Assignment 2

Computer Science Department, University of Crete
MACHINE LEARNING - CS 577, Fall 2022

Student: Mourouzidou Eleni, Msc Bioinformatics

Exercise 1 - Probabilities (Theoretical)

Let the random variable X follow the distribution:

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

a. Find the expression describing the MLE estimators for θ for N independent identically distributed (i.i.d.) samples. How can you be sure that this value you found is indeed the maximum?

First, we will find the likelihood function for θ as:

$L(\theta) = \theta^{2N}(x_1 + 1)(1 - \theta)^{x1} *(x2 + 1)(1 - \theta)^{x2} * (x_n + 1)(1 - \theta)^{xn}$

where x_1 x_2 x_n describe the N samples.

Then we calculate the logarithmic likelihood function as:

$l(\theta) = \log (L(\theta)) = 2N*\log(\theta) + \sum_{i=1}^{N} \log(x_i + 1) + x_i*\log(1 - \theta)$

To find the MLE for θ we will calculate the derivative of l(θ)

$$\frac{dl(\theta)}{d(\theta)} = \frac{2N}{\Theta} - \sum_{i=1}^{N} \frac{xi}{1-\theta}$$

set the derivative equal to zero and calculate θ:

$$\theta = \frac{2N}{\sum_{i=1}^{N} \frac{xi}{1-\theta}}$$

To be sure that this value is indeed the maximum we just need to check whether the second derivative of log likelihood is negative.

$$\frac{d^2 l(\theta)}{d^2(\theta)} = \frac{-2N}{\Theta^2} - \sum_{i=1}^{N} \frac{xi}{(1-\theta)^2}$$

since N, θ and x are positive values we have:

$$\frac{-2N}{\Theta^2} < 0\,, \quad - \sum_{i=1}^{N} \frac{xi}{(1-\theta)^2} < 0$$

So since the summation of two negative values is a negative value, we can conclude that the second derivative of the log likelihood is actually <0 and θ is indeed the maximum.

b. Calculate θ for f(x;θ) using the formula calculated in the first step, and applying it to the following 15 samples:

[3.2, 1.4, 2.2, 7, 0.5, 3.3, 9, 0.15, 2, 3.21, 6.13, 5.5, 1.8, 1.2, 11]

Since the formula found in the previous step is non linear as we have to handle an equation of this format : $l(\theta) = 2N*\log(\theta) + \sum_{i=1}^{N} \log(x_i + 1) + x_i*\log(1 - \theta),$ we will make iterations until converge to solve it, using python using Newton -Raphson method. The code is given in the attached python files and the result obtained is θ = 0.34250411814681014

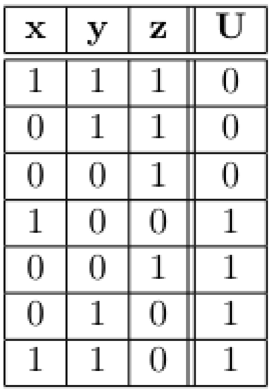
Exercise 2 - Na¨ıve Bayes (Theoretical) [15 points]

Consider Table 1, presenting a dataset with 7 samples, each comprised of three Boolean variables x, y and z, and a Boolean target variable U . You will use this data to train a Na¨ıve Bayes classifier and predict U . Specifically:

a. [5 points] After learning is complete, what would be the predicted probability
P (U = 0|x = 0, y = 0, z = 1)?

We will use the simple Bayesian formula P(A|B) = P(B|A) * P(A) / P(B) to calculate the probability
P (U = 0|x = 0, y = 0, z = 1)?

Table 1:



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

We have :
P(x = 0, y= 0, z = 1) = 2/7
P(x = 0, y= 0, z = 1 | U = 0) = 1/3
P(U = 0) = 3/7

So we plug in those probabilities to the simple form of Bayes Theorem **P(A|B) = P(B|A) * P(A) / P(B)**.

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

$$= \frac{\frac{1}{3} * \frac{3}{7}}{\frac{2}{7}} = \frac{1}{2}$$

b. [5 points] Why — in this case — we did not need to exploit the Laplace trick to solve question a? Hint: Try to solve for P(U = 0|x = 0, y = 0, z = 0) and check which element in the Na¨ıve Bayes classifier formula causes the method to fail.

For the scenario that we have to calculate the probability of P(U = 0|x = 0, y = 0, z = 0) it would be necessary to exploit the Laplace trick. This is because there are no instances of the combination x = 0 , y = 0 , and z = 0 in the given Table 1. Thus, the probability would be equal to 0 and we would not be able to proceed with a denominator = 0, so adding a small constant would serve to calculate the desired probability. In the previous question it was not necessary as we had to calculate the probability of instances that occurred in our dataset.

c. Using the probabilities obtained during the Bayes Classifier training, what would be the predicted probability P (U = 1|x = 0)?

$$p(U=1|x=0) = \frac{p(x = 0 \mid U = 1) * p(U = 1)}{P(x = 0)}$$

From table 1 we have :

- $p(x = 0 \mid U = 1) = \frac{2}{4}$, since U=1 occurs 4 times in total and for 2 of them, x = 0
- $p(U = 1) = \frac{4}{7}$, out of 7 samples 4 have U=1
- $p(x = 0) = \frac{4}{7}$

$$p(U=1|x=0) = \frac{\frac{2}{4} * \frac{4}{7}}{\frac{4}{7}} = \frac{1}{2} \text{ or } 0.5$$

Exercise 3 - Naïve Bayes Classifier

a. Implement the NBC training function:

We first implemented *train* function in order to train the model of Naive Bayes Classifier.

Function Inputs:

- "X" describes the data table where each row represents a sample for which we want to predict a class where it belongs and each column represents a feature that describes this sample.
- "X_dtype" takes two possible inputs , "continuous" or "categorical" that indicate as said, whether our data are continuous (could be measurements like height or salaries) or categorical (could be categories or a True/ False statement represented by 1/0 for example). Defining the d_type is crucial to implement the *train* function since there will be used different calculation methods.

    - For categorical data nbc retrieves the frequency of the data to calculate the conditional probabilities for each feature given the class using also Laplace smoothing.

```
for v in range(D_categorical[i]):
    occurrences = np.sum((X[:, i] == v) & (Y == cl))
    self.feature_probabilities[i][(cl,v)]=(occurrences +
        self.L) / (np.sum(Y == cl) + self.L * D_categorical[i])
```

    - For continuous data nbc calculates the conditional probability for the features by calculating the mean value and the standard deviation for each feature(i) within a class (cl)

```
for i in range(M):
      self.feature_means[i] = {cl: np.mean(X[Y == cl, i]) for cl in
          classes}
      self.feature_stds[i] = {cl: np.std(X[Y == cl, i]) for cl in
          classes}
```

- “Y” this Ix1 dataset “holds” the labels or classes that we would like nbc algorithm to predict and fit the data.
- “L” (for Laplace smoothing) hyperparameter is a scalar used to smooth our data and mainly handle zero counts that would lead to zero probabilities when having categorical data.
- “D_categorical” this variable is set to None if the input data are continuous, otherwise it takes an input dataset describing how many categories can a feature have (each column)

In general the purpose of the *train* function is to train the Naive Bayes Classification model using an input dataset by estimating the probabilities required to classify the data to certain predicted classes.

If d_type == “categorical”:
The function first calculates the prior probability for each class as P(Y) by counting the occurrence of each class and adding the Laplace constant.For each feature (columns of X) it calculates the conditional probabilities as $P(X_i \mid Y)$ for all the possible categories of the features given each class of the Y dataset. Laplace smoothing is also applied to handle the case that some categories might not occur in our dataset

If d_type == “continuous”:
The prior probability is calculated similarly to the categorical data. Then, since the data are continuous the function proceeds to the calculation of the mean and standard deviation for each feature within each class and stores these values to the structures self.feature_means and self_feature_stds respectively. According to these statistics
the probability density function is calculated under the assumption that the continuous data follow a Gaussian distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The probabilities are multiplied together with the prior probability of the class to finally retrieve the probability of a sample to belong to that certain class. The model finally chooses the class with the highest probability for each sample.

Class and conditional probabilities as well as mean and std statistics are stored as attributes of the class NaiveBayes and will further be used to implement the *predict* function.

b.Implement the NBC prediction function

The *predict* function uses the following attributes of NaiveBayes class to predict the class of new unknown datasets:
self.class_probabilities,
self.feature_probabilities,
self.feature_means,
and self.feature_stds,
it calculates the probabilities of each sample based on the input data “X” and “X_dtype”.
The NaiveBayes class stored the prior and conditional probabilities *calculated by train* function and *predict* function uses them to calculate the likelihood of the features of each class. Thus, the model is not directly used as an input to *predict* function, instead, the function has access to the necessary model attributes since it belongs to the same class as the *train* function.

The function, considers the likelihood of each sample’s features given each class combined with the prior probability of each class to occur. For each feature i the function calculates a conditional probability given a class c. Then the conditional probability of each feature is multiplied by the one of all the other features of the sample and the prior probability of the class. In this way it calculates the probability of the sample to belong to each class and chooses as the predicted class, the one with the highest probability. The predicted classes are stored and returned as an output of the *predict* function.

c. Assess the classifier using the datasets uploaded along with the assignment

We first compare the predicted labels derived from the prediction function to the actual labels given by the training dataset, with the function *calculate accuracy*

To evaluate the Naive Bayes Classifier we use *evaluate_naive_bayes* function that will assess the model using different subsets with many iterations -100 in our case-.

The training dataset is passed as an input and for a specified number of iterations the function:

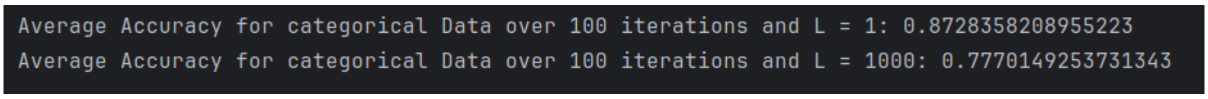
- loads the given dataset and the labels files

- splits the dataset randomly to a defined ratio -75% in this case- training and -25% in this case- test subsets
- calls the model derived from the *train* function to access variables such as th dataset (X), the data type (d_type) and the Laplace constant (L)
- predicts labels and store them to prediction_cat and prediction_cont for categorical and continuous data respectively
- calls *calculate_accuracy* function to finally calculate the accuracy of the model trained on these 100 datasets.

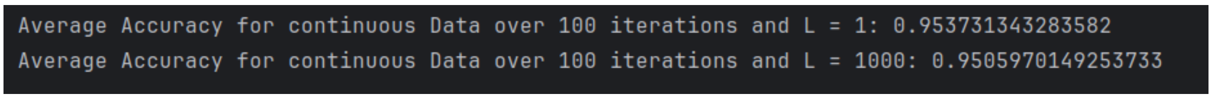
By iterating 100 times using different random subsets we can ensure a more reliable assessment of our model as it becomes less biased and related to the dataset that we feed it. It targets to better generalize the model and perform well on unseen new datasets.

d.How does the choice of the hyperparameter L affect the results, in the case of the categorical classification? Experiment with small and large values of L.

We can observe that for small L values as L=1 (add one smoothing) the accuracy of the model is higher than for bigger L values as L=1000. This is because the model for lower L values is highly related to the feeding data and regardless of the better fitting on the training dataset and the high accuracy there is a greater risk that when applying the model to unseen data the performance would not necessarily be good. Higher L, as we can see, leads to lower accuracy, however it enhances the generalization ability of the model to new data and tends to form a uniform distribution. It is also obvious from our results below, that Laplace smoothing does not significantly affect continuous data since it is also used to handle the prior probability of a class but not for the conditional feature probabilities. It would also be impractical and nonsense to apply a constant to infinite values between a certain range.



```
Average Accuracy for categorical Data over 100 iterations and L = 1: 0.8728358208955223
Average Accuracy for categorical Data over 100 iterations and L = 1000: 0.7770149253731343
```



```
Average Accuracy for continuous Data over 100 iterations and L = 1: 0.953731343283582
Average Accuracy for continuous Data over 100 iterations and L = 1000: 0.9505970149253733
```

To conclude, it seems that for low L values the model performs well to familiar data but tends to overfit to the training dataset. On the other hand higher L values significantly generalizes the model but can lead to a very simplistic model with the risk of underfitting the data.