Dataset: Breast Cancer Diagnostic Data

For this assignment, we will use the **Breast Cancer Wisconsin (Diagnostic)** dataset from `scikit-learn`.

Each observation corresponds to a tumor sample obtained from a fine needle aspirate (FNA) biopsy.

- **Response variable**
  - `1`: Malignant tumor
  - `0`: Benign tumor
- **Predictors**
  - 30 continuous features describing properties of cell nuclei
  - Examples include radius, texture, perimeter, area, smoothness, and concavity
  - Features are reported as means, standard errors, and "worst" values

The dataset contains 569 observations and no missing values.

Problem 1

Load the breast cancer dataset like this:

```
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
X = bc.data
y = bc.target
```

(a) Split the data into training and test sets using a fixed random seed. Then, standardize the predictors using the training data.

(b) We don't typically standardize data when building a decision tree, but it is necessary when using gradient-based optimization methods, like in logistic regression. Why is it necessary for one but not the other?

(c) Explain why the standardization parameters should be computed using only the training data, and not the full dataset.

(d) Let $y_i \in \{0, 1\}$ and suppose

$$P(y_i = 1 \mid X_i) = \sigma(\Sigma_{k=1}^{l} w_k x_{k,i}), \quad \sigma(z) = \frac{1}{1 + e^{-z}},$$

where X_i is the set of feature values $\{x_{1,i}, ..., x_{l,i}\}$ for observation i and w's represent the weights/coefficients for each feature.

Write the log-likelihood for a dataset $\{(X_i, y_i)\}_{i=1}^{n}$.

(e) Explain why minimizing the cross-entropy loss is equivalent to maximizing this log-likelihood. Your explanation should be mathematically precise, but a full symbolic derivation is not required.

(f) Implement (binary) logistic regression from scratch using gradient descent.

Your implementation should:

- initialize coefficients

- compute the cross-entropy loss
- compute the gradient of the loss (see lecture slides)
- update coefficients using gradient descent
- track the loss over iterations
- include some reasonable stopping criterion

It may be useful to create more than one function (some 'helper' functions) and call these in your main implementation function.

(g) Fit your model on the standardized training data and plot the loss as a function of iteration. Describe what this plot tells you about convergence.

(h) Fit a logistic regression model using `sklearn` without regularization (remember C is the regularization parameter in `sklearn`'s implementation). Compare the coefficients and final training loss values obtained from your gradient descent implementation to those from `sklearn`. Explain why the results may differ slightly even though both methods optimize the same objective.

(i) Suppose the coefficients of your logistic regression model satisfy the prior:

$$w_j \sim \mathcal{N}(0, \tau^2).$$

Write the log-prior for w.

(j) Explain how maximizing the posterior leads to an L2-regularized optimization problem.

(k) Explain how the prior affects the bias–variance tradeoff.

Problem 2

(a) Write code to compute the following impurity measures for binary classification:

- Entropy
- Gini impurity
- Information gain

Your functions should operate on binary response vectors and be usable for evaluating candidate splits in a decision tree.

(b) Using the (unstandardized) training data:

- Choose two candidate split features and with corresponding split values.
- Compute the information gain using both entropy and Gini imputity for each candidate split and select the best one.
- At the next depth level, choose two new candidate split features and values, and repeat the process.

This produces a depth-2 decision tree using a **greedy splitting procedure**.

A *greedy algorithm* makes the locally optimal choice at each step, without considering future consequences.

(c) Explain how this greedy procedure differs from gradient-based optimization methods such as those used for logistic regression.

Problem 3

(a) Use both `GridSearchCV` and `RandomizedSearchCV` to tune a decision tree classifier.

- Select the same set of hyperparameters for both methods.
- Try exactly 4 values for each hyperparameter in your search space.
- Run `RandomizedSearchCV` with `n_iter` equal to the total number of fits used by your grid search, so that both methods do the same number of model fits.
- Compare the selected models and validation performance.
- Discuss differences in runtime and results.

(b) Now, instead of pre-pruning the model, compute the cost-complexity pruning path and plot model performance as a function of α.

(c) Select an appropriate value of α and evaluate the pruned tree.

(d) Compare cost-complexity pruning in decision trees to regularization in logistic regression. How do both methods control model complexity?