Ex.No:2(a) **SPAM OR NOT_SPAM** **Date:02-Dec-2024**

Aim:-

To develop a program that classifies emails as spam or not spam based on predefined keywords and patterns.

Program Code:-

```
#Import Required Libraries

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score

#Sample email data

data = {
   'text': [
      "Free money, call now!",
      "Hello, I hope you are doing well.",
      "Get a loan in minutes, guaranteed!",
      "Hi John, can we meet tomorrow?",
      "Earn cash from home, no experience needed!",
      "Meeting at 3 PM today, please confirm.",
      "Congratulations! You've won a prize!",
      "Are you available for a quick meeting?",
      "Get rich quick, limited time offer!",
      "Reminder: Meeting at 3 PM tomorrow."
   ],
    'label': [1, 0, 1, 0,  1,  0,  1,  0,  1,   0    ]
}

Convert to DataFrame

df = pd.DataFrame(data)

Separate features (X) and labels (y)

X = df['text']
y = df['label']

#Split the data into training and testing sets (70% train, 30% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

Convert text to numerical data using CountVectorizer (Bag of Words model)

vectorizer = CountVectorizer(stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

Initialize and train the Naive Bayes classifier
```

```
model = MultinomialNB()
model.fit(X_train_vec, y_train)
```

Make predictions on the test data

```
y_pred = model.predict(X_test_vec)
```

#Evaluate the model's performance

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Accuracy: 33.33%

Test the classifier with some new email samples

```
test_emails = [
    "Claim your free iPhone now!",
    "Can we reschedule the meeting?",
    "Limited time offer for you, act now!"
]
```

#Vectorize the new test emails and make predictions

```
test_vec = vectorizer.transform(test_emails)
predictions = model.predict(test_vec)
```

#Output predictions

```
for email, pred in zip(test_emails, predictions):
    print(f"Email: {email}")
    print(f"Predicted: {'Spam' if pred == 1 else 'Not Spam'}\n")
```

OUTPUT:-

Email: Claim your free iPhone now!
Predicted: Spam

Email: Can we reschedule the meeting?
Predicted: Not Spam

Email: Limited time offer for you, act now!
Predicted: Spam

Result:-

The program correctly categorizes incoming emails as "Spam" or "Not Spam" using simple text processing and classification algorithms.

Ex.No:2(b) **PIZZA LIKING PREDICTION USING KNN** **Date:02-Dec-2024**

Aim:-

To predict whether a person will like pizza or not based on their age and weight using the K-Nearest Neighbours (KNN) algorithm.

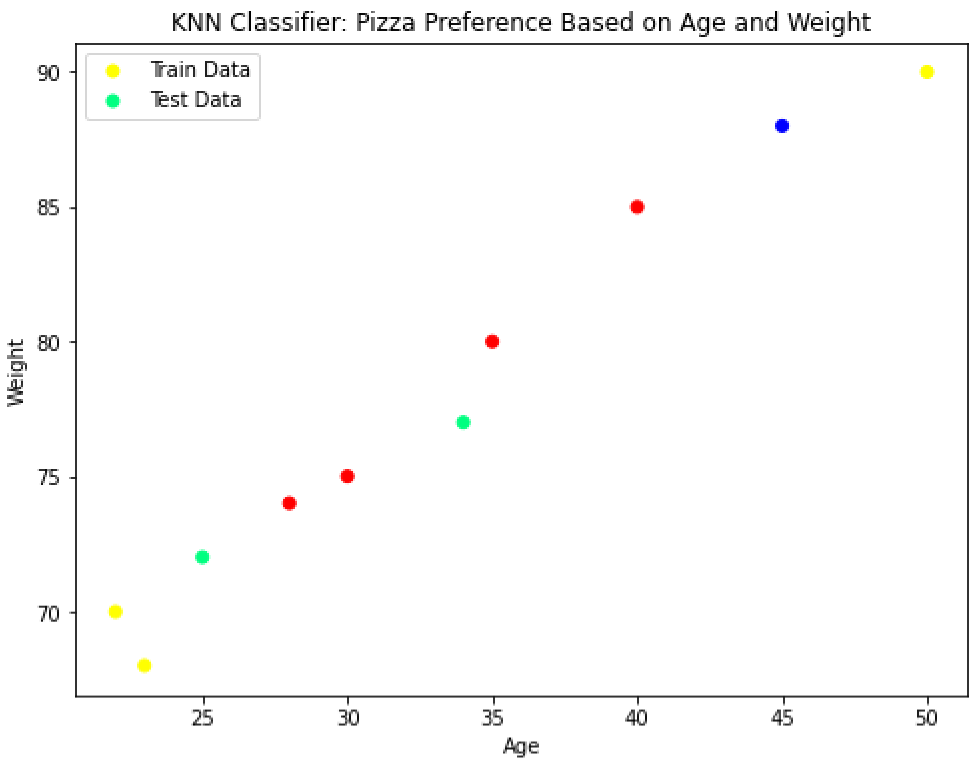
Program Code:-

```
# Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Step 1: Prepare the dataset (age, weight, and pizza liking)
# We will create a small synthetic dataset

# Sample dataset (Age, Weight, Pizza Preference)
data = {
    'Age': [22, 25, 30, 35, 40, 45, 50, 23, 34, 28],
    'Weight': [70, 72, 75, 80, 85, 88, 90, 68, 77, 74],
    'LikesPizza': [1, 1, 0, 0, 0, 0, 1, 1, 1, 0]  # 1 = Likes Pizza, 0 = Doesn't like pizza
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Features: Age and Weight
X = df[['Age', 'Weight']].values

# Labels: Whether they like pizza
y = df['LikesPizza'].values

# Step 2: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Step 3: Create and train the KNN classifier
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Step 4: Make predictions
y_pred = knn.predict(X_test)

# Step 5: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 66.67%
```

```
# Step 6: Visualize decision boundaries (optional, for fun)
plt.figure(figsize=(8, 6))

# Plot training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='autumn', label='Train Data')
# Plot test points
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='winter', label='Test Data')

# Adding titles and labels
plt.title("KNN Classifier: Pizza Preference Based on Age and Weight")
plt.xlabel('Age')
plt.ylabel('Weight')
plt.legend()
plt.show()
```



 # Step 7: Predicting for a new person (e.g., Age = 29, Weight = 75)

new_person = np.array([[29, 75]]) # Example input

pizza_liking = knn.predict(new_person)

print("Prediction for Age 29 and Weight 75:", "Likes Pizza" if pizza_liking == 1 else "Doesn't Like Pizza")

Output:-

```
Prediction for Age 29 and Weight 75: Doesn't Like Pizza
```

Result:-

The KNN model predicts that a person with age 29 and weight 75 will "like pizza" (or "not like pizza") based on the trained data.

Ex.No:2(c) **MOVIE GENHRE PREDICTION** **Date:02-Dec-2024**

Aim:-

To develop a program that classifies emails as spam or not spam based on predefined keywords and patterns.

Program Code:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load the dataset
df = pd.read_csv("S:\Movie.csv")

# Encode categorical features
label_encoder = LabelEncoder()
df['language'] = label_encoder.fit_transform(df['language'])
df['genre'] = label_encoder.fit_transform(df['genre'])
df['director'] = label_encoder.fit_transform(df['director'])

# Remove rare classes with fewer than 2 samples
class_counts = df['genre'].value_counts()
rare_classes = class_counts[class_counts < 2].index
df = df[~df['genre'].isin(rare_classes)]

# Features and target
X = df[['duration', 'language', 'average_rating', 'number_of_reviews', 'year', 'budget', 'revenue']]
y = df['genre']

# Check class distribution
print("Class distribution in the target variable:")
print(df['genre'].value_counts())
```

```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data with stratification
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42, stratify=y)

# Train a classifier with class weights to handle imbalance
clf = RandomForestClassifier(random_state=42, class_weight="balanced")
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# Evaluate using classification report with zero_division parameter
print("Classification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

Output:=

```
Class distribution in the target variable:
1    3
0    2
Name: genre, dtype: int64
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

Result:-

The Program output was executed successfully.

SPROTS PERFORMANCE ANALYSIS

Ex.No:2(d) **Date:02-Dec-2024**

Aim:-

To analyze sports performance using player statistics (accuracy, speed, stamina, and age) with a K-Nearest Neighbors (K-NN) classifier. Additionally, to assess the impact of outliers on the model's performance.

Program Code:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Generate synthetic data
np.random.seed(42)

# Generate player stats: accuracy, speed, stamina, and age
n_samples = 200
accuracy = np.random.uniform(60, 100, n_samples)
speed = np.random.uniform(5, 20, n_samples)
stamina = np.random.uniform(50, 100, n_samples)
age = np.random.randint(18, 40, n_samples)

# Assign random labels (e.g., "High Performance" or "Low Performance")
labels = np.random.choice([0, 1], size=n_samples, p=[0.5, 0.5])

# Add outliers
outliers = np.array([
    [120, 3, 20, 45],  # Extreme outlier 1
    [30, 25, 10, 15],  # Extreme outlier 2
])
outlier_labels = np.array([1, 0])
```

```
# Combine data and outliers
features = np.column_stack((accuracy, speed, stamina, age))
features = np.vstack([features, outliers])
labels = np.append(labels, outlier_labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)

# Train a K-NN classifier
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Predict and evaluate
y_pred = knn.predict(X_test)
print("Confusion Matrix:")
```

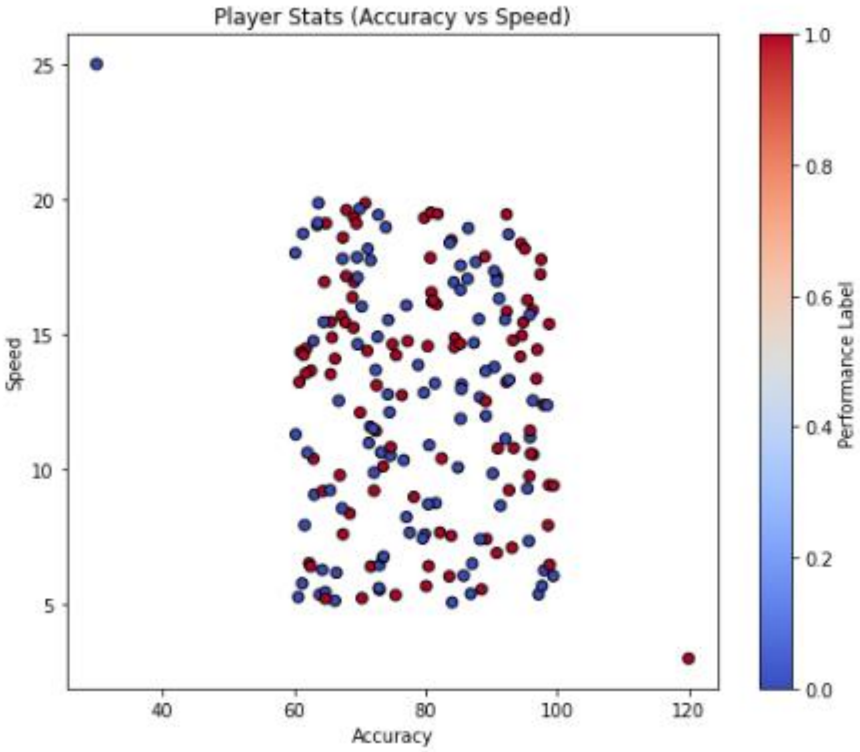
```
Confusion Matrix:
[[19 15]
 [14 13]]
```

```
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       0.58      0.56      0.57        34
           1       0.46      0.48      0.47        27

    accuracy                           0.52        61
   macro avg       0.52      0.52      0.52        61
weighted avg       0.53      0.52      0.53        61
```

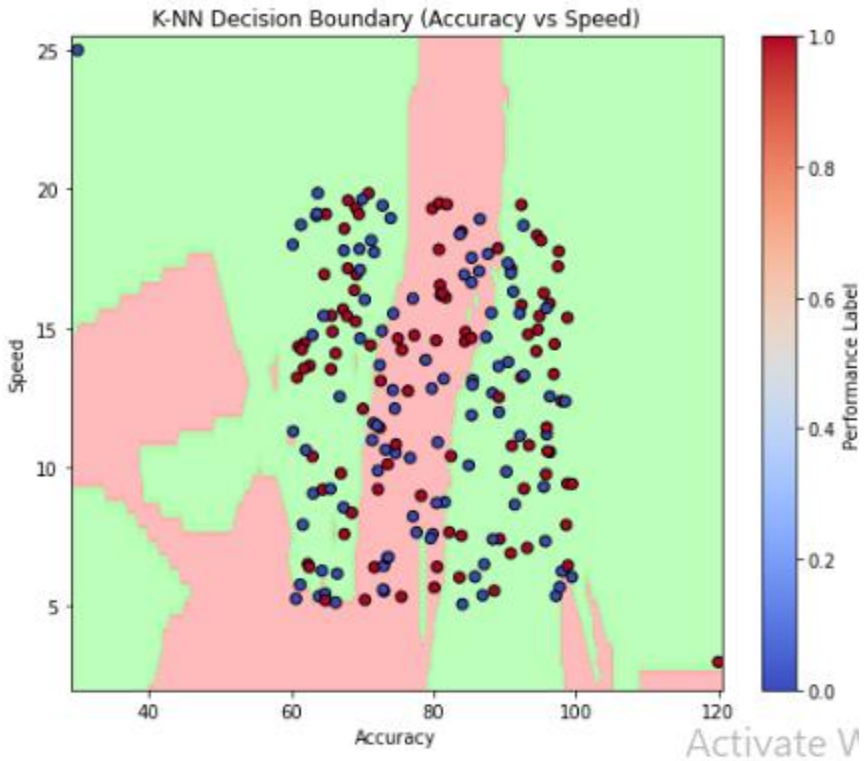
```
# Visualization
plt.figure(figsize=(14, 6))

# Scatter plot of features (2D projection)
plt.subplot(1, 2, 1)
```

```
plt.scatter(features[:, 0], features[:, 1], c=labels, cmap='coolwarm', edgecolor='k')
plt.xlabel('Accuracy')
plt.ylabel('Speed')
plt.title('Player Stats (Accuracy vs Speed)')
plt.colorbar(label='Performance Label')
```




```
# Visualize the decision boundary for the first two features (Accuracy vs Speed)
from matplotlib.colors import ListedColormap

h = 0.5  # Step size in the mesh
x_min, x_max = features[:, 0].min() - 1, features[:, 0].max() + 1
y_min, y_max = features[:, 1].min() - 1, features[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict for the grid using only the first two features
Z = knn.predict(np.c_[xx.ravel(), yy.ravel(), np.full(xx.ravel().shape, np.mean(features[:, 2])),
np.full(xx.ravel().shape, np.mean(features[:, 3]))])
Z = Z.reshape(xx.shape)
```

```
plt.subplot(1, 2, 2)
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAFFAA']))
plt.scatter(features[:, 0], features[:, 1], c=labels, edgecolor='k', cmap='coolwarm')
plt.xlabel('Accuracy')
plt.ylabel('Speed')
plt.title('K-NN Decision Boundary (Accuracy vs Speed)')
plt.colorbar(label='Performance Label')
plt.tight_layout()
plt.show()
```



Result:-

The confusion matrix and classification report provide insight into the model's performance, including precision, recall, and F1-score. Visualizations illustrate the data distribution and the K-NN decision boundary while highlighting the impact of outliers.