Coursework 3: document models

Original (http://mlg.eng.cam.ac.uk/teaching/4f13/1819/cw/coursework3.pdf) by Carl Rasmussen and Manon Kok for CUED course 4f13 (http://mlg.eng.cam.ac.uk/teaching/4f13/1819/). This version adapted by Damon Wischik.

This coursework involves aggregating, summarizing, and joining datasets. This may be done with straight Python, or with MATLAB-style manipulations using `numpy`, or with `pandas` dataframes. If you anticipate future work in machine learning and data science then you should learn to use `pandas` dataframes, and you may find it helpful to follow the walkthrough in Section 3 (https://notebooks.azure.com/djw1005/libraries/cl-scicomp/html/3.%20Working%20with%20data.ipynb) of IA *Scientific Computing*. If you prefer not to use dataframes, and you have questions about how they are being used in the code snippets below, ask your classmates or Dr Wischik.

What to submit. Your answers should contain an explanation of what you do, and 2–4 central commands to achieve it. Complete listings are unnecessary. The focus of your answer should be *interpretation:* explain what the numerical values and graphs you produce *mean,* and why they are as they are. The text of your answer to each question should be no more than a paragraph or two.

In [1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import scipy.io
import pandas
import requests, io
```

Data import

The data is provided as `http://mlg.eng.cam.ac.uk/teaching/4f13/1112/cw/kos_doc_data.mat`. It contains two matrices A and B for training and testing respectively, both matrices with 3 columns: document ID, word ID, and word count. The words themselves are the vector V, where e.g. `V[840]='bush'`. The following snippet reads in the data, and converts A and B to dataframes.

In [2]:
```
r = requests.get('http://mlg.eng.cam.ac.uk/teaching/4f13/1112/cw/kos_doc_data.mat')
with io.BytesIO(r.content) as f:
    data = scipy.io.loadmat(f)
    V = np.array([i[0] for i in data['V'].squeeze()])
    A,B = [pandas.DataFrame({'doc_id': M[:,0]-1, 'word_id': M[:,1]-1, 'count': M[:,2]},
                            columns=['doc_id','word_id','count'])
           for M in (data['A'],data['B'])]
```

Question (a): simple categorical model

Suppose we model words in a document as independent samples from a categorical distribution with parameter β, where β_v is the probability of word $v \in V$. Using A as the training set, find the maximum likelihood estimator $\hat{\beta}$, and plot the 20 most-probable words in a histogram. What is the log probability of the test document `doc_id=2527`, given $\hat{\beta}$? Briefly interpret your answer.

Note: you can plot a histogram with

```
with plt.rc_context({'figure.figsize': (5,8)}):           # set plot size
    plt.barh(np.arange(20), top_20_probs, align='center') # draw bars
plt.yticks(np.arange(20), top_20_words)                   # label the y axis
plt.xlabel(r'$\hat{\beta}$')                              # label the x axis
plt.gca().invert_yaxis()                                  # optionally, flip the y-axis
plt.show()
```

Question (b): Bayesian inference

For the categorical model in part (a), use Bayesian inference to find the posterior distribution of β given the training set A, using a symmetric Dirichlet distribution with concentration parameter $\alpha = 0.1$ as prior. Let $\tilde{\beta}_v$ be the posterior predictive probability of word $v \in V$, i.e. the posterior probability that a newly chosen word is v. Derive an expression for $\tilde{\beta}_v$, and compare it to $\hat{\beta}_v$. Explain the implications, both for common and for rare words.

Hint: $\Gamma(z+1) = z\,\Gamma(z)$.

Question (c): interpretation

In information theory, the *self-information* of a document w is defined as $i(w) = -\log_2 p(w)$, where $p(\cdot)$ is the probability mass function for the document generating model that you have fitted. The self-information can be interpreted as the number of bits needed to encode or transmit w. The number of bits needed per word is thus $i(w)/n$. In text modelling, it is more common to use the terms *perplexity* for $2^{i(w)}$, and *per-word perplexity* for $2^{i(w)/n}$. Loosely speaking, if the per-word perplexity is g then the uncertainty in the next word is the same as the uncertainty in a g-sided die.

For the trained Bayesian model from part (b), what is the per-word perplexity of the test document `doc_id=2000` ? Plot a histogram showing the distribution of per-word perplexity over all the test documents (using `plt.hist` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html?highlight=matplotlib%20pyplot%20hist#matplotlib.pyplot.hist)). Pick out two documents, one with high per-word perplexity and one with low per-word perplexity, show their contents, and interpret the difference between them.

Question (d): Gibbs sampler for the mixture-of-multinomials model

The Bayesian mixture-of-multinomials model can be described by the following code:

In [4]:
```
def bmm_generate(doc_length, V, α, γ, K):
    # doc_length = [num words in doc1, num words in doc2, ...]
    θ = np.random.dirichlet(α * np.ones(K))              # prob dist over document classes {1,...,K}
    β = np.random.dirichlet(γ * np.ones(len(V)), size=K) # for each doc class, a prob dist over words
    z = np.random.choice(K, p=θ, size=len(doc_length))   # doc class of each document
    return [np.random.choice(V, p=β[zd], size=nd) for zd,nd in zip(z, doc_length)]

for doc in bmm_generate(doc_length=[5,2,4], V=V, α=10, γ=.1, K=20):
    print(doc)
```

```
['critic' 'indymedia' 'susan' 'citizenship' 'cycles']
['cool' 'celebrity']
['jennings' 'quarter' 'token' 'governance']
```

The following code implements a collapsed Gibbs sampler. Complete the line that defines `logp` . In each sweep, the Gibbs sampler produces a sample of document classes, and this sample induces a posterior predictive distribution for the probability of each class. Plot how this distribution evolves as a function of the number of Gibbs sweeps. How many iterations does it take to converge?

```
def bmm_gibbs(doc_label, word_id, count, W, α, γ, K):
    # doc_labels = distinct values of doc_label
    # doc_index = a list as long as doc_label
    #             such that doc_labels[doc_index[j]] = doc_label[j]
    doc_labels, doc_index = np.unique(doc_label, return_inverse=True)

    # z[i] = class of document i, where i enumerates the distinct doc_labels
    # doc_count[k] = number of documents of class k
    z = np.random.choice(K, len(doc_labels))
    doc_count = np.zeros(K, dtype=int)
    for k in z: doc_count[k] += 1

    # occurrences[k,w] = number of occurrences of word_id w in documents of class k
    # word_count[k] = total number of words in documents of class k
    x = pandas.DataFrame({'doc_class': z[doc_index], 'word_id': word_id, 'count': count}) \
        .groupby(['doc_class', 'word_id']) \
        ['count'].apply(sum) \
        .unstack(fill_value=0)
    occurrences = np.zeros((K, len(V)))
    occurrences[x.index.values.reshape((-1,1)), x.columns.values] = x
    word_count = np.sum(occurrences, axis=1)

    while True:
        for i in range(len(doc_labels)):

            # get the words,counts for document i
            # and remove this document from the counts
            w,c = word_id[doc_index==i].values, count[doc_index==i].values
            occurrences[z[i], w] -= c
            word_count[z[i]] -= sum(c)
            doc_count[z[i]] -= 1

            # Find the log probability that this document belongs to class k, marginalized ov
er θ and β
            logp = [... for k in range(K)]
            p = np.exp(logp - np.max(logp))
            p = p/sum(p)

            # Assign this document to a new class, chosen randomly, and add back the counts
            k = np.random.choice(K, p=p)
            z[i] = k
            occurrences[k, w] += c
            word_count[k] += sum(c)
            doc_count[k] += 1

        yield np.copy(z)
```

The Gibbs sampler may be run as follows:

In [ ]:
```
g = bmm_gibbs(A['doc_id'], A['word_id'], A['count'], W=len(V), α=10, γ=.1, K=20)
NUM_ITERATIONS = 20
res = np.stack([next(g) for _ in range(NUM_ITERATIONS)])
# this produces a matrix with one row per iteration and a column for each unique doc_id
```

Question (e): interpretation

Let $\alpha = 10, \gamma = 0.1, K = 20$. Run the Gibbs sampler until it converges, and find the posterior predictive probabilities for topics, and for words within each topic. For each the 8 most popular topics, print the probability of the topic and the 8 most probable words and their probabilities. Display probabilities in *shannons*, i.e. display a probability p as $-\log_2 p$. An increase of 1 shannon corresponds to a 50% decrease in probability.

Rerun with different random seeds. Do you think this method has succeeded in identifying topics?

Optional. There are some words that are very common across all topics. How might we pick out the *distinctive* words for each topic?

Question (f): evaluation

Optional. Give a formula for per-word perplexity for the mixture model, in terms of the posterior predictive probabilities for topics and words.

Optional. Plot a histogram showing the distribution of per-word perplexity over all the test documents for the model in part (e). Also plot the histogram obtained from $K = 8$, and the histogram from the plain multinomial model in part (c). Which model do you prefer, and why?