Exercise 1

Random Variable X : $f(x;\theta) = \theta^2(x+1)(1-\theta)^x$, $x = 0,1,2\ldots$, $\theta \in [0,1]$

($f(x;\theta)$: prob. of observing x given θ; $X \sim \{X_1, X_2, \ldots, X_N\}$)

a) We want to find the maximum likelihood estimator for θ. We have independent identically distributed (i.i.d) samples ($=N$), which means that since the events are independent, the probability (likelihood) of observing events together, is a product of each individual probability of each event.
We can thus compute the joint probability (likelihood) as:

likelihood function: $L(\theta) = \prod_{i=1}^{N}[f(x_i;\theta)] = \prod_{i=1}^{N}[\theta^2(x_i+1)(1-\theta)^{x_i}]$ (1)

log likelihood function: $l(\theta) = \sum_{i=1}^{N}[\log(f(x_i;\theta))] \overset{(1)}{=} \sum_{i=1}^{N}[(2\ln(\theta) + \ln(x_i+1) + x_i\ln(1-\theta))]$ (2)

In order to find the θ that maximizes the log likelihood function, we will take its derivative with respect to θ; then solve it for θ:

1st derivative: $\frac{\partial l(\theta)}{\partial \theta} \overset{(2)}{=} \sum_{i=1}^{N}\left[\frac{2}{\theta} - x_i\frac{1}{1-\theta}\right] = \sum_{i=1}^{N}\left[\frac{2}{\theta}\right] - \sum_{i=1}^{N}\left[\frac{x_i}{1-\theta}\right] = \frac{2N}{\theta} - \frac{\sum_{i=1}^{N}[x_i]}{1-\theta}$ (3)

Set $(3) = 0 \rightarrow \frac{2N}{\theta} - \frac{\sum_{i=1}^{N}[x_i]}{1-\theta} = 0 \overset{\times\theta(1-\theta)}{\rightarrow} (1-\theta)2N - \theta\sum_{i=1}^{N}[x_i] = 0 \rightsquigarrow$

$\rightsquigarrow 2N - 2\theta N - \theta\sum_{i=1}^{N}[x_i] = 0 \rightsquigarrow \theta(2N + \sum_{i=1}^{N}[x_i]) = 2N \rightsquigarrow \boxed{\theta = \frac{2N}{2N+\sum_{i=1}^{N}[x_i]}}$ (4)

(4) Is the MLE estimate for θ.

b) Since we have values for x_i, we can find the θ that maximizes the log likelihood function.

$x_i = [3.2, 1.4, 2.2, 7, 0.5, 3.3, 9, 0.15, 2, 3.21, 6.13, 55, 1.8, 1.2, 11]$
So we have 15 values ($N=15$).

We can now use $\theta = \frac{2N}{2N + \sum_{i=1}^{N} [x_i]}$

The summation of $x_i = 57,59$, so

$\theta = \frac{215}{215 + 57,59} = \frac{30}{87,59} \simeq 0,3425$, which is the MLE for the parameter θ. (given x_i values) This means that if it is indeed a maximum, it makes the observed values most probable under our model $f(x;\theta)$.

To check if the value of θ is indeed a maximum (local), we take the second derivative of $l(\theta)$. Depending on the result:

- If $l''(\theta) < 0$: It is indeed maximum (local)
- If $l''(\theta) > 0$: It is minimum (local)
- If $l''(\theta) = 0$: We are uncertain of whether it is a maximum, minimum or neither.

Exercise 2.

$n = 7$

Boolean variables: X, Y, Z (features) (0 or 1)

Use Table to train a Naive Bayes classifier. (to identify U)

features → X, Y, Z; class → U

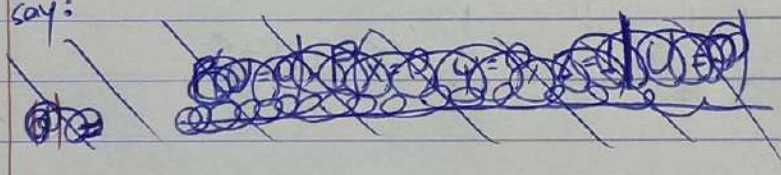
X	Y	Z	U
1	1	1	0
0	1	1	0
0	0	1	0
1	0	0	1
0	0	1	1
0	1	0	1
1	1	0	1

Table / samples

→ a) $P(U=0 \mid x=0, y=0, z=1) = ?$ (1)

We want to find what is the predicted value (class) of U, given $x, y = 0$ and $z = 1$.

Since the inputs are discrete (binary to be exact) we can say:



$$\| = P(U=0) \cdot P(x=0 \mid U=0) \cdot P(y=0 \mid U=0) \cdot P(z=1 \mid U=0) \quad (2)$$

Using the table, we can now calculate each individual probability.

(Prior prob.) • $P(U=0) = \frac{n_{U=0}}{n} = \frac{3}{7}$ and also $P(U=0) + P(U=1) = 1$

So $P(U=1) = \frac{4}{7}$

→ For the rest, since it is given $U=0$, our "total samples" will be $n_{U=0} = 3$

(Conditional prob) • $P(x=0 \mid U=0) = \frac{\text{occurrence of } x=0 \text{ when } U=0}{n_{U=0}} = \frac{2}{3}$

Same for the rest:

" • $P(y=0 \mid U=0) = \frac{1}{3}$

" • $P(z=1 \mid U=0) = \frac{3}{3} = 1$

Now we can calculate (2).

$$(2) = \frac{3}{7} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot 1 = \frac{6}{63} \overset{/3}{=} \frac{2}{21} \simeq 0{,}095$$

In order to find the probability for $P(U=1 \mid x=0, y=0, z=1)$ (3) to compare the two, we use the same approach. We know $P(U=1)$, so we need only the conditional probabilities:

- $P(x=0 \mid U=1) = \frac{2}{4} = \frac{1}{2}$
- $P(y=0 \mid U=1) = \frac{2}{4} = \frac{1}{2}$
- $P(z=1 \mid U=1) = \frac{1}{4}$

$$(3) = \frac{4}{7} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{4}{112} = \frac{1}{28} \simeq 0{,}035$$

Now we can compare: $\frac{(2)}{(3)} = \frac{0{,}095}{0{,}035} > 1$, so $(2) > (3)$

meaning we classify $U=0$ given $x=0, y=0$ and $z=1$ with a probability of*: (Since it could be either 0 or 1, and $P(U=0 \mid x=0, y=0, z=1) + P(U=1 \mid x=0, y=0, z=1) = 1$)

* $$(2) = \frac{\frac{2}{21}}{\frac{2}{21} + \frac{1}{28}} = \frac{\frac{2}{21}}{\frac{8}{84} + \frac{3}{84}} = \frac{\frac{2}{21}}{\frac{11}{84}} = \frac{168}{231} = \frac{8}{11} \simeq \boxed{0{,}727}$$

$$P(U=0 \mid x=0, y=0, z=1) = 0{,}727$$

6) To have a better look at the frequency of each feature, we create:

Frequency table

X	U occurences	
	0	1
0	2	2
1	1	2

Frequency table

Y	U	
	0	1
0	1	2
1	2	2

Frequency table

Z	U	
	0	1
0	0	3
1	3	1

Generally, the Laplace trick helps us when we have a frequency = 0 of a given class. In our case we do encounter this, for $P(Z=0|U=0)$ This can be seen at the tables (denoted with green)

Lets try to calculate for $U=0$ and $Z=0$ the following:

$$P(U=0|X=0,Y=0,Z=0) = P(U=0)\cdot P(X=0|U=0)\cdot P(Y=0|U=0)\cdot P(Z=0|U=0) =$$
$$= \frac{3}{7}\cdot\frac{2}{3}\cdot\frac{1}{3}\cdot 0 \text{ . The problem here arises for } P(Z=0|U=0)$$

Since it is 0, the probability becomes zero, assuming it is of the other class (in our binary case, it would classify as $U=1$).
Since our dataset might be incomplete, this is a problem. Even if we do not encounter this combination, it does not mean it does not exist. That is where Laplace trick would help us, by giving it atleast a very low probability.

→ c) In Naive Bayes we assume conditional independence, thus to compute

$$P(U=1|x=0) \underset{\text{Bayes formula}}{=} \frac{P(x=0|U=1)\cdot P(U=1)}{P(x=0)} = \frac{\frac{2}{4}\cdot\frac{4}{7}}{\frac{4}{7}} = \frac{1}{2} = 50\%$$

From table: $P(x=0|U=1) = \frac{2}{4}$

$P(U=1) = \frac{4}{7}$

$P(x=0) = \frac{4}{7}$

Exercise 3

Naïve Bayes Classifier (NBC) is a supervised machine learning algorithm used for both categorical and continuous data classification. The assumption here is that all features are conditionally independent, given a class label.
During this exercise, we try to implement the algorithm to first train a model and then predict the classes based on the trained model.
As mentioned, there are two different types of data NBC classifies, which are 1) categorical and 2) continuous.

A) Training model:

1)For categorical data**:**

We use 3 different types of information (datasets).
The **first** is a matrix (X) with IxM dimensions (I being samples and M being variables (features).
The **second** is an Ix1 vector (Y), containing class information for each sample (meaning what class each sample belongs to).
The **third** is an 1xM vector (D_categorical) representing the number of different possible values the features can have.
For the implementation, we also use Laplace smoothing, which is a scalar that is used when the frequency of a given class is 0, even for one specific value of a feature. Since when trying to compute the probability of a class given feature values, the conditional probability later on for that variable given a class will be 0. Since we multiply by the probabilities of each variable given a class, if that probability is 0, the initial probability P(Y=y | x1,x2,...xm) for a class y for m variables, becomes 0.
The model takes any value of the hyperparameter L (L>=0), with L=0 meaning no Laplace smoothing.
For the implementation we will use nested dictionaries since they are a fast and easy way to work and iterate through.

First of all we will calculate the prior probability of each class, meaning unconditional to variables. Basically we count the number of occurrences of a class and divide by the number of samples. When Laplace is applied, we simply add the L to the numerator, while we add L multiplied by the number of classes (unique). We do this for each class. Essentially we calculate $P(Y=y) = (Number\_of\_y\_occurrences+L)/(Total\_samples+L)$

After this, we need to calculate likelihoods. We iterate through each feature, then through each of the features possible values, followed by the class.
First we calculate the numerator of the likelihood. For each feature, given a class we simply count the number of encounters of each feature for each specific value for each different class. At the same time, we apply the Laplace trick by adding L to the summation of those.
For the denominator, we use the total number of said class occurrences, while adding the number of possible distinct values that the specific feature can have, multiplied by Laplace hyperparameters.
Essentially, we solve this: $P(Xm = feat\_value \mid Y = y)$, for m features, given class y.

The model returned has 2 basic key nodes, 1 being ***'prior'***, which contains a dictionary with each class as a key, and the prior probability of said class. The second is ***'likelihood'***.
It contains 3 layers of dictionaries, starting from each feature (0 to M-1), then all possible values for that feature as values and keys to all the possible classes, each containing the probability calculated. A pseudo dictionary looks like this: *nbc_model['likelihood'] → {features : {feature values : {class: likelihood}}}*, each having multiple values depending on features, each feature's possible values, and the possible classes.

2)<u>For continuous data:</u>

We follow the same steps to calculate prior probabilities for classes. Laplace smoothing is not applied here. Since we cannot calculate likelihoods due to continuous data having infinite possible values, we assume normal probability distribution (Gaussian) and so we calculate the parameters μ, σ. These will be used to compute the probability of observing a particular value for the feature given a class y through the Gaussian probability density function (PDF). For this, we simply go through each feature, and then for each feature, through each class. Given a class we calculate mean and standard deviation for that feature. It is a class-specific feature mean and class-specific feature std.

The model returned has 2 basic key nodes again, first being the same ***'prior'*** and the second ***'mean_std'***. This time, we have 2 layers, the first is features from 0 to M-1, the second is the possible classes which contains a tuple of (mean, std). A pseudo dictionary look like this:
nbc_model['mean_std'] → {features : {class: (mean, std)}}

B) <u>Training model:</u>

This model works for both categorical and continuous data. As input we use the predicted model, along with the X matrix (IxM). The model will predict what class a given sample will be, based on the training we did before. It returns 2 arrays of length Ix1, ***sample_predictions*** and ***sample_pred_probs***. The former contains the predicted class, the latter contains the posterior probability with which it was predicted.
We go through each sample, creating an empty dictionary with posterior probabilities that will be filled with class specific posteriors. For each sample, we access the ***'prior'*** key, containing class prior probabilities.

1)<u>For categorical data</u>**<u>:</u>**

After accessing the prior probabilities for each class, essentially we want to calculate P(Y = y | Xi), for *y* class and *i* samples. To compute this, we use joint and conditional probabilities as seen in Bayes' theorem. We calculate the product over all features *m*: Π[P(X_*{i,m}* = value | Y=y)] (for *i* samples).
To calculate the posterior probabilities we add the log(prior_probability) of the class and the sum over all features *m*: Σ [log(P(X_*{i,m}* = value | Y=y))] for each feature.
We use the logarithm which helps with converting product to summation and also helps with number underflow.

2)For continuous data:

After we access class prior probabilities, we access the mean and standard deviation contained in the ***'mean_std'*** key, containing class-specific feature mean and standard deviation.
The process here is different, since as mentioned for continuous features, we assume that the data for each feature follows a normal distribution (Gaussian). For this reason, we use the probability density function (PDF) to estimate the likelihood P(X_*{i,m}*=value | Y=y).
*– P(X_{i,m} = x | Y=y) = 1/sqrt(2π*σ^2)*e^(-(x-μ)^2 / (2*σ^2)) for i,m,y : samples, features, class) –*

For each feature, we go through each class and compute the likelihood using the PDF, with the parameters μ,σ of this function determined by the feature's class-specific mean and standard deviation.
To calculate the posterior probabilities, we add the log(prior_probability) of the class and the sum over all features *m*: Σ[log(P(X_*{i,m}* =value | Y=y))]
Again, we use logarithms to transform product operations into summation and to handle potential number underflow.

After the posterior probability for a sample – for each of the classes – is computed (regardless of data type), we store it, and when we have gone through all of the classes for a sample, we classify it. What we do is to extract the maximum probability, along with the class. We are finding yi = argmax P(Y=y | Xi) for *i* samples and *y* class. The probability is stored in ***sample_pred_probs*** and the class in ***sample_predictions.***

C) Using the dataset:

The last step is to actually use our classifier. For this purpose we load our dataset, and we split it into a 75% training set and 25% test set. The first will be used to train our model while the second to predict sample classes. We use ***train_test_split*** from ***sklearn.model_selection*** to simplify the splitting part of the function.
This function requires all of the data that would be required for the initial training model, with extra parameters being **num_runs(=100)** indicating the number of iterations and **size(=0.25)** indicating the test set size (**size***100%). By default the function will split the data randomly into 75%-25%, repeating this as many times indicated in **num_runs** (set to 100 as default).
We iterate many times so we can compute the accuracy of our model. For this purpose, we count the correct predictions divided by the total number of predictions for each iteration, store it, and then return the mean accuracy.

D) Laplace hyperparameter:

To check how the hyperparameter affects the accuracy for categorical classification, we iterate through different values of L and observe the resulting accuracy. For each value of L we will perform 100 random 75%-25% train-test splits on the data and calculate the mean accuracy. For this purpose the values chosen were: L = [0,1,2,4,10,15,30,50,100,1000]
With this we include no smoothing, and very high levels of smoothing.

The results are shown in the table below.

Using Laplace hyperparameter L = 1: Average accuracy for categorical data: 85.91% Using Laplace hyperparameter L = 2: Average accuracy for categorical data: 85.33% Using Laplace hyperparameter L = 4: Average accuracy for categorical data: 85.49% Using Laplace hyperparameter L = 10: Average accuracy for categorical data: 84.93% Using Laplace hyperparameter L = 15: Average accuracy for categorical data: 84.21% Using Laplace hyperparameter L = 30: Average accuracy for categorical data: 82.33% Using Laplace hyperparameter L = 50: Average accuracy for categorical data: 81.63% Using Laplace hyperparameter L = 100: Average accuracy for categorical data: 81.00% Using Laplace hyperparameter L = 1000: Average accuracy for categorical data: 75.46%
Average accuracy for continuous data: 94.76%

We notice a very distinct negative correlation with the values of hyperparameter L and the model's accuracy. As the hyperparameter increases, the model's accuracy decreases.
This suggests that while a small amount of Laplace smoothing (such as L=1) can help dealing with zero probabilities, excessive smoothing decreases the model's performance. This over-smoothing can be clearly seen when comparing accuracy for L=1 (Accuracy = 85.91%) and for L=1000 (Accuracy = 75.45%) where we have a substantial drop of 10% in our accuracy, so our class predictions.

To generalize the steps for both discrete and continuous data, NBC computes the posterior probability of each class for a given sample by multiplying the prior probability of the class with the product of the likelihoods of each feature. The predicted class is the one with the highest posterior probability. The main difference lies in how the likelihood is computed: for discrete data, it's based on frequency counts (with Laplace smoothing), and for continuous data, it's based on the Gaussian distribution.