CPSC532W: Homework 1

Name: Adam Jozefiak, Student No.: 27458158

1. We show that Gamma distribution is conjugate to the Poisson distribution. Let

$$\lambda \sim \text{Gamma}(\alpha, \beta)$$
$$x_1, \ldots, x_N \sim \text{Poisson}(\lambda)$$

Then by Bayes rule, the posterior distribution $p(\lambda|\mathbf{x})$ can be written as

$$p(\lambda|x_{1:N}) = \frac{p(x_{1:N}|\lambda)p(\lambda)}{p(x_{1:N})} \propto p(x_{1:N}|\lambda)p(\lambda)$$

where the last proportionallity follows by the fact that $p(x_{1:N}) = \int p(x_{1:N}|\lambda)p(\lambda)d\lambda$ is constant.

We note the probability density functions of Gamma(α, β) and Poisson(λ):

$$p(\gamma) = \tfrac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}$$

$$p(x_{1:N}) = \textstyle\prod_{i=1}^N \tfrac{\lambda^{x_i}e^{-\lambda}}{x_i!}$$

Then,

$$\begin{aligned}
p(\lambda|x_{1:N}) &\propto p(x_{1:N}|\lambda)p(\lambda) && \text{as argued earlier} \\
&= \left(\textstyle\prod_{i=1}^N \tfrac{\lambda^{x_i}e^{-\lambda}}{x_i!}\right)\left(\tfrac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}\right) \\
&= \tfrac{\beta^\alpha}{\prod_{i=1}^N x_i!}\lambda^{\sum_{i=1}^N x_i+\alpha-1}e^{-(n+\beta)\lambda} \\
&\propto \lambda^{\sum_{i=1}^N x_i+\alpha-1}e^{-(n+\beta)\lambda} && \text{by } \lambda \text{ not appearing in } \tfrac{\beta^\alpha}{\prod_{i=1}^N x_i!}
\end{aligned}$$

Therefore, up to a normalizing constant, $p(\lambda|x_{i:N}) \propto \lambda^{\sum_{i=1}^N x_i+\alpha-1}e^{-(n+\beta)\lambda}$ and hence $p(\lambda|x_{1:N}) \sim$ Gamma$(\sum_{i=1}^N x_i + \alpha, n+\beta)$. Therefore, the a prior Gamma distribution is a conjugate prior for a likelihood Poisson distribution.

2. Let x and x' be vectors of random variables. Then the Gibbs transition operator, for the change in the k^{th} variable or index of x, is defined as $T_k(x, x') = p(x'_k|x_{-k})\prod_{i\neq k}\mathbb{I}(x_i = x'_i)$. Then we can see that the Gibbs transition operator satisfies the detailed balance equattion by hte following argument

$$
\begin{aligned}
& p(x)T(x,x') \\
= \;& p(x)p(x'_k|x_{-k})\textstyle\prod_{i\neq k}\mathbb{I}(x_i = x'_i) \\
= \;& p(x_k|x_{-k})p(x_{-k})p(x'_k|x_{-k})\textstyle\prod_{i\neq k}\mathbb{I}(x_i = x'_i) && \text{by the product rule} \\
= \;& p(x')p(x_k|x_{-k})\textstyle\prod_{i\neq k}\mathbb{I}(x_i = x'_i) && \text{by the product rule on } p(x') = p(x'_k|x_{-k})p(x_{-k}) \\
= \;& p(x')p(x_k|x'_{-k})\textstyle\prod_{i\neq k}\mathbb{I}(x_i = x'_i) && \text{by } p(x_k|x_{-k})\textstyle\prod_{i\neq k}\mathbb{I}(x_i = x'_i) > 0 \Rightarrow x'_{-k} = x_{-k} \\
= \;& p(x')T(x',x)
\end{aligned}
$$

This completes the proof of the Gibbs transition operator satisfying the detailed balance equation and therefore Gibbs transition operator can be interpreted as a Metropolis-Hastings transition operator that always accepts.

3. My solution to this problem is in Julia, based off the sample code. I found that the probability of it being cloudy given that the grass is wet is 57.58% by enumerating all possible world states and conditioning by counting which proportion are cloudy given the grass is wet. Using ancestral sampling and rejection, with 10000 successful samples, I found that the probability that it is cloudy given the grass is wet is 57.50% while 35.43% of samples were rejected. Using Gibbs sampling I found that the probability that it is cloudy given that the grass is wet is 58.66%. In my implementation, an index of 2 indicates a "true" assignment and an index 1 indicates a "false" assignment to a random variable.

   Below is my code for computing the probability by enumerating all possible world states and conditioning by counting which proportion are cloudy given the grass is wet:

```julia
## condition and marginalize:
p_C_given_W = 0.0
p_C_and_W = 0.0
p_W =  0.0
for c in 1:2
    for s in 1:2
        for r in 1:2
            global p_W += p[c,s,r,2]
        end
    end
end
for s in 1:2
    for r in 1:2
        global p_C_and_W += p[2,s,r,2]
    end
end
p_C_given_W = p_C_and_W / p_W

println("There is a ", p_C_given_W*100, "% chance it is cloudly given the grass is wet.")
```

   Below is my code for computing the probability by ancestral sampling and rejection:

```
##2. ancestral sampling and rejection:
num_samples = 10000
samples = zeros(num_samples)
rejections = 0
i = 1
while i <= num_samples
    C = 0
    S = 0
    R = 0
    W = 0
    p = rand()
    if p > p_C(1)
        C = 2
    else
        C = 1
    end
    p = rand()
    if p > p_S_given_C(1,C)
        S = 2
    else
        S = 1
    end
    p = rand()
    if p > p_R_given_C(1,C)
        R = 2
    else
        R = 1
    end
    p = rand()
    if p > p_W_given_S_R(1,S,R)
        W = 2
    else
        W = 1
    end
    if W == 2
        global samples[i] = C-1
        global i += 1
    else
        global rejections += 1
    end
end


println("The chance of it being cloudy given the grass is wet is ", (sum(samples)./num_samples)*100, "%.")
println(100*rejections/(num_samples+rejections), "% of the total samples were rejected.")
```

Below is my code for computing the probability by Gibbs sampling:

```
##gibbs sampling
# num_samples = 10000
num_samples = 10000
samples = zeros(num_samples)
state = ones(Int64,4)
#c,s,r,w, set w = True
state[4] = 2
i = 1
while i <= num_samples

    # Sample r
    prob = rand()
    if prob > p_R_given_C_S_W[state[1],state[2],1,state[4]]
        state[3] = 2
    else
        state[3] = 1
    end

    # Sample s
    prob = rand()
    if prob > p_S_given_C_R_W[state[1],1,state[3],state[4]]
        state[2] = 2
    else
        state[2] = 1
    end

    # Sample c
    prob = rand()
    if prob > p_C_given_S_R[1,state[2],state[3]]
        state[1] = 2
    else
        state[1] = 1
    end
    samples[i] = state[1]
    global i += 1

end


println("The chance of it being cloudy given the grass is wet is, ",100*sum(samples .- 1)/num_samples ,"%.")
```

4. • We first derive the updates for Metropolis Hastings sampling of the $\hat{t}$ block, that is we want to sample from $p(\hat{t}|\hat{x}, t, x, w, \sigma^2, \alpha)$.
Let the proposal distribution be $\hat{t}'|\hat{t} \sim$ Normal$(\hat{t}, \sigma^2)$. Then $\hat{t}'|\hat{t}$ and $\hat{t}|\hat{t}'$ are symmetric, i.e $q(\hat{t}'|\hat{t}) = q(\hat{t}|\hat{t}')$, and so $\frac{q(\hat{t}|\hat{t}')}{q(\hat{t}'|\hat{t})} = 1$. Then after sampling $\hat{t}'$ from $q(\hat{t}'|\hat{t})$ the accepting probability is:

$$
\begin{aligned}
& A(\hat{t}'|\hat{t}) \\
= \quad & \min(1, \tfrac{p(\hat{t}'|\hat{x},t,x,w,\sigma^2,\alpha)q(\hat{t}|\hat{t}')}{p(\hat{t}|\hat{x},t,x,w,\sigma^2,\alpha)q(\hat{t}'|\hat{t})}) \\
= \quad & \min(1, \tfrac{p(\hat{t}'|\hat{x},t,x,w,\sigma^2,\alpha)}{p(\hat{t}|\hat{x},t,x,w,\sigma^2,\alpha)}) && \text{by the symmetry of the proposal distribution} \\
= \quad & \min(1, \tfrac{p(\hat{t}',\hat{x},t,x,w,\sigma^2,\alpha)p(\hat{x},t,x,w,\sigma^2,\alpha)}{p(\hat{t},\hat{x},t,x,w,\sigma^2,\alpha)p(\hat{x},t,x,w,\sigma^2,\alpha)}) && \text{by the product rule} \\
= \quad & \min(1, \tfrac{p(\hat{t}',\hat{x},t,x,w,\sigma^2,\alpha)}{p(\hat{t},\hat{x},t,x,w,\sigma^2,\alpha)}) \\
= \quad & \min(1, \tfrac{\prod_{i=1}^N p(t_i|x,w,\sigma^2)p(w|\alpha)p(\hat{t}'|\hat{x},w,\sigma^2)}{\prod_{i=1}^N p(t_i|x,w,\sigma^2)p(w|\alpha)p(\hat{t}|\hat{x},w,\sigma^2)}) \\
= \quad & \min(1, \tfrac{p(\hat{t}'|\hat{x},w,\sigma^2)}{p(\hat{t}|\hat{x},w,\sigma^2)})
\end{aligned}
$$

Therefore, in the Metropolis Hastings sampling block for $\hat{t}$, we sample $\hat{t}'$ from $\text{Normal}(\hat{t}, \sigma^2)$ and we accept $\hat{t}'$ with probability $\min(1, \frac{p(\hat{t}'|\hat{x},w,\sigma^2)}{p(\hat{t}|\hat{x},w,\sigma^2)})$.
Next, we derive the updates for Metropolis Hastings sampling of the w block, that is we want to sample from $p(w|\hat{t}, \hat{x}, t, x, \sigma^2, \alpha)$. We can treat the $\hat{t}$ and $\hat{x}$ as t and x and so we will derive the updates for Metropolis Hastings sampling of $p(w|t, x, \sigma^2, \alpha)$.
Let the proposal distribution be $w'|w \sim \text{Normal}(w, \alpha I)$. Then $w'|w$ and $w|w'$ are symmetric, i.e $q(w'|w) = q(w|w')$, and so $\frac{q(w|w')}{q(w'|w)} = 1$. Then after sampling w' from $q(w'|w)$ the accepting probability is:

$$
\begin{aligned}
& A(w'|w) \\
= \quad & \min(1, \tfrac{p(w'|t,x,\sigma^2,\alpha)q(w|w')}{p(w|t,x,\sigma^2,\alpha)q(w'|w)}) \\
= \quad & \min(1, \tfrac{p(w'|t,x,\sigma^2,\alpha)}{p(w|t,x,\sigma^2,\alpha)}) && \text{by the symmetry of the proposal distribution} \\
= \quad & \min(1, \tfrac{p(w',t,x,\sigma^2,\alpha)p(t,x,\sigma^2,\alpha)}{p(w,t,x,\sigma^2,\alpha)p(t,x,\sigma^2,\alpha)}) && \text{by the product rule} \\
= \quad & \min(1, \tfrac{p(w',t,x,\sigma^2,\alpha)}{p(w,t,x,\sigma^2,\alpha)}) \\
= \quad & \min(1, \tfrac{\prod_{i=1}^N p(t_i|x,w,\sigma^2)p(w|\alpha)}{\prod_{i=1}^N p(t_i|x,w',\sigma^2)p(w'|\alpha)})
\end{aligned}
$$

Therefore, in the Metropolis Hastings sampling block for w, we sample w' from $\text{Normal}(w, \alpha I)$ and we accept w' with probability $\min(1, \frac{\prod_{i=1}^N p(t_i|x,w,\sigma^2)p(w|\alpha)}{\prod_{i=1}^N p(t_i|x,w',\sigma^2)p(w'|\alpha)})$.

- To perform pure Gibbs sampling for w and $\hat{t}$ we would like to first sample w from $p(w|\hat{t}, \hat{x}, t, x, \sigma^2, \alpha)$. As argued earlier, we can treat $\hat{t}$ and $\hat{x}$ as being part of t and x and so we might as well sample w from $p(w|t, x, \sigma^2, \alpha)$. Then we have that

$$\begin{aligned} & p(w|t,x,\sigma^2,\alpha) \\ = \ & \tfrac{p(w,t,x,\sigma^2,\alpha)}{p(t,x,\sigma^2,\alpha)} && \text{by the product rule} \\ \propto \ & p(w,t,x,\sigma^2,\alpha) \\ = \ & \textstyle\prod_{i=1}^N p(t_i|x_i, w\sigma^2)p(w|\alpha) \end{aligned}$$

Hence, in Gibbs sampling, we can sample w proportionally to the joint distribution or specifically the distribution $\prod_{i=1}^N p(t_i|x_i, w\sigma^2)p(w|\alpha)$. As will be shown in the next bullet point, this distribution is the posterior of w and it is

$$w|t,x,\sigma^2,\alpha \sim \text{ Normal}([\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T]^{-1}\frac{1}{\sigma^2}xt, [\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T]^{-1})$$

Next, we need to sample $\hat{t}$ which requires sampling from $p(\hat{t}|\hat{x}, t, x, w, \sigma^2, \alpha)$. We can see that

$$\begin{aligned} & p(\hat{t}|\hat{x},t,x,w,\sigma^2,\alpha) \\ = \ & \tfrac{p(\hat{t},\hat{x},t,x,w,\sigma^2,\alpha)}{p(\hat{x},t,x,w,\sigma^2,\alpha)} && \text{by the product rule} \\ \propto \ & p(\hat{t},\hat{x},t,x,w,\sigma^2,\alpha) \\ = \ & \textstyle\prod_{i=1}^N p(t_i|x_i,w,\sigma^2)p(w|\alpha)p(\hat{t}|\hat{x},w,\sigma^2) \\ \propto \ & p(\hat{t}|\hat{x},w,\sigma^2) \end{aligned}$$

Hence, Gibbs sampling, we can sample $\hat{t}$ proportionally to the distriubtion $p(\hat{t}|\hat{x}, w, \sigma^2)$ which has distribution Normal$(w^T\hat{x}, \sigma^2)$.

- We compute the analytic form of the posterior predictive: $p(\hat{t}|\hat{x}, t, x, \sigma^2, \alpha)$. We begin by noting the following

$$\begin{aligned} & p(\hat{t}|\hat{x},t,x,\sigma^2,\alpha) \\ = \ & \textstyle\int p(\hat{t}|\hat{x},t,x,\sigma^2,w,\alpha)ntp(\hat{t},w|\hat{x},t,x,\sigma^2,\alpha)dw && \text{by the sum rule} \\ = \ & \textstyle\int p(\hat{t}|\hat{x},t,x,\sigma^2,w,\alpha)p(w|\hat{x},t,x,\sigma^2,\alpha)dw && \text{by the product rule} \\ = \ & \textstyle\int p(\hat{t}|\hat{x},\sigma^2,w)p(w|t,x,\sigma^2,\alpha)dw \end{aligned}$$

Where the last line above follows by the fact that $p(\hat{t}|\hat{x}, t, x, \sigma^2, w, \alpha) = p(\hat{t}|\hat{x}, \sigma^2, w)$ and $p(w|\hat{x}, t, x, \sigma^2, \alpha) = p(w|t, x, \sigma^2, \alpha)$, the latter following by the fact that w can only have a dependency on $\hat{x}$ if $\hat{t}$ is known.

Next, we will argue that both $\hat{t}|\hat{x},\sigma^2,w$ and $w|t,x,\sigma^2,\alpha$ are Gaussian random variables. Firstly, $\hat{t}|\hat{x},\sigma^2,w \sim$ Normal$(w^T\hat{x},\sigma^2)$ by the likelihood given in the question.
As for $w|t,x,\sigma^2,\alpha$ we note that

$$w|\alpha \sim \text{ Normal}(0,\alpha I)$$

$$t|w,x,\sigma^2 \sim \text{ Normal}(w^T x,\sigma^2)$$

Then we note that

$$p(w|t,x,\sigma^2,\alpha) \propto p(t|w,x,\sigma^2,\alpha)p(w,x|\sigma^2,\alpha) \propto p(t|w,x,\sigma^2)p(w|\alpha)$$

Then by 2.116, page 93, in Pattern Recognition and MAchine Learning by Bishop it follows that

$$w|t,x,\sigma^2,\alpha \sim \text{ Normal}([\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T]^{-1}\frac{1}{\sigma^2}xt, [\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T]^{-1})$$

Then by the earlier observations that

$$p(\hat{t}|\hat{x},t,x,\sigma^2,\alpha) \quad = \quad \int p(\hat{t}|\hat{x},\sigma^2,w)p(w|t,x,\sigma^2,\alpha)dw$$

and

$$\hat{t}|\hat{x},\sigma^2,w \quad \sim \quad \text{Normal}(w^T\hat{x},\sigma^2)$$

and by linear combination rules of Gaussian random variables we have that

$$\hat{t}|\hat{x},t,x,\sigma^2,\alpha \sim \text{ Normal}([(\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T)^{-1}\frac{1}{\sigma^2}xt]^T\hat{x}, \hat{x}^T(\frac{1}{\alpha}I + \frac{1}{\sigma^2}xx^T)^{-1}\hat{x} + \sigma^2)$$

This completes the derivation of the analytic for mof hte posterior predictive.

5. For this problem I worked off the python starter code. Below is the implementation of my joint log likelihood function:

```python
def joint_log_lik(doc_counts, topic_counts, alpha, gamma):
    """
    Calculate the joint log likelihood of the model

    Args:
        doc_counts: n_docs x n_topics array of counts per document of unique topics
        topic_counts: n_topics x alphabet_size array of counts per topic of unique words
        alpha: prior dirichlet parameter on document specific distributions over topics
        gamma: prior dirichlet parameter on topic specific distribuitons over words.
    Returns:
        jll: the joint log likelihood of the model
    """
    num_docs = doc_counts.shape[0]
    num_topics = topic_counts.shape[0]
    alphabet_size = topic_counts.shape[1]

    jll = 0.0
    for i in range(num_docs):
        jll -=  num_topics * np.log(num_topics*alpha + np.sum(doc_counts[i,:]))
        for j in range(num_topics):
            jll += np.log((alpha + doc_counts[i,j]))

    for i in range(num_topics):
        jll -= alphabet_size * np.log(alphabet_size*gamma + np.sum(topic_counts[i,:]))
        for j in range(alphabet_size):
            jll += np.log((gamma + topic_counts[i,j]))

    return jll
```

Below is the implementation of my sampling function:

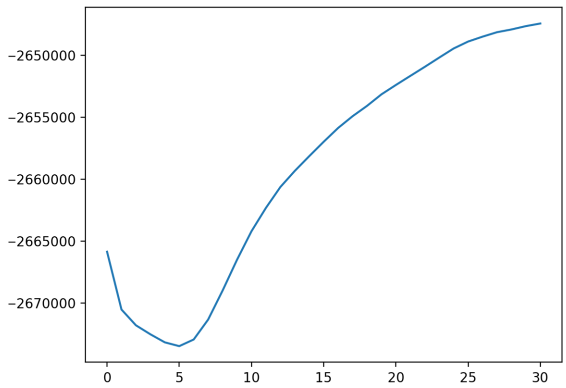
```python
    # Number of topics
    num_topics = topic_counts.shape[0]

    # Number of words
    num_words = len(words)

    #alphabet_size
    alphabet_size = topic_counts.shape[1]

    # sample topic assignment for each of the n words in turn
    for n in range(num_words):

        # Document of nth word
        doc = document_assignment[n]
        # Topic assignment of nth word
        topic = topic_assignment[n]
        # alphabet-word of nth word
        word = words[n]

        prob_sum = 0.0
        prob_dist = np.zeros(num_topics)

        # sample a new topic
        for k in range(num_topics):
            # compute probability of p(z_i,d <= k|...)
            if k == topic:
               t_1 = (alpha + doc_counts[doc,k]-1)/(num_topics*alpha + np.sum(doc_counts[doc,:])-1)
               t_2 = (gamma + topic_counts[k,word]-1) / (alphabet_size*gamma + np.sum(topic_counts[k,:])-1)
            else:
               t_1 = (alpha + doc_counts[doc,k])/(num_topics*alpha + np.sum(doc_counts[doc,:])-1)
               t_2 = (gamma + topic_counts[k,word]) / (alphabet_size*gamma + np.sum(topic_counts[k,:])-1)

            prob_sum += (t_1*t_2)
            prob_dist[k] = t_1*t_2
```

```python
        prob_dist = np.cumsum(prob_dist / np.sum(prob_dist))
        prob = np.random.uniform(0,1)

        for k in range(num_topics):

            # we sample z_id to be topic k
            if prob <= prob_dist[k]:

                # Update topic_assignment
                topic_assignment[n] = k

                topic_counts[topic,word] -= 1
                topic_counts[k,word] += 1

                doc_counts[doc,topic] -= 1
                doc_counts[doc,k] += 1

                topic_N[topic] -= 1
                topic_N[k] += 1

                break

    return (topic_assignment, topic_counts, doc_counts, topic_N)
```

Below is my implementation of determing the most common words for each topic and determining the most similar document to document 1:

```python
### find the 10 most probable words of the 20 topics:
fstr = ''
for k in range(topic_counts.shape[0]):
    fstr += str(k+1) + ':'
    indices = np.argsort(topic_counts[k,:])[-10:]
    indices = indices[::-1]
    for i in range(len(indices)):
        fstr += ', ' + WO[i][0]
    fstr += '\n'

with open('most_probable_words_per_topic','w') as f:
    f.write(fstr)



#most similar documents to document 0 by cosine similarity over topic distribution:
#normalize topics per document and dot product:

doc_0 = doc_counts[0,:] / np.sum(doc_counts[0,:])
closest_doc = 1
sim = 0
for k in range(doc_counts.shape[0]):
    if k != 1:
        doc = doc_counts[k,:] / np.sum(doc_counts[k,:])
        if np.dot(doc_0,doc) > sim:
            closest_doc = k
            sim = np.dot(doc_0,doc)



fstr = str(closest_doc)

with open('most_similar_titles_to_0','w') as f:
    f.write(fstr)
```

Below is a plot of my log likelihood with respect to the number of iterations:



I found that