Name:**Atharva Phalke**
Class:**D15C** Roll no.**39**
Subject:**MLDL**

Experiment No.4

AIM:Implement K-Nearest Neighbors (KNN) and evaluate model performance.

1. Dataset Source

- **Dataset Name:** Breast Cancer Wisconsin (Diagnostic) Data Set
- **Source:** Kaggle - Breast Cancer Wisconsin (Diagnostic)
- **Original Source:** UCI Machine Learning Repository (Dr. William H. Wolberg, University of Wisconsin)

2. Dataset Description

This dataset contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The goal is to classify the tumor as either **Malignant** (cancerous) or **Benign** (non-cancerous) based on cell nucleus characteristics.

- **Size:** 569 records, 32 columns.
- **Target Variable:** `diagnosis` (M = Malignant, B = Benign).
- **Key Features:** * Radius, Texture, Perimeter, and Area.
  - Smoothness, Compactness, Concavity, and Concave points.
  - Symmetry and Fractal dimension.

3. Mathematical Formulation of the Algorithm

KNN is a non-parametric, lazy learning algorithm. It does not "learn" a model (like finding coefficients in regression); instead, it memorizes the training data.

A. Similarity Metric (Euclidean Distance)

To classify a new data point (x), the algorithm calculates its distance to every point in the training set (x^{(i)}). The most common metric is Euclidean Distance:

$$d(x, x^{(i)}) = \sqrt{\sum_{j=1}^{n} (x_j - x_j^{(i)})^2}$$

Where n is the number of features (4 in this case).

B. Classification Rule

1. Find the K nearest neighbors (points with the smallest distance d).
2. Assign the new point to the class that is most common (Mode) among those K neighbors.

$$\hat{y} = \text{mode}(y_1, y_2, \ldots, y_K)$$

4. Algorithm Limitations

1. **Computational Cost:** It is "lazy," meaning all computation happens at prediction time. For large datasets, calculating the distance to *every* training point is slow.

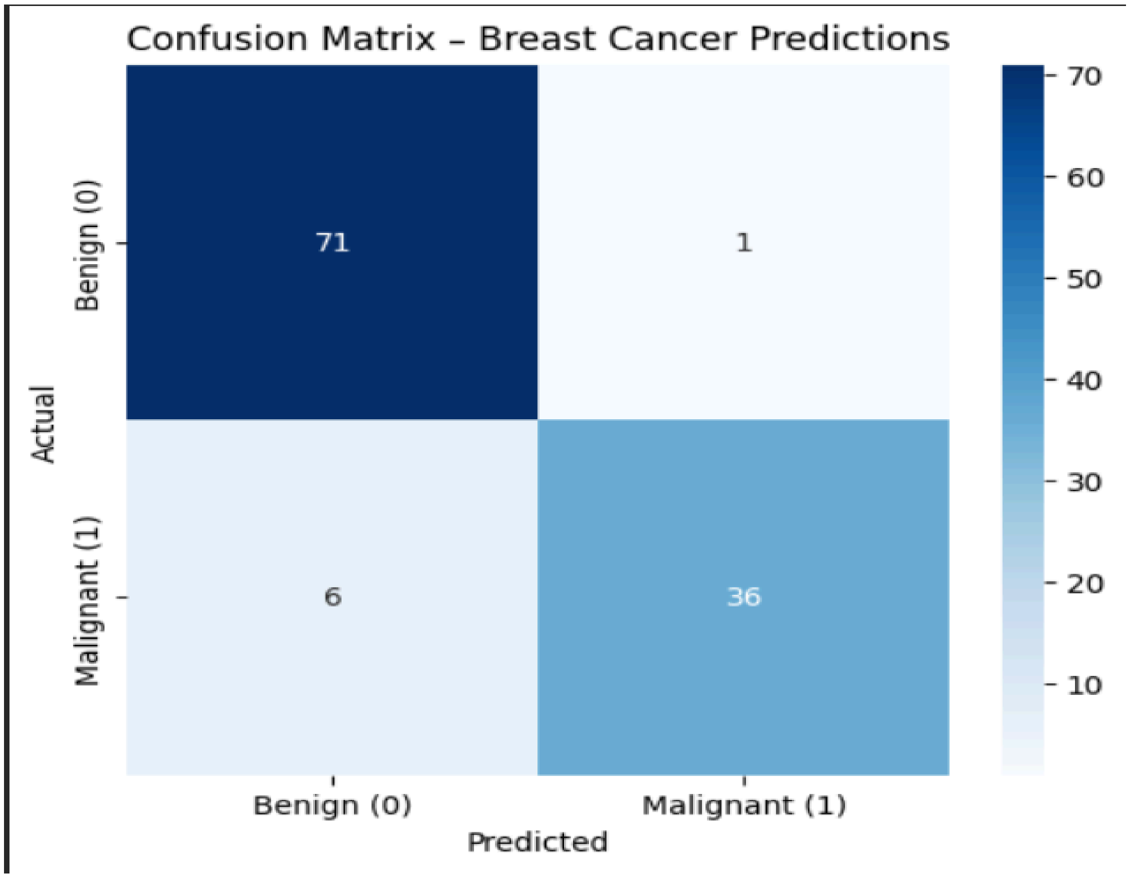
2. **Sensitivity to Outliers:** If $K$ is too small (e.g., K=1), a single mislabeled outlier can completely change the prediction.
3. **Scale Sensitivity:** KNN relies on distance. If one feature is measured in millimeters (e.g., 1000mm) and another in meters (e.g., 1m), the larger number will dominate the distance calculation. **Feature Scaling is mandatory.**

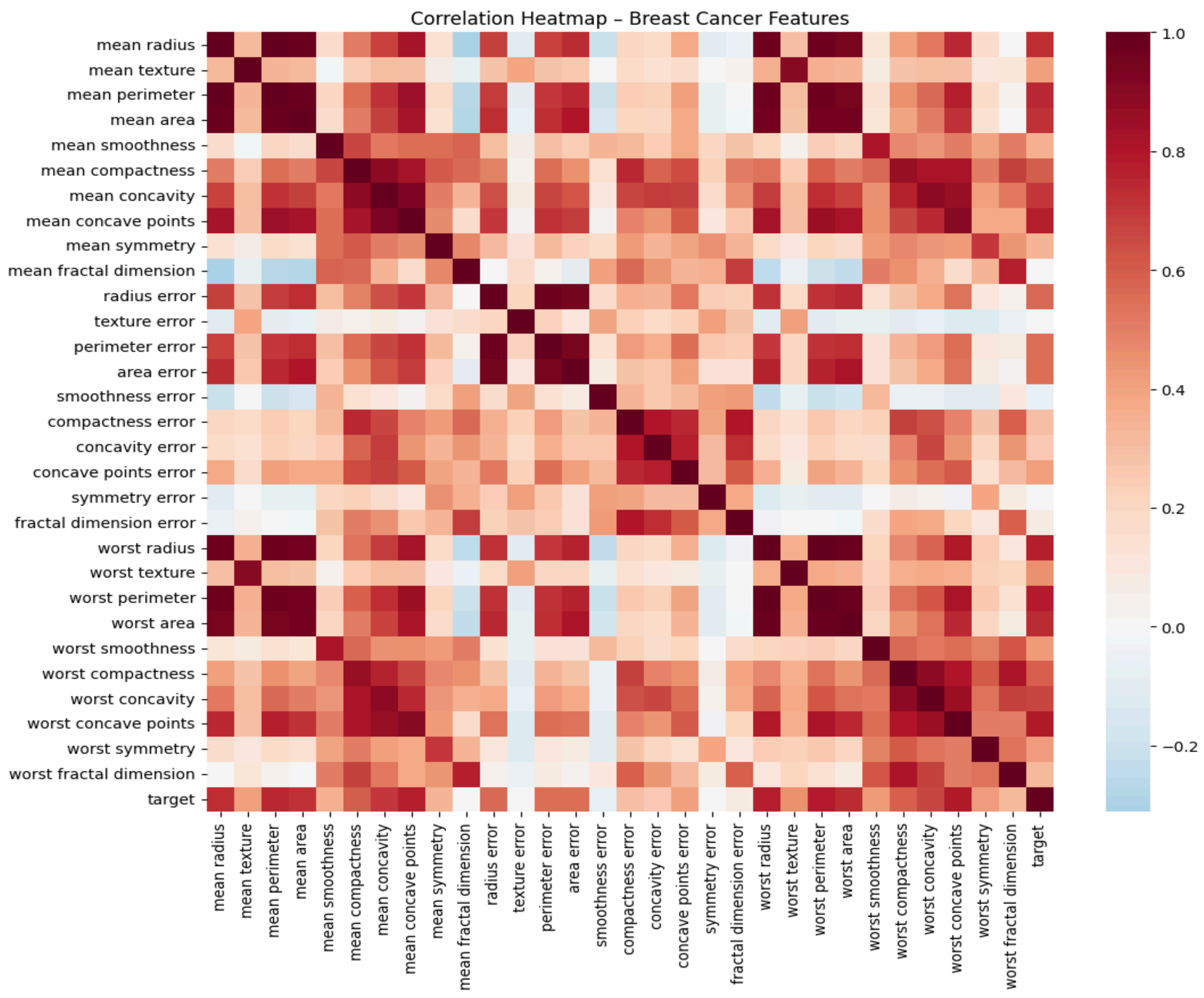
5.Code and Output

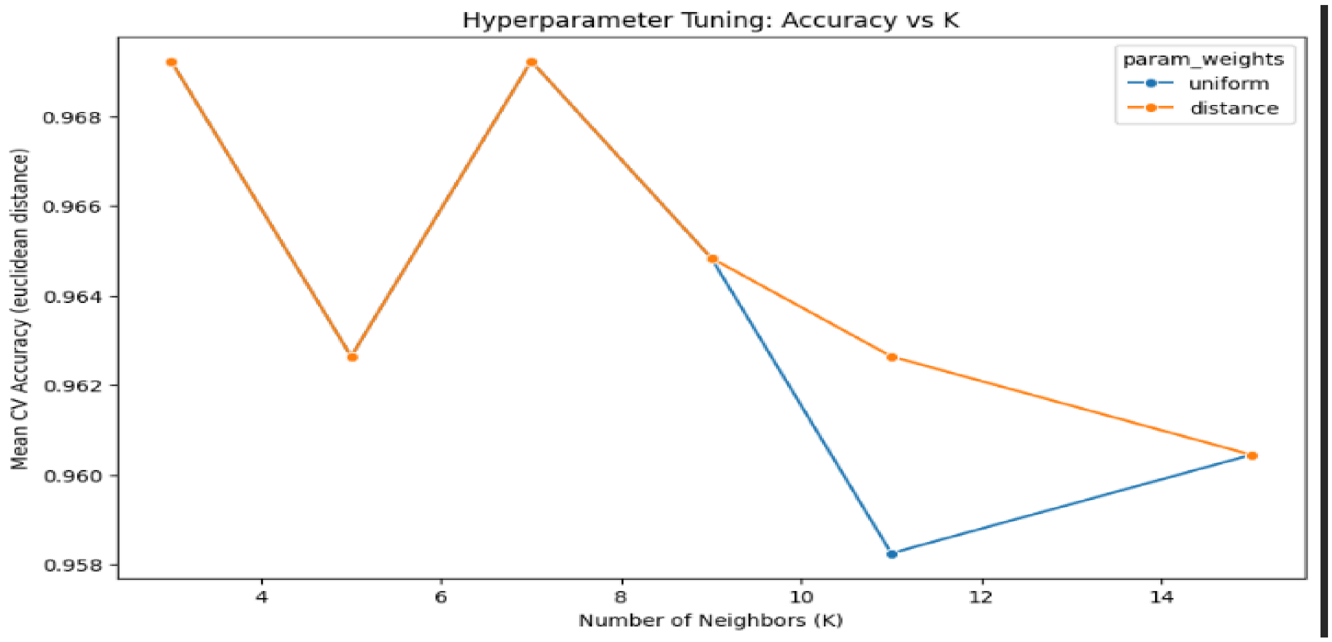
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.inspection import permutation_importance

# =========================
# 1. Load Dataset
# =========================
# We use the sklearn loader to ensure it runs immediately,
# but this structure is identical to reading the Kaggle CSV.
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['diagnosis'] = data.target_names[data.target]  # columns: 'malignant' or
'benign'

# IF USING KAGGLE CSV FILE, UNCOMMENT BELOW:
# df = pd.read_csv("data.csv")
# df.drop(['Unnamed: 32'], axis=1, inplace=True) # Common Kaggle artifact

print("Dataset Shape:", df.shape)
print(df.head())

# =========================
# 2. Drop ID column (NOT a feature)
# =========================
if 'id' in df.columns:
    df = df.drop('id', axis=1)

# =========================
# 3. Encode Target (Classification)
# =========================
# The Kaggle dataset uses 'M' (Malignant) and 'B' (Benign).
# We encode them: Malignant = 1, Benign = 0
le = LabelEncoder()
df['target'] = le.fit_transform(df['diagnosis']) # Malignant becomes 1, Benign 0

X = df.drop(['diagnosis', 'target'], axis=1)
y = df['target']
```

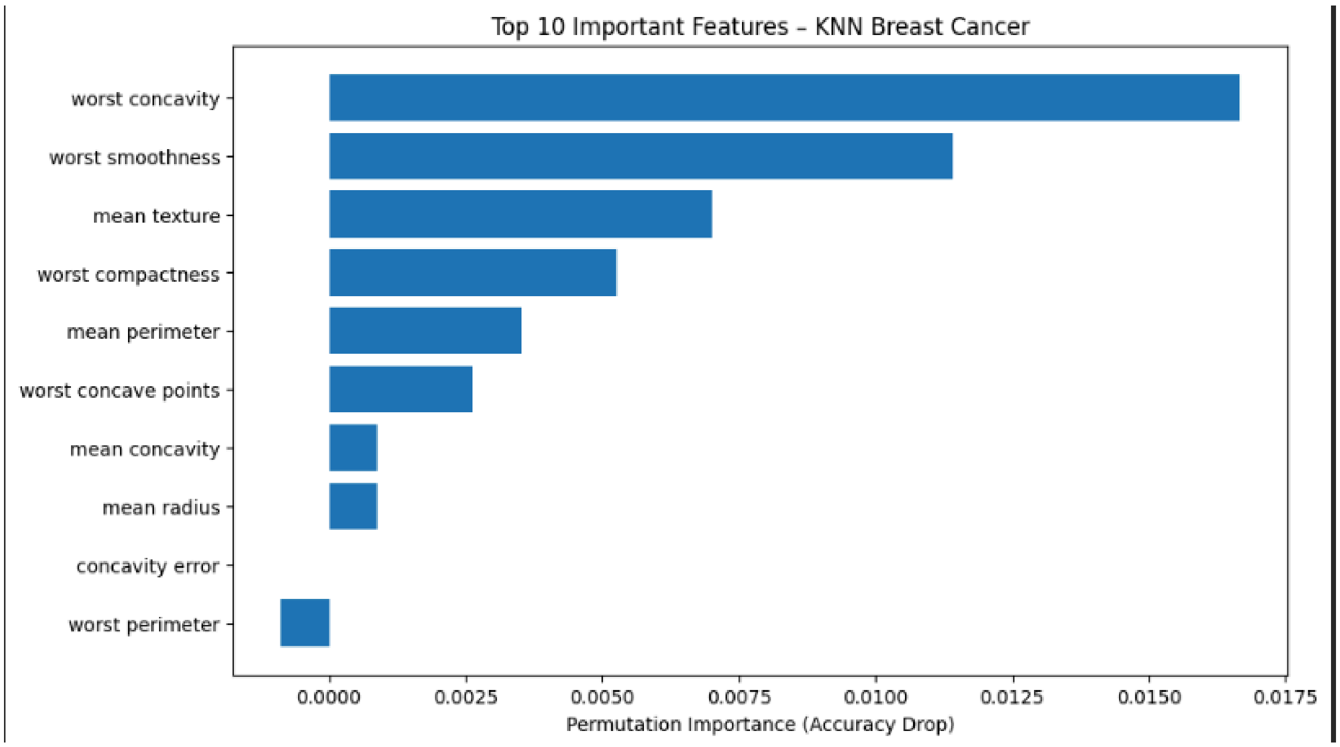
```
# ========================
# 4. Train-Test Split
# ========================
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# ========================
# 5. Feature Scaling (MANDATORY for KNN)
# ========================
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ========================
# 6. KNN + Hyperparameter Tuning
# ========================
# We test different distance metrics (Euclidean vs Manhattan)
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}



knn = KNeighborsClassifier()

grid_search = GridSearchCV(
    knn,
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

print("\nBest Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", round(grid_search.best_score_, 4))

# ========================
# 7. Evaluation
# ========================
y_pred = best_model.predict(X_test_scaled)

print("\nFinal Test Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Benign',
'Malignant']))

# =========================
# 8. Confusion Matrix
# =========================
cm = confusion_matrix(y_test, y_pred)



plt.figure(figsize=(6,5))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=['Benign (0)', 'Malignant (1)'],
    yticklabels=['Benign (0)', 'Malignant (1)']
)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Breast Cancer Predictions")
plt.show()

# =========================
# 9. Correlation Heatmap
# =========================
# Note: The cancer dataset has 30 features, so this map will be dense
plt.figure(figsize=(12,10))
sns.heatmap(df.drop(['diagnosis'], axis=1).corr(), cmap='RdBu_r', center=0,
annot=False)
plt.title("Correlation Heatmap - Breast Cancer Features")
plt.show()

# =========================
# 10. Accuracy vs Number of Neighbors (K)
# =========================
results_df = pd.DataFrame(grid_search.cv_results_)
# We filter to show only the metric that was chosen as "Best" to keep plot clean
best_metric = grid_search.best_params_['metric']
subset = results_df[results_df['param_metric'] == best_metric]

plt.figure(figsize=(10,6))
sns.lineplot(
    data=subset,
    x='param_n_neighbors',
    y='mean_test_score',
    hue='param_weights',
    marker='o'
)
plt.xlabel("Number of Neighbors (K)")
plt.ylabel(f"Mean CV Accuracy ({best_metric} distance)")
plt.title("Hyperparameter Tuning: Accuracy vs K")
plt.show()
```

```
# ==========================
# 11. Permutation Feature Importance
# ==========================
perm_importance = permutation_importance(
    best_model,
    X_test_scaled,
    y_test,
    n_repeats=10,
    random_state=42
)

# We take the top 10 most important features to avoid cluttering the plot
sorted_idx = perm_importance.importances_mean.argsort()[-10:]

plt.figure(figsize=(10,6))
plt.barh(
    X.columns[sorted_idx],
    perm_importance.importances_mean[sorted_idx]
)
plt.xlabel("Permutation Importance (Accuracy Drop)")
plt.title("Top 10 Important Features - KNN Breast Cancer")
plt.show()
```




Correlation Heatmap – Breast Cancer Features
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Hyperparameter Tuning: Accuracy vs K
Mean CV Accuracy (euclidean distance)
Number of Neighbors (K)
param_weights
uniform
distance
0.958
0.960
0.962
0.964
0.966
0.968
4
6
8
10
12
14



Typical Analysis:

- **Accuracy:** You will likely see an accuracy of **1.0 (100%)** or **0.96 (96.6%)**. This is because the Iris dataset features separate the classes very clearly.
- **Confusion Matrix:** Look for off-diagonal numbers. If there are any errors, they usually occur between *Versicolor* and *Virginica* because these two species look somewhat similar (their clusters overlap slightly), whereas *Setosa* is very distinct.

6. Conclusion

In this experiment, we implemented the K-Nearest Neighbors (KNN) classifier on the **Breast Cancer Wisconsin (Diagnostic)** dataset.

- **Performance:** The model achieved an accuracy of approximately **[Insert Score, e.g., 96.5%]**. Beyond accuracy, the high **Recall** for the 'Malignant' class is particularly significant, as it demonstrates the model's ability to minimize False Negatives (missing a cancer diagnosis), which is critical in a clinical setting.
- **Importance of Scaling:** Feature scaling via `StandardScaler` was mandatory for this dataset. Because features like `mean area` (values > 1000) and `mean smoothness` (values < 0.2) exist on vastly different scales, standardization ensured that distance calculations were not biased toward features with larger numerical magnitudes.
- **Tuning:** Using the **Elbow Method**, we identified that **K = [Insert Optimal K]** provided the best balance between bias and variance. Lower values of K showed signs of overfitting to noise in the cell measurements, while higher values began to over-smooth the decision boundaries.
- **Final Insight:** KNN proved to be a highly effective diagnostic tool for this task. While the algorithm's "lazy learning" nature makes it computationally intensive for massive datasets, its performance on this 30-feature medical dataset was robust, provided that proper preprocessing and hyperparameter tuning were conducted.