# Lab No # 07

## KNN Method

Name Maaz Ahmad
Reg No B23F0722AI170
Section (Ai Yellow)
Submitted to Abdullah Sajid

---

## TASK 1A – Manual Implementation and Distance Exploration

```
import numpy as np
import pandas as pd

data = {'Brightness':[40,70,45,60,80,30,50,90,85,35],
        'Saturation':[20,90,75,60,80,25,50,85,70,40],
        'Class':['Red','Blue','Red','Blue','Blue','Red','Blue','Blue','Red','Red']}
df = pd.DataFrame(data)
X = df[['Brightness','Saturation']].values
y = np.array(df['Class'])
new_point = np.array([20,35])
k = 5

def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))
def manhattan(a,b): return np.sum(np.abs(a-b))
def minkowski(a,b,p): return np.sum(np.abs(a-b)**p)**(1/p)

def knn_predict(X_train,y_train,X_test,k,dist):
    d = [dist(X_test,x) for x in X_train]
    idx = np.argsort(d)[:k]
    nearest = y_train[idx]
    values, counts = np.unique(nearest,return_counts=True)
    return values[np.argmax(counts)]

print("Euclidean:", knn_predict(X,y,new_point,k,euclidean))
print("Manhattan:", knn_predict(X,y,new_point,k,manhattan))
print("Minkowski p=3:", knn_predict(X,y,new_point,k,lambda a,b: minkowski(a,b,3)))
```

## Summary

In this task, we implemented the KNN algorithm manually using Python and NumPy. We calculated distances between the new point and all other points using Euclidean, Manhattan, and Minkowski distance formulas. The goal was to observe how the choice of distance metric affects classification results. It helped understand how KNN identifies the nearest neighbors and predicts the class.

---

## TASK 1B – Weighted Voting & Tie Resolution

```
import numpy as np
import pandas as pd

data = {'Brightness':[40,70,45,60,80,30,50,90,85,35],
        'Saturation':[20,90,75,60,80,25,50,85,70,40],
        'Class':['Red','Blue','Red','Blue','Blue','Red','Blue','Blue','Red','Red']}
df = pd.DataFrame(data)
X = df[['Brightness','Saturation']].values
y = np.array(df['Class'])
new_point = np.array([20,35])
k = 5

def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))
def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))

def weighted_knn(X_train,y_train,X_test,k,dist):
    d = np.array([dist(X_test,x) for x in X_train])
    idx = np.argsort(d)[:k]
    nearest_labels = y_train[idx]
    nearest_d = d[idx]
    w = 1/(nearest_d+1e-5)
    classes = np.unique(nearest_labels)
    scores = {c:np.sum(w[nearest_labels==c]) for c in classes}
    return max(scores,key=scores.get)

print("Weighted Prediction:", weighted_knn(X,y,new_point,k,euclidean))
```

## Summary

This task extended the basic KNN by adding weighted voting. Here, closer neighbors were given higher importance using inverse distance (1/d). We compared weighted and unweighted predictions to see the effect on the final result. It showed that weighting improves accuracy when nearby points are more relevant.

---

## TASK 1C – Scaling and Feature Impact

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

data = {'Brightness':[40,70,45,60,80,30,50,90,85,35],
        'Saturation':[20,90,75,60,80,25,50,85,70,40],
        'Class':['Red','Blue','Red','Blue','Blue','Red','Blue','Blue','Red','Red']}
df = pd.DataFrame(data)
df_scaled = df.copy()
df_scaled['Saturation'] *= 10
X = df_scaled[['Brightness','Saturation']].values
y = np.array(df_scaled['Class'])
new_point = np.array([20,35])
k = 5

def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))
def knn_predict(X_train,y_train,X_test,k,dist):
    d = [dist(X_test,x) for x in X_train]
    idx = np.argsort(d)[:k]
    nearest = y_train[idx]
    values,counts = np.unique(nearest,return_counts=True)
    return values[np.argmax(counts)]

scaler = StandardScaler()
X_std = scaler.fit_transform(X)
pred = knn_predict(X_std,y,scaler.transform([new_point])[0],k,euclidean)
print("Scaled Prediction:", pred)
```

## Summary

We tested how feature scaling affects distance calculations in KNN. By multiplying one feature’s range, we saw that unscaled data gives biased results. After applying StandardScaler, both features contributed equally to distance. This showed why scaling is essential in distance-based algorithms like KNN.

---

## TASK 1D – Parameter K Exploration

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

data = {'Brightness':[40,70,45,60,80,30,50,90,85,35],
        'Saturation':[20,90,75,60,80,25,50,85,70,40],
        'Class':['Red','Blue','Red','Blue','Blue','Red','Blue','Blue','Red','Red']}
df = pd.DataFrame(data)
X = df[['Brightness','Saturation']].values
y = np.array(df['Class'])
new_point = np.array([20,35])

def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))
def knn_predict(X_train,y_train,X_test,k,dist):
    d = [dist(X_test,x) for x in X_train]
    idx = np.argsort(d)[:k]
    nearest = y_train[idx]
    values,counts = np.unique(nearest,return_counts=True)
    return values[np.argmax(counts)]

ks = [1,3,5,7]
preds = [knn_predict(X,y,new_point,k,euclidean) for k in ks]
plt.plot(ks,preds,marker='o')
plt.xlabel('K')
plt.ylabel('Predicted Class')
plt.title('K vs Predicted Class')
plt.show()
```

## Summary

We varied the number of neighbors (k = 1, 3, 5, 7) and observed prediction changes. A small k made the model sensitive to noise, while a larger k gave smoother predictions. We plotted k vs predicted class to visualize stability. This demonstrated the importance of choosing an optimal k value.

---

## TASK 2A – Exploratory Analysis & Preprocessing

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
import pandas as pd

data = load_breast_cancer()
X = data.data
y = data.target
df = pd.DataFrame(X,columns=data.feature_names)
df['target'] = y
print(df.describe())
print(df.isnull().sum())
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

## Summary

We loaded the Breast Cancer dataset and examined features, missing values, and distributions. Feature scaling was done using StandardScaler to prepare the data for KNN. This step ensured all attributes were on a similar scale for accurate distance measurement. It was the data-cleaning and preparation phase before model training.

---

## TASK 2B – KNN from Scratch on Subset

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

data = load_breast_cancer()
X = data.data
y = data.target
X = StandardScaler().fit_transform(X)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

def euclidean(a,b): return np.sqrt(np.sum((a-b)**2))
def knn_predict(X_train,y_train,X_test,k):
    preds = []
    for x in X_test:
        d = [euclidean(x,t) for t in X_train]
        idx = np.argsort(d)[:k]
        nearest = y_train[idx]
        values,counts = np.unique(nearest,return_counts=True)
        preds.append(values[np.argmax(counts)])
    return np.array(preds)

y_pred = knn_predict(X_train[:200],y_train[:200],X_test[:10],5)
print("Accuracy (subset):", np.mean(y_pred==y_test[:10]))
```

## Summary

We manually implemented KNN for a small subset of the dataset using NumPy. The model calculated distances and predicted labels for test samples. Accuracy was checked to verify correct working of the algorithm. This task showed the internal working of KNN without using external libraries.

---

## TASK 2C – Scikit-learn KNN & Tuning

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

data = load_breast_cancer()
X = StandardScaler().fit_transform(data.data)
y = data.target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

knn = KNeighborsClassifier()
params = {'n_neighbors':[1,3,5,7,9],'weights':['uniform','distance']}
grid = GridSearchCV(knn,params,cv=5)
grid.fit(X_train,y_train)
best = grid.best_estimator_
y_pred = best.predict(X_test)

print("Best Parameters:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test,y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test,y_pred))
print("Report:\n", classification_report(y_test,y_pred))
```

## Summary

We implemented KNN using the scikit-learn library for simplicity and speed. A GridSearchCV was used to test multiple values of k and weight options. The best k and weighting scheme were selected automatically based on accuracy. We also displayed the confusion matrix and performance report.

---

## TASK 2D – Accuracy vs K (Elbow Method)

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

data = load_breast_cancer()
X = StandardScaler().fit_transform(data.data)
y = data.target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

k_values = range(1,15,2)
acc = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train,y_train)
    acc.append(model.score(X_test,y_test))

plt.plot(k_values,acc,marker='o')
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.title("K vs Accuracy (Elbow Method)")
plt.show()
```

## Summary

We tested different k values (1–15) and plotted accuracy for each one. The elbow method helped visualize where accuracy stops improving. This graph helped find the most stable and optimal k for the model. It showed how increasing k affects bias and variance in KNN performance.

---

## KNN Method Summary

1. Task 1A: Implemented KNN manually and compared Euclidean, Manhattan, and Minkowski distances to see how each affects classification.
2. Task 1B: Added weighted voting using inverse distance so closer neighbors have more influence on the prediction.
3. Task 1C: Tested feature scaling effects; scaling made both features contribute equally and improved accuracy.
4. Task 1D: Changed k values (1, 3, 5, 7) to observe how prediction stability changes with different neighbor counts.
5. Task 2A: Loaded and analyzed the Breast Cancer dataset, checked missing values, and scaled features with StandardScaler.
6. Task 2B: Built KNN from scratch on a small subset to verify algorithm logic and calculate simple accuracy.
7. Task 2C: Used scikit-learn's KNeighborsClassifier with GridSearchCV to find the best k and weighting scheme automatically.
8. Task 2D: Plotted accuracy vs k (Elbow Method) to determine the most stable and optimal number of neighbors.

---