

SCHOOL OF COMPUTER SCIENCES

CPC251W Machine Learning and Computational Intelligence

Academic Session 2024/2025, Semester 1/2

Project 1

Group 3

Prepared By:

Syeda Laiba Faraz USM202212659

Faiq Ahmed Shaikh USM202212583

Ma Bowen USM202312576

Submission Date:

20th May 2025

Table of Contents

<i>No.</i>	<i>Title</i>	<i>Page Number</i>
1.0	Problem Statement	2
2.0	Dataset Selection	3
3.0	Machine Learning Models Selected	3
4.0	Data Preprocessing Random Forest and Decision Tree 4.1 Data Understanding 4.2 Data Cleaning 4.3 Feature Selection	4
5.0	Random Forest Development 5.1 Baseline Random Forest 5.2 Grid Search Random Forest 5.3 Random Search Random Forest	7
6.0	Decision Tree Development 6.1 Baseline Model 6.2 Feature Importance 6.3 Gini vs Entropy 6.4 GridSearchCV	9
7.0	Comparison between the 2 Machine Learning Models 7.1 Improvement between the Models	15
8.0	Performance Metrics 8.1 Random Forest 8.2 Decision Tree	16
9.0	Reflective Report 9.1 Laiba Faraz 9.2 Faiq Ahmed Shaikh 9.3 Ma Bowen	24

1. Problem Statement

According to the World Health Organisation (WHO), “Cervical cancer is the fourth most common cancer in women globally with around 660 000 new cases and around 350 000 deaths in 2022.” Early detection of cervical abnormalities through routine Pap tests and biopsies plays a crucial role in preventing the progression of the disease. However, manual assessment is often time-consuming and subject to human error.

In this project, we aim to develop a predictive model using patient medical data to assist in the early detection of cervical cancer. By applying machine learning techniques to a dataset containing features such as demographic information, sexual and reproductive health history, use of contraceptives, and STD records, we seek to predict whether a patient is likely to test positive in a biopsy.

This model will help healthcare professionals prioritize patients for further examination, potentially saving lives through early intervention.

2. Dataset Selection

For this study we utilised the Cervical Cancer dataset provided in elearning. This dataset consists of anonymous patient records, lifestyle habits and medical history related to sexually transmitted diseases and cervical cancer screening results.

This dataset includes several target variables related to cervical cancer diagnosis (such as Hinselmann, Schiller, Cytology, and Biopsy). These factors make it suitable for developing classification models aimed at early detection and risk prediction of cervical cancer.

3. Machine Learning Models Selected

The two Machine Learning Models used by us for this project are as follows:

- **Random Forest**
- **Decision Tree**

Each model has distinct characteristics and applications, making them suitable for various types of data and problems. Here, we'll provide a concise yet understandable explanation of both models.

i. Random Forest

A Random Forest is an ensemble machine learning model that combines multiple decision trees. Each tree in the forest is trained on a random sample of the data (bootstrap sampling) and considers only a random subset of features when making splits (feature randomization). For classification tasks, the forest predicts by majority voting among trees, while for regression tasks, it averages the predictions. The model's strength comes from its “wisdom of crowds” approach — while individual trees might make errors, the collective decision-making process tends to average out these mistakes and arrive at more reliable predictions.

ii. Decision Tree

Decision Trees is a popular method in machine learning for both classification and regression tasks. It works by breaking down a dataset into smaller and smaller subsets based on different criteria, while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf nodes represent a classification or decision. This structure allows you to make decisions by following the paths from the root to a leaf.

4. Data Preprocessing

4.1 Data Understanding

To determine which patient has cervical cancer, the parameters in the dataset are examined to have a basic understanding.

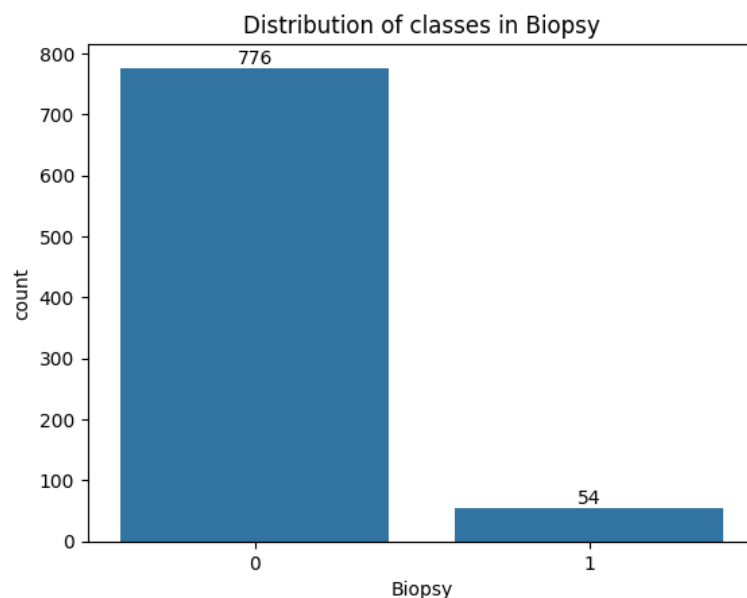


Figure 1: Distribution of “Biopsy” class

The target variable in this analysis is the result of the Biopsy test, which confirms whether a patient is diagnosed with cervical cancer. This target is highly imbalanced, the dataset is heavily skewed towards the '0' class. As a result, only 6.41% of the patients reported positive for "Biopsy" and this bias would affect the decisions made when constructing the models.

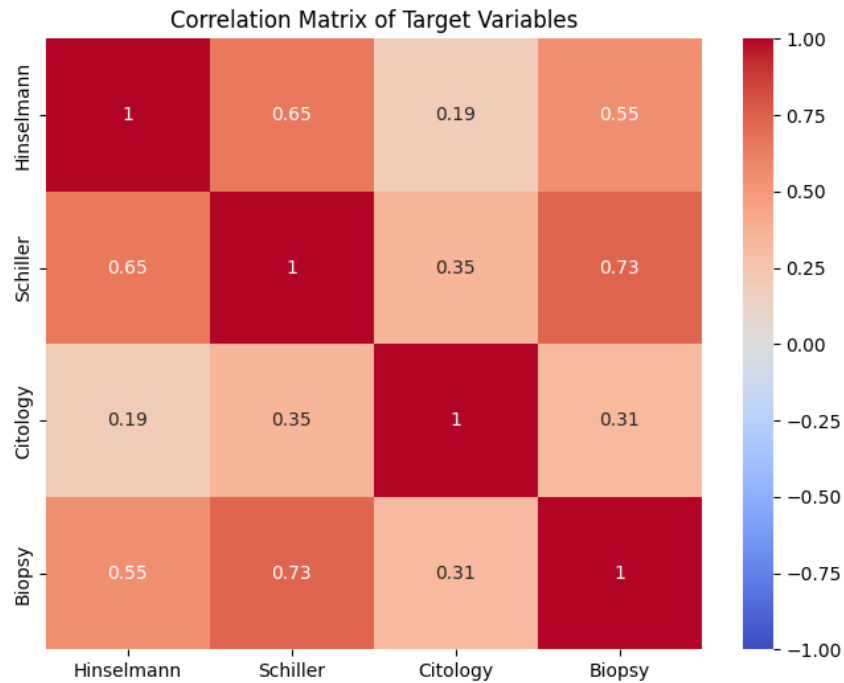


Figure 2: Correlation Matrix

The correlation matrix examines relationships between the four target variables. Low to moderate correlations support binary classification for Biopsy, as each test provides distinct diagnostic information. Multi-class classification is impractical due to class imbalance (6.41% positive class for Biopsy) and clinical practice of interpreting tests separately.

4.2 Data Cleaning

Before building predictive models, it is essential to clean the dataset to ensure data quality and reliability. The initial dataset contained missing values in several attributes, which were handled by either imputing appropriate values or removing incomplete records, depending on the extent and importance of the missing data.

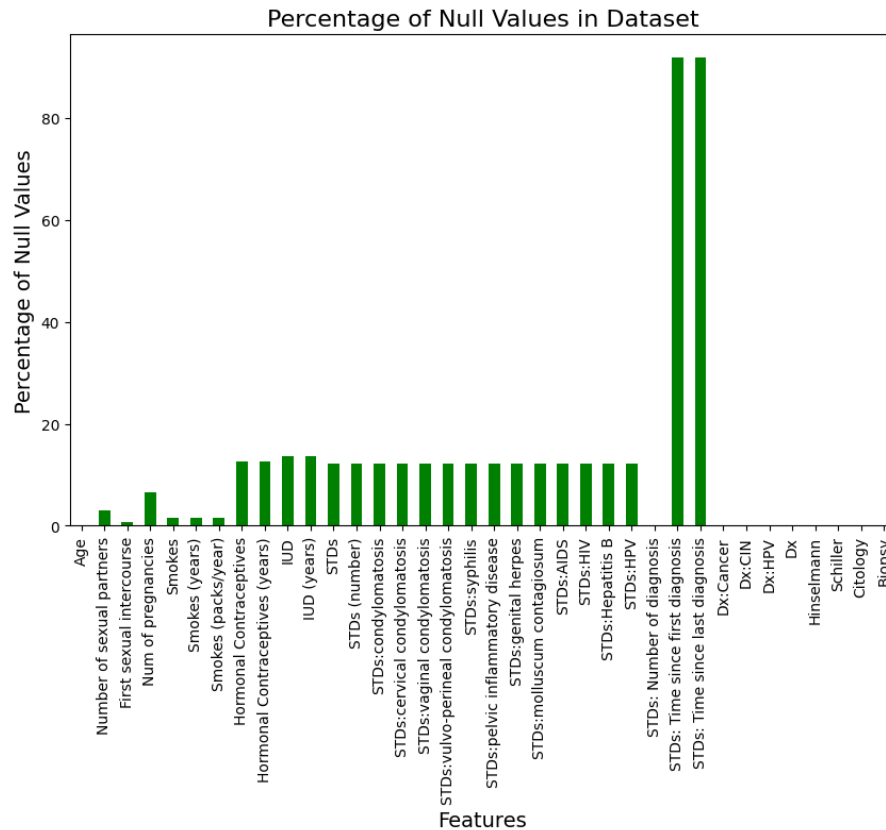


Figure 3: % of Null Values in the Dataset

Features 26 and 27 ("STDs: Time since first diagnosis" and "STDs: Time since last diagnosis" as shown in df.info) were dropped because they have too many nulls.

```
# Impute null values in features with its median
df = df.fillna(df.median())
```

Figure 4: Code to impute Null Values with the median in the dataset

```
df.duplicated().sum() # this prints the True/False to mark the duplicate rows
#returns a boolean Series indicating whether each row is a duplicate of a previous row
```

5

Figure 5: Code to display the sum of duplicates in the dataset

```
df = df.drop_duplicates()
```

Figure 6: Code to drop duplicates in the dataset

Although different patients might coincidentally have identical values, exact matches are rare and usually suggest data entry duplication. To avoid bias and ensure fair model training, we removed these duplicates while preserving the dataset's diversity and size.

4.3 Feature Selection

Feature selection using the chi2 (chi-squared) statistical test was applied, which measures the dependence between each feature and the target variable.

5. Random Forest Development

5.1 Baseline Random Forest:

The baseline Random Forest model was implemented with default parameters to establish a reference performance. It was trained on pre-processed dataset using X-train and evaluated on X-test.

```
In [140]: # Baseline Random Forest with default parameters
print("Training Baseline Random Forest (Default Parameters)...")
rf_baseline = RandomForestClassifier(random_state=42)
rf_baseline.fit(X_train, y_train)
baseline_pred = rf_baseline.predict(X_test)
baseline_accuracy = accuracy_score(y_test, baseline_pred)
```

Training Baseline Random Forest (Default Parameters)...

Figure 7: Baseline Fit with Default parameters

5.2 Grid Search Random Forest:

Grid Search was used to fine-tune the Random Forest model by exhaustively searching a predefined hyperparameter grid (**n_estimators**, **max_depth**, etc.) with **scoring='f1'** and **class_weight='balanced'** to address the imbalanced dataset.

```
[141]: # Random Forest with Grid Search
print("\nTraining Random Forest with Grid Search...")
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': ['balanced']
}
rf_grid = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf_grid, param_grid=param_grid, cv=5, n_jobs=-1, scoring='f1', verbose=1)
grid_search.fit(X_train, y_train)
```

Training Random Forest with Grid Search...
Fitting 5 folds for each of 108 candidates, totalling 540 fits

```
[141]: GridSearchCV ① ?
      best_estimator_: RandomForestClassifier
      RandomForestClassifier(class_weight='balanced', max_depth=20,
                             min_samples_leaf=2, min_samples_split=5,
                             n_estimators=200, random_state=42)
      > RandomForestClassifier ②
```

Figure 8: Fitting model with GridSearch

5.3 Random Search Random Forest:

Random Search was applied to explore a wider hyperparameter space with 50 iterations, using randomized distributions (randint for parameters) and the same scoring and class weighting.


```
# Random Forest with Random Search
print("\nTraining Random Forest with Random Search...")
param_dist = {
    'n_estimators': randint(50, 500),
    'max_depth': randint(1, 20),
    'min_samples_split': randint(2, 16),
    'min_samples_leaf': randint(1, 7),
    'class_weight': ['balanced']
}
rf_random = RandomForestClassifier(random_state=42)
random_search = RandomizedSearchCV(estimator=rf_random,
                                   param_distributions=param_dist,
                                   n_iter=50,
                                   cv=5, n_jobs=-1,
                                   scoring='f1', verbose=1, random_state=42)
random_search.fit(X_train, y_train)
```

Training Random Forest with Random Search...
 Fitting 5 folds for each of 50 candidates, totalling 250 fits

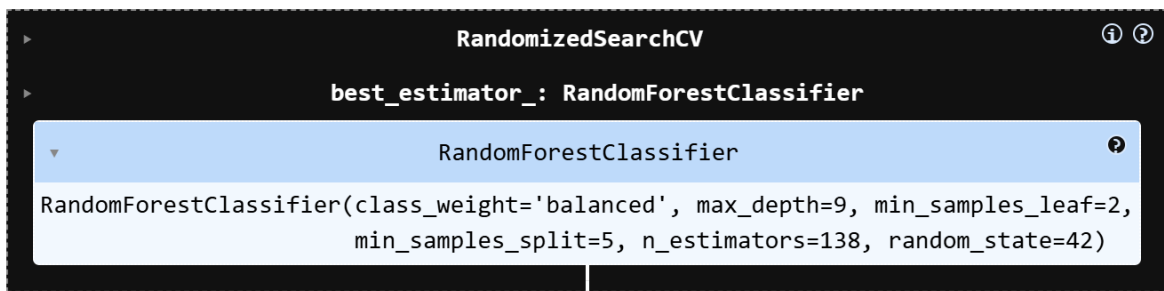


Figure 9: Fitting model with RandomSearch

6. Decision Tree Development

6.1 Baseline Model

The decision tree was first trained using all the default parameters of the classifier listed below, without any hyperparameter tuning.

```

default = DecisionTreeClassifier()
default.get_params()

{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'random_state': None,
 'splitter': 'best'}

```

Figure 10: Default Parameters on Decision Tree

This model was trained as a benchmark to compare with other models of the Decision Tree.

```

# Baseline Model - Test Set
base_clf = DecisionTreeClassifier()
base_clf.fit(X_train,y_train)

# Test the model with the testing data set and prints Accuracy Score
pred_base = base_clf.predict(X_test)
base_acc = accuracy_score(y_test, pred_base)
print("Baseline Accuracy:", base_acc)
print("\nClassification Report:\n", classification_report(y_test, pred_base))

```

Baseline Accuracy: 0.9461077844311377

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	156
1	0.60	0.55	0.57	11
accuracy			0.95	167
macro avg	0.78	0.76	0.77	167
weighted avg	0.94	0.95	0.94	167

Figure 11: Baseline model of Decision Tree (Test Set)

6.2 Feature Importance

Another Decision Tree was trained on a bunch of important features. This model achieved a higher accuracy of 96.41% and significantly better performance on the minority class, F1-score of 0.73. Showing that Feature selection significantly improved class sensitivity.

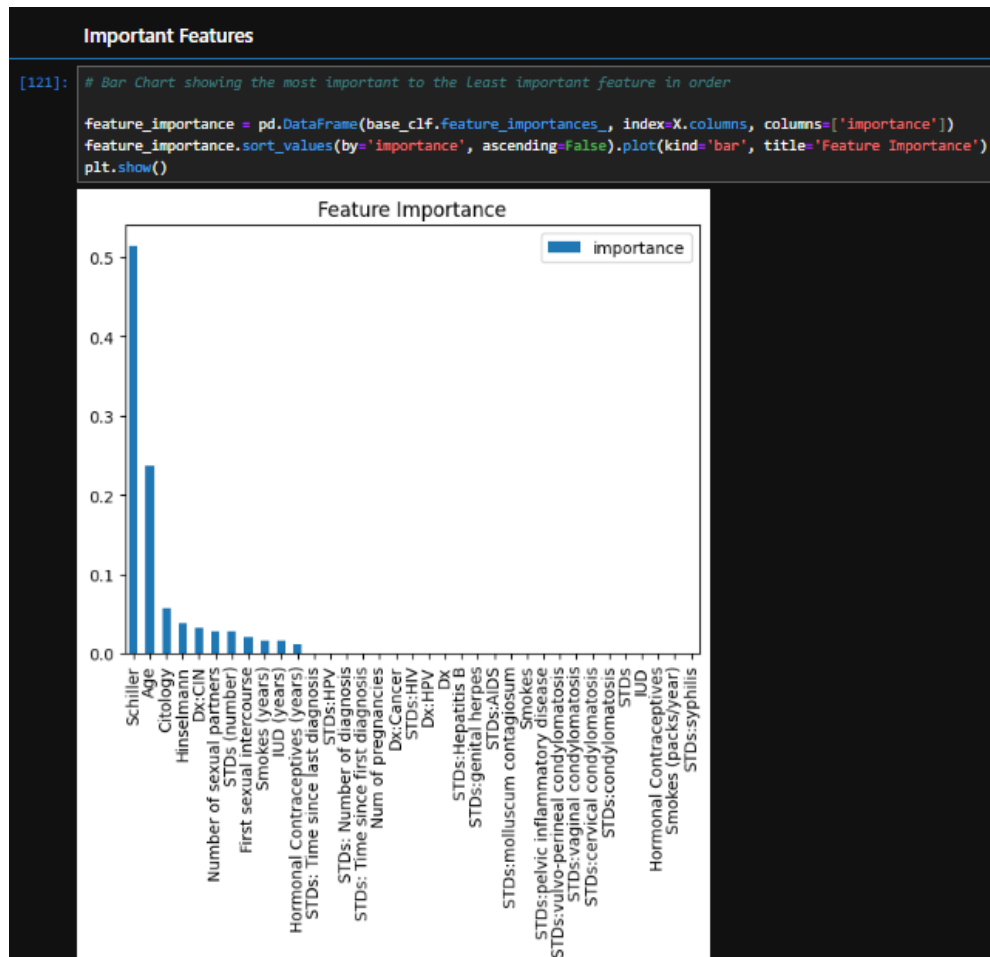


Figure 12: Bar Chart of Important Features in order from Most Important to Least

```

# Set a threshold for importance
threshold = 0.05 # Keep features with >5% importance

selected_features = feature_importance[feature_importance['importance'] > threshold].index.tolist()
print("Selected important features:", selected_features)

X_train_selected = X_train[selected_features]
X_valid_selected = X_valid[selected_features]
X_test_selected = X_test[selected_features]

clf_selected = DecisionTreeClassifier(random_state=42)
clf_selected.fit(X_train_selected, y_train)

y_pred_valid = clf_selected.predict(X_valid_selected)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred_valid))
print("\nClassification Report:\n", classification_report(y_valid, y_pred_valid))

Selected important features: ['Age', 'Schiller', 'Citology']
Validation Accuracy: 0.9401197604790419

```

Figure 13: Trained Model on Selected Features (Validation Set)

```

y_pred = clf_selected.predict(X_test_selected)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

Accuracy: 0.9640718562874252

```

Figure 14: Trained Model on Selected Features (Test Set)

6.3 Gini vs Entropy

This model was manually tuned on Gini and entropy. The Gini based model showed the best performance, in comparison to Entropy.

```

Gini Vs Entropy

# Train Decision Tree with Gini
dtree_gini = DecisionTreeClassifier(criterion='gini', random_state=42)
dtree_gini.fit(X_train, y_train)
y_pred_gini = dtree_gini.predict(X_test)

# Train Decision Tree with Entropy
dtree_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
dtree_entropy.fit(X_train, y_train)
y_pred_entropy = dtree_entropy.predict(X_test)

# Both models
accuracy_gini = accuracy_score(y_test, y_pred_gini)
precision_gini = precision_score(y_test, y_pred_gini, average='weighted')
f1_gini = f1_score(y_test, y_pred_gini, average='weighted')

accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
precision_entropy = precision_score(y_test, y_pred_entropy, average='weighted')
f1_entropy = f1_score(y_test, y_pred_entropy, average='weighted')

# Print scores
print("Gini - Accuracy:", accuracy_gini)
print("Gini - Precision:", precision_gini)
print("Gini - F1 Score:", f1_gini)

print("\nEntropy - Accuracy:", accuracy_entropy)
print("Entropy - Precision:", precision_entropy)
print("Entropy - F1 Score:", f1_entropy)

# Compare models
better_model = []

if accuracy_gini > accuracy_entropy:
    better_model.append("accuracy (Gini)")
elif accuracy_entropy > accuracy_gini:
    better_model.append("accuracy (Entropy)")

if precision_gini > precision_entropy:
    better_model.append("precision (Gini)")
elif precision_entropy > precision_gini:
    better_model.append("precision (Entropy)")

if f1_gini > f1_entropy:
    better_model.append("f1-score (Gini)")
elif f1_entropy > f1_gini:
    better_model.append("f1-score (Entropy)")

print("\nBetter model:")
for metric in better_model:
    print(f"- Better in {metric}")

gini_better = sum('Gini' in m for m in better_model)
entropy_better = sum('Entropy' in m for m in better_model)

print("\nOverall better model:")
if gini_better > entropy_better:
    print("Decision tree with Gini impurity performs better overall.")
elif entropy_better > gini_better:
    print("Decision tree with Entropy performs better overall.")
else:
    print("Both models perform equally well.")

```

Figure 15: Trained Model on Gini Vs Entropy

```

Gini - Accuracy: 0.9768479841916168
Gini - Precision: 0.9768479841916168
Gini - F1 Score: 0.9768479841916168

Entropy - Accuracy: 0.9481197684798419
Entropy - Precision: 0.9345836628629534
Entropy - F1 Score: 0.9369825748345192

Better model:
- Better in accuracy (Gini)
- Better in precision (Gini)
- Better in f1-score (Gini)

Overall better model:
Decision tree with Gini impurity performs better overall.

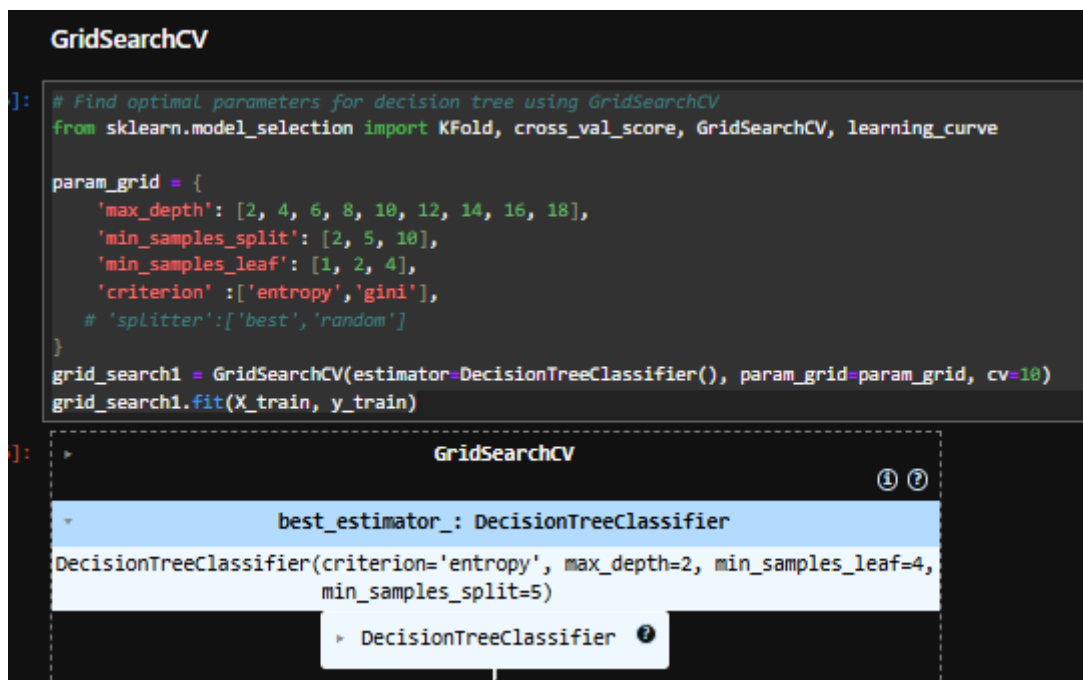
```

Figure 16: Results of Gini Vs Entropy

6.4 GridSearchCV

GridSearchCV with cross validation was applied to systematically find the best hyperparameters. Using **criterion='entropy'**, **max_depth=2**, **min_samples_leaf=4**, and **min_samples_split=2**.

This model was selected as the final model for its strong performance and simplicity.



```
GridSearchCV

In [ ]: # Find optimal parameters for decision tree using GridSearchCV
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV, learning_curve

param_grid = {
    'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['entropy', 'gini'],
    # 'splitter': ['best', 'random']
}

grid_search1 = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=param_grid, cv=10)
grid_search1.fit(X_train, y_train)

In [ ]: ▶ GridSearchCV ① ②

    best_estimator_: DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=2, min_samples_leaf=4,
min_samples_split=5)
    ▶ DecisionTreeClassifier ①
```

Figure 17: GridSearchCV optimal parameters

```
[138]: best_model1 = grid_search1.best_estimator_
y_pred = best_model1.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Best Parameters:", grid_search1.best_params_)
print("Test Accuracy:", accuracy)
#print(classification_report(y_test_20, y_pred))

# Additional metrics
CV1_precision = precision_score(y_test, y_pred, average="weighted")
CV1_recall = recall_score(y_test, y_pred, average="weighted")
CV1_f1 = f1_score(y_test, y_pred, average="weighted")

print("Precision:", CV1_precision)
print("Recall:", CV1_recall)
print("F1 Score:", CV1_f1)
# Confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print('Confusion matrix:')
print(confusion)

Best Parameters: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split': 5}
```

Figure 18: Trained Model on GridSearchCV

7. Comparison Between the 2 Machine Learning Models

The 2 Machine learning Models were compared against each other based on several key metrics including accuracy, F1-score for the minority class (Class 1), precision and recall.

Table 1: Comparison Table showing key metrics:

	Decision Tree	Random Forest
Best accuracy	Moderate Accuracy	High Accuracy
Best F1-Score for Class 1	Balanced precision and Recall	Higher F1 for Class 1
Precision and recall for Class 1	Balanced Precision and Recall	Better Recall
Training time	Faster to train	Slower
Interpretability	easy to visualise	complex model

Based on the Metrics of both Random Forest and Decision Tree:

Grid Search Random Forest and Random Search RF both have the highest recall and F1- Score for class 1 respectively: 0.91 and 0.80

Whereas, Baseline Random Forest has the highest precision 0.75 followed by GridSearchCV Decision Tree 0.73

Random Forest (Grid/Random Search) models outperform Decision Trees in recall and F1-score, meaning they catch more positive cases and balance precision/recall better.

Decision Trees with GridSearchCV or Feature Selection are competitive, especially on precision and balanced F1, but lag behind Random Forest in recall.

DT trains much faster because it's a single tree, while RF requires more time for training many trees and tuning hyperparameters.

7.1 Improvement between the Models

Random Forest:

Hyperparameter tuning through Grid and Random Search

Class weight balancing to address dataset imbalance

Parameter space exploration with Random Search

Decision Tree:

Feature importance selection

Criterion comparison (Gini vs Entropy)

GridSearchCV optimization with cross-validation

Random Forest benefited from ensemble learning and extensive hyperparameter tuning, leading to higher recall and F1-score for minority cases but at the cost of training time.

Perhaps, with further Manual tuning through Trial and Error, Decision tree could be trained to have higher Recall and F1-Score.

In the context of a cervical cancer dataset, where the primary concern is identifying potential cancer cases (Class 1), the Decision Tree model with feature importance selection and GridSearchCV tuning is better for the following reasons:

Decision Trees provide clear, understandable decision paths. This is crucial in medical applications, where clinicians need to understand and trust the model's reasoning.

While Random Forests can sometimes achieve slightly higher recall or F1-scores due to the multiple trees and averaging, they are more complex and less interpretable, which may be a drawback in clinical settings. It's hard to explain exactly why a prediction was made, which is critical in medical contexts where doctors need clear, understandable reasoning to trust and act on the model's output.

If looking at raw numbers, Random Forest is marginally better than Decision Tree, but in a real life medical setting where time-constraints, computational resources and interpretability matters, Decision Tree with GridSearchCV is better in that regard and the small percentage difference does not have a significant impact on the model.

8. Performance Metrics

8.1 Random Forest:

This section presents the performance metrics of the Random Forest models (Baseline, Grid Search, and Random Search) on the test set for predicting Biopsy. The test set consists of 167 samples, with 156 negative and 11 positive cases, reflecting the dataset's imbalance (6.41% positive class). The models were evaluated using classification reports (precision, recall, F1-score, accuracy) and confusion matrices. Due to the severe imbalance, the minority class (class 1) shows low performance.

8.1.1 Baseline Metrics:

The baseline model uses default parameters to establish a reference. Figure below shows the classification report and confusion matrix. High accuracy (0.93) reflects majority class dominance (156 negatives), but class 1 (11 positives) has near-zero recall, precision, and F1-score due to the imbalance.

```
145]: y_pred_baseline = rf_baseline.predict(X_test)
print("\nBaseline - Classification Report:")
print(classification_report(y_test, y_pred_baseline))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_baseline))
```

```
Baseline - Classification Report:
              precision    recall  f1-score   support

     0           0.93       0.99      0.96         156
     1           0.00       0.00      0.00           11

   accuracy              0.93         167
  macro avg           0.47       0.50      0.48         167
 weighted avg           0.87       0.93      0.90         167

Confusion Matrix:
[[155   1]
 [ 11   0]]
```

Figure 19: BaseLine (RF) Metrics

8.1.2 Grid Search Metrics:

Grid Search fine-tunes hyperparameters (`n_estimators`, `max_depth`, etc.) with `scoring='f1'` and `class_weight='balanced'`. Figure 8.2 displays the results, with accuracy (0.92) similar to the baseline. Class 1 metrics remain low, as the model struggles with the minority class despite tuning. The confusion matrix shows a slight increase in false positives compared to the baseline.

```
[146]: # Predictions and evaluations for Grid Search
y_pred_grid = rf_grid_best.predict(X_test)
print("\n📊 Grid Search - Classification Report:")
print(classification_report(y_test, y_pred_grid))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_grid))
```

```
📊 Grid Search - Classification Report:
              precision    recall  f1-score   support

     0       0.93         0.98         0.96         156
     1       0.00         0.00         0.00          11

   accuracy                   0.92         167
  macro avg       0.47         0.49         0.48         167
 weighted avg     0.87         0.92         0.89         167

Confusion Matrix:
[[153   3]
 [ 11   0]]
```


Figure 20: Grid Search (RF) Metrics

8.1.3 Random Search Metrics:

Random Search explores a wider parameter space (50 iterations) with randomized distributions, also using `scoring='f1'` and `class_weight='balanced'`. Figure below presents the metrics, with accuracy (0.90) slightly lower than the baseline. Class 1 performance is still poor, and the confusion matrix shows more false positives than grid search.

```
[147]: # Predictions and evaluations for Random Search
y_pred_random = rf_random_best.predict(X_test)
print("\nRandom Search - Classification Report:")
print(classification_report(y_test, y_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_random))
```

```

 Random Search - Classification Report:
      precision    recall  f1-score   support

     0       0.93      0.97      0.95       156
     1       0.00      0.00      0.00        11

   accuracy          0.90       167
  macro avg       0.47      0.48      0.47       167
 weighted avg       0.87      0.90      0.89       167

Confusion Matrix:
[[151   5]
 [ 11   0]]

```

Figure 21: Random Search (RF) Metrics

8.2 Decision Tree

Multiple Decision Tree models were evaluated using different configurations and techniques to identify the best-performing classifier. The evaluation was based on standard classification metrics including **accuracy**, **precision**, **recall**, **F1-score**, and the **confusion matrix**.

8.2.1 Baseline Decision Tree

Performs reasonably well out of the box, but minority class recall is moderate. The recall and F1-score indicate that almost half of the positive class samples were misclassified.

```
[36]: # Test the model with the testing data set and prints Accuracy Score
pred_base = base_clf.predict(X_test)
base_acc = accuracy_score(y_test, pred_base)
print("Baseline Accuracy :", base_acc)
print("\nClassification Report:\n", classification_report(y_test, pred_base))
# Confusion matrix Base
confusion_base = confusion_matrix(y_test, pred_base)
print('Confusion matrix:')
print(confusion_base)
```

Baseline Accuracy : 0.9520958083832335

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	156
1	0.64	0.64	0.64	11
accuracy			0.95	167
macro avg	0.81	0.81	0.81	167
weighted avg	0.95	0.95	0.95	167

Confusion matrix:

```
[[152  4]
 [ 4  7]]
```

Figure 22: DT Baseline Model Metrics

8.2.2 Feature Importance

This model was trained using a reduced set of features selected based on feature importance rankings. Its performance on class 1 was significantly weaker. This model exhibited lower recall and F1-score for class 1, indicating that it struggled to correctly identify positive cases. This suggests that the model may be underfitting or that the selected features were not sufficient to generalize well to the minority class. Thus, highlighting a common challenge in imbalanced datasets, where optimizing for overall accuracy can mask poor performance on rare but critical classes.

```

# Feature Importance - Selected Features - Test Set

y_pred = clf_selected.predict(X_test_selected)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix Selected
confusion_selected = confusion_matrix(y_test, pred_base)
print('Confusion matrix:')
print(confusion_selected)

Accuracy: 0.9640718562874252

Classification Report:
              precision    recall  f1-score   support

     0       0.98        0.98        0.98        156
     1       0.73        0.73        0.73         11

   accuracy          0.96          0.96          0.96        167
  macro avg       0.85        0.85        0.85        167
weighted avg       0.96        0.96        0.96        167

Confusion matrix:
[[152   4]
 [  4   7]]

```

Figure 23: Feature Importance Metrics

8.2.3 Gini vs. Entropy

The comparison between Gini and Entropy splitting criteria revealed that **Gini impurity provided better and more stable results**. The model using Gini impurity achieved higher recall and F1-score for the class 1, indicating better handling of imbalanced class distribution. In contrast, the entropy-based model struggled with both precision and recall, resulting in lower precision and recall, particularly for class 1. This suggests that information gain did not align well with the characteristics of this dataset. Gini impurity is more effective in this context, especially when dealing with a highly imbalanced dataset.

```

# Gini Vs Entropy

print("\nClassification Report for Gini:")
print(classification_report(y_test, y_pred_gini))

# Confusion matrix Gini
confusion_gini = confusion_matrix(y_test, y_pred_gini)
print('Confusion matrix:')
print(confusion_gini)
print("-----")

print("\nClassification Report for Entropy:")
print(classification_report(y_test, y_pred_entropy))

# Confusion matrix Entropy
confusion_En = confusion_matrix(y_test, y_pred_entropy)
print('Confusion matrix:')
print(confusion_En)

```

Classification Report for Gini:				
	precision	recall	f1-score	support
0	0.97	0.97	0.97	156
1	0.64	0.64	0.64	11
accuracy			0.95	167
macro avg	0.81	0.81	0.81	167
weighted avg	0.95	0.95	0.95	167

Confusion matrix:
[[152 4]
[4 7]]

Classification Report for Entropy:				
	precision	recall	f1-score	support
0	0.96	0.97	0.97	156
1	0.50	0.36	0.42	11
accuracy			0.93	167
macro avg	0.73	0.67	0.69	167
weighted avg	0.93	0.93	0.93	167

Confusion matrix:
[[152 4]
[7 4]]

Figure 24 : Gini vs Entropy Metrics

8.2.4 GridSearchCV

The **GridSearchCV-tuned model** emerged as best-performing model in this study amongst other decision Tree models. This model benefited from systematic hyperparameter optimization, which adjusted key parameters such as max_depth, min_samples_split, and

min_samples_leaf to better reflect the structure of the dataset. In addition to that, the cross-validation helped reduce the risk of overfitting.

In a classification problem where positive cases are rare but clinically important, this balance is critical for developing reliable diagnostic tools.

```
# GridSearchCV

# Classification report
print("\nClassification Report - GridSearchCV:")
print(classification_report(y_test, y_pred))

# Confusion matrix
confusion_Grid = confusion_matrix(y_test, y_pred)
print('Confusion matrix:')
print(confusion_Grid)
```



```
Classification Report - GridSearchCV:
              precision    recall  f1-score   support

      0       0.98        0.98        0.98        156
      1       0.73        0.73        0.73         11

   accuracy          0.96          167
  macro avg       0.85        0.85        0.85        167
weighted avg       0.96        0.96        0.96        167

Confusion matrix:
[[153  3]
 [ 3  8]]
```

Figure 25: Decision Tree GridSearchCV Metrics

9. Reflective Report

9.1 Laiba Faraz

Having previously worked with Decision Trees, this project gave me the opportunity to revisit the model from a more analytical and performance-focused perspective. Rather than simply applying a Decision Tree classifier with the goal of maximizing overall accuracy, I focused on optimizing the model's performance on an imbalanced dataset, a challenge often commonly seen in medical diagnosis.

I focused more on improving recall and F1-score, because in real-life situations like healthcare, missing a positive case can be much more serious than a false alarm.

One of the key challenges was improving the model's performance on the class 1. I initially encountered low recall and F1-score, which made me rethink how I define and evaluate "good performance" in machine learning.

Aside from building and evaluating models, I also worked on creating a clear and structured README file to explain the dataset, preprocessing steps, modelling approach, and results. This helped me better document the workflow and ensure that the project was understandable and reproducible for others.

9.2 Faiq Ahmed Shaikh

I've learned more about the real-world uses of data science (DS) in healthcare through my machine learning (ML) mini-project on the cervical cancer dataset. The real-world stakes involved in working with the Cervical Cancer dataset inspired me to make sure our models were both operational and in line with the project's early detection objective.

My primary focus was on developing and fine-tuning the Random Forest (RF) models, including the baseline, Grid Search, and Random Search variants, as detailed in Sections 5.0 and 8.0. Starting with the baseline model was a humbling experience. It provided a solid starting point with high accuracy (0.93), but the near-zero recall for the minority class (11 positive cases out of 167) highlighted the dataset's imbalance a challenge I hadn't fully anticipated. This led me to explore Grid Search and Random Search to optimize hyperparameters, using `scoring=f1` and `classweight='balanced'` to address the imbalance.

9.3 Ma Bowen

Throughout this machine learning project on cervical cancer prediction, I have gained valuable insights and developed essential skills in data science. Working with a dataset related to such a critical healthcare issue emphasized the real-world impact of our work, motivating me to ensure our models were both accurate and interpretable.

The process of experimenting with different hyperparameters through Grid Search CV taught me the importance of systematic model tuning rather than arbitrary adjustments. One significant challenge we encountered was balancing model performance with interpretability.

I also focused on evaluating our models using appropriate metrics beyond accuracy, particularly precision, recall, and F1-score, as false negatives in cancer detection can have serious consequences. This experience reinforced my understanding that model selection must consider the specific context and potential impact of errors. Collaborating with Laiba and Faiq enhanced my teamwork skills, as we leveraged each other's strengths and addressed knowledge gaps through peer learning.

10. References

World Health Organization. (2023). *Cervical cancer*. Retrieved from <https://www.who.int/news-room/fact-sheets/detail/cervical-cancer>

Sruthi. (2025, May 1). *Random Forest algorithm in machine learning*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>

Prashant. (2020, March 13). *Random Forest Classifier tutorial*. Kaggle. <https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial>

GeeksforGeeks. (2025, January 16). *Random Forest algorithm in machine learning*. <https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/>

Rubric for Part 1

Component	10-9 (Excellent)	8-6 (Good)	5-2 (Average)	1-0 (Poor)	Weightage
Feature selection	<p>The process of feature selection is clearly implemented.</p> <p>The chosen features are clearly justified based on the implementation.</p>	<p>The process of feature selection is fairly implemented.</p> <p>The chosen features are fairly justified based on the implementation.</p>	<p>The process of feature selection is minimally implemented.</p> <p>The chosen features are minimally justified based on the implementation.</p>	<p>The process of feature selection is not implemented.</p> <p>The chosen features are not justified.</p>	1
Model construction	<p>Two models are trained.</p> <p>The models' parameters are completely fine-tuned.</p>	<p>Two models are trained.</p> <p>The models' parameters are fairly fine-tuned.</p>	<p>Two models are trained.</p> <p>The models' parameters are minimally fine-tuned.</p>	<p>The model construction and selection are poorly or not implemented, and the model performance is absent.</p>	3
Model evaluation	<p>Models are completely evaluated using appropriate performance metrics.</p> <p>Model comparison is performed and justified.</p>	<p>Models are fairly evaluated using appropriate performance metrics.</p> <p>Model comparison is performed and justified.</p>	<p>Models are minimally evaluated using appropriate performance metrics.</p> <p>Model comparison is performed.</p>	<p>Model evaluation is incomplete or not performed.</p> <p>No comparison is performed.</p>	3
Runtime and Algorithm	<p>Executes without errors.</p> <p>The algorithm and outputs are correct.</p>	<p>Executes without errors.</p> <p>The algorithm and/or outputs have minor errors.</p>	<p>Executes without errors.</p> <p>The algorithm and/or outputs are partially correct.</p>	<p>Does not execute due to error.</p> <p>Algorithm is incorrect and no output.</p>	2
Documentation	<p>The source codes are well documented and commented.</p>	<p>The source codes are partially documented and commented.</p>	<p>The source codes are minimally documented and commented</p>	<p>The source codes are not documented and commented</p>	1