Breast Cancer Recurrence Prediction using Machine Learning

GITHUB LINK:

mohansasank/Breast-Cancer-Recurrence-Prediction-using-Machine-Learning- (github.com)

PROBLEM STATEMENT:

Predicting the recurrence of breast cancer in women by analyzing various factors that can influence the likelihood of the cancer returning. The factors like progesterone and estrogen levels, size, and nodes of the tumor - are indeed crucial elements in assessing the risk of recurrence. In this project, we experiment with multiple machine learning models in search of a model that best fits the data and has high accuracy.

DATASET:

The data set contains patient records from a 1984-1989 trial conducted by the German Breast Cancer Study Group (GBSG) of 720 patients with node positive breast cancer; it retains the 686 patients with complete data for the prognostic variables. These data sets are used in the paper by Royston and Altman (2013). The Rotterdam data is used to create a fitted model, and the GBSG data for validation of the model. The paper gives references for the data source.

The Rotterdam dataset contains information about patients with node-positive breast cancer. It includes variables such as hormone receptor status, tumor size, lymph node involvement, histological grade, HER2 status, Ki-67 index, and treatment response, among others. The GBSG dataset serves as a validation dataset for the model developed using the Rotterdam dataset. In this context, the GBSG dataset would include similar prognostic variables for patients with nodepositive breast cancer, and it is likely that this dataset was not used in the training phase of the model.

In summary, the paper by Royston and Altman follows a common approach in predictive modeling by using one dataset for model development (Rotterdam) and another for validation (GBSG).

FEATURES:

- **pid (patient identifier):**
  This column likely serves as a unique identifier for each patient in the dataset. It is used to distinguish between individual records. • **age (age, years):**

Represents the age of the patient in years at the time of data collection. Age is often a significant factor in cancer prognosis.

- **meno (menopausal status):**
  A binary variable indicating the menopausal status of the patient.
  0 = premenopausal 1 = postmenopausal
- **size (tumor size, mm):**
  Provides information about the size of the tumor in millimeters. grade (tumor grade):
  Describes the grade or level of differentiation of the tumor cells
- **nodes (number of positive lymph nodes):**
  Indicates the number of lymph nodes with cancer involvement. •

pgr (progesterone receptors):
Represents the concentration of progesterone receptors in femtomoles per liter (fmol/l).

- **er (estrogen receptors):**
  Represents the concentration of estrogen receptors in femtomoles per liter (fmol/l).
- **hormon (hormonal therapy):**
  A binary variable indicating whether the patient received hormonal therapy.
  0 = no
  1 = yes
- **rfstime (recurrence-free survival time):**
  Represents the time in days to the first occurrence of either recurrence, death, or the last follow-up.
- **status:**
  A binary variable indicating the patient's status.
  0 = alive without recurrence
  1 = recurrence or death

EXPLORATORY DATA ANALYSIS (EDA):

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves visually and statistically summarizing the main characteristics of a dataset. The primary goal of EDA is to understand the structure, patterns, relationships, and anomalies within the data

Data Summary:

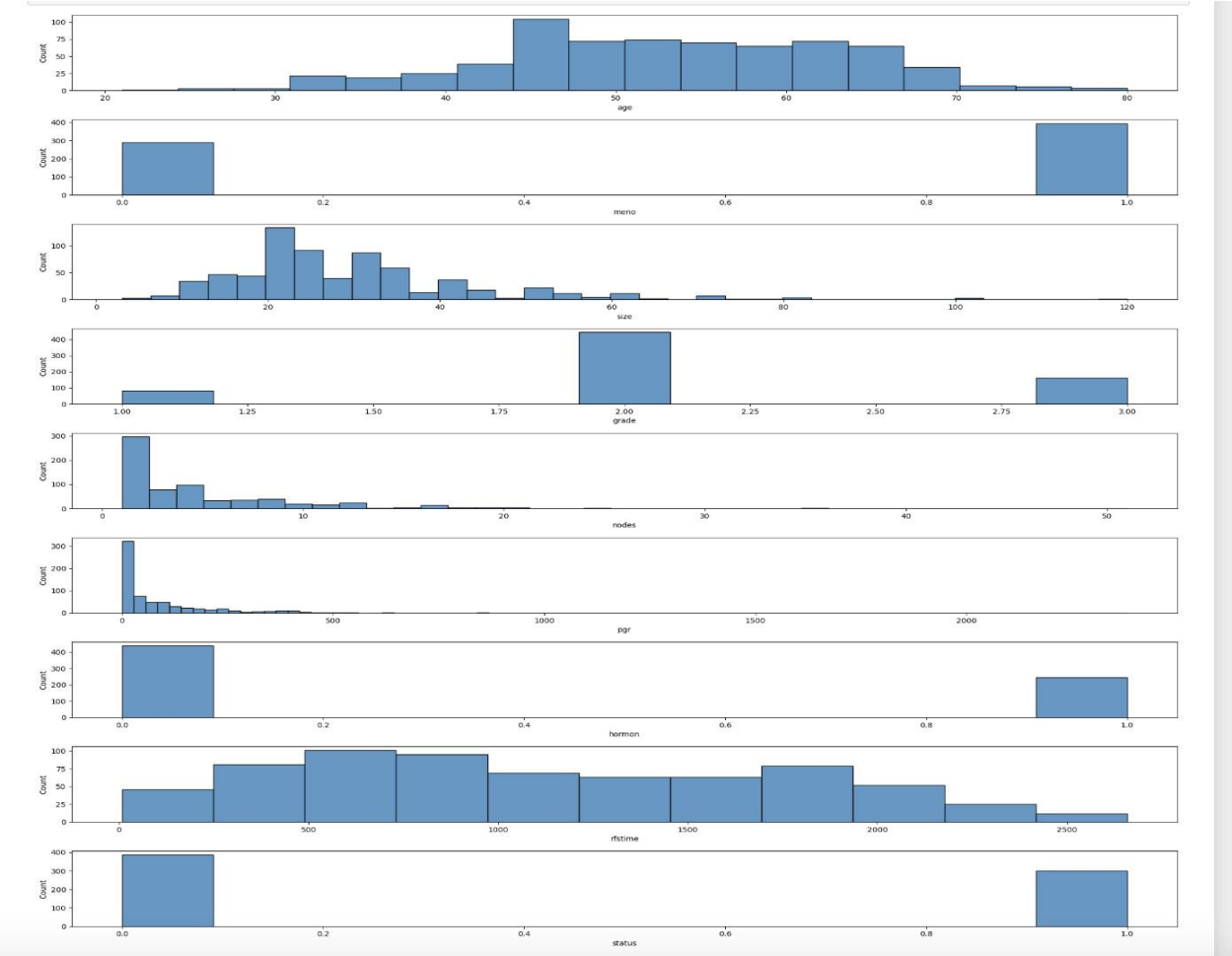
```
In [137]: dataset.describe()
```

Out[137]:

	age	meno	size	grade	nodes	pgr	er	hormon	rfstime	status
count	686.000000	686.000000	686.000000	686.000000	686.000000	686.000000	686.000000	686.000000	686.000000	686.000000
mean	53.052478	0.577259	29.329446	2.116618	5.010204	109.995627	96.252187	0.358601	1124.489796	0.435860
std	10.120739	0.494355	14.296217	0.582808	5.475483	202.331552	153.083963	0.479940	642.791948	0.496231
min	21.000000	0.000000	3.000000	1.000000	1.000000	0.000000	0.000000	0.000000	8.000000	0.000000
25%	46.000000	0.000000	20.000000	2.000000	1.000000	7.000000	8.000000	0.000000	567.750000	0.000000
50%	53.000000	1.000000	25.000000	2.000000	3.000000	32.500000	36.000000	0.000000	1084.000000	0.000000
75%	61.000000	1.000000	35.000000	2.000000	7.000000	131.750000	114.000000	1.000000	1684.750000	1.000000
max	80.000000	1.000000	120.000000	3.000000	51.000000	2380.000000	1144.000000	1.000000	2659.000000	1.000000

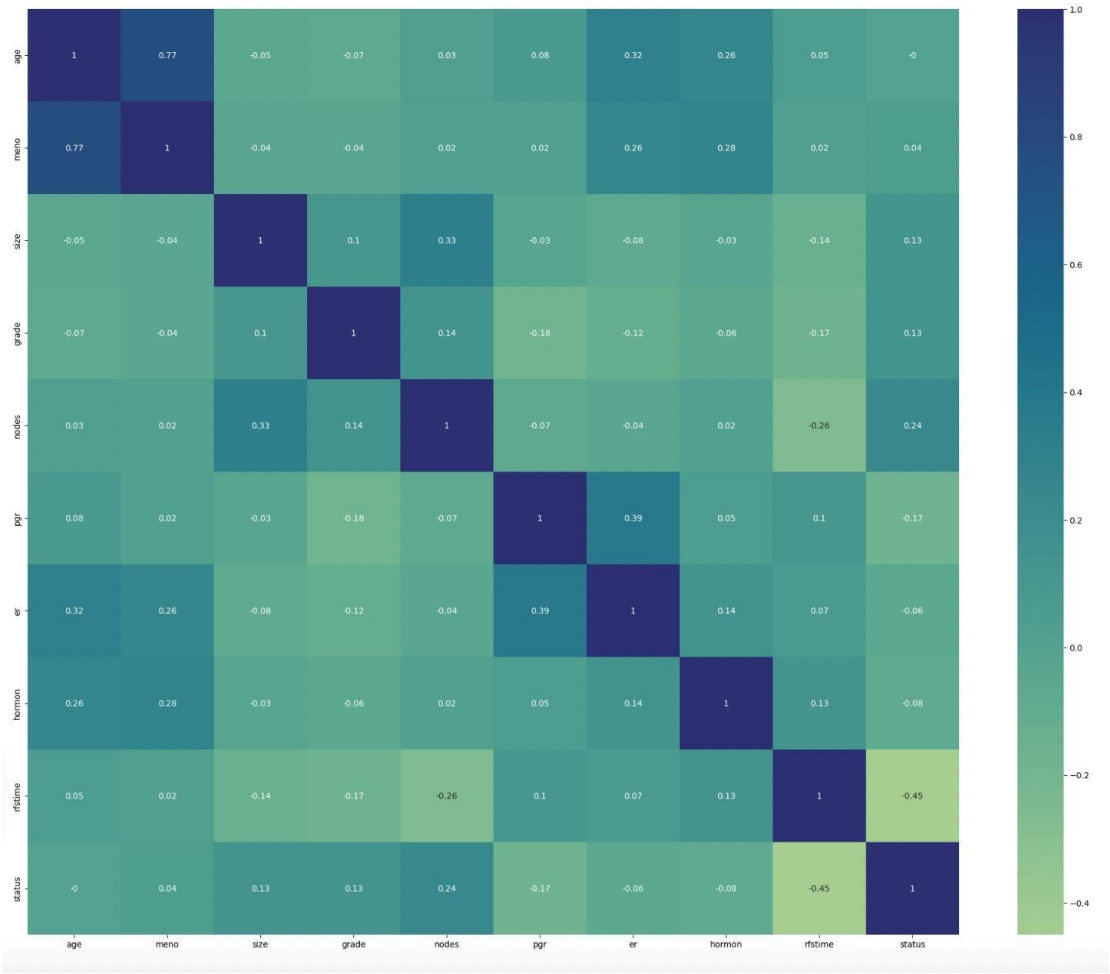
Frequency Distributions:

Create histograms or frequency tables to understand the distribution of values in each variable.



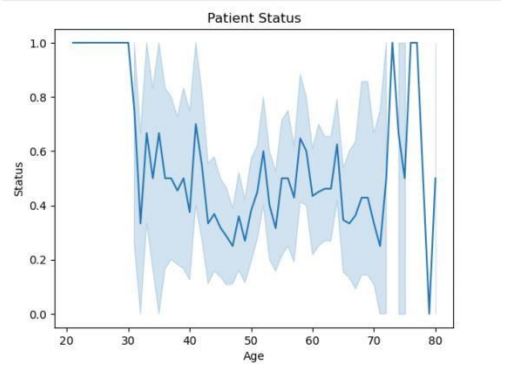
Data Visualization:

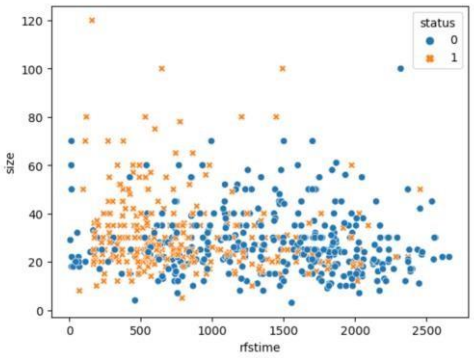
Visualize relationships involving three or more variables using techniques like heatmaps, 3D plots, or parallel coordinates.



Correlation Analysis:

Examine the correlation between variables to identify potential relationships. This can be visualized using correlation matrices or scatter plots.





EDA is an iterative process, and the insights gained during this phase can inform subsequent steps in data analysis, modeling, and decision-making. It helps analysts and data scientists better

understand their data and formulate hypotheses, guiding the development of more accurate and meaningful models.

OUTLIER REMOVAL, SPLIT THE DATA, NORMALIZE

For each feature in the dataset, outliers were removed to increase the robustness of the model. There were 20 rows that were removed in this step.

```
## Remove the outliers
def drop_outliers(data,feature):
    iqr=1.5 * (np.percentile(data[feature],90)-np.percentile(data[feature],10))
    data.drop(data[data[feature]> (iqr+np.percentile(data[feature],90))].index,inplace=True)
    data.drop(data[data[feature]< (np.percentile(data[feature],10)-iqr)].index,inplace=True)
```

```
# apply outlier removal to each numeric feature
for feature in col:
    drop_outliers(df, feature)
```

```
df.shape
```

```
(666, 10)
```

Split the dataset into training and testing sets. The percentage considered is 60% for training, 20% for testing and 20% for validation. The training set is used to train the machine learning model, while the testing set is used to evaluate its performance. In addition to training and testing sets, validation set was considered to fine-tune model hyperparameters.

```
In [327]: X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, train_size=0.6, stratify=Y,random_state=42)
          X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5, stratify=Y_temp,random_state=42)

In [352]: X.shape
Out[352]: (686, 9)

In [348]: X_train.shape
Out[348]: (411, 9)

In [350]: X_val.shape
Out[350]: (137, 9)

In [351]: X_test.shape
Out[351]: (138, 9)
```

Data was standardized using the standard scaler function. Normalization (or feature scaling) is essential to ensure that all features contribute equally to the model training process. It prevents features with larger scales from dominating those with smaller scales.

```
In [328]: X_train[0]
Out[328]: array([  60,    1,   30,    2,    2,   92,   18,    1, 2296])

In [329]: scaler = StandardScaler()
          X_train = scaler.fit_transform(X_train)
          X_val = scaler.fit_transform(X_val)
          X_test = scaler.fit_transform(X_test)

In [330]: X_train[0]
Out[330]: array([ 0.68985999,  0.8871057 ,  0.0125432 , -0.21312323, -0.53335886,
                 -0.08027051, -0.53095631,  1.36898717,  1.72056848])
```

DIMENSIONALITY REDUCTION (PCA)

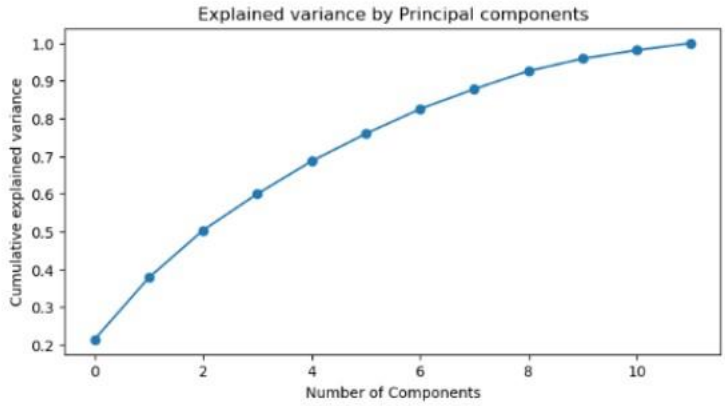
Use PCA to fit the standardized data. PCA identifies the principal components, which are linear combinations of the original features that capture the maximum variance in the data. Transform the standardized data into the principal components. This involves projecting the data onto the subspace defined by the principal components. Each principal component represents a linear combination of the original features. Specify a target variance explained that you want to retain. In this case we have chosen to retain 95% of the variance and set the target variance accordingly.

Determine the number of principal components needed to achieve the specified variance explained (95% in this case). This can be done by looking at the cumulative explained variance plot. Retain only the selected principal components to reduce the dimensionality of the data. The retained components capture most of the variance in the original dataset.

```
# Variance analysis
explained_var_ratio = pca.explained_variance_ratio_
cumulative_var_ratio = np.cumsum(explained_var_ratio)

# Plot the cummulative variance
plt.figure(figsize = (8,4))
plt.plot(cumulative_var_ratio,marker = 'o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative explained variance')
plt.title('Explained variance by Principal components')
```

```
Text(0.5, 1.0, 'Explained variance by Principal components')
```




```
# Choosing the components for 95% variance
n_components = np.where(cumulative_var_ratio >= 0.95)[0][0] + 1
pca_reduced = PCA(n_components=n_components)
reduced_data = pca_reduced.fit_transform(scaled_data)
```

MODEL SELECTION

Model selection involves choosing the most suitable algorithm or model for a given problem. It is a crucial step in the overall machine learning process and can significantly impact the performance of the final model.

As mentioned in Section 1, the machine learning problem we are trying to solve can be defined as a classification model based on the recurrence feature. After careful consideration we have considered having Tree-Based Models which are effective for capturing non-linear relationships and interactions between features. Multiple models of this type were experimented on.

The metrics used for quantifying based on the nature of the problem are accuracy, precision, and F1-score. We have iteratively experimented with different models and parameters.

Random Forest Model :

In [353]:
```
random_forest_model = RandomForestClassifier(n_estimators = 800, max_depth = 10)
random_forest_model.fit(X_train,Y_train)
Y_pred_random_forest = random_forest_model.predict(X_val)
random_forest_accuracy = accuracy_score(Y_val,Y_pred_random_forest)
print(f"Accuracy Score:{random_forest_accuracy}")
random_forest_f1_score = f1_score(Y_val, Y_pred_random_forest, average='weighted')
print(f"F1 Score is:{random_forest_f1_score}")
```

```
Accuracy Score:0.708029197080292
F1 Score is:0.7058202074529389
```

Gradient Boosting Model :

In [332]:
```
dient_boosting_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
dient_boosting_model.fit(X_train,Y_train)
red_gradient_boosting = gradient_boosting_model.predict(X_val)
dient_boosting_accuracy = accuracy_score(Y_val,Y_pred_gradient_boosting)
nt(f"Accuracy Score:{gradient_boosting_accuracy}")
dient_boosting_f1_score = f1_score(Y_val, Y_pred_gradient_boosting, average='weighted')
nt(f"F1 Score:{gradient_boosting_f1_score}")
```

```
Accuracy Score:0.708029197080292
F1 Score:0.7048108240322353
```

Decision Tree Model :

In [333]:
```
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train,Y_train)
Y_pred_decision_tree = decision_tree_model.predict(X_val)
decision_tree_accuracy = accuracy_score(Y_val,Y_pred_decision_tree)
print(f"Accuracy Score:{decision_tree_accuracy}")
decision_tree_f1_score = f1_score(Y_val, Y_pred_decision_tree, average='weighted')
print(f"F1 Score:{decision_tree_f1_score}")
```

```
Accuracy Score:0.6496350364963503
F1 Score:0.6428341101632808
```

KNeighbors Classifier Model :

In [335]:
```
knn_model=KNeighborsClassifier(n_neighbors=16)
knn_model.fit(X_train,Y_train)
Y_pred_knn = knn_model.predict(X_val)
knn_accuracy = accuracy_score(Y_val,Y_pred_knn)
print(f"Accuracy Score:{knn_accuracy}")
knn_f1_score = f1_score(Y_val, Y_pred_knn, average='weighted')
print(f"F1 Score:{knn_f1_score}")
```

```
Accuracy Score:0.6496350364963503
F1 Score:0.6428341101632808
```

Ada Boosting Model :

In [334]:
```
ada_boosting_model=AdaBoostClassifier(n_estimators=500,learning_rate=0.1)
ada_boosting_model.fit(X_train,Y_train)
Y_pred_ada_boosting = ada_boosting_model.predict(X_val)
ada_boosting_accuracy = accuracy_score(Y_val,Y_pred_ada_boosting)
print(f"Accuracy Score:{ada_boosting_accuracy}")
ada_boosting_f1_score = f1_score(Y_val, Y_pred_ada_boosting, average='weighted')
print(f"F1 Score:{ada_boosting_f1_score}")
```

```
Accuracy Score:0.708029197080292
F1 Score:0.7074264759530131
```

Cross-Validation: Initially we have split the dataset in the ratio of 60-20-20 for training, testing and validation sets respectfully. Train the data on this set and once a model is finalized, we have done the same for different splits involving multiple folds to train and test the model on different subsets.

Cross Validation :

In [336]:
```
models = [
    ('Random Forest', random_forest_model),
    ('Gradient Boosting', gradient_boosting_model),
    ('Decision Tree', decision_tree_model),
    ('Ada Boosting', ada_boosting_model),
    ('KNeighbors Classifier', knn_model)
]

# Iterate through each model and perform cross-validation
for model_name, model in models:
    # Perform cross-validation with 5 folds
    cv_scores = cross_val_score(model, X_val, Y_val, cv=5)

    # Output the results
    print(f"Model: {model_name}")
    print("Cross-validation scores:", cv_scores)
    print("Mean CV Accuracy:", cv_scores.mean())
    print("==============================")
```

```
Model: Random Forest
Cross-validation scores: [0.60714286 0.53571429 0.59259259 0.77777778 0.62962963]
Mean CV Accuracy: 0.6285714285714286
==============================
Model: Gradient Boosting
Cross-validation scores: [0.53571429 0.71428571 0.66666667 0.7037037  0.62962963]
Mean CV Accuracy: 0.65
==============================
Model: Decision Tree
Cross-validation scores: [0.53571429 0.64285714 0.66666667 0.66666667 0.55555556]
Mean CV Accuracy: 0.6134920634920634
==============================
Model: Ada Boosting
Cross-validation scores: [0.53571429 0.57142857 0.62962963 0.74074074 0.66666667]
Mean CV Accuracy: 0.6288359788359789
==============================
Model: KNeighbors Classifier
Cross-validation scores: [0.42857143 0.57142857 0.7037037  0.62962963 0.59259259]
Mean CV Accuracy: 0.5851851851851853
==============================
```

At this stage after cross validation, we have observed that Gradient Boost is the best performing model.

Hyper-parameter Tuning

Fine-tune hyperparameters for the Gradient Boost model that we have built. Hyperparameters are configuration settings for a model that are not learned from the data but need to be specified before training.

Hyperparameter tuning for Random forest :

In [337]:
```
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(3, 20)
}

# Create the randomized search model
random_search = RandomizedSearchCV(estimator=random_forest_model, param_distributions=param_dist, n_iter=100, cv=5)

# Fit the randomized search to the data
random_search.fit(X_val, Y_val)

# Get the best parameters and the best score
best_params_rand = random_search.best_params_
best_score_rand = random_search.best_score_

print("Best Parameters (Randomized Search):", best_params_rand)
print("Best Score (Randomized Search):", best_score_rand)
```

```
Best Parameters (Randomized Search): {'max_depth': 16, 'n_estimators': 145}
Best Score (Randomized Search): 0.6579365079365079
```

Hyperparameter tuning for Gradient boost

In [338]:
```
param_dist_gb = {
    'n_estimators': randint(50, 500),
    'learning_rate': uniform(0.01, 0.5),
    'max_depth': randint(3, 10)
}

# Create the Gradient Boosting classifier
gb_model = GradientBoostingClassifier()

# Create the randomized search with cross-validation
random_search_gb = RandomizedSearchCV(estimator=gradient_boosting_model, param_distributions=param_dist_gb, n_iter=1

# Fit the randomized search to the data
random_search_gb.fit(X_val, Y_val)

# Get the best parameters and the best score
best_params_rand_gb = random_search_gb.best_params_
best_score_rand_gb = random_search_gb.best_score_

print("Best Parameters (Randomized Search - Gradient Boosting):", best_params_rand_gb)
print("Best Score (Randomized Search - Gradient Boosting):", best_score_rand_gb)
```

```
Best Parameters (Randomized Search - Gradient Boosting): {'learning_rate': 0.19592594719358958, 'max_depth': 7, 'n_
estimators': 117}
Best Score (Randomized Search - Gradient Boosting): 0.6947089947089947
```

Hyperparameter tuning for Ada Boosting

In [339]:
```
param_dist_ada = {
    'n_estimators': randint(50, 500),
    'learning_rate': uniform(0.01, 0.5),
}


# Create the randomized search with cross-validation
random_search_ada = RandomizedSearchCV(estimator=ada_boosting_model, param_distributions=param_dist_ada, n_iter=100,

# Fit the randomized search to the data
random_search_ada.fit(X_val, Y_val)

# Get the best parameters and the best score
best_params_rand_ada = random_search_ada.best_params_
best_score_rand_ada = random_search_ada.best_score_

print("Best Parameters (Randomized Search - AdaBoost):", best_params_rand_ada)
print("Best Score (Randomized Search - AdaBoost):", best_score_rand_ada)
```

```
Best Parameters (Randomized Search - AdaBoost): {'learning_rate': 0.16298722882926786, 'n_estimators': 70}
Best Score (Randomized Search - AdaBoost): 0.6653439153439153
```

CONCLUSION

In this machine learning project, the objective was to predict the recurrence of breast cancer in women based on a comprehensive set of prognostic variables. Feature engineering, outlier removal, and normalization techniques were applied to enhance the quality of the dataset for model training. Understanding the relative importance of features allows for targeted

interventions and personalized treatment strategies. The focus on patients with node-positive breast cancer ensured relevance to a subset with a higher risk of recurrence.

The developed model(s) could provide insights into the factors most indicative of cancer recurrence. The project likely employed machine learning techniques such as Random Forest, Gradient Boost, ADA Boost, Decision Trees, K-nearest neighbors to build predictive models. Based on the performance, finally, after tuning with hyper parameters, the Gradient Boost algorithm was observed to be best performing and we tested the accuracy with the hyperparameters observed.

In conclusion, the machine learning project on predicting breast cancer recurrence demonstrates the potential to enhance clinical decision-making by leveraging prognostic variables. The insights gained from this project can inform healthcare professionals in their efforts to provide personalized and effective care for women at risk of breast cancer recurrence. Gradient Boost algorithm with the observed parameters can be used in the future predictions with an accuracy of 76%.

FUTURE CONSIDERATIONS

The dataset's temporal context (1984-1989) should be considered, acknowledging potential changes in medical practices and treatment modalities over time. The generalizability of the models to contemporary patient populations should be assessed. Continuous refinement and validation of the predictive models with additional data can enhance their accuracy and robustness. Exploration of newer prognostic variables or advancements in feature selection techniques may contribute to improved model performance.

REFERENCES

1. Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. Proceedings of the Twelfth International Conference on Machine Learning @p. 30-37). San Francisco:
2. Morgan Kaufmann.
3. CHAPTER 13 REINFORCEMENT LEARNING 389
4. Barto, A. (1992). Reinforcement learning and adaptive critic methods. In D. White & S. Sofge (Eds.),
5. Handbook of intelligent control: Neural, fuzzy, and adaptive approaches (pp. 469-491). New
6. York: Van Nostrand Reinhold.
7. Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming.
8. Artificial Intelligence, Special volume: Computational research on interaction and agency,
9. 72(1), 81-138.
10. Singh, S. (1993). Learning to solve markovian decision processes (Ph.D. dissertation).

Also CMPSCI
11. Technical Report 93-77, Department of Computer Science, University of Massachusetts at
12. Amherst.
13. Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. Machine
14. Learning, 22, 123.
15. Sutton, R. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3,
16. 9-44
17. Sutton R. (1991). Planning by incremental dynamic programming. Proceedings of the Eighth Znternational Conference on Machine Learning (pp. 353-357). San Francisco: Morgan Kaufmann.
18. Tesauro, G. (1995). Temporal difference learning and TD-GAMMON. Communications of the ACM,
19. 38(3), 58-68.
20. Thrun, S. (1992). The role of exploration in learning control. In D. White & D. Sofge (Eds.),
21. Handbook of intelligent control: Neural, fizzy, and adaptive approaches (pp. 527-559). New
22. York: Van Nostrand Reinhold.
23. Thrun, S. (1996). Explanation-based neural network learning: A lifelong learning approach. Boston:
24. Kluwer Academic Publishers.
25. Tsitsiklis, J. (1994). Asynchronous stochastic approximation and Q-learning. Machine Learning, 26. 16(3), 185-202.
27. Watkins, C. (1989). Learning from delayed rewards (Ph.D. dissertation). King's College, Cambridge,
28. England.
29. Watkins, C., & Dayan, P. (1992). Q-learning. Machine Learning, 8, 279-292.
30. Zhang, W., & Dietterich, T. G. (1996). High-performance job-shop scheduling with a time-delay
31. TD(A) network. In D. S. Touretzky, M. C. Mozer, &