

# Introduction to Machine Learning and Artificial Neural Networks (CS554)

## Osman Furkan KINLI – S002969 - Assignment-3

### Non-Linear Dimensionality Reduction (Polynomial Regression & KNN Regression & MLP Regression)

#### 1. Introduction

In this assignment, we test three different algorithms (polynomial regression, k-th nearest neighbor and multi-layer perceptron) on a custom data set. Training and test data contains 25 samples for each, and given in the CSV format where each line represent one sample, the first value is independent variable  $x$ , and the second one is dependent variable  $y$ .

#### 2. Methodology

- Implement Polynomial Regressor where the degree is a hyper-parameter
- Implement KNN Regressor where  $K$  is a hyper-parameter
- Implement Multi-layer Perceptron where number of hidden units is a hyper-parameter
- Train these 3 algorithms with train set
- Choose best model for each algorithm by using validation set
- Plot the best model for each algorithm with the test set
- Report mean-squared error of the best model for each algorithm on the test set
- For MLP case, with the best model, plot the learned lines, the output of each hidden unit and the actual output

#### 3. Code

```
import csv
import numpy as np
import matplotlib.pyplot as plt

def read_data(fname):
    x, y = list(), list()
    with open(fname, 'rt') as file:
        lines = csv.reader(file)
        for line in lines:
            x.append(float(line[0]))
            y.append(float(line[1]))
    return np.array(x), np.array(y)

train_x, train_y = read_data('./data/train25.csv')
val_x, val_y = read_data('./data/validation25.csv')
test_x, test_y = read_data('./data/test25.csv')

print("-----")
```

```
print("TASK: Polynomial Regression")
```

```
def vectorize_poly(x, degree):  
    vector = list()  
    for d in range(degree, -1, -1):  
        vector.append(x**d)  
        # print(vector)  
    return np.array(vector).transpose()
```

```
def poly_reg_lse_solver(x, y):  
    return np.dot(np.dot(np.linalg.inv(np.dot(x.transpose(), x)),  
x.transpose()), y)
```

```
def mse_loss(y_true, y_pred):  
    return np.square(np.subtract(y_true, y_pred)).mean()
```

```
def validate(x, w, y):  
    y_hat = np.dot(x, w)  
    return mse_loss(y, y_hat)
```

```
def cross_validate(x_train, y_train, x_val, y_val, max_degree=10):  
    losses = list()  
    for degree in range(1, max_degree+1):  
        vec_x_train = vectorize_poly(x_train, degree)  
        vec_x_val = vectorize_poly(x_val, degree)
```

```
        assert vec_x_train.shape == (25, degree+1)  
        assert vec_x_val.shape == (25, degree+1)
```

```
        w_hat = poly_reg_lse_solver(vec_x_train, y_train)
```

```
        losses.append((degree, validate(vec_x_val, w_hat, y_val), w_hat))  
    return losses
```

```
losses = cross_validate(train_x, train_y, val_x, val_y)  
best_degree, _, best_w_hat = min(losses, key=lambda l: l[1])
```

```
# print(best_degree)  
# print(best_w_hat)
```

```
vec_x_test = vectorize_poly(test_x, best_degree)  
best_model = np.dot(vec_x_test, best_w_hat)
```

```
plt.plot(test_x, test_y, "b+", test_x, best_model, "r")  
plt.xlabel("X")  
plt.ylabel("Value")  
# plt.savefig('./images/poly_reg_best_model_fit_on_test.png')  
plt.show()  
print("MSE Error on Test set: {}".format(validate(vec_x_test, best_w_hat,  
test_y)))  
print("-----")
```

```
print("TASK: K-Nearest Neighbour Regression")
```

```
def calc_dist(x1, x2):  
    return (x1 - x2)**2
```

```
def knn_regressor(k, x_train, y_train, x_val):  
    y_pred = list()  
    for x_v in x_val:  
        distances = list()  
        for x_t, y_t in zip(x_train, y_train):  
            distances.append((y_t, calc_dist(x_v, x_t)))  
        distances = sorted(distances, key=lambda d: d[1])  
        distances = distances[:k]  
  
        y_pred.append(sum([d[0] for d in distances]) / k)  
    return y_pred
```

```
def find_best_k(x_train, y_train, x_val, y_val, max_k=10):  
    losses = list()  
    for k in range(1, max_k+1):  
        y_pred = knn_regressor(k, x_train, y_train, x_val)  
        losses.append((k, mse_loss(y_val, y_pred)))  
    return min(losses, key=lambda l: l[1])[0]
```

```
best_k = find_best_k(train_x, train_y, val_x, val_y)  
y_pred = knn_regressor(best_k, train_x, train_y, test_x)
```

```
plt.plot(test_x, test_y, "b+", test_x, y_pred, "r")  
plt.xlabel("X")  
plt.ylabel("Value")  
# plt.savefig('./images/knn_reg_best_model_fit_on_test.png')  
plt.show()  
print("MSE Error on Test set: {}".format(mse_loss(test_y, y_pred)))  
print("-----")
```

```
print("TASK: Multilayer Perceptron")  
NUM_EPOCH = 10  
LR = 0.00001
```

```
def init_weights(in_channel, out_channel):  
    epsilon = np.sqrt(2.0 / (in_channel * out_channel))  
    w = epsilon * np.random.randn(out_channel, in_channel)  
    return w.transpose()
```

```
def init_bias():  
    return 0
```

```
def run_mlp(num_hidden_unit, X, y, weights=None, bias=None, is_train=True):  
    if weights is None:  
        w1 = init_weights(1, num_hidden_unit)  
        w2 = init_weights(num_hidden_unit, 1)
```

```
else:
    w1, w2 = weights
```

```
if bias is None:
    b1 = init_bias()
    b2 = init_bias()
else:
    b1, b2 = bias
```

```
def _sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def _derivative_sigmoid(x):
    return x * (1 - x)
```

```
def forward(X_):
    z1 = np.dot(X_, w1) + b1
    a1 = _sigmoid(z1)
    z2 = np.dot(a1, w2) + b2
    return z1, z2
```

```
def step(X_, y_, z1_, z2_, w1_, w2_): # TODO
    layer2_error = y_ - z2_
    layer2_delta = layer2_error * _derivative_sigmoid(z2_)
    layer1_error = np.dot(layer2_delta, w2_.transpose())
    layer1_delta = layer1_error * _derivative_sigmoid(z1_)
```

```
    layer1_adjustment = np.dot(X_.transpose(), layer1_delta)
    layer2_adjustment = np.dot(z1_.transpose(), layer2_delta)
```

```
    # Adjust the weights.
    w1_ += LR * layer1_adjustment
    w2_ += LR * layer2_adjustment
    return w1_, w2_
```

```
if is_train:
    losses = list()
    for _ in range(NUM_EPOCH):
        z1, z2 = forward(X)
        losses.append(0.5 * np.square(y - z2).mean())
        w1, w2 = step(X, y, z1, z2, w1, w2)
    final_loss = losses[-1]
else:
    _, z2 = forward(X)
    final_loss = 0.5 * np.square(y - z2).mean()
```

```
    return (w1, w2), (b1, b2), final_loss, z2
```

```
model_lst = list()
for i in range(1, 101):
    weights, bias, _, _ = run_mlp(i, np.expand_dims(train_x, axis=-1),
np.expand_dims(train_y, axis=-1))
    _, _, val_loss, _ = run_mlp(i, np.expand_dims(val_x, axis=-1),
np.expand_dims(val_y, axis=-1),
weights=weights, bias=bias, is_train=False)
    model_lst.append((i, weights, bias, val_loss))
```

```
best_mlp_model = min(model_lst, key=lambda m: m[3])
```

```

best_num_hidden_unit, best_weights, best_bias, _ = best_mlp_model

_, _, test_loss, y_pred = run_mlp(best_num_hidden_unit,
np.expand_dims(test_x, axis=-1),
np.expand_dims(test_y, axis=-1),
weights=best_weights,
bias=best_bias, is_train=False)

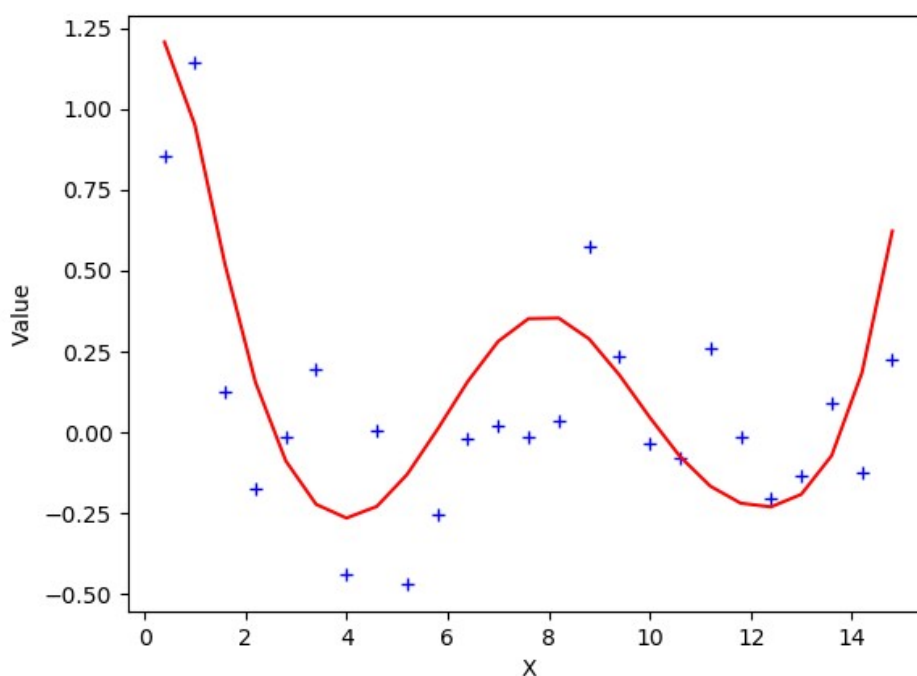
plt.plot(test_x, test_y, "b+", test_x, y_pred, "r")
plt.xlabel("X")
plt.ylabel("Value")
# plt.savefig('./images/mlp_best_model_fit_on_test.png')
plt.show()
print("MSE Error on Test set: {}".format(test_loss))
print("-----")

# plt.plot(train_x, train_y, "b+", train_x, y_pred, "r")
# plt.xlabel("X")
# plt.ylabel("Value")
# # plt.savefig('./images/mlp_best_model_fit_on_test.png')
# plt.show()
# print("MSE Error on Test set: {}".format(test_loss))
# print("-----")

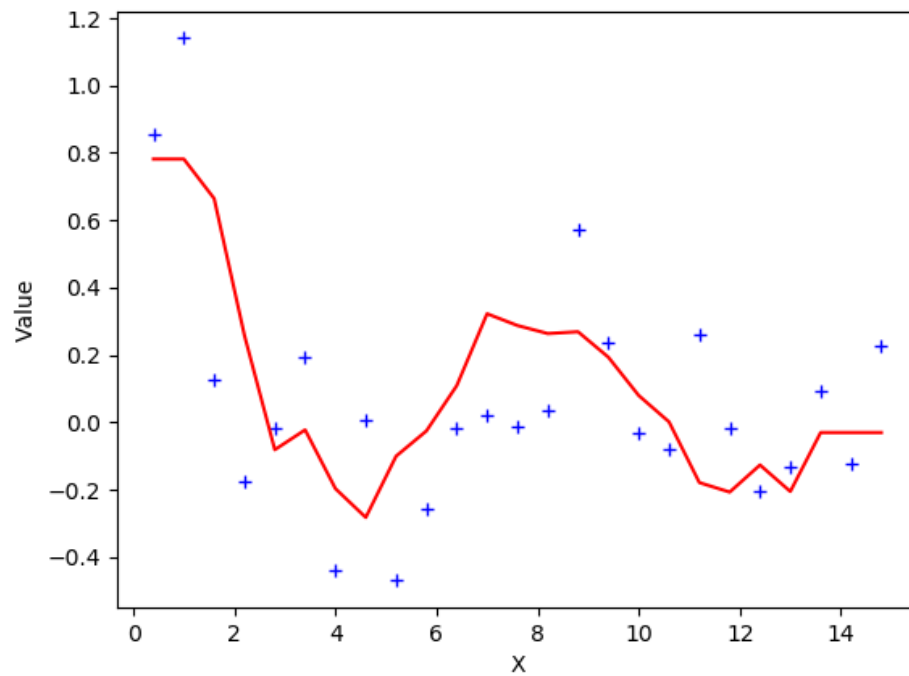
```

## 4. Results

- *Best model fitted on test set for Polynomial Regression*



- ***Best model fitted on test set for KNN Regression***



- ***Best model fitted on test set for MLP Regression***

