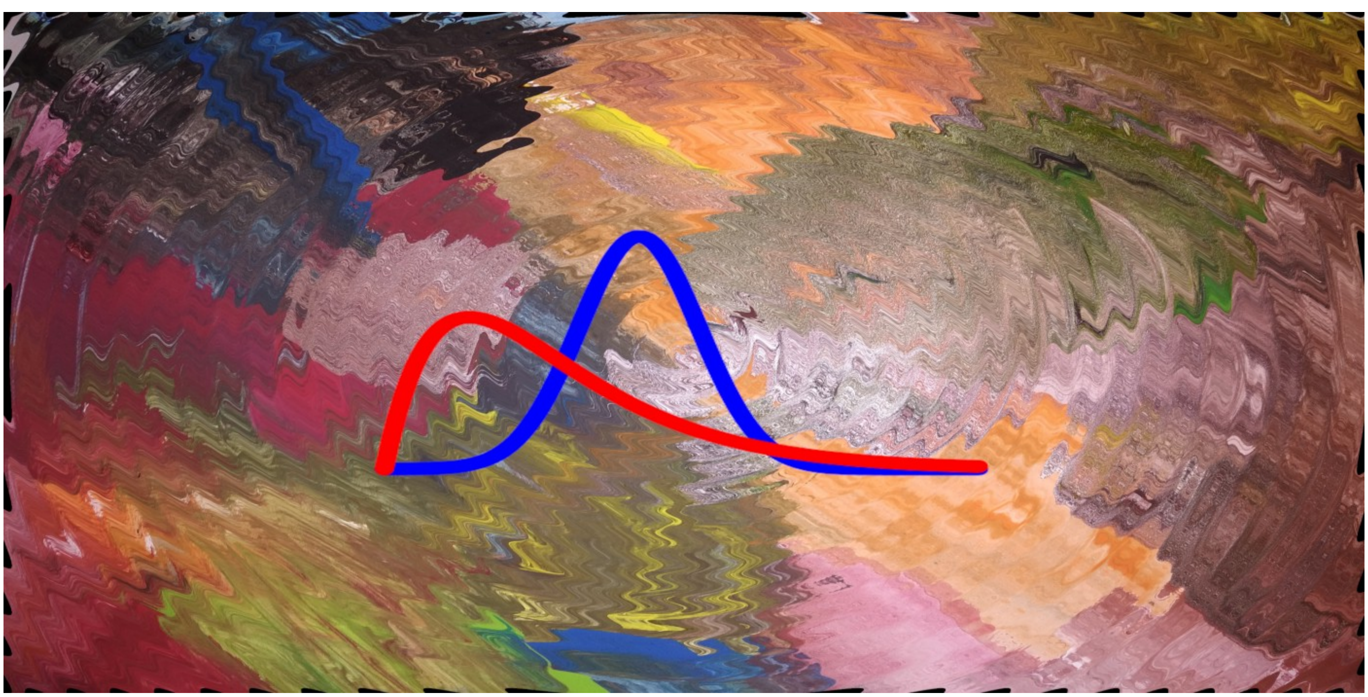
Introduction to Bayesian Logistic Regression

A practical demonstration of the Bayesian approach to classification using Python and PyJAGS.

Michel Kana, Ph.D Feb 7, 2020 · 8 min read ★

This article introduces everything you need in order to take off with Bayesian data analysis. We provide a step-by-step guide on how to fit a Bayesian logistic model to data using Python. You will be able to understand Bayesian fundamentals for classification without dealing with math.



Let's review the concepts underlying Bayesian statistical analysis by walking through a simple classification model.

The data

The data come from the 1988 Bangladesh Fertility Survey, where 1934 observations were taken from women in urban and rural areas. The authors of the dataset, Mn and Cleland aimed to determine trends and causes of fertility as well as differences in fertility and child mortality.

We will use the data in order to train a Bayesian logistic regression model that can predict if a given woman uses contraception.

The dataset is well suited to Bayesian logistic regression because being able to quantify uncertainty when analyzing fertility is the major component of population dynamics that decide the size, structure, and composition of populations (source 1, source 2).

A copy of the raw data can be found here. There are four attributes for each woman, along with a label indicating if she uses contraceptives. The attributes include:

- **district**: identifying code for the district the woman lives in,
- **urban**: type of region of residence,
- **living.children**: number of living children,
- **age-mean**: age of the woman (in years, centered around mean).

```
df_contraceptives = pd.read_csv('dataset_bang_contraceptive.csv')
df_contraceptives.head()
```

bayes_logreg_data.py hosted with ❤ by GitHub view raw

district	urban	living.children	age_mean	contraceptive_use
35	0	4	2.4400	0
22	0	2	-1.5599	1
29	0	2	-8.5599	1
5	0	3	-4.5599	1

The women are grouped into 60 districts.

```python
districts = df_contraceptives.sort_values(by="district").district.unique()
nb_districts = len(districts)
districts
```

bayes_logreg_districts1.py hosted with ❤ by GitHub view raw

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 55, 56, 57, 58, 59, 60, 61])
```

A little bit of pre-processing is required. We map the district number 61 to the number 54 so that the districts are in order, as you can see below.

```python
df_contraceptives.replace({'district': {61: 54}}, inplace=True)
df_contraceptives.sort_values(by="district").district.unique()
```

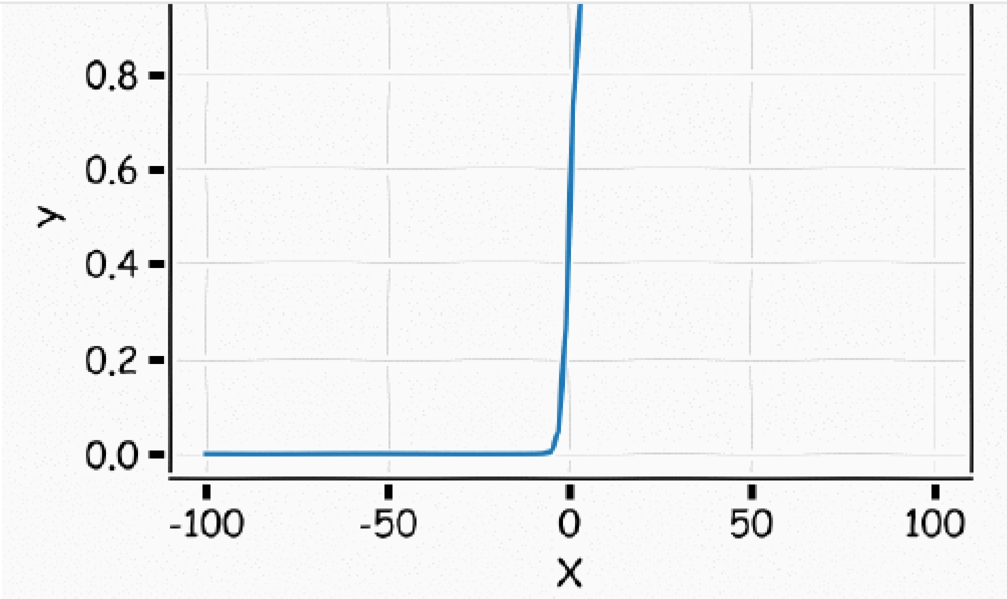
bayes_logreg_districts2.py hosted with ❤ by GitHub view raw

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60])
```

Logistic regression

Predicting whether or not a given woman uses contraceptives is an example of binary classification problem. If we denote attributes of the woman by *X* and the outcome by *Y*, then the likelihood of using contraceptives, *P(Y=1)*, would follow the logistic function below.

$$\boxed{P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}}$$



source: P. Protopapas, Harvard

The outcome depends on two parameters *β0* (intercept) and *β1* (slope). The intercept shifts the curve right or left, while the slope controls how steep the S-shaped curve is. If *β1* is positive, then the predicted *P(Y=1)* goes from zero for small values of *X* to one for large values of *X* and if *β1* is negative, then has the *P(Y=1)* opposite association.

For example, if *β1* is positive for the *age* predictor, it means that older women are more likely to use contraceptives than younger ones. The cut-off threshold where *age* will start increasing the probability of contraceptive usage would be governed by *β0.* Small values of *β0* would indicate that contraceptives usage is a widespread practice in the population.

Understanding the logistic function is important for motivating the Bayesian approach. You can learn more about classical logistic regression in our article below.

Why Deep Learning Works: solving a farmer's problem

In the beginning was the neuron: a gentle review of gradient descent, backpropagation, regression, autoencoder, CNNs...

towardsdatascience.com

and *β1*. However, we will not know how confident we are about these estimates.

Moreover, $\beta 0$ would normally differ from one district to another in real life. *β0* could be small in some districts, but large in others. A classical logistic regression model would still provide a single value for all regions, which could lead to wrong conclusions.

In one of our past articles, we highlighted issues with uncertainty in machine learning and introduced the essential characteristics of Bayesian methods. We gently explained the explicit use of probability for quantifying uncertainty in inferences based on statistical data analysis.

Bayesian nightmare. Solved!

Gentle introduction to Bayesian data analysis by examples and code in Python PyMC3.

towardsdatascience.com

Following Bayes, ideally, we want to take prior information into consideration when building our model for predicting contraceptive usage. If we only use the logistic equation, there is no direct way to include our **prior belief** about those parameters we are estimating.

The Bayesian approach allows us to make a prior good guess of the intercept and slope, based on our real-life domain knowledge and common sense. We can say for example that, from experience, the intercept is drawn from a normal distribution with mean μ=2 and standard deviation σ=1. We can also postulate that the slope for the predictors *urban*, *living.children*, and *age-mean* are 4, -3 and -2 respectively. This prior belief is summarized as follows.

$$\mu_{\beta_0} = 2$$

$$\sigma^2_{\beta_0} = 1$$

$$\beta_0 \sim N(\mu_{\beta_0}, \sigma^2_{\beta_0})$$

$$\beta_3 = -2$$

Given the above distribution, which describes our prior belief, we can generate simulated data using a so-called **generative model**, as depicted in the image below.

$$Y_{ij} \sim \text{Bernoulli}(p_{ij})$$

$$\text{logit } p_{ij} = \beta_{0j} + \beta_1 \times \text{urban} + \beta_2 \times \text{living.children} + \beta_3 \times \text{age-mean},$$

where Y_{ij} is 1 if woman i in district j uses contraceptives, and 0 otherwise, and where $i \in \{1, \dots, N\}$ and $j \in \{1, \dots, J\}$. N is the number of observations in the data, and J is the number of districts.

In this case, the generative model is a Bernoulli distribution with parameters p, which is the probability of a woman using contraceptives. p is the logistic function introduced in the previous section.

In order to get a better grasp of the concept of generative model, let's simulate binary response data Y. We do this by using prior parameter values and data.

```
beta_0_mu = 2
beta_0_sigma = 1
sample_size = df_contraceptives.shape[0]
beta_0 = np.random.normal(beta_0_mu, beta_0_sigma, sample_size)
beta_1 = 4
beta_2 = -3
beta_3 = -2

df_contraceptives_simulated = df_contraceptives.copy()
df_contraceptives_simulated['contraceptive_use_sim'] = expit(beta_0 +
                                                            beta_1*df_contraceptives_sim
                                                            beta_2*df_contraceptives_sim
                                                            beta_3*df_contraceptives_sim
df_contraceptives_simulated['contraceptive_use_sim'] = (df_contraceptives_simulated['cor
df_contraceptives_simulated.head()
```

bayes_logreg_simulated.py hosted with ❤ by GitHub view raw

district	urban	living.children	age_mean	contraceptive_use	contraceptive_use_sim

29	0	2	-8.5599	1	1
5	0	3	-4.5599	1	1
34	1	4	8.4400	0	0

Does contraceptives usage vary by district?

In order to experiment with the Bayesian approach a bit more, we will now specify a varying-intercept logistic regression model, where the intercept varies by district, and we will fit it to the simulated contraceptives data.

We set informative prior distributions on *β0, β1, β2* and *β3,* which are explicitly different from the ones we used to generate the simulated contraceptives data. Our goal is to estimate the true parameter values, which were used to simulate the response variable *Y*.

Prior Distribution:

$\beta_{0j} \sim N(\mu_0, \sigma_0^2)$, with $\mu_0 \sim N(0, 10000)$ and $\frac{1}{\sigma_0^2} \sim \text{Gamma}(0.1, 0.1)$.

$\beta_1 \sim N(0, 10000)$, $\beta_2 \sim N(0, 10000)$, $\beta_3 \sim N(0, 10000)$

Generative Model for data:

$$Y_{ij} \sim \text{Bernoulli}(p_{ij})$$

$$\text{logit } p_{ij} = \beta_{0j} + \beta_1 \times \text{urban} + \beta_2 \times \text{living.children} + \beta_3 \times \text{age-mean},$$

where Y_{ij} is 1 if woman i in district j uses contraceptives, and 0 otherwise, and where $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, J\}$. N is the number of observations in the data, and J is the number of districts. The above notation assumes $N(\mu, \sigma^2)$ is a Normal distribution with mean μ and variance σ^2.

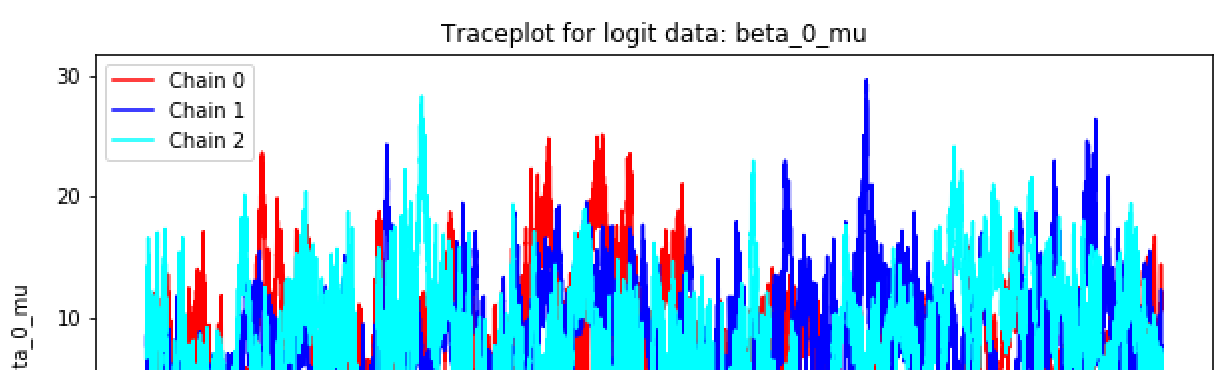
When we have data, priors and a generative model, we can apply Bayes theorem to compute the **posterior probability distribution** of the model parameters conditionally upon the predictors (*district, urban, living.children, age-mean*) and response (*Y*).

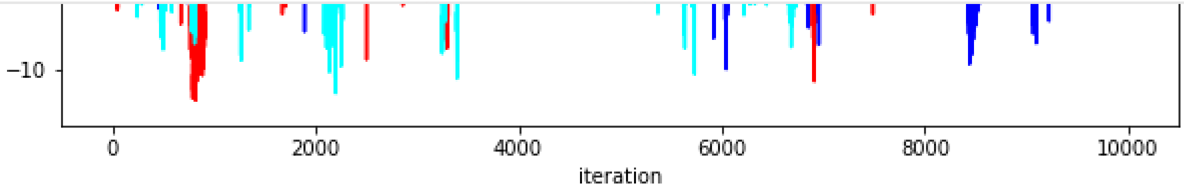
Markov chain Monte Carlo (*MCMC*) is a popular class of algorithms used to find the posterior distribution of the model parameters. Running such algorithms in Python is chains to guide the process towards the simulated data we have at hand. You can see how the MCMC works in the code below.

```
import pyjags

# specs of varying-intercept model in PyJags syntax
logreg_code = '''
model {
    for (i in 1:N){
        Y[i] ~ dbern(p[i])
        p[i] = ilogit(beta_0[district[i]] + beta_1*urban[i] + beta_2*children[i] + beta_
    }
    for (j in 1:J){
        beta_0[j] ~ dnorm(beta_0_mu, beta_0_sigma)
    }
    beta_0_mu ~ dnorm(0, 1/10000)
    beta_0_sigma ~ dgamma(0.1, 1/pow(0.1,2))
    beta_1 ~ dnorm(0, 1/10000)
    beta_2 ~ dnorm(0, 1/10000)
    beta_3 ~ dnorm(0, 1/10000)
}
'''

# number of Markov chains
nb_chains = 3

# builder for PyJags model
def get_pyjags_model(Y):
    logreg_data = dict(district=df_contraceptives_simulated['district'],
                      urban=df_contraceptives_simulated['urban'],
                      children=df_contraceptives_simulated['living.children'],
                      age=df_contraceptives_simulated['age_mean'],
                      Y=Y,
                      N=sample_size,
                      J=nb_districts)
    logreg_init = dict( beta_0_mu=beta_0_mu,
                        beta_0_sigma=beta_0_sigma,
                        beta_1=beta_1,
                        beta_2=beta_2,
                        beta_3=beta_3
                        )
```

```
# varying-intercept PyJags model
logreg_model = get_pyjags_model(Y=df_contraceptives_simulated['contraceptive_use_sim'])

# sampler function
def get_pyjags_samples(logreg_model=logreg_model, n_burnin=1000, n_samples=10000):
    logreg_burnin = logreg_model.sample(n_burnin)
    logreg_samples = logreg_model.sample(n_samples)
    return logreg_burnin, logreg_samples

# sampling from posterior distribution
logreg_burnin, logreg_samples = get_pyjags_samples()

# list of parameters
logreg_parameters = ['beta_0_mu','beta_0_sigma','beta_1','beta_2','beta_3']

# function for plotting the posterior distributions trace
def trace_plot(logreg_samples=logreg_samples, n_samples=10000):
    colors = ['red', 'blue', 'cyan']
    fig, ax = plt.subplots(len(logreg_parameters), 1, figsize=(10,30))
    for i, param in enumerate(logreg_parameters):
        for j in range(nb_chains):
            ax[i].plot(range(n_samples),
                           logreg_samples[param][0,:,j],color=colors[j],label='Chain {}'.for
                          )
            ax[i].set_xlabel('iteration')
            ax[i].set_ylabel(param)
            ax[i].set_title('Traceplot for logit data: {}'.format(param))
            ax[i].legend()

trace_plot()
```

bayes_logreg_pyjags1.py hosted with ❤ by GitHub view raw





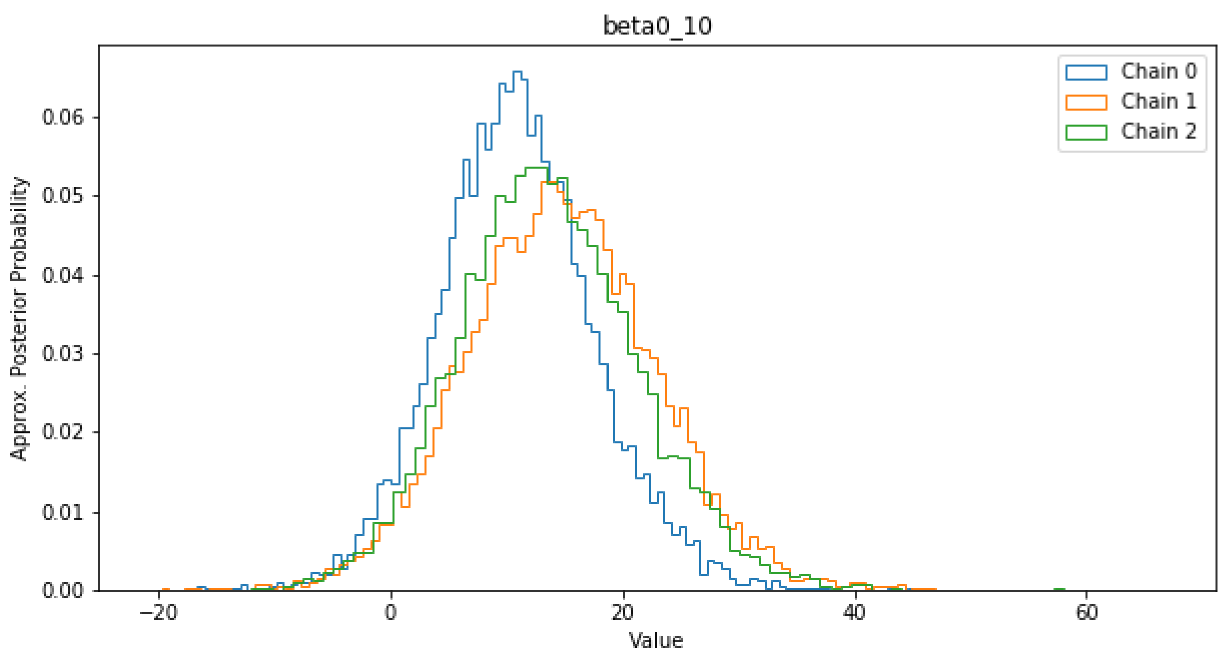
A good practice is to plot the trace plots of the MCMC sampler for the parameters. The trace plot above suggests that the sampler did converge. The average of each of the 3 Markov chains looks roughly the same. Although there is a little wandering within the chain, there is no evidence of divergent chains.

Now, we can plot the histograms of the posterior distributions for the parameters. We only show the plot for *β0* and for district10 only, as illustrative example.

```
# function to extract posterior distributions
def get_pyjags_dataframe(logreg_samples=logreg_samples,
                         logreg_parameters=logreg_parameters,
                         nb_chains=nb_chains):
    df_samples = []
    for chain in range(nb_chains):
        df_chain = []
        for param in logreg_parameters:
            samples = logreg_samples[param]
            chain_data = samples[:,:,chain]
            df = pd.DataFrame(chain_data.T)
            if df.shape[1]==1:
                df = df.rename({0:param}, axis=1)
            else:
                df = df.add_prefix(param)
            df_chain.append(df)

        df_samples.append(pd.concat(df_chain, axis=1))

    return df_samples

# get posterior distributions of beta0 for all districts
beta_0_samples = get_pyjags_dataframe(logreg_samples,['beta_0'],nb_chains)

# plot posterior distributions of beta0 for selected districts
selected_districts = [10,20,30,40,50,60]
```

```
    col_idx = int(col[-2:])+1
    if col_idx in selected_districts:
        for chain in range(nb_chains):
            ax[i].hist(beta_0_samples[chain][col],
                       bins=100, histtype='step', density=True, label="Chain {}".format(
            ax[i].set_xlabel("Value")
            ax[i].set_ylabel("Approx. Posterior Probability")
            ax[i].set_title('beta0_{}'.format(col_idx))
            ax[i].legend()
        i = i + 1
```

bayes_logreg_pyjags_posteriors1.py hosted with ❤ by GitHub view raw



We recall that the true distribution for *β0* that was used to generate simulated data was as follows.

$$\beta_0 \sim N(2, 1)$$

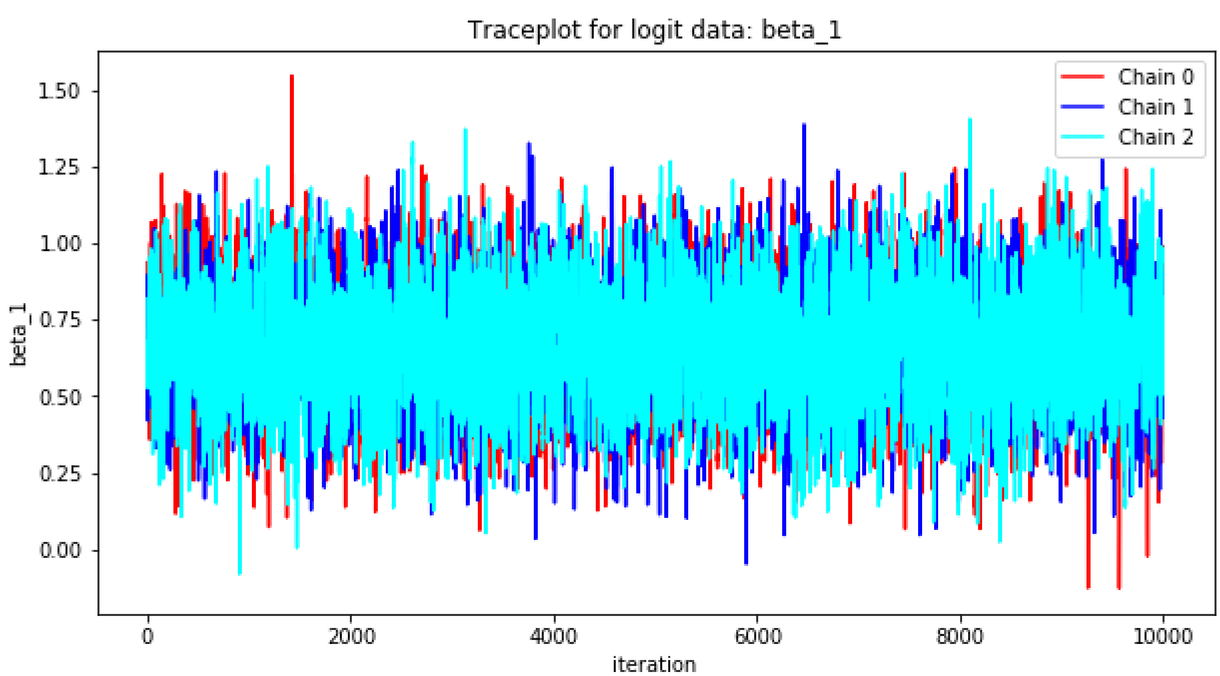
As you can see in the plot above, the true *β0* parameter for district 10 is contained within the posterior distributions from our model. In fact, this was also the case for all remaining parameters, not shown here. This finding suggests that the Bayesian

Women belonging to which district are most likely to use contraceptives?

Below, we run the MCMC sampler once more, this time using training data. We check the convergence by examining the trace plots, as we did with the simulated data. It appears that the samplers also converged here. None of the chains appears to be divergent, because they look uncorrelated, independently random sampled.

```
# create PyJAGS model
logreg_model_train = get_pyjags_model(Y=df_contraceptives_simulated['contraceptive_use'])

# get samples
logreg_train_burnin, logreg_train_samples = get_pyjags_samples(logreg_model=logreg_model_

# plot trace
trace_plot(logreg_samples=logreg_train_samples)
```

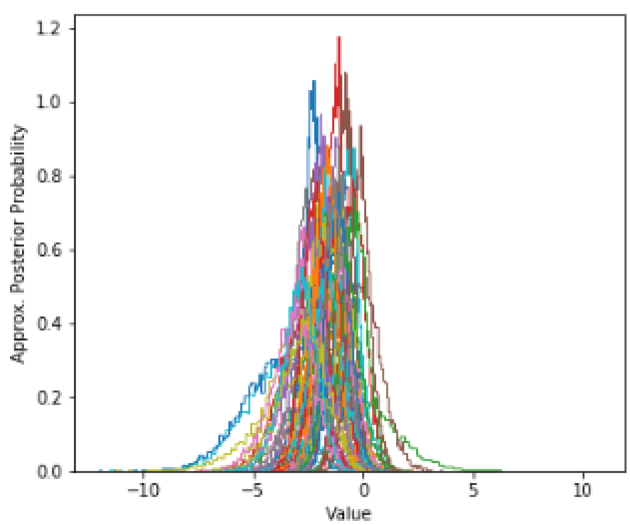
bayes_logreg_pyjags2.py hosted with ❤ by GitHub view raw



The posterior distribution of model parameters *β0* can now be plotted for all districts as follows.

```python
                                               nb_chains=nb_chains)

# plot posterior distributions of beta0 for all districts
beta_0_avg = []
beta_0_std = []
fig, ax = plt.subplots(1, nb_chains, figsize=(20, 5))
for chain in range(nb_chains):
    col_avg = []
    col_std = []
    for col in beta_0_samples_train[0].columns:
        col_avg.append(np.mean(beta_0_samples_train[chain][col]))
        col_std.append(np.std(beta_0_samples_train[chain][col]))
        ax[chain].hist(beta_0_samples_train[chain][col],
                    bins=100, histtype='step', density=True, label=col)
        ax[chain].set_xlabel("Value")
        ax[chain].set_ylabel("Approx. Posterior Probability")
        ax[chain].set_title('Chain {}'.format(chain))
    beta_0_avg.append(col_avg)
    beta_0_std.append(col_std)
```

bayes_logreg_pyjags_posteriors2.py hosted with ❤ by GitHub view raw

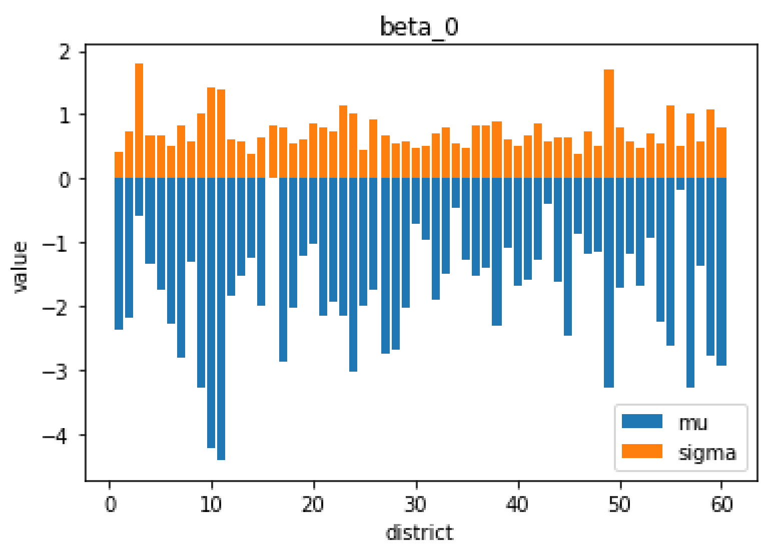


We know that positive values of *β0* are associated with increased probability that women belonging to the corresponding districts are most likely to use contraceptives. Looking at the plot above, only a few districts have positive *β0* values.

group.

```
beta_0_avg_arr = np.mean(np.array(beta_0_avg), axis=0)
beta_0_std_arr = np.mean(np.array(beta_0_std), axis=0)
print('Women belonging to district {} are most likely to use contraceptives.'.format(np.
print('Women belonging to district {} are most likely NOT to use contraceptives.'.format

plt.bar(range(1,61),beta_0_avg_arr, label='mu')
plt.bar(range(1,61),beta_0_std_arr, label='sigma')
plt.xlabel('district')
plt.ylabel('value')
plt.title('beta_0')
plt.legend();
```

bayes_logreg_pyjags_evaluation1.py hosted with ❤ by GitHub view raw



The posterior distributions allow us to compute the mean and standard deviation of *β0* values per district as illustrated in the plot above. Based on these results, we can conclude that the *β0* posteriors are not distributed uniformly across districts. This evidence is in support of the varying-intercept model and lead to the following findings:

> *Women belonging to district 16 are most likely to use contraceptives.*
>
> *Women belonging to district 11 are most likely NOT to use contraceptives.*

Conclusion

When combined with prior beliefs, we were able to quantify uncertainty around point estimates of contraceptives usage per district. In this article, we also offered few take-out on PyJAGS, an easy to use Python library for Bayesian inference.

This area opens a wide door for future work, especially because Bayesian statistical analysis is at the core of several technologies ranging from medical diagnosis to election forecasting. The *Signal and the Noise* 2012's book by Nate Silver is an example of master piece in the art of using probability and statistics as applied to real-world circumstances.

Thanks to Anne Bonner.

Machine Learning Data Science Artificial Intelligence Statistics Towards Data Science