Comp 0086 Assignment 3

Alexandra Maria Proca (SN: 20047328)

November 26, 2020

Question 1.

---

a. The joint probability is
$P(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)$
$= P(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\phi})P(\boldsymbol{\phi}|\beta)P(\boldsymbol{z}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\alpha)$

Written out,

$$= \prod_{k=1}^{K} P(\phi_k|\beta) \prod_{d=1}^{D} \left(P(\theta_d|\alpha) \prod_{i=1}^{W} P(z_{id}|\theta_d) P(x_{id}|z_{id}, \phi) \right)$$

$$= \prod_{k=1}^{K} \left(\frac{\Gamma\left(\sum\limits_{w=1}^{W} \beta\right)}{\prod\limits_{w=1}^{W} \Gamma(\beta)} \prod_{w=1}^{W} \phi_{kw}^{\beta-1} \right) \prod_{d=1}^{D} \left(\left(\frac{\Gamma\left(\sum\limits_{k=1}^{K} \alpha\right)}{\prod\limits_{k=1}^{K} \Gamma(\alpha)} \prod_{k=1}^{K} \theta_{dk}^{\alpha-1} \right) \prod_{w=1}^{W} \theta_{dk}^{A_{dk}} \phi_{kw}^{B_{kw}} \right)$$

which can be simplified,

$$= \prod_{k=1}^{K} \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{w=1}^{W} \phi_{kw}^{\beta-1} \right) \prod_{d=1}^{D} \left(\left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_{k=1}^{K} \theta_{dk}^{\alpha-1} \right) \prod_{w=1}^{W} \theta_{dk}^{A_{dk}} \phi_{kw}^{B_{kw}} \right)$$

$$= \prod_{k=1}^{K} \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{w=1}^{W} \phi_{kw}^{\beta-1} \phi_{kw}^{B_{kw}} \right) \prod_{d=1}^{D} \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_{k=1}^{K} \theta_{dk}^{\alpha-1} \theta_{dk}^{A_{dk}} \right)$$

$$= \prod_{k=1}^{K} \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}} \right) \prod_{d=1}^{D} \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_{k=1}^{K} \theta_{dk}^{\alpha-1+A_{dk}} \right)$$

b. The Gibbs sampling updates for z_{id} can be written as
$P(z_{id}|z_{\sim id}, \boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}, \alpha, \beta)$

$$= \frac{P(z_{id}, z_{\sim id}, \boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)}{P(z_{\sim id}, \boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta)}$$

Substituting the joint probability found in (a) and using the fact that A_{dk} and B_{kw} are counts over $\boldsymbol{z}$, the joint probability of $z_{(\sim i)d}$ can be represented by removing the z_{id}^{th} count in A_{dk} and B_{kw},

$$= \frac{\prod\limits_{k=1}^{K} \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod\limits_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}} \right) \prod\limits_{d=1}^{D} \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod\limits_{k=1}^{K} \theta_{dk}^{\alpha-1+A_{dk}} \right)}{\prod\limits_{k=1}^{K} \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod\limits_{w=1}^{W} \phi_{kw}^{\beta-1+(B_{kw}^{(\sim i)d})} \right) \prod\limits_{d=1}^{D} \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod\limits_{k=1}^{K} \theta_{dk}^{\alpha-1+(A_{dk}^{(\sim i)d})} \right)}$$

Because A_{dk} and B_{kw} are counts over z, they can be written as:
$A_{dk}^{(\sim i)d} = A_{dk} - 1$
$B_{kw}^{(\sim i)d} = B_{kw} - 1$
when $z_{id} = k, d = d$ at z_{id} for A_{dk} and when $x_{id} = w, z_{id} = k, d = d$ at z_{id} for B_{kw}. $A_{dk}^{(\sim i)d} = A_{dk}$ and $B_{kw}^{(\sim i)d} = B_{kw}$ when $k \neq z_{id}$, and $k \neq z_{id}, w \neq i$, respectively, for $d = d$ at z_{id}.

Thus, solving for a particular z_{id} and canceling $k \neq z_{id}, w \neq i$, and $d \neq d$ at z_{id} yields

$$\frac{\left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W}\phi_{kw}^{\beta-1+B_{kw}}\right)\left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\theta_{dk}^{\alpha-1+A_{dk}}\right)}{\left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W}\phi_{kw}^{\beta-1+(B_{kw}-1)}\right)\left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\theta_{dk}^{\alpha-1+(A_{dk}-1)}\right)}$$
$$= \phi_{kw}\theta_{dk}$$

Solving for $P(\boldsymbol{\theta}_d|\boldsymbol{\theta}_{\sim d}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\phi}, \alpha, \beta)$,

$$= \frac{P(\boldsymbol{\theta}_d, \boldsymbol{\theta}_{\sim d}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\phi}\ |\alpha, \beta)}{P(\boldsymbol{\theta}_{\sim d}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\phi}\ |\alpha, \beta)}$$
$$= \frac{P(\boldsymbol{\theta}_d, \boldsymbol{\theta}_{\sim d}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\phi}\ |\alpha, \beta)}{\int P(\boldsymbol{\theta}_d, \boldsymbol{\theta}_{\sim d}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\phi}\ |\alpha, \beta)d\boldsymbol{\theta}_d}$$
$$= \frac{\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}}{\int \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}d\boldsymbol{\theta}_d}$$

The denominator has a form similar to the Dirichlet distribution:

$$\int \frac{\Gamma\left(\sum_{k=1}^{K} A_{dk}+\alpha\right)}{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}d\boldsymbol{\theta}_d = 1$$

Thus, to simplify the denominator,

$$\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}\int\frac{\Gamma\left(\sum_{k=1}^{K} A_{dk}+\alpha\right)}{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}d\boldsymbol{\theta}_d$$
$$= \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}$$

Substituting into the fraction,

$$= \frac{\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}}{\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}}$$
$$= \frac{\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}}{\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}}$$
$$= \frac{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}}{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}$$
$$= \mathrm{Dir}(\boldsymbol{A}_d+\boldsymbol{\alpha})$$

Solving for $P(\boldsymbol{\phi}_k|\boldsymbol{\phi}_{\sim k}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta}, \alpha, \beta)$,

$$= \frac{P(\boldsymbol{\phi}_k, \boldsymbol{\phi}_{\sim k}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\theta}\ |\alpha, \beta)}{P(\boldsymbol{\phi}_{\sim k}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\theta}\ |\alpha, \beta)}$$

$$= \frac{P(\boldsymbol{\phi}_k, \boldsymbol{\phi}_{\sim k}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\theta}\ |\alpha, \beta)}{\int P(\boldsymbol{\phi}_k, \boldsymbol{\phi}_{\sim k}, \boldsymbol{z}, \boldsymbol{x}, \boldsymbol{\theta}\ |\alpha, \beta) d\boldsymbol{\phi}_k}$$

$$= \frac{\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}}}{\int \frac{\Gamma(W\beta)}{\Gamma(\beta)^K} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}} d\boldsymbol{\phi}_k}$$

The denominator has a form similar to the Dirichlet distribution:

$$\int \frac{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)}{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}} d\boldsymbol{\phi}_k = 1$$

Thus, to simplify the denominator,

$$\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \frac{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)} \int \frac{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)}{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}} d\boldsymbol{\phi}_k$$

$$= \frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \frac{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)}$$

Substituting into the fraction,

$$= \frac{\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}}}{\frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \frac{\prod_{w=1}^{W} \Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw}+\beta\right)}}$$

$$= \frac{\prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}}}{\frac{\prod_{w=1}^{W} \Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw}+\beta\right)}}$$

$$= \frac{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right) \prod_{w=1}^{W} \phi_{kw}^{\beta-1+B_{kw}}}{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)}$$

$$= \mathrm{Dir}(\boldsymbol{B}_k + \boldsymbol{\beta})$$

c. Integrating out the parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\phi}_k$ from the joint probability,

$$\int_{\theta_d} \int_{\phi_k} P(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta) d\boldsymbol{\phi}_k d\boldsymbol{\theta}_d$$

$$\int_{\theta_d}\int_{\phi_k}\prod_{k=1}^{K}\left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W}\prod_{w=1}^{W}\phi_{kw}^{\beta-1+B_{kw}}\right)\prod_{d=1}^{D}\left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\prod_{k=1}^{K}\theta_{dk}^{\alpha-1+A_{dk}}\right)d\boldsymbol{\phi}_k d\boldsymbol{\theta}_d$$

Using the integrals found in (b),

$$=\prod_{k=1}^{K}\left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W}\frac{\prod_{w=1}^{W}\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\right)\prod_{d=1}^{D}\left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K}\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}\right)$$

$$=\prod_{k=1}^{K}\left(\frac{\Gamma\left(\sum_{w=1}^{W}\beta\right)}{\prod_{w=1}^{W}\Gamma(\beta)}\frac{\prod_{w=1}^{W}\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\right)\prod_{d=1}^{D}\left(\frac{\Gamma\left(\sum_{k=1}^{K}\alpha\right)}{\prod_{k=1}^{K}\Gamma(\alpha)}\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}\right)$$

which can be rewritten in terms of the Beta function

$$=\prod_{k=1}^{K}\left(\frac{1}{B(\boldsymbol{\beta})}B(\boldsymbol{B}_k+\boldsymbol{\beta})\right)\prod_{d=1}^{D}\left(\frac{1}{B(\boldsymbol{\alpha})}B(\boldsymbol{A}_d+\boldsymbol{\alpha})\right)$$

$$=\prod_{k=1}^{K}\left(\frac{B(\boldsymbol{B}_k+\boldsymbol{\beta})}{B(\boldsymbol{\beta})}\right)\prod_{d=1}^{D}\left(\frac{B(\boldsymbol{A}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right)$$

d. Using the joint probability with the parameters integrated out, the Gibbs sampling update can be written

$$P(z_{id}|z_{(\sim i)d},\boldsymbol{x},\alpha,\beta)$$

$$=\frac{P(z_{id},z_{(\sim i)d},\ \boldsymbol{x}|\alpha,\beta)}{P(z_{(\sim i)d},\ \boldsymbol{x}|\alpha,\beta)}$$

$$=\frac{\prod_{k=1}^{K}\left(\frac{B(\boldsymbol{B}_k+\boldsymbol{\beta})}{B(\boldsymbol{\beta})}\right)\prod_{d=1}^{D}\left(\frac{B(\boldsymbol{A}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right)}{\prod_{k=1}^{K}\left(\frac{B(\boldsymbol{B}_k+\boldsymbol{\beta})}{B(\boldsymbol{\beta})}\right)_{(\sim i)d}\prod_{d=1}^{D}\left(\frac{B(\boldsymbol{A}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\right)_{(\sim i)d}}$$

$$=\prod_{k=1}^{K}\left(\frac{B(\boldsymbol{B}_k+\boldsymbol{\beta})}{B(\boldsymbol{\beta})}\times\frac{B(\boldsymbol{\beta})}{B(\boldsymbol{B}_k^{(\sim i)d}+\boldsymbol{\beta})}\right)\prod_{d=1}^{D}\left(\frac{B(\boldsymbol{A}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}\times\frac{B(\boldsymbol{\alpha})}{B(\boldsymbol{A}_d^{(\sim i)d}+\boldsymbol{\alpha})}\right)$$

$$=\prod_{k=1}^{K}\left(B(\boldsymbol{B}_k+\boldsymbol{\beta})\times\frac{1}{B(\boldsymbol{B}_k^{(\sim i)d}+\boldsymbol{\beta})}\right)\prod_{d=1}^{D}\left(B(\boldsymbol{A}_d+\boldsymbol{\alpha})\times\frac{1}{B(\boldsymbol{A}_d^{(\sim i)d}+\boldsymbol{\alpha})}\right)$$

$$=\prod_{k=1}^{K}\left(\frac{\prod_{w=1}^{W}\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\sum_{w=1}^{W}B_{kw}^{(\sim i)d}+\beta\right)}{\prod_{w=1}^{W}\Gamma\left(B_{kw}^{(\sim i)d}+\beta\right)}\right)\prod_{d=1}^{D}\left(\frac{\prod_{k=1}^{K}\Gamma(A_{dk}+\alpha)}{\Gamma\left(\sum_{k=1}^{K}A_{dk}+\alpha\right)}\times\frac{\Gamma\left(\sum_{k=1}^{K}A_{dk}^{(\sim i)d}+\alpha\right)}{\prod_{k=1}^{K}\Gamma\left(A_{dk}^{(\sim i)d}+\alpha\right)}\right)$$

Because A_{dk} and B_{kw} are counts over z, they can be written as:

$A_{dk}^{(\sim i)d}=A_{dk}-1$

$B_{kw}^{(\sim i)d}=B_{kw}-1$

when $z_{id}=k,d=d$ at z_{id} for A_{dk} and when $x_{id}=w,z_{id}=k,d=d$ at z_{id} for B_{kw}. $A_{dk}^{(\sim i)d}=A_{dk}$ and $B_{kw}^{(\sim i)d}=B_{kw}$ when $k\neq z_{id}$, and $k\neq z_{id},w\neq i$, respectively, for $d=d$ at z_{id}.

Taking the first part of the expression and rewriting in terms of the previous assertions, solving for a particular z_{id},

$$\frac{\prod_{w=1}^{W}\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)}{\Gamma(B_{kw}-1+\beta)\prod_{w\neq i}^{W}\Gamma(B_{kw}+\beta)}$$

$$=\frac{\Gamma(B_{kw}+\beta)\prod_{w\neq i}^{W}\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)}{\Gamma(B_{kw}-1+\beta)\prod_{w\neq i}^{W}\Gamma(B_{kw}+\beta)}$$

$$=\frac{\Gamma(B_{kw}+\beta)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)}{\Gamma(B_{kw}-1+\beta)}$$

$$=\frac{\Gamma(B_{kw}+\beta-1+1)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)}{\Gamma(B_{kw}-1+\beta)}$$

Using the identity $\Gamma(1+x)=x\Gamma(x)$ for $x>0$,

$$=\frac{(B_{kw}+\beta-1)\Gamma(B_{kw}+\beta-1)}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\frac{\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)}{\Gamma(B_{kw}-1+\beta)}$$

$$=\frac{B_{kw}+\beta-1}{\Gamma\left(\sum_{w=1}^{W}B_{kw}+\beta\right)}\times\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)$$

$$=\frac{B_{kw}+\beta-1}{\Gamma\left(\left(\sum_{w=1}^{W}B_{kw}+\beta\right)-1+1\right)}\times\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)$$

$$=\frac{B_{kw}+\beta-1}{\left(\left(\sum_{w=1}^{W}B_{kw}+\beta\right)-1\right)\Gamma\left(\left(\sum_{w=1}^{W}B_{kw}+\beta\right)-1\right)}\times\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)$$

$$=\frac{B_{kw}+\beta-1}{\left(\left(\sum_{w=1}^{W}B_{kw}+\beta\right)-1\right)\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}+\beta-1\right)}\times\Gamma\left(\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}-1+\beta\right)$$

$$=\frac{B_{kw}+\beta-1}{\left(\sum_{w=1}^{W}B_{kw}+\beta\right)-1}$$

$$=\frac{B_{kw}+\beta-1}{M_k+W\beta-1}$$

Written equivalently,

$$=\frac{B_{kw}+\beta-1}{\left(\sum_{w\neq i}^{W}B_{kw}+\beta\right)+B_{kw}+\beta-1}$$

$$\equiv \frac{B_{kw}^{(\sim i)d} + \beta}{\sum\limits_{w=1}^{W} B_{kw}^{(\sim i)d} + \beta}$$

Similarly, taking the second part of the expression and rewriting in terms of the previous assertions,

$$\frac{\prod\limits_{k=1}^{K} \Gamma(A_{dk} + \alpha)}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \frac{\Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)}{\Gamma(A_{dk} - 1 + \alpha) \prod\limits_{k \neq z_{id}}^{K} \Gamma(A_{dk} + \alpha)}$$

$$= \frac{\Gamma(A_{dk} + \alpha) \prod\limits_{k \neq z_{id}}^{K} \Gamma(A_{dk} + \alpha)}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \frac{\Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)}{\Gamma(A_{dk} - 1 + \alpha) \prod\limits_{k \neq z_{id}}^{K} \Gamma(A_{dk} + \alpha)}$$

$$= \frac{\Gamma(A_{dk} + \alpha)}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \frac{\Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)}{\Gamma(A_{dk} - 1 + \alpha)}$$

$$= \frac{\Gamma(A_{dk} + \alpha - 1 + 1)}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \frac{\Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)}{\Gamma(A_{dk} - 1 + \alpha)}$$

$$= \frac{(A_{dk} + \alpha - 1)\Gamma(A_{dk} + \alpha - 1)}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \frac{\Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)}{\Gamma(A_{dk} - 1 + \alpha)}$$

$$= \frac{A_{dk} + \alpha - 1}{\Gamma\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right)} \times \Gamma\left(\left(\sum_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)$$

$$= \frac{A_{dk} + \alpha - 1}{\Gamma\left(\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right) - 1 + 1\right)} \times \Gamma\left(\left(\sum_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)$$

$$= \frac{A_{dk} + \alpha - 1}{\left(\sum\limits_{k=1}^{K} (A_{dk} + \alpha) - 1\right) \Gamma\left(\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right) - 1\right)} \times \Gamma\left(\left(\sum_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)$$

$$= \frac{A_{dk} + \alpha - 1}{\left(\sum\limits_{k=1}^{K} (A_{dk} + \alpha) - 1\right) \Gamma\left(\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} + \alpha - 1\right)} \times \Gamma\left(\left(\sum_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} - 1 + \alpha\right)$$

$$= \frac{A_{dk} + \alpha - 1}{\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right) - 1}$$

Written equivalently,

$$= \frac{A_{dk} + \alpha - 1}{\left(\sum\limits_{k \neq z_{id}}^{K} A_{dk} + \alpha\right) + A_{dk} + \alpha - 1}$$

$$\equiv \frac{A_{dk}^{(\sim i)d} + \alpha}{\sum\limits_{k=1}^{K} A_{dk}^{(\sim i)d} + \alpha}$$

Combing the two expressions yields

$$\frac{B_{kw} + \beta - 1}{\left(\sum\limits_{w=1}^{W} B_{kw} + \beta\right) - 1} \times \frac{A_{dk} + \alpha - 1}{\left(\sum\limits_{k=1}^{K} A_{dk} + \alpha\right) - 1}$$

$$\equiv \frac{B_{kw}^{(\sim i)d} + \beta}{\sum\limits_{w=1}^{W} B_{kw}^{(\sim i)d} + \beta} \times \frac{A_{dk}^{(\sim i)d} + \alpha}{\sum\limits_{k=1}^{K} A_{dk}^{(\sim i)d} + \alpha}$$

e. The Dirichlet probability distribution is defined as:

$$P(\boldsymbol{\theta}_d|\alpha) = \frac{\Gamma\left(\sum\limits_{k=1}^{K} \alpha\right)}{\prod\limits_{k=1}^{K} \Gamma(\alpha)} \prod_{k=1}^{K} \theta_{dk}^{\alpha-1}$$

Thus when $\alpha = 1$, θ_d will be evenly distributed across all θ_{dk}, as $\theta_{dk}^0 = 1$. As $\alpha > 1$ increases, a more dense and uniform distribution will emerge across all θ_{dk}. As $\alpha < 1$ decreases, a more sparse distribution will emerge and condense on particular θ_{dk}, resulting in a high probability of a smaller space of topics in document d. The same is true for ϕ_{kw} and β, also being defined by the Dirichlet distribution. As $\beta > 1$ increases, a more dense and uniform distribution will emerge across all ϕ_{kw} and as $\beta < 1$ decreases, a more sparse and condensed distribution will emerge on particular ϕ_{kw}. In the collapsed Gibbs sampler, θ_d and ϕ_k are integrated out, resulting in a similar, but less pronounced effect of varying α and β, due to the absence of intermediate distributions between α, β and z_{id}. Therefore, hyperpriors for α and β could be better chosen based on the types of documents and topics, if known. For example, if the documents are likely to contain many various topics, evenly distributed across the topic space (possibly encyclopedias), a high value for α may be optimal, as it would allow for a more dense and uniform distribution of topics. Alternatively, if the documents each contain few topics from within the topic space or concentrate on particular topics over the topic space (for example, research papers), a low value for α may be optimal. The same applies for β based on the distribution of words for the topics in the topic space. If the topics are likely to be represented by a broad range of uniformly-distributed words from within the word space, a larger value of β may be more optimal. If the topics are represented by a sparse and condensed distribution of words, a smaller value of β may be more optimal. From these observations, hyperpriors for α and β of smaller values (< 1) would likely be most appropriate across all general document types. This makes the assumption that documents are likely to contain a smaller subspace of concentrated topics from within a larger topic space and topics themselves are likely to be described using a smaller subset of concentrated words from an overall word space, especially if the topic space and word space are vast.

Let the space of hyperpriors be $\boldsymbol{\alpha} = [\alpha'_1, \alpha'_2, \ldots, \alpha'_n]$ and $\boldsymbol{\beta} = [\beta'_1, \beta'_2, \ldots, \beta'_n]$

Then, to generate samples of α,

$$P(\alpha = \alpha'|\boldsymbol{x}, \boldsymbol{z}, \beta) = \frac{P(\boldsymbol{x}, \boldsymbol{z}|\alpha', \beta)P(\alpha')P(\beta)}{\int P(\boldsymbol{x}, \boldsymbol{z}|\alpha', \beta)P(\alpha')P(\beta)d\alpha'}$$

$$\propto \frac{P(\boldsymbol{x}, \boldsymbol{z}|\alpha', \beta)P(\alpha')P(\beta)}{P(\boldsymbol{x}, \boldsymbol{z}|\beta)P(\beta)}$$

Because the denominator remains constant for all α',
$\propto P(\boldsymbol{x}, \boldsymbol{z}|\alpha', \beta)P(\alpha')$

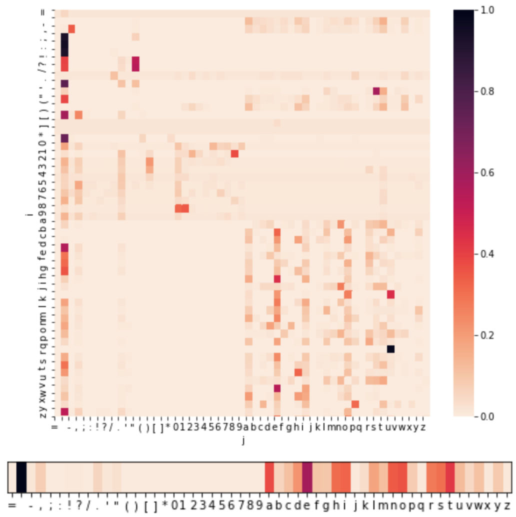
Similarly for β,
$P(\beta = \beta'|\boldsymbol{x}, \boldsymbol{z}, \alpha) = \dfrac{P(\boldsymbol{x}, \boldsymbol{z}|\beta', \alpha)P(\beta')P(\alpha)}{\int P(\boldsymbol{x}, \boldsymbol{z}|\beta', \alpha)P(\beta')P(\alpha)d\beta'}$
$\propto \dfrac{P(\boldsymbol{x}, \boldsymbol{z}|\beta', \alpha)P(\beta')P(\alpha)}{P(\boldsymbol{x}, \boldsymbol{z}|\alpha)P(\alpha)}$
Because the denominator remains constant for all β',
$\propto P(\boldsymbol{x}, \boldsymbol{z}|\beta', \alpha)P(\beta')$

Question 2.

a. The probability of transitioning from any symbol β to any symbol α is
$\psi(\alpha, \beta) \equiv p(s_i = \alpha|s_{i-1} = \beta)$
$= \dfrac{p(s_i = \alpha|s_{i-1} = \beta)}{p(s_{i-1} = \beta)}$
Rewritten as the ML estimate of the probability,
$\approx \dfrac{\sum_{i=2}^{N} \xi_{i-1}(\beta \to \alpha)}{\sum_{i=2}^{N} \phi_{i-1}(\beta)}$
where $\xi_{i-1}(\beta \to \alpha) = 1$ if β transitions to α at positions $i-1$ to i and 0 otherwise, and $\phi_{i-1}(\beta) = 1$ if there is an occurrence of β at position $i-1$ and 0 otherwise.
Rewritten as a function of the counts of numbers of occurrences of symbols and pairs of symbols, the transition probability of any symbol i to symbol j is
$T_{ij} = \dfrac{C_{ij}}{O_i}$
where C_{ij} is the count of the number of pairs of symbols i and j and O_i is the number of occurrences of symbol i.

The stationary probability is defined as
$\lim_{i\to\infty} p(s_i = \gamma) \equiv \phi(\gamma)$
where the stationary distribution remains constant when multiplied by the transition matrix,
$\phi T = \phi$
Taking the transpose,
$(\phi T)^\mathsf{T} = \phi^\mathsf{T}$
$T^\mathsf{T}\phi^\mathsf{T} = \phi^\mathsf{T}$
Rewritten,
$T^\mathsf{T}\phi^\mathsf{T} = \lambda\phi^\mathsf{T}$
where $\lambda = 1$.
From this form, it is clear that ϕ^T is an eigenvector of T^T with an eigenvalue λ of 1. Thus, the stationary distribution can be estimated by finding the eigenvector with eigenvalue $\lambda = 1$ of the transposed estimated transition matrix T above and transposing it.

The transition probabilities from symbols i to symbols j in transition matrix T computed and the stationary probabilities for all symbols i computed are displayed below as heatmaps.



b. The latent variables $\sigma(s)$ for different symbols s are not independent because each latent variable is dependent on whether another latent variable represents a symbol s, as each $\sigma(s)$ is mapped to a unique s. Thus, each latent variable is dependent on all other latent variables.
An encrypted message $e_1e_2...e_n$ is a direct one-to-one mapping by $\sigma^{-1}(e_1)\sigma^{-1}(e_2)\ldots\sigma^{-1}(e_n) = s_1s_2\ldots s_n$. Thus, given σ,
$p(e_1e_2...e_n|\sigma) = p(s_1s_2...s_n)$
$= p(s_1)\prod_{i=2}^{n} p(s_i|s_{i-1})$

c. The proposal probability $S(\sigma \to \sigma')$ is not dependent on the permutations σ and σ' because two symbols are randomly swapped within the one-to-one mappings of the permutations. Thus the proposal probability is equal to choosing s and s' at random: $S(\sigma \to \sigma') = \frac{1}{\binom{n}{2}} = \frac{1}{\binom{53}{2}}$.
The MH acceptance probability depends on the likelihood of the permutations σ and σ' having encoded a sequence of standard English text S into the encrypted message E.
$A(\sigma'(S)|\sigma(S)) = \min\left\{1, \dfrac{p(\sigma'(S) = E)p(\sigma')}{p(\sigma(S) = E)p(\sigma)}\right\}$
Because $p(\sigma') = p(\sigma) = \frac{1}{53!}$, this simplifies to
$A(\sigma'(S)|\sigma(S)) = \min\left\{1, \dfrac{p(\sigma'(S) = E)}{p(\sigma(S) = E)}\right\}$
The probability of $p(\sigma(S) = E)$ for a particular σ can be computed by finding the corresponding sequence of mapped symbols S^* from the encrypted message E, $\sigma^{-1}(E) = S^*$, and calculating the product of all of the transitions $T(s_i^*, s_j^*)$ in the mapped sequence and the stationary probability $\phi(s_1^*)$ of the first encrypted symbol, as it gives the probability that the proposed mapped sequence would occur in the model of standard English given the occurrence of particular transitions and initial value:
$p(\sigma(S) = E) = p(s_1^*)\prod_{i=2}^{N} p(s_i^*|s_{i-1}^*)$
$= \phi(s_1^*)\prod_{i=2}^{N} T\left(s_{i-1}^*, s_i^*\right)$

d. The transition matrix was computed using the text of War and Peace and the stationary probability was then computed with the transition matrix with the ML estimates in part (a). The initial mapping of σ was created by mapping symbols to each other based on their corresponding occurrence in the encrypted message and magnitude of stationary probability (encrypted symbols with highest occurrence mapped to symbols with highest stationary probabilities). The symbols that did not appear in the encrypted message were assigned to the remainder of the symbols with lower stationary probabilities. This 'intelligent' initialization of the chain allowed the algorithm to converge faster, as it attempted to find the encrypted symbols with the highest initial probability of being mapped to the decrypted symbols. The MH Sampler function calls a function to create a new randomized swap of two symbols and calls a function to compute the log-likelihood of the initial and proposed σ. It then computes the acceptance probability and evaluates whether the proposed σ is accepted.

```
import numpy as np
import matplotlib.pyplot as plt
from mpltools import special
import pandas as pd
import collections
from numpy import dot
from numpy.linalg import norm
import random
import seaborn as sns

warandpeace=open('war-and-peace.txt')
wap=warandpeace.read()

symbols=np.genfromtxt('symbols.txt',dtype='str')
symbols=np.asarray(symbols)
symbols=np.insert(symbols, 1, " ")

# Computes the transition matrix using War and Peace
newtransmatr=np.zeros((53,53))
for i in range(0, 53):
    for j in range(0,53):
        pair=symbols[i]+symbols[j]
        # Constant added to ensure no transitions of 0 probability
        newtransmatr[i][j]=wap.lower().count(pair) + 1
    newtransmatr[i]=newtransmatr[i]/np.sum(newtransmatr[i])

# Computes the stationary probability using the transition matrix
eval, evec = np.linalg.eig(newtransmatr.T)
idx = np.argmin(np.abs(eval - 1))
stationary = np.real(evec[:, idx]).T
stationary = stationary/np.sum(stationary)

# Finds number of occurrences of symbols in encrypted message
message=open('message.txt')
mess=message.read()
messar=list(mess)
occur = collections.Counter(messar)
occur=occur.most_common()
# Sorts stationary probabilities by highest values
stat_sorted=np.sort(stationary)[::-1]

# Initializes sigma by mapping symbols with the highest occurrence to the
# corresponding symbols with the highest stationary probabilities
symbdict=dict()
for i in range(0,len(occur)):
    symbind=int(np.where(stationary==stat_sorted[i])[0])
    symbdict[occur[i][0]]=symbollist[symbind]
```

```
# Iteratively assigns the remainder of the symbols that do not appear in
# the encrypted message
for i in range(0,53):
    if symbols[i] not in symbdict:
        for j in range(0,53):
            if symbols[j] not in symbdict.values():
                symbdict[symbols[i]]=symbols[j]
                break

# Swaps two mapped symbols and returns new mapping
def charswap(symbdict):
    sigmas, s=random.choice(list(symbdict.items()))
    sigmasprime, sprime=random.choice(list(symbdict.items()))
    if(sigmas == sigmasprime):
        charswap(symbdict)
    else:
        symbdict[sigmas]= sprime
        symbdict[sigmasprime] = s
    return symbdict

# Computes the log likelihood of a particular mapping
def loglike(transmatr,mess,symbdict,symbols,stationary):
    s1=symbdict.get(mess[0])
    s1=int(np.where(symbols==s1)[0])
    ll=np.log(stationary[s1])
    for i in range(len(mess)-1):
        si=symbdict.get(mess[i])
        si=int(np.where(symbols==si)[0])
        sj=symbdict.get(mess[i+1])
        sj=int(np.where(symbols==sj)[0])
        ll = ll + np.log(transmatr[si][sj])
    return ll

# Computes MH algorithm
def MH_Sampler(symbdict, transmatr, symbols, mess,stationary):
    switchdict=charswap(symbdict.copy())
    like1=loglike(np.copy(transmatr),mess,symbdict.copy(),np.copy(symbols),stationary)
    like2=loglike(np.copy(transmatr),mess,switchdict.copy(),np.copy(symbols),stationary)
    A=np.exp(like2-like1)
    if A > 1:
        A=1
    if random.random() <= A:
        symbdict=switchdict
    return symbdict

# Creates decrypted message given a particular mapping
def decode(mess,symbdict):
    translated=""
    for symb in mess.lower():
        translated=translated+(symbdict.get(symb))
    print(translated[0:59])
    return translated

# Runs the MH Sampler for 10000 iterations and prints decrypted message
# every 100 iterations
for i in range(0,10000):
```

```
        symbdict=MH_Sampler(symbdict, newtransmatr, symbols, mess, stationary)
        if i+1 % 100 == 0:
            print("Iteration: ", i+1)
            decode(mess, symbdict)
    decodedmessage=decode(mess, symbdict)
```

The MH Sampler function was run for 10000 iterations and printed the first 60 symbols of the decrypted message every 100 iterations. The function converged in about 9300 iterations.

```
Iteration:  0
on lw whunges tnm lhse furnestpre wetsi lw ytades gtfe le i
Iteration:  100
on fw whunaes tnm fhse durnestpre wetsg fw ytiles atde fe g
Iteration:  200
on hw waunfes tnm hase durnestpre wetsg hw ytiles ftde he g
Iteration:  300
or wh haurfen irm wane dusrenipse heing wh yitlen fide we g
Iteration:  400
ar wh hourfen irm wone dusrenipse heing wh yitlen fide we g
Iteration:  500
ar wh houryen ird wone musrenipse heing wh fitlen yime we g
Iteration:  600
ar wl louryen ird wone musrenipse leing wl fithen yime we g
Iteration:  700
ar bl louryen ird bone mugrenipge leins bl fithen yime be s
Iteration:  800
ar bl louryen ird bone mugrenipge leins bl fithen yime be s
Iteration:  900
ar bl louryen ird bone mugrenipge leins bl fithen yime be s
Iteration:  1000
ar bl lourken ird bone mugrenipge leins bl fithen kime be s
Iteration:  1100
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1200
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1300
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1400
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1500
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1600
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1700
ar bl lourken ird bone mufrenipfe leins bl githen kime be s
Iteration:  1800
ar pl lourken ird pone mufrenibfe leins pl githen kime pe s
Iteration:  1900
ar pl lourken ird pone mufrenibfe leins pl githen kime pe s
Iteration:  2000
ar pl lourken ird pone gufrenibfe leins pl mithen kige pe s
Iteration:  2100
ar pl lourken ird pone gufrenibfe leins pl mithen kige pe s
Iteration:  2200
ar pl lourken ird pone gufrenibfe leins pl mithen kige pe s
Iteration:  2300
ar pl lourken ird pone gufrenibfe leins pl mithen kige pe s
Iteration:  2400
ar pl lourken ird pone gufrenibfe leins pl mithen kige pe s
Iteration:  2500
ar pf fourken ird pone gulrenible feins pf mithen kige pe s
Iteration:  2600
ir pf fourken ard pone gulrenable feans pf mathen kage pe s
Iteration:  2700
ir pf fourken ard pone gulrenable feans pf mathen kage pe s
Iteration:  2800
ir pf fourken ard pone gulrenable feans pf mathen kage pe s
Iteration:  2900
ir py yourken ard pone gulrenable yeans py mathen kage pe s
Iteration:  3000
ir py yourken ard pone gulrenable yeans py mathen kage pe s
Iteration:  3100
ir py yourken ard pone gulrenable yeans py mathen kage pe s
Iteration:  3200
ir my yourken ard mone gulrenable yeans my pathen kage me s
Iteration:  3300
ir my yourken ard mone culrenable yeans my pathen kace me s
Iteration:  3400
ir my yourken ard mone culrenable yeans my pathen kace me s
Iteration:  3500
ir my yourken ard mone culrenable yeans my pathen kace me s
Iteration:  3600
ir my yourken ard mone culrenable yeans my pathen kace me s
Iteration:  3700
ir my yourken ard mone culrenable yeans my pathen kace me s
Iteration:  3800
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  3900
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  4000
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  4100
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  4200
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  4300
ir my yourken ard mone culrenable yeans my fathen kace me s
Iteration:  4400
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  4500
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  4600
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  4700
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  4800
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  4900
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5000
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5100
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5200
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5300
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5400
ir my yourken ard mone culrenaple yeans my fathen kace me s
Iteration:  5500
in my younker and more culneraple years my father kace me s
Iteration:  5600
in my younker and more gulneraple years my father kage me s
Iteration:  5700
in my younker and more gulneraple years my father kage me s
Iteration:  5800
in my younker and more gulneraple years my father kage me s
Iteration:  5900
in my younker and more gulneraple years my father kage me s
Iteration:  6000
in my younker and more gulneraple years my father kage me s
Iteration:  6100
in my younker and more gulneraple years my father kage me s
Iteration:  6200
in my younker and more gulneraple years my father kage me s
Iteration:  6300
in my younker and more gulneraple years my father kage me s
Iteration:  6400
in my younker and more gulneraple years my father kage me s
Iteration:  6500
in my younker and more gulnerable years my father kage me s
Iteration:  6600
in my younker and more gulnerable years my father kage me s
Iteration:  6700
in my younker and more gulnerable years my father kage me s
Iteration:  6800
in my younker and more gulnerable years my father kage me s
Iteration:  6900
in my younker and more gulnerable years my father kage me s
Iteration:  7000
in my younker and more gulnerable years my father kage me s
Iteration:  7100
in my younker and more gulnerable years my father kage me s
Iteration:  7200
in my younker and more gulnerable years my father kage me s
Iteration:  7300
in my younker and more gulnerable years my father kage me s
Iteration:  7400
in my younker and more gulnerable years my father kage me s
Iteration:  7500
in my younker and more gulnerable years my father kage me s
Iteration:  7600
in my younker and more gulnerable years my father kage me s
Iteration:  7700
in my younker and more gulnerable years my father kage me s
Iteration:  7800
in my younker and more gulnerable years my father kage me s
Iteration:  7900
in my younker and more gulnerable years my father kage me s
Iteration:  8000
in my younker and more gulnerable years my father kage me s
Iteration:  8100
in my younker and more gulnerable years my father kage me s
Iteration:  8200
in my younker and more gulnerable years my father kage me s
Iteration:  8300
in my younker and more gulnerable years my father kage me s
Iteration:  8400
in my younker and more gulnerable years my father kage me s
Iteration:  8500
in my younker and more gulnerable years my father kage me s
Iteration:  8600
in my younker and more gulnerable years my father kage me s
Iteration:  8700
in my younker and more vulnerable years my father kave me s
Iteration:  8800
in my younker and more vulnerable years my father kave me s
Iteration:  8900
in my younker and more vulnerable years my father kave me s
Iteration:  9000
in my younker and more vulnerable years my father kave me s
Iteration:  9100
in my younker and more vulnerable years my father kave me s
Iteration:  9200
in my younker and more vulnerable years my father kave me s
Iteration:  9300
in my younger and more vulnerable years my father gave me s
Iteration:  9400
in my younger and more vulnerable years my father gave me s
Iteration:  9500
in my younger and more vulnerable years my father gave me s
Iteration:  9600
in my younger and more vulnerable years my father gave me s
Iteration:  9700
in my younger and more vulnerable years my father gave me s
Iteration:  9800
in my younger and more vulnerable years my father gave me s
Iteration:  9900
in my younger and more vulnerable years my father gave me s
```

The final converged output yields:

```
in my younger and more vulnerable years my father gave me some advice that i've been turning
over in my mind ever since. "whenever you feel like criticizing any one," he told me, "just r
emember that all the people in this world haven't had the advantages that you've had." he did
n't say any more but we've always been unusually communicative in a reserved way, and i under
stood that he meant a great deal more than that. in consequence i'm inclined to reserve all j
udgments, a habit that has opened up many curious natures to me and also made me the victim o
f not a few veteran bores. the abnormal mind is quick to detect and attach itself to this qua
lity when it appears in a normal person, and so it came about that in college i was unjustly
accused of being a politician, because i was privy to the secret griefs of wild, unknown men.
most of the confidences were unsought--frequently i have feigned sleep, preoccupation, or a h
ostile levity when i realized by some unmistakable sign that an intimate revelation was quive
ring on the horizon--for the intimate revelations of young men or at least the terms in which
they express them are usually plagiaristic and marred by obvious suppressions. reserving judg
ments is a matter of infinite hope. i am still a little afraid of missing something if i forg
et that, as my father snobbishly suggested, and i snobbishly repeat a sense of the fundamenta
l decencies is parcelled out unequally at birth.
```

e. A Markov chain is ergodic if it possible to reach any given state from any other given state in the state space (including the same state transitioning to itself) and thus, in an ergodic chain there must be a probability greater than 0 that each state can be reached through any transition (all transition probabilities must be greater than 0). Therefore, in order for the chain to be ergodic, $\psi(\alpha, \beta) > 0$ for all α, β. In the given case, because some $\psi(\alpha, \beta)$ may be zero, the ergodicity of the chain is affected. It is impossible to reach particular states from other particular states. Furthermore, if α and β for which $\psi(\alpha, \beta) = 0$ are encountered consecutively in the encrypted message, the likelihood for the proposed σ becomes 0. In the current implementation, using log likelihood, taking the log of 0 results in $-\infty$, which will cause numerical errors and make it impossible to accurately compare certain σ. To restore ergodicity, $\psi(\alpha, \beta)$ for all α, β can be scaled by adding a small constant (such as 1) to the number of observed pairs (α, β) when computing the transition probability of English text.

f. Symbol probabilities alone would not be sufficient because many symbols in the English language appear with similar frequencies and, especially within a short message, the frequency of the encrypted symbols may not be accurately mirrored by the frequency of symbols in English as a whole. An implementation using this method is akin to the initialization of σ in (d), which uses the stationary distribution and the frequency of symbols in the encrypted message to create a mapping. The implementation still requires approximately 9000 iterations to converge as the original mapping does not adequately describe the encrypted message. Transition probability provides a more accurate representation of the probability of consecutive symbols and the probability of a sequence of symbols as a whole.

Using a second order Markov chain could potentially be more useful, as it would provide additional information about transition probabilities, giving a more precise probability from the structure and using additional context. Some problems could be the cost, as it would require more computation to calculate the transition probabilities for a second order chain, especially from a large text like War and Peace, and to compute the likelihoods at each iteration. Another potential problem could be that of overfitting the probabilities, as certain symbol patterns and higher structures may be more common in certain texts as opposed to others, making a second order chain potentially less generalizable.

The decoding approach could potentially work if the encryption scheme allowed two symbols to be mapped to the same encrypted value if the two symbols are different enough in their transition probabilities and occurrences that it can be predicted with high probability in which instance each symbol is being used. If the two symbols are clearly distinguishable in their transition probabilities (mapped symbols with orthogonal transition vectors, for example), this is possible. However, if the two symbols are ambiguous and approximately indistinguishable in their transition probabilities, this approach would not be very useful, as it would be difficult to predict which symbol the encrypted value is mapping to. For instance, an encryption using "a" for symbols "2" and "3" would be difficult to decrypt because the transition probabilities of 2 and 3 would likely be very similar. If "a" encoded "2" and "o", this approach may be more feasible. In this encryption scheme, a second order Markov chain may be more beneficial, providing additional predictive context to determine which symbol the encrypted value is using.

This approach to decoding would not work well for Chinese, as there would be an excess of symbols to decrypt and the transition matrix would be very sparse making it difficult to precisely quantify the probability of an encoding based on the transitions. Additionally, because Chinese uses characters to represent words (discrete) rather than an alphabet (combinatorial), it would be difficult, if not impossible, to entirely predict each following character (word), even with additional context, that a first order Markov chain does not provide. The current implementation with 53 symbols takes 9000 iterations to converge with only 53 symbols- even if the approach somehow overcame the other limitations, >10,000 symbols would take an excessive time to converge, especially through random singular mapping proposals at each iteration.

Question 3.

a. The standard and collapsed Gibbs sampling updates and log joint probabilities were implemented using the derived equations in (1). In the standard Gibbs sampler, the parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\phi}_k$ were sampled from $\text{Dir}(\boldsymbol{A}_d + \boldsymbol{\alpha})$ and $\text{Dir}(\boldsymbol{B}_k + \boldsymbol{\beta})$, respectively. A topic was sampled for each (doc,word), using the distribution of $P(z_{id}|z_{\sim id}, \boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}, \alpha, \beta)$, and A_{dk} and B_{kw} were iteratively updated. The log joint probability was computed, by taking the log of the joint probability found in (1a), omitting the constant additive term with Gamma functions of hyperparameters, as specified in the provided code. The collapsed Gibbs sampler sampled a topic for each (doc,word) by computing the probability distribution of $P(z_{id}|z_{\sim id}, \alpha, \beta)$ and creating random samples based on the distribution for each (doc,word).

A_{dk} and B_{kw} were then recomputed. The log joint probability was computed by taking the log of the joint probability found in (1c) with parameters integrated out, omitting the constant additive term with Gamma functions of hyperparameters, as specified in the provided code.

Standard Gibbs sampler functions implemented:

```
def update_params(self):
        """
        Samples theta and phi, then computes the distribution of
        z_id and samples counts A_dk, B_kw from it
        """
        # Randomly samples theta and phi from Dir(A_dk+alpha) and Dir(B_kw+beta),
        # respectively
        for d in range(0,self.n_docs):
            self.theta[d]=np.random.dirichlet(np.add(self.A_dk[d].astype(int),
            self.alpha))
        for k in range(0,self.n_topics):
            self.phi[k]=np.random.dirichlet(np.add(self.B_kw[k].astype(int),
            self.beta))

        self.update_topic_doc_words()
        self.sample_counts()

def sample_counts(self):
        """
        For each document and each word, samples from z_id|x_id, theta, phi
        and adds the results to the counts A_dk and B_kw
        """
        self.A_dk.fill(0)
        self.B_kw.fill(0)

        if self.do_test:
            self.A_dk_test.fill(0)
            self.B_kw_test.fill(0)

        # Updates A_dk by finding the number of z_id=k at d according to the
        # probability distribution computed
        for d in range(0,self.n_docs):
            zidsum=np.sum(self.docs_words[d])
            zidsumtest=np.sum(self.docs_words_test[d])
            for k in range(0, self.n_topics):
                self.A_dk[d][k]=self.A_dk[d][k]+int(zidsum*self.theta[d][k])
                self.A_dk_test[d][k]=self.A_dk_test[d][k]+int(zidsumtest*
                    self.theta[d][k])

        # Updates B_kw by finding the number of z_id=k over all w according to the
        # probability distribution computed
        for k in range(0, self.n_topics):
            for w in range(0,self.n_words):
                xidsum=np.sum(self.docs_words[:,w])
                xidsumtest=np.sum(self.docs_words_test[:,w])
                self.B_kw[k][w]=self.B_kw[k][w]+int(xidsum*self.phi[k][w])
                self.B_kw_test[k][w]=self.B_kw_test[k][w]+int(xidsumtest*
                    self.phi[k][w])
        pass
```

```
def update_loglike(self, iteration):
        """
        Updates loglike of the data
        """
        # Computes the log joint, omitting the constant additive term
        # with Gamma functions of hyperparameters

        # Train likelihood
        thetalike=np.sum((self.alpha-1+self.A_dk)*np.log(self.theta))
        philike=np.sum((self.beta-1+self.B_kw)*np.log(self.phi))
        likesum=thetalike+philike
        self.loglike[iteration]=likesum

        # Test likelihood
        thetalike=np.sum((self.alpha-1+self.A_dk_test)*np.log(self.theta))
        philike=np.sum((self.beta-1+self.B_kw_test)*np.log(self.phi))
        likesum=thetalike+philike
        self.loglike_test[iteration]=likesum

        pass
```

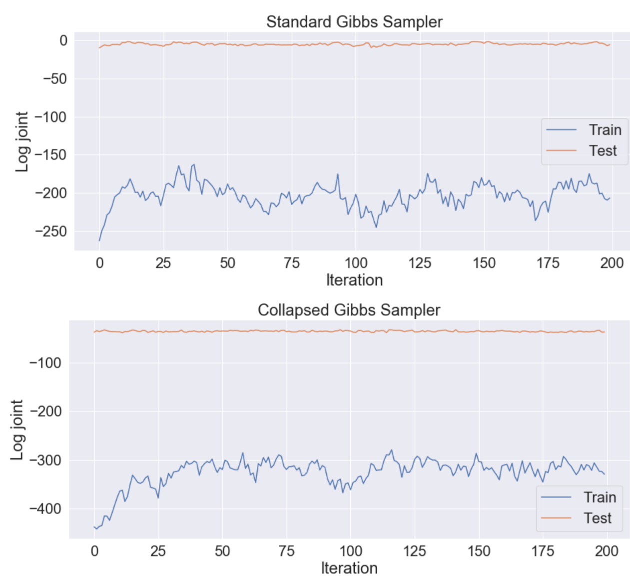
Collapsed Gibbs sampler functions implemented:

```
def update_params(self):
        """
        Computes the distribution of z_id and samples A_dk, B_kw
        """

        # Computes probability of z_id and
        # samples a topic for each (doc, word)
        for d in range(0,self.n_docs):
            for w in range(0,self.n_words):
            # If word i exists in document d (not empty),
            # Computes probability of z_id for each k
                 if self.doc_word_samples[d][w].size > 0:
                     zid=np.zeros(self.n_topics)
                     for k in range(0,self.n_topics):
                         theta=self.A_dk[d][k]+self.alpha-
                             np.count_nonzero(self.doc_word_samples[d][w] == k)
                         theta=theta/(np.sum(self.A_dk[d])+self.n_topics*self.alpha-
                             np.count_nonzero(self.doc_word_samples[d][:]==k))
                         phi=self.B_kw[k][w]+self.beta-
                             np.count_nonzero(self.doc_word_samples[d][w] == k)
                         phi=phi/(np.sum(self.B_kw[k])+self.n_words*self.beta-
                             np.count_nonzero(self.doc_word_samples[:][w]==k))
                         zid[k]=theta*phi
                     zid=zid/np.sum(zid)
                     # Randomly samples according k distribution at (doc,word) and updates
                     randsamp=np.random.choice(np.arange(0,self.n_topics),
                         size=self.doc_word_samples[d][w].size,p=zid)
                     self.doc_word_samples[d][w]=randsamp
                 if self.doc_word_samples_test[d][w].size > 0:
                     zid=np.zeros(self.n_topics)
                     for k in range(0,self.n_topics):
                         theta=self.A_dk_test[d][k]+self.alpha-
```

```
                        np.count_nonzero(self.doc_word_samples_test[d][w] == k)
                    theta=theta/(np.sum(self.A_dk_test[d])+self.n_topics*self.alpha-
                        np.count_nonzero(self.doc_word_samples_test[d][:] == k))
                    phi=self.B_kw_test[k][w]+self.beta-
                        np.count_nonzero(self.doc_word_samples_test[d][w] == k)
                    phi=phi/(np.sum(self.B_kw_test[k])+self.n_words*self.beta-
                        np.count_nonzero(self.doc_word_samples_test[:][w] == k))
                    zid[k]=theta*phi
                zid=zid/np.sum(zid)
                randsamp=np.random.choice(np.arange(0,self.n_topics),
                    size=self.doc_word_samples_test[d][w].size , p = zid)
                self.doc_word_samples_test[d][w]=randsamp

        # Updates A_dk
        self.A_dk=np.zeros((self.n_docs,self.n_topics))
        self.A_dk_test=np.zeros((self.n_docs,self.n_topics))
        for d in range(0,self.n_docs):
            for w in range(0,self.n_words):
                if self.doc_word_samples[d][w].size > 0:
                    for k in range(0,self.n_topics):
                        self.A_dk[d][k]=self.A_dk[d][k]+
                            np.count_nonzero(self.doc_word_samples[d][w] == k)
                if self.doc_word_samples_test[d][w].size > 0:
                    for k in range(0,self.n_topics):
                        self.A_dk_test[d][k]=self.A_dk_test[d][k]+
                            np.count_nonzero(self.doc_word_samples_test[d][w] == k)

        # Updates B_kw
        self.B_kw=np.zeros((self.n_topics,self.n_words))
        self.B_kw_test=np.zeros((self.n_topics,self.n_words))
        for d in range(0,self.n_docs):
            for k in range(0,self.n_topics):
                for w in range(0,self.n_words):
                    if self.doc_word_samples[d][w].size > 0:
                        self.B_kw[k][w]=self.B_kw[k][w]+
                            np.count_nonzero(self.doc_word_samples[d][w]==k)
                    if self.doc_word_samples_test[d][w].size > 0:
                        self.B_kw_test[k][w]=self.B_kw_test[k][w]+
                            np.count_nonzero(self.doc_word_samples_test[d][w]==k)

        pass

    def update_loglike(self, iteration):
        """
        Updates loglike of the data, omitting the constant additive term
        with Gamma functions of hyperparameters
        """
        # Computes Nd and Mk
        Nd=np.zeros(self.n_docs)
        Nd_test=np.zeros(self.n_docs)
        for d in range(0,self.n_docs):
            Nd[d]=np.sum(self.A_dk[d])
            Nd_test[d]=np.sum(self.A_dk_test[d])
        Mk=np.zeros(self.n_topics)
        Mk_test=np.zeros(self.n_topics)
        for k in range(0,self.n_topics):
```

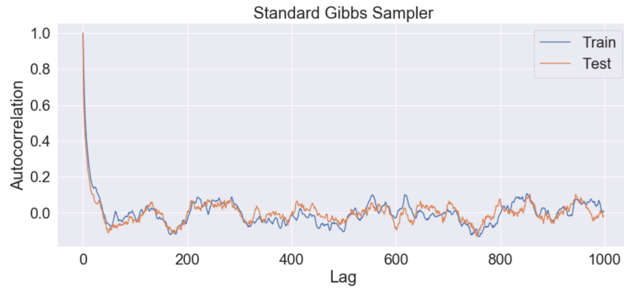
```
            Mk[k]=np.sum(self.B_kw[k])
            Mk_test[k]=np.sum(self.B_kw_test[k])

        # Train likelihood
        Adklike=np.sum(np.sum(gammaln(self.alpha+self.A_dk),axis=0))-
            np.sum(gammaln(Nd+self.n_topics*self.alpha))
        Bkwlike=np.sum(np.sum(gammaln(self.beta+self.B_kw),axis=0))-
            np.sum(gammaln(Mk+self.n_words*self.beta))
        likesum= Adklike+Bkwlike
        self.loglike[iteration]=likesum

        # Test likelihood
        Adklike=np.sum(np.sum(gammaln(self.alpha+self.A_dk_test),axis=0))-
            np.sum(gammaln(Nd_test+self.n_topics*self.alpha))
        Bkwlike=np.sum(np.sum(gammaln(self.beta+self.B_kw_test),axis=0))-
            np.sum(gammaln(Mk_test+self.n_words*self.beta))
        likesum=Adklike+Bkwlike
        self.loglike_test[iteration]=likesum

        pass
```

The standard Gibbs sampler and collapsed Gibbs sampler log joint probabilities are plotted below for toyexample.



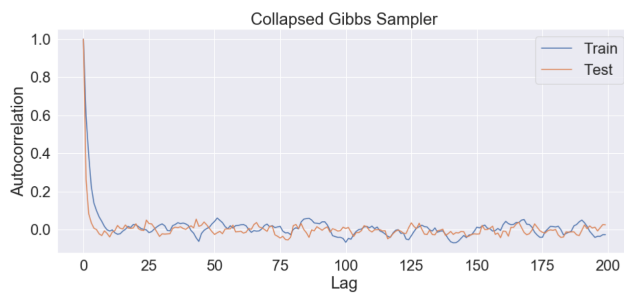
b. Based on the plots of the log joint probability produced by toyexample, the standard Gibbs sampler appears to require approximately 20-25 iterations for burn-in and the collapsed Gibbs sampler appears to require approximately 35 iterations for burn-in, as chosen the point where log joint appears stabilize around an invariant likelihood. A conservative burn-in is chosen, to account for slight fluctuations in burn-in times across runs: 25 for standard and 35 for collapsed.

The autocorrelation was computed across values for lag. For the standard Gibbs sampler, the toy example was run for 3000 iterations in order to reduce the noise in the autocorrelation. The autocorrelation up to a lag of 500 was computed and plotted to better observe the overall pattern of autocorrelation over a large time scale.



From the plot, it can be observed that the autocorrelation displays periodicity with increasing lag. The oscillations appear to consistently reach an autocorrelation of 0 at approximately every 50^{th} value of lag. For the standard Gibbs sampler, a sample of 100 values (a full wavelength) would likely be a representative set of samples from the posterior, as the values within the range of 100 are dependent on each other and the the periodicity of the autocorrelation implies the samples continuously repeat to yield log joint probabilities that are autocorrelated in similar ways at the same intervals.

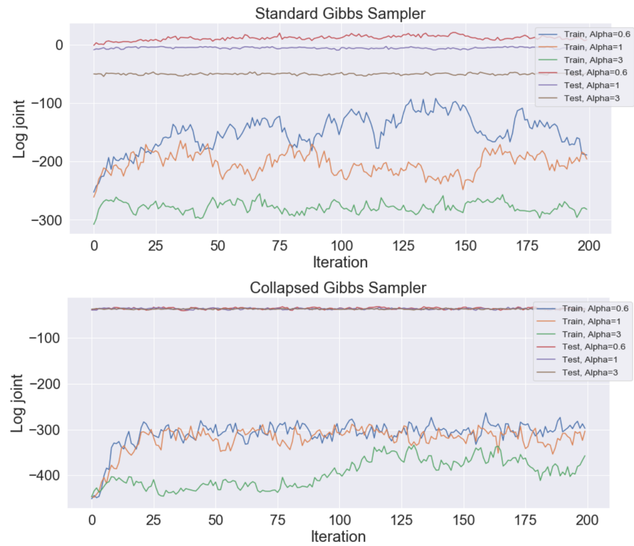
For the collapsed Gibbs sampler, the toy example was run for 3000 iterations in order to reduce the noise in the autocorrelation. The autocorrelation for a lag up to 200 was computed and plotted to observe the pattern of autocorrelation more closely, as the periodicity of the collapsed sampler appears to be shorter than that of the standard sampler.



From the plot, it can be observed that the autocorrelation displays periodicity with increasing lag. The oscillations appear to consistently reach an autocorrelation of 0 at approximately every 12^{th} value of lag. For the collapsed Gibbs sampler, a sample of 50 values (a full wavelength) would likely be a representative set of samples from the posterior, as the values within the range of 50 are dependent on each other and the periodicity of the autocorrelation implies the samples continuously repeat to yield log joint probabilities that are autocorrelated in similar ways at the same intervals.

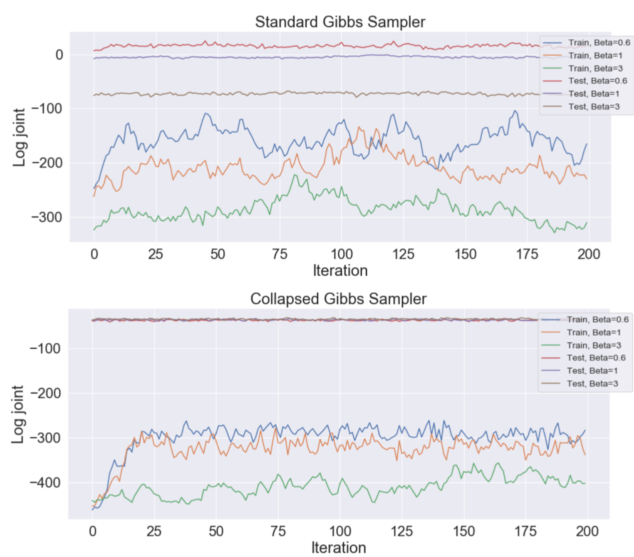
c. The standard Gibbs sampler appears to have a smaller burn-in time than the collapsed Gibbs sampler, meaning it reaches a P_{inv} faster. However, based on the computed autocorrelations, it appears that the collapsed Gibbs sampler converges faster, as it requires a smaller lag to reach an autocorrelation of 0, having a shorter periodicity. Notably, the collapsed sampler's autocorrelation values also fluctuate with less magnitude than than that of the standard sampler. The collapsed sampler may converge faster because it relies only on the priors α and β, rather than having extra dependencies on $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, as in the standard sampler. Thus, with less intermediate probability distributions to sample from, which provide more variation in the values sampled for z_{id}, the collapsed sampler has less variations in intermediate steps and is able to converge faster than the standard sampler.

d. The standard Gibbs sampler and collapsed Gibbs sampler log joint probabilities are plotted below for varying values of α.



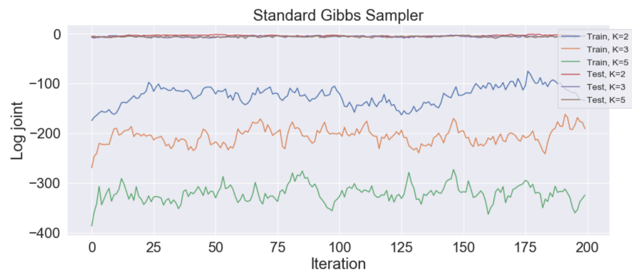
For both samplers, decreasing the value of α increases the log joint probability of both the train and test data, increasing performance on the model, and increasing the value of α decreases the performance of the model. Decreasing α has the effect of creating a more sparse and condensed distribution of $\boldsymbol{\theta}_d$ in the standard sampler and directly onto z_{id} in the collapsed sampler. Due to the lack of intermediate $\boldsymbol{\theta}_d$ in the collapsed sampler, the effect of modifying α on the joint likelihood has a smaller magnitude than that of the standard sampler. As previously explained in (1e), a lower value of α is likely to be more favorable, as a document is more likely to contain a smaller subset of concentrated topics from within a larger topic space.

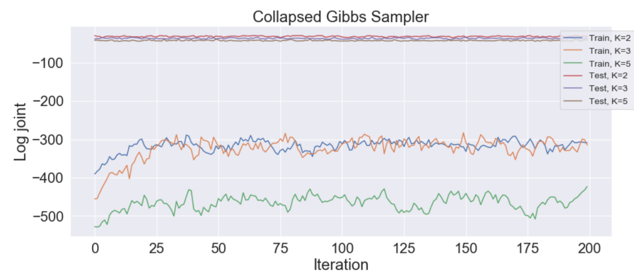
The standard Gibbs sampler and collapsed Gibbs sampler log joint probabilities are plotted below for varying values of β.



For both samplers, decreasing the value of β increases the log joint probability of both the train and test data, increasing performance on the model, and increasing the value of β decreases the performance of the model. Decreasing β has the effect of creating a more sparse and condensed distribution of $\boldsymbol{\phi}_k$ in the standard sampler and directly onto z_{id} in the collapsed sampler. Due to the lack of intermediate $\boldsymbol{\phi}_k$ in the collapsed sampler, the effect of modifying β on the joint likelihood has a smaller magnitude than that of the standard sampler. As previously explained in (1e), a lower value of β is likely to be more favorable, as a topic is more likely to be described using a smaller subset of concentrated words from within a larger word space.

The standard Gibbs sampler and collapsed Gibbs sampler log joint probabilities are plotted below for varying values of K.





For both samplers, decreasing the value of K increases the log joint probability of both the train and test data, increasing performance on the model, and increasing the value of K decreases the performance of the model. Decreasing K has the effect of decreasing the topic space and thus yields similar results to decreasing α and β- rather than condensing on particular topics, some topics are omitted from the topic space all together. Modifying K acts upon both $\boldsymbol{\theta}_d$ and $\boldsymbol{\phi}_k$, therefore causing an even greater magnitude of log joint difference in the standard Gibbs sampler between different values of K. Due to the lack of intermediate $\boldsymbol{\theta}_d$ and $\boldsymbol{\phi}_k$ in the collapsed sampler, the effect of modifying K on the joint likelihood has a smaller magnitude than that of the standard sampler. A lower value of K is likely to be more favorable, as a smaller topic space allows for less variation and for documents and words to be encompassed within a smaller space of broader 'main' topics, rather than a larger space of specific 'sub' topics, making them easier to categorize with high probability.

e. In order to reduce the NIPS dataset, the tf-idf algorithm was implemented to compute the weight of each word in each document, based on its importance, determined by how frequently it occurs in the document and across all documents. Several different weighing schemes were experimented with, with the following scheme yielding the best results, identifying keywords with the highest weights:

$$tf(t,d) = \frac{f_{t,d}}{\{t' \in T : t' \in d\}}$$
$$idf(t,D) = \log \frac{N}{1 + |\{d \in D : t \in D\}|}$$
$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

tf is calculated by dividing the frequency of a particular word in a particular document ($f_{t,d}$) by the number of different words that appear in the document. *idf* is calculated by taking the log of the number of documents (N) by the sum of the number of documents containing the word t and 1. $tfidf$, corresponding to the weight of each word in each document, is calculated by taking the product of tf and idf. The weights of each word are then summed across documents to yield a final weight for each word across all documents. Words with the 500 highest weights are then used to prune the data.

```
train, test= read_data('./nips.data')
nips=np.concatenate((train, test))
vocab=pd.read_csv('./nips.vocab')
vocab=vocab.to_numpy()

# Calculates tf
def tfcalc(t,d):
    ftd=d[t]
    numterms=np.count_nonzero(d[:] != 0)
    return ftd/numterms

# Calculates idf
def idfcalc(t,D):
    N=len(D)
    tinD=abs(np.count_nonzero(D[:,t] != 0))
    return np.log(N/(1+tinD))

# Calculates weight matrix for each (doc,word)
def tfidfcalc(docs):
    tfidf=np.zeros((docs.shape))
    for i in range(0,len(docs)):
        for j in range(0,len(docs[i])):
            if docs[i][j] != 0:
                tf=tfcalc(j,docs[i])
                idf=idfcalc(j,docs)
                tfidf[i][j]=tf*idf
    return tfidf

# Sums words across documents, prints highest weighted words,
# and returns their indices
def wordweights(tfidfmatrix,num):
    ww=np.sum(tfidfmatrix, axis=0)
    temp = np.argpartition(ww, -num)[-num:]
    index = temp[np.argsort((-ww)[temp])]
    for i in range(0,num):
        print(vocab[index[i]])
    return index

# Calculates word weights and prunes nips dataset
matrix=tfidfcalc(nips)
index=wordweights(matrix,500)
newnips=np.zeros((len(nips.shape),500))
for i in range(0,nips.shape[1]):
    if i not in index:
        newnips=np.delete(nips,i,1)
```