Lab 1

Due: Friday 09/12/2025 @ 11:59pm EST

The purpose of labs is to give you some hands on experience programming the things we've talked about in lecture. This lab will focus on probability calculations, and thankfully we will be working with discrete random variables. In fact, we will be predicting whether or not avacados are good to eat given their color (feature 1) and their softness (feature 2). Both of these features are discrete and are implemented using enums in python (more on that later).

Task0: Setup

Contained with this document is a package that contains python source code and a data file. It is organized as follows:

- `data/` This directory contains all data files (so far just a single one: `train_avacados.txt`. Feel free to open this file and look around, however **do not modify this file**. I would encourage you to think of this file as a table (i.e. a dataset where each example is a single row in the table). Each example contains three pieces of information: the color of an avacado, the softness of that avacado, and whether or not it was good to eat. This dataset could be collected by asking people to write down these three attributes (of course "good to eat" is subjective but that's neither here nor there).
- `pysrc/`. This directory contains all python source code. Inside it you will find one file:
  - `main.py` In this file you will find a few things: some enums classes, a few functions (one to load data), and a class called `AvacadoPredictor`. The class `AvacadoPredictor` and the `main` function are the only pieces of code you should modify (of course you are welcome to import packages as you need). The class `AvacadoPredictor` currently contains three fields: each field stores a pmf.

Task1: Storing Probability Mass Functions (60 points)

Predicting whether an avacado is good to eat or not is difficult. This is because we would like to know whether or not an avacado is good to eat *before* we eat it: meaning we are trying to predict something before we measure it. The good news is that we can use Baye's rule to model this unknown attribute using attributes that are much easier to measure: color and softness. In this lab we will be modeling the good to eat attribute off of color and softness *separately*, meaning that while it is much more interesting to try to model $Pr[GoodToEat|Color \cap Softness]$, that is more than I want us to do in this lab, so instead we will be modeling

$$Pr[GoodToEat|Color] = \frac{Pr[GoodToEat]Pr[Color|GoodToEat]}{Pr[Color]}$$

and

$$Pr[GoodToEat|Softness] = \frac{Pr[GoodToEat]Pr[Softness|GoodToEat]}{Pr[Softness]}$$

Inside the `AvacadoPredictor` class, you will find three fields: `good_to_eat_prior`, `color_given_good_to_eat_pmf`, and `softness_given_good_to_eat_pmf`. Below is a short description regarding each of these pmfs:

- `good_to_eat_prior` The attribute `GoodToEat` isn't know before we eat the avacado, and is the attribute we want to predict. Since our data is annotated by other people's opinions on whether avacados they've consumed were good, we can try to answer the question "if I were to pick an avacado at random, what are the chances that this avacado is good to eat regardless of it's other attributes?" This field stores the pmf $Pr[GoodToEat]$ and is implemented with a defaultdict. The key for this dictionary is the value that discrete random variable `GoodToEat` has taken, and the value is the corresponding probability. The way we calculate each element of this pdf is to count. Lets say we want to store the value $Pr[GoodToEat = YES]$. We would, from our data, count

$$Pr[GoodToEat = YES] = \frac{\text{\# of times}(GoodToEat = YES)}{\sum\limits_{a \in GoodToEat} \text{\# of times}(GoodToEat = a)}$$

- `color_given_good_to_eat_pmf` This field should store the distribution $Pr[Color|GoodToEat]$. Since `GoodToEat` is a discrete random variable, this boils down to having to store two pmfs: $Pr[Color|GoodToEat = YES]$ and $Pr[Color|GoodToEat = NO]$. For this reason, the field `color_given_good_to_eat_pmf` is a defaultdict of defaultdicts. The outer-key (first key) is the value of `GoodToEat`, which leads to a defaultdict that stores the pmf for that value of `GoodToEat`. The inner-key (second key) is a value of the `Color` discrete random variable, and this leads to a probability value. To index this data structure, if we wanted to know $Pr[Color = GREEN|GoodToEat = YES]$ we would look up the following entry: `self.color_given_good_to_eat_pmf[GoodToEat.YES][Color.GREEN]`. We also calculate these pmfs by counting. For example, let us try to store the value $Pr[Color = GREEN|GoodToEat = YES]$. We would, from our data, count

$$Pr[Color = GREEN|GoodToEat = YES] = \frac{\text{\# of times}(Color = GREEN \cap GoodToEat = YES)}{\text{\# of times}(GoodToEat = YES)}$$

- `softness_given_good_to_eat_pmf` This field is the same as the `color_given_good_to_eat_pmf` field, only instead of color data, this is for the `Softness` attribute (and therefore stores $Pr[Softness|GoodToEat]$). We would calculate these pmfs also by counting in a manner identical to the $Color$ attribute.

Your job in this section is to complete the implementation for the `fit` method. This method takes in the table of data, and should populate the three fields: once this method is complete those three fields should contain valid pmfs.

Task2: Bayes Rule and Making Predictions (40 points)

Once we can calculate the pmfs, we need to use them to make predictions. In this task you will finish the implementation of two methods: `predict_color_proba` and `predict_softness_proba`. Let us talk about `predict_color_proba`. `predict_softness_proba` will be the same process but focusing on `Softness` rather than `Color`.

The goal of the `predict_color_proba` method is, given a list of `Color` values (i.e. one value per avacado we are trying to predict), we would like to output the entire distribution over the `GoodToEat` discrete random variable. For every `Color` value, we want to return a list of tuples: one tuple per value that the `GoodToEat` attribute can take on. Inside of this tuple we want to store two things: the value of the `GoodToEat` attribute G (the first value in the tuple), and the probability $Pr[GoodToEat = G|Color = c]$ (the second value in the tuple). Remember to calculate this value with Baye's rule (the equation is above in the previous task). I recommend calculating each pmf at a time (i.e. processing one `Color` value at a time), calculating the *numerator* of Baye's rule for the entire pmf, and then normalizing the pmf by the total of the pmf. Please collect these pmfs in a list (with each pmf at the same index as the corresponding `Color` value that generated it in the input list) and return this list.

Task 3: Extra Credit (10 points)

Please add two methods to your `AvacadoPredictor` class: `predict_color` and `predict_softness`. Again, let us only talk about `predict_color` and the same process will be true for `predict_softness` (of course focusing on `Softness` rather than `Color`). The goal of this is to actually produce an prediction for each input example. `predict_color` will behave similarly to `predict_color_proba`, the only difference is that instead of producing an entire pmf: we want to produce the mostly likely value of the `GoodToEat` attribute. I recommend calling `predict_color_proba` within `predict_color` and processing each pmf to find the mostly probably value of `GoodToEat`. Your output should be a list of `GoodToEat` enum values that correspond to the predictions that we want to make for each of the input `Colors`.

Task 4: Submit Your Lab

To complete Lab1, please submit `main.py` on Gradescope. You shouldn't have to worry about zipping it up or anything, just drag and drop it in.