Machine Learning Starter Kit

---

1) What is Machine Learning?

Machine Learning (ML) is a collection of methods that learn patterns from data to make predictions or decisions without being explicitly programmed for each rule.

- **Input → Model → Output**: Given examples (data), an algorithm fits a *model* that maps inputs to outputs.
- **Goal**: Generalize—perform well on *new, unseen* data, not just the training data.

Everyday examples: photo tag suggestions, spam filtering, product recommendations, speech recognition, fraud detection.

---

2) Major Types of ML

1. **Supervised Learning** — Learn from labeled examples.
2. *Tasks*: **Classification** (discrete labels), **Regression** (continuous values)
3. *Examples*: Digit recognition, house price prediction
4. **Unsupervised Learning** — Find structure in unlabeled data.
5. *Tasks*: **Clustering** (k-means), **Dimensionality reduction** (PCA)
6. *Examples*: Customer segmentation, anomaly detection
7. **Semi-Supervised Learning** — Mix of labeled + unlabeled data to improve performance when labels are scarce.
8. **Self-Supervised Learning** — Create labels from the data itself (e.g., predicting masked words/images); often used to pretrain large models.
9. **Reinforcement Learning (RL)** — Learn by trial and error with rewards (games, robotics, operations).
10. **Other distinctions**:
11. **Online vs. Batch** learning (streaming updates vs. periodic retraining)
12. **Generative vs. Discriminative** models

---

3) End-to-End ML Workflow (at a glance)

```
Data → Split → Preprocess/Normalize → Train → Validate/Evaluate → Serialize →
Load & Predict → Monitor
```

- **Data acquisition**: collect/clean data; record provenance and licenses.
- **Split**: hold-out test set (and optionally validation set) *before* peeking at results.
- **Preprocess/Normalize**: handle missing values, scale features, encode categories.
- **Train**: fit the model on the training set.

- **Validate/Evaluate**: tune hyperparameters; measure generalization.
- **Serialize**: save the trained model artifact (e.g., `joblib` file).
- **Load & Predict**: reload model for inference; ensure same preprocessing.
- **Monitor**: check for drift; retrain as needed.

Good practice defaults

- Keep a **strict test set** (never used in training or tuning).
- Use **cross-validation** for robust estimates when data is limited.
- Prefer **simple baselines first** (logistic regression, linear models).
- Track **metrics** and **random seeds** for reproducibility.
- Save the **preprocessing steps** together with the model or document them clearly.

---

4) Essential Concepts & Glossary

- **Feature**: Measurable property used by the model.
- **Label/Target**: The thing you want to predict.
- **Sample/Instance**: One row/example in your dataset.
- **Overfitting**: Model memorizes training data, performs poorly on new data.
- **Underfitting**: Model is too simple; misses important patterns.
- **Bias/Variance**: Systematic error vs. sensitivity to data fluctuations.
- **Regularization**: Techniques (e.g., L2) to discourage overly complex models.
- **Hyperparameters**: Settings chosen before training (e.g., learning rate, C).
- **Cross-Validation (CV)**: Repeated train/validation splits to estimate performance.
- **Normalization/Standardization**: Scale features (e.g., zero mean, unit variance).
- **Confusion Matrix**: Table of TP/FP/TN/FN for classification.
- **Precision/Recall/F1**: Metrics focusing on positive predictions and coverage.
- **ROC-AUC/PR-AUC**: Threshold-independent metrics for classifiers.
- **MSE/RMSE/MAE**: Regression error metrics.
- **Pipeline**: Chaining preprocessing and model steps together.
- **Serialization**: Saving model/preprocessing to disk to reuse later.

---

5) Popular Datasets & Sources

- **Kaggle** (competitions, curated datasets, notebooks). Useful for realistic projects. Requires free account and API token for programmatic download.
- **Hugging Face Hub** (datasets, models). Strong coverage for text, vision, audio; easy programmatic access.
- **UCI Machine Learning Repository** (classic academic datasets).
- **scikit-learn built-ins** (e.g., digits, iris, wine): great for teaching.

  Tip: Start on built-in datasets (no auth, tiny downloads), then move to Kaggle/Hugging Face to practice real-world data handling.

---

6) Evaluation & Experimentation

- **Train/Validation/Test**: 60/20/20 or 70/15/15 are common starting points.
- **Baselines**: Always compare to a simple baseline (e.g., majority class) and a simple model (e.g., logistic regression).
- **Model selection**: Use validation or CV—not the test set—to choose models.
- **Reporting**: Include metric definitions, chosen splits, hyperparameters, and random seeds.

---

7) Common Pitfalls & How to Avoid Them

- **Data leakage**: Accidentally using test information during training. *Fix*: split early; fit scalers/encoders only on training data.
- **Mismatch preprocessing**: Different preprocessing at train vs. predict time. *Fix*: save preprocessors or use a single pipeline.
- **Imbalanced classes**: Accuracy can be misleading. *Fix*: use precision/recall/F1, stratified splits, resampling, or class weights.
- **Non-reproducible results**: Randomness and ad hoc code. *Fix*: set seeds, fix versions, log experiments.

---

8) Minimal Hands-On Example (Digits Classification)

Goal: Recognize hand-written digits using a simple, readable pipeline in Python.

Steps covered: acquire → split → normalize → train → validate → serialize → load/predict.

Tools: `scikit-learn` (datasets, model, metrics), `joblib` (save/load). Minimal imports, no heavy frameworks.

> See the provided `digit_pipeline.py` file for a fully runnable script with comments your students can follow line-by-line.

---

9) Where Kaggle & Hugging Face Fit In (for Data Collection)

- **Kaggle**:

- Create an account → Account settings → Create API token.

- Install `kaggle` CLI locally, place `kaggle.json` in `~/.kaggle/`.
- `kaggle datasets download -d <owner/dataset>` to fetch zip files.

- Unzip, inspect README/licenses, then proceed with your pipeline.

- **Hugging Face Datasets**:
- `pip install datasets`
- `from datasets import load_dataset` → `load_dataset("mnist")`, etc.
- Convert to NumPy/Pandas, then follow the same pipeline steps.

The project starter file (`ml_projects_starter.py`) includes optional snippets and TODOs for both sources.

---

10) Training Process: Parameters, Epochs & Optimization

10.1 Core terminology

- **Loss function**: A number the optimizer tries to *minimize* (e.g., cross-entropy for classification, MSE for regression).
- **Parameters**: Values learned from data (e.g., weights/coefficients).
- **Hyperparameters**: Settings you choose before training (e.g., learning rate, `C`, depth, batch size, number of epochs).
- **Epoch**: One full pass over the training set.
- **Batch (minibatch)**: A small subset of the data used to compute an update.
- **Iteration / Step**: One optimizer update (one batch processed).
- **Optimization**: The procedure that adjusts parameters to reduce loss (e.g., gradient descent variants).
- **Convergence**: When further updates change the loss/parameters only slightly (meeting a tolerance).

In **scikit-learn**: many solvers (e.g., `lbfgs` in `LogisticRegression`) use *iterations* with a stopping **tolerance** (`tol`) and **max_iter**. `SGDClassifier/Regressor` expose **epochs**, **batch size**, and **learning rate** more directly.

10.2 Common optimization algorithms

- **Batch Gradient Descent**: Uses the whole dataset per step (stable but slow).
- **Stochastic Gradient Descent (SGD)**: Uses one sample per step (fast but noisy).
- **Mini-batch SGD**: Uses small batches (common default; good trade-off).
- **Momentum / Nesterov**: Adds velocity to smooth updates and accelerate.
- **Adaptive methods (Adam/Adagrad/RMSProp)**: Scale learning rates per parameter (fast, often good defaults for deep nets).

10.3 Key hyperparameters (what they mean & typical effects)

- **Learning rate (η)**: Size of each update.
- Too high → divergence/instability; too low → slow/underfitting.
- Often the *most important* hyperparameter to tune.

- **Batch size**:
- Small batches → noisier gradients, better generalization sometimes.
- Large batches → faster per-epoch on GPUs, may need **learning-rate warm-up**.
- **Epochs / max_iter**:
- Too few → underfitting; too many → overfitting unless regularized/early stopped.
- **Regularization strength** (e.g., `C` in logistic regression, `alpha` / `lambda` elsewhere):
- Higher regularization → simpler models, less overfitting, possibly lower training accuracy but better test performance.
- **Tolerance (``) & early-stopping criteria**:
- Stop when improvement falls below a threshold for N checks.
- **Model-specific**:
- Trees/forests: depth, number of estimators, min samples per split/leaf.
- SVM: kernel, `C`, `gamma`.
- kNN: number of neighbors `k`, distance metric.

10.4 Preventing overfitting (regularization & friends)

- **L2 / L1 regularization**: Penalize large weights; L1 can induce sparsity.
- **Early stopping**: Monitor validation loss; stop when it stops improving.
- **Data augmentation** (vision/text/audio): Create realistic variations of inputs.
- **Dropout** (deep learning): Randomly zero some activations during training.
- **Ensembling**: Combine models (bagging, boosting, stacking).
- **Proper split**: Keep a *strict* validation/test set; avoid data leakage.

For **linear models & SVMs**, **feature scaling** (standardization) is crucial. For **tree-based models**, scaling is usually unnecessary.

10.5 Learning-rate schedules (when training for many epochs)

- **Step decay**: Reduce η by a factor at fixed epochs.
- **Exponential/cosine decay**: Smoothly reduce η over time.
- **Warm-up**: Start with smaller η for a few epochs, then increase to target η.
- **Cyclical / One-cycle**: Vary η within ranges to escape sharp minima.

10.6 Choosing and monitoring metrics

- **Classification**: Accuracy, precision/recall/F1, ROC-AUC, PR-AUC; inspect the **confusion matrix**.
- **Regression**: MAE (robust), RMSE (penalizes large errors), R^2.
- Track **training vs. validation** curves to spot under/overfitting:
- Both low → underfitting (increase model capacity, train longer, reduce reg).
- Train low / Val high → overfitting (more reg, more data/augmentation, early stopping).

10.7 A practical tuning workflow (checklist)

1. **Start simple**: Small model + reasonable defaults; verify data pipeline & splits.
2. **Scale features** (if needed) and set a **reproducible seed**.
3. **Pick a baseline metric** and log it.
4. **Tune learning rate** (and batch size if applicable) first.
5. **Adjust model capacity / regularization** next.
6. **Use validation curves** or **cross-validation** for robust estimates.
7. **Introduce schedules/early stopping** to stabilize training.
8. **Lock the model** and evaluate once on the **held-out test set**.
9. **Serialize** the exact preprocessing + model; record versions, seeds, and hyperparameters.

10.8 Hyperparameter search strategies

- **Manual search**: Fast to start; good when you know what matters.
- **Grid search**: Systematic but can be wasteful in high-dimensional spaces.
- **Random search**: Surprisingly strong; explores more configurations under the same budget.
- **Bayesian / adaptive search** (e.g., TPE, Gaussian processes): Model the performance surface to choose promising trials.

  In **scikit-learn**: use `GridSearchCV` / `RandomizedSearchCV` with **pipelines** so preprocessing is fit only on training folds.

10.9 Troubleshooting guide

- **Training loss not decreasing**: Lower learning rate; check data/labels; verify scaling; increase `max_iter`.
- **Validation worse than training**: Add regularization, early stopping, augmentation; collect more data.
- **Model is slow**: Reduce batch size memory peaks (DL); simplify model; subsample features; use fewer estimators.
- **Unstable results**: Set random seeds; fix library versions; increase dataset size or use cross-validation.

10.10 Mapping to the digits example (logistic regression)

- Relevant knobs: `max_iter`, `tol`, **regularization** (`C`, `penalty`), and **scaler** choice.
- Typical path:
- Standardize features (already in pipeline).
- Increase `max_iter` until convergence; adjust `tol` if needed.
- Tune `C` on the validation set or via `GridSearchCV`.
- Serialize both **scaler + model**; reload to predict consistently.

11) Further Reading

• *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* (Aurélien Géron)
• *The Hundred-Page Machine Learning Book* (Andriy Burkov)
• scikit-learn User Guide & Tutorials (official)
• Hugging Face Datasets documentation
• Kaggle Learn micro-courses