Department of Electronics and Telecommunication Engineering

University of Moratuwa

EN4553 - Machine Vision

Assignment 03

Name	**Index**
H. D. M. Prematilaka	180497C

This report is submitted in partial fulfillment of the requirements for the module EN 4553 – Machine Vision.

Date of Submission: February 17, 2023

Link to the source code: Colab Notebook - 180497C

Note: Regression model without an intercept was used for the assignment. The data were not centered using the training data even though sklearn recommends to center data when the model doesn't have an intercept.

Question 01

Importing Libraries

[ ]:
```
import numpy as np
import os
from sklearn.linear_model import LinearRegression as linreg
import matplotlib.pyplot as plt
import pandas as pd
```

Q1). a).

Loading the Dataset

[ ]:
```
def load_dataset(src_dir: str):
    """
    Load the dataset as a set of numpy arrays

    Args:
    src_dir (str): Directory where the dataset files are stored

    Returns:
    (x_train, y_train, x_val, y_val, x_test) tuple where each array is one␣
 ↪dimensional

    """

    x_train = np.loadtxt(os.path.join(src_dir, 'x_train.txt'))
    y_train = np.loadtxt(os.path.join(src_dir, 'y_train.txt'))
    x_val = np.loadtxt(os.path.join(src_dir, 'x_val.txt'))
    y_val = np.loadtxt(os.path.join(src_dir, 'y_val.txt'))
    x_test = np.loadtxt(os.path.join(src_dir, 'x_test.txt'))

    return x_train, y_train, x_val, y_val, x_test
```

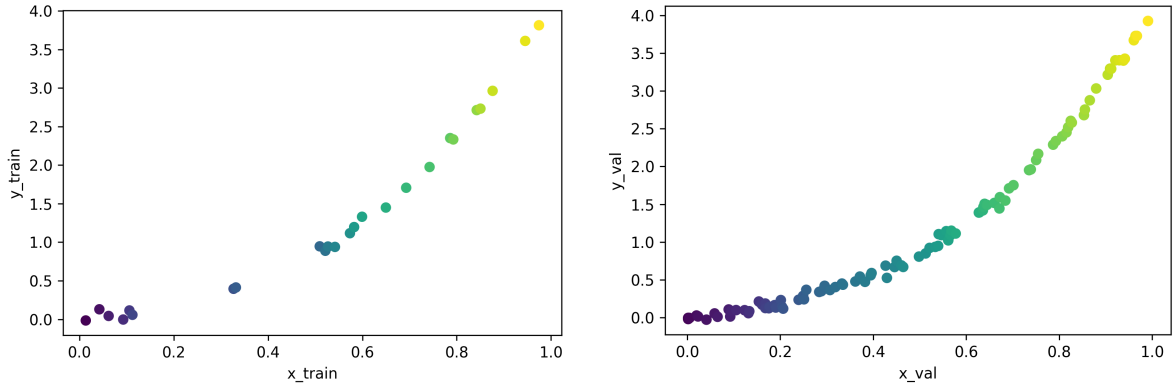

Figure 1: Plots for Train Data (Left) and Validation Data (Right)

Q1). b). i).

Getting Features

[ ]:
```
def get_features(x: np.ndarray, n: int):

    """
    Creates n-th degree polynomial features for the given vector x
    Example usage:
    get_features(np.array([1.0, 2.0, 3.0]), 3) outputs
    np.array([[ 1., 1., 1.],
    [ 2., 4., 8.],
    [ 3., 9., 27.]])

    Args:
    x: A numpy array of shape (num_examples, ) or (num_examples, 1)
    n: The degree of the polynomial features

    Returns:
    A matrix of shape (num_examples, n) where the j-th column is equal to
    the vector x raised, elementwise, to the power j

    """
    input_features = np.array([i**(j + 1)  for i in x for j in range(n)])
    input_features = np.reshape(input_features, (x.shape[0], n))
    return input_features
```

Q1). b). ii).

Fit and Evaluate

$MSE = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ where y is the actual value and $\hat{y}$ is the predicted value.

[ ]:
```
def fit_and_evaluate(x_train: np.ndarray, y_train: np.ndarray, x_val: np.
↪ndarray, y_val: np.ndarray, n: int):
    """
    Fits an n-th degree polynomial and outputs train and validation MSE
    Fits a linear regression model y = sum_{i=1}^n w_i x^i to the given train
    set and outputs the mean-squared-error (MSE) on train and validation sets

    Args:
    x_train: Input features for the train set. Has shape (num_train, )
    y_train: Targets (labels) for the train set. Has shape (num_train, )
    x_val: Input features for the validation set. Has shape (num_val, )
    y_val: Targets (labels) for the validation set. Has shape (num_val, )
    n: The degree of the polynomial fit. See the above equation

    Returns:
    (train_mse, val_mse), tuple of MSE on train and validation sets.
    """
    # Fit the model on the train set.
    input_features_train = get_features(x_train, n)
    reg_model = linreg(fit_intercept = False).fit(input_features_train,␣
↪y_train)

    # Generate model predictions for the train set and calculate the MSE.
    y_predict_train = reg_model.predict(input_features_train)
    y_diff_train = y_predict_train - y_train
    train_mse = np.dot(y_diff_train, y_diff_train)/(2*y_diff_train.shape[0])

    # Similarly, calculate the MSE on the val set.
    input_features_val = get_features(x_val, n)
    y_predict_val = reg_model.predict(input_features_val)
    y_diff_val = y_predict_val - y_val
    val_mse = np.dot(y_diff_val, y_diff_val)/(2*y_diff_val.shape[0])

    return train_mse, val_mse
```

Q1). c).

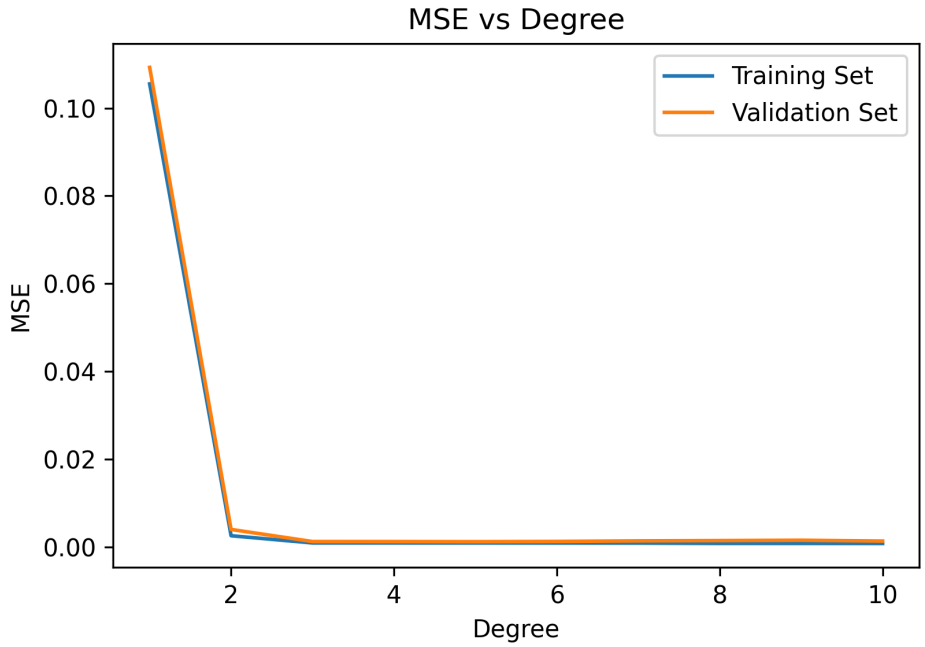

Figure 2: Variation of MSE with the Degree

n (Degree)	Training MSE	Validation MSE
1	0.105510856	0.109263238
2	0.002561212	0.003992644
3	0.000915605	0.001235369
4	0.000914634	0.001230391
5	0.000911332	**0.001204071**
6	0.000895046	0.001259852
7	0.000880826	0.001391642
8	0.000807672	0.00145661
9	0.000806882	0.001534682
10	0.000793088	0.001347577

Table 1: Training and Validation MSE for Different n Values

- As expected, the training MSE continues to decrease with increasing degree (n) values. An appropriate degree should be selected by observing the variation of the validation MSE.
- From the above table, we can observe that the validation MSE is decreasing until n = 5 and starts to increase for $n > 5$. That is, the regression model tends to overfit the given training data when $n > 5$.
- Accordingly, $\mathbf{n = 5}$ was selected as the most suited degree for the given dataset.

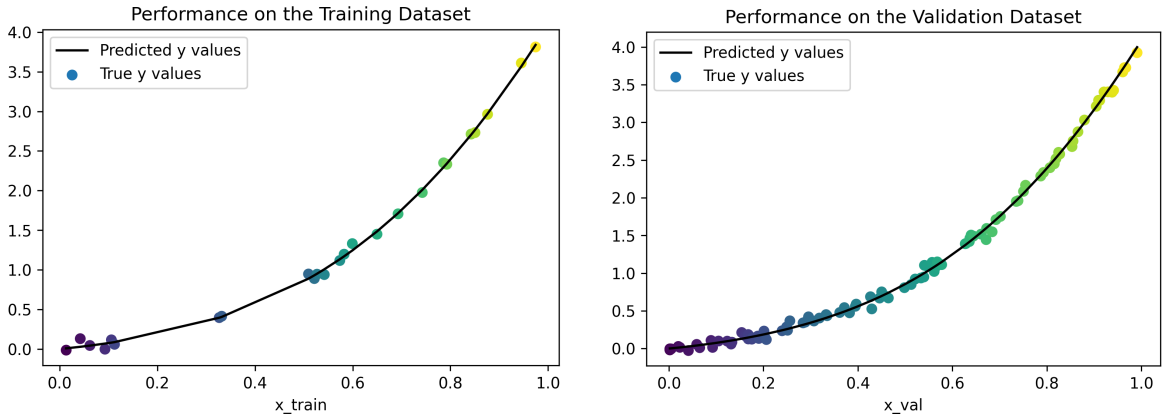

Figure 3: Performance of the Regression Model (Degree = 5, No Intercept)

Q1). d).

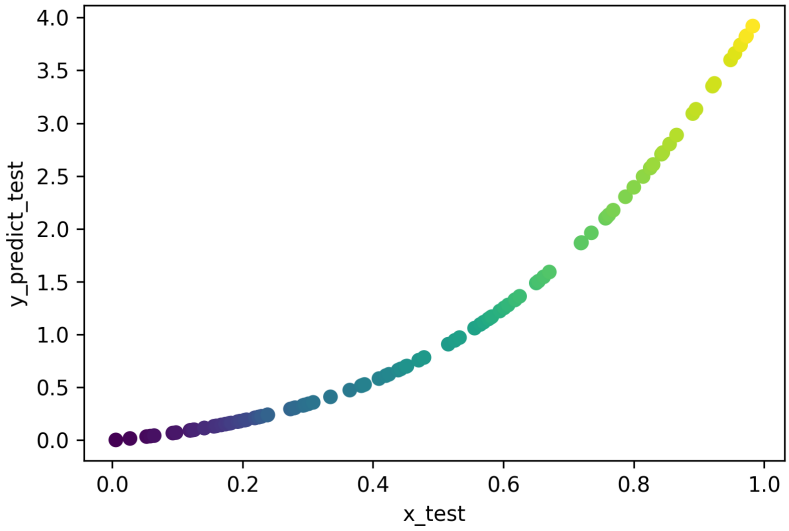

Figure 4: Predicted y Values for Test Data

Table 2: Inference Results

X Test	Prediction
0.120295125	0.094251281
0.159607685	0.1363602
0.555885863	1.060826573
0.409183485	0.583456396
0.092902904	0.068341402
0.382131781	0.515980104
0.238678034	0.240745942
0.564575982	1.095798678
0.141123103	0.115814241

Table 2: Inference Results

X Test	Prediction
0.56985869	1.117461246
0.064349696	0.044175601
0.238139762	0.239936777
0.470175362	0.75758257
0.387091472	0.527931788
0.814398434	2.495586747
0.424538422	0.624327774
0.23767409	0.239237918
0.452108346	0.702651655
0.308317409	0.358539206
0.280618629	0.308443965
0.273420773	0.296150559
0.82535145	2.576045276
0.224194551	0.219472831
0.119195631	0.093158419
0.842697103	2.706936771
0.890658001	3.091116569
0.364425918	0.474792866
0.206010209	0.194189122
0.982667378	3.919162275
0.419838088	0.611612761
0.653864944	1.505041913
0.768513979	2.176826346
0.761810435	2.132705451
0.515597269	0.909138817
0.19232126	0.17615836
0.141359054	0.116068028
0.125767104	0.099757328
0.442740821	0.675309155
0.278573942	0.304922253
0.20291296	0.190035593
0.438764505	0.663932491
0.719164897	1.866359329
0.580388063	1.161560012
0.295554144	0.334894555
0.948633998	3.599090407
0.844663447	2.722044348

Table 2: Inference Results

X Test	Prediction
0.525610943	0.945272116
0.756858211	2.100507884
0.963817295	3.73991151
0.027205843	0.016947621
0.383959908	0.520363997
0.229137258	0.226616992
0.450637434	0.698307549
0.576974038	1.147126171
0.478315643	0.783301969
0.228572199	0.225794253
0.123675133	0.097639107
0.895516469	3.131860067
0.973072001	3.827308782
0.532753231	0.9716694
0.829783959	2.609087657
0.855061976	2.80285091
0.624943429	1.362181003
0.173572163	0.152797154
0.758664402	2.112212117
0.052709362	0.035134906
0.719910599	1.870806397
0.450979339	0.699315615
0.582326962	1.169815509
0.618498505	1.331724536
0.182219569	0.163384072
0.670550978	1.59217036
0.224538899	0.21996672
0.218343805	0.211168064
0.96390193	3.740705422
0.005519119	0.003231941
0.650590576	1.48835274
0.800198693	2.39379526
0.301131676	0.345106923
0.661841573	1.546256973
0.972150533	3.818554107
0.923886491	3.376458353
0.195856428	0.180734451

Table 2: Inference Results

X Test	Prediction
0.292732532	0.329798338
0.09787352	0.072841621
0.92095799	3.350682684
0.601291759	1.252815779
0.219763818	0.213168685
0.607331047	1.280116306
0.594525549	1.222731032
0.179121061	0.159554091
0.735036745	1.962617007
0.865826185	2.888119629
0.167148383	0.14513611
0.334814147	0.410849028
0.058496272	0.039571244
0.787350718	2.304130709
0.617512748	1.327109791
0.155815032	0.132033653
0.955289399	3.660423882