Data Mining and Warehouse

TA Activity

Report on

"Naïve Bayes Algorithm"

Submitted By

1. Vedant Shashikant Deshmukh [211101011]
2. Shrutik Kailas Patil [211101012]
3. Jagruti Mahendrasing Rajput [211101013]
4. Sanika Vijay Joshi [211101014]
5. Mitesh Suryakant Patil [211101015]

Under the Guidance of

Prof. Dr. D. R. Patil

R. C. PATEL
INSTITUTE OF TECHNOLOGY
An Autonomous Institute

Department of Computer Engineering

The Shirpur Education Society's

R. C. Patel Institute of Technology, Shirpur - 425405.

[2023-24]

R. C. PATEL
INSTITUTE OF TECHNOLOGY
An Autonomous Institute

Department of Computer Engineering

Vision of the Institute

To achieve excellence in engineering education with strong ethical values.

Mission of the Institute

To impart high quality Technical Education through:

- Innovative and Interactive learning process and high quality, internationally recognized instructional programs.
- Fostering a scientific temper among students by the means of a liaison with the Academia, Industries and Government.
- Preparing students from diverse backgrounds to have aptitude for research and spirit of Professionalism.
- Inculcating in students a respect for fellow human beings and responsibility towards the society.

Vision of the Department

To provide prominent computer engineering education with socio-moral values.

Mission of the Department

M1 To provide state-of-the-art ICT based teaching-learning process.

M2 To groom the students to become professionally sound computer engineers to meet growing needs of industry and society.

M3 To make the students responsible human being by inculcating ethical values.

Program Educational Objectives (PEOs) of the Department

PEO1 To provide the foundation of lifelong learning skills for advancing their careers being a professional, entrepreneur and leader.

PEO2 To develop computer professionals to fulfill industry expectations.

PEO3 To foster ethical and social values to be socially responsible human being.

INDEX

1. INTRODUCTION

One of the most important machine learning techniques for solving classification issues is the Naive Bayes algorithm. It comes from the probability theory of Bayes and is applied to text categorization, where high-dimensional datasets are trained. The Naive Bayes algorithm is best demonstrated by sentiment analysis, spam filtering, and article classification. For the uninitiated data, classification techniques are employed to group fresh observations into predetermined classes. The Naive Bayes Algorithm is renowned for being both easy to use and efficient. Using this approach, model building and prediction are completed more quickly. The Bayes theorem should always be used when developing any machine learning model. Professional machine learning developers must be involved in the application of Naive Bayes algorithms[1].

Why it is called naïve bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of colour , shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other[1].
- **Bayes:** It is called Bayes because it depends on the principle of Bayes' Theorem[1].

Features of Naïve Bayes Algorithm:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles[2].

Probability is the base for the Naive Bayes algorithm. This algorithm is built based on the probability results that it can offer for unsolvable problems with the help of prediction. Let's understand about probability, Bayes theory, and conditional probability below:

Probability:

Probability helps to predict an event's occurrence out of all the potential outcomes. The mathematical equation for probability is as follows:

```
Probability of an event = Number of Favorable Events/ Total number of outcomes
```

0 < = probability of an event < = 1. The favorable outcome denotes the event that results from the probability. Probability is always between 0 and 1, where 0 means no probability of it happening, and 1 means the success rate of that event is likely.

For better understanding, you can also consider a case where you predict a fruit based on its colour and texture. Here are some possible assumptions that you can make. You can either choose the correct fruit that you have in mind or get confused with similar fruits and make mistakes. Either way, the probability of choosing the right fruit is 50%.

Bayes Theorem:

Bayes Theory works on coming to a hypothesis (H) from a given set of evidence (E). It relates to two things: the probability of the hypothesis before the evidence P(H) and the probability after the evidence P(H|E). The Bayes Theory is explained by the following equation:

P(H|E) = (P(E|H} * P(H))/P(E)

In the above equation,

P(H|E) denotes how event H happens when event E takes place.

P(E|H) represents how often event E happens when event H takes place first.

P(H) represents the probability of event X happening on its own.

P(E) represents the probability of event Y happening on its own.

The Bayes Rule is a method for determining P(H|E) from P(E|H). In short, it provides you with a way of calculating the probability of a hypothesis with the provided evidence[2].

Conditional Probability:

Conditional probability is a subset of probability. It reduces the probability of becoming dependent on a single event. You can compute the conditional probability for two or more occurrences.

When you take events X and Y, the conditional probability of event Y is defined as the probability that the event occurs when event X is already over. It is written as P(Y|X). The mathematical formula for this is as follows[2]:

P(Y|A) = P(X and Y) /P(X)

Bayesian Probability:

Bayesian Probability allows to calculate the conditional probabilities. It enables to use of partial knowledge for calculating the probability of the occurrence of a specific event. This algorithm is used for developing models for prediction and classification problems like Naive Bayes.

The Bayesian Rule is used in probability theory for computing - conditional probabilities. What is important is that you cannot discover just how the evidence will impact the probability of an event occurring, but you can find the exact probability[2].

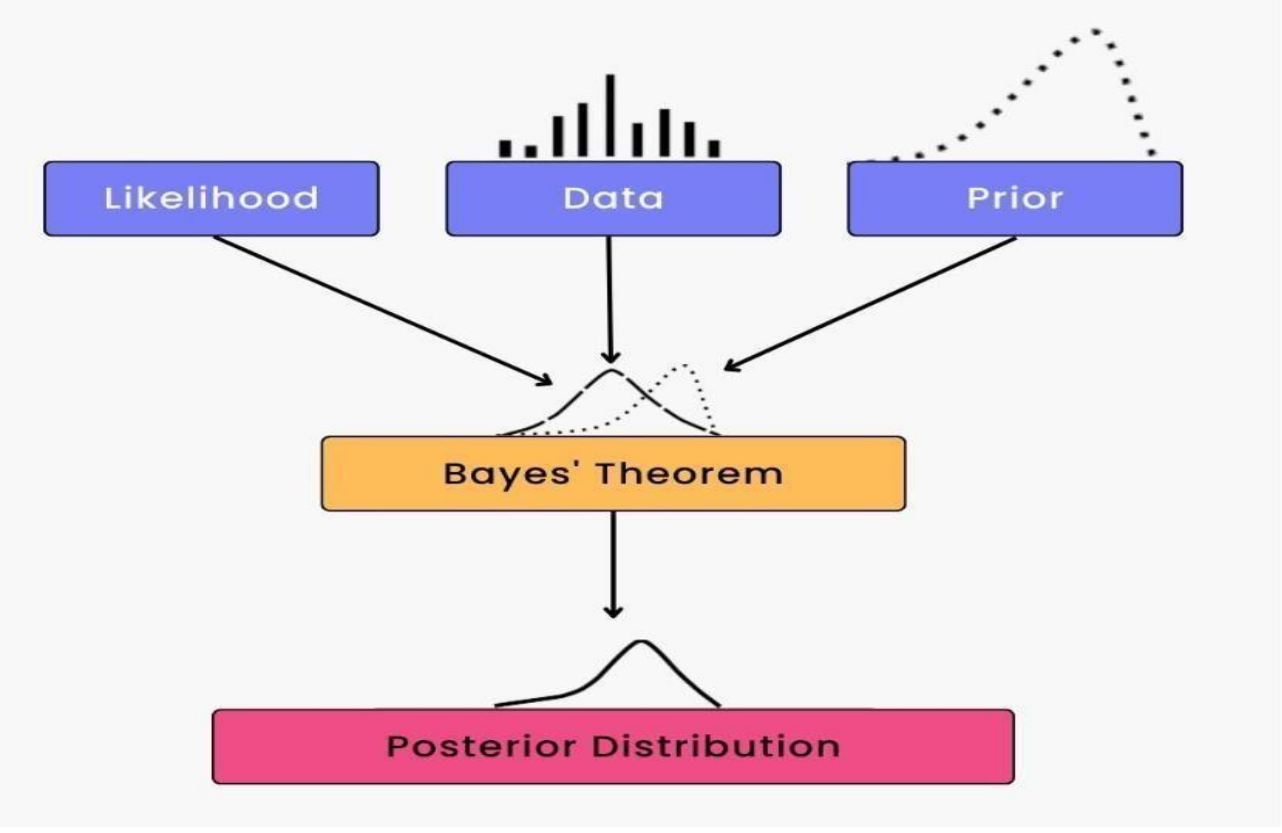

Figure 1:Workflow of Naive Bayes

Working:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability[4].

About Dataset :

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced bythe British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis. It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus"[3].

The data set consists of 50 samples from each of three species of Iris (**Iris setosa, Iris virginica and Iris versicolor**). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other[3].

2. ALGORITHM (Pseudocode)

```
1. for j = 1...z // loop for each vector
2. //increment count of vectors for value xj.t of the dependent tth attribute of vector xj;
   m[o1 + xj.t] ++;
3.for k=1.. p // loop for each attribute
4. // increment count of vectors with value xj.k of the independent kth attribute
    // and value xj.t of the dependent tth attribute of vector x;
         m[il + kx | Defk | +xj.k+xj.t] ++;
   end for;
end for;
```

3. Implementation:

```
from collections import defaultdict
from math import pi
from math import e
import requests
import random
import csv
import re
import pandas as pd
class GaussNB:
    def _init_(self):
        pass
    def load_csv(self, datas, header=False):
        ##load_csv, is designed to load CSV (Comma Separated Values) data and convert
     string values within it into floats
        dataset=[]
        lines = csv.reader(datas.splitlines())      #pratek record 2-D list made store hoil
        dataset = list(lines)
        if header:
            dataset = dataset[1:]
        for i in range(len(dataset)):
            dataset[i] = [float(x) if re.search('\d', x) else x for x in dataset[i]]          #convert
     string to float
        return dataset
    def split_data(self, data, weight):
        train_size = int(len(data) * weight)
        train_set = []
        for i in range(train_size):
            index = random.randrange(len(data))
            train_set.append(data[index])
            data.pop(index)
        return [train_set, data]          #train set menas training data and data means testing
     data
    def group_by_class(self, data, target):
        ## to make the records of features i.e.inputs
        target_map = defaultdict(list)
        for index in range(len(data)):
            features = data[index]
            if not features:
                continue
```

```
  x = features[target]
        target_map[x].append(features[:-1])          # target map made features store  zle
    #print(dict(target_map))                          #uncomment it to see details
    return dict(target_map)
def mean(self, numbers):
    #to find the mean of any parameter passed i.e. numbers
    result = sum(numbers) / float(len(numbers))
    return result
def stdev(self, numbers):
    ##calculate standard deviation for a list of numbers
    avg = self.mean(numbers)
    squared_diff_list = []
    for num in numbers:
        squared_diff = (num - avg) ** 2
        squared_diff_list.append(squared_diff)
    squared_diff_sum = sum(squared_diff_list)
    sample_n = float(len(numbers) - 1)
    var = squared_diff_sum / sample_n
    return var ** .5                #  return standard deviation

def summarize(self, test_set):
    for feature in zip(*test_set):
        yield {
            'stdev': self.stdev(feature),    #func call for S.D
            'mean': self.mean(feature)       #func call for mean
        }

def prior_prob(self, group, target, data):
    #returns: The probability of each target class/label
    #print(group[target])                            #can uncomment
    total = float(len(data))
    result = len(group[target]) / total
    print("probability of each target class is given below")
    print(result)
    return result
```

```
def train(self, train_list, target):

        group = self.group_by_class(train_list, target)
        self.summaries = {}
        for target, features in group.items():
            self.summaries[target] = {
                'prior_prob': self.prior_prob(group, target, train_list),
                'summary': [i for i in self.summarize(features)],
            }
        return self.summaries     #{summaries} navachi dict bavnavli jyat prior prob and
     summary store ahe

    def normal_pdf(self, x, mean, stdev):
        ## calculate prob. density. func. (PDF) of a Gaussian distribution for a given value
     x, mean "mean", and standard deviation "stdev"
        variance = stdev ** 2
        exp_squared_diff = (x - mean) ** 2
        exp_power = -exp_squared_diff / (2 * variance)
        exponent = e ** exp_power
        denominator = ((2 * pi) ** .5) * stdev
        normal_prob = exponent / denominator
        return normal_prob

    def marginal_pdf(self, joint_probabilities):
        #Joint Probability = prior * likelihood
        ## Marginal Probability is the sum of all joint probabilities for all classes.
        marginal_prob = sum(joint_probabilities.values())
        return marginal_prob

    def joint_probabilities(self, test_row):
        joint_probs = {}
        for target, features in self.summaries.items():
            total_features = len(features['summary'])
            likelihood = 1
            for index in range(total_features):
                feature = test_row[index]
                mean = features['summary'][index]['mean']
                stdev = features['summary'][index]['stdev']
                normal_prob = self.normal_pdf(feature, mean, stdev)likelihood *=
     normal_prob
```

```
 prior_prob = features['prior_prob']
            joint_probs[target] = prior_prob * likelihood
        return joint_probs

    def posterior_probabilities(self, test_row):
        posterior_probs = {}
        joint_probabilities = self.joint_probabilities(test_row)
        marginal_prob = self.marginal_pdf(joint_probabilities)
        for target, joint_prob in joint_probabilities.items():
            posterior_probs[target] = joint_prob / marginal_prob
        return posterior_probs

    def get_map(self, test_row):
        posterior_probs = self.posterior_probabilities(test_row)
        map_prob = max(posterior_probs, key=posterior_probs.get)
        return map_prob

    def predict(self, test_set):
        map_probs = []
        for row in test_set:
            map_prob = self.get_map(row)
            map_probs.append(map_prob)
        return map_probs

    def accuracy(self, test_set, predicted):
        correct = 0
        actual = [item[-1] for item in test_set]
        for x, y in zip(actual, predicted):
            if x == y:
                correct += 1
        return correct / float(len(test_set))

 nb = GaussNB()
 # Existing code to load data
 data = pd.read_csv("iris.csv")
 data_list = data.values.tolist()  # Convert DataFrame to a list of lists
train_list, test_list = nb.split_data(data_list, weight=0.67)  # Use data_list instead of data

 print("Using %s rows for training and %s rows for testing" % (len(train_list),
     len(test_list)))
 group = nb.group_by_class(data_list, -1) # designating the last column as the class
```

```
    column
print("Grouped into %s classes: %s" % (len(group.keys()), group.keys()))
nb.train(train_list, -1)
predicted = nb.predict(test_list)
accuracy = nb.accuracy(test_list, predicted)
print('Accuracy: %.3f' % accuracy)
print("enter samples you wanna test below")
i=int(input("enter any no="))
print("Enter comma-separated values for features (sepal length, sepal width, petal
    length, petal width):")
for i in range(i):
   l=[]
   print("enter number",i+1," query")
   #print("Enter comma-separated values for features (sepal length, sepal width, petal
    length, petal width):")
   user_input = input().strip().split(',') # Accept comma-separated input values
   shr=",".join(user_input)
   user_input = [float(i) for i in user_input]  # Convert input values to floats

   # Use the input as a single row for testing
   predicted_class = nb.get_map(user_input)

   print(" $ Query {} label is:- {} ---> {}".format(i+1,shr,predicted_class))

#6,3,5.1,1.8
```

4. Output :

```
Using 105 rows for training and 45 rows for testing
Grouped into 3 classes: dict_keys(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
probability of each target class is given below
0.3333333333333333
probability of each target class is given below
0.3238095238095238
probability of each target class is given below
0.34285714285714286
Accuracy: 0.956
enter samples you wanna test below
enter any no=6
Enter comma-separated values for features (sepal length, sepal width, petal length, petal width):
>>>Enter query number 1
6,3,5.1,1.8
 $  Query 1 label is:- [['6,3,5.1,1.8']] ------> Iris-virginica
>>>Enter query number 2
6.3,2.5,4.9,1.9
 $  Query 2 label is:- [['6.3,2.5,4.9,1.9']] ------> Iris-virginica
```

```
6.3,2.5,4.9,1.9
 $  Query 2 label is:- [['6.3,2.5,4.9,1.9']] ------> Iris-virginica
>>>Enter query number 3
5.8,2.7,4.1,1.5
 $  Query 3 label is:- [['5.8,2.7,4.1,1.5']] ------> Iris-versicolor
>>>Enter query number 4
4.7,3.2,1.3,0.4
 $  Query 4 label is:- [['4.7,3.2,1.3,0.4']] ------> Iris-setosa
>>>Enter query number 5
6.1,2.8,4,1.3
 $  Query 5 label is:- [['6.1,2.8,4,1.3']] ------> Iris-versicolor
>>>Enter query number 6
4.7,3.2,1.3,0.2
 $  Query 6 label is:- [['4.7,3.2,1.3,0.2']] ------> Iris-setosa
```

5. Advantages & Disadvantages :

Advantages:

- It doesn’t require larger amounts of training data.
- It is straightforward to implement.
- Convergence is quicker than other models, which are discriminative.
- It is highly scalable with several data points and predictors.
- It can handle both continuous and categorical data.
- It is not sensitive to irrelevant data and doesn’t follow the assumptions it holds.
- It is used in real-time predictions[3].

Disadvantages:

- The Naive Bayes Algorithm has trouble with the ‘zero-frequency problem’. It happens when you assign zero probability for categorical variables in the training dataset that is not available. When you use a smooth method for overcoming this problem, you can make it work the best.
- It will assume that all the attributes are independent, which rarely happens in real life. It will limit the application of this algorithm in real-world situations.
- It will estimate things wrong sometimes, so you shouldn’t take its probability outputs seriously[3].

6. Applications of Naïve Bayes Algorithm:

The Naive Bayes Algorithm is used for various real-world problems like those below:

- **Text classification:** The Naive Bayes Algorithm is used as a probabilistic learning technique for text classification. It is one of the best-known algorithms used for document classification of one or many classes.
- **Sentiment analysis:** The Naive Bayes Algorithm is used to analyze sentiments or feelings, whether positive, neutral, or negative.
- **Recommendation system:** The Naive Bayes Algorithm is a collection of collaborative filtering issued for building hybrid recommendation systems that assist you in predicting whether a user will receive any resource.
- **Spam filtering:** It is also similar to the text classification process. It is popular for helping you determine if the mail you receive is spam.
- Medical diagnosis: This algorithm is used in medical diagnosis and helps you to predict the patient's risk level for certain diseases.
- **Weather prediction:** You can use this algorithm to predict whether the weather will be good.
- **Face recognition:** This helps to identify faces[5].

.

7. Conclusion:

During the implementation of Naïve Bayes classifier on iris entire dataset , we obtained the accuracy of 96%. And error rate of 4%. We have performed the attribute selection using information gain method. After performing attribute selection on 3 attributes having highest information gain, we obtained the accuracy of 97% and error rate of 3%. So, attribute selection is providing more accuracy than considering entire attributes. from this we can say that Naïve Bayes algorithm has the highest rate of success when compared to other algorithms because of its speed and efficiency.

8. References:

[1] Naive Bayes classifier - Wikipedia

[2] Naive Bayes Algorithm in ML: Simplifying Classification Problems (turing.com)

[3] Naive Bayes Classifier in Machine Learning - Javatpoint

[4] Iris Dataset (kaggle.com)

[5] Data Mining - Naive Bayes (NB) (datacadamia.com)