Assignment 1

● Graded

Student

Manish Kumar Krishne Gowda

Total Points

100 / 100 pts

Question 1

Section 2 Load data

10 / 10 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Incorrect variable names
- **– 10 pts** Incorrect loader/No submission

Question 2

Section 3 Split data into training and test sets and normalize data

20 / 20 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** No random seed
- **– 5 pts** Incorrect test_size
- **– 10 pts** Incorrect data split
- **– 10 pts** Incorrect data transform
- **– 20 pts** No submission

Question 3

Section 4 Initialize and train the logistic regression model

40 / 40 pts

- ✔ **– 0 pts** Above 55%
- **– 5 pts** No/incorrect solver
- **– 10 pts** Incorrect data fitting
- **– 40 pts** Incorrect model/no submission

Question 4

Section 5 Evaluate with built-in function

10 / 10 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Incorrect prediction/Incorrect accuracy calculation
- **– 10 pts** Incorrect calculation/No submission

Question 5

Section 6 Implement your own accuracy calculation and compare

20 / 20 pts

✔ **− 0 pts** Correct

− 10 pts Incorrect calculation of correct prediction

− 10 pts Incorrect calculation of accuracy

− 20 pts Incorrect calculation/No submission

Question 6

Admin only (for late penalty) - assign to the first page

0 / 0 pts

✔ **− 0 pts** Correct

− 10 pts One Day Late

− 20 pts More than one day late

− 0 pts override late penalty

− 100 pts More than two days late

No questions assigned to the following page.

ECE 57000 Assignment 1 Exercises

Name: Manish Kumar

Important submission information

1. Follow the instructions in the provided "uploader.ipynb" to convert your ipynb file into PDF format.
2. Please make sure to select the corresponding pages for each exercise when you submitting your PDF to Gradescope. Make sure to include both the **output** and the **code** when selecting pages. (You do not need to include the instruction for the exercises)

We may assess a 20% penalty for those who do not correctly follow these steps.

1. Background

In this assignment, we will explore the application of logistic regression to a binary classification problem in the field of medical diagnostics. The objective is to predict whether a breast tumor is benign or malignant based on features extracted from digitized images of fine needle aspirate (FNA) of breast mass.

The dataset used is the Breast Cancer dataset from the UCI Machine Learning Repository, incorporated into scikit-learn as `load_breast_cancer`. This dataset includes measurements from 569 instances of breast tumors, with each instance described by 30 numeric attributes. These features include things like the texture, perimeter, smoothness, and symmetry of the tumor cells.

You will split the data into training and test sets, with 80% of the data used for training and the remaining 20% for testing. This setup tests the model's ability to generalize to new, unseen data. We set the `random_state` as 42 to ensure reproducibility. The logistic regression model, initialized with the 'liblinear' solver, will be trained on the training set.

Your tasks include training the model, predicting tumor classifications on the test set, and then calculating the accuracy of these predictions. You will calculate the accuracy both manually and using scikit-learn's built-in `accuracy_score` function, and then verify if both methods yield the same result.

The primary goal of this assignment is to familiarize you with logistic regression in a practical, real-world setting, and to understand the general machine learning workflows.

Question assigned to the following page: 1

2. Load data (10/100 points)

You can load the Breast Cancer dataset by using this function from the `sklearn.datasets` module (we have imported the function for you). Refer to the official documentation to understand more about this function.

Implement the Following:

1. `data` : Use the built-in function to load the dataset and store it in this variable.
2. `X` : This should store the feature matrix from the dataset.
3. `y` : This should store the target vector, which includes the labels indicating whether the tumor is benign or malignant.

Make sure to write your code between the comments `<Your code>` and `<end code>`. After implementing, the dimensions of `X` and `y` will be printed out to confirm correct data handling.

In [1]:
```
from sklearn.datasets import load_breast_cancer

# <Your code>
# Load the dataset
data = load_breast_cancer()

# Extract the feature matrix (X) and target vector (y)
X = data.data
y = data.target
# <end code>

print(f'The data has a shape of {X.shape}, and the target has a shape of {y.shape}')
```

```
The data has a shape of (569, 30), and the target has a shape of (569,)
```

3. Split data into training and test sets and normalize data (20/100 points)

Part 1: Splitting the Dataset

Use the function `train_test_split()` from the `sklearn.model_selection` module to divide your data into training and testing sets. This is crucial for evaluating your model on unseen data.

Implement the Following:

1. `X_train, X_test, y_train, y_test` : Split `X` and `y` into training and testing sets.
    - Set `test_size` to 0.2, allocating 20% of the data for testing.
    - Use `random_state=42` to ensure that your results are reproducible.

Question assigned to the following page: 2

Ensure your code is placed between the first set of comments `<Your code>` and `<end code>`. After running your code, the output will indicate the number of datapoints in your training and test sets.

Part 2: Normalizing the Dataset

Normalize the training and testing sets using `MinMaxScaler` from the `sklearn.preprocessing` module. This step ensures that the feature values are scaled to a uniform range, which is beneficial for many machine learning algorithms.

Implement the Following:

1. Initialize and fit a `MinMaxScaler` on `X_train` to learn the scaling parameters.
2. `X_train, X_test` : Transform both `X_train` and `X_test` using the fitted scaler to scale the data to the range [0, 1] and assign the normalized variables to the variable names.

Place your code for this part between the second set of comments `<Your code>` and `<end code>`. After implementation, check the maximum and minimum values of the scaled training and test data to verify successful normalization.

In [4]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# <Your code>
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
# <end code>

print(f'The training set has {X_train.shape[0]} datapoints and the test set has {X_tes

# <Your code>
# Step 2: Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Step 3: Fit the scaler on the training data to learn the scaling parameters
scaler.fit(X_train)

# Step 4: Transform both X_train and X_test using the fitted scaler
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# <end code>

print(f'The max of training data is {X_train.max():.2f} and the min is {X_train.min():
```

```
The training set has 455 datapoints and the test set has 114 datapoints.
The max of training data is 1.00 and the min is 0.00.
```

4. Initialize and train the logistic regression model (40/100 points)

You will initialize and train a logistic regression model using the `LogisticRegression` class from the `sklearn.linear_model` module. Read the official documentation to understand

Questions assigned to the following page: 3 and 4 more about the function's parameters and usage.

Implement the Following:

1. `model` : Instantiate the `LogisticRegression` class with the `liblinear` solver, and assign it to this variable. There is no need to specify other parameters and we will use the defaults.
2. Use the `fit` method of `model` to train it on `X_train` and `y_train` . This method adjusts the model parameters to best fit the training data.

Ensure your code is placed between the comments `<Your code>` and `<end code>` . This structure is intended to keep your implementation organized and straightforward.

In [6]:
```
from sklearn.linear_model import LogisticRegression

# <Your code>
# Step 1: Instantiate the LogisticRegression model with the 'liblinear' solver
model = LogisticRegression(solver='liblinear')

# Step 2: Train the model on the training data (X_train and y_train)
model.fit(X_train, y_train)
# <end code>
```

Out[6]:
```
▾        LogisticRegression
LogisticRegression(solver='liblinear')
```

5. Evaluate with built-in function (10/100 points)

To evaluate the performance of your trained logistic regression model, you will use the function `accuracy_score()` from the `sklearn.metrics` module. This function computes the accuracy, the fraction of correctly predicted instances, of the model. Check the official documentation to better understand how this function works.

Implement the Following:

1. `predictions` : Use the `predict` method of your trained `model` to make predictions on the test set `X_test` , store the predicted results in this variable.
2. `accuracy` : Calculate the accuracy of these predictions by comparing them to the actual labels `y_test` using the `accuracy_score` function.

In [8]:
```
from sklearn.metrics import accuracy_score

# <Your code>
# Step 1: Make predictions on the test set X_test using the trained model
predictions = model.predict(X_test)

# Step 2: Calculate the accuracy by comparing the predictions to the actual labels y_t
```

Questions assigned to the following page: 4 and 5

```
accuracy = accuracy_score(y_test, predictions)
# <end code>

print(f'The accuracy is {accuracy:.4f}')
```

```
The accuracy is 0.9737
```

6. Implement your own accuracy calculation and compare (20/100 points)

In this task, you will manually calculate the accuracy of your logistic regression model's predictions to better understand the underlying computation.

Task: Calculate the accuracy manually and store the result in the variable named `my_accuracy`. Compare your calculated accuracy to the previously obtained accuracy from the built-in function to ensure consistency.

Hint: Count how many predictions exactly match the actual labels (`y_test`) and divide this number by the total number of predictions to get the accuracy.

In [9]:

```
# <Your code>

# Step 1: Count the number of correct predictions
correct_predictions = sum(predictions == y_test)

# Step 2: Calculate the total number of predictions
total_predictions = len(y_test)

# Step 3: Calculate the accuracy manually
my_accuracy = correct_predictions / total_predictions

# <end code>

print(f'My accuracy is {my_accuracy:.4f} and the accuracy calculated from built-in fur
```

```
My accuracy is 0.9737 and the accuracy calculated from built-in function is 0.9737
```