Name: Ojas Patil

UID: 2019130048

TE COMPS

Batch C

**Experiment 4**

**Aim**: To train and test a machine learning model using naïve bayes algorithm.

**Theory**:

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome.

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred.

**Code**:

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# %%
df_temp = pd.read_csv('Iris.csv')
df_temp = df_temp.drop(columns=['Id'], axis= 1)
df_temp.head()

# %%
df_temp.isnull().sum()
df_temp = df_temp.drop_duplicates()
df_temp.duplicated().sum()

# %%
print(df_temp.info())
```

```
df_temp.describe()
type(df_temp)

# %%
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df_temp, df_temp.Species,
random_state=4, test_size=0.2)


df = pd.DataFrame(x_train)
print(x_test)

# %%
print(sorted(df.Species.unique()))
print('Iris-setosa:', df.Species[df.Species == 'Iris-setosa'].count(), "/",
117, "=", df.Species[df.Species == 'Iris-setosa'].count()/117)
print('Iris-versicolor:', df.Species[df.Species == 'Iris-versicolor'].count(),
"/", 117, "=", df.Species[df.Species == 'Iris-versicolor'].count()/117)
print('Iris-virginica:', df.Species[df.Species == 'Iris-virginica'].count(),
"/", 117, "=", df.Species[df.Species == 'Iris-virginica'].count()/117)

p_setosa = df.Species[df.Species == 'Iris-setosa'].count()/117
p_versicolor = df.Species[df.Species == 'Iris-versicolor'].count()/117
p_virginica = df.Species[df.Species == 'Iris-virginica'].count()/117

count_setosa = df.Species[df.Species == 'Iris-setosa'].count()
count_versicolor = df.Species[df.Species == 'Iris-versicolor'].count()
count_virginica = df.Species[df.Species == 'Iris-virginica'].count()

# %%
sns.scatterplot(df.Species, df.PetalLengthCm)

# %%
sns.scatterplot(df.Species, df.PetalWidthCm)

# %%
sns.scatterplot(df.Species, df.SepalLengthCm)

# %%
sns.scatterplot(df.Species, df.SepalWidthCm)

# %%
dataDict = {
    'PetalLengthCm': {},
    'PetalWidthCm': {},
    'SepalLengthCm': {},
    'SepalWidthCm': {}
    }
```

```
# %%
for i in range(1, 8):
    x = len(df[(df.PetalLengthCm.isin(range(i, i+1))) & (df.Species == 'Iris-
setosa')])
    dataDict['PetalLengthCm'][f'{i}-{i+1}'] = list()
    dataDict['PetalLengthCm'][f'{i}-{i+1}'].append(x)
    x = len(df[(df.PetalLengthCm.isin(range(i, i+1))) & (df.Species == 'Iris-
versicolor')])
    dataDict['PetalLengthCm'][f'{i}-{i+1}'].append(x)
    x = len(df[(df.PetalLengthCm.isin(range(i, i+1))) & (df.Species == 'Iris-
virginica')])
    dataDict['PetalLengthCm'][f'{i}-{i+1}'].append(x)

print(dataDict)

# %%
for i in np.arange(0, 2.5, 0.5):
    x = len(df[(df.PetalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-setosa')])
    dataDict['PetalWidthCm'][f'{i}-{i+0.5}'] = list()
    dataDict['PetalWidthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.PetalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-versicolor')])
    dataDict['PetalWidthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.PetalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-virginica')])
    dataDict['PetalWidthCm'][f'{i}-{i+0.5}'].append(x)
for i in np.arange(4, 8, 0.5):
    x = len(df[(df.SepalLengthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-setosa')])
    dataDict['SepalLengthCm'][f'{i}-{i+0.5}'] = list()
    dataDict['SepalLengthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.SepalLengthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-versicolor')])
    dataDict['SepalLengthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.SepalLengthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-virginica')])
    dataDict['SepalLengthCm'][f'{i}-{i+0.5}'].append(x)
for i in np.arange(0, 2.5, 0.5):
    x = len(df[(df.SepalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-setosa')])
    dataDict['SepalWidthCm'][f'{i}-{i+0.5}'] = list()
    dataDict['SepalWidthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.SepalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-versicolor')])
    dataDict['SepalWidthCm'][f'{i}-{i+0.5}'].append(x)
    x = len(df[(df.SepalWidthCm.isin(np.arange(i, i+0.5, 0.1))) & (df.Species
== 'Iris-virginica')])
```

```
    dataDict['SepalWidthCm'][f'{i}-{i+0.5}'].append(x)
print(dataDict)

# %%
petallength = float(input("Enter petal length in cm: "))
sepallength = float(input("Enter sepal length in cm: "))
petalwidth = float(input("Enter petal width in cm: "))
sepalwidth = float(input("Enter sepal width in cm: "))

print(f"""
petalLength: {petallength},
sepallength: {sepallength},
petalwidth: {petalwidth},
sepalwidth: {sepalwidth}
""")


if petallength <= 1:
    petallength = dataDict['PetalLengthCm']['1-2']
elif petallength > 8:
    petallength = dataDict['PetalLengthCm']['7-8']
else:
    for i in range(1, 8):
        if petallength > i and petallength <= i+1:
            petallength = dataDict['PetalLengthCm'][f'{i}-{i+1}']
            break

if sepallength <= 4.0:
    sepallength = dataDict['SepalLengthCm']['4.0-4.5']
elif sepallength > 8.0:
    sepallength = dataDict['SepalLengthCm']['7.5-8.0']
else:
    for i in np.arange(4, 8, 0.5):
        if sepallength > i and sepallength <= i+1:
            sepallength = dataDict['SepalLengthCm'][f'{i}-{i+0.5}']
            break

if petalwidth <= 0:
    petalwidth = dataDict['PetalWidthCm']['0.0-0.5']
elif petalwidth > 2.5:
    petalwidth = dataDict['PetalWidthCm']['2.0-2.5']
else:
    for i in np.arange(0, 2.5, 0.5):
        if petalwidth > i and petalwidth <= i+0.5:
            petalwidth = dataDict['PetalWidthCm'][f'{i}-{i+0.5}']
            break

if sepalwidth <= 0:
    sepalwidth = dataDict['SepalWidthCm']['0.0-0.5']
```

```
elif sepalwidth > 2.5:
    sepalwidth = dataDict['SepalWidthCm']['2.0-2.5']
else:
    for i in np.arange(0, 2.5, 0.5):
        if sepalwidth > i and sepalwidth <= i+0.5:
            sepalwidth = dataDict['SepalWidthCm'][f'{i}-{i+0.5}']
            print(i, i+0.5, sepalwidth)
            break

print(petallength)
print(petalwidth)
print(sepallength)
print(sepalwidth)

# %%
p_t_by_setosa = (petallength[0] * petalwidth[0] * sepallength[0] *
sepalwidth[0]) / count_setosa ** 4
p_t_by_versicolor = (petallength[1] * petalwidth[1] * sepallength[1] *
sepalwidth[1]) / count_versicolor ** 4
p_t_by_virginica = (petallength[2] * petalwidth[2] * sepallength[2] *
sepalwidth[2]) / count_virginica ** 4

likelihood_of_being_setosa = p_setosa * p_t_by_setosa
likelihood_of_being_versicolor = p_versicolor * p_t_by_versicolor
likelihood_of_being_virginica = p_virginica * p_t_by_virginica

p_t = likelihood_of_being_setosa + likelihood_of_being_versicolor +
likelihood_of_being_virginica

test = max(likelihood_of_being_virginica, likelihood_of_being_setosa,
likelihood_of_being_versicolor)

if test == likelihood_of_being_setosa:
    print("Iris-setosa")
elif test == likelihood_of_being_versicolor:
    print("Iris-versicolor")
elif test == likelihood_of_being_virginica:
    print("Iris-virginica")

# %%
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df.iloc[:, :4],
df.Species, random_state=4, test_size=0.2)

# %%
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
clf = GaussianNB()
```

```
clf.fit(x_train, y_train)

result = clf.predict(x_test)

accuracy_score(y_test, result) * 100
```

**Output**:

```
petalLength: 1.7,
sepallength: 5.0,
petalwidth: 0.4,
sepalwidth: 3.9
```

```
Iris-setosa
```

**Conclusion**:

From above experiment of naïve bayes, I learned about the basic bayes theorem. **Bayes' theorem** describes the probability of occurrence of an event related to any condition. In naïve bayes algorithm, we find out the probability of the category of the output and select the category with highest probability.

Naïve bayes algorithm works on two assumptions that every data point in the dataset contributes independently and equally to the dataset.

Using naïve bayes algorithm, we can predict the category with fair accuracy of probably greater than 90-95%.

Github: https://github.com/PatilOjas/AIML-Lab