**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5710 Machine Learning**

**Fall 2025**

**Home Assignment 2.**

**Student name: Ashmitha Kumbham**

**700773518**

**Submission Requirements:**

* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Comment your code appropriately ***IMPORTANT.***
* Any submission after provided deadline is considered as a late submission.

**Part A: Calculation (6 Questions)**

**Q1. Decision Stump Prediction**  
Given a decision stump:

h(x)=+ if Sneezing=Yes, − otherwise.

Dataset:

* (Sneezing=Yes, Label=+)
* (Sneezing=No, Label=-)
* (Sneezing=Yes, Label=-)
* (Sneezing=No, Label=-)

1. What is the **training error rate** of this stump?

Stump: predict “+” if Sneezing = Yes, else “−”.  
Dataset:

(Yes, +) → correct  
(No, −) → correct  
(Yes, −) → wrong  
(No, −) → correct

Errors = 1 out of 4 → training error = 1/4 = 0.25 = **25%**

1. Compare it to the **memorizer** model (predicts perfectly).

* **Comparison with memorizer model**

**Decision stump:** makes 1 error out of 4 → training error = 25%.

**Memorizer model:** remembers every record exactly → training error = 0%.

The memorizer looks better on training data because it is perfect, but it can easily overfit and fail on new data. The stump is less accurate on training but simpler and may generalize better.

**Q2. Training Error as Splitting Criterion**  
A dataset has 6 records:

| **Age (x1)** | **Exercise (x2)** | **Diet (x3)** | **Label** |
| --- | --- | --- | --- |
| Young | High | Poor | Yes |
| Young | Medium | Good | Yes |
| Mid | Low | Poor | No |
| Old | Medium | Poor | No |
| Old | High | Good | Yes |
| Mid | Low | Poor | No |

1. Compute the **training error rate** if you split on each feature (x1, x2, x3).

Training Error as Splitting Criterion

Rule: after a split, each leaf predicts its majority class; errors in a leaf = (leaf size − majority count). Ties count as 1 error.

Split on x1 = Age

* Young: {Yes, Yes} → predict Yes → 0 errors
* Mid: {No, No} → predict No → 0 errors
* Old: {No, Yes} → tie → 1 error  
  Total = 1/6 ≈ 16.67%

Split on x2 = Exercise

* High: {Yes, Yes} → predict Yes → 0 errors
* Medium: {Yes, No} → tie → 1 error
* Low: {No, No} → predict No → 0 errors  
  Total = 1/6 ≈ 16.67%

Split on x3 = Diet

* Poor: {Yes, No, No, No} → majority No → 1 error (the Yes)
* Good: {Yes, Yes} → predict Yes → 0 errors  
  Total = 1/6 ≈ 16.67%

1. Which feature is the best root split using **training error**?

All three features (Age, Exercise, Diet) give the same result: each split misclassifies 1 out of 6 records, so the training error rate is **16.67%**. Therefore, they all tie as the best root split based on training error.

* This means training error alone cannot help us decide which feature is better.
* In such cases, we usually rely on other measures like **entropy** or **information gain** to break the tie.
* It also shows that the dataset is small and balanced in such a way that different splits give the same level of accuracy.

**Q3. Entropy & Information Gain**  
For the same dataset above:

1. Compute the **entropy of the labels**.

**Entropy of labels (Yes=3, No=3):**p(Yes)=0.5,  p(No)=0. 5  
H(Y)=−∑p=−[0.50.5+0.50.5]=1.0 bit

1. Compute entropy after splitting on **Exercise (x2)**.

High: 2 examples (both Yes) → entropy = 0 (pure)

Medium: 2 examples (1 Yes, 1 No) → p=0.5/0.5 → entropy = 1.0 bit

Low: 2 examples (both No) → entropy = 0 (pure)

Weighted average entropy after split:

1. Calculate the **information gain**.

Information gain = H(Y) − H\_after = 1.0 − 0.333333 = 0.666667 bits.

1. Decide if Exercise is a good split.

Interpretation: splitting on Exercise reduces entropy substantially (gain ≈ 0.667 bits), so **Exercise is a good split** (it produces mostly-pure leaves).

**Q4. Confusion Matrix Metrics**  
A binary classifier produces the following confusion matrix on 100 test samples:

|  | **Predicted +** | **Predicted -** |
| --- | --- | --- |
| Actual + | 25 | 5 |
| Actual - | 15 | 55 |

1. Compute Accuracy, Precision, Recall, Specificity, and F1-score.

So: **TP=25, FN=5, FP=15, TN=55**

Formulas and numbers:

* **Accuracy** = (TP+TN) / total = (25+55)/100 = 80/100 = **0.80 (80%)**.
* **Precision** = TP / (TP+FP) = 25 / (25+15) = 25/40 = **0.625 (62.5%)**.
* **Recall (Sensitivity)** = TP / (TP+FN) = 25 / (25+5) = 25/30 ≈ **0.8333 (83.33%)**.
* **Specificity** = TN / (TN+FP) = 55 / (55+15) = 55/70 ≈ **0.7857 (78.57%)**.
* **F1-score** = 2 \* (Precision \* Recall) / (Precision + Recall)  
  = 2 \* (0.625 \* 0.833333) / (0.625 + 0.833333) ≈ **0.7143 (71.43%)**.

1. Suppose the dataset was imbalanced (80 negatives, 20 positives). Which metric is most informative?

Accuracy becomes misleading when classes are imbalanced (a dumb classifier predicting "negative" always could get 80% accuracy). For imbalanced problems you should focus on **precision**, **recall**, and **F1** (or PR AUC). Which one matters most depends on the cost of errors:

* If **missing positives** is costly (false negatives bad), emphasize **recall**.
* If **false positives** are costly, emphasize **precision**.
* **F1** balances precision and recall and is a common single-number summary.

**Q5. Distance Calculations (kNN)**  
You have 3 labeled points:

* A(2,4), Red
* B(4,4), Blue
* C(4,6), Red

Classify new point P(5,4):

1. Compute **Euclidean distance** from P to A, B, C.

Points:

* A = (2,4), Red
* B = (4,4), Blue
* C = (4,6), Red  
  New point P = (5,4)

Euclidean distances (compute step-by-step):

1. Predict label using **1-NN**.

**1-NN prediction:** nearest is B (distance 1.0) → label **Blue**.

1. Predict label using **3-NN (majority vote)**.

**3-NN prediction:** the three nearest are B (Blue, 1.0), C (Red, ~2.236), A (Red, 3.0) → votes: Red 2 vs Blue 1 → **Red**.

(If a tie occurs in k-NN you need a tie-breaking rule — common options: choose label of closest neighbor among tied labels, break arbitrarily or choose smaller label id.)

**Q6. K-fold Cross-Validation**  
You want to evaluate kNN with k=1,3,5 using 4-fold CV. Errors observed on each fold:

| **Fold** | **k=1** | **k=3** | **k=5** |
| --- | --- | --- | --- |
| 1 | 0.20 | 0.15 | 0.10 |
| 2 | 0.25 | 0.20 | 0.15 |
| 3 | 0.15 | 0.10 | 0.10 |
| 4 | 0.30 | 0.20 | 0.20 |

1. Compute **mean CV error** for each k.

Compute mean CV error (average over folds):

* **k=1:** mean = (0.20+0.25+0.15+0.30)/4 = 0.90/4 = **0.225 = 22.5%**
* **k=3:** mean = (0.15+0.20+0.10+0.20)/4 = 0.65/4 = **0.1625 = 16.25%**
* **k=5:** mean = (0.10+0.15+0.10+0.20)/4 = 0.55/4 = **0.1375 = 13.75%**

1. Which value of k generalizes best?

k=5 has the lowest mean CV error (13.75%), so it appears to generalize best on this cross-validation experiment. (Interpretation: increasing k smooths decision boundary, reduces variance; as k gets too large you may add bias, but here k=5 is the sweet spot.)

**Part B: Programming (3 Questions)**

**Q7. Build a Decision Tree (sklearn)**

1. Use sklearn.tree.DecisionTreeClassifier on the **Iris dataset**.
2. Train trees with max\_depth = 1, 2, 3.
3. Report training and test accuracy for each depth.
4. Discuss signs of **underfitting** vs **overfitting**.

**Q8. kNN Classification (sklearn)**

1. Use sklearn.neighbors.KNeighborsClassifier on the Iris dataset (only 2 features: sepal length, sepal width).
2. Train models with k=1,3,5,10.
3. Plot the **decision boundaries** for each k.
4. Comment on how the boundaries change.

**Q9. Performance Evaluation Programming**

1. Train a kNN classifier (k=5) on Iris dataset.
2. Compute **confusion matrix** and display it with sklearn.metrics.confusion\_matrix.
3. Compute **accuracy, precision, recall, F1** using classification\_report.
4. Plot the **ROC curve** and compute **AUC**.