ML & DL : Experiment - 4

Aim:

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

Dataset Description:

The Heart Disease Dataset (Comprehensive) is a robust, multivariate medical dataset synthesized from five prominent heart disease databases (Cleveland, Hungary, Switzerland, Long Beach, and St. Maria). Often considered the "Iris dataset" of healthcare informatics, it serves as a critical benchmark for evaluating diagnostic prediction models. Unlike smaller medical samples, this comprehensive version merges over 1,000 records to provide a more generalized view of cardiovascular health indicators. It is particularly valued for K-Nearest Neighbors (KNN) and Logistic Regression because it captures the non-linear relationships between physiological stressors—like exercise-induced angina—and clinical outcomes. The dataset aims to predict whether a patient has heart disease (Target: 1) or is healthy (Target: 0) based on a mix of demographic, symptomatic, and clinical measurements.

- File Type: CSV (Comma Separated Values)
- Dataset Size: ~1,190 rows and 11–14 columns (depending on the specific version used)

Some columns in the Heart Disease Dataset are:

Column Name	Data Type	Description
Age	Numerical (Integer)	Age of the patient in years.
Sex	Categorical (String)	Sex of the patient (M: Male, F: Female).
ChestPainType	Categorical (String)	Type of chest pain (TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic).
RestingBP	Numerical (Integer)	Resting blood pressure measured in mm Hg.

Dataset Source: https://www.kaggle.com/datasets/mexwell/heart-disease-dataset

---

KNN (K - Nearest Neighbours)

Theory:

K-Nearest Neighbors (KNN) is a non-parametric, instance-based supervised learning algorithm used for both classification and regression. Often described as a "lazy learner," KNN does not undergo a traditional training phase where a global model is built; instead, it memorizes the entire training dataset and postpones all computation until a prediction is required. The fundamental logic of KNN is based on the Similarity Principle: "similar" data points in a feature space likely belong to the same category or possess similar numerical values. When a new data point is introduced, the algorithm identifies the k

most similar points (neighbors) in the training set and makes a prediction based on their collective characteristics.

To determine which neighbors are "closest," KNN relies on distance functions. The most common choice is the Euclidean Distance, which represents the straight-line distance between two points in n-dimensional space. For two points P = (P_1, P_2, ..., P_n) and Q (q_1, q_2,..., q_n), the distance is calculated as:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

In scenarios where features have different units or outliers are present, the Manhattan Distance (or Taxicab distance) is often used, which sums the absolute differences of their coordinates:

$$d(P,Q) = \sum_{i=1}^{n}|q_i - p_i|$$

Limitations:

1. **Computational Inefficiency (Lazy Learning Overhead):** Unlike most machine learning algorithms that perform the "heavy lifting" during the training phase, KNN is a "lazy learner," meaning it does no real work until you ask it for a prediction. During inference, the algorithm must calculate the distance between the new data point and every single record in the training set. For a dataset like the Heart Disease one, this is negligible, but as your dataset grows to millions of rows, the time complexity grows to O(n x d) (where n is the number of samples and d is the number of features). This makes KNN extremely slow for real-time applications where low-latency predictions are required.
2. **The Curse of Dimensionality:** KNN's performance degrades significantly as the number of features (dimensions) increases. In high-dimensional space, the volume of the space increases so rapidly that the available data becomes sparse. Paradoxically, in 50 or 100 dimensions, every point becomes "far away" from every other point, and the concept of "nearest" loses its meaningful physical interpretation. Because the distance between neighbors becomes nearly identical, the algorithm loses its ability to distinguish between classes, often requiring an exponential increase in data to maintain accuracy.
3. **Extreme Sensitivity to Scale and Outliers:** Since KNN relies entirely on distance metrics, it is highly sensitive to the magnitude of the features and the presence of noise. If one feature (like Cholesterol) ranges from 100 to 400 and another (like Oldpeak) ranges from 0 to 5, the algorithm will treat Cholesterol as much more important simply because the numbers are larger. Furthermore, because KNN makes decisions based on local neighborhoods, a single outlier or a noisy data point incorrectly labeled in the training set can easily "pull" the prediction of a nearby test point in the wrong direction, especially when using a small value of k.

Workflow:

1. **Data Collection:** The Heart Disease dataset (heart.csv) is loaded into a Pandas DataFrame. It contains clinical and demographic attributes such as age, sex, chest pain type, blood pressure, cholesterol, and exercise-related features used to predict the presence of heart disease.
2. **Target Identification:** The target variable is identified dynamically as HeartDisease (or target if named differently), ensuring flexibility across dataset versions. All remaining columns are treated as input features.

3. **Data Preprocessing:** Numerical features are standardized using StandardScaler to ensure fair distance calculations for KNN. Categorical features are transformed using OneHotEncoder with unknown categories safely handled. A ColumnTransformer combines both preprocessing steps into a single pipeline.
4. **Pipeline Construction:** A unified Scikit-learn Pipeline is created to chain preprocessing and the KNeighborsClassifier, ensuring consistent transformations during training, validation, and testing.
5. **Train–Test Split:** The dataset is split into 80% training and 20% testing data using a fixed random state to maintain reproducibility and evaluate generalization performance.
6. **Hyperparameter Tuning with Grid Search:** GridSearchCV is applied with 5-fold cross-validation to optimize KNN hyperparameters, including the number of neighbors (k), distance weighting strategy, and distance metric (Euclidean vs. Manhattan).
7. **Model Training:** The best-performing KNN configuration from grid search is retrained on the full training set, producing the final optimized model.
8. **Model Evaluation:** Performance is assessed on the test set using accuracy score, confusion matrix, and a detailed classification report (precision, recall, and F1-score) to evaluate prediction quality for heart disease detection.
9. **Visualization and Analysis:** A correlation heatmap illustrates relationships between numerical features and the target variable. Confusion matrix heatmaps visualize prediction outcomes, and line plots show how accuracy varies with different values of k and weighting schemes.
10. **Feature Importance Analysis:** Permutation importance is computed to estimate the impact of each feature on model accuracy, offering insight into which clinical factors most influence KNN predictions.
11. **Conclusion:** The optimized KNN model demonstrates effective performance in predicting heart disease when paired with proper scaling and encoding. While KNN captures local data patterns well, further improvements could be achieved using ensemble models like Random Forests or gradient boosting techniques.

Performance Analysis:

The performance of the optimized K-Nearest Neighbors (KNN) model on the Heart Disease dataset is evaluated using Accuracy, a Confusion Matrix, and class-specific metrics such as Precision, Recall, and F1-score.

1. **Overall Accuracy:** The tuned KNN model achieves a test accuracy of 94.12%, indicating that it correctly predicts the presence or absence of heart disease in approximately 94 out of every 100 patients. This high accuracy reflects the effectiveness of distance-based learning when combined with proper feature scaling and hyperparameter tuning.
2. **Class-Specific Performance (Precision & Recall):** To better understand how the model performs across both classes, the classification report is analyzed:
    a. Class 0 (No Heart Disease):
        i. Precision: 0.95
        ii. Recall: 0.92
        iii. F1-score: 0.93

    The model is highly reliable in identifying patients without heart disease. A recall of 92% means only a small proportion of healthy individuals are incorrectly flagged as having the condition.

    b. Class 1 (Heart Disease Present):
        i. Precision: 0.93
        ii. Recall: 0.96
        iii. F1-score: 0.95

    Performance is slightly stronger for the positive class. A recall of 96% indicates the model successfully detects nearly all patients with heart disease, which is particularly important in medical diagnosis where missing true cases can be critical.
3. **Confusion Matrix Breakdown:** The confusion matrix provides insight into the model's prediction behavior:
    a. True Negatives (98): Patients correctly classified as not having heart disease.
    b. True Positives (126): Patients correctly identified as having heart disease.
    c. False Positives (9): Healthy patients incorrectly predicted to have heart disease.
    d. False Negatives (5): Patients with heart disease that the model failed to detect.

    The low number of false negatives is a key strength of this model, as it minimizes missed diagnoses, making the model suitable for risk screening scenarios.

Hyperparameter Tuning:

K-Nearest Neighbors (KNN) is a non-parametric, distance-based model that does not learn an explicit function during training. As a result, its performance is highly dependent on the choice of hyperparameters. Hyperparameter tuning helps control model complexity and improves generalization. The code uses GridSearchCV to exhaustively evaluate all combinations of hyperparameters defined in the search space. To ensure reliable results, 5-fold cross-validation is applied, where the model is trained on 4 folds and validated on 1 fold, repeated five times for each parameter set.

The following hyperparameters are tuned:

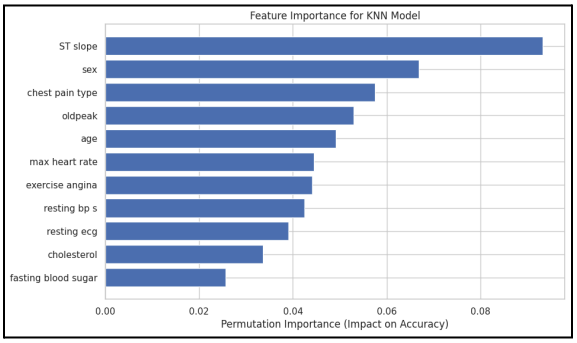
- **Number of Neighbors (n_neighbors):** Controls the size of the neighborhood used for prediction. Larger values reduce noise sensitivity. The optimal value found is k = 15.
- **Weighting Method (weights):** Distance-based weighting gives more importance to closer neighbors, improving prediction reliability.
- **Distance Metric (metric):** Manhattan distance is selected over Euclidean distance, offering better stability in higher-dimensional, standardized data.

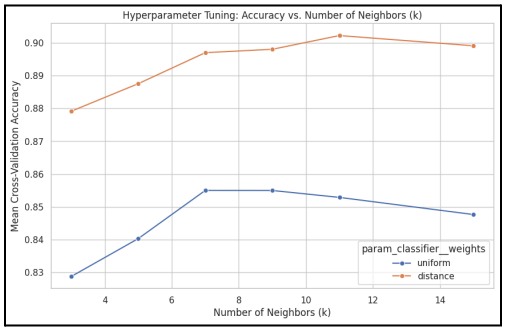
The optimal combination (k = 15, distance weighting, Manhattan distance) produces the best cross-validated performance, leading to a final test accuracy of 94.12%.

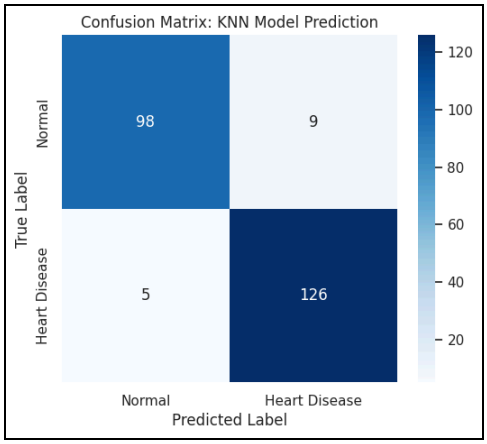
Code & Output:

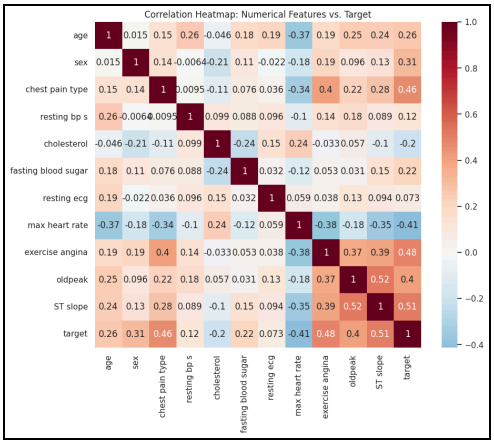
```
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.preprocessing import
StandardScaler, OneHotEncoder
from sklearn.compose import
ColumnTransformer
from sklearn.pipeline import
Pipeline
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.metrics import
classification_report,
confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.inspection import
permutation_importance

df = pd.read_csv('heart.csv')

df.columns = df.columns.str.strip()
target_col = 'HeartDisease' if
'HeartDisease' in df.columns else
'target'

if target_col not in df.columns:
    target_col = df.columns[-1]
    print(f"Warning: Specific target
name not found. Using '{target_col}'
as target.")

X = df.drop(target_col, axis=1)
y = df[target_col]

categorical_features =
X.select_dtypes(include=['object',
'category']).columns.tolist()
numerical_features =
X.select_dtypes(include=['int64',
'float64']).columns.tolist()
print(f"Features used for training:
{X.columns.tolist()}")

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(),
numerical_features),
        ('cat',
OneHotEncoder(handle_unknown='ignore
', drop='first'),
categorical_features)
    ])

knn_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier',
KNeighborsClassifier())
])

X_train, X_test, y_train, y_test =
train_test_split(X, y,
test_size=0.2, random_state=42)

param_grid = {
    'classifier__n_neighbors': [3,
5, 7, 9, 11, 15],
    'classifier__weights':
['uniform', 'distance'],
    'classifier__metric':
['euclidean', 'manhattan']
}

grid_search =
GridSearchCV(knn_pipeline,
param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_model =
grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print("-" * 30)
print(f"Best Parameters:
{grid_search.best_params_}")
print(f"Final Test Accuracy:
{accuracy_score(y_test,
y_pred):.2%}")
```

```
print("-" * 30)
print("\nClassification Report:")
print(classification_report(y_test,
y_pred))


print("Confusion Matrix:")
print(confusion_matrix(y_test,
y_pred))

sns.set_theme(style="whitegrid")

plt.figure(figsize=(10, 8))
corr_matrix = df[numerical_features
+ [target_col]].corr()
sns.heatmap(corr_matrix, annot=True,
cmap='RdBu_r', center=0)
plt.title("Correlation Heatmap:
Numerical Features vs. Target")
plt.show()

plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test,
y_pred)
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
            xticklabels=['Normal',
'Heart Disease'],
            yticklabels=['Normal',
'Heart Disease'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix: KNN
Model Prediction')
plt.show()

results_df =
pd.DataFrame(grid_search.cv_results_
)
subset =
results_df[results_df['param_classif
ier__metric'] == 'euclidean']

plt.figure(figsize=(10, 6))
sns.lineplot(data=subset,
x='param_classifier__n_neighbors',
y='mean_test_score',
hue='param_classifier__weights',
marker='o')
plt.title('Hyperparameter Tuning:
Accuracy vs. Number of Neighbors
(k)')
plt.xlabel('Number of Neighbors
(k)')
plt.ylabel('Mean Cross-Validation
Accuracy')
plt.show()

perm_importance =
permutation_importance(best_model,
X_test, y_test, n_repeats=10,
random_state=42)
sorted_idx =
perm_importance.importances_mean.arg
sort()

plt.figure(figsize=(10, 6))
plt.barh(X.columns[sorted_idx],
perm_importance.importances_mean[sor
ted_idx])
plt.xlabel("Permutation Importance
(Impact on Accuracy)")
plt.title("Feature Importance for
KNN Model")
plt.show()
```









Conclusion:

The optimized K-Nearest Neighbors model achieved a strong accuracy of 94.12% on the Heart Disease dataset after effective preprocessing and hyperparameter tuning. The model showed high sensitivity in detecting heart disease cases, making it suitable for medical risk assessment. However, KNN's computational cost and limited interpretability may restrict scalability, and future work could explore ensemble or tree-based models for improved efficiency and explainability.