Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Bhuvi Ghosh
60009210191

Experiment 4

(Naïve Bayes Classifier)

Aim: Implement Naïve Bayes Classifier on a given Dataset.

Theory:

Naïve Bayes Classifier 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 which helps in building the fast machine learning models 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**.

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 color, 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.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.
  The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Breastcancer.csv
Dataset 2: Social_Network_Ads.csv

1. Perform required preprocessing on Dataset 1 and fit a Naïve Bayes classifier built from scratch. Evaluate the f1 score of classifiers built for categorical and continuous features.
2. Using sklearn library fit a Naïve Bayes classifier on Dataset 2.

Q1)

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

```
[ ] sns.set_style('darkgrid')
```

```
[ ] df=pd.read_csv('/content/breastcancer(1).csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
```

```
[ ] df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

```
df.isnull().sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
```

```
[ ] df.drop(columns=['Unnamed: 32','id'],inplace=True,axis=1)
```

```
[ ] from sklearn import preprocessing

    # label_encoder object knows how to understand word labels.
    label_encoder = preprocessing.LabelEncoder()

    # Encode labels in column 'species'.
    df['diagnosis']= label_encoder.fit_transform(df['diagnosis'])
```

```
plt.figure(figsize=(25,15))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

```
[ ] df1=df.drop('diagnosis',axis=1)
```

```
[ ] def removefeatures(x,threshold):
      corr_matrix = x.corr()
      iters = range(len(corr_matrix.columns) - 1)
      drop_cols = []
      for i in iters:
        for j in range(i+1):
          item = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
          col = item.columns
          row = item.index
          val = abs(item.values)
          if(val >= threshold):
            drop_cols.append(col.values[0])
      drops = set(drop_cols)
      x = x.drop(columns=drops,axis=1,inplace=True)
      print('Removed Columns {}'.format(drops))
      return(x)
```

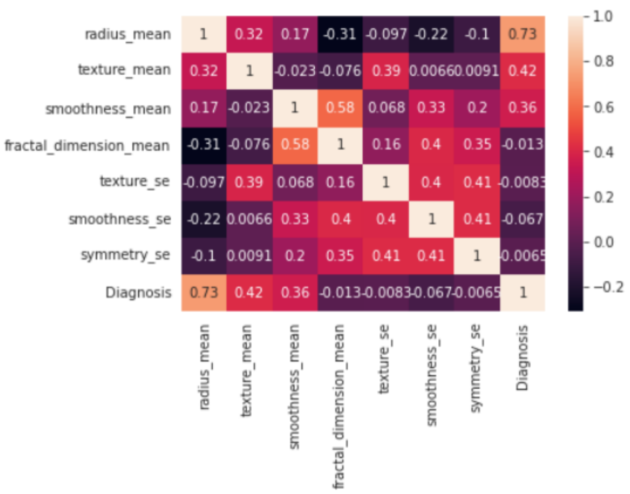
```
[ ] removefeatures(df1,0.6)

Removed Columns {'symmetry_worst', 'compactness_mean', 'perimeter_se', 'compactness_se', 'concave points_se', 'fractal_dimension_worst', 'smoothness_worst',
```

```
[ ] df1['Diagnosis']=df['diagnosis']
```

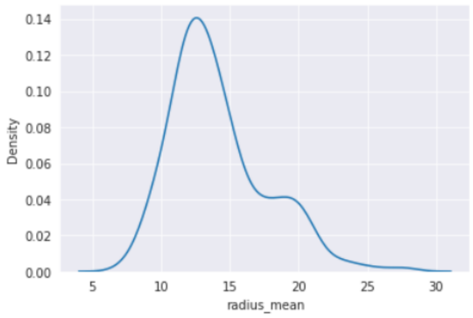
```
[ ] df1.columns

Index(['radius_mean', 'texture_mean', 'smoothness_mean',
       'fractal_dimension_mean', 'texture_se', 'smoothness_se', 'symmetry_se',
       'Diagnosis'],
      dtype='object')
```

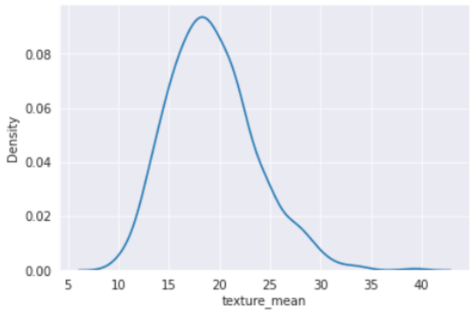
```
sns.heatmap(df1.corr(),annot=True)
```

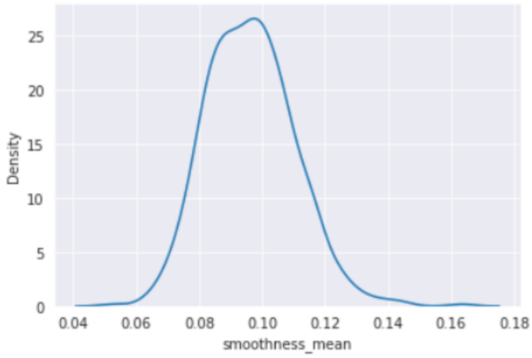
<AxesSubplot:>

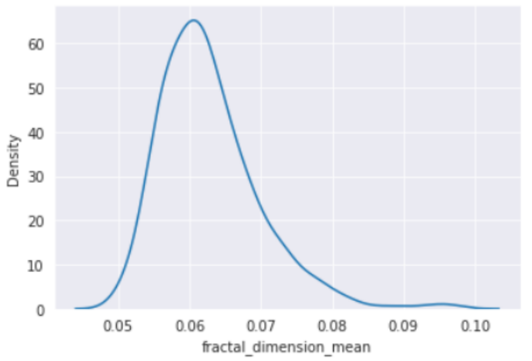


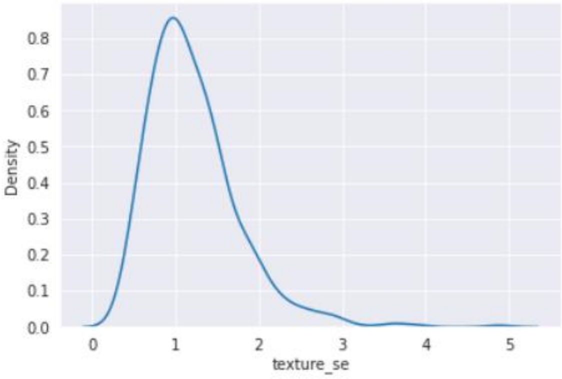
```python
for i in df1:
  sns.kdeplot(df1[i])
  plt.show()
```

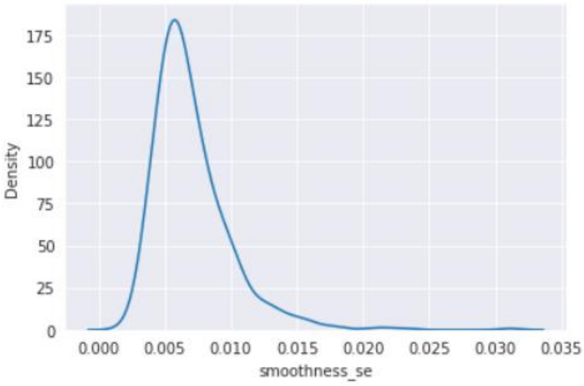


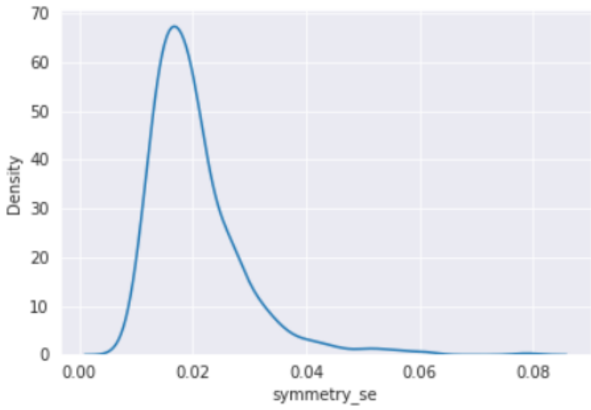













```
[ ] df2=df1.drop(columns=['fractal_dimension_mean', 'texture_se', 'smoothness_se', 'symmetry_se'],axis=1)
```

```
[ ] df2.head()
```

	radius_mean	texture_mean	smoothness_mean	Diagnosis
0	17.99	10.38	0.11840	1
1	20.57	17.77	0.08474	1
2	19.69	21.25	0.10960	1
3	11.42	20.38	0.14250	1
4	20.29	14.34	0.10030	1

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   radius_mean      569 non-null    float64
 1   texture_mean     569 non-null    float64
 2   smoothness_mean  569 non-null    float64
 3   Diagnosis        569 non-null    int64  
dtypes: float64(3), int64(1)
memory usage: 17.9 KB
```

```
[ ] # Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
 separated = dict()
 for i in range(len(dataset)):
   vector = dataset[i]
   class_value = vector[-1]
   if (class_value not in separated):
     separated[class_value] = list()
     separated[class_value].append(vector)
 return separated
```

```
[ ] def mean(numbers):
 return sum(numbers)/float(len(numbers))
```

```
[ ] def stdev(numbers):
 avg = mean(numbers)
 variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
 return sqrt(variance)
```

```
[ ] def summarize_dataset(dataset):
 summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
 del(summaries[-1])
 return summaries
```

```
from math import sqrt
from math import pi
from math import exp
# Calculate the mean of a list of numbers
def mean(numbers):
 return sum(numbers)/float(len(numbers))
```

```
[ ] # Calculate the standard deviation of a list of numbers
def stdev(numbers):
 avg = mean(numbers)
 variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
 return sqrt(variance)
```

```
[ ] # Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
 summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
 del(summaries[-1])
 return summaries
```

```
[ ] def summarize_by_class(dataset):
 separated = separate_by_class(dataset)
 summaries = dict()
 for class_value, rows in separated.items():
   summaries[class_value] = summarize_dataset(rows)
   return summaries
```

```
def calculate_probability(x, mean, stdev):
 exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
 return (1 / (sqrt(2 * pi) * stdev)) * exponent
```

```
[ ] def calculate_class_probabilities(summaries, row):
   total_rows = sum([summaries[label][0][2] for label in summaries])
   probabilities = dict()
   for class_value, class_summaries in summaries.items():
     probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
     for i in range(len(class_summaries)):
       mean, stdev, _ = class_summaries[i]
```

```
[ ]     probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities
```

```
[ ] dataset = [[3.393533211,2.331273381,0],
     [3.110073483,1.781539638,0],
     [1.343808831,3.368360954,0],
     [3.582294042,4.67917911,0],
     [2.280362439,2.866990263,0],
     [7.423436942,4.696522875,1],
     [5.745051997,3.533989803,1],
     [9.172168622,2.511101045,1],
     [7.792783481,3.424088941,1],
     [7.939820817,0.791637231,1]]
    summaries = summarize_by_class(dataset)
    probabilities = calculate_class_probabilities(summaries, dataset[0])
    print(probabilities)
```

```
{0: 0.05032427673372076, 1: 0.00011557718379945765}
```

```
[ ] type(dataset)
```

```
list
```

```
[ ] df3=df2.to_numpy()
```

```
summaries = summarize_by_class(df3)
probabilities = calculate_class_probabilities(summaries, df3[0])
print(probabilities)
```

```
{1.0: 0.0008723013785674848, 0.0: 5.0267209472116714e-05}
```

```
[ ] def predict(summaries, row):
     probabilities = calculate_class_probabilities(summaries, row)
     best_label, best_prob = None, -1
     for class_value, probability in probabilities.items():
       if best_label is None or probability > best_prob:
         best_prob = probability
         best_label = class_value
     return best_label
```

```
[ ] model = summarize_by_class(df3)
    row=[17,10,0.1]
    predict(model,row)
```

```
1.0
```

Q2)

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

```
[ ] sns.set_style('darkgrid')
```

```
[ ] df=pd.read_csv('/content/Social_Network_Ads.csv')
```

```
[ ] df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
[ ] df.duplicated().sum()
```

```
0
```

```
[ ] df.drop_duplicates(inplace=True)
```

```
[399] df.drop(columns='User ID',axis=1,inplace=True)
```

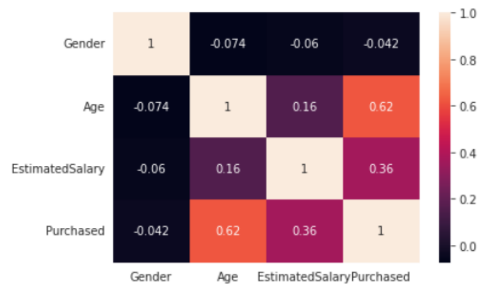
```
[400] from sklearn import preprocessing
      labelencoder=preprocessing.LabelEncoder()
      df['Gender']=labelencoder.fit_transform(df['Gender'])
```

```
[401] df.isnull().sum()
```

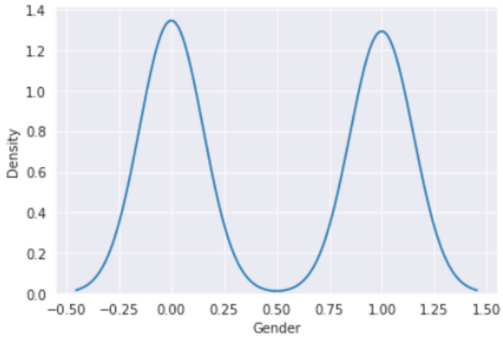
```
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

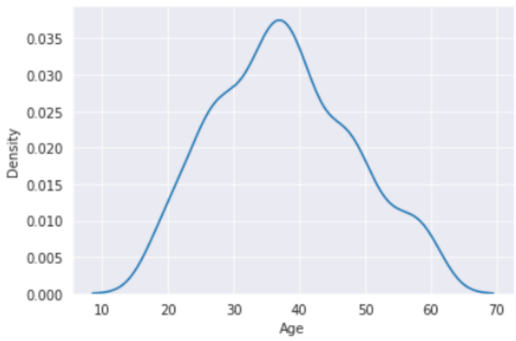
```
sns.heatmap(df.corr(),annot=True)
```

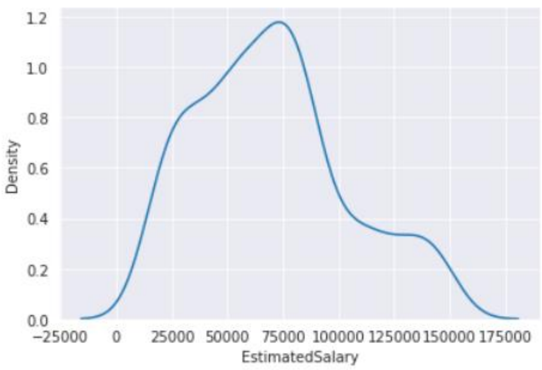
```
<AxesSubplot:>
```

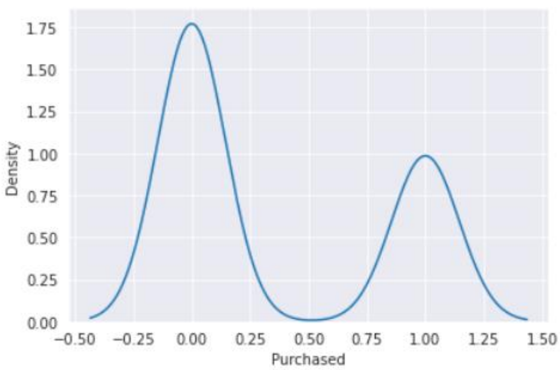


```
[403] x=df.iloc[:,0:3].values
      y=df.iloc[:,3].values
```









```
[405] from sklearn.preprocessing import StandardScaler
      sc=StandardScaler()
      x=sc.fit_transform(x)
```

```
[406] from sklearn.model_selection import train_test_split
      x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=50)
```

```
[407] from sklearn.naive_bayes import GaussianNB
      model=GaussianNB()
      model.fit(x_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
[408] y_pred=model.predict(x_test)
```

```
[409] from sklearn.metrics import accuracy_score
      accuracy_score(y_test,y_pred)
```

```
0.8863636363636364
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
array([[82,  4],
       [11, 35]])
```

Classification report:

```
[411] from sklearn.metrics import classification_report
      classification_report(y_test,y_pred)
```

```
'              precision    recall  f1-score   support\n\n           0       0.88      0.95      0.92        86\n           1       0.90      0.76      0.82        46\n\n    accuracy
0.89       132\n   macro avg       0.89      0.86      0.87       132\nweighted avg       0.89      0.89      0.88       132\n'
```

Using cross validation for finding accuracy:

```
[412] from sklearn.model_selection import cross_val_score
      model=GaussianNB()
      cvscores=cross_val_score(model,x,y,cv=25)
      cvscores.mean()
```

```
0.8875
```

```
[413] from sklearn.model_selection import RepeatedStratifiedKFold
      cv_method = RepeatedStratifiedKFold(n_splits=5, n_repeats=3,random_state=999)
```

Using GridSearchCV to find the best accuracy:

```
[414] from sklearn.preprocessing import PowerTransformer
      from sklearn.model_selection import GridSearchCV
      params_NB = {'var_smoothing': np.logspace(0,-9,num=100)}
      gs_NB = GridSearchCV(estimator=model,
                      param_grid=params_NB,
                      verbose=4,
                      cv=cv_method,
                      scoring='accuracy')
```

```
datatransformed= PowerTransformer().fit_transform(x_test)
gs_NB.fit(datatransformed,y_test)
```

```
[416] gs_NB.best_score_
```

```
0.8940170940170942
```