**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5710 Machine Learning**

**Fall 2025**

**Home Assignment 2.**

**Student name: Sai Kumar Beesa**

**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 — Calculations (answers with steps)**

**Q1. Decision Stump Prediction**

Stump: h(x) = + if Sneezing = Yes, otherwise -.

Dataset (4 examples):

1. (Yes, Label = +) → h = + → **correct**
2. (No, Label = -) → h = - → **correct**
3. (Yes, Label = -) → h = + → **wrong**
4. (No, Label = -) → h = - → **correct**
5. **Training error rate** = (# wrong) / (total) = 1 / 4 = **0.25 = 25%**.
   * Step-by-step: wrong = 1; total = 4; 1 ÷ 4 = 0.25.
6. **Compare to memorizer**: a memorizer that predicts perfectly has **0%** training error. So the stump (25%) is worse than the memorizer (0%).

**Q2. Training Error as Splitting Criterion**

Data (6 records):

| **#** | **Age (x1)** | **Exercise (x2)** | **Diet (x3)** | **Label** |
| --- | --- | --- | --- | --- |
| 1 | Young | High | Poor | Yes |
| 2 | Young | Medium | Good | Yes |
| 3 | Mid | Low | Poor | No |
| 4 | Old | Medium | Poor | No |
| 5 | Old | High | Good | Yes |
| 6 | Mid | Low | Poor | No |

Count of labels: Yes = rows 1,2,5 = 3; No = rows 3,4,6 = 3.

We compute training-error if we split on each feature and at each child predict the majority label.

**Split on Age (x1)**

* Young: rows 1 & 2 → labels {Yes, Yes} → majority = Yes → errors = 0.
* Mid: rows 3 & 6 → {No, No} → majority = No → errors = 0.
* Old: rows 4 & 5 → {No, Yes} → tie; any majority choice gives 1 error (min possible = 1).
* Total errors = 1 → **error rate = 1 / 6 = 0.1667 = 16.67%**.

**Split on Exercise (x2)**

* High: rows 1 & 5 → {Yes, Yes} → errors = 0.
* Medium: rows 2 & 4 → {Yes, No} → tie → errors = 1.
* Low: rows 3 & 6 → {No, No} → errors = 0.
* Total errors = 1 → **error rate = 1/6 = 16.67%**.

**Split on Diet (x3)**

* Good: rows 2 & 5 → {Yes, Yes} → errors = 0.
* Poor: rows 1, 3, 4, 6 → {Yes, No, No, No} → majority = No (3 No vs 1 Yes) → errors = 1 (row 1).
* Total errors = 1 → **error rate = 1/6 = 16.67%**.

**Answer (2.1)**: All three features give the same training error rate **1/6 ≈ 16.67%**.

**Answer (2.2)**: Since all tie, **any of x1, x2, or x3** could be chosen as best root split by training-error criterion.

**Q3. Entropy & Information Gain (same dataset)**

We previously found Yes = 3, No = 3, total n = 6.

**3.1 Entropy of labels**

Entropy H(Y)=−∑p(y)log⁡2p(y)H(Y) = -\sum p(y) \log\_2 p(y).

* p(Yes)=3/6=0.5p(Yes) = 3/6 = 0.5.
* p(No)=3/6=0.5p(No) = 3/6 = 0.5.

So H(Y)=−0.5log⁡2(0.5)−0.5log⁡2(0.5)H(Y) = -0.5\log\_2(0.5) - 0.5\log\_2(0.5).  
log⁡2(0.5)=−1\log\_2(0.5) = -1, so:  
H(Y)=−0.5(−1)−0.5(−1)=0.5+0.5=1H(Y) = -0.5(-1) - 0.5(-1) = 0.5 + 0.5 = 1.

**Entropy = 1.0 bit.**

**3.2 Entropy after splitting on Exercise (x2)**

Exercise values and child node entropies:

* High (2 samples): labels {Yes, Yes} → class distribution: (1.0, 0.0) → Entropy = 0
* Medium (2 samples): labels {Yes, No} → distribution (0.5,0.5) → Entropy = 1
* Low (2 samples): labels {No, No} → distribution (0,1.0) → Entropy = 0

Weighted entropy:

H(Y∣x2)=26⋅0+26⋅1+26⋅0=26=13≈0.3333H(Y|x2) = \frac{2}{6}\cdot0 + \frac{2}{6}\cdot1 + \frac{2}{6}\cdot0 = \frac{2}{6} = \frac{1}{3} \approx 0.3333

**Entropy after split ≈ 0.3333 bits.**

**3.3 Information gain**

IG=H(Y)−H(Y∣x2)=1.0−13=23≈0.6667 bits.IG = H(Y) - H(Y|x2) = 1.0 - \frac{1}{3} = \frac{2}{3} \approx 0.6667 \text{ bits.}

**3.4 Is Exercise a good split?**

Yes. An information gain of ~0.667 bits (out of maximum 1 bit) is substantial — Exercise meaningfully reduces label uncertainty. So **Exercise is a good split**.

**Q4. Confusion Matrix Metrics**

Confusion matrix (100 samples):

|  | **Predicted +** | **Predicted -** |
| --- | --- | --- |
| Actual + | 25 (TP) | 5 (FN) |
| Actual - | 15 (FP) | 55 (TN) |

Compute metrics:

* **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, TPR)** = TP / (TP + FN)  
  = 25 / (25 + 5) = 25 / 30 = **0.8333 = 83.33%**.
* **Specificity (TNR)** = TN / (TN + FP)  
  = 55 / (55 + 15) = 55 / 70 ≈ **0.7857 = 78.57%**.
* **F1-score** = 2⋅Precision⋅RecallPrecision+Recall2 \cdot \dfrac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}  
  Precision = 0.625, Recall = 0.833333...  
  Numerator = 2⋅0.625⋅0.833333=1.04166672 \cdot 0.625 \cdot 0.833333 = 1.0416667.  
  Denominator = 0.625+0.833333=1.45833330.625 + 0.833333 = 1.4583333.  
  F1 = 1.0416667 / 1.4583333 ≈ **0.7142857 = 71.43%**.

**4.2 If dataset were imbalanced (80 negatives, 20 positives)**

When classes are imbalanced, **accuracy can be misleading** (a classifier that predicts always negative would get 80% accuracy). The most informative metrics are **Precision and Recall** (and their harmonic mean, **F1-score**) because they focus on the positive class performance.

* Use **Recall** if missing positives is costly (you want to catch as many positives as possible).
* Use **Precision** if false positives are costly.
* **F1-score** is a single-number compromise summarizing both.

**Q5. Distance Calculations (kNN)**

Points:

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

**5.1 Euclidean distances**

Euclidean distance formula d((x1,y1),(x2,y2))=(x1−x2)2+(y1−y2)2d((x\_1,y\_1),(x\_2,y\_2))=\sqrt{(x\_1-x\_2)^2+(y\_1-y\_2)^2}.

* d(P,A)=(5−2)2+(4−4)2=32+02=9=3.0.d(P,A) = \sqrt{(5-2)^2 + (4-4)^2} = \sqrt{3^2 + 0^2} = \sqrt{9} = 3.0.
* d(P,B)=(5−4)2+(4−4)2=1+0=1.0.d(P,B) = \sqrt{(5-4)^2 + (4-4)^2} = \sqrt{1 + 0} = 1.0.
* d(P,C)=(5−4)2+(4−6)2=1+(−2)2=1+4=5≈2.23607.d(P,C) = \sqrt{(5-4)^2 + (4-6)^2} = \sqrt{1 + (-2)^2} = \sqrt{1+4} = \sqrt{5} \approx 2.23607.

**5.2 1-NN prediction**

Nearest neighbor is **B (distance 1.0)** which is **Blue** → **predict Blue**.

**5.3 3-NN prediction (majority vote)**

Nearest 3: B (Blue), C (Red), A (Red) → votes: Red = 2, Blue = 1 → **predict Red**.

**Q6. K-fold Cross-Validation**

Given 4-fold CV errors:

| **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 |

**6.1 Mean CV error for each k**

Compute means (sum errors across folds ÷ 4):

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

**6.2 Which k generalizes best?**

Lower mean CV error = better generalization estimate. **k = 5** has the lowest mean error **0.1375**, so **k = 5** generalizes best among these options.

**Part B — Programming (code, instructions, and guidance)**

Below are ready-to-run Python scripts (with comments) for Q7–Q9. Put these scripts into your GitHub repo, add a clear README.md (explain dataset, dependencies, how to run), and comment code heavily as required by the assignment.

You can run them in any standard Python environment with scikit-learn, matplotlib, numpy, and pandas installed.

**Environment / Dependencies**

Install (example):

pip install numpy pandas scikit-learn matplotlib seaborn

(Seaborn only used optionally for nicer plotting; core tasks use matplotlib & sklearn.)

**Q7 — Decision Tree on Iris (sklearn)**

"""

Q7: Train DecisionTreeClassifier on Iris with max\_depth=1,2,3 and report train/test accuracy.

Run: python q7\_decision\_tree.py

"""

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Load Iris

data = load\_iris()

X = data.data     # 4 features

y = data.target   # 3 classes

# Train/test split (stratified)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3, random\_state=42, stratify=y)

depths = [1, 2, 3]

results = []

for d in depths:

    clf = DecisionTreeClassifier(max\_depth=d, random\_state=42)

    clf.fit(X\_train, y\_train)

    y\_train\_pred = clf.predict(X\_train)

    y\_test\_pred = clf.predict(X\_test)

    train\_acc = accuracy\_score(y\_train, y\_train\_pred)

    test\_acc = accuracy\_score(y\_test, y\_test\_pred)

    results.append((d, train\_acc, test\_acc))

    print(f"max\_depth={d} | train\_acc={train\_acc:.4f} | test\_acc={test\_acc:.4f}")

s

# Discussion guidance:

# - If train\_acc and test\_acc both low -> underfitting (too simple)

# - If train\_acc >> test\_acc -> overfitting

# - If both high and close -> good generalization

# output:

A number of numbers on a black background

AI-generated content may be incorrect.

# Discussion guidance:

# - If train\_acc and test\_acc both low -> underfitting (too simple)

# - If train\_acc >> test\_acc -> overfitting

# - If both high and close -> good generalization

**What to report**:

* For each max\_depth, print training and test accuracy (as shown).
* Discuss signs:
  + max\_depth=1 likely underfits (low train & test accuracy).
  + max\_depth=2 or 3 often improves training accuracy; if training accuracy climbs to ~1.0 while test accuracy drops, that indicates overfitting. If both train/test high and similar → good fit.

**Q8 — kNN classification and decision boundaries (Iris, 2 features)**

Q8: kNN decision boundaries using only 2 features: sepal length & sepal width.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from matplotlib.colors import ListedColormap

# Load Iris and select first two features

data = load\_iris()

X = data.data[:, :2]   # sepal length, sepal width

y = data.target

# Train/test split (optional; we'll fit on entire data to show boundary)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42, stratify=y)

ks = [1, 3, 5, 10]

h = .02  # step size in the mesh

# Create color maps

cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])

cmap\_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for i, k in enumerate(ks):

    clf = KNeighborsClassifier(n\_neighbors=k)

    clf.fit(X\_train, y\_train)

    # Create meshgrid

    x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

                         np.arange(y\_min, y\_max, h))

    Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(6, 4))

    plt.contourf(xx, yy, Z, cmap=cmap\_light, alpha=0.6)

    plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cmap\_bold,

                edgecolor='k', s=50, label='train')

    plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cmap\_bold,

                marker='x', s=60, label='test')

    plt.title(f"k-NN (k={k}) — sepal length vs sepal width")

    plt.xlabel("Sepal length (cm)")

    plt.ylabel("Sepal width (cm)")

    plt.legend()

    plt.tight\_layout()

    plt.show()

# Comments to write in report:

# - k=1: decision boundary will be very jagged (high variance), small regions around points.

# - As k increases, boundaries smooth out (less variance, more bias).

# - Very large k may over-smooth and cause underfitting (too coarse).

A diagram of a graph

AI-generated content may be incorrect.

A diagram of a map with red and blue dots

AI-generated content may be incorrect.

A diagram of a variety of colors

AI-generated content may be incorrect.

A diagram of a graph

AI-generated content may be incorrect.

**Q9 — Performance Evaluation (kNN k=5)**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, auc

from sklearn.preprocessing import label\_binarize

from itertools import cycle

# Load Iris

data = load\_iris()

X = data.data

y = data.target

n\_classes = len(np.unique(y))

# Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3, random\_state=42, stratify=y)

# Train kNN

k = 5

clf = KNeighborsClassifier(n\_neighbors=k)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

# Classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, digits=4))

# ROC & AUC (multiclass one-vs-rest)

y\_score = clf.predict\_proba(X\_test)

y\_test\_binarized = label\_binarize(y\_test, classes=np.arange(n\_classes))

# Compute ROC curve and AUC for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

    fpr[i], tpr[i], \_ = roc\_curve(y\_test\_binarized[:, i], y\_score[:, i])

    roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average AUC

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test\_binarized.ravel(), y\_score.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curves

plt.figure()

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])

for i, color in zip(range(n\_classes), colors):

    plt.plot(fpr[i], tpr[i], color=color,

             label=f'ROC curve of class {i} (area = {roc\_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=0.6)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic — multiclass (one-vs-rest)')

plt.legend(loc="lower right")

plt.show()

# What to report:

# - Confusion matrix numbers (per-class errors)

# - Classification report: accuracy, precision, recall, F1 for each class and avg

# - AUC values per class and micro-average

# Note: For multiclass ROC/AUC we use one-vs-rest; report per-class AUCs and micro/macro average as desired.

#

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

A graph of a function

AI-generated content may be incorrect.

**Classification Report**:

precision recall f1-score support

setosa 1.00 1.00 1.00 15

versicolor 1.00 0.88 0.93 16

virginica 0.88 1.00 0.93 14

accuracy 0.96 45

macro avg 0.96 0.96 0.96 45

weighted avg 0.96 0.96 0.96 45

**ROC Curve Plot** with **AUC values** for each class (one-vs-rest).