CYBR 486 - Lab #4 – Polynomial Regression, Ridge and Lasso Regularization

This lab we will making, training and evaluating some more regression models. It will be pretty similar in some ways to lab 3, using the same housing dataset and starting with a linear regression model from lab 3. You will be performing 5 primary tasks for this lab:

1) Splitting the data into 80% training and 20% testing sets

2) Use the training dataset to build the linear regression model from the previous lab

3) Use the training dataset to build a polynomial regression model for the given degree

4) Use the training dataset to build the Ridge regression model.

5) Compare the performance of the models using RMSE and R2-score

Steps 1 and 2 can be copied from lab 3 if you wish, as training, predicting and evaluating the linear regression model is the same here as it was for lab 3.

The new parts for this lab involve transforming features to create a polynomial model, creating a lasso and ridge regression model, and then individually evaluating these 3 new models along with the previous linear regression model. The final task will be to record your observations on the performance scores of these models with respect to the housing dataset.

1.) Starting off, we have some new imports for some new handy tools for the tasks at hand. You can typically copy any imports from previous labs as many of them will frequently be reused, and then just remove the ones that we ended up not using for any particular lab when finished.

```
from sklearn.preprocessing import PolynomialFeatures
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
from sklearn.linear_model import LinearRegression
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
from sklearn.linear_model import Ridge, Lasso
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
from sklearn.linear_model import LassoCV
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html
from sklearn.linear_model import RidgeCV
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
```

I've added a small change to how I was doing documentation links before. I'm going to link the documentation references along with the import statements from now on, unless my fickleness wins out and I change my mind again.

2a.) Read the housing dataset into memory and split it into 80% train, 20% test. Attach a screenshot to this document or directly to canvas of your code for this

```
C:\Users\Bhavy\PycharmProjects\Lab33\venv\Scripts\python.exe C:/Users/Bhavy/PycharmProjects/Lab3374a.py
Training data shape (X): (404, 13)
Testing data shape (X): (102, 13)
Training data shape (y): (404,)
Testing data shape (y): (102,)

Process finished with exit code 0
```

2b.) Recreate the linear regression model you did from lab3. Train it on the training set and generate predictions from the test set. Also calculate the Root Mean Square Error and R2 score of the linear regression model Attach a screenshot to this document or directly to canvas of your code for this.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display shapes of the resulting datasets
print("Training data shape (X):", X_train.shape)
print("Testing data shape (X):", X_test.shape)
print("Training data shape (y):", y_train.shape)
print("Testing data shape (y):", y_test.shape)

# Step 2b: Recreate Linear Regression model, train, predict, and evaluate

# Create a linear regression model
linear_model = LinearRegression()

# Train the linear regression model on the training data
linear_model.fit(X_train, y_train)

# Generate predictions on the test data
y_pred = linear_model.predict(X_test)

# Calculate Root Mean Square Error (RMSE)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

#Calculate R2 score
r2 = r2_score(y_test, y_pred)

# Print the evaluation results
print("Linear Regression RMSE:", rmse)
print("Linear Regression R2 Score:", r2)
```

```
C:\Users\Bhavy\PycharmProjects\lab33\venv\Scripts\python.exe C:/Users/Bhavy/PycharmProjects/lab33/4a.py
Training data shape (X): (404, 13)
Testing data shape (X): (102, 13)
Training data shape (y): (404,)
Testing data shape (y): (102,)
Linear Regression RMSE: 4.928602182665359
Linear Regression R2 Score: 0.6687594935356289

Process finished with exit code 0
```

3.) Polynomial model

Starting the new stuff, we need to create a polynomial model so we can see how it compares to the other ones at the end. When doing a polynomial model, we actually need to create 2 objects, one is a PolynomialFeatures() object which has methods that allow us to take our typical X variable for linear regression and transform it into polynomial style, but we also need a LinearRegression() object to generate predictions. I recommend making a new Linear Regression object for this part as re-using the previous model when it was already fitted can cause issues.

```python
poly_linear_model = LinearRegression()
degrees = 2
```

```
poly_model_2 = PolynomialFeatures(degree=degrees)
```

Pay attention to the degrees variable which sets how many dimensions the data is transformed for. You will be asked to change this variable and make some observations for different degrees.

With the objects created, we need to create new variables and have the PolynomialFeatures object transform the feature data for us.

```
poly_X_train = poly_model_2.fit_transform(X_train)
poly_X_test = poly_model_2.transform(X_test)
```

We call fit_transform() for the training set. This is because we need to transform the features to polynomial form, but when we similarly transform the test set, we want to do so with the fitted poly model. Fit_transform performs both steps in one method call, then we can transform the test set into a new polynomial variable as well.

Next, we need to use our newly created polynomial data variables, to fit the linear regression model and generate our predictions.

```
poly_linear_model.fit(poly_X_train, y_train)
poly_predictions = poly_linear_model.predict(poly_X_test)
```

For those looking for a bit of added explanation, we transformed our X_train (currently to the degree of 2) to try to find more complex relationships between the features, but the y-values for any given sample is still the same, so we fit the model with our poly_X, and the original y sets for training. Similarly, we make our predictions based on the new poly_X_test, but when we evaluate, we'll compare this to the old y_test as again, the y values aren't being changed.

Now we need to evaluate the polynomial model. We need this to compare scores when changing the polynomial degree variable.

Calculate the Root Mean Square Error and R2 score of the polynomial model with a degree of 2. For comparison you also need to compute the scores on the training set. The goal is to see how the scores compare for our predictions, versus the scores on the training data as the degree increases (NOTE: this relationship won't always be identical but will be distinctive for this dataset and a polynomial model). HINT: to get the scores on the training set you'll be comparing the actual original y-values to the polynomial predictions. Compute the scores at degree 2, then increase degree to 4, and finally 6. Take a

screenshot of your code for this and also include your observations as to what is happening here as you increase the dimensions.

```
# Function to train polynomial regression models for different degrees
def polynomial_regression(degree):
    # Create polynomial features
    poly_model = PolynomialFeatures(degree=degree)

    # Transform the training and test sets into polynomial features
    poly_X_train = poly_model.fit_transform(X_train)
    poly_X_test = poly_model.transform(X_test)

    # Create a new linear regression model for the polynomial features
    poly_linear_model = LinearRegression()

    # Train the model on the polynomial features of the training set
    poly_linear_model.fit(poly_X_train, y_train)

    # Generate predictions for the test set
    poly_predictions = poly_linear_model.predict(poly_X_test)

    # Calculate RMSE and R2 score
    poly_rmse = np.sqrt(mean_squared_error(y_test, poly_predictions))
    poly_r2 = r2_score(y_test, poly_predictions)

    # Print the results
    print(f"Polynomial Regression (degree {degree}) RMSE:", poly_rmse)
    print(f"Polynomial Regression (degree {degree}) R2 Score:", poly_r2)


# Train and evaluate polynomial regression models for degrees 2, 4, and 6
for degree in [2, 4, 6]:
    polynomial_regression(degree)
```

As the degree of the polynomial regression model increases, the model captures more complex relationships in the data, improving performance up to a certain point. At degree 4, the model shows the best results with a lower RMSE and higher R^2 score. However, further increasing the degree to 6 leads to overfitting, where the model becomes too complex, starts capturing noise, and performs slightly worse on the test data. Therefore, while increasing the degree can improve accuracy, it also risks overfitting if taken too far.

4a.) Ridge regression

As a reminder from the lectures, both Ridge and Lasso are techniques to “regularize” a polynomial model. The idea is to try to avoid weight parameters getting overinflated. There are 2 ways to approach this, you can use the standard Ridge and Lasso libraries (documentation provided along with the import statements), or you can use the cross-validation libraries instead. You can play around with Lasso if you like, but for this lab you’ll only be required to create and evaluate the ridge model

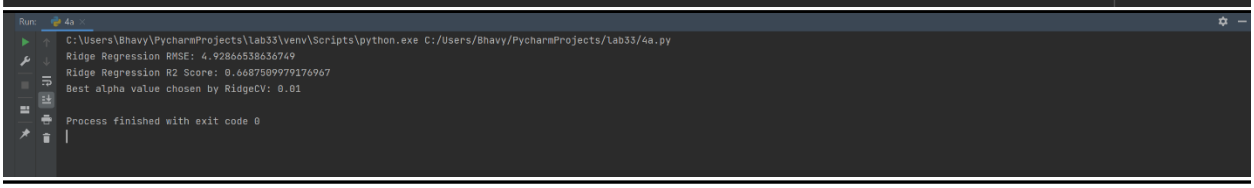
To refresh our memory about regularization, we’re adding a new value “alpha” to serve as the regularization parameter. When using Ridge or Lasso, this “alpha” value must be supplied by us, but there’s a catch. We want to use the best possible value for alpha for our model and dataset, but without years of training in data analysis, trial and error is typically the only way to go about this. We would need to try a value for alpha, train and test the model, then repeat evaluating the score each time to pick the best option. Let’s not do that, let’s make things easier.

```
alpha_v = [0.001, 0.01, 0.1, 1, 10]
ridge_cv_model = RidgeCV(alphas=alpha_v)
ridge_cv_model.fit(X_train, y_train)
ridge_predictions = ridge_cv_model.predict(X_test)
```

Notice I’m calling RidgeCV, and not Ridge. This library leverages a technique called cross-validation which, in short, will automatically run through multiple values for alpha (supplied by us as an “array-like” object) and settle on the best of the options provided. In this case 5 possible values for alpha are provided and when you fit the model, the best option is selected. You can manually determine scoring methods as to how the “best” option is selected but the default will work for this lab. There are a ton of internal settings you can play with in RidgeCV as well as LassoCV if you want to dive deeper into them.

Calculate Root Mean Square Error and R2 score for Ridge Regression. Take a screenshot of your code for the Ridge Model and the score calculation

```python
# Import necessary libraries
from sklearn.linear_model import RidgeCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define the possible values for alpha
alpha_v = [0.001, 0.01, 0.1, 1, 10]

# Create RidgeCV model with cross-validation to select the best alpha
ridge_cv_model = RidgeCV(alphas=alpha_v)

# Train the model on the training data
ridge_cv_model.fit(X_train, y_train)

# Generate predictions on the test data
ridge_predictions = ridge_cv_model.predict(X_test)

# Calculate Root Mean Square Error (RMSE)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_predictions))

# Calculate R2 score
ridge_r2 = r2_score(y_test, ridge_predictions)

# Print the evaluation results
print("Ridge Regression RMSE:", ridge_rmse)
print("Ridge Regression R2 Score:", ridge_r2)

# Print the best alpha value chosen by RidgeCV
print("Best alpha value chosen by RidgeCV:", ridge_cv_model.alpha_)
```



5.) Record your observations comparing Linear, Polynomial, and Ridge model scores. NOTE: you will likely need to run your code multiple times and observe across all runs. As there are some values that are randomly being selected like test/train samples so the scores will vary. I just want your general observations given what you know from the lectures so far on the scores.

- Linear **Regression** is good for capturing simple relationships but struggles with more complex patterns.
- Polynomial **Regression** improves accuracy by modeling more complex relationships but risks overfitting at higher degrees.
- Ridge **Regression** effectively regularizes the model, reducing the chances of overfitting while maintaining strong performance, making it a robust choice for balancing complexity and generalization.