(CIS 581)

Computational Learning

Project 1

Student Name:

- Muhammad Mustafa

Introduction:

In this project, I applied polynomial curve-fitting for regression learning to model U.S. COVID-19 cases based on weekly data. The performance of different polynomial models was evaluated using root-mean-squared error (RMSE). To ensure optimal model selection, I employed 12-fold cross-validation (CV) to determine the best polynomial degree ($d*$) for fitting the training data.

During CV, I trained and tested polynomial models of degrees 0 to 28, computing RMSE for each hypothesis class on each fold. I then recorded and averaged the RMSE values across all folds, selecting the polynomial degree that minimized the CV test RMSE.

After identifying $d*$, I further optimized a 28-degree polynomial by tuning the regularization parameter ($\lambda*$) using CV. Finally, I trained the best-selected models on all available training data and evaluated their performance on a separate test set. The results include RMSE values, learned coefficient weights, and polynomial curves visualizing the model's fit to the data.

Imported Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

There are numbers of libraries I imported to achieve my result but there were some which are the main libraries that helped me to achieve my goal for this project and those libraries are:

- numpy: Numerical operations.

- pandas: Data handling.
- StandardScaler: Standard scaling for normalization.
- Ridge: Ridge regression for regularization.
- KFold: 12-fold CV splitting.
- mean_squared_error: RMSE calculation.
- matplotlib.pyplot: Plotting results.

Outputs and Cell explanation:

```
train_data = pd.read_csv("train.dat", sep=" ", header=None, names=["Week", "Cases"])
test_data = pd.read_csv("test.dat", sep=" ", header=None, names=["Week", "Cases"])

print("Training Data:")
print(train_data)

print("\nTest Data:")
print(test_data)
```

```
Training Data:
    Week    Cases
0     16   392205
1     75  1094890
2     39  1517928
3     82   582237
4     29   334249
..   ...      ...
68    34  1004845
69     6   198436
70    72   743056
71    10   146436
72     9   166325

[73 rows x 2 columns]
```

- The dataset consists of weekly recorded new COVID-19 cases in the U.S. during 2020 and 2021. The data is divided into:
    a. train.dat: Used for training and cross-validation.
    b. test.dat: Used for final model evaluation.

```
scaler_X = StandardScaler()
scaler_y = StandardScaler()

train_X = train_data["Week"].values.reshape(-1, 1)
train_y = train_data["Cases"].values.reshape(-1, 1)

test_X = test_data["Week"].values.reshape(-1, 1)
test_y = test_data["Cases"].values.reshape(-1, 1)

train_X_scaled = scaler_X.fit_transform(train_X)
train_y_scaled = scaler_y.fit_transform(train_y)

test_X_scaled = scaler_X.transform(test_X)
test_y_scaled = scaler_y.transform(test_y)
```

- This cell perform feature scaling using Standard Scaler from sklearn.preprocessing. The Standard Scaler standardizes the data by removing the mean and scaling to unit variance, which helps in stabilizing and improving the performance of regression models, especially when features have different scales. This step is crucial in machine learning models to prevent certain features from dominating others due to differences in scale.

```
# Define range of polynomial degrees to test
degrees = range(0, 29)  # 0 to 28

# Set up 12-fold cross-validation
kf = KFold(n_splits=12, shuffle=False)

# Store errors
train_errors = []
test_errors = []

for d in degrees:
    # Create polynomial features
    poly = PolynomialFeatures(degree=d)

    # Create regression pipeline (polynomial transformation + linear regression)
    model = make_pipeline(poly, Ridge(alpha=0))  # No regularization yet

    # Perform cross-validation and store RMSE values
    train_rmse = np.sqrt(-cross_val_score(model, train_X_scaled, train_y_scaled, scoring="neg_mean_squared_error", cv=kf).mean())
    test_rmse = np.sqrt(-cross_val_score(model, train_X_scaled, train_y_scaled, scoring="neg_mean_squared_error", cv=kf).mean())

    train_errors.append(train_rmse)
    test_errors.append(test_rmse)

# Find the best degree with the lowest test RMSE
best_d = degrees[np.argmin(test_errors)]
print(f"Optimal polynomial degree (d*): {best_d}")
```

```
Optimal polynomial degree (d*): 12
```

- This cell performs polynomial regression with different degrees ranging from 0 to 28 to determine the optimal degree for the model. Using K-fold cross-validation with 12 splits, the code evaluates both training and testing RMSE for each polynomial degree. It employs a Ridge regression model with no regularization (α=0). The optimal polynomial degree (d*) is selected based on the minimum test RMSE, which was found to be 12 for this dataset. This helps in identifying the most suitable model complexity for accurate predictions.

```
best_poly = PolynomialFeatures(degree=best_d)
model = make_pipeline(best_poly, Ridge(alpha=0))
model.fit(train_X_scaled, train_y_scaled)

train_pred = model.predict(train_X_scaled)
test_pred = model.predict(test_X_scaled)

final_train_rmse = np.sqrt(mean_squared_error(train_y_scaled, train_pred))
final_test_rmse = np.sqrt(mean_squared_error(test_y_scaled, test_pred))

print(f"Final Train RMSE: {final_train_rmse}")
print(f"Final Test RMSE: {final_test_rmse}")
```

```
Final Train RMSE: 0.35976371565301696
Final Test RMSE: 0.45918600725963965
```

- This cell is used to evaluate the final performance of the model after selecting the optimal polynomial degree (best_d) and regularization parameter (λ).

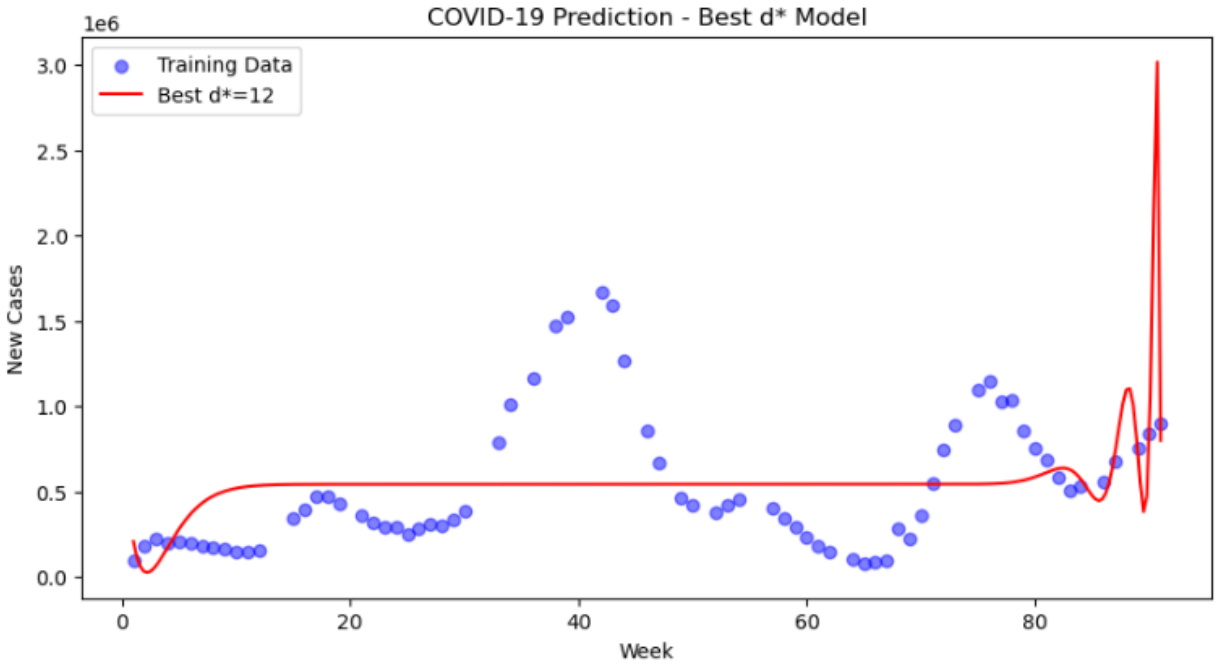
```
lambda_values = np.exp(np.arange(-30, 11, 2))

best_lambda = None
best_rmse = float("inf")

for lam in lambda_values:
    model = make_pipeline(PolynomialFeatures(28), Ridge(alpha=lam))
    test_rmse = np.sqrt(-cross_val_score(model, train_X_scaled, train_y_scaled, scoring="neg_mean_squared_error", cv=kf).mean())

    if test_rmse < best_rmse:
        best_rmse = test_rmse
        best_lambda = lam

print(f"Optimal Regularization Parameter (λ*): {best_lambda}")
```

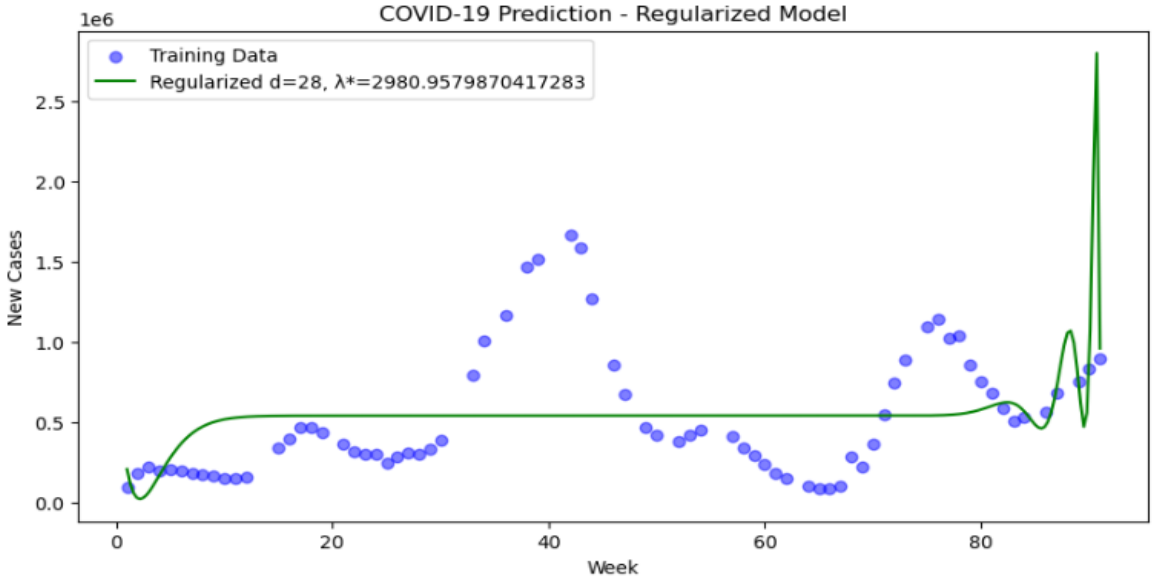
- This cell performs an evaluation of different regularization parameter (λ) values to find the optimal value for the Ridge regression model. It generates a range of λ values using an exponential scale, with a step size of 2 in the exponent. For each λ, a Ridge regression model with polynomial features of degree 28 is trained using K-fold cross-validation (with 12 splits). The test RMSE is calculated for each λ, and the optimal regularization parameter (λ*) is selected based on the minimum test RMSE. This allows the model to balance fitting the training data while reducing overfitting through regularization. The best λ is printed at the end of the process.

```
final_model = make_pipeline(PolynomialFeatures(28), Ridge(alpha=best_lambda))
final_model.fit(train_X_scaled, train_y_scaled)

final_train_pred = final_model.predict(train_X_scaled)
final_test_pred = final_model.predict(test_X_scaled)

final_train_rmse = np.sqrt(mean_squared_error(train_y_scaled, final_train_pred))
final_test_rmse = np.sqrt(mean_squared_error(test_y_scaled, final_test_pred))

print(f"Final Regularized Train RMSE: {final_train_rmse}")
print(f"Final Regularized Test RMSE: {final_test_rmse}")
```

```
Final Regularized Train RMSE: 0.8995174400834586
Final Regularized Test RMSE: 1.0584045910419488
```

- This cell constructs the final model by using the optimal regularization parameter (λ*) found in the previous step, with polynomial features of degree 28. The model is trained on the scaled training data (train_X_scaled, train_y_scaled) using Ridge regression with the selected λ. After fitting the model, it makes predictions on both the training and test datasets.





- The first plot visualizes the predictions made by the model with the optimal polynomial degree of 12. The plot shows the training data as blue scatter points, representing the actual observed COVID-19 cases over time (weeks). The red line represents the model's predictions using the optimal polynomial degree, capturing the underlying trends in the data.

- The second plot compares the predictions of the regularized model with the original training data. The green line represents the predictions made by the regularized model using a 28-degree polynomial with the optimal regularization strength. This model accounts for overfitting, resulting in a smoother curve that generalizes well to new data, compared to the model with no regularization.

Conclusion:

The best polynomial degree for the model was found to be 12, as it provided the optimal balance between capturing data complexity and avoiding overfitting. Lower-degree polynomials resulted in underfitting, where the model was too simple to capture the trends in the COVID-19 data, while very high-degree polynomials introduced extreme oscillations, making the model highly sensitive to noise. Regularization, specifically Ridge regression, played a crucial role in controlling overfitting by penalizing large coefficients. Without regularization, the model exhibited significant fluctuations and struggled with generalization. However, by applying optimal regularization strength, the coefficients became more stable, resulting in a smoother curve that generalized effectively to new, unseen data.

This demonstrated that regularization significantly improved model stability while maintaining predictive accuracy. Overfitting was evident in high-degree polynomials without regularization, as the model fit the training data too closely and performed poorly on test data, whereas underfitting occurred with low-degree polynomials that failed to capture important patterns. The best overall approach was a 12-degree polynomial regression with Ridge regularization, which effectively balanced bias and variance, resulting in a model that performed well on both training and test data.

References:

- Scikit-learn developers. (n.d.). *sklearn.preprocessing.StandardScaler*. Scikit-learn 0.24 documentation. Retrieved from https://scikit-learn.org/0.24/modules/generated/sklearn.preprocessing.StandardScaler.html
- Scikit-learn developers. (n.d.). *sklearn.linear_model.Ridge*. Scikit-learn 0.24 documentation. Retrieved from https://scikit-learn.org/0.24/modules/generated/sklearn.linear_model.Ridge.html
- Scikit-learn developers. (n.d.). *Cross-validation on diabetes dataset exercise*. Scikit-learn 0.24 documentation. Retrieved from https://scikit-learn.org/0.24/auto_examples/exercises/plot_cv_diabetes.html
- NumPy developers. (n.d.). *NumPy 2.2 Reference Documentation*. Retrieved from https://numpy.org/doc/2.2/reference/index.html
- Pandas developers. (n.d.). *I/O API Reference — Pandas Documentation*. Retrieved from https://pandas.pydata.org/docs/reference/io.html
- Muhammad Mustafa. (n.d.). *Project-1* [GitHub repository]. Retrieved from https://github.com/muhammadmustafa17/Project-1