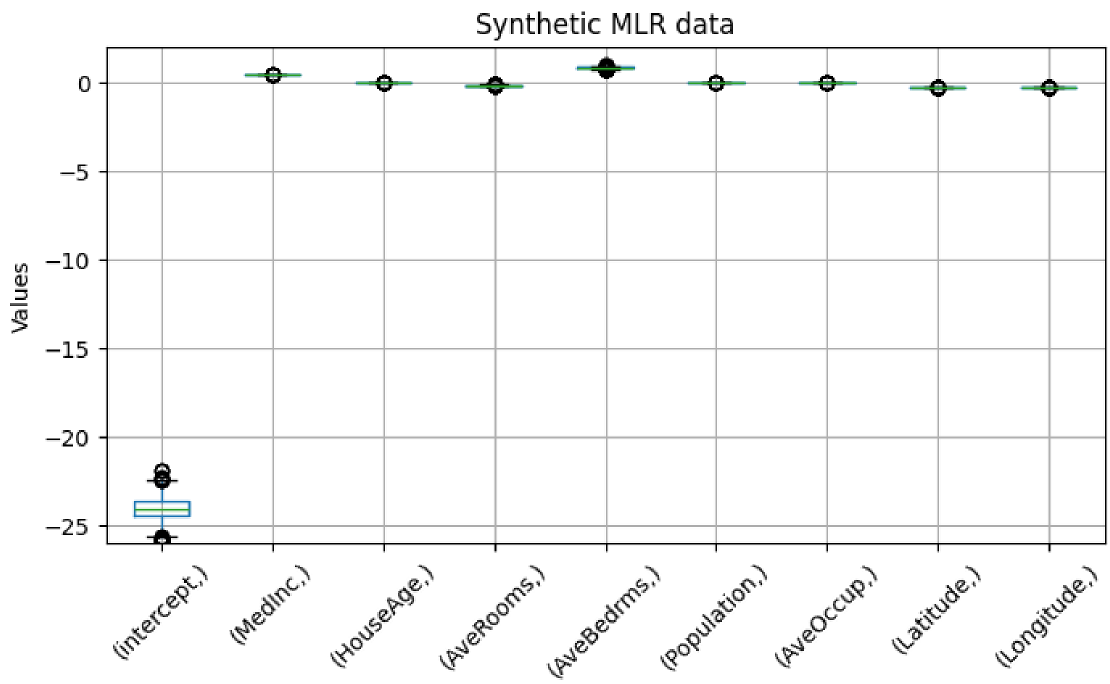
Save the samples

[19]:
```python
import os
install_dir = os.getenv('HOME')
outputpath =install_dir+'/stan_outputs'
fit2.save_csvfiles(dir=outputpath)
```

[ ]:
```python
# Reload using:
import pandas as pd
data = pd.read_csv(data_file)
# Convert the DataFrame to a dictionary
data_dict = data.to_dict(orient='list')
# Compile the Stan model (again if it has been deleted)
stan_file = os.path.join(install_dir, '/stan_model_code.stan')
model = cmdstanpy.CmdStanModel(stan_file=stan_file)
# Fit the model with the loaded data
fit_from_disk = model.sample(data=data_dict)
```

0.2 Visualisation

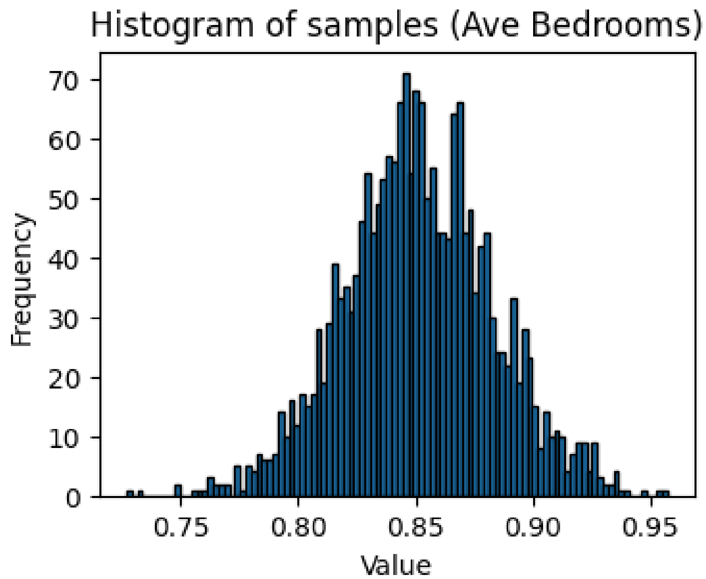
[27]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
# Create a box and whisker plot for each feature
beta_stan_df = pd.DataFrame(beta_samples,columns=[['intercept'] + housing.
 ↪feature_names])

# Create a box and whisker plot for each feature
beta_stan_df.boxplot()
plt.title('Synthetic MLR data')
plt.xticks(rotation=45)

plt.ylim(-26, 2)

plt.ylabel('Values')
plt.grid(True)
plt.show()
```



[28]:

```
import matplotlib.pyplot as plt
print(beta_samples.shape)
plt.figure(figsize=(4, 3))
plt.hist(beta_samples[:, 4], bins=100, edgecolor='black')
plt.title('Histogram of samples (Ave Bedrooms)')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

```
(2000, 9)
```


Histogram of samples (Ave Bedrooms)
Frequency
0
10
20
30
40
50
60
70
0.75
0.80
0.85
0.90
0.95
Value

[ ]: