problem5

February 13, 2023

Bi/Be/Cs 183 2022-2023: Intro to Computational Biology TAs: Meichen Fang, Tara Chari, Zitong (Jerry) Wang

Submit your notebooks by sharing a clickable link with Viewer access. Link must be accessible from submitted assignment document.

Make sure Runtime → Restart and run all works without error

HW 5 (Midterm) Problem 5

For this problem you will be exploring various models which can be used to describe count data i.e. the gene-count matrices we use in single-cell.

Single-cell gene counts, which describe stochastically sampled, discrete measurements of UMI counts, are often modeled as being generated from a negative binomial (or Gamma-Poisson) distribution. However, there is a common assumption that droplet-based methods for single-cell RNA seq incur an overabundance of zeros (more zero counts) than would be predicted by random sampling. Thus it is also common to see single-cell data modeled with zero-inflated negative binomials (the ZINB distribution, with an extra parameter for the probability of zero counts).

You will explore how these assumptions and models fit to real datasets.

```
In [1]: #To run a code cell, select the cell and hit Command/Ctrl+Enter or click the run/play
        #Click Insert --> Code Cell or the '+ Code' option to insert a new code cell

In [2]: #Click Insert --> Text Cell or the '+ Text' option to insert a cell for text as below

In [3]: # This is  used to time the running of the notebook
        import time
        start_time = time.time()
```

Text here for descriptions, explanations, etc

0.1 Import data and install packages

```
In [4]: !pip --quiet install anndata

In [5]: import numpy as np
        import scipy.io as sio
        import pandas as pd
        import matplotlib.pyplot as plt #Can use other plotting packages like seaborn

        import anndata
```

```
        from scipy import optimize
        from scipy.special import gammaln
        from scipy.special import psi
        from scipy.special import factorial
        from scipy.optimize import fmin_l_bfgs_b as optim
```

```
In [6]: # ! allows you to run commands in the command line, as you would in your normal termin
```

```
In [7]: # Download control sample from indrops platform
        # File format is h5ad

        import requests
        from tqdm import tnrange, tqdm_notebook
        def download_file(doi,ext):
            url = 'https://api.datacite.org/dois/'+doi+'/media'
            r = requests.get(url).json()
            netcdf_url = r['data'][0]['attributes']['url']
            r = requests.get(netcdf_url,stream=True)
            #Set file name
            fname = doi.split('/')[-1]+ext
            #Download file with progress bar
            if r.status_code == 403:
                print("File Unavailable")
            if 'content-length' not in r.headers:
                print("Did not get file")
            else:
                with open(fname, 'wb') as f:
                    total_length = int(r.headers.get('content-length'))
                    pbar = tnrange(int(total_length/1024), unit="B")
                    for chunk in r.iter_content(chunk_size=1024):
                        if chunk:
                            pbar.update()
                            f.write(chunk)
                return fname

        download_file('10.22002/xsret-sb590','.gz')
```

```
/tmp/ipykernel_11951/3748164775.py:21: TqdmDeprecationWarning: Please use `tqdm.notebook.trange
  pbar = tnrange(int(total_length/1024), unit="B")


  0%|          | 0/19383 [00:00<?, ?B/s]
```

```
Out[7]: 'xsret-sb590.gz'
```

```
In [8]: !gunzip *.gz
        !mv xsret-sb590 Klein.h5ad
```

```
In [9]: indrops = anndata.read('Klein.h5ad')
```

```
In [10]: indrops

Out[10]: AnnData object with n_obs × n_vars = 953 × 25435
             obs: 'total_counts'
             var: 'empirical_mean', 'empirical_variance', 'empirical_zero_fraction', 'ml_mean'
             uns: 'global_dispersion', 'name'
```

Use the function below for b).

```
In [82]: # X = numpy array of the data (e.g. 1D array with all the counts for one gene)
         # initial params is a numpy array representing the initial values of
         # size and prob parameters
         # Returns: Dict with 'r' and 'p' fits
         def fit_nbinom(X, initial_params=None):
             ''' This code is adapted from https://github.com/gokceneraslan/fit_nbinom
             '''
             infinitesimal = np.finfo(float).eps

             def log_likelihood(params, *args):
                 r, p = params
                 X = args[0]
                 N = X.size

                 # MLE estimate based on the formula on Wikipedia:
                 # http://en.wikipedia.org/wiki/Negative_binomial_distribution#Maximum_likelih
                 result = np.sum(gammaln(X + r)) \
                     - np.sum(np.log(factorial(X))) \
                     - N * (gammaln(r)) \
                     + N * r * np.log(p) \
                     + np.sum(X * np.log(1 - (p if p < 1 else 1 - infinitesimal)))

                 return -result

             if initial_params is None:
                 # reasonable initial values (from fitdistr function in R)
                 m = np.mean(X)
                 v = np.var(X)
                 size = (m ** 2) / (v-m) if v > m else 10

                 # convert mu/size parameterization to prob/size
                 p0 = size / ((size + m) if size + m != 0 else 1)
                 r0 = size
                 initial_params = np.array([r0, p0])

             bounds = [(infinitesimal, None), (infinitesimal, 1)]
             optimres = optim(log_likelihood,
                              x0=initial_params,
                              args=(X,),
```