P Harish Ragavender- S20220020301
S Murugan-S20220020306

# Molecular Communication and Disease Biomarker Diffusion Modeling Using Numerical Simulations and Machine Learning Classification

**P Harish Ragavender, S Murugan.**

[1,2] Electronics and Communication Engineering Indian Institute of Information Technology Sri City.

## ABSTRACT

This project explores the concept of **molecular communication** through the simulation of disease biomarker diffusion in different environmental scenarios. Molecular communication refers to the use of molecular signals to convey information across biological systems. By simulating the diffusion of disease biomarkers (acting as molecular signals) in media, the project studies the impact of varying diffusion coefficients, initial concentrations, and spatial heterogeneity. Using machine learning, specifically the Random Forest Classifier, this project classifies diffusion scenarios based on extracted statistical features from the concentration profiles of these biomarkers. This approach not only provides insights into how biomarkers spread under various conditions but also aids in advancing molecular communication models, crucial for diagnostic and therapeutic applications in the medical field.

**Keywords:** molecular communication, machine learning, Random Forest Classifier.

## I. INTRODUCTION

Molecular communication, which involves the transport of molecules (biomarkers, in this case) as signals across biological or synthetic media, plays a crucial role in systems like disease detection and therapeutic interventions. Biomarkers such as proteins, lipids, and DNA serve as indicators of disease states. Their diffusion patterns can provide crucial insights into the presence, spread, and response to disease.

This project combines numerical modeling of disease biomarker diffusion with machine learning classification to predict the type of diffusion scenario based on concentration profile features. The aim is to simulate various **molecular communication scenarios** such as high diffusion, low diffusion, anisotropic diffusion, and multiple sources of diffusion, and then classify these scenarios using the **Random Forest Classifier**. These diffusion scenarios mimic how molecular signals behave in real-life biological systems.

The integration of numerical simulations with machine learning helps to classify and understand these diffusion patterns more efficiently, ultimately advancing the application of molecular communication in diagnostics. By leveraging features like the mean, max, min, standard deviation, skewness, and kurtosis from diffusion profiles, we explore how these biomarkers communicate information about disease presence and progression.

## II. Literature Survey

Molecular communication (MC) in biological systems has been a topic of increasing interest, particularly in how it can be used to understand biological signaling and disease biomarker behavior. Early research focused on the transport of molecules through biological media, including tissues and blood. However, advancements in molecular communication systems (MCS) now focus on the precision and timing of these molecular signals, which is crucial for applications in medicine, particularly in **disease biomarker diffusion**.

For instance, **biomarker diffusion modeling** is used to simulate the movement of molecules within a medium and predict the concentration profile over time. Recent studies have used molecular communication principles to develop strategies for more accurate and faster disease detection (Martinez et al., 2019). These models take into account factors like diffusion rates, environmental heterogeneity, and multiple molecular sources.

Machine learning, particularly classification algorithms like **Random Forests**, has been used in molecular communication research to predict the behavior of molecules. By using extracted features from diffusion data, machine learning models can predict the type of diffusion scenario and offer valuable insights into disease states.

This project extends these concepts by using diffusion models and machine learning to analyze the behavior of disease biomarkers, providing insights for diagnostics.

## III. System Model / State of the Art

### A. Molecular Communication Framework

n molecular communication systems, molecules (acting as signals) spread across a medium and carry information about environmental conditions. The diffusion of these molecules can be represented using a one-dimensional diffusion equation

$$\frac{\partial \mathrm{C}(\mathrm{x},\mathrm{t})}{\partial \mathrm{t}} = \mathrm{D} \cdot \frac{\partial^2 \mathrm{C}(\mathrm{x},\mathrm{t})}{\partial \mathrm{x}^2}$$

Where C(x,t) represents the concentration of the biomarker at position $x$ x and time $t$ t, and $D$ D is the diffusion coefficient. The diffusion process is governed by molecular interaction, which dictates how molecules spread through the medium.

**Diffusion Scenarios:**

1. **High Diffusion**: Biomarkers spread quickly, simulating fast molecular communication.
2. **Low Diffusion**: Biomarkers spread slowly, modeling slower biological signal transmission.
3. **Normal Diffusion**: Represents a standard diffusion scenario with Gaussian noise in the medium.
4. **Anisotropic Diffusion**: Models diffusion that varies based on direction, resembling molecular communication in complex biological environments.
5. **Multiple Source Diffusion**: Models the case where biomarkers originate from multiple sources, representing more complex molecular signaling mechanisms.

### B. Feature Extraction for Machine Learning

To classify these diffusion scenarios, we extract several features from the concentration profiles: mean, max, min, standard deviation, skewness, and kurtosis. These features act as statistical summaries of the concentration data and help the machine learning model to distinguish between different diffusion types.

Machine Learning Model : The Random Forest Classifier is trained to classify the diffusion scenario based on these extracted features. This model is evaluated using metrics such as accuracy, confusion matrix, and misclassification analysis.

## IV. Mathematical Analysis

### A. Diffusion Equation Discretization

The one-dimensional diffusion equation is discretized using the finite difference method to model the concentration profiles over time. The equation is expressed as:

$$C_j^{n+1} = C_j^n + \frac{D \cdot \Delta t}{\Delta x^2}\left(C_{j+1}^n - 2C_j^n + C_{j-1}^n\right)$$

where:

- C(j,t) is the concentration at the spatial point j at time t.
- D is the diffusion coefficient.
- Δt is the time step.
- Δx is the spatial step.

This discretization helps to simulate how disease biomarkers diffuse over time in different scenarios.

### B. Feature Extraction

From the concentration profiles, we extract:

- **Mean**: Average concentration over the profile.
- **Max and Min**: The highest and lowest values in the profile, indicating peak concentration.
- **Standard Deviation**: Measures the spread of concentration values, indicating the variability of biomarker diffusion.

- **Skewness**: Indicates asymmetry in the concentration distribution, useful for identifying skewed diffusion patterns.
- **Kurtosis**: Measures the "tailedness" of the concentration distribution, helping to identify outliers or extreme values in the data.

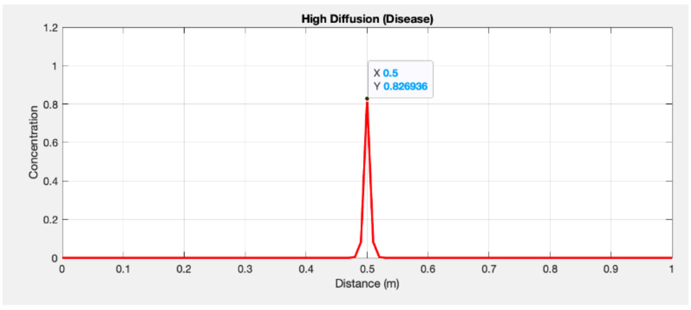
C. **Machine Learning Model**

The Random Forest Classifier is employed to predict the diffusion scenario based on these statistical features. The model is evaluated for accuracy and confusion matrix analysis.

## V. Results / Graph / Analysis

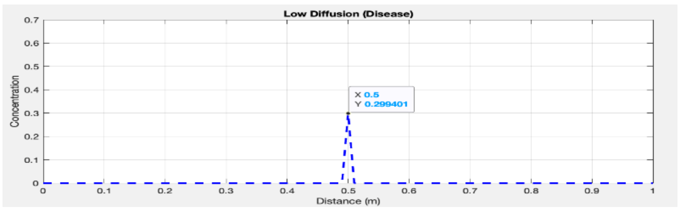
1. **High Diffusion (Disease)**

- **Description**: This case represents a scenario where the disease biomarker spreads rapidly due to high diffusion..
- **Value used**:
  - Diffusion coefficient D=1×10^-7 (high diffusion).
  - Initial concentration at the center of the domain: C0=1.



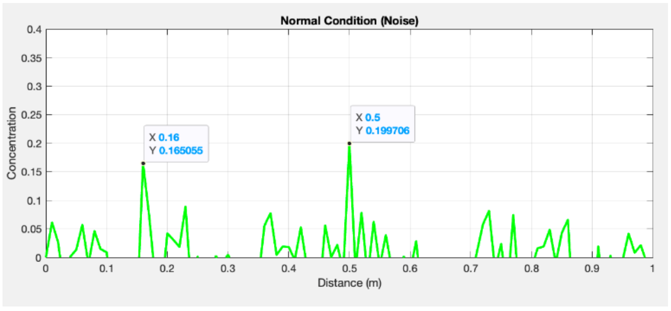
2. **Low Diffusion (Disease)**

- **Description**: This case represents a disease biomarker with slow diffusion (low D).
- **Formula**: Same formula , with a lower value for D
- **Value used**:
  - Diffusion coefficient D=1×10^-9 (low diffusion).
  - Initial concentration at the center of the domain: C0=0.3



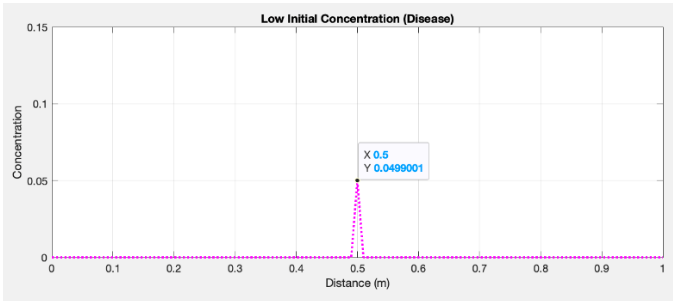
3. **Normal Condition (Noise)**

- **Description**: This represents a normal scenario where the biomarker diffusion occurs with added noise.
- **Formula**: random noise is added at each time step:
- **Value used**:
  - Diffusion coefficient D=1×10^-9 (same as low diffusion).
  - Initial concentration at the center: C0=0.2.
  - Added random noise: noise_level=0.05.



4. **Low Initial Concentration (Disease)**

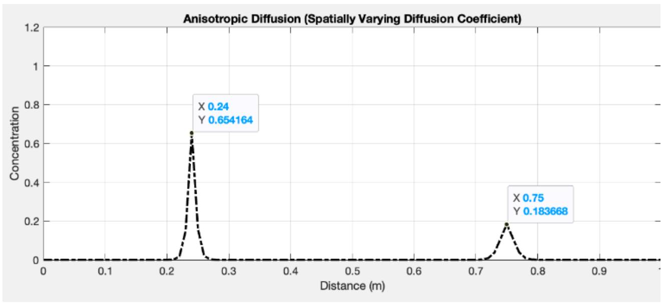
- **Description**: This case assumes an initial low concentration of the disease biomarker, which can happen in the early stages of a disease.
- **Formula**: used the same way as the other cases.
- **Value used**:
  - Diffusion coefficient D=1×10^-9 (same as low diffusion).
  - Initial concentration at the center: C0=0.05(low initial concentration).



5. **Anisotropic Diffusion**

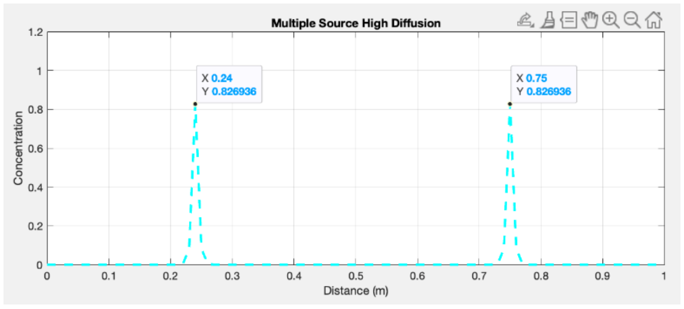
- **Description**: In this case, the diffusion coefficient D is not constant throughout the domain but varies spatially, representing **anisotropic diffusion**.
- **Formula**: The diffusion equation is modified to use a spatially varying diffusion coefficient:
- where D(j) is the diffusion coefficient at position j, which can vary across space.
- **Value used**:

- Diffusion coefficient D(j)=linspace(1×10^−9,1×10^−6,num_x)(linearly varying from low to high diffusion).
- Initial concentration at the 1/4th and 3/4th positions: C0=1 at x=L/4, C0=0.5 at x=3L/4 (two sources).



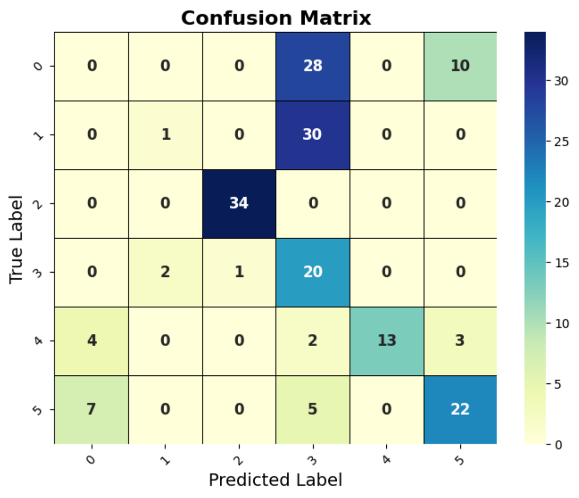
### 6. Multiple Source High Diffusion

- **Description**: In this case, multiple sources of high concentration of disease biomarkers are placed at different locations in the domain, and the diffusion spreads from these sources.
- **Formula**: The same diffusion equation is used as in other cases, but the initial concentration at multiple points is set to high values.
- **Value used**:
  - Diffusion coefficient D=1×10^−7(high diffusion).
  - Initial concentration at multiple points: C0=1 at x=L/4 and x=3L/4



### Confusion Matrix

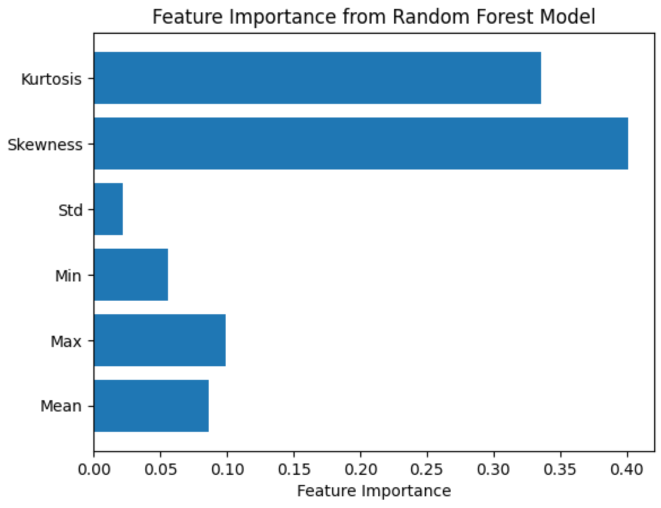
The confusion matrix is visualized using a heatmap, which shows the true vs predicted labels. It provides an insight into how well the model performed for each class of diffusion scenario. Misclassifications are observed, and the misclassification rate per class is calculated.



| True \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 28 | 0 | 10 |
| 1 | 0 | 1 | 0 | 30 | 0 | 0 |
| 2 | 0 | 0 | 34 | 0 | 0 | 0 |
| 3 | 0 | 2 | 1 | 20 | 0 | 0 |
| 4 | 4 | 0 | 0 | 2 | 13 | 3 |
| 5 | 7 | 0 | 0 | 5 | 0 | 22 |

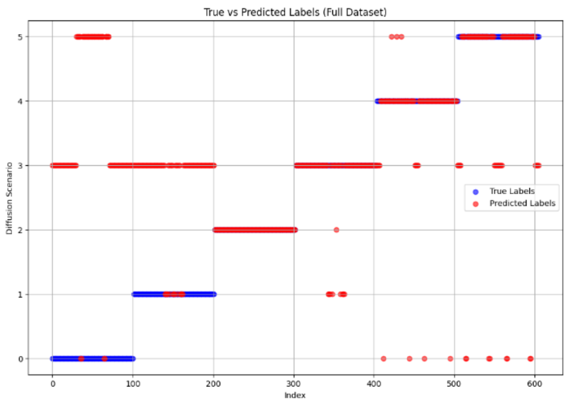
### Feature Importance

Feature importance analysis reveals that features like mean, skewness, and standard deviation are the most influential in predicting the diffusion type. This is visualized using a bar plot.

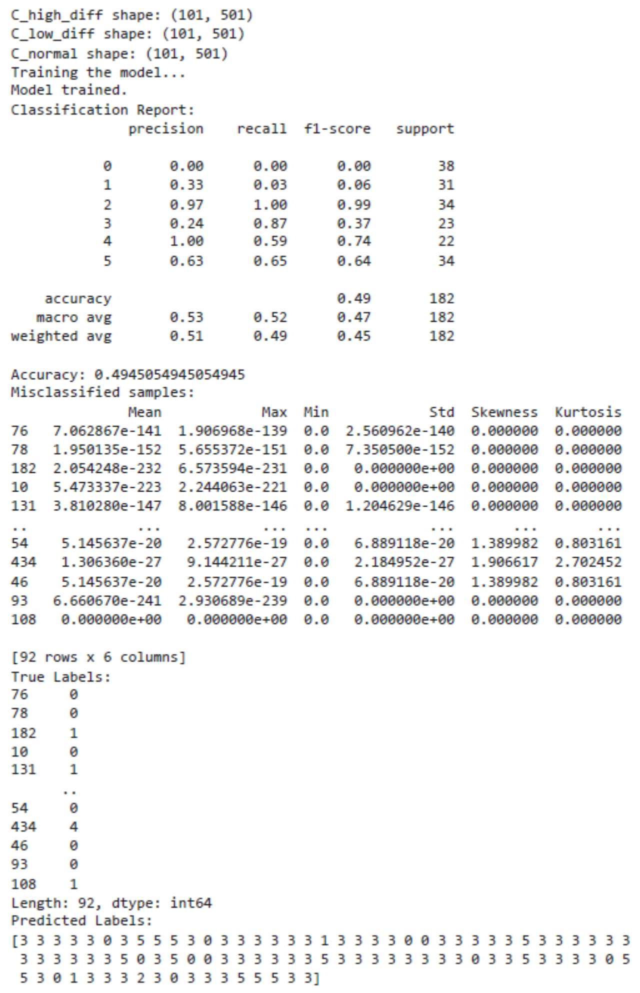


### True vs Predicted Labels

A scatter plot compares true and predicted labels for the entire dataset, illustrating how well the model generalizes across various diffusion scenarios.



## Model Performance



```
C_high_diff shape: (101, 501)
C_low_diff shape: (101, 501)
C_normal shape: (101, 501)
Training the model...
Model trained.
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        38
           1       0.33      0.03      0.06        31
           2       0.97      1.00      0.99        34
           3       0.24      0.87      0.37        23
           4       1.00      0.59      0.74        22
           5       0.63      0.65      0.64        34

    accuracy                           0.49       182
   macro avg       0.53      0.52      0.47       182
weighted avg       0.51      0.49      0.45       182

Accuracy: 0.4945054945054945
Misclassified samples:
              Mean           Max  Min           Std  Skewness  Kurtosis
76   7.062867e-141  1.906968e-139  0.0  2.560962e-140  0.000000  0.000000
78   1.950135e-152  5.655372e-151  0.0  7.350500e-152  0.000000  0.000000
182  2.054248e-232  6.573594e-231  0.0   0.000000e+00  0.000000  0.000000
10   5.473337e-223  2.244063e-221  0.0   0.000000e+00  0.000000  0.000000
131  3.810280e-147  8.001588e-146  0.0  1.204629e-146  0.000000  0.000000
..             ...            ...  ...           ...       ...       ...
54    5.145637e-20   2.572776e-19  0.0   6.889118e-20  1.389982  0.803161
434   1.306360e-27   9.144211e-27  0.0   2.184952e-27  1.906617  2.702452
46    5.145637e-20   2.572776e-19  0.0   6.889118e-20  1.389982  0.803161
93   6.660670e-241  2.930689e-239  0.0   0.000000e+00  0.000000  0.000000
108   0.000000e+00   0.000000e+00  0.0   0.000000e+00  0.000000  0.000000

[92 rows x 6 columns]
True Labels:
76     0
78     0
182    1
10     0
131    1
      ..
54     0
434    4
46     0
93     0
108    1
Length: 92, dtype: int64
Predicted Labels:
[3 3 3 3 3 0 3 5 5 5 3 0 3 3 3 3 3 3 1 3 3 3 3 0 0 3 3 3 3 3 5 3 3 3 3 3 3
 3 3 3 3 3 3 5 0 3 5 0 0 3 3 3 3 3 3 5 3 3 3 3 3 3 3 3 0 3 3 5 3 3 3 3 0 5
 5 3 0 1 3 3 3 2 3 0 3 3 3 5 5 5 3 3]
```

## VI. Conclusion

This project demonstrates how **molecular communication** and **disease biomarker diffusion** can be simulated and analyzed using numerical models and machine learning. The Random Forest Classifier successfully classifies different biomarker diffusion scenarios based on features extracted from concentration profiles. This approach contributes to the advancement of molecular communication models, providing insights into the spread of disease biomarkers and enhancing diagnostic techniques.

Future work could explore higher-dimensional diffusion models, more complex environmental factors, and the application of deep learning techniques to further improve classification accuracy. The integration of molecular communication principles with machine learning holds promise for future advancements in biomedical diagnostics and molecular systems.

```
% Define spatial domain and time step parameters
length_domain = 10; % Length of the domain (adjust this to your need)
dx = 0.1;           % Spatial step size (adjust based on resolution needed)
dt = 0.01;          % Time step size (adjust based on time resolution)

% Scenario-Specific Parameters
D_high = 1e-7; % High diffusion coefficient (for noticeable diffusion)
D_low = 1e-9;  % Low diffusion coefficient (for slow-moving molecules)
D_normal = 1e-9; % Standard diffusion coefficient (for normal case)
C0_high = 1; % High initial concentration (disease case)
C0_low_diffusion = 0.3; % Modified initial concentration for low diffusion
C0_low_init = 0.05; % Very low initial concentration (early disease)
C0_normal = 0.2; % Lower initial concentration for normal condition

% Spatial and time vectors
x = 0:dx:length_domain; % Space vector
t = 0:dt:5; % Time vector (adjust the end time as needed)

% Number of points in space and time
num_x = numel(x); % Number of spatial points
num_t = numel(t); % Number of time points

% Initialize concentration arrays for each scenario
C_high_diff = zeros(num_x, num_t); % Disease with high diffusion
C_low_diff = zeros(num_x, num_t);  % Disease with low diffusion
C_normal = zeros(num_x, num_t);    % Normal condition with noise
C_low_init = zeros(num_x, num_t);  % Disease with low initial concentration
C_anisotropic = zeros(num_x, num_t); % Anisotropic diffusion
C_multiple_high = zeros(num_x, num_t); % Multiple source high diffusion

% Initial condition (biomarker placed at the center)
C_high_diff(round(end/2),1) = C0_high;   % High diffusion (disease)
C_low_diff(round(end/2),1) = C0_low_diffusion; % Modified low diffusion
C_normal(round(end/2),1) = C0_normal;    % Normal condition
C_low_init(round(end/2),1) = C0_low_init;     % Very low initial concentration (disease)

% Add random noise to the normal condition scenario
noise_level = 0.05; % Moderate noise level
C_normal(:,1) = C_normal(:,1) + noise_level * randn(num_x, 1); % Adding noise

% Anisotropic Diffusion Parameters
D_anisotropic = linspace(1e-9, 1e-6, num_x); % Diffusion coefficient varying spatially
C_anisotropic(round(end/4),1) = 1; % Source 1
C_anisotropic(round(3*end/4),1) = 0.5; % Source 2

% Multiple Source High Diffusion Parameters
C_multiple_high(round(end/4),1) = 1;  % Source 1 (high concentration)
C_multiple_high(round(3*end/4),1) = 1; % Source 2 (high concentration)

% Diffusion simulation loop for each scenario
for i = 1:num_t-1
    for j = 2:num_x-1
        % High diffusion (disease)
        C_high_diff(j,i+1) = C_high_diff(j,i) + D_high*dt/dx^2 * ...
            (C_high_diff(j+1,i) - 2*C_high_diff(j,i) + C_high_diff(j-1,i));
        % Low diffusion (disease) - Modified initial concentration
        C_low_diff(j,i+1) = C_low_diff(j,i) + D_low*dt/dx^2 * ...
            (C_low_diff(j+1,i) - 2*C_low_diff(j,i) + C_low_diff(j-1,i));

        % Low initial concentration (disease)
        C_low_init(j,i+1) = C_low_init(j,i) + D_low*dt/dx^2 * ...
            (C_low_init(j+1,i) - 2*C_low_init(j,i) + C_low_init(j-1,i));

        % Normal condition (noise already added at the start)
        C_normal(j,i+1) = C_normal(j,i) + D_normal*dt/dx^2 * ...
            (C_normal(j+1,i) - 2*C_normal(j,i) + C_normal(j-1,i));

        % Anisotropic diffusion (varying D across space)
        C_anisotropic(j,i+1) = C_anisotropic(j,i) + D_anisotropic(j)*dt/dx^2 * ...
            (C_anisotropic(j+1,i) - 2*C_anisotropic(j,i) + C_anisotropic(j-1,i));

        % Multiple Source High Diffusion
        C_multiple_high(j,i+1) = C_multiple_high(j,i) + D_high*dt/dx^2 * ...
            (C_multiple_high(j+1,i) - 2*C_multiple_high(j,i) + C_multiple_high(j-1,i));
    end
end

% Plot results for each scenario
figure;

% High Diffusion (Disease)
subplot(3,2,1);
plot(x, C_high_diff(:,end), 'LineWidth', 2, 'Color', 'r', 'LineStyle', '-');
xlabel('Distance (m)');
ylabel('Concentration');
title('High Diffusion (Disease)');
ylim([0, 1.2]);
xlim([0, length_domain]);
grid on;

% Low Diffusion (Disease)
subplot(3,2,2);
plot(x, C_low_diff(:,end), 'LineWidth', 2, 'Color', 'b', 'LineStyle', '--');
xlabel('Distance (m)');
ylabel('Concentration');
title('Low Diffusion (Disease)');
ylim([0, 0.7]);
xlim([0, length_domain]);
grid on;

% Normal Condition (Noise)
subplot(3,2,3);
plot(x, C_normal(:,end), 'LineWidth', 2, 'Color', 'g');
xlabel('Distance (m)');
ylabel('Concentration');
title('Normal Condition (Noise)');
ylim([0, 0.4]);
xlim([0, length_domain]);
grid on;

% Low Initial Concentration (Disease)
```

```
subplot(3,2,4);
plot(x, C_low_init(:,end), 'LineWidth', 2, 'Color', 'm', 'LineStyle', ':');
xlabel('Distance (m)');
ylabel('Concentration');
title('Low Initial Concentration (Disease)');
ylim([0, 0.15]);
xlim([0, length_domain]);
grid on;

% Anisotropic Diffusion
subplot(3,2,5);
plot(x, C_anisotropic(:,end), 'LineWidth', 2, 'Color', 'k', 'LineStyle', '-.');
xlabel('Distance (m)');
ylabel('Concentration');
title('Anisotropic Diffusion (Spatially Varying Diffusion Coefficient)');
ylim([0, 1.2]);
xlim([0, length_domain]);
grid on;

% Multiple Source High Diffusion
subplot(3,2,6);
plot(x, C_multiple_high(:,end), 'LineWidth', 2, 'Color', 'c', 'LineStyle', '--');
xlabel('Distance (m)');
ylabel('Concentration');
title('Multiple Source High Diffusion');
ylim([0, 1.2]);
xlim([0, length_domain]);
grid on;

% Display overall figure
sgtitle('Diffusion Scenarios for Disease Biomarker Detection with Anisotropic↙
Diffusion');
% File paths for saving the CSV files
file_high_diff = 'C_high_diff.csv';
file_low_diff = 'C_low_diff.csv';
file_normal = 'C_normal.csv';
file_low_init = 'C_low_init.csv';
file_anisotropic = 'C_anisotropic.csv';
file_multiple_high = 'C_multiple_high.csv';

% Export each scenario to a CSV file
writematrix(C_high_diff, file_high_diff);
writematrix(C_low_diff, file_low_diff);
writematrix(C_normal, file_normal);
writematrix(C_low_init, file_low_init);
writematrix(C_anisotropic, file_anisotropic);
writematrix(C_multiple_high, file_multiple_high);

% Confirm export
disp('Simulation data exported as CSV files.');
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from scipy.stats import skew, kurtosis

# Load the CSV files containing concentration profiles
C_high_diff = pd.read_csv('/content/C_high_diff.csv', header=None)
C_low_diff = pd.read_csv('/content/C_low_diff.csv', header=None)
C_normal = pd.read_csv('/content/C_normal.csv', header=None)
C_low_init = pd.read_csv('/content/C_low_init.csv', header=None)
C_anisotropic = pd.read_csv('/content/C_anisotropic.csv', header=None)
C_multiple_high = pd.read_csv('/content/C_multiple_high.csv', header=None)

# Inspect the data shapes
print("C_high_diff shape:", C_high_diff.shape)
print("C_low_diff shape:", C_low_diff.shape)
print("C_normal shape:", C_normal.shape)

# Define a function to extract features from the concentration profiles
def extract_features(df_row):
    """
    Safely extract features from a single row of concentration profile data.
    """
    if df_row.isnull().any():  # Check for missing values
        return [np.nan] * 7  # Return placeholder values

    last_col = df_row
    mean = last_col.mean()
    max_val = last_col.max()
    min_val = last_col.min()
    std_dev = last_col.std()
    skewness = skew(last_col)
    kurt = kurtosis(last_col)

    # Check for very large or infinite values and replace with 0
    if np.isinf(mean) or np.isnan(mean):
        mean = 0
    if np.isinf(max_val) or np.isnan(max_val):
        max_val = 0
    if np.isinf(min_val) or np.isnan(min_val):
        min_val = 0
    if np.isinf(std_dev) or np.isnan(std_dev):
        std_dev = 0
    if np.isinf(skewness) or np.isnan(skewness):
        skewness = 0
    if np.isinf(kurt) or np.isnan(kurt):
        kurt = 0

    return [mean, max_val, min_val, std_dev, skewness, kurt]

# Extract features from all datasets
features = []
labels = []

# High Diffusion
labels.extend([0] * len(C_high_diff))
features.extend(C_high_diff.apply(lambda row: extract_features(row), axis=1))

# Low Diffusion
labels.extend([1] * len(C_low_diff))
features.extend(C_low_diff.apply(lambda row: extract_features(row), axis=1))

# Normal Diffusion
labels.extend([2] * len(C_normal))
features.extend(C_normal.apply(lambda row: extract_features(row), axis=1))

# Low Initial Concentration
labels.extend([3] * len(C_low_init))
features.extend(C_low_init.apply(lambda row: extract_features(row), axis=1))

# Anisotropic Diffusion
labels.extend([4] * len(C_anisotropic))
features.extend(C_anisotropic.apply(lambda row: extract_features(row), axis=1))

# Multiple Source High Diffusion
labels.extend([5] * len(C_multiple_high))
features.extend(C_multiple_high.apply(lambda row: extract_features(row), axis=1))
```

```
# Convert to pandas DataFrame
features_df = pd.DataFrame(features, columns=['Mean', 'Max', 'Min', 'Std', 'Skewness', 'Kurtosis'])
labels_df = pd.Series(labels)

# Check for inf or NaN values in the features
if np.any(np.isinf(features_df)) or np.any(np.isnan(features_df)):
    print("Features contain NaN or Inf values. Replacing with column mean.")
    features_df = features_df.replace([np.inf, -np.inf], np.nan).fillna(features_df.mean())

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_df, labels_df, test_size=0.3, random_state=42)

# Check for inf or NaN values in X_train and X_test
if np.any(np.isinf(X_train)) or np.any(np.isnan(X_train)):
    print("X_train contains NaN or Inf values. Replacing with column mean.")
    X_train = X_train.replace([np.inf, -np.inf], np.nan).fillna(X_train.mean())

if np.any(np.isinf(X_test)) or np.any(np.isnan(X_test)):
    print("X_test contains NaN or Inf values. Replacing with column mean.")
    X_test = X_test.replace([np.inf, -np.inf], np.nan).fillna(X_test.mean())

# Initialize the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
print("Training the model...")
clf.fit(X_train, y_train)
print("Model trained.")

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model's performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

# Misclassification Analysis:
misclassified_samples = X_test[y_test != y_pred]
misclassified_labels = y_test[y_test != y_pred]
misclassified_preds = y_pred[y_test != y_pred]

print(f"Misclassified samples:\n{misclassified_samples}")
print(f"True Labels:\n{misclassified_labels}")
print(f"Predicted Labels:\n{misclassified_preds}")

# Confusion Matrix with colorful visualization
cm = confusion_matrix(y_test, y_pred)

# Create a more vibrant and informative heatmap using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='YlGnBu', cbar=True,
            xticklabels=np.unique(labels), yticklabels=np.unique(labels),
            linewidths=0.5, linecolor='black', annot_kws={'size': 12, 'weight': 'bold'})

# Titles and labels
plt.title('Confusion Matrix', fontsize=16, weight='bold')
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.xticks(rotation=45)
plt.yticks(rotation=45)

# Display the plot
plt.show()

# Misclassification Rate per Class
misclassification_rate = 1 - np.diagonal(cm) / np.sum(cm, axis=1)
print(f"Misclassification rate per class:\n{misclassification_rate}")

# Visualize the feature importance
feature_importances = clf.feature_importances_
plt.barh(features_df.columns, feature_importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importance from Random Forest Model')
plt.show()

# Predict on the full dataset (for visualization purposes)
y_full_pred = clf.predict(features_df)

# Visualize true vs predicted labels
plt.figure(figsize=(12, 8))
plt.scatter(range(len(labels)), labels, color='blue', label='True Labels', alpha=0.6)
```