Project 2.1

February 7, 2023

0.1 Approach

Creating specific function for each quantities, reading the data file first, performing calculations and returning the values resulting from the calculations in the form of a dictionary so that we would be able to retrieve the values as required.

[1]:
```
# Imporing the required library
import pandas as pd
```

[2]:
```
def prior(data):
    df = pd.read_csv(data, sep=" ")
    prior_Yes = df[df.Oracle == "Yes"].shape[0]/df.shape[0]
    prior_No = 1-prior_Yes
    return {"prior_Yes": prior_Yes, "prior_No": prior_No}

prior("sample.data")
```

[2]:
```
{'prior_Yes': 0.48, 'prior_No': 0.52}
```

0.2 Function to get Prior Probabilities

The function above, called *prior*, has one parameter that needs to be passed when it is called and that would be the name of the file with the file extension as string. Since the data that we are using has columns separated by one spaces (" "), we read in the file as a pandas data frame and specify how the columns are separated. This will be the same for all the functions that are created for this part of the project.

Once the data is read in we focus our attention towards getting prior probabilities for each class in our data. We have two classes (Yes and No), so we create two filters for our data frame that can filter out rows for each of the classes as separate data frames. The *shape* method from the pandas library is used to get a list, the first element of which is the number of rows and the second element is the number of columns in the data frame ([rows, columns]). Using this method, we divide the number of rows that contain one of the classes by the total number of rows in the original data frame which would give us the prior probability of that class. Since we only have 2 classes, subtracting the prior probability for one class from 1 would give us the prior probability of another class. After the calculation, we return a dictionary with the quantities and we do this for all the functions created below too.

```
Prior: estimate the probability of each class:
    P(cj) = #cj / #training examples
```

[3]:
```
def likelihood(data):
    df = pd.read_csv(data, sep=" ")
    like_A_Yes = df[df.Oracle == "Yes"][df[df.Oracle == "Yes"].Attribute ==␣
 ↪"A"].shape[0] / df[df.Oracle == "Yes"].shape[0]
    like_A_No = df[df.Oracle == "No"][df[df.Oracle == "No"].Attribute == "A"].
 ↪shape[0] / df[df.Oracle == "No"].shape[0]
    like_B_Yes = 1-like_A_Yes
    like_B_No = 1-like_A_No
    return {"like_A_Yes": like_A_Yes, "like_A_No": like_A_No, "like_B_Yes":␣
 ↪like_B_Yes, "like_B_No": like_B_No}
likelihood("sample.data")
```

[3]:
```
{'like_A_Yes': 0.6666666666666666,
 'like_A_No': 0.38461538461538464,
 'like_B_Yes': 0.3333333333333333,
 'like_B_No': 0.6153846153846154}
```

0.3 Function to get Likelihood

```
Likelihood: estimate the probability of each attribute value ai, given a class of type j
    P(ai | cj) = #ai / #cj
```

According to the above description and formula to calculate likelihood, we need to get the number of rows with a combination of each attribute and class. In our case, since we have 2 attributes and 2 classes, we would have to get 4 values (number of rows where attribute is "A" and class is "Yes", number of rows where attribute is "A" and class is "No", number of rows where attribute is "B" and class is "Yes", number of rows where attribute is "B" and class is "No")

To get these values we filter our original data frame to get a data frame that has the specific class we want and we filter the resulted data frame yet again to get the data frame with the specific attribute we want. We get the value for number of rows using the *shape* method on each of these data frames and divide these values by the number of rows of the respected class to get the likelihood according to the formula mentioned above.

[4]:
```
def evidence(data):
    df = pd.read_csv(data, sep=" ")
    evi_A = df[df.Attribute == "A"].shape[0]/df.shape[0]
    evi_B = 1-evi_A
    return {"evi_A": evi_A, "evi_B": evi_B}

evidence("sample.data")
```

[4]:
```
{'evi_A': 0.52, 'evi_B': 0.48}
```

0.4 Function to get Evidence

```
Evidence: estimate the probability of each attribute value:
    P(ai) = #ai / #training examples
```

Getting the evidence is pretty much the same process as getting the prior probabilities but instead of getting the probabilities of getting each of the different classes, we get the probabilities of getting each of the different attributes. We filter out the data frame for each class, get it's number of rows and divide it with the total rows in the training set to get the evidence.

[5]:
```
def posterior(data):
    post_Yes_A = round((likelihood(data)["like_A_Yes"] *␣
 ↪prior(data)["prior_Yes"])/evidence(data)["evi_A"], 3)
    post_Yes_B = round((likelihood(data)["like_B_Yes"] *␣
 ↪prior(data)["prior_Yes"])/evidence(data)["evi_B"], 3)
    post_No_A = round((likelihood(data)["like_A_No"] * prior(data)["prior_No"])/
 ↪evidence(data)["evi_A"], 3)
    post_No_B = round((likelihood(data)["like_B_No"] * prior(data)["prior_No"])/
 ↪evidence(data)["evi_B"], 3)
    return {"post_Yes_A": post_Yes_A, "post_Yes_B": post_Yes_B, "post_No_A":␣
 ↪post_No_A, "post_No_B": post_No_B}
posterior("sample.data")
```

[5]:
```
{'post_Yes_A': 0.615,
 'post_Yes_B': 0.333,
 'post_No_A': 0.385,
 'post_No_B': 0.667}
```

[10]:
```
print("The likelihood that a new instance is of class 'Yes', given an attribute␣
 ↪value of 'A' is {}.".format(posterior("sample.data")["post_Yes_A"]))
```

```
The likelihood that a new instance is of class 'Yes', given an attribute value
of 'A' is 0.615.
```

0.5 Function to get Posterior Probabilities

```
Posterior: estimate the probability of a class j after seeing attribute i
    P(cj | ai) = [P(ai | cj) · P(cj)] / P(ai)
```

According to the formula to get the posterior probability, we multiply the likelihood with the prior probability and divide the result by the evidence for specific attribute and class. For example, to find the probability of getting a "Yes" class given that the attribute is "A" (Posterior), we multiply the probability of having attribute "A" given class of "Yes" (Likelihood) with the probability of getting a "Yes" class (Prior) and divide the result with the probability of getting an attribute value of "A" (Evidence).

Since we have all the functions required to calculate the posterior probability, we just call those functions with the required values and perform the calculation and return all the posterior probabilies in a **dictionary**.

0.6 Conclusions

It was a pretty straightforward project and we did not find any specific difficulty while working on it. We were just confused between returning all of the quantities in a single function or returning

them separately by creating a function for each. At last, we decided to use a spearate function for each quantities thinking it might be more useful in Part II of the project.