Assignment 2
BIOENG 145/245
Due Monday 2/06 at midnight

*****PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!*****

1 Python Functions

In this section, you will implement several functions to brush up on your basic Python skills. **More detailed specifications of the inputs and outputs and example test cases are included in the code**. Please look at `q1.py` for this problem. Submit `q1.py` to Gradescope when completed, there are no written responses for this problem.

1.1 Reverse Complement

Find the reverse complement of the input sequence. Assume that the inputs are restricted to A, T, G, and C (all capitalized).

1.2 GC Content

Find the GC content of the input sequence. The GC content is determined by dividing the number of G's and C's by the total number of nucleotides in the sequence. Assume that the inputs are restricted to A, T, G, and C (all capitalized).

1.3 Find Motif Frequency

Given a target motif, find its frequency (count how many times this motif appears) in the given DNA strand.

1.4 Find Binding Site

Given a sequence `seq`, find the first position of the binding site of `bind`, a shorter sequence. Note that the binding site is the reverse complement of `bind`. Use 0-indexing. Return -1 if the binding site is NOT found in the seq.

Hint: you can call the rev_complement() method earlier.

2 Simulation Potpourri

In this section, we will have you develop simulation algorithms with a focus on probability. The primary goal is to understand how we can reason complicated situations and difficult probability problems with code. Please implement and fill out the methods in `q2.py`. For all of the methods, simulate the given situation `trials` times, and use the results to calculate the quantity of interest. The secondary goal is to help you build a solid foundation in `NumPy`, a very important Python package used in Machine Learning. Submit `q2.py` to Gradescope when you are done. There is no written response for this section. (**More detailed specifications of the inputs and outputs and example test cases are included in the code**)

Here are a few methods that may be very(!) helpful for implementing this section.

- `np.random.rand`
- `np.random.randint`
- `np.sum`
- `np.unique`
- `np.random.permutation`

Note: you do NOT need to use all these methods!

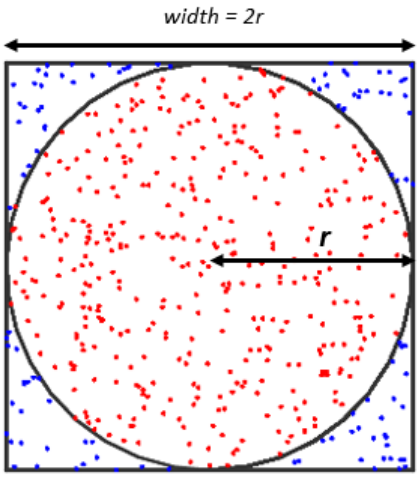
2.1 `num_die_sum`

Given n fair 6-sided die, what is the probability that we roll a certain sum?

2.2 `correct_papers`

A professor got mad at his students and throws a pile of n papers on the floor and asks each student to pick up a random paper from the floor. On average, how many students get their own paper back?

2.3 `monte_carlo_pi`



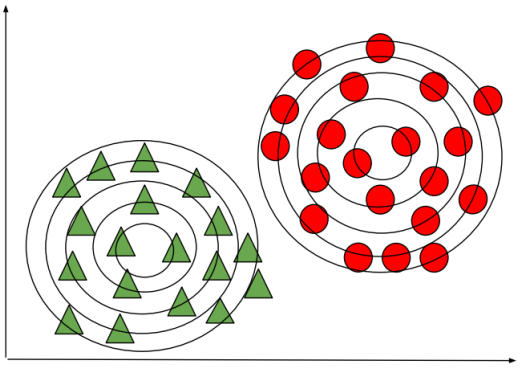
Estimate pi using num_points. Create a unit circle inside a unit square, then generate a a big amount of points inside the unit square. Divide the number of points that falls in the unit circle by the total number of points generated. Looking at the image above, what does this ratio tell us? Now, having understand what this ratio means, use this number to estimate π.

Note: the equation of a unit circle on the Cartesian space is $x^2 + y^2 = 1$

2.4 `roll_until_repeat`

On average, how many times do we have to roll a 6-sided dice until we get a two consecutive rolls with the same value?

3 Gaussian Discriminant Analysis



In this section, (most of) you will write your very first statistical learning model from scratch. In this problem, we will implement the Gaussian Discriminator (`GaussianDiscriminator.class`), a classic yet simple statistical model. A Gaussian Discriminator simply fits a multivariate Gaussian distribution for each class in a dataset, and then, given a new datapoint, predicts which class the datapoint belongs to based on which multivariate Gaussian distribution was most likely to generate it.

Let's say we have a classification task where `num_classes=2`, aka a binary classification task. An example of such a classification task is trying to classify tumors into benign or malignant. Additionally, we are provided a dataset that consists of data (with multiple features such as mass, dimensions, size, etc.) of both benign and malignant tumors.

For each class, the model will fit a Gaussian Distribution by calculating the mean and covariance matrix of each class. In our tumor example, the model will calculate the average and covariance matrices of different features for each class. Then, when a new unclassified datapoint comes in, the model will compare whether the datapoint is more likely to belong to the benign or the malignant Gaussian distributions. The image above shows a binary classification task with GDA, where each "group" is a multivariate Gaussian Distribution.

Code explanation:

- `__init__(self)`: the initiation method, will create a list of Gaussian distributions. Each item of the list will contain a tuple of a class index, the class' mean, and the class' covariance matrix. (DO NOT MODIFY THIS METHOD).
- `fit(self, X, y, num_classes)`: the fit function will calculate the mean and the covariance matrix of each class using the given data and its corresponding labels (X and y, respectively) and store the `(class, mean, covariance_matrix)` in the `self.gaussians` list. `num_classes` is the number of different classes we will classify. The NumPy methods `np.mean` and `np.cov` will be needed here.
- `predict(self, x)`: the predict function will take a new data point and classify it according to the statistics it learned using the fit function. The helper method included in q3.py `pdf(x, mean, cov)` will be needed in this function. Iterate through every class' Gaussian distribution using the `self.gaussians` variable and find the corresponding probability density of the data. Find the class with the Gaussian distribution that generates the largest probability density, this is the class we assign to the data point.

Open q3.py and implement the `fit` and the `predict` functions. You will NOT need to modify the initialization method. There is no written portion for this question.

4 Submission

Upload your completed `q1.py`, `q2.py`, and `q3.py` to Gradescope's HW 2 Coding Portion. For the coding portion, ***PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!*** The autograder will run your program and test the code, and whatever score you see after the autograder finishes running is the score you receive for the coding portion. Note: For the simulation problems, the large number of trials means that your answers should be within range of the autograder's expectations, but there is always some possibility they aren't. If it grades your answers incorrectly, feel free to run it several more times to get a better score.