#This section of code is designed to classify data from a metamaterial sensor setup. We have four sensors, though you can easily adjust that number. The code will specifically extract the peak values of the S_{21} parameter. Each sensor has four resonating frequencies, and the code will classify these four frequency peaks into different categories or labels.#

 # This section of the code is divided into five parts, each corresponding to a different algorithm. All of these algorithms perform the same classification task, but they use different methods, such as neural networks, support vector machines (SVM), and kernel-based methods (KVM). While their underlying approaches differ, they all aim to achieve the same classification goal, with varying levels of accuracy.#

#We have utilized this code in Google Colab for several publications. You can find the references to these papers below, where you can see how the code was applied. If you find it useful, please feel free to cite these papers in your own work.#

#" Machine learning-optimized metamaterial sensor for real-time insoluble particle position monitoring in multiphase oil flow" DOI: 10.1109/TIM.2024.3500059#

Neural network

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import keras

# Path to your new 2-element dataset
csv_path = '/content/MainDataBS.xlsx'

# Load dataset - now with 2 features and 1 label
df = pd.read_excel(csv_path, header=None, names=["Sensor1", "Sensor2", "Label"])

# Preprocess data
X = df.iloc[:, :-1].values  # Features (2 columns)
y = df.iloc[:, -1].values   # Labels

# Encode labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Normalize input data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)
```

```
# Determine number of classes for the output layer
num_classes = len(np.unique(y))

# Build the neural network model
model = Sequential()
model.add(Dense(64, input_dim=2, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])

# Train the model with early stopping
early_stopping = EarlyStopping(patience=10, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=100, batch_size=2, validation_data=(X_test,
y_test), callbacks=[early_stopping])

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy*100:.2f}%")

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.tight_layout()
plt.show()

# Predict classes
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Confusion Matrix
conf_mat = confusion_matrix(y_test, y_pred)

# Classification Report
class_labels = label_encoder.classes_.astype(str)
class_report = classification_report(y_test, y_pred, target_names=class_labels,
zero_division=1)
print("Classification Report:\n", class_report)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Save the model
model.save('sensor_model.h5')
model.save('my_model.keras')
```

SVM

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Path to your 2-element dataset
csv_path = '/content/MainDataBS.xlsx'

# Load the dataset: now with 2 features and 1 label
df = pd.read_excel(csv_path, header=None, names=["Sensor1", "Sensor2", "Label"])

# Preprocess data
X = df.iloc[:, :-1].values  # Features (2 columns)
y = df.iloc[:, -1].values   # Labels

# Convert labels to numerical values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
class_labels = label_encoder.classes_.astype(str)

# Normalize input data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Build and train the SVM model
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)
```

```
# Evaluate the SVM model
accuracy = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, target_names=class_labels)

# Display the results
print(f"Test Accuracy: {accuracy*100:.2f}%")
print("Confusion Matrix:\n", conf_mat)
print("Classification Report:\n", class_report)

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.tight_layout()
plt.show()

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

KNN

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Path to your 2-element dataset
csv_path = '/content/MainDataBS.xlsx'

# Load the dataset - now with 2 features and 1 label
df = pd.read_excel(csv_path, header=None, names=["Sensor1", "Sensor2", "Label"])

# Preprocess data
X = df.iloc[:, :-1].values  # Features (2 columns)
y = df.iloc[:, -1].values   # Labels

# Convert labels to numerical values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Normalize input data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

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

```
# Convert label_encoder.classes_ to a list of strings
class_labels_str = [str(label) for label in label_encoder.classes_]

# Evaluate the model
class_report = classification_report(y_test, y_pred, target_names=class_labels_str)
accuracy = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)

# Display the results
print("\nAccuracy: {:.2f}%".format(accuracy * 100))
print("Confusion Matrix:\n", conf_mat)
print("\nClassification Report:\n", class_report)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues",
        xticklabels=label_encoder.classes_,
        yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.tight_layout()
plt.show()
```

Naive Bayes

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Path to your 2-element dataset
csv_path = '/content/MainDataBS.xlsx'

# Load the dataset - now with 2 features and 1 label
df = pd.read_excel(csv_path, header=None, names=["Sensor1", "Sensor2", "Label"])

# Preprocess data
X = df.iloc[:, :-1].values  # Features (2 columns)
y = df.iloc[:, -1].values   # Labels

# Convert labels to numerical values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Normalize input data (optional but common)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Build the Naive Bayes classifier
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train, y_train)

# Predict on the test set
```

```
y_pred = naive_bayes_classifier.predict(X_test)

# Convert label_encoder.classes_ to strings
class_labels_str = [str(label) for label in label_encoder.classes_]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, target_names=class_labels_str)

# Display the results
print("\nAccuracy: {:.2f}%".format(accuracy * 100))
print("Confusion Matrix:\n", conf_mat)
print("\nClassification Report:\n", class_report)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues",
        xticklabels=class_labels_str, yticklabels=class_labels_str)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()



# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.tight_layout()
plt.show()
```

Decision tree

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text, export_graphviz
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from io import StringIO
import pydotplus
from IPython.display import Image

# Path to your 2-element dataset
csv_path = '/content/MainDataBS.xlsx'

# Load the dataset - now with 2 features and 1 label
df = pd.read_excel(csv_path, header=None, names=["Sensor1", "Sensor2", "Label"])

# Preprocess data
X = df.iloc[:, :-1].values  # Features (2 columns)
y = df.iloc[:, -1].astype(str).values  # Labels as strings

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# Build the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")

# Confusion Matrix
```

```
conf_mat = confusion_matrix(y_test, y_pred, labels=model.classes_)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues',
        xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Classification Report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)

# OPTIONAL: Plot Decision Tree
dot_data = StringIO()
export_graphviz(model, out_file=dot_data, filled=True,
            feature_names=["Sensor1", "Sensor2"],
            class_names=model.classes_)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision_tree.png')  # Save the tree as an image

# Display the Decision Tree plot
Image(filename='decision_tree.png')

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.tight_layout()
plt.show()
```