CS4070 - MULTIVARIATE DATA ANALYSIS - PART 2

ASSIGNMENT 3

Konstantinos Krachtopoulos

Student Number: 5472539

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Technical University Delft

January 2022

1 Exercise 1

In order to get a prediction for the classification of new data $y*$, we need to calculate the probability $p(y*|x*, X, y)$, which can be decomposed as:

$$\begin{aligned} p(y^*|x^*, X, y) &= \int p(y^*|f^*, x^*, X, y)p(f^*|x^*, X, y)df^* \\ &= \int p(y^*|f^*, x^*, X, y) \int p(f^*|f, x^*, X, y)p(f|x^*, X, y)df df^* \end{aligned} \tag{1}$$

By removing the conditional independent parameters, we have:

$$p(y^*|x^*, X, y) = \int p(y^*|f^*) \int p(f^*|f)p(f|X, y)df df^* \tag{2}$$

Hence, to sample from the predictive density, the following steps should be followed:

1. Sample f from the posterior $p(f|X, y)$
2. Sample f^* from $p(f*|f)$
3. Sample $y^* \sim Ber(\Phi(f*)$

Our first objective is to sample from $f|y$. For that reason, an auxilliary parameter z is introduced, for which

$$y_i = \mathbf{1}_{\{z_i \geq 0\}} \tag{3}$$

$$z_i|f_i \sim N(f_i, 1) \tag{4}$$

Since z is an unobserved variable, our objective is updated to $z, f|y$.

By exploring the calculations for the update step of f, we see that it amounts to drawing from the Multivariate Normal Distribution:

$$\begin{aligned} p(f|y, z) &\propto \phi_n(z; f, I_n)\phi_n(f; 0, \kappa) = \\ (2\pi)^{-n/2} exp(-\frac{1}{2}||z - f||^2) * (2\pi)^{-n/2}||K||^2 exp(-\frac{1}{2}f^T K^{-1} f) \\ &\propto exp(-\frac{1}{2}(-2z^T f + f^T f) - \frac{1}{2}f^T K^{-1} f) = \\ exp(z^T f - \frac{1}{2}(f^T f + f^T K^{-1} f)) &= \\ exp(-\frac{1}{2}f^T (I_n + K^{-1})f + z^T f) & \end{aligned} \tag{5}$$

The above implies:

$$f|z, y \sim N_n^{can}((I_n + K^{-1})z^T, I_n + K^{-1}) \tag{6}$$

with $\mu = z^T$ and $\Sigma^{-1} = I_n + K^{-1}$.

2 Exercise 2

The Gibbs sampler will iteratively update $\{f_i\}_{i-1}^n$ and $\{z_i\}_{i-1}^n$. First, the prediction will be updated. This will be done by sampling from the Multivariate Normal distribution, as calculated in Exercise 1, and using the Squared exponential kernel. Afterwards, the auxiliary variable will be updated, based on the following:

1. If $y_i = 1$, draw z_i from $N(f_i, 1)$ distribution, conditioned to be positive
2. If $y_i = 0$, draw z_i from $N(f_i, 1)$ distribution, conditioned to be negative

The complete code implementation can be seen in the appendix.

3 Exercise 3

Based on the exercise description, f is now sampled from $GP(0, vK)$, where v is a random variable. In order to account for this change, we also need to add a step for updating v, which is initially sampled from $IG(2, 2)$. Hence, the new structure of dependencies is as follows:

$$v \Rightarrow f_i \Rightarrow z_i \Rightarrow y_i \tag{7}$$

since v only influences f_i. The conditional probability of $v|z, f, y$ is given below. By removing all components that do not contain v, we derive the final distribution:

$$\begin{aligned} & p(v|z, f, y) \propto p(v, z, f, y) = p(v)p(f|v)p(z|f)p(y_i|z_i) \\ = (2\pi)^{-\frac{n}{2}}||vK||^{-\frac{1}{2}}exp(-\frac{1}{2}f^T v^{-1}K^{-1}f) * (2\pi)^{-\frac{n}{2}}exp(-\frac{1}{2}||z-f||^2) * \mathbf{1}_{\{z_i \geq 0\}} * \frac{\beta^\alpha}{\Gamma(\alpha)}(1/v)^{\alpha+1}exp(-\frac{\beta}{v}) \\ & \propto ||vK||^{-\frac{1}{2}}exp(-\frac{1}{2}f^T v^{-1}K-1f)(1/v)^{\alpha+1}exp(-\frac{\beta}{v}) \\ & = (1/v)^{\frac{n}{2}}||K||^{-\frac{1}{2}}(1/v)^{\alpha+1}exp(-\frac{1}{2}f^T v^{-1}K^{-1}f - \frac{\beta}{v}) \\ & \propto (1/v)^{\frac{n}{2}+\alpha+1}exp(-\frac{1}{v}(\beta + \frac{1}{2}f^T v^{-1}K^{-1}f)) \end{aligned} \tag{8}$$

This implies:

$$v|z, f, y \sim IG(a + \frac{n}{2}, \beta + \frac{1}{2}f^T K^{-1}f) \tag{9}$$

4 Exercise 4

A small dataset of 100 samples with one parameter was constructed, sampled from a uniform distribution between -1 and 1. The dataset was labeled with two clusters; one containing all samples with a value greater than zero, and one with all the values less than zero.

Afterwards, the Gibbs sampling algorithm was executed in order to compute the f_i. For that reason, f_i, z_i and v were updated in turn.

The following traceplots present the coefficients' values over 10000 iterations:

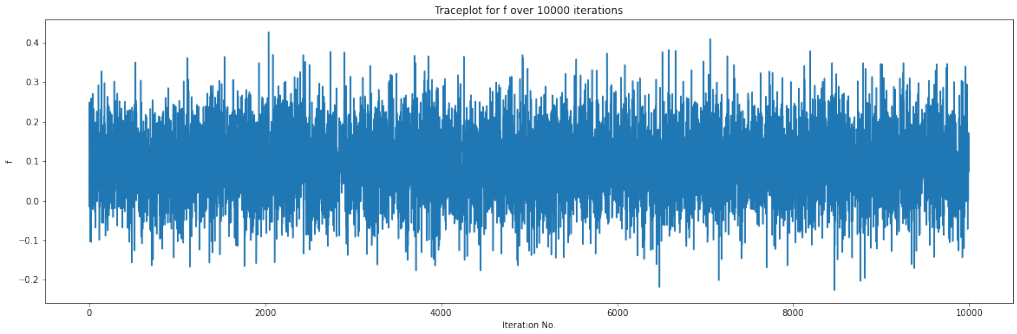

Figure 1: Traceplot of f_i for i=3.

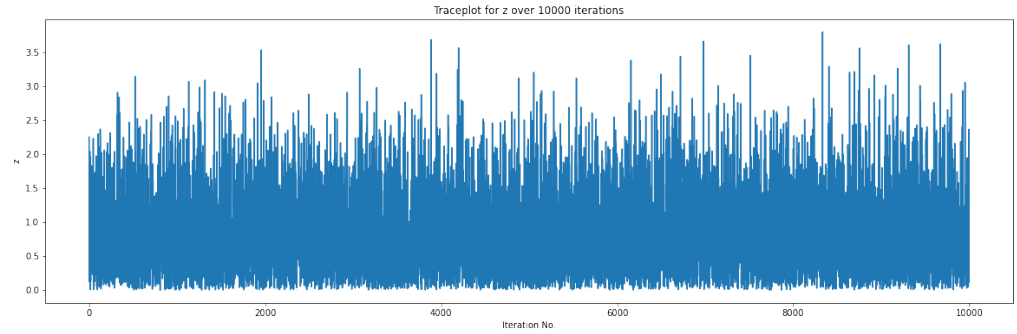

Figure 2: Traceplot of z_i for i=3.

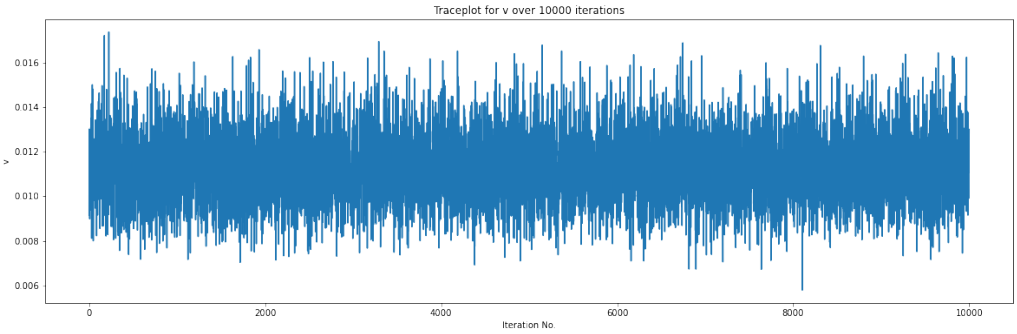

Figure 3: Traceplot of parameter v.

We can see that for the traceplots of f_3 and v a stationary regime is clearly observed. For the traceplot of z_3, we see that all values are positive, meaning that the sample is always assigned to the positive cluster.

5 Appendix

The code used for implementing Gibbs Sampling is presented below, together with the appropriate comments:

```
# Basic imports
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,6)

# Main functions definition
# Function to select only positive or negative values of a standard normal gaussian
def selectBySign(sample, sign):
        """
        Select draw based of conditional gaussian
    :param sample: Sample to be used as the mean of the Gaussian (real)
    :param sign: Can take values "+" or "-" if the sign is +, the below flag is true. Else, is
        is false (string)
    :return: draw with the desired sign (real)
```

```
    """
        positiveFlag = sign == "+"
        while True:
                draw = np.random.normal(sample, scale=1)
                if draw > 0 and positiveFlag:  # in case of positive gaussian
                        return draw
                elif draw < 0 and not positiveFlag:  # in case of negative gaussian
                        return draw

# Exponential Function Kernel Function
def sqrExpKernel(X1, X2, lam=1.0, v=1.0):
        """
    Squared exponential kernel
    :param X1: Array of m samples and p parameters (real - mxp)
    :param X2: Array of n samples and p parameters (real - nxp)
    :param l: Characteristic length (real)
    :param v: Signal variance (real)
    :return: Array with kernel functions between the points (real - mxn)
    """
        sqrDist = np.sum(X1 ** 2, 1).reshape(-1, 1) + np.sum(X2 ** 2, 1) - 2 * X1 @ X2.T
        kernel = v * np.exp(-1 / 2 * lam ** 2 * sqrDist) + 1e-5 * np.eye(X1.shape[0])
        return kernel

# Function to sample v.
def sample_v(f, kernel, samples, a_init=2, b_init=2):
        """
        Sample v from an inverse gamma distribution
        :param f: Array of n labels (real)
        :param kernel: Array with kernel functions (real - nxn)
        :param samples: number of samples (int)
        :param a_init: initial value of alpha parameter (int)
        :param b_init: initial value of beta parameter (int)
        :return: v parameter (real)
        """
        b_new = (b_init + 0.5 * f.T @ np.linalg.inv(kernel) @ f)
        return np.random.gamma(a_init + samples/2, b_new**(-1))**(-1)

# Create test dataset
# Sample one-dimensional design matrix from uniform distribution between -1 and 1.
x = np.random.uniform(-1, 1, 100).reshape(100, 1)

# Add labels to dataset. Classify the data based on their sign
y = (x >= 0).astype(float)

# Initialize auxilliary variable z
zvec = np.zeros((100, 1))

# Initialize v
a_init = 2
b_init = 2
v = 1 / np.random.gamma(2, b_init**(-1))

# Iterate and update f, z, v variables
for i in range(samples):
```

```
        ## Compute kernel and canonical parameters
        # Compute covariance matrix with latest v using square exponential kernel
        cov = sqrExpKernel(x, x, 1, v)
        # Compute updated covariance for P(F|z,Y)
        fcov = np.linalg.inv(np.linalg.inv(cov) + np.eye(100, 100))
        # Compute updated mean vector for P(F|z,Y)
        fmean = np.dot(fcov, zvec)
        ## UPDATE F
        # Sample from the multivariate normal for P(F|z,Y).
        f_samples = np.random.multivariate_normal(fmean.flatten(), fcov, 1).T
        f_buffer.append(f_samples)

        ## UPDATE Z
        zz = np.zeros(100)
        for i in range(100):
                if y[i] == 0:
                        # Sample from conditional P(z_i|f_i,Y=0)
                        val = selectBySign(f_samples[i], "-")
                        zvec[i] = val
                        zz[i] = val
                else:
                        # Sample from conditional P(z_i|f_i,Y=1)
                        val = selectBySign(f_samples[i], "+")
                        zvec[i] = val
                        zz[i] = val

        # Add z vector to the buffer
        z_buffer.append(zz)
        ## UPDATE V
        v = sample_v(f_samples, cov, samples)
        v_buffer.append(v)  #add v to the buffer

# Plot the traceplot for a given f
sample = 2
f = np.stack(f_buffer)
plt.plot(f[:, sample])
plt.title("Traceplot for f over 10000 iterations")
plt.xlabel("Iteration No.")
plt.ylabel("f")
# plt.savefig("traceplot_f")
plt.show()

# Plot the traceplot for a given z
sample = 2
z = np.stack(z_buffer)
plt.plot(z[:, sample])
plt.title("Traceplot for z over 10000 iterations")
plt.xlabel("Iteration No.")
plt.ylabel("z")
# plt.savefig("traceplot_z")
plt.show()

# Plot the traceplot for v
v = np.stack(v_buffer[1:])[:, 0, 0]
```

```
plt.plot(v)
plt.title("Traceplot for v over 10000 iterations")
plt.xlabel("Iteration No.")
plt.ylabel("v")
# plt.savefig("traceplot_v")
plt.show()
```