T4

22510064

ML ASSIGNMENT 8

# Import required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, KFold

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.pipeline import make\_pipeline

from sklearn.metrics import r2\_score

# -----------------------------

# Load the dataset

# -----------------------------

df = pd.read\_csv("polynomial\_regression.csv")

X = df[['x']].values

y = df['y'].values

# -----------------------------

# 1. Split data (80:20 train-test)

# -----------------------------

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# -----------------------------

# 2. Create 30 samples (each of size 20) and fit polynomials (degree 1 to 10)

#    Compute train error (1-R²), test error (1-R²) and error gap (train R² - test R²)

# -----------------------------

num\_samples = 30

sample\_size = 20

max\_degree = 10

train\_errors = {deg: [] for deg in range(1, max\_degree + 1)}

test\_errors = {deg: [] for deg in range(1, max\_degree + 1)}

gap\_errors = {deg: [] for deg in range(1, max\_degree + 1)}

np.random.seed(42)

for \_ in range(num\_samples):

    indices = np.random.choice(len(X\_train), sample\_size, replace=False)

    X\_sample = X\_train[indices]

    y\_sample = y\_train[indices]

    for degree in range(1, max\_degree + 1):

        # Fit polynomial regression model for the given degree

        model = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

        model.fit(X\_sample, y\_sample)

        y\_train\_pred = model.predict(X\_sample)

        y\_test\_pred = model.predict(X\_test)

        train\_r2 = r2\_score(y\_sample, y\_train\_pred)

        test\_r2 = r2\_score(y\_test, y\_test\_pred)

        train\_errors[degree].append(1 - train\_r2)  # lower error is better

        test\_errors[degree].append(1 - test\_r2)

        gap\_errors[degree].append(train\_r2 - test\_r2)

# -----------------------------

# Print summary statistics for each polynomial degree

# -----------------------------

print("Summary of Errors Across 30 Samples (for each polynomial degree):\n")

for deg in range(1, max\_degree + 1):

    train\_err\_mean = np.mean(train\_errors[deg])

    train\_err\_std = np.std(train\_errors[deg])

    test\_err\_mean = np.mean(test\_errors[deg])

    test\_err\_std = np.std(test\_errors[deg])

    gap\_mean = np.mean(gap\_errors[deg])

    gap\_std = np.std(gap\_errors[deg])

    print(f"Degree {deg}:")

    print(f"    Train Error (1-R²): {train\_err\_mean:.4f} ± {train\_err\_std:.4f}")

    print(f"    Test  Error (1-R²): {test\_err\_mean:.4f} ± {test\_err\_std:.4f}")

    print(f"    Gap (Train R² - Test R²): {gap\_mean:.4f} ± {gap\_std:.4f}\n")

# -----------------------------

# 3. Violin Plots for error metrics

# -----------------------------

# Prepare data for the violin plots

violin\_data\_test = pd.DataFrame({

    'Degree': sum([[deg] \* len(test\_errors[deg]) for deg in test\_errors], []),

    'Test Error (1 - R²)': sum([test\_errors[deg] for deg in test\_errors], [])

})

violin\_data\_gap = pd.DataFrame({

    'Degree': sum([[deg] \* len(gap\_errors[deg]) for deg in gap\_errors], []),

    'Train R² - Test R²': sum([gap\_errors[deg] for deg in gap\_errors], [])

})

# Plot 1: Degree vs Test Error (1 - R²)

plt.figure(figsize=(12, 6))

sns.violinplot(

    x='Degree',

    y='Test Error (1 - R²)',

    data=violin\_data\_test,

    hue='Degree',

    palette="coolwarm",

    legend=False

)

plt.title('Degree vs Test Error (1 - R²)')

plt.grid(True)

plt.tight\_layout()

plt.savefig("violin\_test\_error.png")

plt.show()

# Plot 2: Degree vs (Train R² - Test R²)

plt.figure(figsize=(12, 6))

sns.violinplot(

    x='Degree',

    y='Train R² - Test R²',

    data=violin\_data\_gap,

    hue='Degree',

    palette="viridis",

    legend=False

)

plt.title('Degree vs Gap (Train R² - Test R²)')

plt.grid(True)

plt.tight\_layout()

plt.savefig("violin\_gap\_error.png")

plt.show()

# -----------------------------

# 4. 5-Fold Cross-Validation on a sample of size 20 (Regularized version)

#    Using a pipeline: StandardScaler + PolynomialFeatures + Ridge(alpha=1.0)

#    This modification improves stability for higher degrees.

# -----------------------------

sample\_indices = np.random.choice(len(X\_train), sample\_size, replace=False)

X\_sample = X\_train[sample\_indices]

y\_sample = y\_train[sample\_indices]

best\_degree = 1

best\_score = -np.inf

print("5-Fold Cross-Validation Scores using Regularized Pipeline for each Degree on a sample of size 20:")

for degree in range(1, max\_degree + 1):

    model\_cv = make\_pipeline(StandardScaler(), PolynomialFeatures(degree), Ridge(alpha=1.0))

    scores = cross\_val\_score(model\_cv, X\_sample, y\_sample, cv=5, scoring='r2')

    mean\_score = scores.mean()

    print(f"    Degree {degree}: CV Scores = {scores}, Mean R² = {mean\_score:.4f}")

    if mean\_score > best\_score:

        best\_score = mean\_score

        best\_degree = degree

print("\nBest Degree chosen via Regularized 5-Fold CV on the sample:", best\_degree)

# Train final model (using the best degree) with the same regularized pipeline for consistency

final\_model = make\_pipeline(StandardScaler(), PolynomialFeatures(best\_degree), Ridge(alpha=1.0))

final\_model.fit(X\_sample, y\_sample)

y\_pred\_final = final\_model.predict(X\_test)

r2\_test\_cv = r2\_score(y\_test, y\_pred\_final)

print("Test R² for the Regularized CV-selected model (Degree {}): {:.4f}".format(best\_degree, r2\_test\_cv))

# -----------------------------

# 5. 10-Fold Cross-Validation on full training set with L1 and L2 Regularization

#    Now, both Ridge and Lasso use alpha=0.1.

# -----------------------------

kf = KFold(n\_splits=10, shuffle=True, random\_state=42)

ridge\_model = make\_pipeline(

    StandardScaler(),

    PolynomialFeatures(5),

    Ridge(alpha=0.1)

)

lasso\_model = make\_pipeline(

    StandardScaler(),

    PolynomialFeatures(5),

    Lasso(alpha=0.1, max\_iter=10000)

)

ridge\_scores = cross\_val\_score(ridge\_model, X\_train, y\_train, cv=kf, scoring='r2')

lasso\_scores = cross\_val\_score(lasso\_model, X\_train, y\_train, cv=kf, scoring='r2')

ridge\_model.fit(X\_train, y\_train)

lasso\_model.fit(X\_train, y\_train)

ridge\_r2\_test = r2\_score(y\_test, ridge\_model.predict(X\_test))

lasso\_r2\_test = r2\_score(y\_test, lasso\_model.predict(X\_test))

print("\n10-Fold CV for Regularized Models (using polynomial degree = 5) with alpha=0.1 for both:")

print("    Ridge CV Scores: ", ridge\_scores)

print("    Ridge Mean CV R²: {:.4f}".format(ridge\_scores.mean()))

print("    Ridge Test R² (alpha=0.1): {:.4f}".format(ridge\_r2\_test))

print("")

print("    Lasso CV Scores: ", lasso\_scores)

print("    Lasso Mean CV R²: {:.4f}".format(lasso\_scores.mean()))

print("    Lasso Test R² (alpha=0.1): {:.4f}".format(lasso\_r2\_test))

# -----------------------------

# 6. Final Results

# -----------------------------

print("\n----- Final Results -----")

print("Best Degree from Regularized 5-Fold CV on Sample:", best\_degree)

print("Test R² (Regularized CV-Selected Model, Degree {}): {:.4f}".format(best\_degree, r2\_test\_cv))

print("Ridge Test R² (Degree 5, alpha=0.1): {:.4f}".format(ridge\_r2\_test))

print("Lasso Test R² (Degree 5, alpha=0.1): {:.4f}".format(lasso\_r2\_test))

print("Ridge CV Mean R²: {:.4f}".format(ridge\_scores.mean()))

print("Lasso CV Mean R²: {:.4f}".format(lasso\_scores.mean()))

# -----------------------------

# 7. Single Plot Comparison of Predictions vs Test Data

# -----------------------------

# Sort test data so that lines can be plotted smoothly

sorted\_indices = np.argsort(X\_test.ravel())

X\_test\_sorted = X\_test[sorted\_indices]

y\_test\_sorted = y\_test[sorted\_indices]

# Predictions from the Regularized CV-selected final model

y\_pred\_final\_sorted = final\_model.predict(X\_test\_sorted)

# Predictions from the adjusted Ridge model

y\_pred\_ridge\_sorted = ridge\_model.predict(X\_test\_sorted)

# Predictions from the adjusted Lasso model

y\_pred\_lasso\_sorted = lasso\_model.predict(X\_test\_sorted)

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, color='gray', edgecolor='k', alpha=0.7, label='Test Data')

# Plot the best polynomial model from regularized sample CV

plt.plot(X\_test\_sorted, y\_pred\_final\_sorted, color='blue', lw=2, label=f'Degree {best\_degree} (Reg. CV)')

# Plot the Ridge model

plt.plot(X\_test\_sorted, y\_pred\_ridge\_sorted, color='red', lw=2, linestyle='--', label='Ridge (alpha=0.1)')

# Plot the Lasso model

plt.plot(X\_test\_sorted, y\_pred\_lasso\_sorted, color='green', lw=2, linestyle=':', label='Lasso (alpha=0.1)')

plt.title("Model Predictions vs. Test Data (Both Regularization Models with alpha=0.1)")

plt.xlabel("x")

plt.ylabel("y")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.savefig("model\_comparison\_plot\_alpha\_0.1.png")

plt.show()

# -----------------------------

# 8. Final Observations

# -----------------------------

observations = """

Observations:

1. Violin Plots:

   - The test error (1 - R²) decreases initially with polynomial degree and then increases, reflecting overfitting at high degrees.

   - The gap (Train R² - Test R²) increases for higher degrees.

2. 5-Fold CV (with regularization):

   - Using a standardized Ridge pipeline in CV stabilizes performance and eliminates the highly negative R² scores for high degrees.

   - The optimized degree (shown above) achieves a mean R² close to or exceeding 0.9.

3. Regularization:

   - Both Ridge and Lasso models using alpha=0.1 show improved generalization on the test set.

   - Their CV and test R² scores are consistent, indicating controlled model complexity.

4. Single Plot Comparison:

   - The final comparison plot overlays predictions from the CV-selected model, Ridge, and Lasso models.

   - This visualization helps in assessing which model provides the best fit across the range of x.

5. R² Scores:

   - Higher R² values reflect better performance on unseen data.

   - The comprehensive statistics guide further parameter tuning if necessary.

"""

print("\n----- Final Observations -----")

print(observations)

