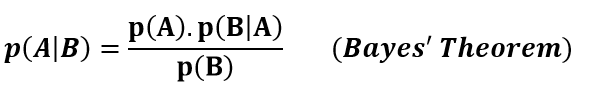
|  |  |
| --- | --- |
| Algorithm: Naive Bayes | |
| USN : 1MS17CS143 | NAME : Sathvik K P |
| USN : 1MS17CS148 | NAME : Sathvik B |

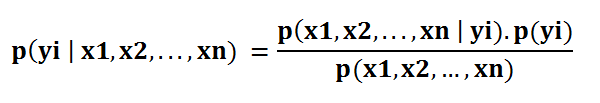
**Description of the Algorithm:**

Naive Bayes is a supervised learning algorithm used for classification tasks. Hence, it is also called Naive Bayes Classifier.

As other supervised learning algorithms, naive bayes uses features to make a prediction on a target variable. The key difference is that naive bayes assumes that features are independent of each other and there is no correlation between features. However, this is not the case in real life. This naive assumption of features being uncorrelated is the reason why this algorithm is called “naive”.

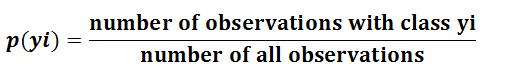


Naive bayes is a supervised learning algorithm for classification so the task is to find the class of observation (data point) given the values of features. Naive bayes classifier calculates the probability of a class given a set of feature values (i.e. p(yi | x1, x2 , … , xn)). Input this into Bayes’ theorem:



p(x1, x2 , … , xn | yi) means the probability of a specific combination of features given a class label. To be able to calculate this, we need extremely large datasets to have an estimate on the probability distribution for all different combinations of feature values. To overcome this issue, naive bayes algorithm assumes that all features are independent of each other. Furthermore, denominator (p(x1,x2, … , xn)) can be removed to simplify the equation because it only normalizes the value of conditional probability of a class given an observation ( p(yi | x1,x2, … , xn)).

The probability of a class ( p(yi) ) is very simple to calculate:



Under the assumption of features being independent, p(x1, x2 , … , xn | yi) can be written as:



The conditional probability for a single feature given the class label (i.e. p(x1 | yi) ) can be more easily estimated from the data. The algorithm needs to store probability distributions of features for each class independently. For example, if there are 5 classes and 10 features, 50 different probability distributions need to be stored. The type of distributions depend on the characteristics of features:

For binary features (Y/N, True/False, 0/1): Bernoulli distribution

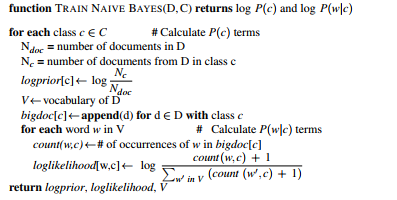
For discrete features (i.e. word counts): Multinomial distribution

For continuous features: Gaussian (Normal) distribution

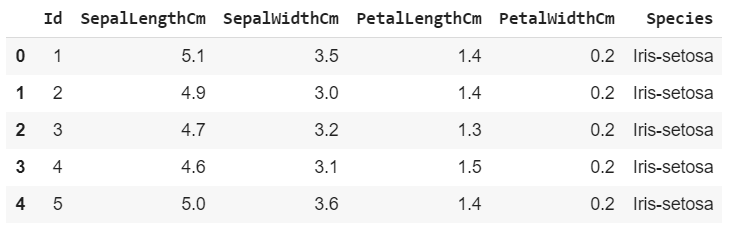
It is common to name the naive bayes with the distribution of features (i.e. Gaussian naive bayes classifier). For mixed type datasets, a different type of distribution may be required for different features.

Adding all these up, it became an easy task for naive bayes algorithm to calculate the probability to observe a class given values of features (p(yi | x1, x2 , … , xn) )

**Algorithm Pseudocode:**



**Data set Used: (Attach Screen shot of the few rows)**

****

**Challenges faced during the implementation of the program:**

1. The assumption that all features are independent is not usually the case in real life
2. Naive Bayes algorithm is less accurate than complicated algorithms.
3. Outliers can sometimes cause the algorithm to fail.

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

df = pd.read\_csv("iris.csv")

df.drop(['Id'], axis=1,inplace=True)

sns.FacetGrid(df, hue="Species", size=5) \

   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \

   .add\_legend()

ax = sns.boxplot(x="Species", y="PetalLengthCm", data=df)

ax = sns.stripplot(x="Species", y="PetalLengthCm", data=df, jitter=True, edgecolor="gray")

sns.FacetGrid(df, hue="Species", size=6) \

   .map(sns.kdeplot, "PetalLengthCm") \

   .add\_legend()

df['Species'] = df['Species'].apply(lambda x: 0 if x=='Iris-setosa' else (1 if x=='Iris-versicolor' else 2))

X=df.drop(['Species'],axis=1)

y=df['Species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

clf = MultinomialNB()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

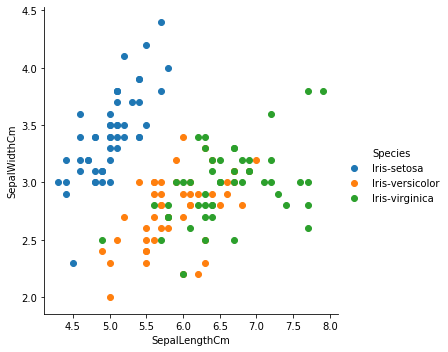
sns.heatmap(cm,annot=True)

print(accuracy\_score(y\_test,y\_pred))

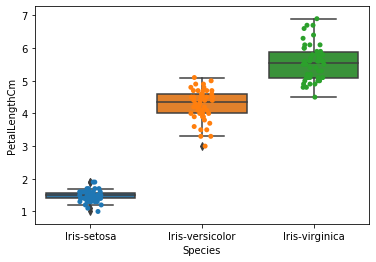
**Output: (Screen shots)**

Exploratory Data Analysis

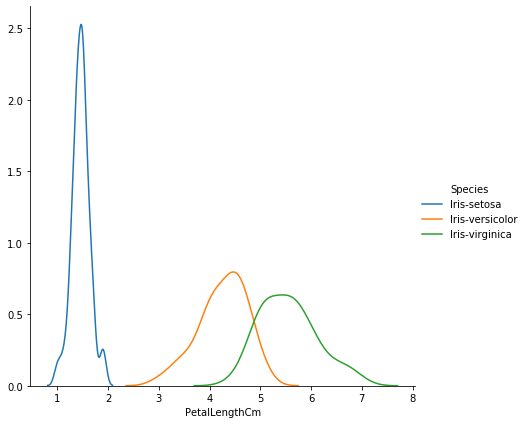
Scatter plot of the different species with SepalLength as X axis and SepalWidth as Y axis



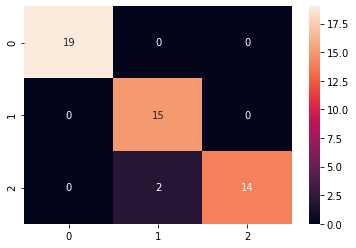
Boxplot to determine if outliers are present



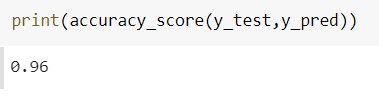
Graph to plot petal length per species



Confusion matrix



Results



**References:**

1. <https://www.edureka.co/blog/naive-bayes-tutorial/>
2. <https://medium.com/@johnm.kovachi/implementing-a-multinomial-naive-bayes-classifier-from-scratch-with-python-e70de6a3b92e>
3. <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>
4. <https://en.wikipedia.org/wiki/Naive_Bayes_classifier>