Assessing the Safety of Rosiglitazone for the Treatment of Type 2 Diabetes

Konstantinos Vamvourellis

March 14, 2018

1 Python 3 Environment

You can find instructions for installing PyStan over here.

```
In [1]: import pystan

        print(pystan.__version__)
```

```
2.17.1.0
```

```
In [2]: import pickle

        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import seaborn as sns

        from numpy.linalg import inv, cholesky
        from numpy.random import uniform
        from scipy.special import expit
        from scipy.stats import multivariate_normal, bernoulli
```

```
In [3]: # Plot Styling
        %matplotlib inline
        sns.set()
        sns.set_style("darkgrid")
        color = 'salmon'
        alpha = .7
        bins = 100
        lw = 2
        linestyle = '-'
        linecolor = 'r'
        linealpha = .8
```

2 Introduction

Rosiglitazone was authorized to enter the market for the treatment of type 2 diabetes in the United States in 1999 and in the European Union in 2000. New data subsequently emerged about possible cardiovascular risks associated with Rosiglitazone, confirmed by a meta-analysis in Nissen and Wolski (2007), which resulted in a European suspension of the Marketing Authorisation in 2010. This suspension included its use as a fixed-dose combination with metformin or glimepiride for type 2 diabetes, which had been approved in 2003 for metformin and 2006 for glimepiride. The drug remained available in the United States, but only under a restricted-access program implemented in 2010.

2.1 When to approve a drug?

Regulators focus on a few key factors when deciding whether a drug is fit to enter the market. In the case of Rosiglitazone, for example, previous work (Phillips et al. 2013). concentrated on 11 of the drug's effects, weighing positive effects against negative effects. Clinical trials data are presented to experts and clinicians to assess the safety of the drug. The clinical analysis of trial data is based on statistical summaries of the data, including averages, standard deviations, and signifiance levels. However, dependencies between the effects are the subject of clinical judgment and are rarely included in the statistical summaries.

In this study, we address these issues by building a Bayesian model to do a full benefit-risk balance analysis of the dataset at the individual patient's level. Specifically, we construct a latent variable model to account for the whole joint distribution of the effects. This model will allow us to simulate the effect of the drug on a new patient conditioned on all the observations in the clinical trials. By quantifying the uncertainty of the effects of a drug treatment, given the available clinical trial datasets, our approach can inform whether regulators should approve a drug or not. This is done by combining clinical judgement as well samples for the model posterior using multi-criteria decision analysis (MCDA) via a Bayesian decision-theoretic framework, discussed in more detail in the Application section.

2.2 How to model the dependence between discrete and continuous observations?

It is common in clinical trials to collect "yes/no" data. We want to fully model this process of inter-related dependencies, incorporate the dependence between the different measurements for each person, and account for uncertainty. Furthermore, datapoints collected in clinical trials are routinely of mixed type: binary, continuous, counts, etc. The main purpose of this work is to extend the current framework so that it can incorporate interdependencies between different features, both discrete and continuous.

Our data is organized with one subject per row and one effect per column. For example, if our clinical trial dataset records 3 effects per subject, 'Hemoglobin Levels' (continuous), 'Nausea' (yes/no) and 'Dyspepsia' (yes/no) the dataset would look like this:

Subject ID	Group Type	Hemoglobin Level	Dyspepsia	Nausea
123	Control	3.42	1	0
213	Treatment	4.41	1	0
431	Control	1.12	0	0
224	Control	-0.11	1	0
224	Treatment	2.42	1	1

Subject ID	Group Type	Hemoglobin Level	Dyspepsia	Nausea

To model the effects of a drug we need a generative model for these 3 effects that also allows for dependencies between these effects. It stands to reason that the probability of a subject experiencing Nausea is not independent of the probability of experiencing Dyspepsia. To this end, we adopt a parametric generative model to learn the covariance matrix directly.

We denote the observed data by y and the parameters of the model by θ. We are then interested in the posterior distribution $\pi(\theta|y)$, with which we can draw samples from the distribution of effects $f(y'|y)$ on future, so far unseen, patients y' conditional on observations y as follows:

$$f(y'|y) = \int f(y'|y,\theta)\pi(\theta|y)d\theta$$

In practice we cannot analytically derive the full posterior $\pi(\theta|y)$, but we can get samples from it using *Stan*. Consequently, we can approximate the expectation of any function $h(y')$ on the future data

$$\mathbb{E}(h(y')|y) = \int\int h(y')f(y'|y,\theta)\pi(\theta|y)d\theta dy' = \int h(y')f(y'|y)dy'$$

using Monte Carlo.

We assume that each subject is independently and identically distributed within its group. We run inference for each group separately and get two sets of parameters, one for the treatment group and one for the placebo, also known as control group. Using the samples from the predictive distribution of each group we can produce samples for the difference between the two groups. Generally, with these posterior samples we can compute any value of interest currently used to decide whether to approve the drug. As an application, we will later show what such an evaluation function looks like and work through a complete example (see Application section).

3 Methodology

Let Y be a $N \times K$ matrix where each column represents an effect and each row refers to an individual subject. This is our observations, our clinical trials dataset. In order to distinguish the treatment from placebo subjects, we will analyse the data for each group Y^T and Y^C separately. As the model for Y^T and Y^C is identical, we suppress the notation into Y in the remainder of this section for convenience. Recall that the important feature of the data is that each column in Y may be measured on different scales, i.e. binary, count or continuous etc. The main purpose of this work is to extend the current framework so that it can incorporate interdependencies between different features, both discrete and continuous.

We consider the following general latent variable framework. The idea to assign appropriate distributions on each column and apply appropriate transformations on their parameters via user specified link functions $h_j(\cdot)$, so that everything is brought on the same scale. For example, let's fix our attention on the i-th subject for a moment. Then if the j-th effect is measured in the binary scale, the model can be

$$\begin{cases} Y_{ij} \sim \text{Bernoulli}(\eta_j),\ i = 1,\dots,N,\ Y_{ij} \text{ independent, for fixed } j \\ h_j(\eta_j) = \mu_j + Z_{ij}, \end{cases} \tag{1}$$

where the link function can be the logit, probit or any other bijection from $[0,1]$ to the real line. Similarly, for count data on the j-th column we can adopt the following model

$$\begin{cases} Y_{ij} \sim \text{Poisson}(\eta_j),\ i = 1, \ldots, N,\ Y_{ij} \text{ independent, for fixed } j \\ h_j(\eta_j) = \mu_j + Z_{ij}. \end{cases} \tag{2}$$

where $h_j(\cdot)$ could be the natural logarithm, whereas for continuous data one can simply write

$$Y_{ij} = \mu_j + Z_{ij},\ i = 1, \ldots, N. \tag{3}$$

In order to complete the model we need to define the $N \times K$ matrix Z. Here we use a K-variate Normal distribution $\mathcal{N}_K(\cdot)$ on each $Z_{i:}$ row, such that

$$Z_{i\cdot} \sim \mathcal{N}_K(0_K, \Sigma), \tag{4}$$

where Σ is a $K \times K$ covariance matrix, O_K is a row K-dimensional vector with zeros and $Z_{i\cdot}$ are independent for all i. Of course other options are available, e.g. a multivariate t.

In the model above, vector $\mu = (\mu_1, \ldots, \mu_K)$ represents quantities related with the mean of each effect, whereas matrix Σ models their covariance. Note that the variance of binary variables is non identifiable (Chib and Greenberg (1998), Talhouk, Doucet, and Murphy (2012)), so we focus on the correlation matrix instead.

4 Stan Code

For this case study we use the Bernoulli likelihood for the binary data with a logit link function. The Stan program encoding this model is the following:

```
In [4]: with open('./modelcode.stan', 'r') as file:
            print(file.read())
```

```
data{
 int<lower=0> N;
 int<lower=0> K;
 int<lower=0> Kb;
 int<lower=0> Kc;
 int<lower=0, upper=1> yb[N,Kb];
 vector[Kc] yc[N];
}

transformed data {
  matrix[Kc, Kc] I = diag_matrix(rep_vector(1, Kc));
}
parameters {
  vector[Kb] zb[N];
  cholesky_factor_corr[K] L_R;  // first continuous, then binary
  vector<lower=0>[Kc] sigma;
  vector[K] mu;
}

transformed parameters{
  matrix[N, Kb] z;
```

```
  vector[Kc] mu_c = head(mu, Kc);
  vector[Kb] mu_b = tail(mu, Kb);
  {
    matrix[Kc, Kc] L_inv = mdivide_left_tri_low(diag_pre_multiply(sigma, L_R[1:Kc, 1:Kc]), I);
    for (n in 1:N){
      vector[Kc] resid = L_inv * (yc[n] - mu_c);
      z[n,] = transpose(mu_b + tail(L_R * append_row(resid, zb[n]), Kb));
    }
  }
}

model{
  mu ~ normal(0,10);
  L_R ~ lkj_corr_cholesky(2);
  sigma~cauchy(0,2.5);
  yc ~ multi_normal_cholesky(mu_c, diag_pre_multiply(sigma, L_R[1:Kc, 1:Kc]));
  for (n in 1:N) zb[n] ~ normal(0,1);
  for (k in 1:Kb) yb[,k] ~ bernoulli_logit(z[,k]);

}

generated quantities{
  matrix[K,K] R = multiply_lower_tri_self_transpose(L_R);
  vector[K] full_sigma = append_row(sigma,rep_vector(1, Kb));
  matrix[K,K] Sigma = multiply_lower_tri_self_transpose(diag_pre_multiply(full_sigma,L_R));

}
```

```
In [5]: with open('./modelcode.stan', 'r') as file:
            model_code = file.read()
```

We will fit the model with synthetic data that we generate as follows:

```
In [6]: def C_to_R(M):
            """
            Send a covariance matrix M to the corresponding
            correlation matrix R
            Inputs
            ============
            - M : covariance matrix
            Output
            ============
            - correlation matrix
            """

            d = np.asarray(M.diagonal())
```

```
    d2 = np.diag(d**(-.5))
    R = np.dot(np.dot(d2, M), d2)
    return R


def gen_data(N=400, Kb=1, Kc=2, random_seed=234234999):
    """
    Generate data of dimension [N, Kb+Kc]
    Inputs
    ============
    - N : number of rows/subjects
    - Kb : number of binary effects
    - Kc : number of conitnuous effects
    Output
    ============
    - correlation matrix
    """
    np.random.seed(random_seed)

    N1 = N // 2  # size of Group1
    N2 = N - N1  # size of Group2
    K = Kb + Kc  # total number of effects

    A = np.tril(uniform(-1, 1, size=(K, K)))
    C = np.dot(A, A.T)
    R = C_to_R(C)

    # Choose variances
    sigma = uniform(1, 4, size=Kc)

    # Construct the covariance matrix
    D = np.diag(np.concatenate((sigma, np.ones(Kb))))
    Sigma =  D.dot(R).dot(D)

    # Means for Group 1
    mu_c = uniform(-10, 10, size=Kc)
    mu_b = uniform(-1, 1, size=Kb)
    mu1 = np.concatenate([mu_c, mu_b], axis=0)

    # Means for Group 2
    mu2 = mu1.copy()
    mu2[:Kc] = mu2[:Kc] + uniform(-5, -3, size=Kc)
    mu2[Kc:] = mu2[Kc:] + uniform(-.5, -2, size=Kb)


    # group1
    z = multivariate_normal.rvs(mean=mu1, cov=Sigma, size=N1)
    y1 = np.empty(z.shape)
```

```
        y1[:, Kc:] = bernoulli.rvs(p=expit(z[:, Kc:]))
        y1[:, :Kc] = z[:, :Kc]

        # group2
        z = multivariate_normal.rvs(mean=mu2, cov=Sigma, size=N2)
        y2 = np.empty(z.shape)
        y2[:, :Kc] = z[:, :Kc]
        y2[:, Kc:] = bernoulli.rvs(p=expit(z[:, :Kb]))


        return y1, y2, mu1, mu2, R, sigma, N1, N2, Kb, Kc
```

It's good practice to save the data and posterior samples of the model fit to it. In our case, we will need it again when we demonstrate how to fit the model on the real datasets. For the purposes of this notebook, we will re-use this data in place of the hypothetical control group dataset. We can save the data and the samples as follows:

```
In [7]: y_control, y_treat, mu_control, mu_treat, R, sigma, N_control, N_treat, Kb, Kc = gen_dat

In [8]: data_control = dict(N=N_control, Kb=Kb, Kc=Kc, K=Kb+Kc, yb=y_control[:, Kc:].astype(int)
        pickle.dump(data_control, open("data_control.pkl", "wb"))

        data_treat = dict(N=N_treat, Kb=Kb, Kc=Kc, K=Kb+Kc, yb=y_treat[:, Kc:].astype(int),yc=y_
        pickle.dump(data_treat, open("data_treat.pkl", "wb"))
```

We compile the model with the following code.

```
In [9]: sm = pystan.StanModel(model_code=model_code, verbose=True)

INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_347c6f238e43a7a07dd740d8cd0f121c NOW.
INFO:pystan:OS: darwin, Python: 3.6.4 | packaged by conda-forge | (default, Dec 23 2017, 16:54:0
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)], Cython 0.27.3


Compiling /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/stanfit4anon_model_347c6f
[1/1] Cythonizing /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/stanfit4anon_mode
building 'stanfit4anon_model_347c6f238e43a7a07dd740d8cd0f121c_5806305159344542127' extension
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var/folders
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var/folders/9j
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var/folders/9j/qb9gcwnj2ln
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var/folders/9j/qb9gcwnj2ln
creating /var/folders/9j/qb9gcwnj2lnb77886qjkmnfm0000gn/T/tmpolcedxb5/var/folders/9j/qb9gcwnj2ln
gcc -Wno-unused-result -Wsign-compare -Wunreachable-code -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-
g++ -bundle -undefined dynamic_lookup -Wl,-rpath,/Users/itemgmt/anaconda2/envs/rosi_py/lib -L/Us
```

We fit the model with the following code

```
In [10]: fit = sm.sampling(data=data_control, iter=1000, chains=4)
```

```
/Users/itemgmt/anaconda2/envs/rosi_py/lib/python3.6/site-packages/pystan/misc.py:399: FutureWarn
  elif np.issubdtype(np.asarray(v).dtype, float):
```

We save the extracted samples to be used later again

```
In [11]: post_samples = fit.extract(permuted=True)  # return a dictionary of arrays
         pickle.dump(post_samples, open("fit_control.pkl", "wb"))
```

To avoid waiting we can load the pre-fit result as follows:

```
In [12]: post_samples = pickle.load(open("fit_control.pkl", "rb"))
```

4.1 Model Diagnostics

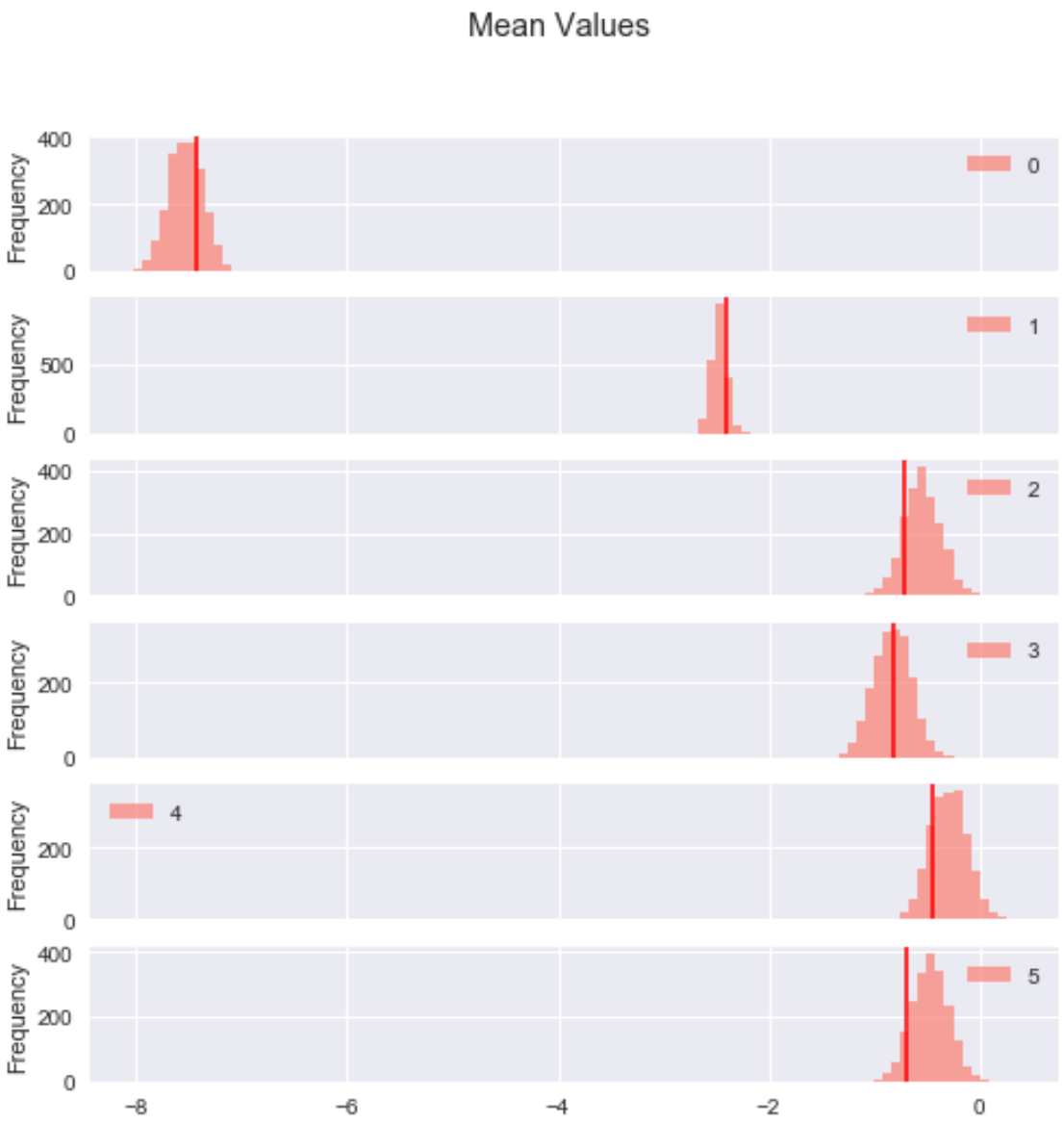
We see that max `Rhat` values are good, below 1.01. The effective sample size `n_eff` is good and the rest of the diagnostics are clean. Below we plot histograms of posterior samples for the mean, correlations and variance of the effects against the true values.

```
In [13]: print(pystan.misc.stansummary(fit, pars=['mu', 'R', 'sigma']))
```

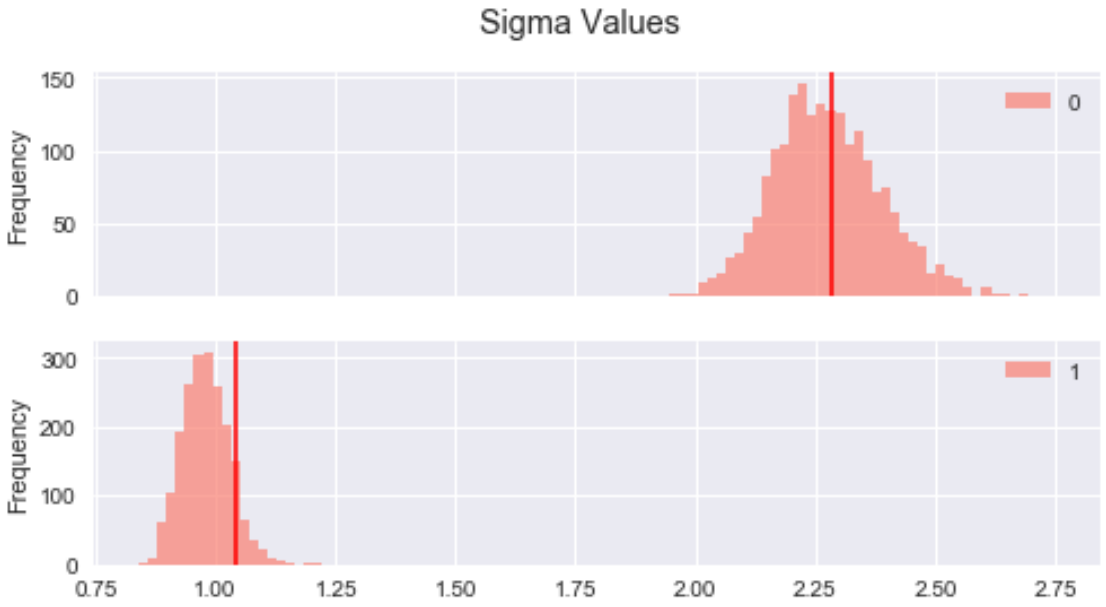
```
Inference for Stan model: anon_model_347c6f238e43a7a07dd740d8cd0f121c.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2000.

          mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu[0]    -7.53  3.5e-3   0.16  -7.84  -7.64  -7.53  -7.42  -7.23   2000    1.0
mu[1]    -2.48  1.6e-3   0.07  -2.62  -2.52  -2.48  -2.43  -2.34   2000    1.0
mu[2]    -0.55  3.8e-3   0.17  -0.89  -0.66  -0.54  -0.43  -0.22   2000    1.0
mu[3]    -0.83  4.0e-3   0.18  -1.17  -0.95  -0.82  -0.71  -0.49   2000    1.0
mu[4]     -0.3  3.8e-3   0.17  -0.63  -0.42   -0.3  -0.18   0.04   2000    1.0
mu[5]    -0.47  3.8e-3   0.17  -0.81  -0.59  -0.47  -0.36  -0.14   2000    1.0
R[0,0]     1.0     0.0    0.0    1.0    1.0    1.0    1.0    1.0   2000    nan
R[1,0]    0.17  1.4e-3   0.06   0.05   0.13   0.17   0.21    0.3   2000    1.0
R[2,0]    0.22  3.1e-3   0.14  -0.05   0.12   0.22   0.32    0.5   2000    1.0
R[3,0]    0.25  3.4e-3   0.15  -0.05   0.15   0.26   0.35   0.56   2000    1.0
R[4,0]    0.46  3.2e-3   0.14   0.17   0.37   0.46   0.56   0.72   2000    1.0
R[5,0]    0.08  3.6e-3   0.16  -0.24  -0.03   0.08   0.19   0.39   2000    1.0
R[0,1]    0.17  1.4e-3   0.06   0.05   0.13   0.17   0.21    0.3   2000    1.0
R[1,1]     1.0 2.1e-189.3e-17    1.0    1.0    1.0    1.0    1.0   2000    nan
R[2,1]   -0.64  2.7e-3   0.12  -0.84  -0.72  -0.64  -0.56  -0.39   2000    1.0
R[3,1]    0.44  3.0e-3   0.14   0.18   0.35   0.44   0.54   0.69   2000    1.0
R[4,1]    -0.1  3.5e-3   0.16   -0.4  -0.21   -0.1 6.3e-3   0.21   2000    1.0
R[5,1]   -0.41  3.3e-3   0.15  -0.69  -0.52  -0.41  -0.31  -0.11   2000    1.0
R[0,2]    0.22  3.1e-3   0.14  -0.05   0.12   0.22   0.32    0.5   2000    1.0
R[1,2]   -0.64  2.7e-3   0.12  -0.84  -0.72  -0.64  -0.56  -0.39   2000    1.0
R[2,2]     1.0 9.5e-194.2e-17    1.0    1.0    1.0    1.0    1.0   1934    nan
R[3,2]   -0.15  8.7e-3   0.23  -0.59  -0.32  -0.16   0.02   0.28    721    1.0
R[4,2]    0.29  8.3e-3   0.21  -0.14   0.14   0.29   0.44   0.68    667    1.0
```

```
R[5,2]     0.34    0.01    0.22  -0.14    0.2   0.35    0.5   0.74    468    1.0
R[0,3]     0.25  3.4e-3    0.15  -0.05   0.15   0.26   0.35   0.56   2000    1.0
R[1,3]     0.44  3.0e-3    0.14   0.18   0.35   0.44   0.54   0.69   2000    1.0
R[2,3]    -0.15  8.7e-3    0.23  -0.59  -0.32  -0.16   0.02   0.28    721    1.0
R[3,3]      1.0 2.0e-188.6e-17    1.0    1.0    1.0    1.0    1.0   1876    nan
R[4,3]     0.21  7.8e-3    0.23  -0.28   0.06   0.21   0.38   0.64    912    1.0
R[5,3]    -0.27  9.6e-3    0.25   -0.7  -0.45  -0.29   -0.1   0.22    656    1.0
R[0,4]     0.46  3.2e-3    0.14   0.17   0.37   0.46   0.56   0.72   2000    1.0
R[1,4]     -0.1  3.5e-3    0.16   -0.4  -0.21   -0.1 6.3e-3   0.21   2000    1.0
R[2,4]     0.29  8.3e-3    0.21  -0.14   0.14   0.29   0.44   0.68    667    1.0
R[3,4]     0.21  7.8e-3    0.23  -0.28   0.06   0.21   0.38   0.64    912    1.0
R[4,4]      1.0 1.7e-187.6e-17    1.0    1.0    1.0    1.0    1.0   2000    nan
R[5,4]     0.38  6.8e-3    0.22  -0.08   0.24   0.39   0.53   0.76    997    1.0
R[0,5]     0.08  3.6e-3    0.16  -0.24  -0.03   0.08   0.19   0.39   2000    1.0
R[1,5]    -0.41  3.3e-3    0.15  -0.69  -0.52  -0.41  -0.31  -0.11   2000    1.0
R[2,5]     0.34    0.01    0.22  -0.14    0.2   0.35    0.5   0.74    468    1.0
R[3,5]    -0.27  9.6e-3    0.25   -0.7  -0.45  -0.29   -0.1   0.22    656    1.0
R[4,5]     0.38  6.8e-3    0.22  -0.08   0.24   0.39   0.53   0.76    997    1.0
R[5,5]      1.0 2.3e-187.5e-17    1.0    1.0    1.0    1.0    1.0   1083    nan
sigma[0]   2.28  2.6e-3    0.11   2.07    2.2   2.27   2.35   2.52   2000    1.0
sigma[1]   0.98  1.1e-3    0.05   0.89   0.95   0.98   1.01   1.08   2000    1.0

Samples were drawn using NUTS at Tue Mar 13 15:46:32 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
In [14]: K = Kb + Kc
         title = "Mean Values"
         true_values = mu_control
         df = pd.DataFrame(post_samples['mu'])
         df.columns = [str(x) for x in range(K)]

         ax = df.plot.hist(subplots=True, title=title, color=color,
                           sharex=True, sharey =False,
                           alpha=alpha, bins=bins,
                           figsize=(8,8));

         for i in range(df.shape[1]):
             ax[i].axvline(true_values[i], color=linecolor, alpha=linealpha, linestyle=linestyle
```




```
In [15]: title = "Sigma Values"
         true_values = sigma
         df = pd.DataFrame(post_samples['sigma'])
         df.columns = [str(x) for x in range(Kc)]

         ax = df.plot.hist(subplots=True, title=title, color=color,
                           sharex=True, sharey =False,
                           alpha=alpha, bins=bins,
                           figsize=(8,4));

         for i in range(df.shape[1]):
             ax[i].axvline(true_values[i], color=linecolor, alpha=linealpha, linestyle=linestyle
```




```
In [16]: def flatten_corr(a):
             return a[np.triu_indices(a.shape[0], k=1)]


         title = "Correlation Values"
         true_values= R[np.triu_indices(R.shape[0], k=1)]


         Rs = post_samples['R']
         num_of_samples = Rs.shape[0]

         colnames = np.array([str(x) for x in range(K)])
         cnames = zip(colnames[np.triu_indices(colnames.shape[0], k=1)[0]],
                      colnames[np.triu_indices(colnames.shape[0], k=1)[1]])

         cnames = list(cnames)

         M = len(cnames)
         fRs = np.empty((num_of_samples, M))
         for i in range (num_of_samples):
             fRs[i, :] = flatten_corr(Rs[i, :, :])

         df = pd.DataFrame(fRs)
         df.columns = cnames


         ax = df.plot.hist(subplots=True, title=title, color=color,
```

```
                    sharex=True, sharey =False,
                    alpha=alpha, bins=bins,
                    figsize=(8,20));

for i in range(df.shape[1]):
    ax[i].axvline(true_values[i], color=linecolor, alpha=linealpha, linestyle=linestyle
```


Correlation Values
Frequency
(0, 1)
(0, 2)
(0, 3)
(0, 4)
(0, 5)
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(2, 3)
(2, 4)
(2, 5)
(3, 4)
(3, 5)
(4, 5)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

5 Application

In order to use the measurements in the clinical trial to make a final decision on market readiness, we need an evaluation function. Here we describe one such function that uses Multicriteria Decision Analysis (MCDA). The process of constructing such a function is somewhat involved because it requires collaboration of expertise from various fields. This section is adapted from Phillips et al. (2015).

5.1 How to compare disparate effects?

In order to meaningfully compare the various effects, such as glucose levels, weight loss, and cardiovascular arrest, we need to transform physical measurements to a scale appropriate to the task in hand, in this case, treating type II diabetes. For example, how are we to compare an average reduction of 5% in Hemoglobin levels against an average 3% increase in weight? We compare these effects and their relevant importance in a two-step process. First, we bring all measurements to a common denominator, called "preference score." Then, we ask clinicians to assign a weight to each effect based on its importance relative to the drug's treatment potential. Note that, by design, this mapping is subjective so that it can reflect the judgement of the clinicians. We leave it to the clinicians to decide whether a 1% reduction of Hemoglobin levels outweigh the damage of 3% increase in weight.

For example, one of effect for the assessment of Rosiglitazone is "Nausea". For step one, we need to start from a reasonable expected range for this effect in any given trial, let's say the percentage of subjects experiencing "Nausea" ranges from 0 to 10 in percentage units (%). The "preference score" is essentially a map from [0,10] onto [0,100] such that 0 corresponds to the least desirable measurement, 10 %, and 100 the most desirable one, 0%. With the two extremes fixed, the map can take any form in between. In this study will use linear maps for simplicity. Each effect gets it's own "preference score" map from its own range to [0,100]. This way we can track, on a common scale, how *clinically desirable* each observation is. In step two, each effect is given a weight w_j that corresponds to the importance of moving from "preference score" 0 to 100. Finally, we need to normalize the weights to ensure they sum to 1.

5.2 How to score a drug?

With this scoring system we can estimate the effect of a drug on a new patient. For a given measurement we simply take a weighted sum of the "preference scores". Specifically for the j-th effect, let's assume that $c_j(\cdot)$ is the "preference score" map. Also let $y_j^{(T)}$ be the measurement in the treatment group and $y_j^{(C)}$ be the corresponding measurement in the control group. We then get a final score of

$$s = \sum_j w_j \cdot \left(c_j\left(y_j^{(T)}\right) - c_j\left(y_j^{(C)}\right)\right)$$

Thus we can decided if the treatment is beneficial (when the sum is positive), or not (the sum is negative).

Since there is noise in the data, the final score is noisy too. With the Bayesian model suggested here, we can use our posterior samples to propagate the uncertainty to the final score. We do

this for each group separately and with the posterior samples we can estimate the probability of interest, $P(s^{(T)} > s^{(C)})$.

5.3 A worked out example

Here we will present a full example starting from the observational data to the final score. In this example we assess the safety of Rosiglitazone drug for type II diabetes by comparing the distribution of the final score for the treatment group (152 subjects) and the control group (150 subjects). The first two columns capture Hemoglobin and Glucose levels respectively, as deviations from the baseline recorded when the subjects entered treatment. The last four columns record four different events, Diarrhea, Nausea/Vomiting, Dyspepsia and Edema respectively. According to clinical judgment we we will assume that the clinical weights are (59.2,11.8, 8.9,17.8,1.8,0.5). This means that, to take the first two effects for example, a full swing from the least desirable observation to the most desirable one is judged to be $\frac{59.2}{11.8}$ more important for the first effect than for the second. The following two functions allow us to calculate the preference scores for a vector of measurements for these 6 effects. Note that for the binary data we consider the underlying probability of observing the effect.

```
In [17]: def pref_score(x, m1, m2, sign=-1):
             """
             map from the effect range [m1, m2] to
             the score range [0, 100]. Sign = 1
             indicates higher effect meaasurements are more
             desirable, otherwise the opposite.
             """
             m = 100./float((m2 - m1))
             b = m * m1
             if sign == 1:
                 return -m * m1 + m*x
             else:
                 return m * m2 - m*x


         def get_scores_perrow(x):
             """
             Return the preference score for a row of measurements
             for only 4 binary and 2 cont
             """
             res = np.empty(6)

             res[0] = pref_score(x[0], -6., 3., -1)
             res[1] = pref_score(x[1], -15., 7.5, -1)
             res[2] = pref_score(x[2], 0.1, .35, -1)
             res[3] = pref_score(x[3], 0.1, .25, -1)
             res[4] = pref_score(x[4], 0.1, .2, -1)
             res[5] = pref_score(x[5], 0., .15, -1)
             return res
```

For example, the measurement vector (-1.1,-2, .3,0.17, .1, .14) gets a preference score of (45.6,

42.2, 20, 53.3, 100, 6.7). The best possible measurement would correspond necessarily to a preference score of (100,100,100,100,100), while the worst measurement would correspond to (0,0,0,0,0).

```
In [18]: x = [-1.1, -2, .3, 0.17, .1, .14]
         get_scores_perrow(x)
```

```
Out[18]: array([ 45.55555556,  42.22222222,  20.        ,  53.33333333,
                 100.        ,   6.66666667])
```

The final preference score, for this measurement, is the sum of the preference scores weighted by the clinicians weights, which gives us a final score of 4505.778, as follows:

```
In [19]: def get_final_score_perrow(x, weights=None):
             """
             dot product of measurments and clinical
             weights.
             """
             scores = get_scores_perrow(x)
             if weights is None:
                 weights = np.array([59.2, 11.8 , 8.9, 17.8, 1.8, 0.5])
             else:
                 weights = weights.reshape(x.shape)
             return np.dot(scores, weights)
```

```
In [20]: get_final_score_perrow(x)
```

```
Out[20]: 4505.777777777777
```

We are interested in the posterior distribution of the final score for a new subject in each of the groups. This way we can calculate the posterior distribution of the difference between the two groups. We do that by sampling one latent variable vector Z for each posterior sample of μ, R, σ. For each Z we calculate a final score, which becomes a posterior sample for the final score.

The function to calculate the posterior samples for the two groups is the following:

```
In [21]: def final_score(post_samples, Kc=2, Kb=4, weights=None):
             """
             get final score for posterior samples
             """
             K = Kb + Kc
             N_iter = post_samples['mu'].shape[0]

             mus = post_samples['mu']
             sigmas = post_samples['sigma']
             Rs = post_samples['R']

             if weights is None:
                 ws = np.array([59.2, 11.88, .9, 17.8, 1.8, 0.6])

             final_score = np.empty(N_iter)
```

```
        for i in range(N_iter):

            D = np.diag(np.concatenate((sigmas[i, :], np.ones(Kb))))
            Sigma = np.diag(D.dot(R).dot(D))
            z = multivariate_normal.rvs(mean=mus[i, :], cov=Sigma)
            final_score[i] = get_final_score_perrow(z, ws)
        return final_score
```

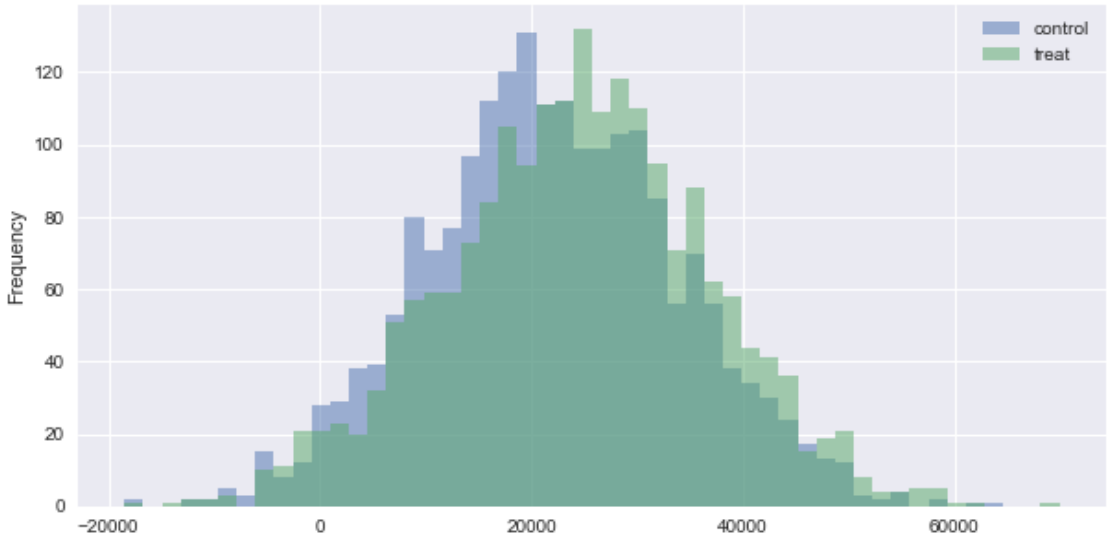
We can draw posterior samples from fitting our model to the treatment group data as follows

```
In [22]: data_list = dict(N=N_treat, Kb=Kb, Kc=Kc, K=Kb+Kc, yb=y_control[:, Kc:].astype(int), yc
         pickle.dump(data_list, open("data_treat.pkl", "wb"))
         fit = sm.sampling(data=data_list, iter=1000, chains=4)
         post_samples = fit.extract(permuted=True)  # return a dictionary of arrays
         pickle.dump(post_samples, open("fit_treat.pkl", "wb"))  # save samples for later use
```

```
/Users/itemgmt/anaconda2/envs/rosi_py/lib/python3.6/site-packages/pystan/misc.py:399: FutureWarn
  elif np.issubdtype(np.asarray(v).dtype, float):
```

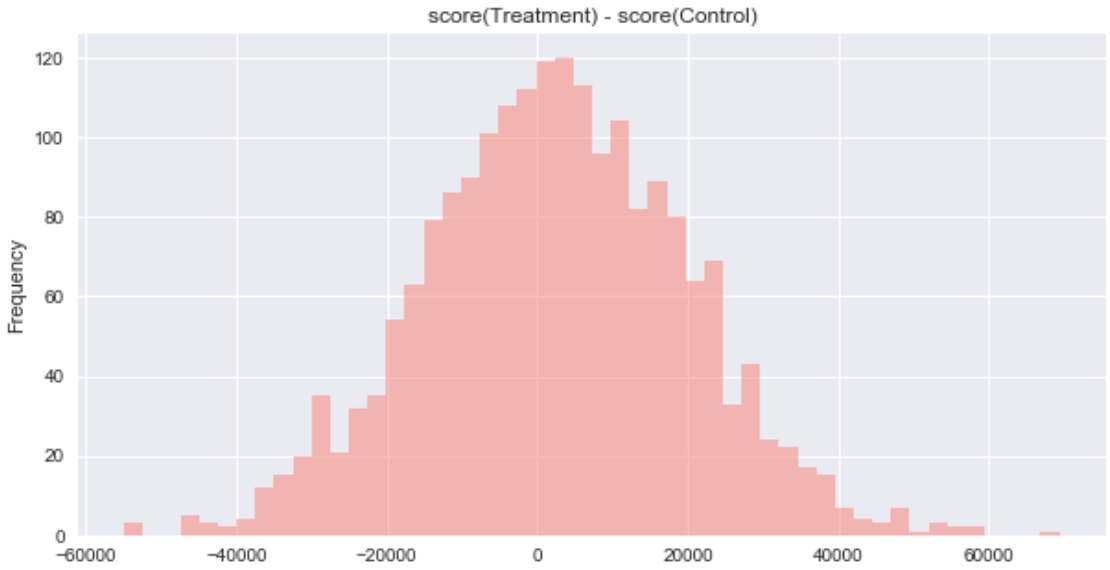
We can evaluate the above function for each group separately. The value we are interested in is the difference of the final score between the treatment and the control group. We have pre-fit our models to the two groups and saved the posterior samples. We load the samples and compute the values of interest as follows:

```
In [23]: scores = pd.DataFrame({"control": final_score(pickle.load(open("fit_control.pkl", "rb")
                                "treat": final_score(pickle.load(open("fit_treat.pkl", "rb")))})
         scores.plot.hist(figsize=(10, 5), alpha=.5, bins=50);
```



We observe that under this model, the final score is above 0 with probability 56% for a new patient. We can also make a plot of the difference to visualize the final result.

```
In [24]: scores['diff'] = scores.treat - scores.control
         float((scores['diff'] > 0).sum()) / len(scores)

Out[24]: 0.5575

In [25]: pd.DataFrame(scores['diff']).plot.hist(figsize=(10, 5), title='score(Treatment) - score
                                                alpha = .5, bins=50, color='salmon',
                                                legend=False);
```



5.3.1 Discussion, Future Work

One natural question is how sensitive the final result is to the choice of preference score functions. The preference scores could in principle have any form. In this study we chose linear mappings because they are easy to interpret and to work with. Phillips et al. (2015) looked at non-linear preference functions, guided by clinical experts who suggested that "desirability" of some effects is better modeled with sigmoid-like functions. Their study found "that model results are very robust to imprecision and disagreements about weights. Even non-linear value functions on the most discriminating effects did not tip this balance [sign of score difference between treatment and control]".

Another source of variability we examined is the choice of clinical weights. Based on preliminary experimentation we conclude that the final probability seems relatively stable to changes in the weights. When we perturbed the weights by 10%, we observed a difference in the final score distribution that was close to 5%.

Acknowledgments

The author would like to thank Jonah Gabry, Bob Carpenter, Andrew Gelman, and Ben Goodrich (who practically wrote the Stan code) for their feedback and help during the process of writing this report.

License

6 References

- Chib, S., and E. Greenberg. 1998. "Analysis of Multivariate Probit Models." Biometrika 85 (2).
- Nissen, S. E., and K. Wolski. 2007. "Effect of Rosiglitazone on the Risk of Myocardial Infarction and Death from Cardiovascular Causes." New England Journal of Medicine 356: 2457–71.
- Phillips, Lawrence, Billy Amzal, Alex Asiimwe, Edmond Chan, Chen Chen, Diana Hughes, Juhaeri Juhaeri, et al. 2015. "Wave 2 Case Study Report: Rosiglitazone."
- Phillips, Lawrence, and others. 2013. "IMI Work Package 5: Report 2:b:ii Benefit - Risk Wave 2 Case Study Report: Rosiglitazone."
- Talhouk, A., A. Doucet, and K. Murphy. 2012. "Efficient Bayesian Inference for Multivariate Probit Models with Sparse Inverse Correlation Matrices." Journal of Computational and Graphical Statistics 21 (3).