machine-learning-assignment-2

January 10, 2024

Q1. Use Gaussian Naive bayes to find:

a) Wine auality in wine.csv.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

[2]:
```
wine_data = pd.read_csv('wine.csv')
```

[3]:
```
X = wine_data.drop('quality', axis=1)
y = wine_data['quality']
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[4]:
```
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

[4]:
```
GaussianNB()
```

[5]:
```
y_pred = gnb.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
```

[6]:
```
print(f'Model Accuracy: {accuracy * 100:.2f}%')
```

```
Model Accuracy: 57.59%
```

b)Penguin species from penguin.csv

[19]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Read the CSV file into a pandas DataFrame
penguin_data = pd.read_csv('penguins.csv')
```

```
# Drop missing values for simplicity (if needed)
penguin_data = penguin_data.dropna()

# Encode the 'species' column using LabelEncoder
label_encoder = LabelEncoder()
penguin_data['species'] = label_encoder.fit_transform(penguin_data['species'])

# Assume 'species' is the target variable, and the rest are features
X = penguin_data.drop(['species', 'island', 'sex'], axis=1) # Exclude␣
 ↪non-numeric
y = penguin_data['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=10)

# Create and train the Gaussian Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Make predictions on the test set
y_pred = gnb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.9701492537313433
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96        28
           1       0.92      0.92      0.92        13
           2       1.00      1.00      1.00        26

    accuracy                           0.97        67
   macro avg       0.96      0.96      0.96        67
weighted avg       0.97      0.97      0.97        67
```

2. Use Bernaulli Naive bayes to create model that find:

a) Mushroom class in mushroom.csv

[21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Read the CSV file into a pandas DataFrame
mushroom_data = pd.read_csv('mushrooms.csv')

# Encode the 'class' column using LabelEncoder
label_encoder = LabelEncoder()
mushroom_data['class'] = label_encoder.fit_transform(mushroom_data['class'])

# Assume 'class' is the target variable, and the rest are features
X = mushroom_data.drop('class', axis=1)
y = mushroom_data['class']

# Convert categorical features into numerical using one-hot encoding
X_encoded = pd.get_dummies(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.
 ↪25, random_state=42)

# Create and train the RandomForestClassifier model
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1040
           1       1.00      1.00      1.00       991

    accuracy                           1.00      2031
   macro avg       1.00      1.00      1.00      2031
weighted avg       1.00      1.00      1.00      2031
```

b) Class of breast cancer diagonisis result from breastcancerdata.csv

[22]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, classification_report

# Load the breast cancer dataset
bcan = load_breast_cancer()
data = pd.DataFrame(bcan.data, columns=bcan.feature_names)
data['diagnosis'] = bcan.target

# Assume 'diagnosis' is the target variable, and the rest are features
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create and train the Bernoulli Naive Bayes model
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

# Make predictions on the test set
y_pred = bnb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.6228070175438597
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        43
           1       0.62      1.00      0.77        71

    accuracy                           0.62       114
   macro avg       0.31      0.50      0.38       114
weighted avg       0.39      0.62      0.48       114


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

c) Time is good for play from weather-weka.csv

[23]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import OneHotEncoder

# Load the weather dataset
weather_data = pd.read_csv('weather.csv')

# Convert boolean 'windy' column to strings
weather_data['windy'] = weather_data['windy'].astype(str)

# Convert categorical features to numerical using OneHotEncoder
encoder = OneHotEncoder(sparse=False, drop='first')
weather_encoded = pd.DataFrame(encoder.fit_transform(weather_data[['outlook',␣
 ↪'temperature', 'humidity', 'windy']]))

# Concatenate the encoded features with the original dataset
weather_data = pd.concat([weather_data, weather_encoded], axis=1)

# Assume 'play' is the target variable, and the rest are features
X = weather_data.drop(['play', 'outlook', 'temperature', 'humidity', 'windy'],␣
 ↪axis=1)
y = weather_data['play']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Create and train the Bernoulli Naive Bayes model
bnb = BernoulliNB()
bnb.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = bnb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.5
Classification Report:
               precision    recall  f1-score   support

          no       0.00      0.00      0.00         2
         yes       0.50      1.00      0.67         2

    accuracy                           0.50         4
   macro avg       0.25      0.50      0.33         4
weighted avg       0.25      0.50      0.33         4


/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868:
FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will
be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

d) Survived or not from titanic.csv

[25]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, classification_report
```

```python
from sklearn.preprocessing import LabelEncoder

# Load the Titanic dataset
titanic_data = pd.read_csv('titanic.csv')

# Handle missing values or other preprocessing steps if needed
titanic_data = titanic_data.dropna() # Drop rows with missing values for␣
 ↪simplicit

# Convert categorical features to numerical using LabelEncoder
label_encoder = LabelEncoder()
titanic_data['Sex'] = label_encoder.fit_transform(titanic_data['Sex'])
titanic_data['Embarked'] = label_encoder.fit_transform(titanic_data['Embarked'])

# Assume 'Survived' is the target variable, and the rest are features
X = titanic_data.drop(['Survived', 'Name', 'Ticket', 'Cabin'], axis=1) # Drop␣
 ↪non-
y = titanic_data['Survived']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Create and train the Bernoulli Naive Bayes model
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

# Make predictions on the test set
y_pred = bnb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Survived Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Survived Accuracy: 0.6521739130434783
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.50      0.53        18
           1       0.70      0.75      0.72        28

    accuracy                           0.65        46
   macro avg       0.63      0.62      0.63        46
weighted avg       0.65      0.65      0.65        46
```

Q3.. Use MutinomialNB to predict the movie review from comments.Sample dataset is given: data= [[“I liked the movie”, “positive”],[“It’s a good movie. Nice story”,“positive”],[“Hero’s acting is bad but heroine looks good. Overall nice movie”, “positive”], [“Nice songs. But sadly boring ending.”, “negative”],[“sad movie, boring movie”, “negative”]]

[27]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Sample dataset
data = [
    ["I liked the movie", "positive"],
    ["It’s a good movie. Nice story", "positive"],
    ["Hero’s acting is bad but heroine looks good. Overall nice movie",␣
 ↪"positive"],
    ["Nice songs. But sadly boring ending.", "negative"],
    ["sad movie, boring movie", "negative"]
]

# Create a DataFrame from the sample dataset
df = pd.DataFrame(data, columns=["view", "Boolean"])

# Correct the mapping for 'Boolean' column
df['Boolean'] = df['Boolean'].map({"negative": 0, "positive": 1})

# Display the DataFrame
print(df)

# Split the data into training and testing sets using train_test_split
x_train, x_test, y_train, y_test = train_test_split(df['view'], df['Boolean'],␣
 ↪test_size=0.2)

# Feature extraction using CountVectorizer
count_vector = CountVectorizer()
x_train_vec = count_vector.fit_transform(x_train)
x_test_vec = count_vector.transform(x_test)

# Create and train the Multinomial Naive Bayes model
mnb = MultinomialNB()
mnb.fit(x_train_vec, y_train)

# Make predictions on the test set
y_pred = mnb.predict(x_test_vec)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
                                                view  Boolean
0                                  I liked the movie        1
1                        It's a good movie. Nice story        1
2  Hero's acting is bad but heroine looks good. O…        1
3                Nice songs. But sadly boring ending.        0
4                              sad movie, boring movie        0
Accuracy: 0.0
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       0.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
```

```
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

4. Use KKN classifier and predict the anomaly, by using appropriate features from Dataset3.csv

[28]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the dataset
Dataset3 = pd.read_csv('Dataset_3 - Dataset_3.csv')

# Step 2: Choose appropriate features
# Assuming you select features X_1, X_2, and X_3
X = Dataset3[['avg_num_trades', 'anomaly', 'avg_num_orders']]

# Assuming 'anomaly' is the target variable
y = Dataset3['anomaly']

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Step 4: Standardize the features (optional)
# Standardization may be necessary for KNN
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Train the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train_scaled, y_train)

# Step 6: Make predictions
y_pred = knn_classifier.predict(X_test_scaled)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2010
           1       1.00      1.00      1.00       490

    accuracy                           1.00      2500
   macro avg       1.00      1.00      1.00      2500
weighted avg       1.00      1.00      1.00      2500
```

[31]:
```
from sklearn.model_selection import GridSearchCV
import numpy as np
knn2=KNeighborsClassifier()
```

[32]:
```
param_grid={'n_neighbors':np.arange(1,20)}
#Use grid search
knn_gscv=GridSearchCV(knn2,param_grid,cv=5)
```

[33]:
```
knn_gscv.fit(X,y)
```

[33]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19])})
```

[34]:
```
knn_gscv.best_params_
```

[34]:
```
{'n_neighbors': 2}
```

[35]:
```
knn_gscv.best_score_
```

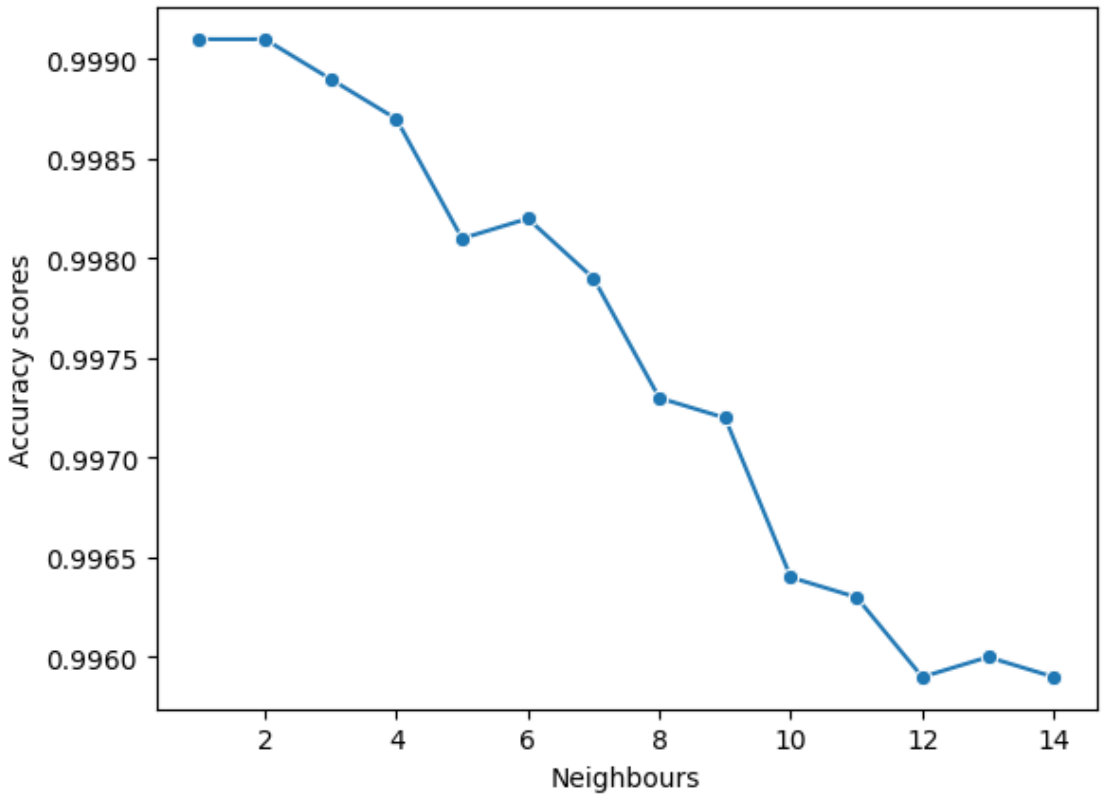
[35]:
```
0.9991000000000001
```

[36]:
```
#Cross-validation scores
from sklearn.model_selection import cross_val_score
```

[38]:
```
k_values=[i for i in range(1,15)]
scores=[]
for k in k_values:
  knn3=KNeighborsClassifier(n_neighbors=k)
  score=cross_val_score(knn3,X,y,cv=5)
  scores.append(np.mean(score))
```

[39]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(x=k_values,y=scores,marker='o')
plt.xlabel("Neighbours")
plt.ylabel("Accuracy scores")
```

[39]:
```
Text(0, 0.5, 'Accuracy scores')
```



5. Do the mobile price classifications on the test.csv file. Given train.csv and find best neighbors using GridsearchCV in KNN classifier.

[41]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Load the training dataset
train_data = pd.read_csv('train - train.csv')

# Load the test dataset
test_data = pd.read_csv('test - test.csv')
```

```
# Assuming the last column is the target variable (price_range)
X_train = train_data.iloc[:, :-1]
y_train = train_data.iloc[:, -1]

# Extract common features between train and test sets
battery_power = X_train.columns.intersection(test_data.columns)
X_test = test_data[battery_power]
y_test = test_data.iloc[:, -1]

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a KNN classifier
knn_classifier = KNeighborsClassifier()

# Define the parameter grid for GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9, 11], 'weights': ['uniform',␣
 ↪'distance']}

# Create GridSearchCV object
grid_search = GridSearchCV(knn_classifier, param_grid, cv=5, scoring='accuracy')

# Fit the model with the best parameters
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Make predictions on the test set
y_pred = grid_search.predict(X_test_scaled)

# Evaluate the model
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Best Parameters: {'n_neighbors': 11, 'weights': 'distance'}
Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.22      0.30       493
           1       0.52      0.27      0.36       507
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
```

```
    accuracy                           0.25      1000
   macro avg       0.25      0.12      0.16      1000
weighted avg       0.49      0.25      0.33      1000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Q6. Take bank.csv file and find whether a person is going to deposit or not. Select appropriate features and do that with different classifier and compare the results.

[45]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('bank - bank.csv')

# Encoding categorical variables
label_encoder = LabelEncoder()
categorical_columns = ['job', 'marital', 'education', 'default', 'housing',␣
 ↪'loan', 'contact', 'month', 'poutcome']
for column in categorical_columns:
    data[column] = label_encoder.fit_transform(data[column])

# Select features and target variable
X = data.drop('deposit', axis=1)
y = data['deposit']

# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define classifiers
classifiers = {
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Support Vector Machine': SVC()
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    print(f"Classifier: {name}")
    print(f"Accuracy: {accuracy}")
    print("Classification Report:\n", report)
    print("="*50)
```

```
Classifier: Random Forest
Accuracy: 0.8289296909986565
Classification Report:
               precision    recall  f1-score   support

           no       0.86      0.81      0.83      1166
          yes       0.80      0.85      0.83      1067

    accuracy                           0.83      2233
   macro avg       0.83      0.83      0.83      2233
weighted avg       0.83      0.83      0.83      2233

==================================================
Classifier: Gradient Boosting
Accuracy: 0.8217644424540976
Classification Report:
               precision    recall  f1-score   support

           no       0.84      0.81      0.83      1166
          yes       0.80      0.84      0.82      1067

    accuracy                           0.82      2233
   macro avg       0.82      0.82      0.82      2233
weighted avg       0.82      0.82      0.82      2233
```

```
================================================
Classifier: Support Vector Machine
Accuracy: 0.7259292431706225
Classification Report:
              precision    recall  f1-score   support

          no       0.71      0.81      0.75      1166
         yes       0.75      0.64      0.69      1067

    accuracy                           0.73      2233
   macro avg       0.73      0.72      0.72      2233
weighted avg       0.73      0.73      0.72      2233

================================================
```

Q7. Take the for_knn.csv and predict the target class column by splitting data in training and test set. Do the problem with KNN, apply standard scaler on dataset compare the results with normal knn also.

[48]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('for_KNN.csv')

# Separate features (X) and target variable (y)
X = data.drop('TARGET CLASS', axis=1)
y = data['TARGET CLASS']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Train KNN on the original data
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Evaluate the performance of KNN on original data
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

```python
print("KNN Classifier Performance on Original Data:")
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
print("="*50)

# Apply StandardScaler to normalize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train KNN on scaled data
knn.fit(X_train_scaled, y_train)

# Make predictions on the scaled test set
y_pred_scaled = knn.predict(X_test_scaled)

# Evaluate the performance of KNN on scaled data
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)
report_scaled = classification_report(y_test, y_pred_scaled)

print("KNN Classifier Performance on Scaled Data:")
print(f"Accuracy: {accuracy_scaled}")
print("Classification Report:\n", report_scaled)
```

```
KNN Classifier Performance on Original Data:
Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       100
           1       0.79      0.77      0.78       100

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.78       200
weighted avg       0.78      0.78      0.78       200

==================================================
KNN Classifier Performance on Scaled Data:
Accuracy: 0.82
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       100
           1       0.81      0.83      0.82       100

    accuracy                           0.82       200
```

```
   macro avg       0.82      0.82      0.82       200
weighted avg       0.82      0.82      0.82       200
```

Q8. A single card is drawn from a standard deck of playing cards. What is the probability that the card is a face card provided that a queen is drawn from the deck of cards?(Use bayes theorem)

Ans= To find the probability that the card drawn is a face card given that a queen is already drawn from the deck, we can use Bayes' theorem.

Let:

```
Event A: Drawing a face card
Event B: Drawing a queen
```

We want to find P(A|B), the probability of drawing a face card given that a queen is already drawn.

Bayes' theorem states:

P(A B)=P(B A)×P(A)P(B)P(A B)=P(B)P(B A)×P(A)

Where:

```
P(A B)P(A B) is the probability of A given B.
P(B A)P(B A) is the probability of B given A.
P(A)P(A) is the probability of A.
P(B)P(B) is the probability of B.
```

In this case:

```
P(A)P(A) is the probability of drawing a face card, which is 12525212 (since there are 12 face
P(B A)P(B A) is the probability of drawing a queen given that it's a face card, which is 112121
P(B)P(B) is the probability of drawing a queen, which is 452524 (since there are 4 queens in a
```

Now, let's calculate P(A B)P(A B):

P(A B)=P(B A)×P(A)P(B)P(A B)=P(B)P(B A)×P(A) P(A B)=112×1252452P(A B)=524121×5212
P(A B)=14P(A B)=41

Therefore, the probability that the card drawn is a face card given that a queen is drawn from the deck is 1441 or 25%.

Q9. A meeting has 12 employees. Given that 8 of the employees is a woman, find the probability that all the employees are women?(Use bayes theorem show calculation in python code)

[49]:
```
# Probability that 8 out of 12 employees are women (P(B))
P_B = (8 / 12)

# Probability that 8 out of 12 employees are women given that all 12 are women␣
↪(P(B|A))
P_B_given_A = 1
```

```
# Using Bayes' theorem to find the probability that all 12 employees are women␣
 ↪(P(A|B))
P_A_given_B = (P_B_given_A * P_B) / P_B

print(f"The probability that all 12 employees are women given that 8 of them␣
 ↪are women is: {P_A_given_B}")
```

```
The probability that all 12 employees are women given that 8 of them are women
is: 1.0
```

Q10.Use playsheet datasheet create the table of prior probability, marginal probability and likelihood for implementing bayes theorem. And do the prediction with a new set of value.

[52]:
```
from sklearn.datasets import load_iris
import pandas as pd

# Load the Iris dataset from scikit-learn
iris = load_iris()

# Convert the dataset into a Pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['Class'] = iris.target

# Calculate prior probabilities
prior_probabilities = iris_df['Class'].value_counts(normalize=True)

# Calculate marginal probabilities for each feature
marginal_probabilities = iris_df.drop('Class', axis=1).apply(lambda col: col.
 ↪value_counts(normalize=True))

# Group data by class for computing likelihoods
grouped = iris_df.groupby('Class')

# Calculate likelihoods for each feature value given each class
likelihoods = {}
for feature in iris_df.columns[:-1]:
    likelihoods[feature] = grouped[feature].value_counts(normalize=True)

# Assuming new values for prediction (you can replace these values)
new_values = {
    'sepal length (cm)': 5.0,
    'sepal width (cm)': 3.2,
    'petal length (cm)': 1.5,
    'petal width (cm)': 0.2
}

# Implementing Bayes' theorem for prediction
```

```
predicted_class = None
max_posterior_probability = 0

for class_label in iris_df['Class'].unique():
    posterior_probability = prior_probabilities[class_label]
    for feature, value in new_values.items():
        if value in likelihoods[feature][class_label]:
            posterior_probability *= likelihoods[feature][class_label][value]
        else:
            # Handle unseen feature values using Laplace smoothing or other␣
 ↪techniques
            posterior_probability *= 1 /␣
 ↪(len(likelihoods[feature][class_label]) + len(iris_df[feature].unique()))

    if posterior_probability > max_posterior_probability:
        max_posterior_probability = posterior_probability
        predicted_class = class_label

print(f"The predicted class for the new values is: {iris.
 ↪target_names[predicted_class]}")
```

```
The predicted class for the new values is: setosa
```