Project 2.2

February 14, 2023

1 Naïve Bayes for Classification

[2]:
```
# Imporing the required library
import pandas as pd
```

We made few changes to the functions that we created in part I as the one attribute in part I only had two labels (A and B). However, in part II we have two labels in some attributes and 3 in some. To accomodate this, we make use of **kwargs, which is a way to pass variable length list of keyworded arguments. With this in place, we don't need to worry about the number of labels that we have in any of the attributes. It should be noted that, **for this implementation we need to pass the arguements in a certain order.** The first argument should be the data that we want to work on, second would be the name of the variable or column, and labels/classes after that. The functions we submitted in part I had probabilities for each label hardcoded but here we make use of a loop which loops through all the classes an attribute might have and returns those probabilities as a dictionary, which is the data structure we have used throughout this project. The reason behind using a dictionary is the ability it gives us to access the values we need easily without having to remember the index number where each element is if we would have to if we used list instead.

[3]:
```
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}
```

[4]:
```
def likelihood(**kwargs):
    df = pd.read_csv(list(kwargs.values())[0], sep=" ")
    attr = list(kwargs.values())[1]
    d = {}
    for i in range(2,len(kwargs)):
        d["like_"+list(kwargs.values())[i]+"_Yes"] = df[df.Oracle ==␣
↪"Yes"][df[df.Oracle == "Yes"][attr] == list(kwargs.values())[i]].shape[0] /␣
↪df[df.Oracle == "Yes"].shape[0]
        d["like_"+list(kwargs.values())[i]+"_No"] = df[df.Oracle == "No"][df[df.
↪Oracle == "No"][attr] == list(kwargs.values())[i]].shape[0] / df[df.Oracle␣
↪== "No"].shape[0]
    return(d)
```

```
[5]: def evidence(**kwargs):
         df = pd.read_csv(list(kwargs.values())[0], sep=" ")
         attr = list(kwargs.values())[1]
         d = {}
         for i in range(2,len(kwargs)):
             d["evi_"+list(kwargs.values())[i]] = df[df[attr] == list(kwargs.
      ↪values())[i]].shape[0]/df.shape[0]
         return(d)
```

```
[6]: def posterior(**kwargs):
         df = pd.read_csv(list(kwargs.values())[0], sep=" ")
         attr = list(kwargs.values())[1]
         d = {}
         for i in range(2,len(kwargs)):
             d["post_Yes_"+list(kwargs.values())[i]] =␣
      ↪round((likelihood(**kwargs)["like_"+list(kwargs.values())[i]+"_Yes"] *␣
      ↪prior(list(kwargs.values())[0])["prior_Yes"])/
      ↪evidence(**kwargs)["evi_"+list(kwargs.values())[i]], 3)
             d["post_No_"+list(kwargs.values())[i]] =␣
      ↪round((likelihood(**kwargs)["like_"+list(kwargs.values())[i]+"_No"] *␣
      ↪prior(list(kwargs.values())[0])["prior_No"])/
      ↪evidence(**kwargs)["evi_"+list(kwargs.values())[i]], 3)
         return(d)
```

Now that we have a function to calculate diferent kinds of probabilities we need, it is now time to read in our data and make use of the functions to get probabilities that will help us classify new instances as whether it is a good day to fish or not.

```
[65]: fish = pd.read_csv("fishing.data", sep = " ") # Reading in the data which is␣
       ↪separated by spaces (" ").
```

```
[25]: fish
```

```
[25]:   Oracle    Wind   Air     Water     Sky
      0    Yes  Strong   Hot      Warm   Sunny
      1     No    Weak   Hot      Warm   Sunny
      2    Yes  Strong   Hot      Warm  Cloudy
      3    Yes  Strong   Hot  Moderate   Rainy
      4     No  Strong  Cold      Cool  Cloudy
      5     No    Weak  Cold      Cool   Rainy
      6     No    Weak  Cold      Cool   Sunny
      7    Yes  Strong   Hot  Moderate   Sunny
      8    Yes  Strong  Cold      Cool   Sunny
      9     No  Strong  Cold  Moderate   Rainy
```

```
10    Yes    Weak  Cold  Moderate   Sunny
11    Yes    Weak   Hot  Moderate   Sunny
12    Yes  Strong  Cold      Warm   Sunny
13     No    Weak   Hot  Moderate   Rainy
```

1.0.1 Prior Probability for the dependent variable

To classify any given instance based on Naive Bayes' Classification, we need the prior probability for the different classes in the dependent variables. This will be calculated and returned by the *prior* function as a dictionary. It returns a dictionary with two key-value pairs as we have two classes (Yes or No). We decided not to use a *round* function here in an effort to get a good probability estimate which would be calculated later.

[140]:
```
prior("fishing.data")
```

[140]:
```
{'prior_Yes': 0.5714285714285714, 'prior_No': 0.4285714285714286}
```

Once we have the prior probability, we will now look at the likelihood, estimates and posterior probability which we would calculate and return with the functions we created above for all the independent variables. We will store each of the results in a variable and use these variables below where we create another function called *model* where we will pass new instances and return a result as to whether or not it is a good day to fish.

1.0.2 Wind

[69]:
```
wind_likelihood = likelihood(data = "fishing.data", attr = "Wind", c1 =␣
 ↪"Strong", c2 = "Weak")
wind_likelihood
```

[69]:
```
{'like_Strong_Yes': 0.75,
 'like_Strong_No': 0.3333333333333333,
 'like_Weak_Yes': 0.25,
 'like_Weak_No': 0.6666666666666666}
```

[70]:
```
wind_evidence = evidence(data = "fishing.data", attr = "Wind", c1 = "Strong",␣
 ↪c2 = "Weak")
wind_evidence
```

[70]:
```
{'evi_Strong': 0.5714285714285714, 'evi_Weak': 0.42857142857142855}
```

[71]:
```
wind_posterior = posterior(data = "fishing.data", attr = "Wind", c1 = "Strong",␣
 ↪c2 = "Weak")
wind_posterior
```

[71]:
```
{'post_Yes_Strong': 0.75,
 'post_No_Strong': 0.25,
 'post_Yes_Weak': 0.333,
 'post_No_Weak': 0.667}
```

1.0.3 Air

[72]:
```
air_likelihood = likelihood(data = "fishing.data", attr = "Air", c1 = "Hot", c2␣
 ↪= "Cold")
air_likelihood
```

[72]:
```
{'like_Hot_Yes': 0.625,
 'like_Hot_No': 0.3333333333333333,
 'like_Cold_Yes': 0.375,
 'like_Cold_No': 0.6666666666666666}
```

[73]:
```
air_evidence = evidence(data = "fishing.data", attr = "Air", c1 = "Hot", c2 =␣
 ↪"Cold")
air_evidence
```

[73]:
```
{'evi_Hot': 0.5, 'evi_Cold': 0.5}
```

[74]:
```
air_posterior = posterior(data = "fishing.data", attr = "Air", c1 = "Hot", c2 =␣
 ↪"Cold")
air_posterior
```

[74]:
```
{'post_Yes_Hot': 0.714,
 'post_No_Hot': 0.286,
 'post_Yes_Cold': 0.429,
 'post_No_Cold': 0.571}
```

1.0.4 Water

[76]:
```
water_likelihood = likelihood(data = "fishing.data", attr = "Water", c1 =␣
 ↪"Warm", c2 = "Moderate", c3 = "Cool")
water_likelihood
```

[76]:
```
{'like_Warm_Yes': 0.375,
 'like_Warm_No': 0.16666666666666666,
 'like_Moderate_Yes': 0.5,
 'like_Moderate_No': 0.3333333333333333,
 'like_Cool_Yes': 0.125,
 'like_Cool_No': 0.5}
```

[77]:
```
water_evidence = evidence(data = "fishing.data", attr = "Water", c1 = "Warm",␣
 ↪c2 = "Moderate", c3 = "Cool")
water_evidence
```

[77]:
```
{'evi_Warm': 0.2857142857142857,
 'evi_Moderate': 0.42857142857142855,
 'evi_Cool': 0.2857142857142857}
```

[78]:
```
water_posterior = posterior(data = "fishing.data", attr = "Water", c1 = "Warm",␣
↪c2 = "Moderate", c3 = "Cool")
water_posterior
```

[78]:
```
{'post_Yes_Warm': 0.75,
 'post_No_Warm': 0.25,
 'post_Yes_Moderate': 0.667,
 'post_No_Moderate': 0.333,
 'post_Yes_Cool': 0.25,
 'post_No_Cool': 0.75}
```

1.0.5 Sky

[79]:
```
sky_likelihood = likelihood(data = "fishing.data", attr = "Sky", c1 = "Sunny",␣
↪c2 = "Cloudy", c3 = "Rainy")
sky_likelihood
```

[79]:
```
{'like_Sunny_Yes': 0.75,
 'like_Sunny_No': 0.3333333333333333,
 'like_Cloudy_Yes': 0.125,
 'like_Cloudy_No': 0.16666666666666666,
 'like_Rainy_Yes': 0.125,
 'like_Rainy_No': 0.5}
```

[80]:
```
sky_evidence = evidence(data = "fishing.data", attr = "Sky", c1 = "Sunny", c2 =␣
↪"Cloudy", c3 = "Rainy")
sky_evidence
```

[80]:
```
{'evi_Sunny': 0.5714285714285714,
 'evi_Cloudy': 0.14285714285714285,
 'evi_Rainy': 0.2857142857142857}
```

[81]:
```
sky_posterior = posterior(data = "fishing.data", attr = "Sky", c1 = "Sunny", c2␣
↪= "Cloudy", c3 = "Rainy")
sky_posterior
```

[81]:
```
{'post_Yes_Sunny': 0.75,
 'post_No_Sunny': 0.25,
 'post_Yes_Cloudy': 0.5,
 'post_No_Cloudy': 0.5,
 'post_Yes_Rainy': 0.25,
 'post_No_Rainy': 0.75}
```

To classify a new instance using Naive Bayes', we need the prior probability for each class of the dependent variable and multiply them with the likelihood for all independent variable's classes given that particular class of dependent variable. In our case where our new instance has Strong Wind, Hot Air, Cool Water and Sunny Sky, we need to multiply the probability that it is a good day to fish with the likelihoods for each classes given it is a good day to fish and also multiply the probability that it is not a good day to fish with the likelihoods for each classes given it is not a good day to fish. This gives us to probabilities, one that says it is a good day to fish given the probabilities of the observed data and the other that says it is not a good day to fish given the probabilities of the observed data. Which ever's probability is higher, we assign that label to the instance.

1.0.6 New Instance: Strong Hot Cool Sunny

[115]:
```python
new_instance = {"Wind": "Strong", "Air": "Hot", "Water":"Cool", "Sky": "Sunny"}
```

[119]:
```python
def model(new_instanace):

    #Wind
    if new_instance["Wind"] == "Strong":
        prob1_Yes = wind_likelihood["like_Strong_Yes"]
        prob1_No = wind_likelihood["like_Strong_No"]
    else:
        prob1_Yes = wind_likelihood["like_Weak_Yes"]
        prob1_No = wind_likelihood["like_Weak_No"]

    # Air
    if new_instance["Air"] == "Hot":
        prob2_Yes = air_likelihood["like_Hot_Yes"]
        prob2_No = air_likelihood["like_Hot_No"]
    else:
        prob2_Yes = air_likelihood["like_Cold_Yes"]
        prob2_No = air_likelihood["like_Cold_No"]

    # Water
    if new_instance["Water"] == "Warm":
        prob3_Yes = water_likelihood["like_Warm_Yes"]
        prob3_No = water_likelihood["like_Warm_No"]
    elif new_instance["Water"] == "Moderate":
        prob3_Yes = water_likelihood["like_Moderate_Yes"]
        prob3_No = water_likelihood["like_Moderate_No"]
    else:
        prob3_Yes = water_likelihood["like_Cool_Yes"]
        prob3_No = water_likelihood["like_Cool_No"]

    # Sky
    if new_instance["Sky"] == "Sunny":
        prob4_Yes = sky_likelihood["like_Sunny_Yes"]
        prob4_No = sky_likelihood["like_Sunny_No"]
    elif new_instance["Sky"] == "Cloudy":
        prob4_Yes = sky_likelihood["like_Cloudy_Yes"]
        prob4_No = sky_likelihood["like_Cloudy_No"]
    else:
        prob4_Yes = sky_likelihood["like_Rainy_Yes"]
        prob4_No = sky_likelihood["like_Rainy_No"]

    yes = prior("fishing.data")["prior_Yes"] * prob1_Yes * prob2_Yes *␣
  ↪prob3_Yes * prob4_Yes
    no = prior("fishing.data")["prior_No"] * prob1_No * prob2_No * prob3_No *␣
  ↪prob4_No

    return {"Yes": yes, "No": no, "Result": "Yes" if yes>no else "No"}

print("Therefore C(NB) =", model(new_instance)["Result"]+".")
print("The conditional probability that the class is Yes, given the observed␣
  ↪attribute values, is:", str(100 * (round(model(new_instance)["Yes"]/
  ↪(model(new_instance)["Yes"]+model(new_instance)["No"]), 4)))+"%.")
```

```
Therefore C(NB) = Yes.
The conditional probability that the class is Yes, given the observed attribute
values, is: 75.98%.
```

[122]:
```
print(model(new_instance)["Yes"])
print(model(new_instance)["No"])
```

```
0.025111607142857144
0.007936507936507936
```

The function called *model* that is created and used above makes use of the *prior* and *likelihood* function that we used above to calculate the probability of each class, given the probabilities of the observed data for any new instance we pass. *model* takes one argument which is a dictionary of the new instance where the keys are the names of the variables and values are the classes for each of those variables.

When we pass that dictionary in the function, it checks the values for each of the independent variables and stores the value of likelihoods for that value given each class. In our instance, the value for Wind is “Strong” so the function stores the probability of strong given a good day to fish in *prob1_Yes* and probability of weak given not a good day to fish in *prob1_No*. The function does this for all the independent variables and finally multiplies all probabilites of a given class together with prior probability for that class. The return statement in the function then returns a dictionary where we have the calculated probability for Yes class and for No class and a result which has a value of “Yes” or “No” depending on whichever probability is higher.

For our given new instance the probability of Yes class, given the probabilities of the observed data came out to be about 2.51% and the probability of No class, given the probabilities of the observed data came out to be about 0.793%. Because of this, we label the instance as “Yes”.