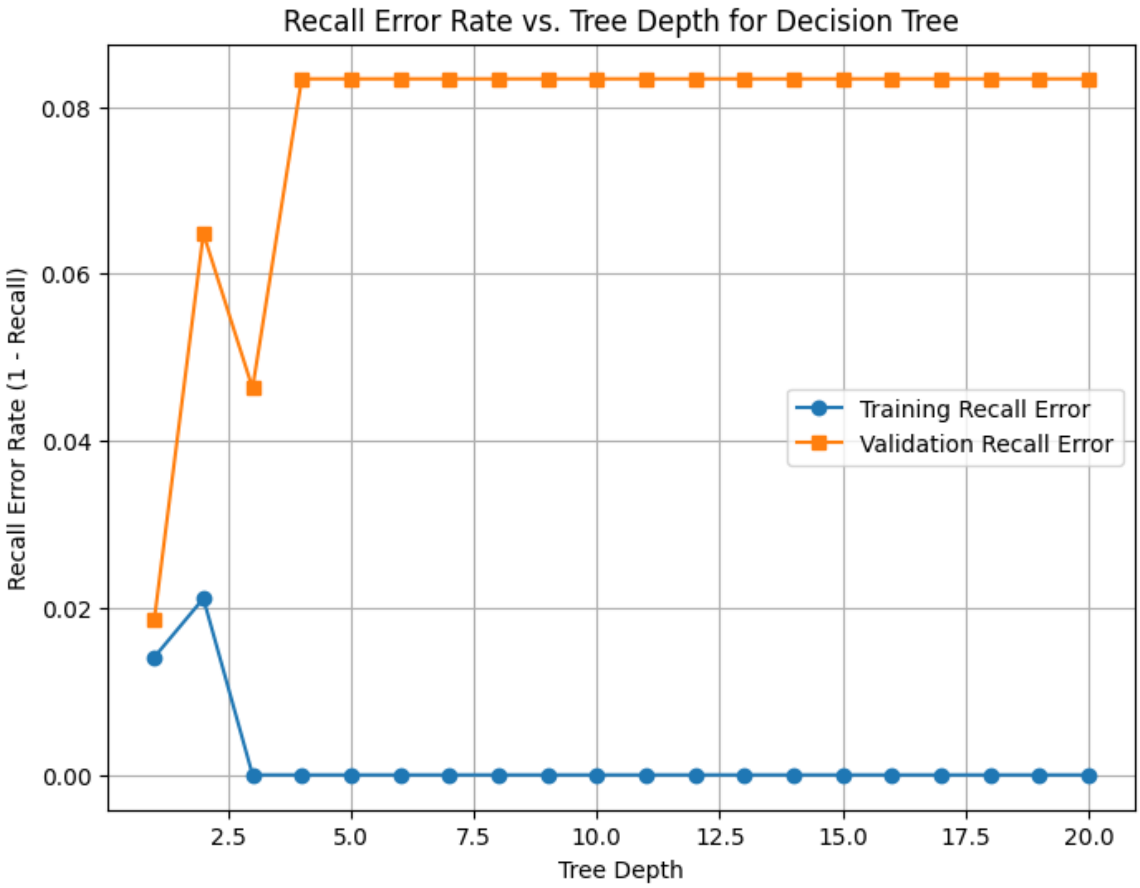
Decision Tree

In [142…

```
# Re-import necessary libraries after execution state reset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, recall_score
from sklearn.datasets import load_breast_cancer

# Load the Wisconsin Breast Cancer dataset from sklearn
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)  # Target variable (0 = malignant, 1 = b

# Split data into training (60%), validation (20%), and testing (20
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size

# Split temp further into validation (50% of temp) and testing (50%
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, tes

# Experiment with different depths and track recall error rates
depths = range(1, 21)  # Try depths from 1 to 20
train_recall_errors = []
val_recall_errors = []

for depth in depths:
    # Train Decision Tree model
    model = DecisionTreeClassifier(max_depth=depth, random_state=42
    model.fit(X_train, y_train)

    # Compute recall scores
    train_recall = recall_score(y_train, model.predict(X_train))
    val_recall = recall_score(y_val, model.predict(X_val))

    # Compute recall error (1 - recall)
    train_recall_errors.append(1 - train_recall)
    val_recall_errors.append(1 - val_recall)

# Plot recall error rate vs. tree depth
plt.figure(figsize=(8, 6))
plt.plot(depths, train_recall_errors, marker='o', label='Training R
plt.plot(depths, val_recall_errors, marker='s', label='Validation R
plt.xlabel('Tree Depth')
plt.ylabel('Recall Error Rate (1 - Recall)')
plt.title('Recall Error Rate vs. Tree Depth for Decision Tree')
plt.legend()
plt.grid(True)
plt.show()
```



Observations:

Underfitting :

- There does not appear to be an issue of underfitting, as both training and validation recall error rates remain below 10%, which is relatively low.

Overfitting at Higher Depths (Depth >= 4):

- There is a significant disparity between the training and validation recall error rates, beginning at depth = 4. This gap remains consistently high, with the training recall error dropping close to 0%, indicating that the model is overly memorizing the training data. As a result, the model fails to generalize well to unseen data, leading to severe overfitting.

Lack of an Optimal Depth:

- Unlike an ideal model where the validation recall error should decrease and then stabilize, here, it stays high throughout.
- This suggests that the decision tree may not be capturing the malignant class effectively regardless of tree depth.

Conclusion:

- The decision tree is highly overfitting after depth = 4, as indicated by zero training recall error while the validation recall error remains high.

In [143…

```
# Import necessary library for classification report
from sklearn.metrics import classification_report

# Train Decision Tree model with the optimal depth
optimal_depth = 3  # Based on error rate analysis
final_model = DecisionTreeClassifier(max_depth=optimal_depth, rando
final_model.fit(X_train, y_train)  # Train on the training set

# Evaluate on test data (separate test set now available)
y_pred_test = final_model.predict(X_test)

# Generate classification report for test data
report_test = classification_report(y_test, y_pred_test, target_nam

# Print the classification report
print("\n**Decision Tree Model Performance on Test Set (depth=3)**"
print(report_test)
```

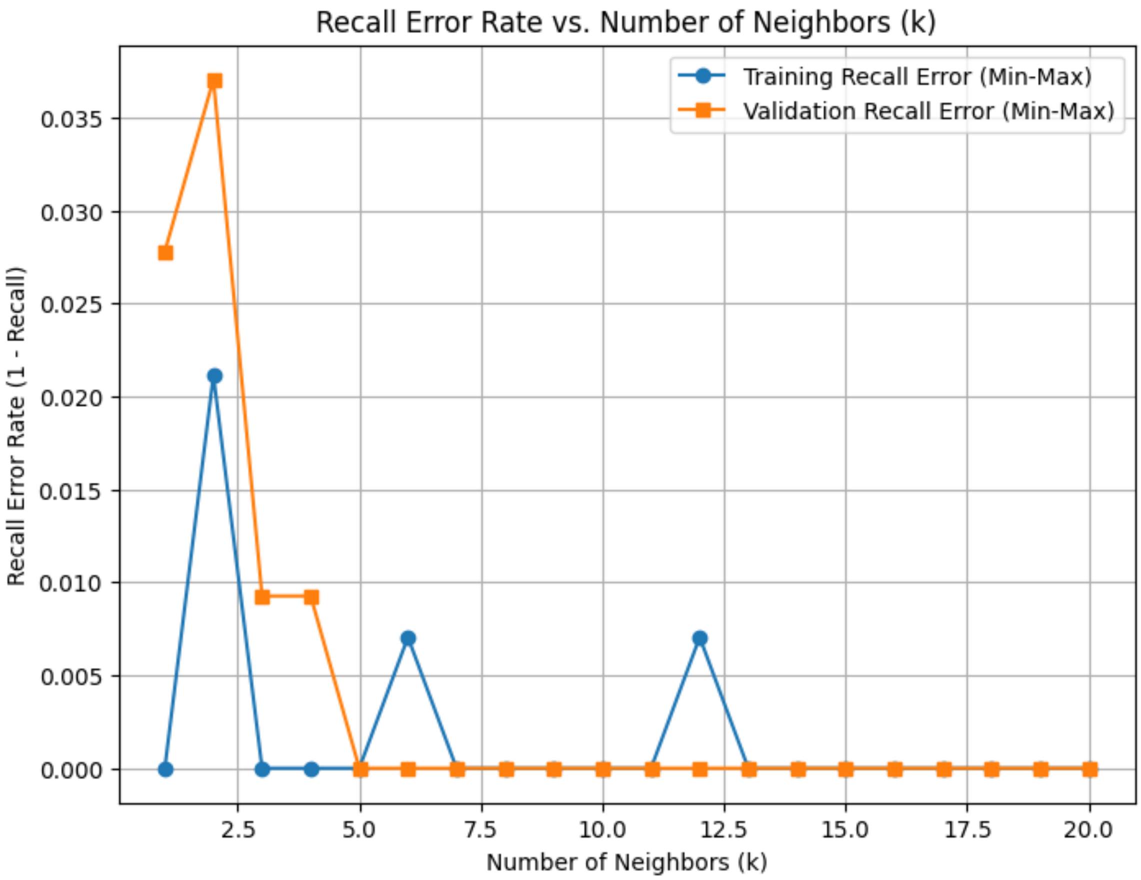
```
**Decision Tree Model Performance on Test Set (depth=3)**
              precision    recall  f1-score   support

   Malignant       0.85      0.88      0.86        64
      Benign       0.92      0.91      0.92       107

    accuracy                           0.89       171
   macro avg       0.89      0.89      0.89       171
weighted avg       0.90      0.89      0.90       171
```

K-Nearest Neighbors

In [144…

```
# Re-import necessary libraries after execution state reset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import recall_score
from sklearn.datasets import load_breast_cancer

# Load the Wisconsin Breast Cancer dataset from sklearn
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)  # Target variable (0 = malignant, 1 = b

# Ensure proper train-validation-test split before normalization
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, tes

# Apply Min-Max Scaling before training and testing
scaler = MinMaxScaler()
X_train_minmax = scaler.fit_transform(X_train)
X_val_minmax = scaler.transform(X_val)
```

```python
X_test_minmax = scaler.transform(X_test)

# Experiment with different values of k for KNN after Min-Max Norma
k_values = range(1, 21)  # Trying k from 1 to 20
train_recall_errors_knn = []
val_recall_errors_knn = []

for k in k_values:
    # Train KNN model on Min-Max normalized data
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train_minmax, y_train)

    # Compute training and validation recall
    train_recall = recall_score(y_train, knn_model.predict(X_train_
    val_recall = recall_score(y_val, knn_model.predict(X_val_minmax

    # Compute recall error (1 - recall)
    train_recall_errors_knn.append(1 - train_recall)
    val_recall_errors_knn.append(1 - val_recall)

# Plot recall error rate vs. number of neighbors (k) after Min-Max 
plt.figure(figsize=(8, 6))
plt.plot(k_values, train_recall_errors_knn, marker='o', label='Trai
plt.plot(k_values, val_recall_errors_knn, marker='s', label='Valida
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Recall Error Rate (1 - Recall)')
plt.title('Recall Error Rate vs. Number of Neighbors (k)')
plt.legend()
plt.grid(True)
plt.show()
```



Observations:

Underfitting at Low k Values (k = 1-3):

- The validation recall error is high and fluctuates significantly.
- This suggests that the model is too sensitive to individual data points, leading to high variance and misclassification of malignant cases.

Optimal k (Balanced Performance at k = 4-6):

- The validation recall error stabilizes at a low value, while training recall error remains low.
- This indicates strong generalization, effectively capturing malignant cases without overfitting.

Overfitting at Higher k Values (k > 6):

- Training recall error remains at zero, meaning the model is memorizing training data.
- Validation recall error remains low, showing the model is still performing well.

Conclusion:

- The optimal k value lies between 4 and 6, where the validation recall error is at its lowest, ensuring good generalization without overfitting.
- k < 3 leads to high variance, making the model unstable for real-world predictions.
- k > 10 shows signs of slight underfitting, where the model might fail to capture subtle malignant patterns.

In [145…

```
# Combine Training and Validation Data for Final Training
X_final_train = np.vstack((X_train_minmax, X_val_minmax))
y_final_train = np.hstack((y_train, y_val))

# Train KNN with the previously determined optimal k (chosen from v
final_knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

# Train on Min-Max scaled combined training + validation data
final_knn.fit(X_final_train, y_final_train)

# Predict on test set
y_pred_knn_test = final_knn.predict(X_test_minmax)

# Generate classification report for KNN after Min-Max Normalizatio
from sklearn.metrics import classification_report

report_knn_test = classification_report(y_test, y_pred_knn_test, ta

# Print the classification report
```

```
print("\n**KNN Model Performance on Test Set (k=5, Euclidean Distan
print(report_knn_test)
```

```
**KNN Model Performance on Test Set (k=5, Euclidean Distance)**
              precision    recall  f1-score   support

   Malignant       0.98      0.91      0.94        64
      Benign       0.95      0.99      0.97       107

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Logistic Regression

In [146…

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_breast_cancer

# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Ensure proper train-validation-test split before normalization
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, tes

# Apply Min-Max Scaling (fit on training set, transform on validati
scaler = MinMaxScaler()
X_train_minmax = scaler.fit_transform(X_train)
X_val_minmax = scaler.transform(X_val)
X_test_minmax = scaler.transform(X_test)

# Define C values for iteration
C_values = [100, 10, 1, 0.1]

# Store recall error rates
recall_errors_l1 = []
recall_errors_l2 = []

# Iterate through each C value and train both L1 and L2 regularizat
for C in C_values:
    clf_l1 = linear_model.LogisticRegression(C=C, penalty='l1', tol
    clf_l2 = linear_model.LogisticRegression(C=C, penalty='l2', tol

    # Train the Models on Min-Max Normalized Training Data
    clf_l1.fit(X_train_minmax, y_train)
    clf_l2.fit(X_train_minmax, y_train)
```

```
    # Predictions on Validation Set
    y_val_pred_l1 = clf_l1.predict(X_val_minmax)
    y_val_pred_l2 = clf_l2.predict(X_val_minmax)

    # Compute Classification Reports
    report_l1 = classification_report(y_val, y_val_pred_l1, target_
    report_l2 = classification_report(y_val, y_val_pred_l2, target_

    recall_l1 = report_l1["Malignant"]["recall"]
    recall_l2 = report_l2["Malignant"]["recall"]

    # Store recall error rates (1 - recall)
    recall_errors_l1.append(1 - recall_l1)
    recall_errors_l2.append(1 - recall_l2)

# Plot Recall Error Rate vs. Regularization Strength (C)
plt.figure(figsize=(8, 6))
plt.plot(C_values, recall_errors_l1, marker='o', linestyle='-', col
plt.plot(C_values, recall_errors_l2, marker='s', linestyle='-', col
plt.xscale("log")  # Log scale for better visualization
plt.xlabel("Regularization Strength (C)")
plt.ylabel("Recall Error Rate (1 - Recall)")
plt.title("Logistic Regression: Recall Error Rate vs. Regularizatio
plt.legend()
plt.grid(True)
plt.show()
```

Observations:

- For L1 (Lasso) regularization (blue line), recall error decreases as C

increases, reaching zero at C=1.
- For L2 (Ridge) regularization (red line), recall error fluctuates, peaking at C=1 before dropping back to zero for larger C values.

L1 vs. L2 Regularization Performance:

- L1 (Lasso) maintains a decreasing recall error rate, indicating that it consistently improves as regularization is relaxed.
- L2 (Ridge) shows instability at C=1, with a sudden increase in recall error before stabilizing at C=10.

Conclusion:

- L1 regularization (Lasso) with C=1 appears to be the best choice, as it achieves a recall error of zero.
- L2 regularization (Ridge) may not be ideal due to its fluctuation in recall error, making it less reliable for recall-sensitive applications.

In [147…

```
from sklearn import linear_model
from sklearn.metrics import classification_report, confusion_matrix

# Define the best model (based on previous analysis, best C = 1, L1
best_model = linear_model.LogisticRegression(C=1, penalty='l1', tol

# Train on the full training set (train + validation)
X_full_train_minmax = np.concatenate((X_train_minmax, X_val_minmax)
y_full_train = np.concatenate((y_train, y_val), axis=0)
best_model.fit(X_full_train_minmax, y_full_train)

# Predictions on Test Set (Fixing Variable Name)
y_test_pred = best_model.predict(X_test_minmax)  # Fixed variable n

# Print Classification Report
print("\n**Final Model Performance on Test Set**")
print(classification_report(y_test, y_test_pred, target_names=["Ben
```

```
**Final Model Performance on Test Set**
              precision    recall  f1-score   support

      Benign       0.97      0.88      0.93        42
   Malignant       0.93      0.99      0.96        72

    accuracy                           0.95       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Final Conclusion:

In medical diagnosis, particularly in detecting malignant (cancerous) tumors, recall is critical because false negatives are very dangerous

If a malignant tumor is misclassified as benign, the patient won't receive

necessary treatment, leading to severe consequences.

Logistic Regression has 99% recall for malignant cases (no false negatives), while Decision Tree has 88% and KNN has 91%. Therefore, Logistic Regression outperforms the other models in this case.

In [ ]:

```
```