Day52_K-Nearest_Neighbors_(KNN)_Classifier

July 25, 2025

Introduction

K-Nearest Neighbors (KNN) is a simple and powerful machine learning algorithm used for **classification** and **regression**. It works by finding the **'k' closest data points** to a new input and making predictions based on them.

There are two types of KNN:

- **KNN Classifier**: Assigns the class that most of the nearest neighbors belong to (majority voting).
- **KNN Regressor**: Predicts a value by averaging the values of the nearest neighbors.

Distance Metrics in KNN

- **Euclidean Distance (p=2)**: Straight-line distance between two points in space, calculated as:
$$\text{Euclidean} = \sqrt{\sum (x_i - y_i)^2}$$
- **Manhattan Distance (p=1)**: Distance measured along axes at right angles (like a grid):
$$\text{Manhattan} = \sum |x_i - y_i|$$

Both methods measure "closeness," and the best choice depends on your data's shape and scale.

Scaling is Important: Always apply scaling (like StandardScaler) to ensure all features are treated equally in distance calculations.

Balancing Imbalanced Data If your dataset has 99% of one class and 1% of another, your model may become biased.

- **Case 1 (Model 1)**: Downsample the majority class to 70%, increase minority samples → Accuracy: a1
- **Case 2 (Model 2)**: 75% downsample + 25% upsample → Accuracy: a2
- **Case 3 (Model 3)**: 80% downsample + 20% upsample → Accuracy: a3
- Final balanced model = (a1 + a2 + a3) / 3

This process is known as **SMOTE** – Synthetic Minority Over-sampling Technique.

How KNN Works

1. Choose the number of neighbors **k**.
2. Calculate the distance between the new data point and every other point in the dataset.
3. Sort the distances and choose the top **k** nearest neighbors.
4. For **classification**, return the most common class.

5. For **regression**, return the average of their values.

Manual Example (Math + CS Marks)

We want to predict the result (pass/fail) of a student with: - **Math = 6**, **CS = 8** using **K = 3**

Math	CS	Result
4	3	F
6	7	P
7	8	P
5	5	F
8	8	P

We'll use **Euclidean distance**:

[2]:
```
import numpy as np
from collections import Counter

samples = np.array([[4,3],[6,7],[7,8],[5,5],[8,8]])
labels = np.array(['f','p','p','f','p'])
new_point = np.array([6,8])
```

[3]:
```
# Step 1: Compute distances
distances = [np.sqrt(((new_point - pt)**2).sum()) for pt in samples]
```

[4]:
```
# Step 2: Show distances
for i, d in enumerate(distances):
    print(f"Distance to {samples[i]} (label={labels[i]}): {d:.2f}")
```

```
Distance to [4 3] (label=f): 5.39
Distance to [6 7] (label=p): 1.00
Distance to [7 8] (label=p): 1.00
Distance to [5 5] (label=f): 3.16
Distance to [8 8] (label=p): 2.00
```

[5]:
```
# Step 3: Top 3 Neighbors
sorted_idx = np.argsort(distances)[:3]
nearest_labels = labels[sorted_idx]
print("\nTop 3 Nearest Labels:", nearest_labels)
```

```

Top 3 Nearest Labels: ['p' 'p' 'p']
```

[6]:
```
# Step 4: Predict by majority
print("Predicted class:", Counter(nearest_labels).most_common(1)[0][0])
```

```
Predicted class: p
```

1 Import Required Libraries

[7]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

2 Load Dataset

[8]:
```
dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\logit classification.csv")  #␣
↪Adjust path as needed
dataset.head()
```

[8]:

	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

3 Feature Selection

[9]:
```
X = dataset[["Age", "EstimatedSalary"]].values
y = dataset["Purchased"].values
```

4 Train-Test Split

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=0)
```

5 Apply StandardScaler

[11]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

6 Try KNN with different parameters (one by one for beginners)

6.1 Model A: k=3, p=1 (Manhattan)

[12]:
```
# Model A: k=3, p=1 (Manhattan)
model_a = KNeighborsClassifier(n_neighbors=3, p=1)
model_a.fit(X_train_scaled, y_train)
y_pred_a = model_a.predict(X_test_scaled)
print("Model A - k=3, p=1 (Manhattan):", accuracy_score(y_test, y_pred_a))
```

```
Model A - k=3, p=1 (Manhattan): 0.93
```

6.2 Model B: k=3, p=2 (Euclidean)

[13]:
```
# Model B: k=3, p=2 (Euclidean)
model_b = KNeighborsClassifier(n_neighbors=3, p=2)
model_b.fit(X_train_scaled, y_train)
y_pred_b = model_b.predict(X_test_scaled)
print("Model B - k=3, p=2 (Euclidean):", accuracy_score(y_test, y_pred_b))
```

```
Model B - k=3, p=2 (Euclidean): 0.93
```

6.3 Model C: k=4, p=1 (Manhattan)

[14]:
```
# Model C: k=4, p=1 (Manhattan)
model_c = KNeighborsClassifier(n_neighbors=4, p=1)
model_c.fit(X_train_scaled, y_train)
y_pred_c = model_c.predict(X_test_scaled)
print("Model C - k=4, p=1 (Manhattan):", accuracy_score(y_test, y_pred_c))
```

```
Model C - k=4, p=1 (Manhattan): 0.93
```

6.4 Model D: k=4, p=2 (Euclidean)

[15]:
```
# Model D: k=4, p=2 (Euclidean)
model_d = KNeighborsClassifier(n_neighbors=4, p=2)
model_d.fit(X_train_scaled, y_train)
y_pred_d = model_d.predict(X_test_scaled)
print("Model D - k=4, p=2 (Euclidean):", accuracy_score(y_test, y_pred_d))
```

```
Model D - k=4, p=2 (Euclidean): 0.92
```

6.5 Model E: k=5, p=1 (Manhattan)

[16]:
```
# Model E: k=5, p=1 (Manhattan)
model_e = KNeighborsClassifier(n_neighbors=5, p=1)
model_e.fit(X_train_scaled, y_train)
y_pred_e = model_e.predict(X_test_scaled)
print("Model E - k=5, p=1 (Manhattan):", accuracy_score(y_test, y_pred_e))
```

```
Model E - k=5, p=1 (Manhattan): 0.93
```

6.6 Model F: k=5, p=2 (Euclidean)

[17]:
```python
# Model F: k=5, p=2 (Euclidean)
model_f = KNeighborsClassifier(n_neighbors=5, p=2)
model_f.fit(X_train_scaled, y_train)
y_pred_f = model_f.predict(X_test_scaled)
print("Model F - k=5, p=2 (Euclidean):", accuracy_score(y_test, y_pred_f))
```

```
Model F - k=5, p=2 (Euclidean): 0.93
```

7 Without scaling

[18]:
```python
# Without scaling
model_raw = KNeighborsClassifier(n_neighbors=4, p=1)
model_raw.fit(X_train, y_train)
y_pred_raw = model_raw.predict(X_test)
acc_raw = accuracy_score(y_test, y_pred_raw)
print("\nAccuracy without scaling:", acc_raw)
```

```
Accuracy without scaling: 0.81
```

8 Results Comparison Table

[20]:
```python
results_df = pd.DataFrame({
    "Model": ["A", "B", "C", "D", "E", "F", "G (No Scale)"],
    "k": [3, 3, 4, 4, 5, 5, 4],
    "Distance Metric (p)": [1, 2, 1, 2, 1, 2, 1],
    "Accuracy": [
        accuracy_score(y_test, y_pred_a),
        accuracy_score(y_test, y_pred_b),
        accuracy_score(y_test, y_pred_c),
        accuracy_score(y_test, y_pred_d),
        accuracy_score(y_test, y_pred_e),
        accuracy_score(y_test, y_pred_f),
        acc_raw
    ]
})
print("\nComparison Table:\n")
print(results_df)
```

```
Comparison Table:

          Model  k  Distance Metric (p)  Accuracy
```

0	A	3	1	0.93
1	B	3	2	0.93
2	C	4	1	0.93
3	D	4	2	0.92
4	E	5	1	0.93
5	F	5	2	0.93
6	G (No Scale)	4	1	0.81

9 Bonus: Loop all combinations for flexibility

9.1 Try different KNN parameters and compare using list logic (for advanced learners)

```
results_loop = []
for k in [3, 4, 5]:
    for p in [1, 2]:
        model = KNeighborsClassifier(n_neighbors=k, p=p)
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        acc = accuracy_score(y_test, y_pred)
        results_loop.append((k, p, acc))
        print(f"k={k}, p={p} → Accuracy: {acc:.4f}")
```

10 Conclusion

- KNN works well for both classification and regression when properly scaled.
- Euclidean (p=2) and Manhattan (p=1) distances give slightly different results depending on data.
- Accuracy varies with k; optimal value of k must be found using testing or cross-validation.
- Scaling improved accuracy from 0.81 (raw) to over 0.93 in some configurations.
- We manually verified prediction with distance calculations.
- Use a comparison table to select the best k and distance combination.