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)
```

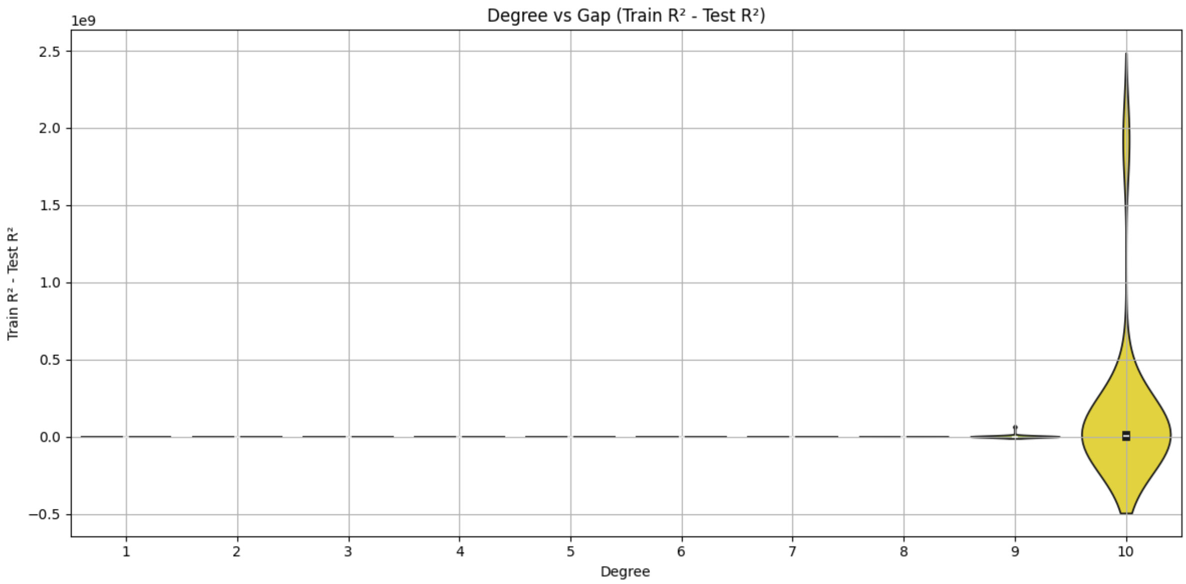
```
plt.figure(figsize=(12, 6))
sns.violinplot(
    x='Degree',
    y='Train R² - Test R²',
    data=violin_data_gap,
    hue='Degree',
    palette="viridis",
    legend=False
)
plt.title('Degree vs Gap (Train R² - Test R²)')
plt.grid(True)
plt.tight_layout()
plt.savefig("violin_gap_error.png")
plt.show()

# ----------------------------
# 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
```

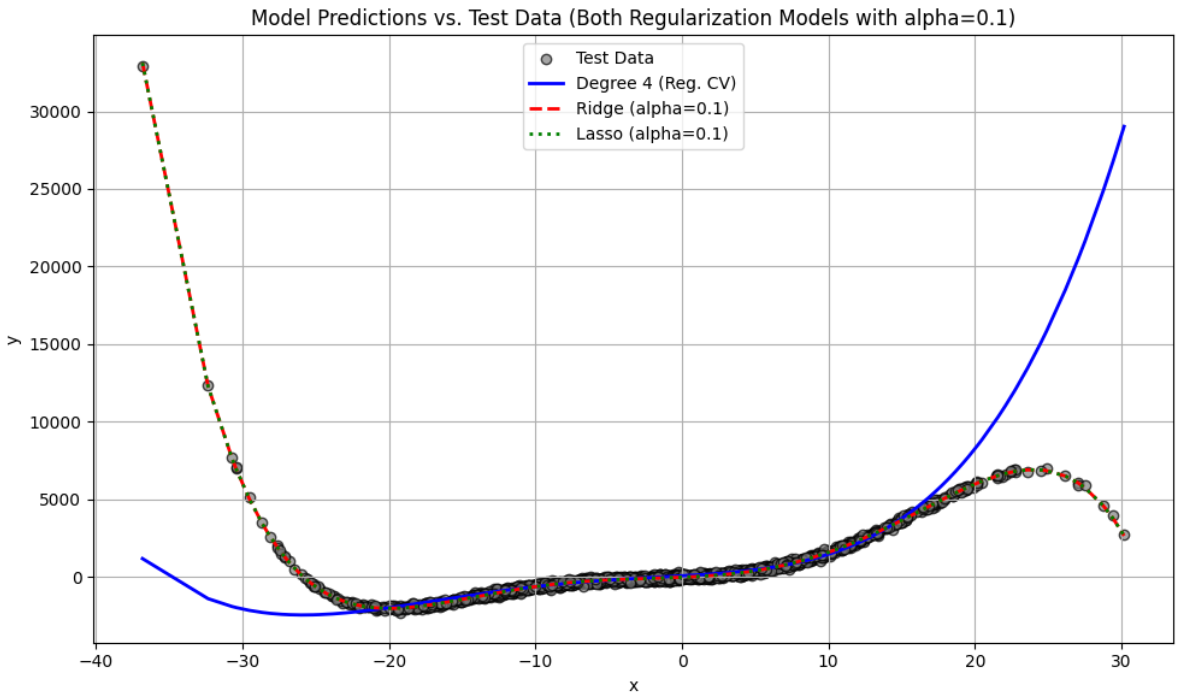
```
        best_degree = degree

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))
```

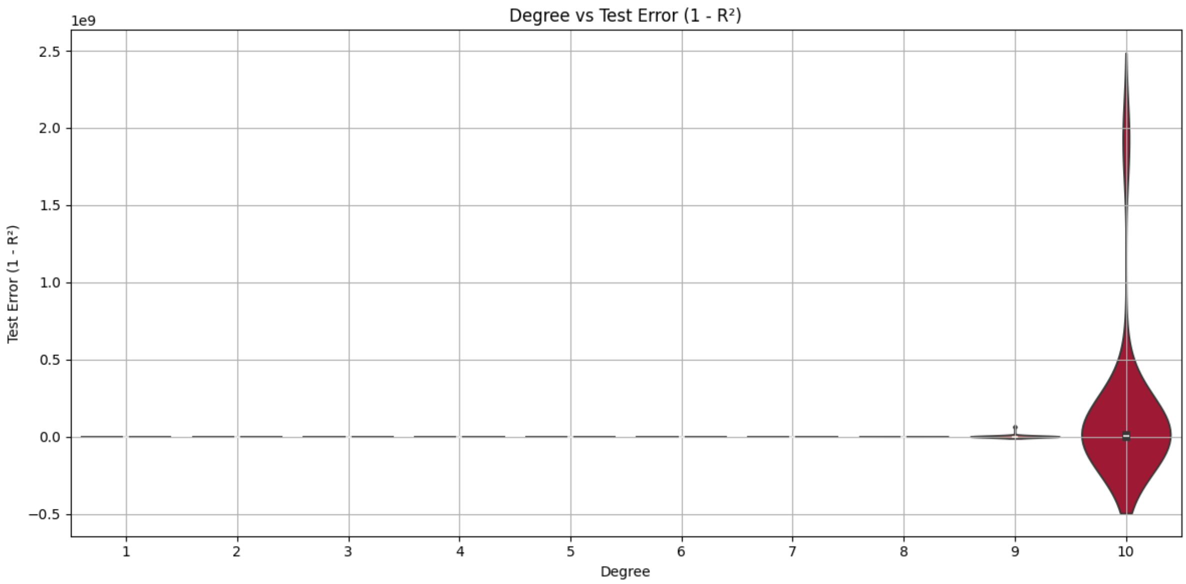
```
print("\n10-Fold CV for Regularized Models (using polynomial degree
= 5) with alpha=0.1 for both:")
print("    Ridge CV Scores: ", ridge_scores)
print("    Ridge Mean CV R²: {:.4f}".format(ridge_scores.mean()))
print("    Ridge Test R² (alpha=0.1): {:.4f}".format(ridge_r2_test))
print("")
print("    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)
```


Degree vs Gap (Train R² - Test R²)
1e9
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Train R² - Test R²
1
2
3
4
5
6
7
8
9
10
Degree


Model Predictions vs. Test Data (Both Regularization Models with alpha=0.1)
Test Data
Degree 4 (Reg. CV)
Ridge (alpha=0.1)
Lasso (alpha=0.1)
30000
25000
20000
15000
10000
5000
0
y
−40
−30
−20
−10
0
10
20
30
x


Degree vs Test Error (1 - R²)
1e9
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Test Error (1 - R²)
1
2
3
4
5
6
7
8
9
10
Degree