NUST COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING

MATH361 Probability and Statistics

PROJECT REPORT

WEATHER PREDICTION USING NAÏVE BAYES

Submitted To	A/P Kamran Aziz Bhatti
Submitted By	• Muhammad Ali Bin Mushtaq (335701) • Haiqa Ansar (348488) • Aiza Naeem Rana (331722)
Degree	EE-42-B
Date	19th January, 2023

Feedback

ABSTRACT

Naïve Bayes algorithm is one of the simplest prediction algorithms in Machine learning which works on the probabilistic model created by Bayes under a naïve assumption that all the features are independent of each other. In this project we will create our own Naïve Bayes algorithm to predict the weather of a dataset, use the sklearn library to predict the weather using already made functions and compare our results.

THEORETICAL BACKGROUND

Naïve Bayes:

Naïve Bayes is a probabilistic model which works using conditional probability to find the probability of a data using n number of features represented by vector **x.** Where:

$$P(W_k|x) = \frac{P(W_k)P(x|W_k)}{P(x)}\ ; x = x1, x2, x3 \ldots . xn$$

Now in English we can simply write as:

$$posterior = priori * \frac{likelihood}{evidence}$$

And here comes the Naïve part. If we implement the above equation as it is, we will get an extremely complex equation which will be almost impossible to solve. So, we make a naïve assumption that all the features are independent of each other.

> There are many Naïve Bayes models but in this project we will be using the Gaussian Naïve Bayes.

Gaussian Naïve Bayes:

In Gaussian Naïve Bayes, when dealing with data we typically assume that our data is disturbed normally (gaussian distribution). For this we take the mean and variance of our data and find the conditional probability using the Gaussian equation which is:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Where **x** are our features, sigma square is variance and u is mean. Using this equation we can calculate the conditional probability of each feature and then put it into our Naïve Bayes equation to calculate the posterior probability.

Steps of implementing a ML algorithm

Following are the steps for implementing any Machine Learning algorithm to our dataset using python. We will be using the following pattern in our code as well.

1. **Importing all the libraries:** The first step of implementing any ML algorithm is importing all the necessary libraries into the first block of code.
2. **Import dataset:** The second step is to import our dataset using pandas. We can either connect google collab to google drive and import the csv file, upload the file, or import the data using **sklearn.dataset**. In our code, we will be uploading our file into the google collab.
3. **Split the data:** In this step we will be splitting the data both column and row wise. We will store our features (Independent) in x variable and our dependent column in y variable. After that we will split our data row wise, usually in a ratio of 80/20. This ratio can change but since it is normal practice, we will be using this ratio. The 80% of our data will be train data and the remaining 20% will be used to test our model.
4. **Create Model:** In this step we will create our model using any of the libraries or functions. In our project we will be making a model using gaussian naïve bayes.
5. **Model.fit():** This step will train our model. We will input our training data to model.fit command and train our model.
6. **Predicting test data:** In this step we will input our dependent variable (x_test) to our Model using predict command and get a predicted y as output.
7. **Finding accuracy:** In the final step we will compare our predicted y to the actual y and see how much accurate our predictions are.

Now we will implement all these steps to our Gaussian Naïve Bayes algorithm.

$$p(C_k \mid x_1, \dots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

Where, evidence is $Z = p(\mathbf{x}) = \sum_k p(C_k)\, p(\mathbf{x} \mid C_k)$

Complete Code

```
#Importing all the necessary libraries
import pandas as pd #Pandas library to access and change our csv file
import numpy as np #numpy to perform necessary calculations
import math #math to use pi
from sklearn.model_selection import train_test_split #sklearn train test s
plit to split our data in test and train data
from sklearn.metrics import accuracy_score#to find the accuracy of our mod
el.

#Importing sklearn's naive bayes to check our function by comparing
from sklearn.naive_bayes import GaussianNB

#importing our csv file
df = pd.read_csv('/content/Weather Data.csv')

#Creating our own Gaussian Naive Bayes class. This class is designed to wo
rk exactly like sklearn library. We will later compare the results as well
.
class GaussianNaiveBayes():
  def __init__(self): #The first function which is automatically run when
a new object is created. This will create necessary variables for the clas
s
    print("Gaussian Naive Baye's Object is created")
    self.mean = []
    self.std = []
    self.y = []
    self.predicted = []

  def fit(self, x_train, y_train): #fit command to train our model using t
rain data. This will find mean and standard deviation of our data which wi
ll later be used in gaussian.
    self.find_mean_and_std(x_train)
    self.y = y_train #saving the y train data to find priori probability l
ater.


  def find_mean_and_std(self,data): #this funcion calculates the mean and
std of every column and saves it into our class's vairables.
    mean = []
    std = []
    for col in data.columns:
```

```
      mean.append(data[col].mean())
      std.append(data[col].std())
    self.mean = mean
    self.std = std

  def predict(self, x_test): #This function will predict the y using given
 test data.
    weather_p = []
    self.predicted = []
    i = 0
    while i < int(len(x_test)):
      for weather in self.y.unique():
        inputvalues = x_test.iloc[i] # we will be sending every input valu
e for each weather cell and see which weather has the highest probability.

        weather_p.append(self.naive_bayes(weather, inputvalues)) #Find pos
terior probabilities for each weather.
      i = i + 1
      prediction = self.y.loc[weather_p.index(max(weather_p))]
      self.predicted.append(prediction) # The weather with highest probabi
lity is stored in predicted array. This array is then returned.
      continue
    return self.predicted

  def naive_bayes(self, y, input):
    #Finding P(y) priori probability
    priori = int(self.y.value_counts()[y])/int(len(self.y))
    #Finding P(x|y)
    likelihood = []
    #Find likelihood probability of each column of features and storing th
em in likelihood list
    for i in range(len(self.mean)):
      likelihood.append(self.find_likelihood_probability(self.mean[i], sel
f.std[i], input[i]))
    #Finding the posterior probability by implementing the equation as der
ived in theoretical background.
    likelihood_product = math.prod(likelihood)
    numerator = priori*likelihood_product
    denominator = 0
    for y1 in self.y:
      priori1 = int(self.y.value_counts()[y1])/int(len(self.y))
      denominator = denominator + priori1*likelihood_product
    return (numerator/denominator)
```

```
  def find_likelihood_probability(self, m, std, v): #This function will fi
nd the likelihood probabilities using gaussian equation.
    expo = np.exp(-((float(v)-float(m))**2)/(2*math.pi*float(std)**2))
    return (expo/(np.sqrt(2*np.pi*float(std)*float(std))))

y = df['Weather']#Separating the dependent and independent variables.
x = df.drop(['Weather', 'Date/Time'], axis=1) #Removing unnecessary column
s
#In this block we are taking a sample from our data to check the model fas
ter.
x = x.head(300)
y = y.head(300)
#using the train test split command to split our data in xtrain, xtest, yt
rain and ytest in a ratio of 80/20
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20, ran
dom_state = 0)
#In this block we will create a model, train it using training data and ge
t a predicted y by inputing test data.
model = GaussianNaiveBayes()
model.fit(x_train, y_train)
predicted_y = model.predict(x_test)
#Finding accurary of our prediction
accuracy = accuracy_score(y_test, predicted_y)
accuracy = accuracy*100 #converting to percentage
print(f"{accuracy}%")
#Implementing naive bayes from sklearn library
model2 = GaussianNB()
model2.fit(x_train, y_train)
predictedy2 = model2.predict(x_test)
#Finding accuracy of sklearn's prediction
accuracy2 = accuracy_score(y_test, predictedy2)
accuracy2 = accuracy2*100
print(f"The accuracy of our model is {accuracy} and the accuracy of sklear
n's model is {accuracy2}")
#We can see that the accuracy of sklearn's model is higher than ours and i
t takes less time than our class.
```

Link to code:
https://colab.research.google.com/drive/1ohqM2HGCLni_cSjFTbOW95gpOz9cp0xs?usp=sharing

Conclusion

We designed our naïve bayes algorithm and created a class which worked like the sklearn gaussian naïve bayes. Although our accuracy was way lower but we implemented it and got acceptable results.