

## CS 273P: Machine Learning and Data Mining

### Homework 2

Due date: **See Canvas**

Instructor: Xiaohui Xie

This homework is graded automatically on **Gradescope** using an autograder. All questions are designed to be **unit-testable**: you will implement specific functions that return the specified results. **No plots and no PDF writeup are required.**

**Allowed libraries:** You should use the standard Python scientific stack: `numpy`, `scipy`, `pandas`, and `scikit-learn`.

#### Submission rules (READ CAREFULLY):

1. Download the starter files from Canvas.
2. Complete your work by editing only the provided Python files: `problem1.py` and `problem2.py`.
3. Create a plain text file called `statement.txt` containing your collaboration statement.
4. Submit to Gradescope by uploading a single **ZIP file** containing exactly:
  - `problem1.py`
  - `problem2.py`
  - `statement.txt`

No folders/subdirectories; all files must be at the top level of the zip.

5. You may resubmit before the deadline; your **highest score** counts.

If you run into issues with the autograder or submission format, post on Ed Discussion so others can benefit.

**Summary:** (1) Edit `problem1.py`, `problem2.py`; (2) Add `statement.txt`; (3) Zip all three files and upload to Gradescope; (4) Autograder score determines your grade.

**Points:** This homework adds up to a total of **100 points**, as follows:

Problem 1: Linear/Polynomial Regression + Model Selection	60 points
Problem 2: Cross-Validation	35 points
Statement of Collaboration	5 points

### Problem 1: Linear/Polynomial Regression + Model Selection (60 points)

In this problem we explore regression, feature construction, rescaling, and model selection. You may use `numpy` for array operations and `scikit-learn` utilities/models where appropriate.

**Dataset.** Load the dataset `data/curve80.txt`. The first column is  $x$  and the second column is  $y$ . **Do not shuffle the data.** Split into 75% train and 25% test by taking the first 75% as training and the remaining as test.

All required functions for Problem 1 are in `problem1.py`. The autograder calls them directly.

1. **Data loading and split (10 points).** Implement:
  - `load_curve80(path) -> (X, y)`: return  $X \in R^{N \times 1}$  and  $y \in R^N$ .
  - `split_data(X, y, frac=0.75) -> (Xtr, Xte, ytr, yte)`: deterministic split without shuffling.
  - `shapes(Xtr, Xte, ytr, yte) -> (sxtr, sxte, sytr, syte)`: return the shapes as tuples.

2. **Baseline linear regression (20 points).** Fit a degree-1 regression model (with intercept) on the training set and evaluate MSE. Implement:
  - `fit_linear(Xtr, ytr)` -> `model`: fit a linear regression model with intercept (you may use `sklearn.linear_model.LinearRegression`).
  - `predict(model, X)` -> `yhat`: return predictions as a 1D array.
  - `mse(yhat, y)` -> `float`: return mean squared error.
  - `eval_linear(Xtr, ytr, Xte, yte)` -> `(mse_tr, mse_te)`.
3. **Polynomial regression with scaling (20 points).** Train polynomial regression models by expanding  $x$  to polynomial features and standardizing the features. You may use `sklearn.preprocessing.PolynomialFeatures` and `sklearn.preprocessing.StandardScaler`. Implement:
  - `make_poly_pipeline(degree)` -> `model`: return a fitted or configurable model/pipeline that (i) creates polynomial features up to `degree`, (ii) standardizes features, and (iii) performs linear regression with intercept. (*Hint*: `Pipeline(PolynomialFeatures, StandardScaler, LinearRegression)`).
  - `fit_poly(Xtr, ytr, degree)` -> `model`: fit the pipeline on training data.
  - `eval_poly(Xtr, ytr, Xte, yte, degree)` -> `(mse_tr, mse_te)`.
4. **Degree sweep + recommendation (10 points).** Train polynomial models for degrees  $d \in \{1, 3, 5, 7, 10, 18\}$  and compute train/test MSE for each. Implement:
  - `eval_degrees(Xtr, ytr, Xte, yte, degrees)` -> `(mse_tr, mse_te)` returning two 1D arrays aligned with degrees.
  - `recommend_degree(degrees, mse_te)` -> `int` returning the degree with the smallest test MSE; break ties by choosing the smaller degree.

## Problem 2: Cross-Validation (35 points)

In Problem 1 you selected a polynomial degree using a held-out test set. Now assume you do **not** use the test labels for selection, and instead use  $K$ -fold cross-validation on the training set  $(X_{tr}, y_{tr})$ .

All required functions for Problem 2 are in `problem2.py`.

1.  **$K$ -fold splitting (10 points).** Implement:
  - `kfold_indices(n, K)` -> `list_of_folds`: deterministically split indices  $\{0, \dots, n-1\}$  into  $K$  folds in order (no shuffling). Return a list of length  $K$ , each an integer array of validation indices.
  - `train_val_split(X, y, folds, i)` -> `(Xti, yti, Xvi, yvi)`: use fold  $i$  as validation and the rest as training.
2.  **$K$ -fold CV error (15 points).** Compute the average validation MSE of polynomial regression for a given degree. You may reuse your pipeline from Problem 1. Implement:
  - `cv_mse_poly(Xtr, ytr, degree, K)` -> `float`: perform  $K$ -fold CV on  $(X_{tr}, y_{tr})$ ; on each fold fit the polynomial+scaling+linear-regression pipeline using only the fold's training split, then compute validation MSE; return the average across folds.
3. **Degree selection by CV (10 points).** Using degrees  $d \in \{1, 3, 5, 7, 10, 18\}$ :
  - `cv_curve(Xtr, ytr, degrees, K=5)` -> `cv_ms`: return a 1D array of CV MSEs aligned with degrees.
  - `recommend_degree_cv(degrees, cv_ms)` -> `int`: return the degree with smallest CV MSE; break ties by choosing the smaller degree.

## Statement of Collaboration (5 points)

Add a plain text file `statement.txt` to your submission. List the names of collaborators and the nature of collaboration, or write “No collaboration.” if you worked alone. Do not share code. Academic honesty policies apply.