Midterm

David Angeles Albores, bi183

February 8, 2018

Problem 1. Let $X_1, \dots, X_n$ be independent identically distributed Poisson random variables, i.e. each X_i is Poisson with rate λ. Show that $X_1 + \cdots + X_n$ is a Poisson random variable and find its rate.

Let $Z = \sum_i X_i$. We are interested in the statistical distribution of Z_i. A statistical distribution can often be characterized by its expectation and variance. The Poisson distribution is characterized by a mean that is equal to the variance.

The expectation of Z is:

$$\begin{aligned}
\mathbb{E}[Z] &= \mathbb{E}[\sum_i X_i], \\
&= \sum_i \mathbb{E}[X_i], \\
&= \sum_i \lambda_i = \gamma
\end{aligned} \tag{1}$$

which follows from the linearity of the expectation operator. Similar logic applies to the variance operator. This strongly suggests that the statistical distribution of Z_i will be Poisson. To prove this, we should show that the moment generating function of Z_i is Poisson.

The moment generating function is:

$$M_x(t) = \mathbb{E}[e^{tx}] \tag{2}$$

We would like to know the moment generating function of Z:

$$\begin{aligned}
M_Z(t) &= \mathbb{E}[e^{tZ}] \\
&= \mathbb{E}[e^{t(\sum_i X_i)}] \\
&= \mathbb{E}[\prod_i e^{t \cdot X_i}] \\
&= \prod_i \mathbb{E}[e^{tX_i}] \\
&= \prod_i M_x(X_i)
\end{aligned} \tag{3}$$

We find that the moment generating function of Z is the product of the moment generating functions of the X_i. For each X_i, the moment generating function is:

$$\begin{aligned}
M_X(t) &= \sum_{x=0}^{\infty} e^{tx} P(x), \\
&= \sum e^{tx} \frac{\lambda_i^x}{x!} e^{-\lambda}, \\
&= \sum \frac{(e^t \lambda)^x}{x!} e^{-\lambda}, \\
&= e^{-\lambda} \sum \frac{(\lambda e^t)^x}{x!}, \\
&= e^{-\lambda} e^{\lambda e^t}, \\
&= e^{\lambda(e^t - 1)},
\end{aligned} \tag{4}$$

where I have used the fact that $\sum_x A^x/x! = e^A$. Now, all we need to do is plug into equation 3:

$$\begin{aligned}
M_Z(t) &= \prod_i e^{\lambda_i(e^t-1)}, \\
&= e^{(e^t-1)\sum_i \lambda_i}, \\
&= e^{\gamma(e^t-1)}.
\end{aligned} \tag{5}$$

Lo and behold, the moment generating function of Z corresponds to the mgf of a Poisson random variable, with rate $\gamma = \sum_i \lambda_i$. Thus, Z is distributed as a Poisson variable with rate equal to the sum of the rates of its constituents.

Problem 2. Find the string whose Burrows-Wheeler transform is AABBAB$ABBAABBAA (assume that $ is precedes all other characters lexicographically).

To solve this problem, it is sufficient to implement the inverse Burrows-Wheeler transform with a slight modification. In the standard inverse transform, the columns are generated iteratively by placing the transformed string columnwise in front of the previously existing columns, then sorting rows alphabetically and repeating this procedure a number of times equal to the length of the string. The algorithm ends by returning the rows that contains $ at the **end**, since this symbol is preceded lexicographically by all the letters.

In the problem, we are asked to generate an algorithm such that $ lexicographically precedes all other characters. To solve this, we reimplement the same algorithm and searching for the row that **begins** with $. See last page for code.

In the code, I let the substitution 1 stand for $ for simplicity. I then tested my implementation by checking that the Burrow-Wheeler transform of the word '1BANANA' could successfully be inverted. Then, I input the given Burrows-Wheeler transformed string, and found the original.

Problem 3. Consider a hidden Markov model with binary hidden states $\{0, 1\}$, observed states $\{A, B\}$, that has initial probabilities of $\frac{1}{2}$ for starting in each state, and that has length $n = 6$. Find an observed sequence and a set of parameters for the HMM for which the Viterbi sequence (the sequence determined by the Viterbi algorithm) is unique and is 010100, or prove that no such parameters exist.

Let $T(x, I)$ be the probability that the model in state X, where $x \in \{0, 1\}$ emits an observation $I \in \{A, B\}$. Similarly, let $S(x, y)$ be the transition probability from state $x \in \{0, 1\}$ to state $y \in \{0, 1\}$. Let the most probable hidden sequence output by the Viterbi algorithm be represented as $\sigma = 010100$.

We would like to find a sequence of observations, Σ, such that σ is the only answer output by the Viterbi algorithm, or otherwise show that there is no Σ such that σ is the only output answer.

Let:

$$\begin{aligned} T(0, A) &= 1, \\ T(1, A) &= 0, \\ T(0, B) &= 0, \\ T(1, B) &= 1. \end{aligned} \tag{6}$$

Then when the machine is in state 0, it can only output A, and when it is in state 1, it can only output B. That is, the machine outputs deterministic answers in each state.

To guarantee that any sequence can be reached, it suffices that all entries of the matrix, $\hat{S} = \{S(x, y)\} \neq 0$. In other words, if all the entries in $\hat{S}$ are non-zero and the columns add up to unity, then all the states are reachable from any other state. This guarantees that we can always find an optimal and unique σ given a sequence of observations, Σ.

Under these conditions, if the observation is A, then it follows the hidden state must be 0. The sequence, $\sigma = 010100$ then corresponds to the observation sequence, $\Sigma = ABABAA$.

Problem 4. The branch point of an intron is a position located about 18-–40 nucleotides upstream of the 3' end of introns. It is known to contain an "A" (adenine) nucleotide (see diagram on next page). Describe a) a computational strategy for learning what sequences typically occur around the branchpoint in human introns, i.e. whether there are any consensus sequences and what they are, and b) an experimental strategy for identifying branchpoints.

a) The answer will depend on whether we assume we already know the intronic sequences or not. If we do not know what sequences are intronic, then we are being asked to fit an exon/intron/branchpoint/non-coding HMM. The strategy here would literally be the same as for the exon/intron/non-coding HMM described in class, but would add a state called branchpoint that may hopefully identify the probability of emission of a given base given that it is located at a branchpoint.

If, on the other hand, we have good knowledge of the transcriptome and know intronic/exonic sequences, then we could perform other explorations. For one, we could align the N bases up and downstream from an intron/exon junction for all such junctions. We could then ask whether specific bases happen much more often than expected, deriving a consensus sequence around branchpoints.

b) Assuming we don't know what the intron/exon sequences are, then to learn branchpoint sequences, we should extract processed mRNA (without introns). Mapping these mRNA back to the genome will allow us to identify exons. Then we could follow the same strategy as outlined above.

Another approach would be to study what sequences are required and/or sufficient to induce branchpoints. Necessity could be established by taking a short intron, then systematically mutating every basepair to an 'A', 'C', 'G' or 'T'. To establish sufficiency, we should take an exon, and introduce the sequences thought to cause branchpoints into the exon, sequence it and observe whether sequence is missing or not.

Midterm

February 8, 2018

1 Table of Contents

```
In [1]: import pandas as pd
        import numpy as np
        from Bio import SeqIO

        # Graphics
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import seaborn as sns
        from matplotlib import rc

        rc('text', usetex=True)
        rc('text.latex', preamble=r'\usepackage{cmbright}')
        rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

        # Magic function to make matplotlib inline;
        %matplotlib inline

        # This enables SVG graphics inline.
        # There is a bug, so uncomment if it works.
        %config InlineBackend.figure_formats = {'png', 'retina'}

        # JB's favorite Seaborn settings for notebooks
        rc = {'lines.linewidth': 2,
              'axes.labelsize': 18,
              'axes.titlesize': 18,
              'axes.facecolor': 'DFDFE5'}
        sns.set_context('notebook', rc=rc)
        sns.set_style("dark")

        mpl.rcParams['xtick.labelsize'] = 16
        mpl.rcParams['ytick.labelsize'] = 16
        mpl.rcParams['legend.fontsize'] = 14
```

```
In [2]: def BT(S):
            """Given a string, finds its Burrows-Wheeler transform"""
            def circular_permute(S):
                """Returns an array with all the circular permutations of S"""
                perms = [None]*len(S)
                for i in range(len(S)):
                    pre = S[:i]
                    end = S[i:]
                    row = end + pre
                    perms[i] = row
                return perms

            perms = circular_permute(S)
            perms.sort()

            S_BTed = ''
            for p in perms:
                S_BTed += p[len(S)-1]

            return S_BTed

In [3]: def inverse_BT(S):
            """Given a Burrows-Wheeler transformed string, finds the original string."""
            for i in range(len(S)):
                if i == 0:
                    cols = sorted(S)
                else:
                    for i in range(len(S)):
                        cols[i] = S[i] + cols[i]
                    cols = sorted(cols)
            for word in cols:
                # this is the line that changed:
                if word[0] == '1':
                    return word

In [4]: # Check whether this works
        inverse_BT(BT("1BANANANA"))

Out[4]: '1BANANANA'

In [5]: # invert the provided string:
        inverse_BT('AABBAB1ABBAABBAA')

Out[5]: '1ABABAAABBABBBAA'

In [ ]:
```