# Practical 3: Breast Cancer Classification Task

COMP SCI 1400 AI Technologies

## 1 Introduction

In this practical, we will explore how to use the Scikit-learn library, a popular tool in both industry and academia for machine learning tasks. Specifically, we will perform breast cancer classification using two different models: Support Vector Classifier (SVC) and Random Forest Classifier (RFC). We will also evaluate their performance using various metrics such as accuracy, F1-score, and AUC-ROC score.

## 2 Required Libraries and Modules

To complete this task, we will need to import several libraries and modules from Scikit-learn. Below is the Python code snippet that includes all necessary imports:

Listing 1: Importing Required Libraries and Modules

```
# Importing libraries and modules
from sklearn import datasets
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from statistics import mean
```

### 2.1 Explanation of Imported Modules

- **datasets**: A module in Scikit-learn that provides access to a variety of datasets, including the breast cancer dataset used in this task.
- **load_breast_cancer**: A function within the datasets module to load the breast cancer dataset.
- **KFold**: A module that provides cross-validation splitting strategies, in this case, K-Fold Cross Validation.

- **SVC**: A module for creating a Support Vector Classifier.
- **RandomForestClassifier**: A module for creating a Random Forest Classifier.
- **accuracy_score**: A function that computes the accuracy of the classification model.
- **f1_score**: A function that computes the F1-score, which is the harmonic mean of precision and recall.
- **roc_auc_score**: A function that computes the Area Under the Receiver Operating Characteristic Curve (AUC-ROC).
- **mean**: A function from the statistics module to calculate the mean of a list of numbers.

## 3 Obtaining the Training and Test Datasets

In this section, we will demonstrate how to split the breast cancer dataset into training and test datasets using 10-fold cross-validation. This ensures that the dataset is divided into 10 equal parts, and each part is used once as a test set while the remaining nine parts form the training set.

Listing 2: Splitting the Dataset into Training and Test Sets

```
# Load the breast cancer dataset and extract the features (X) and target labels
X_features = load_breast_cancer().data
y_labels = load_breast_cancer().target

# Initialize a K-Fold cross-validator with 10 splits for cross-validation
kfold_10_splits = KFold(n_splits=10)

# Iterate over each fold generated by the K-Fold cross-validator
for train_indices, test_indices in kfold_10_splits.split(X_features):

    # Create training and testing sets using the current fold's indices
    X_train, X_test = X_features[train_indices], X_features[test_indices]
    y_train, y_test = y_labels[train_indices], y_labels[test_indices]
```

### 3.1 Conceptual Explanation

- **load_breast_cancer**: This function loads the breast cancer dataset into two arrays: *X_features* (input features) and *y_labels* (target labels). The dataset consists of features related to malignant and benign breast cancer cases, which are used to classify each case.

- **KFold(n_splits=10)**: This initializes a K-Fold cross-validator with 10 splits, dividing the dataset into 10 equal parts. During each iteration, one part is used as the test set, while the remaining nine parts are used for training. This process is repeated 10 times to ensure that each part of the dataset is used as the test set exactly once.
- **kfold_10_splits.split(X_features)**: This method generates indices to split the dataset into training and test sets according to the number of splits specified (10 in this case). The indices are used to create training and testing datasets for each fold.
- **X_train, X_test, y_train, y_test**: These arrays represent the training and test datasets for each fold in the cross-validation process. (**X_train**, **y_train**) make up the labelled training dataset, where **X_train** contains the input features and **y_train** contains the corresponding target labels. Similarly, (**X_test**, **y_test**) form the labelled test dataset, with **X_test** holding the features and **y_test** providing the associated labels. This separation is critical for training the model on one portion of the data while evaluating its performance on an unseen portion, thereby helping to assess the model's generalisation ability.

## 4 Constructing Classifiers and Evaluating Their Performance

In this section, you will complete Task 3 of Assignment 1, which involves building and evaluating machine learning models using Scikit-learn. For part 3 of Task 3, you are required to implement a second classifier in addition to the Support Vector Classifier (SVC). While you may choose to use a Random Forest Classifier as suggested in this practical, you are also encouraged to explore other classifiers that may be suitable for the breast cancer classification task. The choice is yours, providing an opportunity for open-ended experimentation and learning.

Here are the links to resources that provide good introductions to each of the functions imported for Task 3:

- **SVC (Support Vector Classifier)**
  - Documentation: Scikit-learn: Support Vector Machines
  - Tutorial: SVM with Scikit-learn - A Complete Guide to Support Vector Classifier
- **RandomForestClassifier**
  - Documentation: Scikit-learn: Random Forests
  - Tutorial: Random Forest Classification with Scikit-learn
- **accuracy_score**

  - Documentation: Scikit-learn: Accuracy Score

- **f1_score**
  - Documentation: Scikit-learn: F1 Score
- **roc_auc_score**
  - Documentation: Scikit-learn: ROC AUC Score
- **mean (from the statistics module)**
  - Documentation: Python: statistics.mean