February 2019

IDC Herzliya
Efi Arazi School of Computer Science
Scientific Computing with Python | Yoav Ram

Final Project, Fall 2018

Please read these instructions carefully. Points may be deducted if you do not follow the instructions. Read this entire document before working on the project.

1. **Complete two out of the following four exercises**. I will not check more than two.
2. **Work alone**. You may talk, but you may not share code.
3. **Make clear and engaging figures**, as they are the main output of your analyses. Axes should be properly labeled, colors and shapes should be meaningful, lines and markers should not be cluttered, axes should be scaled appropriately. The notebook should be saved with the plots inline (I will not run your code for you). Make sure that you re-generate the plots if you changed the code!
4. **Write high-performance code**. Use idioms learned in class, e.g. NumPy, “losing your loops”, numba, multiprocessing, etc. Correctness comes first, but efficiency is important.
5. **Import** whatever you need. You only need packages that we used in class.
6. **Questions** should be posted to the course forum at the designated group (i.e. *project*). You can post questions anonymously, but please do not email the course staff with questions about the project.
7. **Office Hours** are posted to the forum.
8. **Submit via Moodle** before the deadline (see course website), one notebook file per exercise (a total of two .ipynb files).
9. **Submitted files** should be named *ex1.ipynb, ex2.ipynb, ex3.ipynb,* or *ex4.ipynb*. Points will be deducted for any other filename.

Good luck!

Ex 1: Model adaptive peak shifts

We model a population of bacteria and focus on two specific genes responsible for metabolism of lactose (a sugar present in milk). The first gene has two variants, or alleles: bacteria with the A allele will transport lactose into their cell, whereas bacteria with the a allele will not, and therefore will not metabolize lactose. The second gene also has two alleles: bacteria with the B allele will be able to digest lactose and convert it to energy, whereas bacteria with the b allele will not (note that digestion only occurs inside the cell).

Therefore, the fittest bacteria are of type AB, which can both transport lactose into the cell and digest it. However, types Ab and aB (transport without digestion, digestion without transport, respectively), are less fit then type ab. This is because ab cells will neither transport nor digest, but will also avoid the costs of producing the proteins required for transportation and digestion. We therefore denote the fitness of ab as 1, the fitness of Ab and aB as $1-s$ (for $s>0$) and the fitness of AB as $1+sH$ (for $H>0$).

Mutations can change the alleles: with probability u the allele at either gene changes, so that A becomes a and vice versa or B becomes b and vice versa. u is called the *mutation rate*.

We focus on a population with N cells of type ab, and ask: *how can the double-mutant AB appear and fix in the population, if both Ab and aB are less fit than ab?* Fixation of AB in a population of ab is often called *adaptive peak shift* (Sewall Wright, 1931): the types ab and AB are two fitness "peaks", and the types Ab and aB are two fitness "valleys"; in order to move from the local fitness peak ab to the global fitness peak AB, the population has to "go through" one of the valleys Ab or aB (you can see some demonstrations on Bjørn Østman's website). More formally, a single-mutant individual has to appear by mutation, and a second mutation must occur before this single-mutant is purged from the population due to its fitness disadvantage ($1-s$). Then, the double-mutant must survive random events (i.e. genetic drift) and fix in the population, due to its fitness advantage ($1+sH$). Survival of random events is not certain, as we saw in Lecture 3.

Therefore the process is composed of two stages:

1. Waiting for the double-mutant to appear,
2. Waiting for the double-mutant to go to extinction due to random effects (back to stage 1) or fixation due to its advantage (finish adaptation process).

Here, we are interested in characterizing the probability of fixation of a newly appeared double-mutant and the distribution of both waiting times.

Type	*ab*	*Ab*	*aB*	*AB*
Frequency	x_0	x_1	x_2	x_3
Fitness	w_0	w_1	w_2	w_3

Wright-Fisher model. We follow the Wright-Fisher model (stochastic discrete-time model, see Lecture 3). Given the frequencies of the four types in the previous generation x_0, x_1, x_2, x_3, the frequencies after mutation are

$$x_0^m = (1-u)^2 x_0 + (1-u)\,u\,x_1 + (1-u)\,u\,x_2 + u^2 x_3$$

$$x_1^m = (1-u)^2 x_1 + (1-u)u\,x_0 + (1-u)u\,x_3 + u^2 x_2$$

$$x_2^m = (1-u)^2 x_2 + (1-u)u\,x_0 + (1-u)u\,x_3 + u^2 x_1$$

$$x_3^m = (1-u)^2 x_3 + (1-u)u\,x_1 + (1-u)u\,x_2 + u^2 x_0$$

because mutations can occur either in one gene (with probability $u(1-u)$) or in the other gene (with probability $u(1-u)$) or in both genes (with probability u^2) or in neither gene (with probability $(1-u)^2$).

The frequencies after selection (i.e. differential growth and reproduction) are (see Lecture 3)

$$x_i^s = \frac{w_i x_i^m}{\sum_{j=0}^{3} w_j x_j^m}$$

where w_0=1, w_1=w_2=1−s, and w=1+sH.

The frequencies in the next generation x'_0, x'_1, x'_2, x'_3 after genetic drift (i.e. random sampling, see Lecture 3) are

$$(n_0, n_1, n_2, n_3) \sim Multinomial\big(N, (x_0^s, x_1^s, x_2^s, x_3^s)\big)$$

$$x_i' = \frac{n_i}{N}$$

where *N* is the population size (number of cells), kept at a constant value. We assume that the population is initially all *ab*, that is $x_0=1$, $x_1=x_2=x_3=0$.
Tip: This model can also be written in matrix form, as matrix implementations would run faster
Goal. Our goal is to simulate this stochastic model using Python and use these simulations to characterize:

1. The waiting time for the appearance of a double-mutant *AB*,
2. The fixation probability of the double-mutant after it appears,
3. The waiting time for fixation of the double-mutant, if it fixes, and
4. The waiting time for extinction of the double-mutant, if it doesn't fix.

You can implement the simulation using a function that iteratively computes x_i' from x_i, first until $x_3>0$ (appearance), then until either $x_3=0$ (extinction) or $x_3\sim 1$ (getting exactly $x_3=1$ is not reasonable as *AB* can mutate to all other types).

Model parameters. Use these parameters: $N=10^7$, $u=10^{-6}$, $s=0.05$, $H=2.0$. But write the code in a generic way to allow for other parameter values.

Results. You need to produce two figures. The **first figure** should show example dynamics (type frequencies x_i over time) for two scenarios: fixation and extinction of *AB*. **The second figure** should illustrate:

1. The fixation probability of a newly appeared double-mutant,
2. The distribution of the waiting time for appearance of the double-mutant,
3. The distribution of the waiting time for fixation of a double-mutant (given it appeared and fixed),
4. The distribution of the waiting time for extinction of a double-mutant (given it appeared and did not fix).

Run as many simulations as you think are needed to demonstrate the above.
Tip: Consider using multithreading or multiprocessing. Start with a few simulations, make sure everything works fine, then go big.

References.

1. Lecture 3: Population Genetics (incl. YouTube of shorter talk I gave at PyConIL)
2. Book: A Biologist's Guide to Mathematical Modeling in Ecology and Evolution by Otto & Day 2007, Ch. 13.4. Available via IDC Library

Ex 2: Fit Lotka-Volterra competition models to experimental data

In this section we will analyze competitions between two strains of bacteria. We'll fit two related models to some experimental data and select the best model out of the two.

Lotka-Volterra Competition Models. To model competitions between two strains (or species) we will use the classical competitive Lotka–Volterra equations (not to be confused with the predator-prey Lotka-Volterra equations that we saw in class).

This model describes the change in the population sizes of the two strains, x_1 and x_2:

$$\frac{dx_1}{dt} = r_1 x_1 \left(1 - \left(\frac{(x_1 + \alpha_2 x_2)}{K_1}\right)\right)$$
$$\frac{dx_2}{dt} = r_2 x_2 \left(1 - \left(\frac{(\alpha_1 x_1 + x_2)}{K_2}\right)\right)$$

where r_i and K_i are the per-capita growth rate and maximum population size of strain i, and α_1 and α_2 are the competition coefficients.

Note the resemblance of this model to the single-population logistic growth model from Lecture 8; here, the growth-limiting term accounts for both strains, rather than just one.

The competition coefficients account for the relative effect individuals of strain i have on growth of individuals of strain j compared to other individuals of strain i. Specifically, different α_i values can model competition, parasitism, or even charity (see these concepts explained by Ernie and Bert).

Data. The data file is at ***data/flow_df_2015-11-18.csv***. Here are the first 5 rows in the data file:

	strain	date	hour	frequency
0	Green	11/18/2015	10:30	0.432883
1	Red	11/18/2015	10:30	0.567117
2	Green	11/18/2015	11:25	0.386035
3	Red	11/18/2015	11:25	0.613965
4	Green	11/18/2015	12:10	0.337225

The data consists of results of a competition experiment between two bacteria strains, one marked as Green and the other as Red. Approximately every hour, a sample was taken from the tube in which the bacteria were competing. The sample was then processed using *flow cytometry* to count the number of cells with either a green or a red fluorescent protein (GFP or RFP). Then, the frequencies of the green and red cells were calculated by dividing the number of colored cells by the total number of cells (i.e. $f_1= x_1/ x_1+ x_2$ where x_i is the number of cells of type *i* and f_i is the frequency of cells of type *i*).

Goal. In the following, we will

1. Fit the above model (LV6) to the data,

2. Fit a nested, simpler, model (LV4) to the data, in which α_1=α_2=1; that is, strain i has the same effect on strain j as it does on itself.

3. Select the best model out of the two.

The names **LV6** and **LV4** are due to the model name (Lotka-Volterra) and the number of free parameters.

Results. First, **fit the models** (LV6 and LV4) to the data. Then, **print** the minimum loss and the estimated parameters and **plot** the data together with the fitted model. Arrange the two plots (for LV6 and LV4) side-by-side in the same figure.

Note:

- The model follows the number or density of individuals of each type (x_1 and x_2), whereas the data provides the frequency of each type (f_1 and f_2).
- The time units you use for fitting will determine the units of some of the estimated model parameters. Please use **hours** for measuring time. That is, the time variable should be the number of hours since the experiment started.
- What should be a good-enough loss function, and how you can minimize it? See *scipy.optimize* for some methods.
- What should be the initial population sizes in the model?

Second, because it may be difficult to decide which model is better based on the minimum loss and the plots, perform **model selection** to select one of the models and answer the question: *can the nested model LV4 be rejected in favor of the full model LV6?* **Print and plot** any required steps to support your decision, and finish with a clear statement answering the question above.

References.

1. Lecture 8: Population Dynamics 1
2. Lecture 9: Population Dynamics 2
3. Book: A Biologist's Guide to Mathematical Modeling in Ecology and Evolution by Otto & Day 2007, Ch. 3.4.1. Available via IDC Library.

Ex 3: Object detection with linear models

In this exercise we develop a linear model for object detection in images. Specifically, we will train the model to detect the area of an image in which a hand is shown to gesture a sign language sign.

We will develop a linear model that, given an image $\mathbf{x}$ (an array of pixels), produces a bounding box $\mathbf{y}$. The bounding box is a rectangle defined by its top-left and bottom-right corners. Hence $\mathbf{y}=(y_1, y_2, y_3, y_4)$. The detected object -- a hand gesturing a sign -- is to be inside this bounding box.

Data. This is a supervised learning problem, as we have a set of images with their corresponding bounding boxes. We will use a sign language dataset that includes both images and bounding box examples. The data is available as a zip file at https://github.com/yoavram/Sign-Language/raw/master/Dataset.zip. After extracting the zip file to the folder *data/sign-lang*, we have 7 folders, one for each person (user). Each folder has 10 images of that person gesturing one of 24 signs: the ABC letters, not including J and Z. This is the file structure (the hyperlinks don't lead anywhere):

(see in the next page)

```
├── user_3
│   ├── A0.jpg
│   ├── A1.jpg
│   ├── A2.jpg
│   ├── ...
│   ├── B0.jpg
│   ├── B1.jpg
│   ├── B2.jpg
│   ├── ...
│   ├── Y8.jpg
│   ├── Y9.jpg
│   └── user_3_loc.csv
├── user_4
│   ├── A0.jpg
│   ├── A1.jpg
│   ├── A2.jpg
│   ├── ...
│   ├── Y8.jpg
│   ├── Y9.jpg
│   └── user_3_loc.csv
...
└── user_10
    ├── A0.jpg
    ├── A1.jpg
    ├── A2.jpg
    ├── ...
    ├── Y8.jpg
    ├── Y9.jpg
    └── user_10_loc.csv
```

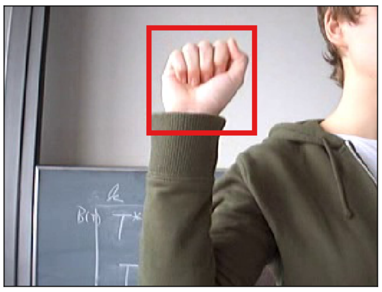
The *jpg* files are images, for example these are *user_3/A0.jpg* and *user_9/K7.jpg*:

The image name has the sign (A and K in these examples) and the repetition number.

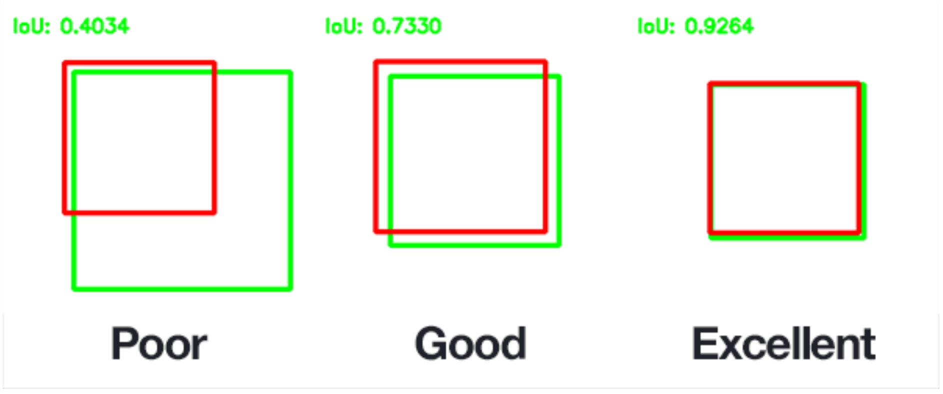
In addition, each folder contains a metadata file; for example, the first 5 rows in *user_3/user_3_loc.csv* are

	image	top_left_x	top_left_y	bottom_right_x	bottom_right_y
0	user_3/A0.jpg	124	18	214	108
1	user_3/A1.jpg	124	18	214	108
2	user_3/A2.jpg	123	19	213	109
3	user_3/A3.jpg	122	21	212	111
4	user_3/A4.jpg	122	20	212	110

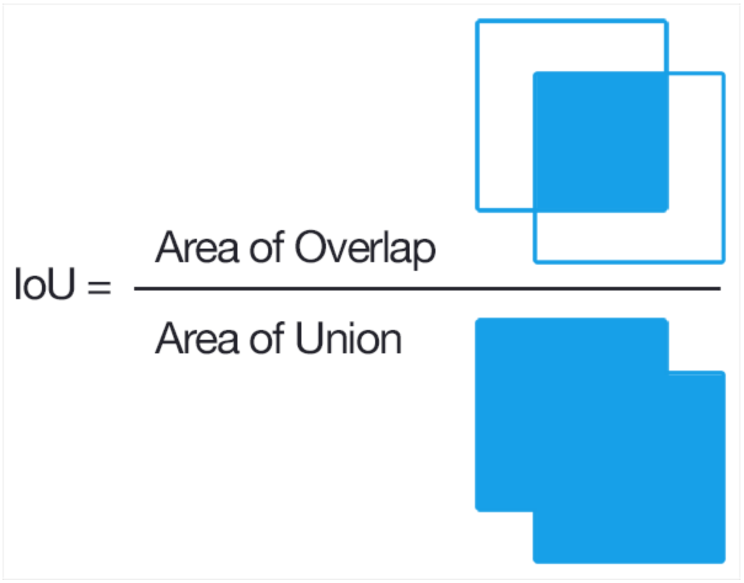
This table has 5 columns. The first column **image** provides the name of the image. The other columns provide the **bounding box** *y* for that image. For example, the bounding box for *user_3/A0.jpg* is a box with corners at (124, 18) (214, 108):



Goal. We want to collect all images into an array *X* and all bounding boxes into an array *Y*. Then, we want to train a model that, given an example image *x* produces the bounding box *y* for that image, such that the bounding box will contain a gesturing hand.
We would like to have a simple, rather than complex model: if we can produce good results with GLM, than we prefer them over neural networks.
We also need to evaluate and visualize our model. Visualization is easy enough – we plot the image, together with the true bounding box and the predicted bounding box. To evaluate the model, however, we need to define a metric called IoU (intersection over union): the ratio of the intersection and the union of the true and predicted bounding boxes. The intuition is given by this illustration:

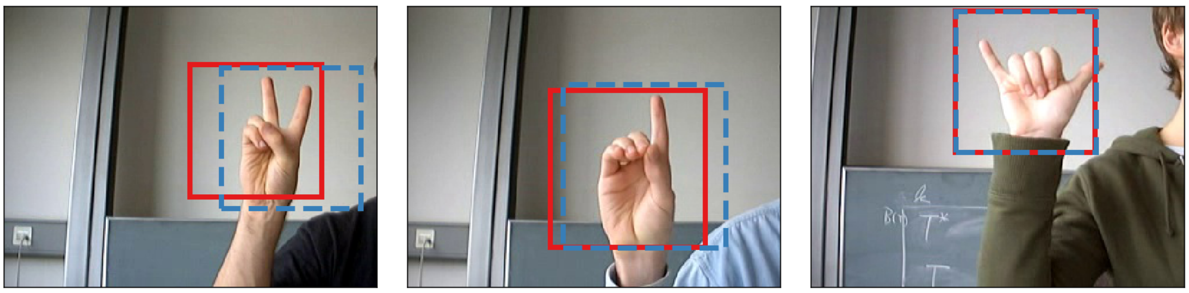


The definition is illustrated here:



You can find more details on IoU, including an implementation, here. However, you should implement IoU using NumPy such that it works on arrays of images, rather than just using the implementation in the link. Hint: use *np.minimum* and *np.maximum*.

Results. First, **train a generalized linear model** on a training set.
Second, **Visualize the model performance** by plotting the image, the original bounding box, and the predicted bounding box, for example:



Plot the predictions for 9 random images from the test set and print the real and predicted bounding boxes, together with the IoU.
Third, **summarize model performance** by computing the IoU over all test images, plotting the IoU histogram, together with the mean and 95% confidence intervals. Also **print** the mean and 95% confidence intervals of the IoU scores.

References.

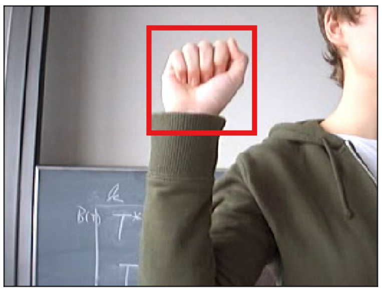
1. Lecture 6: GLM 1
2. Lecture 7: GLM 2
3. Lecture 12: Softmax model

Ex 4: Classification of sign language hand gestures with CNN

This exercise is a continuation of Ex 3, but can be solved separately (you can submit Ex 4 without Ex 3, or submit Ex 3 without Ex 4).
In Ex 3 we developed a linear model to detect the region of an image in which there is a hand that gestures a sign.
In this exercise we will develop a convolutional network model to classify the hand gestures. We will develop a model that, given an image $\mathbf{x}$ (an array of pixels), produces a classification $(y_A, y_B, \ldots, y_Y)$ such that y_i is the probability that the image is of the hand sign i and $\sum y_i = 1$.
This is a supervised learning problem, as we have a set of images with their corresponding hand signs.

Data. The data is the same as in Ex 3 (see above). However, we are going to work on cropped versions of the original images: we will use the bounding boxes, given in the metadata *csv* files, to crop the images. The idea is that the model from Ex 3 and the model we develop here will consist of a pipeline: the first model looks at an image and finds a bounding box, the second model only looks inside the bounding box and classifies the hand gesture.

For example, here's the image *user_3/A0.jpg* with a bounding box, and then its cropped version:



Original **Cropped**

The classes or labels are given in the filename (which also appears as a column in the *csv* files): for example, *A0.jpg* has the class *A*, whereas *H8.jpg* has the class *H*.

Goal. We want to collect all cropped images into an array *X* and all classes into an array *Y* (don't forget one-hot encoding the classes). Then, we want to train a **convolutional neural network** that, given an example cropped image *x* produces the predicted class probabilities *y* for that image.
Notes:

- Each cropped image has a different shape. **Resize or rescale** them (using *scipy.ndimage* or *skimage.transform* or whatever) to a common shape: this is a requirement for convolutional neural networks, which in principle don't work on variable array sizes.
- Cropping and resizing must be done in a ***preprocess*** function that will be shipped together with the model so that users can use it on their own images; the *preprocess(image, box)* function should take the bounding box as an argument.
- You should also prepare a ***decode*** function that translates the model output $\hat{y}$ to a string of the most likely hand sign.
- The convolutional neural network can be similar to those in Lecture 13 and Assignment 5. If they do not perform well, increase their complexity (depth and/or width). You can also use *BatchNormalization* as a replacement for *Dropout*, or use *transfer learning*, or any other technique you want (whatever gets the job done).
- You can work on *Google Colaboratory*, which provides free access to GPU (see instructions in A5).

Results. First, **train a Keras CNN** on a training set. Then **print** its accuracy on the test set. You should be able to achieve an accuracy of at least 85%.
Second, **visualize** the model performance by randomly selecting 9 images, plotting them, and printing their real and predicted classes (as text strings like 'H' and 'B').
Third, identify which classes are more commonly mis-classified: **compute and plot** the accuracy of each separate class (hand sign), and illustrate which classes have relatively lower accuracy. These should be the classes we attempt to deal with next (if we were to continue this exercise), maybe by checking what they are confused with.

References.

1. Lecture 12: FFN
2. Lecture 13: CNN
3. Assignment 5: Solution

End of document