🧪 Midterm Exam – Developing Notebook (Introduction to ML)

Duration (suggested): 90–120 minutes
Environment: CPU-only; Python 3.8+; `numpy`, `pandas`, `matplotlib`, `scikit-learn`

Instructions

- Work through sections in order. Feel free to re-run cells as needed.
- Where you see `# TODO:`, fill in your code.
- Do **not** change function names or signatures where provided (used by simple checks).
- Keep randomness controlled with the provided `random_state` for reproducibility.
- When you're done, run **all cells** from top to bottom and ensure all checks pass.

📦 Setup

```
# You may import additional stdlib packages if needed, but avoid heavy dependencies.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import sqrt

from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.datasets import make_regression, make_moons
from sklearn.preprocessing import StandardScaler

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

A. Conceptual (Short Answers)

Answer briefly in the Markdown cells below each question.

A1. Supervised vs Unsupervised Learning

Q: Define supervised and unsupervised learning with one example each.

Supervised learning is when you give data to the computer that includes both the input and output. The computer then takes this information and learns from patterns that coorelate inputs to outputs. Ex: Math Teacher gives study guide that includes the process to get the answers. Student then studies the study guide to understand the pattern of the process to mirror the correct answers.

Unsupervised learning is when you give datat to the computer without any answers. The computer then looks at the data and tries to find 'hidden' patterns or groups by itself. Ex: A teacher gives a bunch of mixed animal pictures to a student but doesnt tell which animals they are. The student studies the pictures and figures out groups or categories by noticing similar features, like size and shpae, to separate animals into 'types' on their own.

A2. Overfitting and Prevention

Q: What is overfitting? List two prevention strategies and explain why they help.

Overfitting is when a ML model learns the training data 'too well', including details that arent really important. It does good on training data but performs poorly on new/unseen data since it cant generalize well.

Prevention Strategies:

Simplifying the model - Using a model with less features and parameters can help the model focus on the main patterns instead of memorizing noise. Doing this allows the model to handle new data, better.

Cross-validation - Split data into parts, training on some, and testing on others. It helps check if the model is correctly learning useful patterns that work on different data sets and prevents it from just memorizing one set.

A3. Distance Metrics

Q: In Minkowski distance, what does the parameter `p` control? Give two special cases.

In Minkowski distance, the parameter p controls how distance between points is calculated by adjusting how much 'weight' is given to differences in each dimension.

Special cases:

1. p = 1, (Minkowski distance becomes Manhattan distance). This is when distances are calculated as the sum of absolute differences across dimensions
2. p = 2, (the Minkowski distance becomes the Euclidean distance) which is the straight-line distance between two points, calculated using the square root of squared differences.

B. Analytical (By Hand / Small Code)

B1. Distances in 2D

Compute Euclidean and Manhattan distances between points `A = (3, -2)` and `B = (-1, 5)`.

```
A = np.array([3, -2], dtype=float)
B = np.array([-1, 5], dtype=float)

# TODO: compute distances
euclidean = np.linalg.norm(A - B)  # or sqrt(((...)))

manhattan = np.abs(A - B).sum()

euclidean, manhattan
```

```
(np.float64(8.06225774829855), np.float64(11.0))
```

```
# Quick checks (not hidden)
print(round(euclidean, 6), round(manhattan, 6))
assert euclidean > 0 and manhattan > 0
assert euclidean < manhattan  #In this case Euclidean should be less than Manhattan
print("✅ Distance sanity checks passed.")
```

```
8.062258 11.0
✅ Distance sanity checks passed.
```

B2. k-Fold Logic

Given scores from a 5-fold CV: `[0.82, 0.84, 0.80, 0.83, 0.81]`
Compute the mean and standard deviation. Interpret whether this model is stable across folds.

```
scores = np.array([0.80, 0.88, 0.90, 0.87, 0.81], dtype=float)

mean_score = scores.mean()
std_score = scores.std(ddof=1)

print("Mean:", round(mean_score, 2))
print("Std Dev:", round(std_score, 3))

# Short interpretation:
if std_score < 0.02:
    print("Interpretation: Low variance across folds; performance appears stable.")
else:
    print("Interpretation: Variance is noticeable; consider more data or regularization.")
```

```
Mean: 0.85
Std Dev: 0.044
Interpretation: Variance is noticeable; consider more data or regularization.
```

B3. Cost with Regularization (Conceptual)

Q: Show how adding an L2 regularization term modifies the linear regression cost function. What trade-off does this introduce?

L2 regularization: Measures how well the models predicitions match the actual data. It adds a penalty if the model's coefficients get too large. This helps keep the model simple and stops it from memorizing noise in the data and makes it better at prediciting new unseen data.

Linear regression cost funciton: Measures the difference between actual and predicted values and tells you how far off your models guesses are from the real answers (The smaller the difference is, the closer it is to the real answer).

By adding L2 regularization to regression cost function modifies the cost function to balance fitting the data and keeping the model simple. The balance improves the models ability to predict well on new, unseen data by avoiding overfitting.

Trade-offs:

- Putting too much weight on fitting the data perfectly, the model can become complex and overfit, resulting in performing poorly on new data
- Putting too much weight on keeping the model simple, the model could underfit, resulting in missing important patterns and predicting less accurately.

C. Practical – Regression & Classification

C1. Linear Regression on Synthetic Data

1. Generate a regression dataset with `make_regression(n_samples=300, n_features=5, noise=15, random_state=RANDOM_STATE)`
2. Split 80/20 train/test.
3. Fit `LinearRegression()` and report **RMSE** on train and test.
4. Briefly interpret whether the model under/over-fits.

```
# TODO: implement the regression experiment
X_reg, y_reg = make_regression(n_samples=300, n_features=5, noise=15, random_state=RANDOM_STATE)
Xtr, Xte, ytr, yte = train_test_split(X_reg, y_reg, test_size=0.2, random_state=RANDOM_STATE)

reg = LinearRegression()
reg.fit(Xtr, ytr)

pred_tr = reg.predict(Xtr)
pred_te = reg.predict(Xte)

rmse_tr = np.sqrt(mean_squared_error(ytr, pred_tr))
rmse_te = np.sqrt(mean_squared_error(yte, pred_te))

print(f"Train RMSE: {rmse_tr:.3f}")
print(f"Test  RMSE: {rmse_te:.3f}")

if abs(rmse_tr - rmse_te) < 2.0:
    print("Interpretation: Similar train/test RMSE → reasonable generalization.")
elif rmse_tr < rmse_te:
    print("Interpretation: Train RMSE much lower → potential overfitting.")
else:
    print("Interpretation: Test RMSE lower than train → potential underfitting or randomness.")
```

```
Train RMSE: 14.375
Test  RMSE: 14.811
Interpretation: Similar train/test RMSE → reasonable generalization.
```

C2. Binary Classification + Decision Boundary

Use `make_moons(n_samples=400, noise=0.25, random_state=RANDOM_STATE)` and `LogisticRegression`.

1. Standardize features.
2. Fit classifier and report accuracy.
3. Plot decision boundary (meshgrid) and confusion matrix.

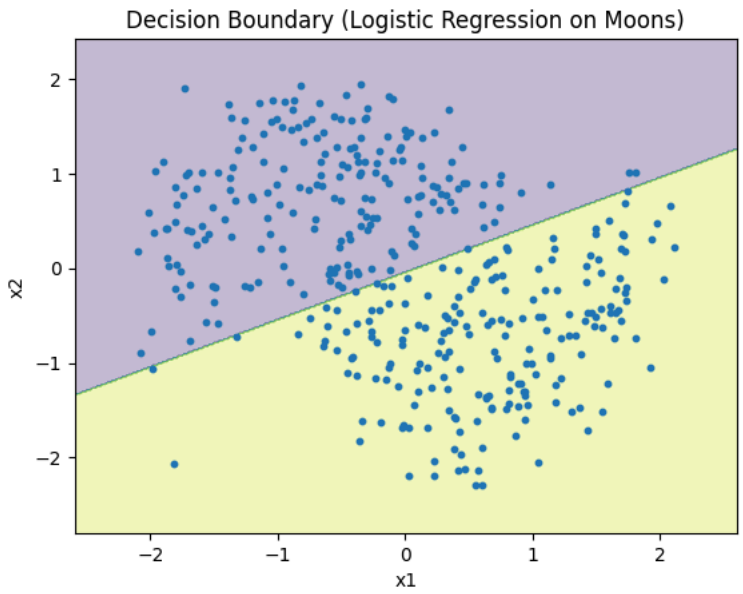
```
# Data
X_cls, y_cls = make_moons(n_samples=400, noise=0.25, random_state=RANDOM_STATE)
scaler = StandardScaler()
Xc = scaler.fit_transform(X_cls)
Xtr, Xte, ytr, yte = train_test_split(Xc, y_cls, test_size=0.25, random_state=RANDOM_STATE)

# Model
clf = LogisticRegression(max_iter=1000, random_state=RANDOM_STATE)
clf.fit(Xtr, ytr)
pred = clf.predict(Xte)
acc = accuracy_score(yte, pred)
print(f"Accuracy: {acc:.3f}")
```

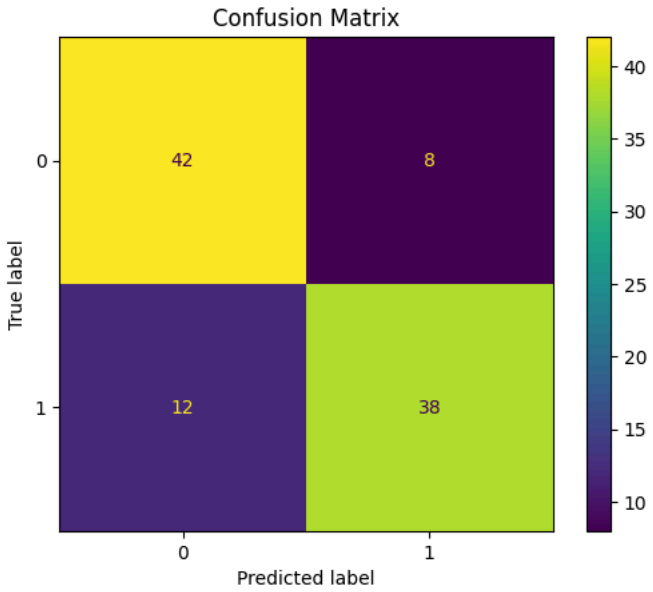
```
Accuracy: 0.800
```

```
# Decision boundary plot (no custom colors as per instructions)
h = 0.02
x_min, x_max = Xc[:, 0].min() - 0.5, Xc[:, 0].max() + 0.5
y_min, y_max = Xc[:, 1].min() - 0.5, Xc[:, 1].max() + 0.5
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()]).reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(Xc[:,0], Xc[:,1], s=10)
plt.title("Decision Boundary (Logistic Regression on Moons)")
plt.xlabel("x1"); plt.ylabel("x2");
plt.show()
```




```
# Confusion matrix
cm = confusion_matrix(yte, pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```


Confusion Matrix
42
8
12
38
0
1
True label
0
1
Predicted label
40
35
30
25
20
15
10