# Assignment 3

## Assignment 3: Probabilistic models and decision trees

## Assignment Overview:

This assignment focuses on probabilistic and tree-based models for classification. You will implement generative probabilistic classifiers and decision-tree-based models from scratch, evaluate them on real datasets, and compare their performance and modelling assumptions.

### Learning Objectives:

- Probabilistic Modelling: Implement a Gaussian generative classifier and a Gaussian Naive Bayes classifier.
- Tree-Based Models: Implement a decision tree classifier for continuous features.
- Ensemble Methods: Extend your decision tree implementation to a random forest (bonus).
- Model Evaluation: Use consistent train/validation/test split and report standard classification metrics.
- Comparative Analysis: Analyze and compare the behavior, strengths, and limitations of different models.

### Problem Statement:

You will build and compare several classifiers on standard benchmark datasets. First, you will implement a Gaussian generative model with a shared covariance matrix and a Gaussian Naive Bayes classifier for handwritten digit recognition. Then, you will implement a decision tree classifier and, as a bonus, a random forest for breast cancer diagnosis. You will use a 70/15/15 train/validation/test split, perform basic hyperparameter tuning using the validation set, and report performance on the held-out test set.

---

## Part A: Probabilistic Gaussian Generative Classifier (20 points)

**A1. Dataset and Setup**

Dataset: Handwritten Digits

- Use: sklearn.datasets.load_digits()
- This dataset contains small 8×8 grayscale images of handwritten digits.
- Each image is flattened into a 64-dimensional feature vector.
- There are 10 classes: digits 0, 1, 2, …, 9.

Data splitting:

1. Load the full dataset (features X and labels y).
2. Create a 70% / 15% / 15% split:
    - 70% training set
    - 15% validation set
    - 15% test set
3. Use a stratified split, so that the proportion of each digit class is similar in all three sets.
4. standardize the features (for example, using StandardScaler from scikit-learn).

## A2. Gaussian Generative Model

You will implement a generative classifier that models how the data is generated in each class and then uses Bayes' rule for prediction.

Model assumptions:

- Class priors:
  $p(y = k) = \pi_k$ for each class k in {0, 1, …, 9}.
- Class-conditional distribution:
  $p(x \mid y = k) = N(x ; \mu_k, \Sigma)$

where:

- $\mu_k$ is the mean vector for class k (a 64-dimensional vector),
- $\Sigma$ is a single shared covariance matrix used by all classes.

Parameter estimation (on the training set):

1. Estimate the class priors $\pi_k$:
   - For each class k, count how many training samples belong to that class.
   - $\pi_k$ = (number of training samples in class k) / (total number of training samples).
2. Estimate the class means $\mu_k$:
   - For each class k, compute the average of all training samples with label k.
   - This gives one 64-dimensional mean vector for each digit class.
3. Estimate the shared covariance matrix $\Sigma$:
   - Use all training samples.
   - For each training sample $x_i$ with label $y_i$, compute the difference $(x_i - \mu_{y_i})$.
   - Accumulate $(x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$ over all training samples.
   - Divide by the total number of training samples to obtain $\Sigma$.
4. Regularise the covariance:
   - To ensure $\Sigma$ is numerically stable and invertible, you must add a small regularisation term $\lambda$ to the diagonal:
     $\Sigma_\lambda = \Sigma + \lambda I$
   - Here $\lambda > 0$ is a hyperparameter that you will tune using the validation set.

Prediction rule:

- For a new input x:
  - Compute, for each class k, a score proportional to:
    $\log p(y = k \mid x) \propto \log \pi_k + \log N(x ; \mu_k, \Sigma_\lambda)$
  - You do not need the exact probability, only scores that are comparable across classes.
  - Predict the class k with the largest score.

You must implement this model yourself

---

## A3. Hyperparameter Tuning and Evaluation

Hyperparameter: $\lambda$ (covariance regularization strength)

- $\lambda$ controls how strong covariance regularisation is.
- If $\lambda$ is too small, the covariance may be poorly conditioned, and the model may overfit.

- If λ is too large, the covariance becomes too "smooth" and the model may underfit.

Steps:

1. Choose a set of candidate values for λ, for example:
   λ ∈ { 1e-4, 1e-3, 1e-2, 1e-1 }
2. For each candidate value of λ:
   a) Train the Gaussian generative classifier on the training set.
   b) Evaluate its accuracy on the validation set.
   c) Record the validation accuracy.
3. Select the value of λ that gives the best validation accuracy.
4. Final model:
   - Combine the training and validation sets into a single training set.
   - Retrain your Gaussian generative classifier on this combined set using the chosen λ.
   - Evaluate this final model on the test set only once.
5. For the final model, you must report:
   - Test accuracy
   - Macro-averaged precision
   - Macro-averaged recall
   - Macro-averaged F1-score
   - The confusion matrix over the 10 digit classes

---

## A4. What to Include in Your Report for Part A

In your report, for Part A you should include:

1. A short explanation (in your own words) of:
   - The generative model you used: what you assume for p(y) and for p(x | y).
   - How you estimate the parameters π_k, μ_k, and Σ from the training data.
   - Why you need to regularize the covariance (Σ_λ = Σ + λ I) and what effect λ has.
2. A table showing, for each tested value of λ:
   - The corresponding validation accuracy.
3. A summary of the final test results:
   - Test accuracy
   - Macro precision, macro recall, macro F1-score
   - Confusion matrix (for example as a table or as a figure)
4. A short discussion (about 1–2 paragraphs):
   - Which digits are most often confused with each other.
   - How much the choice of λ affected performance.
   - Any observations about the strengths and weaknesses of this Gaussian generative model on the digits dataset.

---

# Part B: Naïve Bayes (20 points)

## B1. Dataset and Setup

**Primary Dataset: Use the Adult Income Dataset (Census Income)**

- Features: Mix of categorical features (workclass, education, marital-status, etc.)
- Target: Income >50K or ≤50K
- Focus: Use only discrete/categorical features for this part

**Data Processing:**

- Select relevant categorical features
- Handle missing values by treating them as a separate category
- Encode categories as integers while maintaining interpretability
- Create 70/15/15 train/validation/test split
- Analyze class distributions and feature-target relationships

## B2. Naïve Bayes Implementation

**Mathematical Implementation:**

- Class Priors: $P(C_k) = (\text{count of samples in class } k + \alpha) / (\text{total samples} + \alpha * \text{num\_classes})$
- Feature Likelihoods: $P(x_i|C_k) = (\text{count of feature value } i \text{ in class } k + \alpha) / (\text{count of class } k + \alpha * \text{num\_feature\_values})$
- Prediction: $\text{argmax}_k P(C_k) * \Pi P(x_i|C_k)$

## B3. Analysis and Evalutation

**Required Analysis:**

- Smoothing Parameter: Test α values [0.1, 0.5, 1.0, 2.0, 5.0]
- Feature Selection: Analyze impact of different feature subsets
- Probability Analysis: Examine predicted probabilities distribution
- Independence Assumption: Discuss violations and their impact
- Performance Comparison: Compare with sklearn's MultinomialNB

# Part C: Decision Trees (40 points)

## C1. Dataset and setup

- Dataset: Breast Cancer Wisconsin (Diagnostic)
    - Use sklearn.datasets.load_breast_cancer().
    - This is a binary classification problem (malignant vs. benign) with continuous-valued features.
    - Split the data into 70% training, 15% validation, and 15% test (stratified).

## C2. Decision Tree implementation

- Implement a binary decision tree classifier for continuous features.
- At each internal node, split on a single feature $x_j$ with a threshold $\tau$, creating branches $x_j \leq \tau$ and $x_j > \tau$.
- Use entropy as the impurity measure and information gain as the split criterion.
- Leaf nodes output a class label (majority class among samples reaching that leaf).

## C3. Split Search and Stopping Criteria

- For each feature at a node, consider thresholds between sorted distinct feature values.
    - For each candidate (feature, threshold), compute the information gain and choose the best split.
- Implement stopping criteria based on:
    - Maximum tree depth (max_depth).
    - Minimum number of samples required to split a node (min_samples_split).
    - Nodes where all samples belong to the same class.

## C4. Hyperparameter tuning and analysis

- Explore max_depth $\in \{2, 4, 6, 8, 10\}$ and min_samples_split $\in \{2, 5, 10\}$.
- Use the validation set to choose the best (max_depth, min_samples_split) combination.
- Retrain the tree on training + validation using the selected hyperparameters and evaluate on the test set.
- Examine how training and validation accuracy change with max_depth (fixing min_samples_split) and summarise in a table or plot.

## C5. Analysis and Evaluation

Required Analysis:

- Hyperparameter Tuning: Test different max_depth values
- Performance Metrics: Accuracy, Precision, Recall, F1-score for both classes
- Confusion Matrix: Detailed analysis of classification errors
- Feature Importance: Rank features by their information gain contribution
- Tree Complexity: Analyze relationship between tree depth and performance
- Overfitting Analysis: Compare training vs validation performance

# Part D: Random Forest (Bonus - 20 points)

## D1. Dataset and Setup

- Dataset: Breast Cancer Wisconsin (Diagnostic, same as Part C)
    - Reuse your preprocessing choices and the 70/15/15 split from Part C (or clearly document any changes).

## D2. Random Forest Implementation

- Build a random forest classifier using your own decision tree implementation.
- Train T trees, each on a bootstrap sample of the training data (sampling with replacement).
- At each split inside a tree, consider only a random subset of features of size max_features.
- Use majority voting over all trees to obtain the final prediction.

## D3. Hyperparameter Tuning and Evaluation

- Use the best max_depth and min_samples_split values found in Part B for all trees.
- Explore at least T ∈ {5, 10, 30, 50} and max_features ∈ {⌊√d⌋, ⌊d/2⌋}, where d is the number of features.
- Use the validation set to select the best (T, max_features) combination.
- Retrain the random forest on training + validation using the selected hyperparameters and evaluate on the test set.

In your report, compare the performance of a single decision tree (Part C) with the random forest (Part D) and discuss the effect on bias and variance.

### Deliverables

- Python code: Well-structured and documented scripts or notebooks for all parts.
- Report (PDF): A concise report including model descriptions, hyperparameters, metrics, tables/plots, and analysis.

## Grading Rubric

- Component – Points – Criteria
    - Part A: Probabilistic Gaussian Generative Classifier – 20 points – Implementation correctness, parameter estimation, evaluation and analysis
    - Part C: Decision Trees – 40 points – Correct tree construction, split selection, hyperparameter tuning, bias–variance discussion
    - Part B: Gaussian Naive Bayes – 20 points – Model implementation, hyperparameter tuning, comparison with Part A
    - Part D: Random Forest (Bonus) – 20 points – Ensemble implementation, tuning, comparison to single tree
- Total: 80 points + 20 bonus points.

Prof. Dr. Marwan Torki

Eng. Youssef El-Ebiary  Eng. Youssef El-Kady
Eng. Ziad Fahmy  Eng. Sarah Mahmoud

**Academic Integrity Reminder**: While collaboration within teams is encouraged, copying from other teams or external sources without attribution is strictly prohibited. All implementations should be original work by your team.

## Final Notes

1. You should work in groups of maximum three, not finding a team is NOT an excuse.
2. You should deliver with a naming scheme id _assigment.zip.
3. Delivery will be ignored if you didn't follow the naming scheme provided in 2, any one of the team ids can be used.
4. Any form of academic dishonesty, including but not limited to using AI tools (such as ChatGPT or other code generation platforms), copying open-source code without proper attribution, or engaging in 'vibe coding' without genuine understanding, will be considered a serious violation and will be heavily penalized.

Prof. Dr. Marwan Torki

Eng. Youssef El-Ebiary
Eng. Youssef El-Kady
Eng. Ziad Fahmy
Eng. Sarah Mahmoud