Course Name

Neural Networks and Deep Learning (COMP-8610)

Document Type

Assignment 1

Professor

Dr. Alioune Ngom

Team - Members	Student ID
Manjinder Singh	110097177
Harbhajan Singh	110100089
Navjot Makkar	110100116

Table of Contents

Question #1: In this question you are to create some simulated data sets and then use Ordinary Least Square Regression to perform some prediction. Use whatever programming language you want to use.

Task 1: Generate 5000 synthetic data points (x, y) as follows:

- **Using the rnorm() function in R (or equivalent in Matlab or Python or etc), create a vector, x, containing 5000 observations drawn from a Gaussian distribution N(0, 1) [ie, a normal distribution with mean 0 and variance 1]. This vector x represents your set of inputs x.**
- **Using the rnorm() function in R (or equivalent in Matlab or Python or etc), create a vector, eps, containing 5000 observation drawn from a N(0, 0.25) distribution; ie, a normal distribution with mean 0 and variance 0.25.**
- **Using vectors x and eps, generate a vector y according to the following model-**

  $y = -1 + 0.5x - 2x2 + 0.3x3 + eps.$

Solution: By utilizing NumPy's random functions, the code generates synthetic data for x, eps, and y, representing a polynomial relationship with added noise-

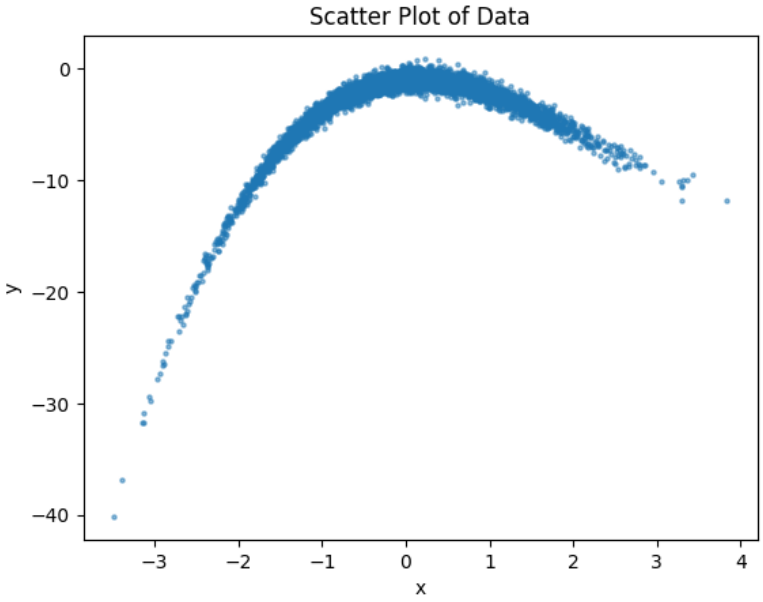
- The np.random.normal() function is used to generate random samples from a normal distribution.
- The x vector is created with 5000 random values drawn from a normal distribution with mean 0 and standard deviation 1.
- The eps vector is created with 5000 random values drawn from a normal distribution with mean 0 and standard deviation 0.25.
- The y vector is constructed using a polynomial equation involving the x values, along with the quadratic and cubic terms. The coefficients for each term are specified (-1, 0.5, -2, and 0.3), and the eps values are added as noise to the polynomial equation.

Code:

```
# Create vector x
x = np.random.normal(0, 1, size=5000)

# Create vector eps
eps = np.random.normal(0, np.sqrt(0.25), size=5000)

# Create vector y
y = -1 + 0.5*x - 2*x**2 + 0.3*x**3 + eps

#Reshape x
x = x.reshape(-1,1)
```

```
# Scatter plot of the data
plt.scatter(x, y, s=5, alpha=0.5, label='Data Points')

# Set plot labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter Plot of Data')

# Display the plot
plt.show()
```

Output:



Task 2: Implement Ordinary Least Squares method (for linear regression) given the dataset you have created above, for predicting the value of any given input x. Use the whole data set as your training set; no test set. Perform cross-validation (LOOCV or 10-fold-cv) of your choice but without a test set.

Solution:

a. Applying linear regression on all data without cross validation

1. Created a Linear Regression object using scikit-learn's LinearRegression class.
2. Fit the linear regression model on the training data using the fit() method from scikit-learn.
3. Predicted the target variable using the predict() method from scikit-learn.
4. Calculated the mean squared error (MSE) using the mean_squared_error() function from scikit-learn.metrics.
5. Printed the calculated mean squared error.
6. Retrieved the intercept and weight (coefficient) of the linear regression model using the intercept_ and coef_ attributes provided by the scikit-learn LinearRegression object.
7. Printed the retrieved intercept and weight.

Code:

```
# Create Linear Regression object
lr = LinearRegression()

# Fit the model on the training data
lr.fit(x, y)

# Get the predicted target variable
y_pred = lr.predict(x)

# Calculate mean squared error
mse = mean_squared_error(y, y_pred)

print("Mean Squared Error:", mse)

# Get the intercept and weight
intercept = lr.intercept_
weight = lr.coef_

print("Coefficient of x (weight):", weight)
print("Intercept:", intercept)
```

Output:

```
Mean Squared Error: 8.844749521787975
Coefficient of x (weight): [1.43031179]
Intercept: -3.0020452617736098
```

b. Applying linear regression on all data with cross validation

1. Created a Linear Regression object using scikit-learn's LinearRegression class.
2. Applied k-fold cross-validation with 10 folds using the cross_val_score function from scikit-learn.
3. Calculated the mean squared errors for each fold.
4. Converted the negative mean squared errors to positive values.
5. Printed the mean squared errors and the average mean squared error.
6. Fitted the linear regression model on the training data using the fit() method.
7. Retrieved the intercept and weight (coefficient) of the linear regression model.
8. Printed the retrieved intercept and weight.

Code:

```
# Create Linear Regression object
lr = LinearRegression()

# Apply k-fold cross-validation with 10 folds
scores = cross_val_score(lr, x, y, scoring='neg_mean_squared_error', cv=10)

# Convert the negative mean squared errors to positive
mse_scores = -scores

print("Mean Squared Error:", mse_scores)
print("Mean Squared Error (Average):", mse_scores.mean())

# Fit the model on the training data
lr.fit(x, y)

# Get the intercept and weight
intercept = lr.intercept_
weight = lr.coef_[0]

print("Coefficient of x (weight):", weight)
print("Intercept:", intercept)
```

Output:

```
Mean Squared Error: [ 7.49934424  9.57666906  6.76476233 10.79176893  6.78582239  9.11170957
  9.15761867 10.70371697  7.71263772 10.4956839 ]
Mean Squared Error (Average): 8.859973376309298
Coefficient of x (weight): 1.4303117929763975
Intercept: -3.0020452617736098
```

Task 3: Repeat the above with cross-validation method of your own choice (LOOCV or 10-fold-cv) to find the best polynomial degree d which yield best accuracy. You must first randomly create a test set of size between 20% and 30% drawn from your original full data set. If this is correctly done, your methods should not only find d = 3 to be the best degree, but they should also find the best weight vector, wbest, to be as close as possible to wtrue = (-1, +0.5, -2, +0.3). Compare your results of OLS with or without cross-validation scheme.

Solution:

a. Splitting data into training set (75%) and test set (25%)
   1. The data is split into training and testing sets using the train_test_split function from scikit-learn.
   2. The input features (x) and target variable (y) are divided into x_train, x_test, y_train, and y_test sets.
   3. The split is performed with a test size of 0.25 (25% of the data) and a random state of 42.

   Code:

```
# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
print("Number of records in training set: ", x_train.shape[0])
print("Number of records in training set: ", x_test.shape[0])
```

   Output:

```
Number of records in training set:  3750
Number of records in training set:  1250
```

b. Applying linear regression without cross-validation on training set and calculating MSE on test set.

1. Defined a range of polynomial degrees from 1 to 10 to test different degrees of polynomial features.
2. Initialized variables to store the best degree and the best mean squared error.
3. Performed 10-fold cross-validation for each degree of the polynomial.
4. Created a pipeline that includes polynomial feature transformation and linear regression.
5. Set the degree of polynomial features in the pipeline for each iteration.
6. Fitted the pipeline model on the training data (x_train and y_train).
7. Predicted the target variable (y_pred) for the test data (x_test) using the fitted pipeline model.
8. Calculated the mean squared error (MSE) between the predicted and actual target values.
9. Appended the MSE scores to the cv_scores list and the min score represents the optimal degree value for the data.

Code:

```
# Define a range of polynomial degrees to test
degrees = range(1, 11)

# Initialize variables to store the best degree and best mean squared error
best_degree = None
best_mse = float('inf')

scores = []
for degree in degrees:
    # Create pipeline for polynomial features and linear regression
    pipe = Pipeline([
        ('poly', PolynomialFeatures()),
        ('lr', LinearRegression())
    ])
    pipe.set_params(poly__degree=degree)

    # Fit the model on the training data
    pipe.fit(x_train, y_train)

    # Predict the target variable for the test data
    y_pred = pipe.predict(x_test)

    # Calculate the mean squared error
    mse = mean_squared_error(y_test, y_pred)

    # Append the scores
    scores.append(mse)

# Find the degree with the minimum MSE
best_degree = degrees[np.argmin(scores)]

print("Mean Squared Errors For All Iterations:", scores)
print("Best MSE:", scores[best_degree-1])
print("Best degree of polynomial:", best_degree)
```

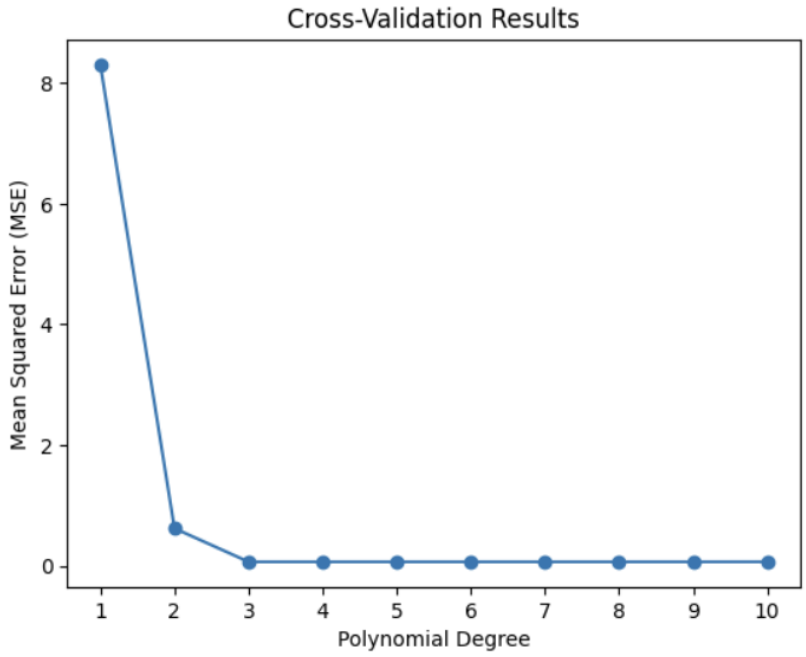
```
# Fit the pipeline on the training data, using the best degree of the polynomial
pipe.set_params(poly__degree=best_degree)
pipe.fit(x_train, y_train)
# Extract the coefficients of the linear regression model
coefs = pipe.named_steps['lr'].coef_
true_coefs = [-1, 0.5, -2, 0.3]
# Compare the coefficients to the true values
print("Coefficient of x^0 i.e. w0: : {} (true value: {})".format(pipe.named_steps['lr'].intercept_, true_coefs[0]))
print("Coefficient of x^1 i.e. w1: : {} (true value: {})".format(coefs[1], true_coefs[1]))
print("Coefficient of x^2 i.e. w2: : {} (true value: {})".format(coefs[2], true_coefs[2]))
print("Coefficient of x^3 i.e. w3: : {} (true value: {})".format(coefs[3], true_coefs[3]))
```

```
#Plotting
plt.plot(degrees, cv_scores, marker='o')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Cross-Validation Results')
plt.xticks(degrees)
plt.show()
```

Output:

```
Mean Squared Errors For All Iterations: [7.722892755025049, 0.642745952286897, 0.2613752927918514, 0.26141366800792487, 0.26148
61933823593, 0.26143617772905037, 0.26158073939001725, 0.2617328697520028, 0.2617084353291443, 0.2615002866927923]
Best MSE: 0.2613752927918514
Best degree of polynomial: 3
```

```
Coefficient of x^0 i.e. w0: : -0.9850529470031639 (true value: -1)
Coefficient of x^1 i.e. w1: : 0.4950562048456799 (true value: 0.5)
Coefficient of x^2 i.e. w2: : -1.997883720489212 (true value: -2)
Coefficient of x^3 i.e. w3: : 0.2990448779967233 (true value: 0.3)
```



c. Applying linear regression with cross-validation on training set and calculating MSE on test set.

1. Defined a range of polynomial degrees from 1 to 10 to test different degrees of polynomial features.
2. Performed 10-fold cross-validation for each degree of the polynomial.

3. Created a pipeline using the Pipeline class from scikit-learn, which includes PolynomialFeatures and LinearRegression.
4. Set the degree of polynomial features in the pipeline for each iteration using the set_params method.
5. Applied cross-validation on the pipeline model using the cross_val_score function from scikit-learn.
6. Used 'neg_mean_squared_error' as the scoring metric to evaluate the model's performance.
7. Calculated the average mean squared error (MSE) score for each degree by taking the mean of the negative MSE scores.
8. Stored the average MSE scores in the cv_scores list.
9. Found the degree with the lowest average MSE score using the argmin function from numpy.

Code:

```
# Define a range of polynomial degrees to test
degrees = range(1, 11)

# Perform 10-fold cross-validation for each degree of the polynomial
cv_scores = []
for degree in degrees:
    # Create pipeline for polynomial features and linear regression
    pipe = Pipeline([
        ('poly', PolynomialFeatures()),
        ('lr', LinearRegression())
    ])
    pipe.set_params(poly__degree=degree)
    pipe.set_params(poly__degree=degree)
    scores = cross_val_score(pipe, x_train, y_train, cv=10, scoring='neg_mean_squared_error')
    cv_scores.append(np.mean(-scores))

# Find the degree with the highest mean R^2 score
best_degree = degrees[np.argmin(cv_scores)]
print("Best degree of polynomial:", best_degree)
print("Best MSE:", cv_scores[best_degree])
```

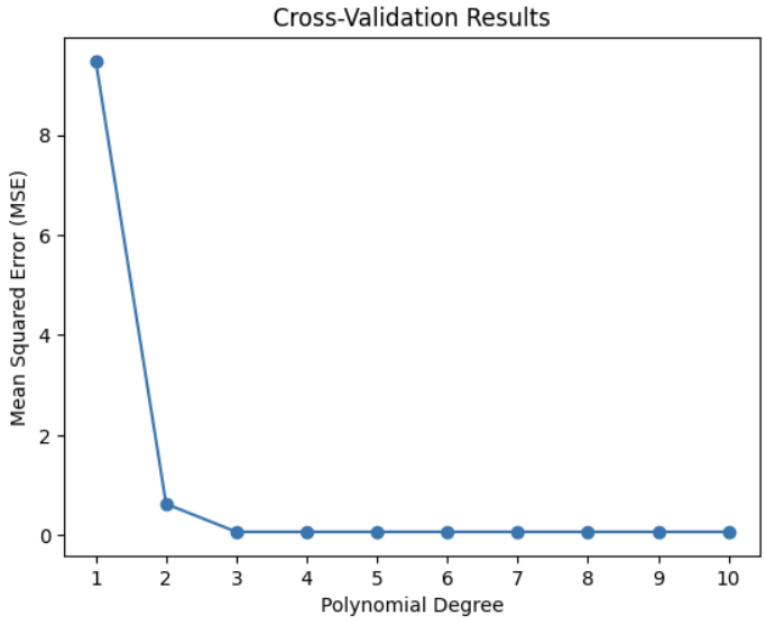
```
# Fit the pipeline on the training data, using the best degree of the polynomial
pipe.set_params(poly__degree=best_degree)
pipe.fit(x_train, y_train)
# Extract the coefficients of the linear regression model
coefs = pipe.named_steps['lr'].coef_
true_coefs = [-1, 0.5, -2, 0.3]
# Compare the coefficients to the true values
print("Coefficient of x^0 i.e. w0: : {} (true value: {})".format(pipe.named_steps['lr'].
                                                                 intercept_, true_coefs[0]))
print("Coefficient of x^1 i.e. w1: : {} (true value: {})".format(coefs[1], true_coefs[1]))
print("Coefficient of x^2 i.e. w2: : {} (true value: {})".format(coefs[2], true_coefs[2]))
print("Coefficient of x^3 i.e. w3: : {} (true value: {})".format(coefs[3], true_coefs[3]))
```

```
#Plotting
plt.plot(degrees, cv_scores, marker='o')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Cross-Validation Results')
plt.xticks(degrees)
plt.show()
```

Output:

```
Mean Squared Errors For All Iterations: [9.259740380845264, 0.8757286420179572, 0.24725856624277592, 0.24731743362959713, 0.247
43076741838993, 0.2475668940142021, 0.2480348563698626, 0.24859732675760662, 0.25727279412532034, 0.27113083049636466]
Best MSE: 0.24725856624277592
Best degree of polynomial: 3
```

```
Coefficient of x^0 i.e. w0: : -0.9850529470031639 (true value: -1)
Coefficient of x^1 i.e. w1: : 0.4950562048456799 (true value: 0.5)
Coefficient of x^2 i.e. w2: : -1.997883720489212 (true value: -2)
Coefficient of x^3 i.e. w3: : 0.2990448779967233 (true value: 0.3)
```



Summary

Experiment 1: Linear Regression on Entire Dataset without Cross Validation

When applying linear regression on the entire dataset without cross validation, we observed a high mean squared error (MSE) value of 8.8447. This result can be attributed to the attempt to fit a polynomial data to a linear model. The coefficient of x was found to be 1.4303, and the intercept was -3.0020. We can see that intercept value is still clos to true value of w0 but as it was linear model, it is not a great fit for our data.

Experiment 2: Linear Regression on Entire Dataset with Cross Validation

In this experiment, we applied linear regression on the entire dataset using k-fold cross validation with k=10. Interestingly, the results were similar to Experiment 1, with a mean MSE value of 8.8599, coefficient of x as 1.4303, and intercept as -3.0020. Since the model parameters were not changed, we obtained consistent results across the different folds of data.

Experiment 3: Linear Regression on Training Set without Cross Validation and MSE on Test Set

For this experiment, we split the data into a 75% training set and a 25% test set. We performed linear regression without cross validation on the training set, testing polynomial models of degrees 1 to 10. The optimal results were obtained when the degree was 3, with an

MSE of 0.2613. The weight vector for the optimal model was w0= -0.9850, w1= 0.4950, w2= -1.9978, and w3=0.2990. The low MSE value indicates a good fit between the model and the test data, demonstrating that our model closely approximates the real values in the polynomial function y = -1 + 0.5x – 2(x^2) + 0.3(x^3) where the expected weight values are w0=-1, w1=0.5, w2=2, and w3=0.3.

Experiment 4: Linear Regression on Training Set with Cross Validation and MSE on Test Set

Similar to Experiment 3, we used a 75% training set and a 25% test set. We performed linear regression on polynomial models of degrees 1 to 10 using k-fold cross validation with k=10. As expected, the optimal results were obtained at degree=3, with an MSE of 0.2472. The weight vector for the optimal model was the same as in Experiment 3: was w0= -0.9850, w1= 0.4950, w2= -1.9978, and w3=0.2990. Although the weight values remained consistent, the slight difference in the MSE value can be attributed to the averaging of the 10 MSE values obtained during the 10-fold cross validation process.

Conclusion

Overall, these experiments revealed that fitting the polynomial data to a linear model resulted in high MSE values. However, when utilizing a polynomial model of degree 3, the OLS linear regression approach demonstrated a strong fit to the test data, closely approximating the actual values of the polynomial function. The application of cross validation had a minimal impact on the weight vector but provided consistent MSE values.