cs584 f25 - assignment 1 (regression)

Due by: 9/23/2025

General Instructions

Objectives

By the end of this assignment you will: (i) implement and evaluate linear, polynomial, and kernel regressors; (ii) study model complexity and the bias–variance tradeoff; (iii) apply ridge regularization; (iv) implement a robust regressor; (v) properly assess performance with cross-validation. You will use Python and NumPy.

- Unless a question explicitly allows it, implement algorithms *from scratch* (NumPy/SciPy/Matplotlib allowed; `scikit-learn` only for sanity-check baselines, dataset loading, and cross-validation).
- Use 10-fold cross-validation (CV) unless told otherwise. Fix a random seed and report it.
- Turn in a single python notebook with answers, figures, and tables.
- Include a 1–2 paragraph reflection at the end: key findings, pitfalls, and lessons learned.

Submission instructions:

- Upload your submission to Canvas.
- Submit a fully executed Python notebook with no gaps in execution. To receive full credit, please ensure the following:
  - The notebook includes all cell outputs and contains no error messages.
  - It is easy to match each problem with its corresponding code solution using clear markdown or comments (e.g., “Problem 2”).
  - Re-run your notebook before submitting to ensure that cells are numbered sequentially, starting at [1].

Datasets

- Provided (single-feature): `svar-set1.dat`, `svar-set2.dat`, `svar-set3.dat`, `svar-set4.dat`. Text, last column is y.
- Provided (multi-feature): `mvar-set1.dat`, `mvar-set2.dat`, `mvar-set3.dat`, `mvar-set4.dat`. Text, last column is y.
- Real (UCI or similar): pick one continuous-target dataset ($n \geq 500$, $d \geq 5$). `http://archive.ics.uci.edu/ml/index.php`
- Note: `mvar-set3.dat`, `mvar-set4.dat` have many data points (100,000) which will make the Gram matrix too large (cannot be stored or inverted). To solve the issue, when using the Gram matrix, use a subset of points instead of the entire set (an alternative not covered in this assignment is a numerical solution).

Performance Metrics

Report at minimum: RMSE, MAE, R^2. For cross validation (CV), report mean $\pm$ std across folds. When comparing models, include a concise table and a brief interpretation.

Programming

1. **Single-variable regression (linear & polynomial)**
    1. Load each of the single-feature sets: `svar-set1..4.dat`. Plot (x, y) scatter for each; visually comment on apparent complexity and noise.
    2. Implement ordinary least squares (OLS) for linear regression using:
        - Normal equations: $\theta = (X^\top X)^{-1} X^\top y$ with an intercept column (bias).
        - Gradient descent. Plot training MSE vs. epochs; show convergence.
    3. Evaluate linear models with 10-fold CV. Report RMSE/MAE/R^2 (mean $\pm$ std). Overlay fitted line on a held-out fold (figure).
    4. Implement polynomial features of degree $d \in \{2, 3, \ldots, 10\}$ (with intercept), standardize columns (zero-mean, unit-variance). Use 10-fold CV (Cross Validation) to select d by validation RMSE. Plot: degree vs. CV-RMSE (with error bars). Justify the chosen degree.
    5. Data ablation: randomly subsample training to $\{20\%, 40\%, 60\%, 80\%\}$ and re-evaluate linear and chosen polynomial model. Plot learning curves (train/test RMSE vs. train size). Briefly discuss bias vs. variance behavior you observe.
    6. Compare your best models to `sklearn.linear_model.LinearRegression` and `PolynomialFeatures`+`LinearRegression`. Report any differences (precision, conditioning).
2. **Multivariate regression (feature mapping & solvers)**
    1. Load each multi-feature set: `mvar-set1..4.dat`. Standardize X; keep raw y. Report basic stats (mean/var, correlations).
    2. Construct higher-dimensional maps: (i) pairwise products $x_i x_j$ for $i \leq j$; (ii) polynomial of total degree up to 3. Control explosion with an option to drop near-constant or highly collinear features (compute variance of each feature; if it's below a threshold, e.g., variance < 10^{-6}, drop it.).
    3. Fit OLS in mapped spaces using both (i) normal equations with *Tikhonov* regularization for numerical stability (

       $\theta = (Z^\top Z + \lambda I)^{-1} Z^\top y, \quad \lambda \approx 10^{-8}$) and (ii) gradient-based solver. Compare test RMSE and wall-clock time (actual elapsed time) for each mapping and solver; pick a final model per dataset with justification.
    4. Implement ridge regression (closed-form): $\theta = (X^\top X + \lambda I)^{-1} X^\top y$. Tune λ on a log grid (e.g., 10^{-6} to 10^3) via 10-fold CV. Plot validation RMSE vs. λ; overlay $\|\theta\|_2$ vs. λ. Discuss shrinkage (coefficients shrink towards 0) and model stability (condition number of the matrix that needs to be inverted). Discuss how λ affects bias and variance.
    5. Robust regression: Implement *Huber* loss regression. Compare OLS vs. Huber: RMSE/MAE and sensitivity to outliers. You should be able to observe improvement in some of the data sets. Explain why.
3. **Kernel methods (dual ridge / kernel ridge)**
    1. Implement kernel ridge in dual: $\alpha = (K + \lambda I)^{-1} y, \quad \hat{y}(x) = k(x, X)^\top \alpha$

with RBF kernel $k(x,x') = \exp\left(-\gamma\|x-x'\|^2\right)$. Use 10-fold CV to tune (λ, γ) on log grids. Plot a 2D heatmap of CV-RMSE over (λ, γ).
    2. Compare primal ridge (on explicit polynomial map) vs. kernel ridge (RBF) on one `mvar` dataset: accuracy, runtime, and memory (show n, d, and effective feature count).

4. **Real-data study**
    1. Pick a UCI (or similar) regression dataset with a continuous target ($n \geq 500$, $d \geq 5$). Document: source, preprocessing (missing, scaling, categorical encoding), and train/test split strategy.
    2. Evaluate: (i) OLS; (ii) ridge (with tuned λ); (iii) robust (Huber with tuned δ); (iv) kernel ridge (RBF with tuned γ). Use 10-fold CV on training, then one final hold-out test. Report a table of metrics (CV mean $\pm$ std and final test scores).
    3. Model complexity & bias–variance: produce learning curves for (i) and (ii) (at least 5 train sizes), plus a variance estimate via repeated CV. Provide a concise interpretation linking capacity, regularization strength, and generalization.

Starter Code

```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import make_scorer, mean_absolute_error, r2_score

# ------------------- Hardcoded settings -------------------
DATA_PATH = "data/svar-set1.dat"   # change to svar-set2.dat, etc.
OUTDIR    = Path("out_q1_simple")  # output folder
POLY_DEG  = 6                      # degree for overlay visualization
SEED      = 42
OUTDIR.mkdir(parents=True, exist_ok=True)

# ------------------- Metrics -------------------
def rmse(y_true, y_pred):
    return float(np.sqrt(np.mean((np.asarray(y_true) - np.asarray(y_pred))**2)))

rmse_scorer = make_scorer(lambda yt, yp: -rmse(yt, yp))  # sklearn wants higher=better
mae_scorer  = make_scorer(mean_absolute_error, greater_is_better=False)
r2_scorer   = make_scorer(r2_score)

# ------------------- Utilities -------------------
def load_txt_dataset(path: str):
    arr = np.loadtxt(path)
    X, y = arr[:, :-1], arr[:, -1]
    return X, y

def print_cv(model, X, y, k=10, seed=42, label="model"):
```

```
    kf = KFold(n_splits=k, shuffle=True, random_state=seed)
    rmse_scores = -cross_val_score(model, X, y, scoring=rmse_scorer, cv=kf)
    mae_scores  = -cross_val_score(model, X, y, scoring=mae_scorer,  cv=kf)
    r2_scores   =  cross_val_score(model, X, y, scoring=r2_scorer,   cv=kf)

    print(f"[{label} | {k}-fold CV] "
          f"RMSE={rmse_scores.mean():.4f}±{rmse_scores.std():.4f}  "
          f"MAE={mae_scores.mean():.4f}±{mae_scores.std():.4f}  "
          f"R2={r2_scores.mean():.4f}±{r2_scores.std():.4f}")
    return rmse_scores.mean(), rmse_scores.std()

def overlay_plot(Xtr, ytr, Xte, yte, model, outpath, title):
    model.fit(Xtr, ytr)
    xs = np.linspace(Xtr.min(), Xtr.max(), 300).reshape(-1, 1)
    ycurve = model.predict(xs)

    plt.figure()
    plt.scatter(Xte, yte, s=16, alpha=0.8, label="test")
    plt.plot(xs, ycurve, lw=2, label="fit")
    plt.xlabel("x"); plt.ylabel("y"); plt.title(title)
    plt.grid(alpha=0.3); plt.legend()
    plt.savefig(outpath, bbox_inches="tight")

# ------------------- Main workflow -------------------
def main():
    # 1) Load & scatter
    X, y = load_txt_dataset(DATA_PATH)
    assert X.shape[1] == 1, "Single-feature dataset expected"
    plt.figure()
    plt.scatter(X, y, s=16, alpha=0.8)
    plt.xlabel("x"); plt.ylabel("y"); plt.title("Scatter: single-variable")
    plt.grid(alpha=0.3)
    plt.savefig(OUTDIR / "01_scatter.png", bbox_inches="tight")

    # 2) Linear regression (baseline)
    linear_pipeline = Pipeline([
        ("scaler", StandardScaler(with_mean=True, with_std=True)),
        ("linreg", LinearRegression())  # replace with your solver later
    ])
    print_cv(linear_pipeline, X, y, k=10, seed=SEED, label="Linear baseline")

    # Overlay linear fit
    Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.25, random_state=SEED)
    overlay_plot(Xtr, ytr, Xte, yte, linear_pipeline,
                 OUTDIR / "02_overlay_linear.png",
                 title="Linear fit (held-out overlay)")

    # 3–4) Polynomial degree selection
    degrees = list(range(2, 11))
    means, stds = [], []
    for d in degrees:
        poly_pipeline = Pipeline([
            ("scaler", StandardScaler(with_mean=True, with_std=True)),
            ("poly",   PolynomialFeatures(degree=d, include_bias=False)),
```

```
            ("linreg", LinearRegression())
        ])
        m, s = print_cv(poly_pipeline, X, y, k=10, seed=SEED, label=f"Poly deg={d}")
        means.append(m); stds.append(s)

    plt.figure()
    plt.errorbar(degrees, means, yerr=stds, marker="o")
    plt.xlabel("Polynomial degree"); plt.ylabel("CV RMSE")
    plt.title("Degree selection (10-fold CV)")
    plt.grid(alpha=0.3)
    plt.savefig(OUTDIR / "03_cv_degree.png", bbox_inches="tight")

    best_deg = degrees[int(np.argmin(means))]
    print(f"[Degree selection] Best degree = {best_deg}")

    # Overlay polynomial fit with chosen POLY_DEG
    best_poly = Pipeline([
        ("scaler", StandardScaler(with_mean=True, with_std=True)),
        ("poly",   PolynomialFeatures(degree=POLY_DEG, include_bias=False)),
        ("linreg", LinearRegression())
    ])
    overlay_plot(Xtr, ytr, Xte, yte, best_poly,
                 OUTDIR / f"04_overlay_poly_deg{POLY_DEG}.png",
                 title=f"Polynomial fit (deg={POLY_DEG})")

if __name__ == "__main__":
    main()
```