EXPERIMENT – 11

AIM: To implement the K-Nearest Neighbors (KNN) algorithm on a sample dataset stored in a .CSV file and evaluate its performance by calculating accuracy, precision, and recall.

DESCRIPTION:

K-Nearest Neighbors (KNN) is a supervised learning algorithm commonly used for classification. It classifies a data point based on how its neighbors are classified. Given a labeled dataset, the algorithm calculates the distances between data points and assigns the class of the majority of its nearest neighbors (K nearest) to the target.

To determine the nearest neighbors in KNN, various distance metrics are used:

1. **Euclidean Distance**: This measures the straight-line distance between two points in a plane. It's calculated as the square root of the sum of squared differences between corresponding coordinates.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2. **Manhattan Distance**: Used when the total path traveled is more important than the displacement. It's the sum of absolute differences between coordinates.

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

3. **Minkowski Distance**: A generalized metric where Euclidean and Manhattan distances are special cases. For p=2p = 2p=2, it's Euclidean, and for p=1p = 1p=1, it's Manhattan.

$$d(x,y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

Other metrics, such as **Hamming Distance**, are useful for comparing categorical or binary data by counting differences in corresponding elements.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# load data from CSV
df = pd.read_csv('/content/Social_Network_Ads.csv')
df.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

t steps: Generate code with df | View recommended plots | New interactive sheet

```
# Obtain Train data and Train output
X = df.drop(columns=['Purchased'])# Features
y = df['Purchased']   # Target label
```

```
X.head()
```

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000

```
y.head()
```

	Purchased
0	0
1	0
2	0
3	0
4	0

dtype: int64

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
def evaluate_knn(metric):
    knn = KNeighborsClassifier(n_neighbors=3, metric=metric)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')

    print(f"Metrics for KNN using {metric} distance:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("----------")
```

```
evaluate_knn("euclidean")
evaluate_knn("manhattan")
evaluate_knn("minkowski")
```

```
Metrics for KNN using euclidean distance:
Accuracy: 0.81
Precision: 0.8023897058823529
Recall: 0.7822822822822822
----------
Metrics for KNN using manhattan distance:
Accuracy: 0.81
Precision: 0.8023897058823529
Recall: 0.7822822822822822
----------
Metrics for KNN using minkowski distance:
Accuracy: 0.81
Precision: 0.8023897058823529
Recall: 0.7822822822822822
----------
```

OUTPUT ANALYSIS:

- **Accuracy (81%)**: This indicates that 81% of the predictions made by the KNN model were correct. This is a decent result, suggesting the model performs relatively well on this dataset.
- **Precision (0.80)**: The precision of 0.80 (or 80%) implies that out of all the instances predicted as positive, 80% were actually positive. This suggests a moderate level of false positives, meaning the model occasionally misclassifies instances that are not positive as positive.
- **Recall (0.78)**: A recall of 0.78 (or 78%) indicates that out of all the actual positive instances, 78% were correctly identified by the model. This is reasonably good but indicates there is still room for improvement in identifying true positives, as 22% of positives were missed.

CONCLUSION:

"In conclusion, the kNN algorithm performed well in terms of [specific metrics], showing strong predictive power with an optimal k of [value]. However, misclassification occurred mainly in [specific classes or regions], indicating that feature selection or alternative models could improve performance. Given these results, kNN provides a simple but effective approach to this task, although further optimization may be required for deployment."
In summary, in this example Naïve Bayes classifier is effective at capturing positive cases, but further optimization may be necessary to improve the precision and overall reliability of the predictions.