Engineering and Applied Science Programs for Professionals
Whiting School of Engineering
Johns Hopkins University
685.701 Data Science: Modeling and Analytics
Programming Assignment 1
Assigned with Module 1
Due at the end of Module 7

Total Points 100/100

Submission Instructions and Problem Selection:

Problems 1 and 2 are required. In addition, complete *exactly one* of Problems 3, 4, or 5. Clearly indicate which optional problem you selected at the top of your submission. Unless otherwise noted, use the provided notebook, Python, and follow standard notebook practices (pseudocode → code → analysis → conclusions)

While student discussions, sharing of recommendations, and some collaboration are encouraged for team building and peer learning, **each student is required to submit their own complete programming solutions.** Every submission must demonstrate the individual student's ability to design, document, implement in code, and test their work independently.

Grading Criteria (100% for each problem)

Completeness & Problem Coverage (20%): Implement all required parts (pseudocode, code, performance analysis, outputs) with nothing missing or vague.

Writing Quality, Technical Accuracy & Justification (20%): Explanations are clear, concise, and technically correct; design choices and conclusions are justified with sound reasoning.

Quantitative Work: Assumptions, Derivations & Calculations (20%): List assumptions up front; show derivations or intermediate formulas/metrics; present final results correctly and clearly.

Code Quality, Documentation & Execution (20%): The notebook runs end-to-end without errors; names are clear; formatting is consistent; comments explain key logic; the work is organized and reproducible.

Examples, Test Cases & Visuals (10%): Include labeled test cases and informative figures/tables with titles, captions, and axes; use appropriate metrics (e.g., precision, recall, F1, ROC/PR curves) rather than accuracy alone [8].

Notebook README & Reproducibility (10%): Provide Python version, package list with install steps, dataset details and download instructions, and step-by-step run instructions; use relative paths and fixed random seeds.

1. **Preprocessing (Required) — Modules 1, 2, and 3**

   20 points total (see per-problem grading breakdown below).

   We will develop a robust *classification pipeline* [9, 1, 13] to predict a target event using a structured dataset of your choice (approved public datasets encouraged; e.g., Kaggle or Hugging Face Datasets). Your workflow should explicitly follow the data-processing practices covered in Modules 2–3 (collection, cleaning, transformation, outlier handling; feature engineering/ranking/selection; dimensionality reduction; and principled data splitting). Provide a concise dataset description (features, target, size, splits) and include either the dataset (if licensing allows) or a reproducible link/instructions to obtain it.

   Important implementation notes (align with Modules 2–3):

   Train/validation/test splits with a fixed random seed; use `Stratified` procedures for classification and report exact sizes.[1]

   Data quality & cleaning: detect and remove duplicates; impute missing values (mean/median for numeric; mode for nominal); identify and treat outliers (e.g., IQR rule) with justification.[2]

   Feature preprocessing: apply appropriate transformations (standardization vs. normalization), categorical encoding (e.g., one-hot encoding), and date/time expansion when relevant; explain why each step is needed for your downstream model(s).

   Reproducibility: use relative paths, set random seeds, and record Python/package versions. Provide minimal run steps for end-to-end execution.

   (a) **Data retrieval [5 points]**
   Obtain, cite, and briefly summarize the dataset. Identify the target variable. Perform a principled split into train/validation/test (e.g., 70/15/15 or via stratified k-fold CV for model selection plus a held-out test). State the seed and rationale for the split design.

   (b) **Data quality & cleaning [5 points]**
   Describe and apply cleaning steps (duplicates, missingness, incorrect entries). Show summary statistics and sanity checks. Identify potential outliers; justify your treatment (e.g., flag via IQR thresholds and compare robustness of results with and without outliers).

   (c) **Feature preprocessing [10 points]**
   Apply appropriate transforms (standardization/normalization; categorical encoding with `OneHotEncoder`; date-time handling). Justify each step, show pre/post distributions, and ensure all transforms are fit on train and applied unchanged to validation/test to avoid leakage [13]. If you use any feature ranking/selection or dimensionality reduction prior to modeling, specify where it occurs in the pipeline and why (e.g., PCA for visualization vs. supervised selection embedded in a model).

   Recommended public datasets (choose one; all support *both* Supervised Learning and Unsupervised Learning extensions):

   Hugging Face Datasets — `adult` (Census Income): classic structured classification (predict \$50K+). Balanced feature mix (categorical + numeric) makes it ideal for cleaning, one-hot encoding, scaling, and fairness analysis. Unsupervised: cluster socioeconomic profiles or run anomaly detection on rare categories. (Load via `datasets.load_dataset("adult")`).

   Hugging Face Datasets — `covertype`: large tabular dataset predicting forest cover type from cartographic features; good stress test for preprocessing and model scalability. Unsupervised: cluster ecoregions or compare internal validity indices across k. (Load via `datasets.load_dataset("covertype")`).

   Kaggle — Telco Customer Churn: binary classification with rich categoricals (contracts, services, billing) for end-to-end encoding, imputation, and class-imbalance handling. Unsupervised: segment customers for retention strategies; compare k-means vs. GMM.

[1] Avoid leakage: fit any scalers/encoders/imputers on train only and apply to val/test.

[2] For high-cardinality categoricals, motivate your encoding strategy. Avoid using ordinal encodings that induce spurious order unless justified.

Kaggle — Credit Card Fraud Detection: highly imbalanced binary classification; excellent for rigorous preprocessing (scaling, stratified splits) and appropriate metrics (PR curves). Unsupervised: anomaly detection (e.g., Isolation Forest) as a baseline to compare against supervised models.

Deliverables:

i. Concise dataset card (source, license, features, target, size; exact split sizes) and a link or script to fetch the data.

ii. Cleaning report (duplicates, missingness table, outlier analysis), with before/after summaries.

iii. Preprocessing pipeline diagram or pseudocode (`pseudocode` → `code` → analysis → conclusions) and evidence that transforms were learned on train only.

iv. (Optional, encouraged) Brief feature-ranking/selection or DR experiment (e.g., PCA for visualization) and what it revealed for modeling.

2. **Supervised Learning (Required) — Module 4**

40 points total (see per-problem grading breakdown below).

Use the same dataset you selected and preprocessed in Problem 1. Your splits (train/val/test) and preprocessing choices from Problem 1 must carry forward *unchanged* here (fit on train, applied to val/test) to avoid leakage. If you decide to switch datasets, you must update Problem 1 accordingly and keep the provenance/splits consistent across both problems.

Build a classification pipeline [9, 1, 13] and compare multiple supervised learners aligned with Module 4 (e.g., Logistic Regression, k-NN, Naive Bayes, SVM, Decision Trees, Random Forests, Gradient Boosting, and/or a shallow MLP). Use the training data for model development; reserve the held-out test set strictly for the final report. Report metrics beyond accuracy (precision, recall, F1, ROC-AUC, PR-AUC) and interpret model behavior (e.g., coefficients, feature importances, or partial effects) [8].

(a) **Model development [10 points]**
Implement and compare at least *three distinct families* of classifiers, e.g.: Logistic Regression, SVM (linear or RBF) [5], Random Forests [2], Gradient Boosting, k-NN, Naive Bayes, or a shallow MLP. Use `Pipeline` to include the preprocessing from Problem 1 (e.g., `ColumnTransformer`) so that all steps are reproducible and leakage-free [13]. Provide a brief rationale for each model choice (e.g., linear vs. non-linear boundary, robustness, interpretability), connecting to Module 4's algorithm characteristics.

(b) **Hyperparameter optimization & model selection [20 points]**
Perform tuned model selection with stratified k-fold cross-validation (*e.g.*, k=5 or 10). Use `GridSearchCV` or `RandomizedSearchCV` with scoring suited to your problem (e.g., F1 or PR-AUC for class imbalance) [13]. Clearly document the search space and justify key hyperparameters, for example:

LR: `C`, penalty

SVM: kernel, `C`, `gamma`

RF: `n_estimators`, `max_depth`, `min_samples_leaf`, `max_features`

GB: `n_estimators`, learning rate, `max_depth`

MLP: `hidden_layer_sizes`, `alpha`, activation, solver

Summarize CV results in a compact table (mean ± std). Select the best model based on your primary metric and discuss trade-offs across secondary metrics.

```
# Illustrative pattern: pipelines + stratified CV + model selection
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# X_train, y_train, X_val, y_val, X_test, y_test prepared in Problem 1
# numeric_features, categorical_features defined in Problem 1

pre = ColumnTransformer([
    ("num", StandardScaler(), numeric_features),
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features),
])

pipe_lr  = Pipeline([("pre", pre), ("clf", LogisticRegression(max_iter=2000))])
pipe_svm = Pipeline([("pre", pre), ("clf", SVC(probability=True))])
pipe_rf  = Pipeline([("pre", pre), ("clf", RandomForestClassifier())])
```

```
param_lr = {"clf__C": [0.01, 0.1, 1, 10], "clf__penalty": ["l2"]}
param_svm = {"clf__kernel": ["linear", "rbf"], "clf__C": [0.1, 1, 10],
             "clf__gamma": ["scale", "auto"]}
param_rf = {"clf__n_estimators": [200, 500],
            "clf__max_depth": [None, 5, 10],
            "clf__min_samples_leaf": [1, 2, 5]}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grids = []
for pipe, params in [(pipe_lr, param_lr), (pipe_svm, param_svm), (pipe_rf, param_rf)]:
    grid = GridSearchCV(pipe, params, scoring="f1", cv=cv, n_jobs=-1, refit=True)
    grid.fit(X_train, y_train)
    grids.append(grid)

# Pick best by mean CV score
best = max(grids, key=lambda g: g.best_score_)
print("Best model:", best.best_estimator_)
print("Best params:", best.best_params_, "CV F1:", best.best_score_)
```

(c) **Final model, robust evaluation & interpretation [10 points]**
Retrain the selected model on *train+validation* and evaluate once on the held-out test set. Report a full classification report (precision, recall, F1), ROC and PR curves, and a confusion matrix; if classes are imbalanced, emphasize PR-AUC and class-specific metrics [8]. Provide model interpretation: coefficients (LR), support vectors/kernels (SVM), or feature importances (tree ensembles). Quantify uncertainty (e.g., bootstrap CIs on the primary metric) and discuss error costs and ethical considerations if relevant to the task.

3. **Unsupervised Learning for Data Science (Optional; choose exactly one of 3/4/5) Module 5**

40 points total (see per-problem grading breakdown below).

Use the *same dataset* and *preprocessing* you established in Problem 1. Your feature matrix X for clustering/anomaly detection must be derived from the identical cleaning, encoding, and scaling steps (fit on **train** only; apply unchanged to validation/test partitions if you use them for external checks) to ensure comparability and avoid leakage [13]. If you change datasets, update Problem 1 accordingly and keep provenance/splits consistent across both problems.

Following Module 5, design a rigorous unsupervised workflow that compares multiple methods and validates results with *internal indices* and, when labels exist, *external indices*. Emphasize K-Means (with the *Elbow Method* for k selection; see pp. 1–5 and p. 7), Hierarchical Clustering (dendrogram/Linkage; pp. 6–8), and at least one complementary approach (e.g., Gaussian Mixture via EM, DBSCAN; pp. 19–23). Provide clear, reproducible code and well-labeled figures.

(a) **Data alignment & documentation [5 points]**
Briefly summarize the *same* dataset from Problem 1 (source, license, features, target if available) and restate the exact preprocessing carried over (encoding, scaling). If labels exist, explain how they will be used *only* for external validation (e.g., ARI/NMI) after clustering.

(b) **Preprocessing and dimensionality reduction (optional) [8 points]**
Confirm leakage-free transforms (fit on train). If you use PCA/UMAP for visualization, report explained variance (PCA) and show 2D/3D projections. Discuss why scaling matters for distance-based methods.

(c) **Clustering/anomaly models [15 points]**
Fit at least *three* methods, including:

- **K-Means** with k chosen via the *Elbow Method* (pp. 1–5, 7) [12];
- **Hierarchical** (agglomerative) with a chosen linkage (Ward/complete/average) and a dendrogram (pp. 6–8);
- **One of:** Gaussian Mixture via EM [6], **DBSCAN** (density-based; pp. 19–23), or another module-approved method (e.g., SOM for visualization).

For DBSCAN, justify (ε, *minPts*) and comment on sensitivity to scale; for hierarchical, justify linkage and dendrogram cut; for GMM, justify the number of components.

(d) **Model selection, stability, and validation [8 points]**
Use *internal indices* to compare models: **Silhouette** [14], **Calinski–Harabasz** [4], and **Davies–Bouldin** (DB). For K-Means, show inertia/WCSS and an elbow plot. Discuss stability across seeds (and, where relevant, linkages/initializations). If labels exist, add *external* indices (e.g., ARI/NMI) but do *not* use labels for fitting.

(e) **Cluster profiling and insights [4 points]**
Describe cluster sizes, centroids/medoids, and discriminative features; provide 2–3 actionable insights (business/scientific). If you ran DBSCAN, discuss the role and meaning of the "noise" cluster.

Illustrative pattern (scikit-learn):

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.metrics import (silhouette_score, calinski_harabasz_score,
                             davies_bouldin_score)

# X: numeric feature matrix after Problem 1 preprocessing (fit on train only)
X_scaled = StandardScaler().fit_transform(X)  # if scaling used in P1
```

```python
# --- KMeans with elbow ---
ks, inertias = range(2, 11), []
for k in ks:
    km = KMeans(n_clusters=k, n_init=20, random_state=42).fit(X_scaled)
    inertias.append(km.inertia_)
best_k = 3  # choose via elbow inspection

# --- Compare three methods ---
kmeans = KMeans(n_clusters=best_k, n_init=20, random_state=42).fit(X_scaled)
agg    = AgglomerativeClustering(n_clusters=best_k, linkage="ward").fit(X_scaled)
gmm    = GaussianMixture(n_components=best_k, covariance_type="full",
                         random_state=42).fit(X_scaled)
labels = {
    "kmeans": kmeans.labels_,
    "agg":    agg.labels_,
    "gmm":    gmm.predict(X_scaled),
}

def scores(y):
    return (silhouette_score(X_scaled, y),
            calinski_harabasz_score(X_scaled, y),
            davies_bouldin_score(X_scaled, y))

for name, y in labels.items():
    sil, ch, db = scores(y)
    print(f"{name:7s}  Sil={sil:.3f}  CH={ch:.1f}  DB={db:.3f}")

# Optional: DBSCAN sensitivity analysis
for eps in [0.3, 0.5, 0.7]:
    db = DBSCAN(eps=eps, min_samples=5).fit(X_scaled)
    if len(set(db.labels_)) > 1 and -1 in db.labels_:
        sil = silhouette_score(X_scaled[db.labels_!=-1], db.labels_[db.labels_!=-1])
        print(f"DBSCAN eps={eps}  clusters={len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)}  S
```

Notes:

i. Keep the *exact* Problem 1 preprocessing embedded in a `Pipeline`/`ColumnTransformer` whenever possible to ensure repeatability [13].

ii. K-Means assumptions and elbow usage follow Module 5 (pp. 1–5, 7); hierarchical linkage/dendrogram usage follows pp. 6–8; DBSCAN guidance (density/parameters) follows pp. 19–23.

iii. Report at least Silhouette, Calinski–Harabasz, and Davies–Bouldin; if ground-truth labels exist, add ARI/NMI as secondary checks.

4. **Sequential Decision Making / Reinforcement Learning — Module 6**

40 points total (see per-problem grading breakdown below). Optional: You may complete this as your third problem.

This problem explores *sequence modeling for control with offline trajectories*, informed by Module 6 topics (value-based RL with Q-learning/SARSA and Deep RL with neural approximators). You may (A) train a *sequence model* (Transformer-style) on offline replay and evaluate in a Gym environment, or (B) implement a *value-based RL* baseline (Q-learning/SARSA or Deep Q-Learning) and compare behaviors/returns. Clearly state your design choices, training details, and evaluation protocols; discuss differences between traditional RL pipelines and sequence modeling [15, 16, 17, 3].

(a) **Problem framing & environment (5 points)**
Define the state, action, and reward. Specify discount factor γ, episode termination, and any reward shaping. If you use offline replay, describe the trajectory format (observations, actions, rewards, terminals, timeouts) and any return-to-go or normalization used.

(b) **Dataset & preprocessing (5 points)**
Use one of the recommended datasets below. Document how you load, filter, and batch trajectories; show basic stats (episodes, horizon, reward ranges). For offline training, ensure *no environment interactions* during training; use the environment only for final evaluation rollouts.

(c) **Model(s) & training (20 points)**
Sequence model track: Train a Transformer-style policy/value model on offline replay (*e.g.*, conditioning on states, actions, return-to-go; teacher-forced rollouts). Report training loss and validation control metrics.
Value-based baseline track: Implement Q-learning or SARSA (tabular or function approximator). For Deep Q-Learning, describe replay buffer and target network updates. In both tracks, report hyperparameters (optimizer, LR, batch size, context length, exploration schedule) and justify them.

(d) **Evaluation & analysis (7 points)**
Evaluate in the chosen Gym environment: average episodic return over $N \geq 10$ seeds, success rate, and (if relevant) constraint violations. Include ROC/PR-style diagnostics for success/failure tasks when appropriate. Provide uncertainty estimates (bootstrap or across seeds) and compare sequence vs. baseline methods.

(e) **Ablation & discussion (3 points)**
Run at least one ablation (e.g., context length, reward scaling, replay filtering, exploration ϵ). Discuss lessons about exploration–exploitation trade-offs, stability (replay/targets), and limitations of offline training.

Recommended datasets (choose one):

Hugging Face Datasets — `edbeeching/decision_transformer_gym_replay`: Offline trajectories for classic Gym tasks (e.g., `hopper-medium-v2`, `halfcheetah-medium-expert-v2`, `walker2d-medium-v2`). Suited for sequence models and for evaluating against value-based baselines. Load with `datasets.load_dataset("edbee "<task>")`.

Kaggle — Huge Stock Market Dataset: Time-series data for simulating a trading RL environment (states: technical indicators; actions: buy/hold/sell; reward: portfolio P&L). Good for comparing tabular Q-learning/SARSA vs. deep approximators under realistic noise/constraints (see Module 6 notes).

Illustrative loaders (sequence model track and value-based baseline):

```
# Sequence-model track: load offline replay
from datasets import load_dataset
ds = load_dataset("edbeeching/decision_transformer_gym_replay", "hopper-medium-v2")
print(ds)  # ds["train"] has: observations, actions, rewards, terminals, timeouts, etc.
```

```python
# Example: pack fixed-length context windows (states, actions, returns-to-go) for training
def make_context_batches(dataset, context_len=20):
    # Implement: slice trajectories -> windows with (s_{t:t+K}, a_{t:t+K}, RTG_{t:t+K})
    ...

# Value-based baseline: tabular Q-learning/SARSA scaffold (e.g., FrozenLake or discretized env)
import numpy as np
nS, nA = 100, 4   # example sizes for a discrete env
Q = np.zeros((nS, nA))
alpha, gamma, eps = 0.1, 0.99, 0.1
for episode in range(5000):
    s = env.reset()
    done = False
    while not done:
        a = np.random.randint(nA) if np.random.rand() < eps else np.argmax(Q[s])
        s2, r, done, info = env.step(a)
        Q[s, a] += alpha * (r + gamma * np.max(Q[s2]) - Q[s, a])  # Q-learning update
        s = s2
```

Notes:

i. Keep seeds fixed and report them.

ii. For offline training, do not query the environment during training;

iii. For Deep Q-Learning, describe replay buffer sampling and target network updates.

iv. Provide environment wrappers (reward scaling/clipping) if used and justify them.

5. **Computer Vision for Data Science (Optional; choose exactly one of 3/4/5) — Module 7**

40 points total (see per-problem grading breakdown below). Optional: You may complete this as your third problem.

Design an end-to-end computer vision experiment that compares (**A**) a *from-scratch* CNN and (**B**) a *transfer-learning* approach (e.g., ResNet-50) for **image classification**, *or* implements a modern **object detector** (e.g., YOLO-style or a Transformer-based detector) with proper IoU/NMS handling. Incorporate practices emphasized in Module 7: principled augmentation, overfitting control, and task-appropriate metrics (accuracy/F1 for classification; mAP for detection). Discuss ethical considerations (privacy/bias) where relevant. [10, 11, 7]

(a) **Task framing & dataset card [5 points]**
Choose either **classification** or **detection**. Provide a concise dataset card (source, license, classes, size, splits) and specify normalization, resize policy, and label schema. If you used an image dataset in Problem 1, you may continue with the same dataset for continuity; otherwise select from the recommended list below.

(b) **Preprocessing & augmentation [8 points]**
Implement resizing, normalization (e.g., ImageNet mean/std), and *on-the-fly* augmentation (random crop/flip/color jitter, CutOut/MixUp/CutMix if appropriate). Address class imbalance (weights or sampling). Explain why these choices reduce overfitting in the context of your model capacity and dataset scale.

(c) **Model development [17 points]**
Classification track: Compare (A) a compact CNN you design from scratch vs. (B) transfer learning (e.g., ResNet-50 fine-tuned). Specify which layers are frozen/unfrozen, optimizer/schedule, and regularization (weight decay, dropout, label smoothing).
Detection track: Implement a detector (e.g., YOLO-style or a Transformer-based detector) with IoU-based assignment and NMS. Document anchor/box settings (if applicable) and training schedule.
In both tracks, report hyperparameters (batch size, LR schedule, epochs, image size) and total training time/compute.

(d) **Evaluation & error analysis [7 points]**
Classification: Report macro-F1 (primary), accuracy, per-class precision/recall, ROC/PR curves, and a confusion matrix; include a small *calibration* or *confidence* analysis if possible.
Detection: Report mAP at IoU=0.50 and, if feasible, mAP@[.50:.95]; show qualitative detections and analyze common failure modes (missed small objects, wrong class, duplicate boxes).
Summarize findings in a table (mean ± std over $\geq$3 seeds) and highlight trade-offs (accuracy vs. compute/latency).

(e) **Ablation & discussion [3 points]**
Ablate at least one factor (e.g., augmentation on/off, image size, freezing strategy, LR schedule) and explain its impact. Reflect on ethical aspects (e.g., bias in faces or sensitive categories) relevant to your dataset.

Recommended datasets (choose one; classification or detection):

Hugging Face — `food101` (classification): 101 classes of food; balanced scale and strong intra-class variation. Great for augmentation and transfer learning. (`datasets.load_dataset("food101")`)

Hugging Face — `oxford_iiit_pet` (classification): Fine-grained cats/dogs; good for per-class error analysis and feature transfer. (`datasets.load_dataset("oxford_iiit_pet")`)

Hugging Face — `cifar100` (classification): Small images, 100 classes; encourages careful augmentation and regularization. (`datasets.load_dataset("cifar100")`)

Kaggle — Global Wheat Detection (detection): Field images with wheat-head boxes; ideal for IoU/NMS/mAP evaluation and data augmentation for scale variance.

Kaggle — RSNA Pneumonia Detection (detection): Bounding boxes on chest X-rays; strong example for medical imaging detection with careful augmentation/ethics.

Illustrative starters (PyTorch/HF):

```
# --- Classification (transfer learning) ---
import torch, torchvision as tv
from torchvision import transforms as T
from torch import nn, optim

# Data
size = 224
tf_train = T.Compose([T.Resize((size,size)), T.RandomHorizontalFlip(),
                      T.ColorJitter(0.2,0.2,0.2,0.1),
                      T.ToTensor(),
                      T.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])])
tf_val   = T.Compose([T.Resize((size,size)), T.ToTensor(),
                      T.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])])

# Model
model = tv.models.resnet50(weights=tv.models.ResNet50_Weights.IMAGENET1K_V2)
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Train loop (sketch)
opt = optim.AdamW(model.parameters(), lr=3e-4, weight_decay=1e-4)
# ... standard PyTorch training with early stopping/checkpointing

# --- Detection (Transformer-based) ---
from transformers import YolosImageProcessor, YolosForObjectDetection
proc = YolosImageProcessor.from_pretrained("hustvl/yolos-small")
det  = YolosForObjectDetection.from_pretrained("hustvl/yolos-small")
# Prepare a batch: pixel_values, pixel_mask, labels (boxes in [x0,y0,x1,y1] and class ids)
# outputs = det(pixel_values=pv, labels=labels); loss = outputs.loss; loss.backward()
```

Notes:

i. For transfer learning, specify which layers are frozen/unfrozen and why.

ii. For detection, clearly define IoU thresholds and your NMS strategy.

iii. Include compute/latency considerations and any ethical issues germane to your dataset (privacy, consent, bias).

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. URL: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[2] Leo Breiman. *Random Forests*. Apr. 2001. URL: https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf.

[3] Greg Brockman, Vicki Cheung, et al. "OpenAI Gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[4] Tadeusz Calinski and Jerzy Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics* 3.1 (1974), pp. 1–27.

[5] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[6] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[7] Hugging Face. *Transformers: State-of-the-Art Natural Language Processing*. Accessed: 2024-05-16. 2024. URL: https://huggingface.co/docs/transformers.

[8] Tom Fawcett. *An Introduction to ROC Analysis*. Vol. 27. 8. Elsevier, 2006, pp. 861–874.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. 2nd. Springer, 2009.

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: (2012), pp. 1097–1105.

[12] J MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. University of California Press. 1967, pp. 281–297.

[13] Fabian et al. Pedregosa. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[14] Peter J Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[16] Ashish Vaswani et al. "Attention Is All You Need". In: vol. 30. 2017.

[17] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing*. https://arxiv.org/abs/1910.03771. 2020. arXiv: arXiv:1910.03771 [cs.CL].