Machine Learning Assignment-1

Autumn Semester, 2022

Team Members:

Anand Manojkumar Parikh (20CS10007)

Soni Aditya Bharatbhai (20CS10060)

Naïve Bayes Classifier

[* Refer the readme file in this folder for directions to run the code]

Contents:

1) Step-wise approach

The following procedure has been used to code the Naïve Bayes Classifier from scratch (some details have separate sections dedicated to to them specifically)

- Extracting the dataset from a csv file into a pandas dataframe
- Observe that all labels are “Abnormal” or “Normal”, so wee need to encode these values to 0 and 1 respectively
- Also observe that no data points are missing or empty, so there is no requirement of handling missing data. (In case it is required, we can replace that by the mean of the feature value)
- Next, some training examples might have outliers, so those training samples are deleted from the dataframe.
- Once we have all the useful data, we randomize the dataset to achieve maximum variation in our future training and testing runs. In case this step is skipped, the accuracy and reliability of the model might drop drastically, since we might encounter cases where all the training examples are positive, but all the testing examples are negative, leading to bad classification!
- Now, we separate the X (feature values) and y (target labels) from the dataframe named “df”.
- Then, we perform a 70%-30% split of the X and y features, and never use the 30% for training anywhere. We basically keep 30% of our data reserved (unseen) to the trainer, so that we can actually check if the model learns.
- Then, we use the Naïve Bayes Classifier to classify the values of the testing dataset and store it in “y_predicted”. The accuracy can easily be found by calculating the percentage of examples correctly classified.
- The internal details of the Naïve Bayes Classifier are as follows:

    1. Calculation of prior probabilities of classifications 0 and 1 (function calculate_priors):
        - prior[0] = no. of 0’s / (no. of 0’s + no. of 1’s)
        - prior[0] = no. of 1’s / (no. of 0’s + no. of 1’s)

2. Calculation of likelihoods (function calculate_likelihoods)
    - We iterate over each testing example, and find the likelihood of seeing those set of attributes, given classification as 0 or 1. For each training example, we individually call calculate_likelihoods for each attribute.
    - Each call to this function computes the likelihood of seeing that particular attribute value, given the classification as either 0 or 1.
    - Since we have continuous data, a Normal Distribution is fitted to the training set to find the probability (x_i = observed value | y = 0/1) for each testing example.
    - After doing this for each attribute, we find the product of these values to find the likelihood of observing the whole training example, given the classification. We can do this only if we assume independence of attribute values within each example.

3. Calculate posterior probabilities :
    - The title might be misleading since we do not actually calculate the exact values of the posterior probabilities (we can, but we do not need it)
    - We only need the classification (0 or 1) which has the maximum posterior probability. Notice that the denominator of the posterior probability expression does not have the classification, so simply drop it while finding argmax of the probabilities! Now, we can just find the numerator and conclude on basis of that.
    - Finally classify the testing example as the one having higher probability, 0 or 1 (“Abnormal” or “Normal”)

- We then do a 5-fold cross validation of the training examples to find average cross-validation accuracy. And we also re-train the model on the best split and try it out on the 30% testing examples.
- We can use Laplace correction to try and improve the accuracy further.

2. Encoding

- The inbuilt Label Encoder from the python library scikit-learn has been used to convert the target classification values from “Abnormal” and “Normal” to 0 and 1 respectively.
- The objective is to convert the data from a human-readable format to a machine-readable format, hence this is an important part of pre-processing the data.

3. Normalization

- Here, we have not used any explicit normalization of the dataset values.
- When the Naïve Bayes Classifier calls the calculate_likelihoods function, this function uses the feature mean and feature variance to construct a Normal Distribution (on the spot) that best fits that column to calculate the likelihood of individual attribute values, given a classification. This is exactly the same as normalizing the values before-hand and later using a standard normal distribution (mean = 0, std = 1) but is easier to visualise and also does not tamper with the original data.

4. Cross Validation

- The 70% training data that we have is cross-validated within itself to arrive at some useful results and is performed as follows.
- First, partition the training data into 5 equal (nearly equal) sets, which we call folds. We now have fold[0], fold[1], fold[2], fold[3] and fold[4]
- Now, we treat one the folds as the testing data and the remaining 4 as training data. We repeat this process 5 times, such that each fold acts as a test set one time and part of a training set 4 times.
- This stage is generally used to tune parameter values (which are assumed by the programmer) and to reduce overfitting. In our case, we do not have any parameters yet, so this stage is just for observing the cross-validating accuracy.
- The cross-validation accuracy has been defined as the average of the 5 accuracies found each time when the Classifier is trained on the folds.

5. Laplace Correction

- In the final step, we attempt to further improve the model's performance by using Laplace Correction as follows.
- If while calculating the posterior probabilities (while multiplying the likelihoods) any of the probabilities tend to 0 or have a very small value, the whole likelihood value is compromised and might lead to incorrect classification.
- So, we use a self-defined parameter alpha (a) and add a to the numerator and denominator each of the individual attribute likelihoods.
- On multiple trials and tests, alpha = 0.09 seems to yield a good accuracy compared to other values.
- Reason why alpha = 1 is not used : If we use alpha = 1, then we are basically estimating small probabilities to 0.5! Since we convert probability p/1 to (p+1)/(1+1) = (p+1)/2, and p tends to 0, we simply get 0.5. This is bad, since we will never be able to distinguish properly between small values of probabilities! For eg, 0.1 and 0.2 are distinguishable, since the latter is twice the former, but (0.1+1)/2 = 0.55 and (0.2+1)/2 = 0.6 are not, since the ratio is now destroyed!
- So, alpha = 0.09 (trial and error) value is used.

6. Assumptions

- We assume that the dataset, over a large number of observations, fits a Normal Distribution, since it can be concluded from the Central Limit Theorem.
- While calculating the likelihoods of a training example, we assume independence of attributes, and simply multiply individual likelihoods of attribute values (based on the above Normal distribution generated by the training examples).

7. Accuracies

The following accuracies were obtained by running the code 10 times consecutively:

Test Run	Accuracy 1 (70% training, 30% testing)	Accuracy 2 (average of cross-validation accuracies)	Accuracy 3 (best cross-validation split as training, 30% testing)	Accuracy 4 (70% training, 30% testing, with Laplace correction, alpha = 0.09)
1	77.41%	78.68%	75.26%	73.12%
2	75.26%	79.21%	76.34%	74.19%
3	78.49%	77.30%	78.49%	77.41%
4	72.04%	81.01%	72.82%	77.41%
5	84.94%	74.08%	83.22%	86.02%
6	75.26%	77.73%	77.42%	79.56%
7	76.34%	78.22%	77.41%	82.79%
8	76.34%	78.67%	76.34%	73.11%
9	74.19%	79.14%	75.26%	68.81%
10	75.37%	78.16%	75.43%	80.64%

8. Observations

- Sometimes, the values of Accuracy 1 and Accuracy 3 turn out to be EXACTLY the same! This indicates that for these instances, the model learns exactly the same normal distribution from 56% (best 80% split of the 70% data) and the total 70% data.
- Laplace Correction works well at times, but at times it may deteriorate the accuracies also.