Applied Probabilistic Programming & Bayesian Machine Learning

§1-2: Linear Regression in Stan

MIT IAP 2017 Max W. Shen

1 Linear Regression

Within this folder you find three files: regression_1-2.py, regression_1-2.R, and regression_1-2.stan.

```
data {
  int N;
  int D;
  vector[N] observations;
  matrix[N,D] features;
}

parameters {
  vector[D] weights;
  real<lower=0> sigma;
}

model {
  observations ~ normal(features * weights, sigma);
  weights ~ normal(0, 100);
  sigma ~ normal(0, 100);
}
```

Here, our N-by-1 dimensional vector of observations arises from matrix multiplying the N-by-D matrix of features with a hidden or latent D-by-1 dimensional vector of weights.

We can write this out, letting X denote the feature matrix, O the observations, and W the weight vector.

N-by-D matrix $\times$ D-by-1 vector $\rightarrow$ N-by-1 vector.

$X \times W \rightarrow O$.

$$\begin{aligned} O &\sim N(XW, \sigma) \\ W &\sim \text{weakly-informative}() \\ \sigma &\sim \text{weakly-informative}() \end{aligned} \tag{1}$$

Expanding the matrix notation and letting i index $1, ..., N$ the N observations, we can write:

$$\begin{aligned} E[O_i] &= X_{i,1}W_1 + X_{i,2}W_2 + ... + X_{i,D}W_D \\ E[O_i] &= \sum_{j=1}^{D} X_{i,j}W_j \end{aligned} \tag{2}$$

The mean of the i-th observation, $E[O_i]$, is a weighted sum of the D features we have for observation i. The observation itself, O_i, is normally distributed around this mean $E[O_i]$ with standard deviation σ.

It turns out this Bayesian setup is equivalent to the optimization setup you may be more familiar with:

$$arg \min_{W} (O - XW)^2 \tag{3}$$

Exercise 1

Simulations for Regressions

Our primary interest is recovering the hidden weight vector. Write simulation code for model 1.

Note that there are quite a few decisions you will have to make while simulating data. We encourage you to think this through on your own, though we describe the decisions we made on the next page.

2 Assessing Model Fit

We chose values $N = 30, D = 10, weights = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]$, and $\sigma = 1$. We drew the features independently from a normal distribution centered at 0 with standard deviation 1.

We observed the following results:

```
             mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
weights[0]  -5.11  2.9e-3   0.24  -5.58  -5.27  -5.11  -4.94  -4.62 7200.0    1.0
weights[1]  -4.05  3.8e-3   0.26  -4.56  -4.22  -4.05  -3.88  -3.53 4744.0    1.0
weights[2]  -3.16  3.7e-3   0.26  -3.68  -3.33  -3.16  -2.99  -2.64 5001.0    1.0
weights[3]  -1.91  3.5e-3   0.25  -2.42  -2.08  -1.91  -1.75  -1.41 5245.0    1.0
weights[4]  -1.19  3.3e-3   0.28  -1.73  -1.37  -1.19  -1.01  -0.64 7200.0    1.0
weights[5]   0.33  2.6e-3    0.2  -0.07    0.2   0.33   0.46   0.73 6066.0    1.0
weights[6]   1.21  2.4e-3    0.2    0.8   1.08   1.21   1.34   1.61 7200.0    1.0
weights[7]   1.91  3.2e-3   0.26   1.41   1.74    1.9   2.07   2.42 6479.0    1.0
weights[8]   3.18  3.1e-3   0.24   2.71   3.03   3.18   3.34   3.66 5955.0    1.0
weights[9]   3.86  4.4e-3    0.3   3.27   3.67   3.86   4.05   4.45 4590.0    1.0
sigma        1.17  3.0e-3   0.19   0.85   1.03   1.15   1.29   1.62 4088.0    1.0
lp__       -18.39    0.06   2.92 -24.96 -20.12 -18.01 -16.25 -13.92 2210.0    1.0
```

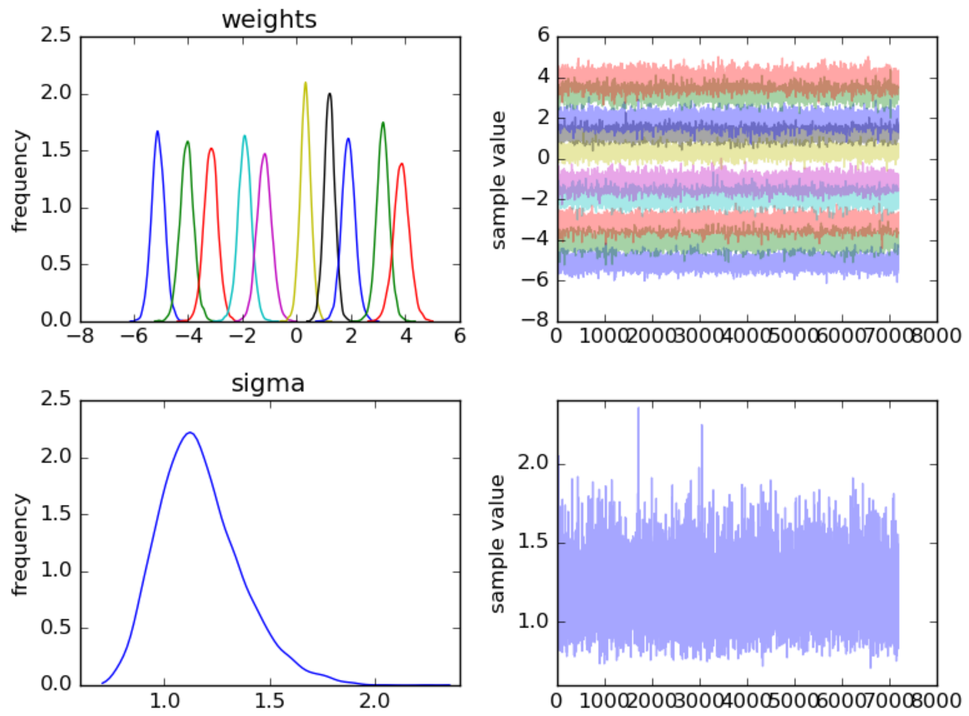

Figure 1: fit.png

3 Hierarchical Regression

Here, we present a more challenging exercise that studies LCVK's model for those who feel ready. If you would like more preparatory material, feel free to come back to this at a later time - the next assignment presents exercises in writing Stan code from the ground up.

This paper from Daphne Koller's lab describes a method for hierarchical regression:

`http://ai.stanford.edu/koller/Papers/Lee+al:ICML07.pdf`

In this paper, a method is given for finding the MAP (maximum a posteriori) estimates of parameters under the model. Stan enables us to write this model quickly without performing any analytic calculations while also finding the full posterior distribution rather than point estimates of the posterior mode (MAP). Stan also allows us to tweak the model quickly without having to re-derive a method for inference.

Let y_r denote the response variable for prediction task r and let there be k features per task denoted x_{rk}. Let each feature x_{rk} have an associated meta-feature vector $f_{rk} \in \mathbb{R}^l$. LCVK consider the following model:

$$\begin{aligned} y_r &\sim N(w_r^T x_r, \sigma) \\ w_{rk} &\sim N(0, \gamma_{rk}) \\ \gamma_{rk} &\sim gamma(C, D) \\ \gamma_{rk} &= \beta^T f_{rk} \end{aligned} \tag{4}$$

This model essentially is a double regression model, where w is the weights for the first level's k-length feature vector x, and β is the weights for the second level's l-length feature vector f.

One motivating example for this model is collaborative filtering as applied to Netflix recommendations. To predict a particular user's rating for a particular movie, we can assume a linear regression model given a vector of features encoding the particular user's ratings for all other movies they have rated. However, movies have different amounts of usefulness for predicting the rating on a particular movie (more similar genres, for example, is likely to be more predictive). Each movies' features would be the meta-features in this model.

The model not only learns to predict ratings for movies, but also learns to predict the relative importance of various features of movies which are most useful in predicting ratings. For example, the model will quantify just how helpful it is for two movies to share genres for the task of predicting ratings.

Exercise 2

Hierarchical Regression

Write simulation code and Stan code for the hierarchical regression model. Though LCVK notes that meta-features can be task-dependent, we recommend for simplicity to start with a task-independent set of meta-features. Concretely, this means each feature is associated with an l-length set of the same metafeatures. We also suggest setting concrete values for C and D as they are not quantities of interest. Also, note the constraint that $\gamma > 0$.

3.1 Transformed Parameters Block

```
data {
  ...
}

parameters {
  vector[D] weights;
  vector<lower=0>[L] hyperweights;
}

transformed parameters{
  vector[D] gammas;
  gammas = metafeatures * hyperweights;
}

model {
  ...
  gammas ~ gamma(C, D);
}
```

Our parameters of interest are the two weight vectors. Our gammas' values depend in a non-stochastic manner on the "hyperweights", but we also wish to place a distribution on them. The above code highlights how this can be done. Posterior distributions for the transformed parameter "gammas" will be returned by Stan as well, just like variables defined in the regular parameters block.

4 Bibliography and Additional Resources

The Stan reference is available here: http://mc-stan.org/documentation/

Chapter 14 of Bayesian Data Analysis 3 discusses regression in greater detail.