MT1

November 7, 2023

1 Mid-Term 1

[2]:
```
import time
import scipy
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
```

[3]:
```
def load_dataset(name):
    X, y = [], []
    with open("{}".format(name), 'r') as f:
        for line in f.readlines():
            splitted = line.split(",")
            X.append(splitted[:-1])
            y.append(splitted[-1])
    X, y = np.asarray(X, dtype=np.float32), np.asarray(y, dtype=np.float32)
    return X, y
```

1.0.1 Prelude

Load the following 7 datasets:

- dataset 1: (X1, y1) with $X1 \in \mathbb{R}^{50\times 10}$ and noise variance $\sigma = 0.1$
- dataset 2: (X2, y2) with $X2 \in \mathbb{R}^{500\times 10}$ and noise variance $\sigma = 0.1$
- dataset 3: (X3, y3) with $X3 \in \mathbb{R}^{1000\times 10}$ and noise variance $\sigma = 0.1$
- dataset 4: (X4, y4) with $X4 \in \mathbb{R}^{500\times 500}$ and noise variance $\sigma = 0.1$
- dataset 5: (X5, y5) with $X5 \in \mathbb{R}^{500\times 5000}$ and noise variance $\sigma = 0.1$
- dataset 6: (X6, y6) with $X6 \in \mathbb{R}^{500\times 10000}$ and noise variance $\sigma = 0.1$
- dataset 7: (X7, y7) with $X7 \in \mathbb{R}^{500\times 1}$ and noise variance $\sigma = 0.3$

i.e. datasets from 1 to 3 have fixed input space dimension $d = 10$ and different number of points, while datasets from 4 to 6 have fixed number of points and different number of dimensions.

[59]:
```
# Load dataset 1
(Xtr_1, ytr_1), (Xte_1, yte_1) = load_dataset(
    "./datasets/dataset_1_train"), load_dataset("./datasets/dataset_1_test")

# Load dataset 2
(Xtr_2, ytr_2), (Xte_2, yte_2) = load_dataset(
```

```
    "./datasets/dataset_2_train"), load_dataset("./datasets/dataset_2_test")

# Load dataset 3
(Xtr_3, ytr_3), (Xte_3, yte_3) = load_dataset(
    "./datasets/dataset_3_train"), load_dataset("./datasets/dataset_3_test")

# Load dataset 4
(Xtr_4, ytr_4), (Xte_4, yte_4) = load_dataset(
    "./datasets/dataset_4_train"), load_dataset("./datasets/dataset_4_test")

# Load dataset 5
(Xtr_5, ytr_5), (Xte_5, yte_5) = load_dataset(
    "./datasets/dataset_5_train"), load_dataset("./datasets/dataset_5_test")

# Load datset 6
(Xtr_6, ytr_6), (Xte_6, yte_6) = load_dataset(
    "./datasets/dataset_6_train"), load_dataset("./datasets/dataset_6_test")

# Load datset 7
Xtr_7, ytr_7 = load_dataset("./datasets/dataset_7_train")

# SUGGESTION!!! Check the size of each dataset

# Example
print(np.shape(Xtr_7))
```

```
(400, 1)
```

Tips and tricks: - to estimate the computational time of a certain portion of code use the following `

```
t0 = time.time()
... my code ...
mycode_time = time.time()-t0`
```

[4]:
```
# Example

t0 = time.time()
_ = load_dataset("./datasets/dataset_3_train")
DeltaT = time.time() - t0

print("[--] Dataset 3 loaded in {} seconds".format(DeltaT))
```

```
[--] Dataset 3 loaded in 0.002925872802734375 seconds
```

1.0.2 Activity 1

Compare the behavior of K-NN and RLS on the datasets according to the following tasks: - Task 1.1: Compare training, validation, test errors and training and test time of the two methods on datasets 1, 2, 3 - Task 1.2: Compare training, validation, test errors and training and test time of the two methods on datasets 4, 5, 6

[45]:
```python
# Utils
def euclidDistance(P1, P2):
    return np.linalg.norm(P1-P2, 2)


def allDistances(X1, X2):
    D = np.zeros((X1.shape[0], X2.shape[0]))
    for idx1 in range(len(X1)):
        for idx2 in range(len(X2)):
            D[idx1, idx2] = euclidDistance(X1[idx1, :], X2[idx2, :])
    return D


def flipLabels(Y, P):
    if P < 1 or P > 100:
        print("p should be a percentage value between 0 and 100.")
        return -1

    if any(np.abs(Y) != 1):
        print("The values of Ytr should be +1 or -1.")
        return -1

    Y_noisy = np.copy(np.squeeze(Y))
    if Y_noisy.ndim > 1:
        print("Please supply a label array with only one dimension")
        return -1

    n = Y_noisy.size
    n_flips = int(np.floor(n * P / 100))
    idx_to_flip = np.random.choice(n, size=n_flips, replace=False)
    Y_noisy[idx_to_flip] = -Y_noisy[idx_to_flip]

    return Y_noisy

def calcError(Ypred, Ytrue):
    return np.mean((Ypred-Ytrue)**2)

def plot_knn_errors(k_list, val_mean, val_var, tr_mean, tr_var, dataset_n):
    _, ax = plt.subplots()
    ax.set_title(f"KNN error (Dataset {dataset_n})")
    ax.errorbar(k_list, val_mean, val_var, label="Validation error")
    ax.errorbar(k_list, tr_mean, tr_var, label="Training error")
    # Only show integer labels on x-axis
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))
```

```
    ax.legend(loc="best")
    ax.set_ylabel("Error")
    ax.set_xlabel("K")

def plot_rls_errors(lam_list, bestlam, Vm, Tm, dataset_n):
    _, ax = plt.subplots()
    ax.set_title(f"RLS error (Dataset {dataset_n})")
    ax.plot(lam_list, Vm, '-o', label="Validation error")
    ax.plot(lam_list, Tm, '-o', label="Train error")
    ax.axvline(bestlam, linestyle="--", c="red", alpha=0.7,
               label=f"best $\lambda$ ({bestlam:.2e})")
    ax.set_xscale("log")
    ax.set_xlabel("$\lambda$")
    ax.set_ylabel("MSE")
    ax.legend(loc="best")
```

```
<>:55: SyntaxWarning: invalid escape sequence '\l'
<>:57: SyntaxWarning: invalid escape sequence '\l'
<>:55: SyntaxWarning: invalid escape sequence '\l'
<>:57: SyntaxWarning: invalid escape sequence '\l'
/var/folders/q5/2vqyn_hx6v7gdx1c0f59g4300000gn/T/ipykernel_89066/3847833817.py:5
5: SyntaxWarning: invalid escape sequence '\l'
  label=f"best $\lambda$ ({bestlam:.2e})")
/var/folders/q5/2vqyn_hx6v7gdx1c0f59g4300000gn/T/ipykernel_89066/3847833817.py:5
7: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel("$\lambda$")
```

[46]:

```
# CV for KNN (regression)

def kNNRegression(Xtr, Ytr, k, Xte):
    n_train = Xtr.shape[0]
    n_test = Xte.shape[0]

    if k > n_train:
        k = n_train

    Ypred = np.zeros(n_test)

    dist = allDistances(Xte, Xtr)

    for idx in range(n_test):
        neigh_indexes = np.argsort(dist[idx, :])[:k]
        avg_neigh = np.mean(Ytr[neigh_indexes])
        Ypred[idx] = avg_neigh

    return Ypred
```

```python
def KFoldCVkNN(Xtr, Ytr, KF, k_list):
    if KF <= 0:
        print("Please supply a positive number of repetitions")
        return -1

    # Ensures that k_list is a numpy array
    k_list = np.array(k_list)
    num_k = k_list.size

    n_tot = Xtr.shape[0]
    # Number of values in the interval of the validation set
    n_val = int(n_tot // KF)

    # We want to compute 1 error for each `k` and each fold
    tr_errors = np.zeros((num_k, KF))
    val_errors = np.zeros((num_k, KF))

    for kdx, k in enumerate(k_list):
        # `split_idx`: a list of arrays, each containing the validation indices␣
↪for 1 fold
        # We generate a random vector of n_tot elements with no repetitions
        rand_idx = np.random.choice(n_tot, size=n_tot, replace=False)
        # Then we split it into KF subarrays
        split_idx = np.array_split(rand_idx, KF)
        for fold in range(KF):
            # Set the indices in boolean mask for all validation samples to␣
↪`True`
            # We generate a boolean array of n_tot elements all to False
            val_mask = np.zeros(n_tot, dtype=bool)
            # Then we get the validation set by setting to True the starting␣
↪index of the current fold
            val_mask[split_idx[fold]] = True

            # NOTE: with this notation Xtr[~val_mask] we are taking all those␣
↪elements
            # of which val_mask = False (like Xtr[val_mask == False])
            # The training set are the one that are val_mask = False
            X = Xtr[~val_mask]
            Y = Ytr[~val_mask]
            # The validation set is the one with val_mask = True
            X_val = Xtr[val_mask]
            Y_val = Ytr[val_mask]

            # Compute the training error of the kNN classifier for the given␣
↪value of k
            tr_errors[kdx, fold] = calcError(kNNRegression(X, Y, k, X), Y)
```

```
            # Compute the validation error of the kNN classifier for the given␣
↪value of k
            val_errors[kdx, fold] = calcError(
                kNNRegression(X, Y, k, X_val), Y_val)

    # Calculate error statistics along the repetitions
    # NOTE: axis=1 means we are taking the mean/variance the rows
    # np.mean([[1, 2], [3, 4]], axis=1) = 2+1/2, 3+4/2
    tr_mean = np.mean(tr_errors, axis=1)
    tr_var = np.var(tr_errors, axis=1)
    val_mean = np.mean(val_errors, axis=1)
    val_var = np.var(val_errors, axis=1)

    # The best k is the one that minimizes the validation error expectation
    best_k_idx = np.argmin(val_mean)
    best_k = k_list[best_k_idx]

    return best_k, val_mean, val_var, tr_mean, tr_var
```

1.0.3 regularizedLSTrain

if $n > d \implies w_\lambda = (X^T X + \lambda n I)^{-1} \ X^T Y$

if $n \leq d \implies$

$$c = (X X^T + \lambda n I)^{-1} \ Y$$

$$w_\lambda = X^T c$$

[50]:
```
# CV for RLS (regression)

def regularizedLSTrain(Xtr, Ytr, lam):
    n = Xtr.shape[0]
    d = Xtr.shape[1]

    if n > d:
        XTY = Xtr.T @ Ytr
        XTX = Xtr.T @ Xtr
        return np.linalg.inv(XTX + lam * n * np.identity(d)) @ XTY
    # else n <= d
    XXT = Xtr @ Xtr.T
    c = np.linalg.inv(XXT + lam * n * np.identity(n)) @ Ytr
    return Xtr.T @ c
```

```python
def regularizedLSTest(w, Xte):
    return np.dot(Xte, w)


def KFoldCVRLS(Xtr, Ytr, KF, regpar_list):
    if KF <= 1:
        raise Exception("Please supply a number of fold > 1")

    # Ensures that regpar_list is a numpy array
    regpar_list = np.array(regpar_list)
    num_regpar = regpar_list.size

    n_tot = Xtr.shape[0]
    n_val = int(n_tot // KF)

    # We want to compute 1 error for each `k` and each fold
    tr_errors = np.zeros((num_regpar, KF))
    val_errors = np.zeros((num_regpar, KF))

    for idx, regpar in enumerate(regpar_list):
        # `split_idx`: a list of arrays, each containing the validation indices␣
↪for 1 fold
        rand_idx = np.random.choice(n_tot, size=n_tot, replace=False)
        split_idx = np.array_split(rand_idx, KF)
        for fold in range(KF):
            # Set the indices in boolean mask for all validation samples to␣
↪`True`
            val_mask = np.zeros(n_tot, dtype=bool)
            val_mask[split_idx[fold]] = True

            # Use the boolean mask to split X, Y in training and validation part

            X = Xtr[~val_mask]  # training input
            Y = Ytr[~val_mask]  # training output
            X_val = Xtr[val_mask]  # validation input
            Y_val = Ytr[val_mask]  # validation output

            # Train a RLS model for a single fold, and the given value of␣
↪`regpar`
            currW = regularizedLSTrain(X, Y, regpar)

            # Compute the training error of the RLS regression for the given␣
↪value of regpar
            YpredTR = regularizedLSTest(currW, X)
            tr_errors[idx, fold] = calcError(YpredTR, Y)
```

```python
            # Compute the validation error of the RLS regression for the given
↪value of regpar
            YpredVAL = regularizedLSTest(currW, X_val)
            val_errors[idx, fold] = calcError(YpredVAL, Y_val)

    # Calculate error statistics along the repetitions
    tr_mean = np.mean(tr_errors, axis=1)
    tr_var = np.var(tr_errors, axis=1)
    val_mean = np.mean(val_errors, axis=1)
    val_var = np.var(val_errors, axis=1)

    bestlam_idx = np.argmin(val_mean)
    bestlam = regpar_list[bestlam_idx]

    return bestlam, val_mean, val_var, tr_mean, tr_var
```

2 Task 1.1

Tips: to compare the methods, you should

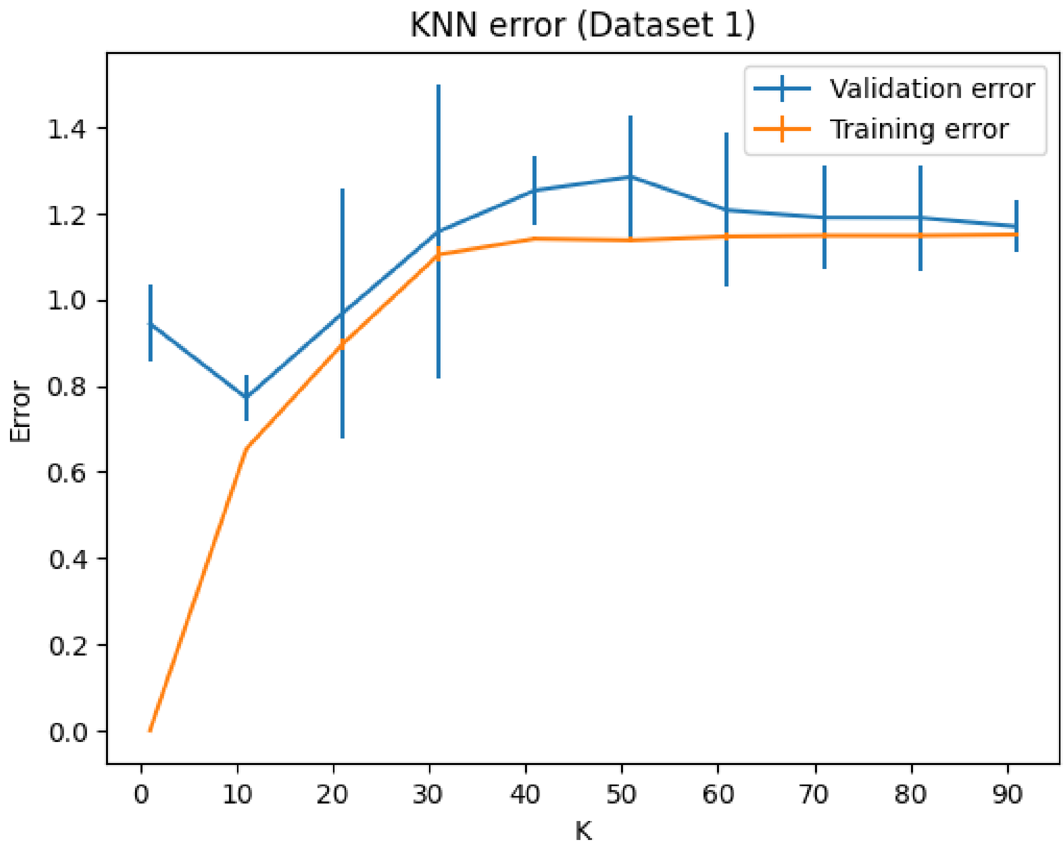
- plot training and validation errors for the different hypeparameter values considered in the cross-validation procedure
- print in output the training, validation and test errors corresponding to the final model

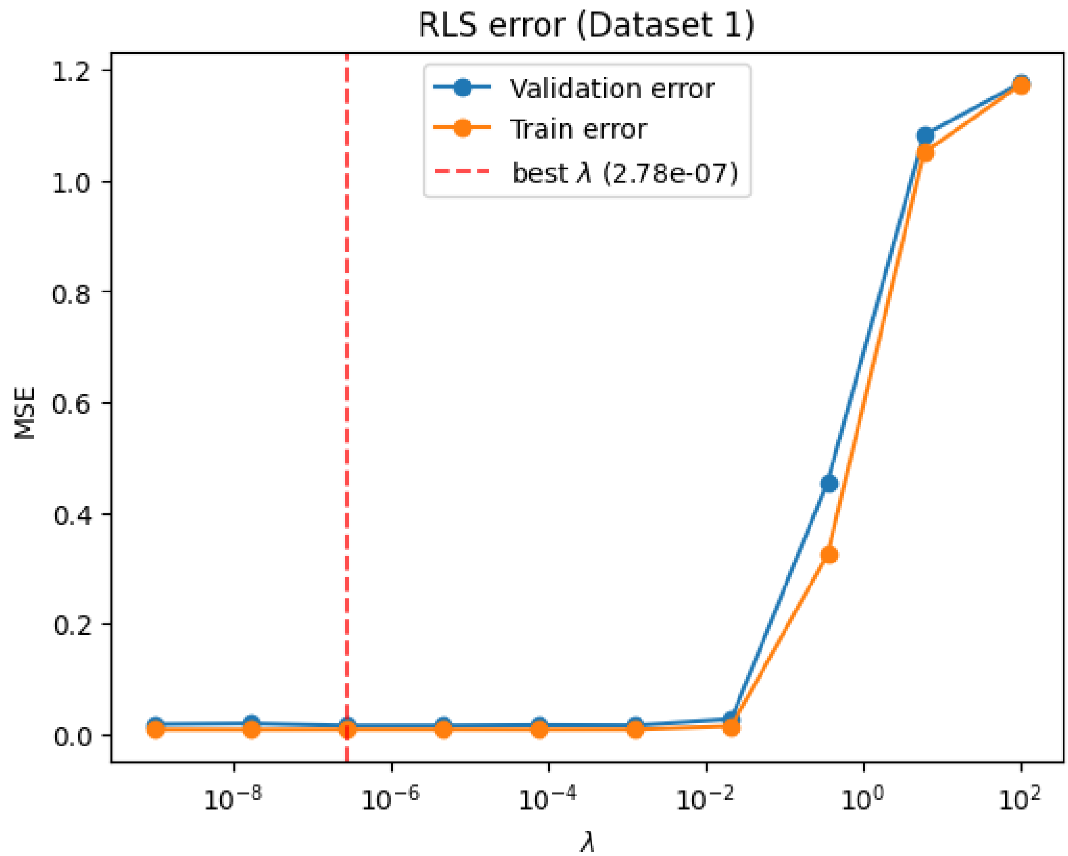
2.0.1 Dataset 1

[51]:
```python
def analyze_dataset(Xtr, Ytr, Xte, Yte, i):
    """
    Apply K-NN and RLS, then measures the elapsed time and the errors for both
    the algorithms
    """
    Ks = list(range(1, 100, 10))
    lams = np.logspace(-9, 2, 10)
    KF = 5

    # cross validation of KNN for dataset i
    best_k, val_mean_knn, val_var, tr_mean, tr_var = KFoldCVkNN(
        Xtr, Ytr, KF, Ks)
    plot_knn_errors(Ks, val_mean_knn, val_var, tr_mean, tr_var, i)

    # cross validation of RLS for dataset i
    bestlam, val_mean_rls, val_var, tr_mean, tr_var = KFoldCVRLS(
        Xtr, Ytr, KF, lams)
    plot_rls_errors(lams, bestlam, val_mean_rls, tr_mean, i)

    # train KNN with parameter obtained by KFold-Cross Validation and estimate
↪computational time
```

```
    t0 = time.time()
    y_pred_tr = kNNRegression(Xtr, Ytr, best_k, Xtr)
    DeltaT_tr = time.time() - t0
    t0 = time.time()
    y_pred_te = kNNRegression(Xtr, Ytr, best_k, Xte)
    DeltaT_te = time.time() - t0

    print(f"Dataset {i}:")
    print("\tKNN")
    print("[--] Training time:", DeltaT_tr, "s")
    print("[--] Training error:", calcError(y_pred_tr, Ytr))
    print("[--] Test time:", DeltaT_te, "s")
    print("[--] Test error:", calcError(y_pred_te, Yte))
    print("[--] Validation error:", val_mean_knn[best_k // 10])

    # train RLS with parameter obtained by KFold-Cross Validation and estimate
 ↪computational time
    t0 = time.time()
    w = regularizedLSTrain(Xtr, Ytr, bestlam)
    y_pred_rls = regularizedLSTest(w, Xte)
    DeltaT = time.time() - t0

    print("\tRLS")
    print("[--] Training time", DeltaT, "s")
    print("[--] Training error", calcError(y_pred_rls, Yte))
    bestlam_idx = np.argmin(val_mean_rls)
    print("[--] Validation error:", val_mean_rls[bestlam_idx])
    print("\n")


analyze_dataset(Xtr_1, ytr_1, Xte_1, yte_1, 1)
```

```
Dataset 1:
        KNN
[--] Training time: 0.002990245819091797 s
[--] Training error: 0.5867178441367273
[--] Test time: 0.0007507801055908203 s
[--] Test error: 0.47875564129291276
[--] Validation error: 0.7715256062025372
        RLS
[--] Training time 0.006372213363647461 s
[--] Training error 0.02064345384193937
[--] Validation error: 0.015474526879158595
```

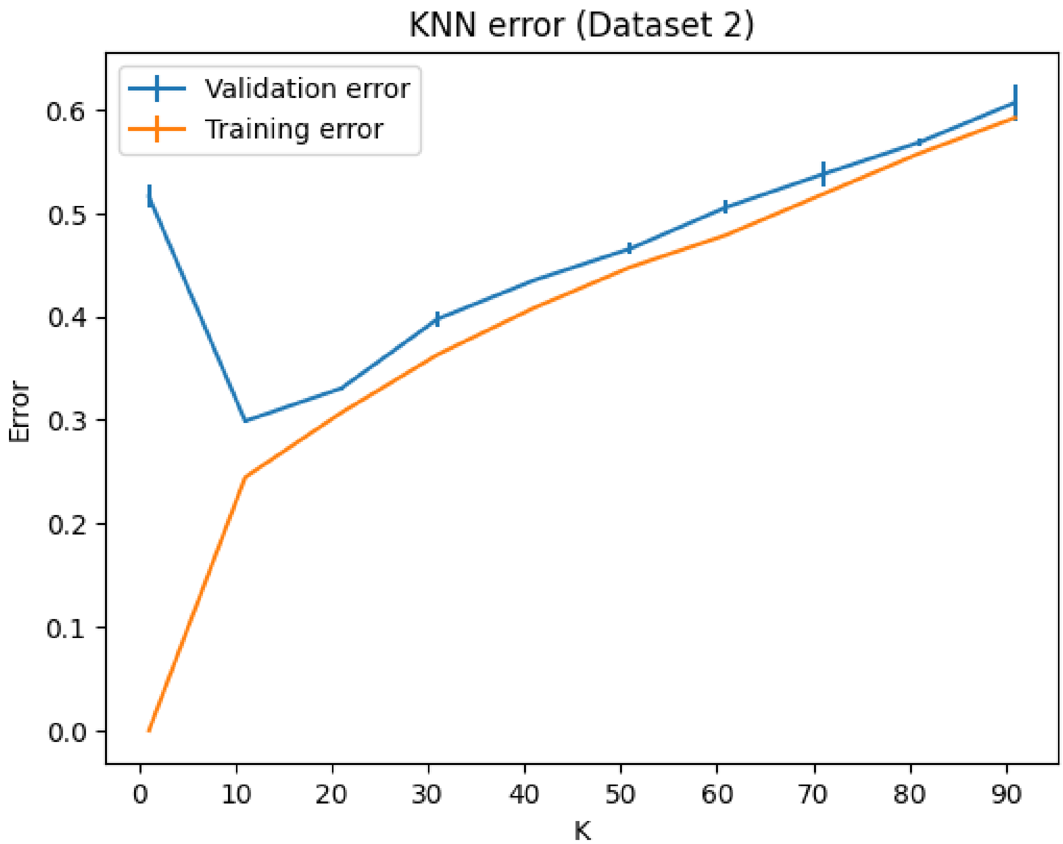

KNN error (Dataset 1)
Validation error
Training error
Error
K
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50
60
70
80
90

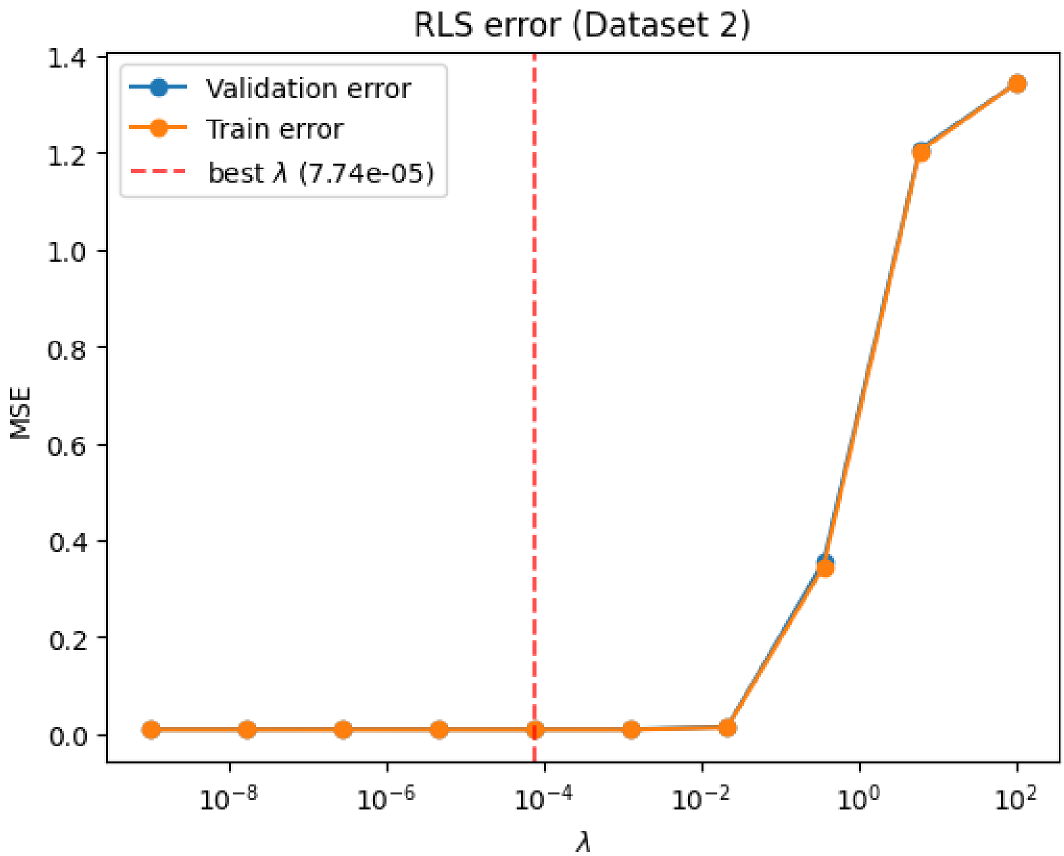


2.0.2 Dataset 2

[52]:
```
analyze_dataset(Xtr_2, ytr_2, Xte_2, yte_2, 2)
```

```
Dataset 2:
        KNN
[--] Training time: 0.36838603019714355 s
[--] Training error: 0.21508438776540637
[--] Test time: 0.06739211082458496 s
[--] Test error: 0.18747983784491468
[--] Validation error: 0.29901946033994686
        RLS
[--] Training time 0.0011429786682128906 s
[--] Training error 0.009834272600519266
[--] Validation error: 0.010288093813573696
```

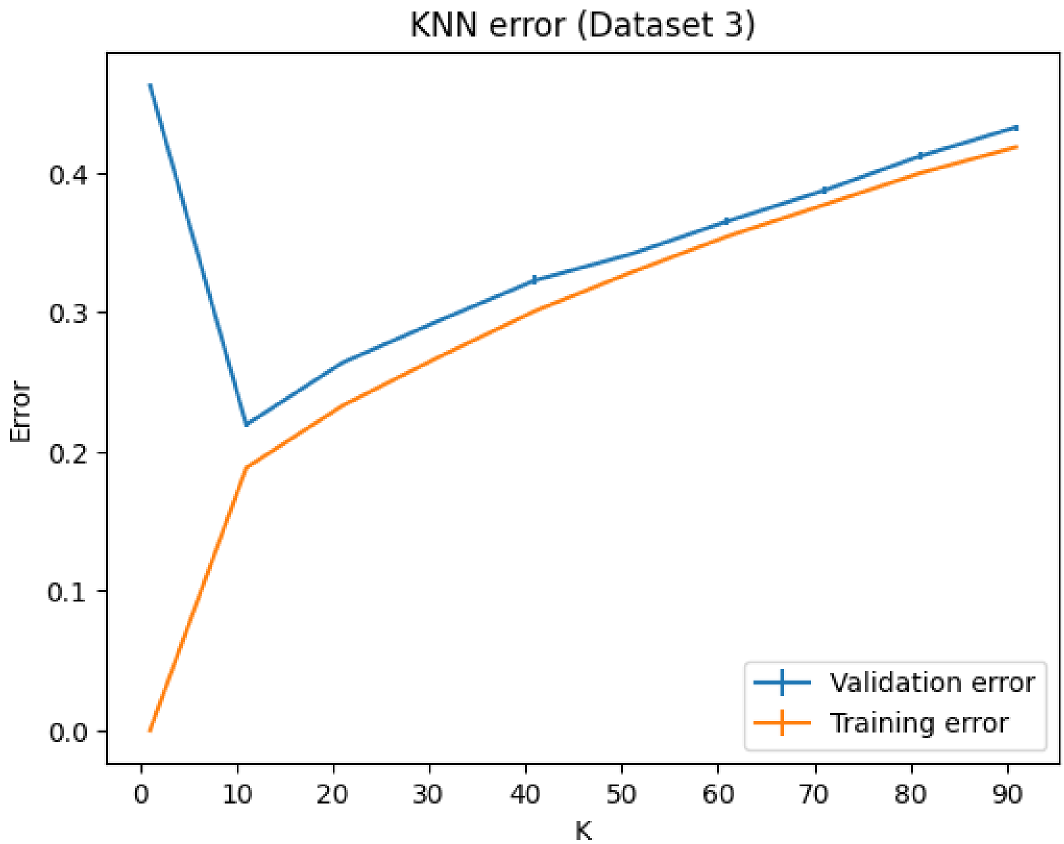

KNN error (Dataset 2)
Validation error
Training error
Error
K
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70
80
90



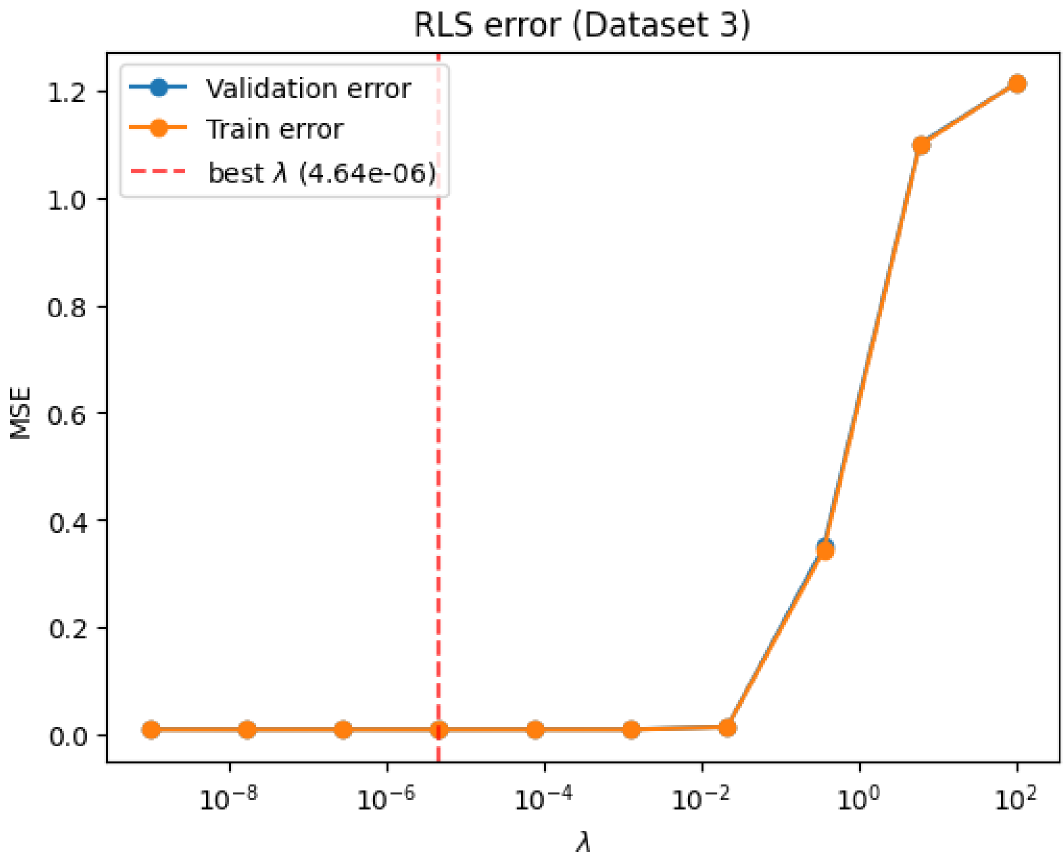
2.0.3 Dataset 3

[53]:
```
analyze_dataset(Xtr_3, ytr_3, Xte_3, yte_3, 3)
```

```
Dataset 3:
        KNN
[--] Training time: 1.1813061237335205 s
[--] Training error: 0.17270680939455182
[--] Test time: 0.2698047161102295 s
[--] Test error: 0.2127557443601381
[--] Validation error: 0.21941296304566324
        RLS
[--] Training time 0.000141143798828125 s
[--] Training error 0.010529865347268385
[--] Validation error: 0.009612262152373569
```



KNN error (Dataset 3)
Error
0.0
0.1
0.2
0.3
0.4
0
10
20
30
40
50
60
70
80
90
K
Validation error
Training error

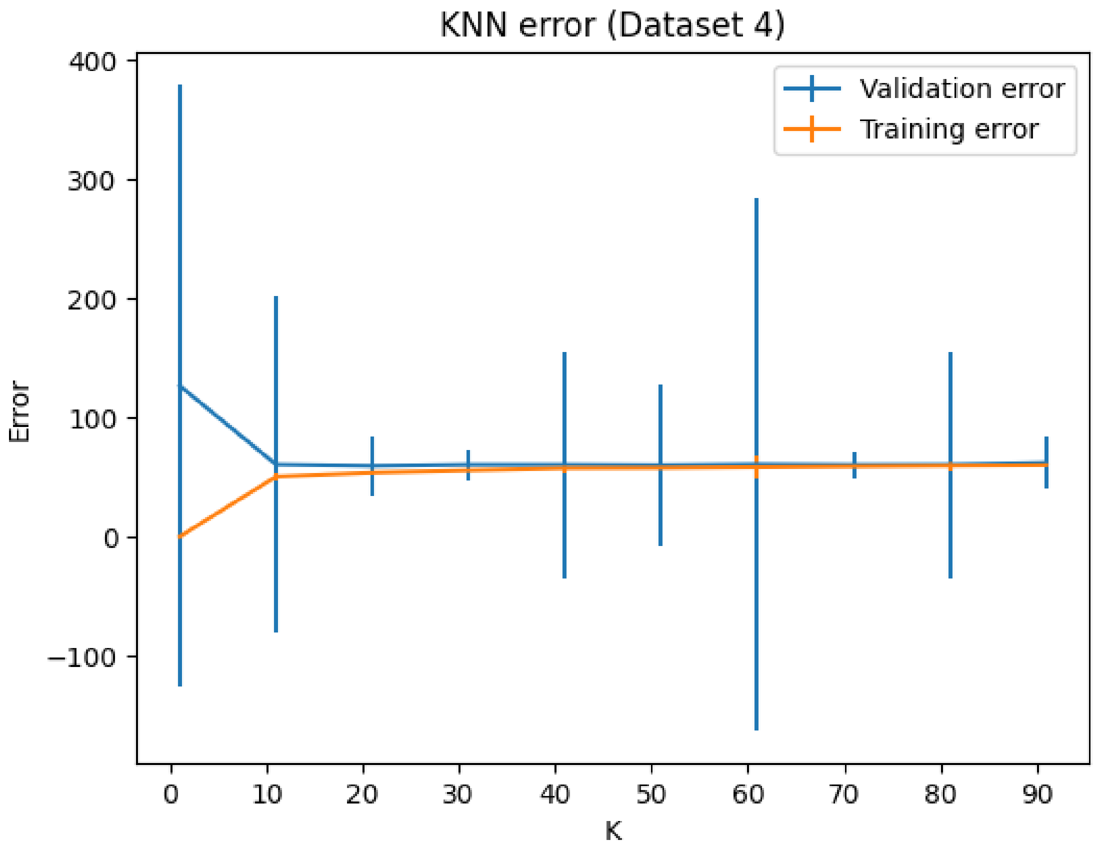


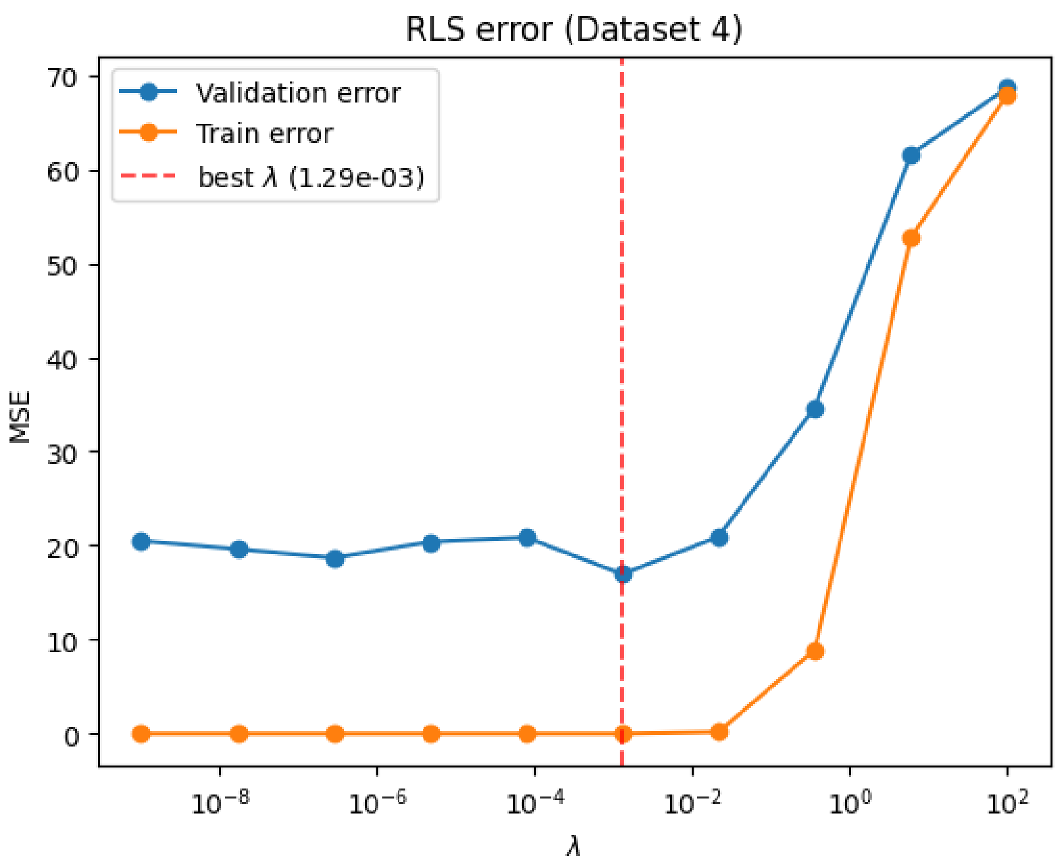
2.1 Task 1.2

2.1.1 Dataset 4

[54]:
```
analyze_dataset(Xtr_4, ytr_4, Xte_4, yte_4, 4)
```

```
Dataset 4:
        KNN
[--] Training time: 0.3524959087371826 s
[--] Training error: 52.92230913710432
[--] Test time: 0.07196784019470215 s
[--] Test error: 47.723560315247205
[--] Validation error: 59.390297609198555
        RLS
[--] Training time 0.004643201828002930 s
[--] Training error 13.590005161758738
[--] Validation error: 16.953315385564434
```

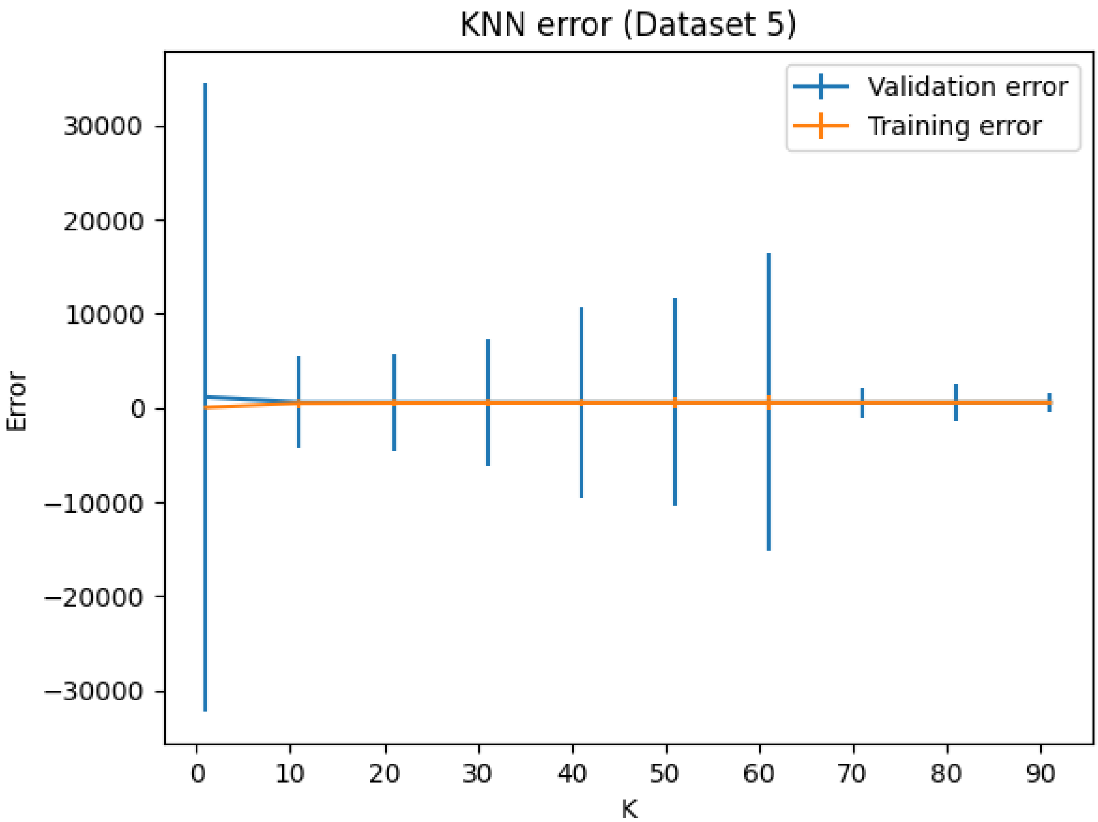

KNN error (Dataset 4)
Validation error
Training error
Error
K
400
300
200
100
0
−100
0
10
20
30
40
50
60
70
80
90

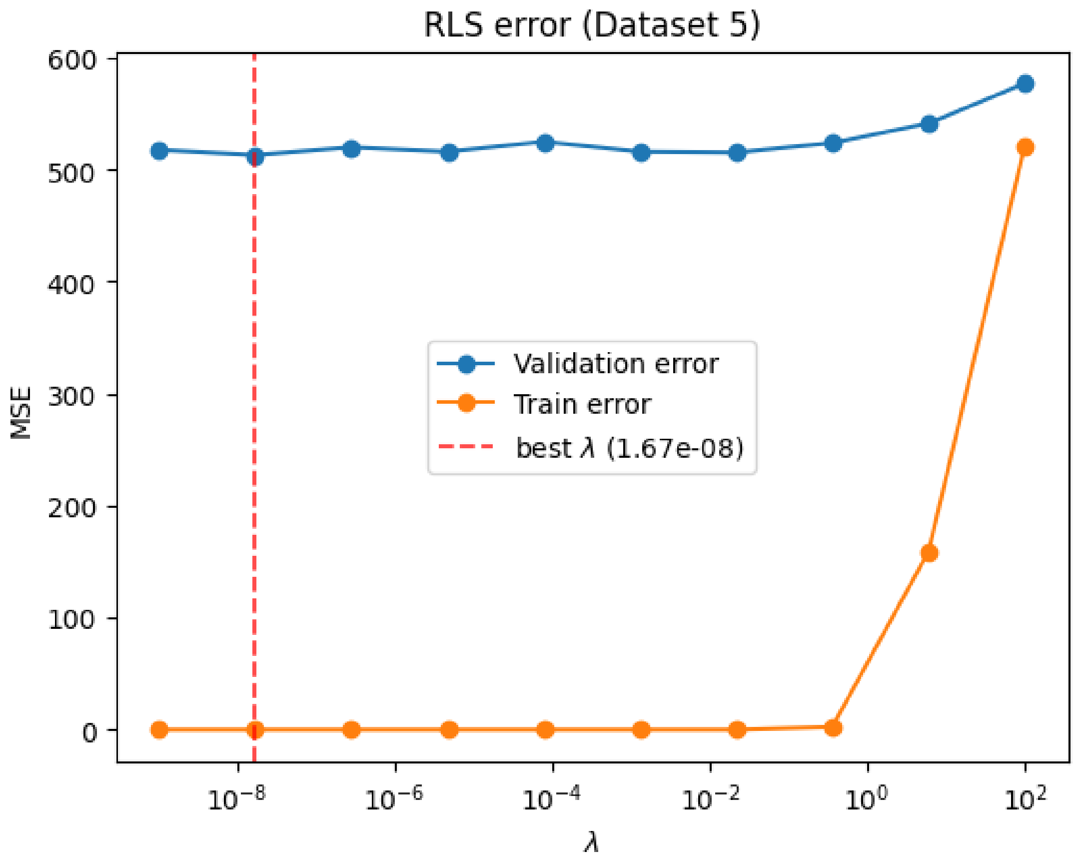


2.1.2 Dataset 5

[55]:
```
analyze_dataset(Xtr_5, ytr_5, Xte_5, yte_5, 5)
```

```
Dataset 5:
        KNN
[--] Training time: 0.5303466320037842 s
[--] Training error: 542.6013473552445
[--] Test time: 0.10509824752807617 s
[--] Test error: 508.85959609930507
[--] Validation error: 555.5617285913282
        RLS
[--] Training time 0.012933731079101562 s
[--] Training error 487.59238340490367
[--] Validation error: 513.1826620919869
```

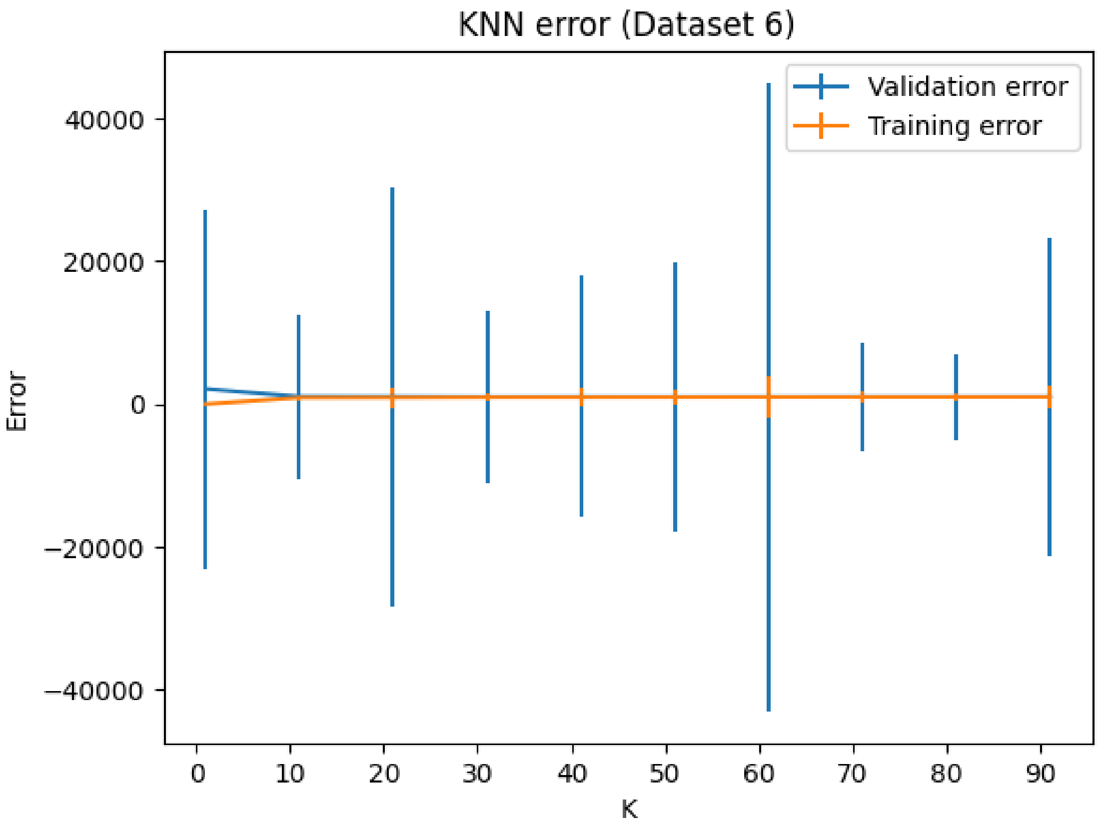

KNN error (Dataset 5)
Validation error
Training error
30000
20000
10000
0
−10000
−20000
−30000
Error
0
10
20
30
40
50
60
70
80
90
K

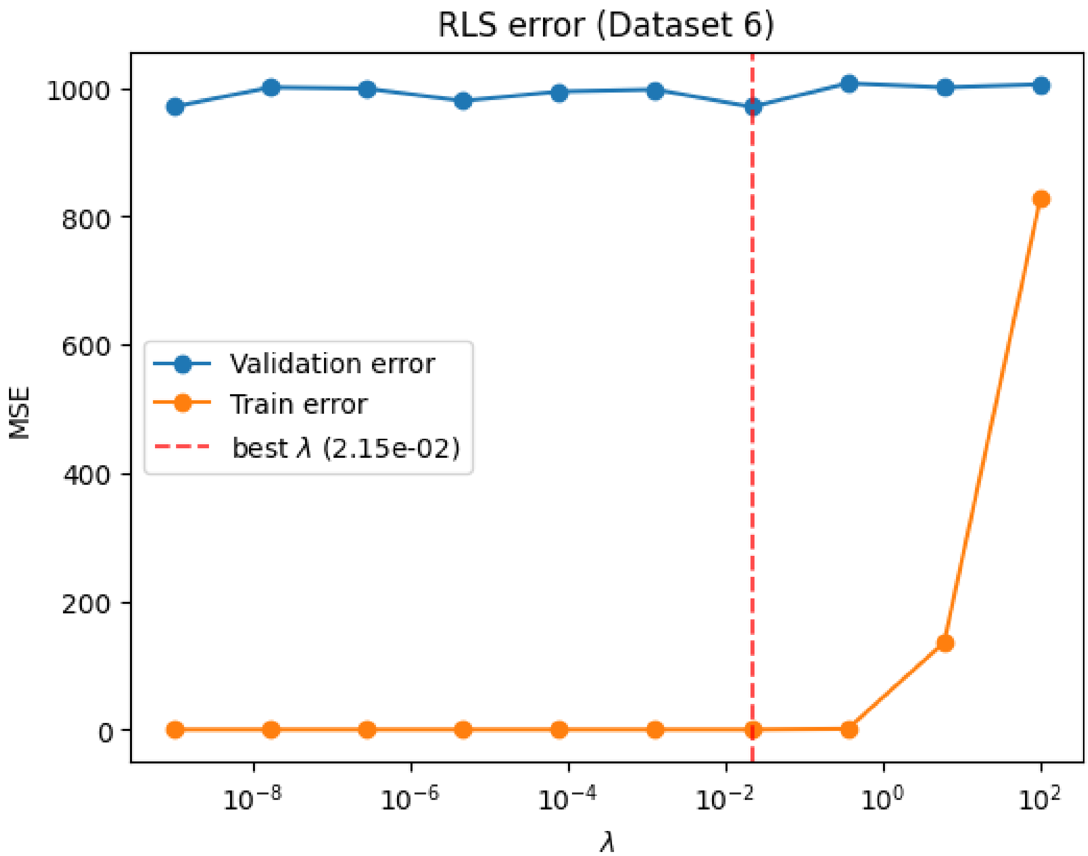


2.1.3 Dataset 6

[56]:
```
analyze_dataset(Xtr_6, ytr_6, Xte_6, yte_6, 6)
```

```
Dataset 6:
        KNN
[--] Training time: 0.7038910388946533 s
[--] Training error: 930.476349451124
[--] Test time: 0.16206693649291992 s
[--] Test error: 1032.8584618894429
[--] Validation error: 1000.755075685854
        RLS
[--] Training time 0.020161867141723633 s
[--] Training error 975.2638178969787
[--] Validation error: 970.8024337897762
```


KNN error (Dataset 6)
Validation error
Training error
40000
20000
0
−20000
−40000
Error
0
10
20
30
40
50
60
70
80
90
K



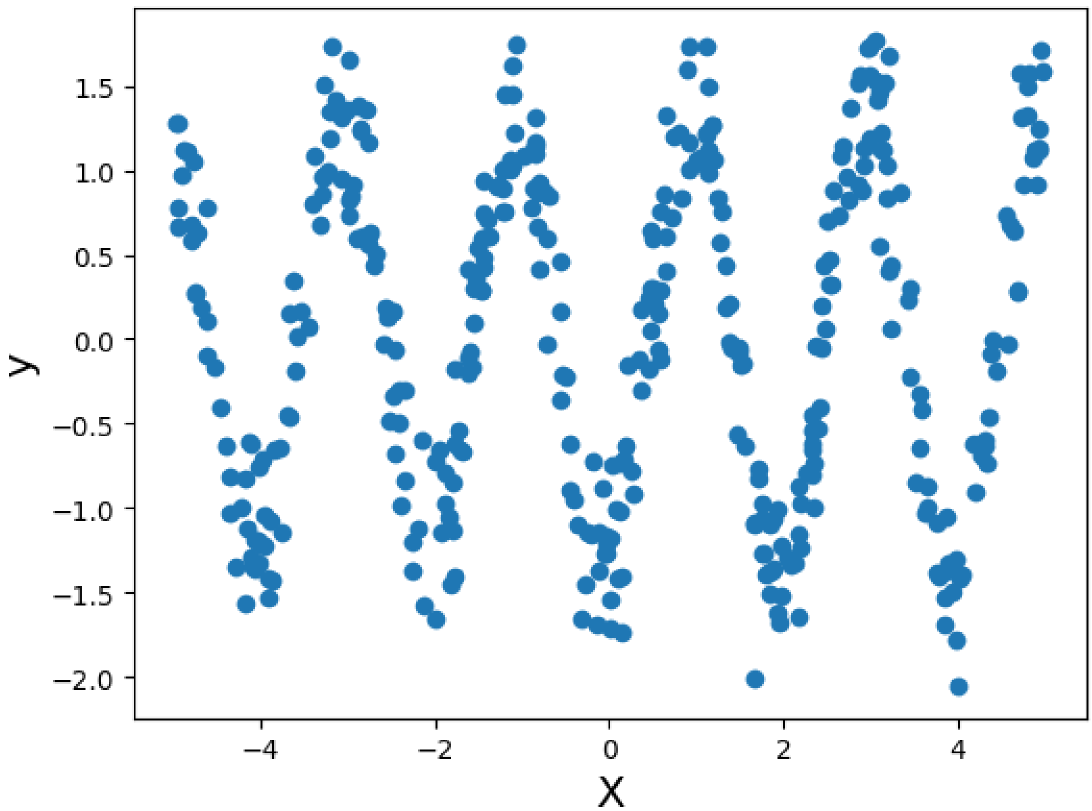
What can you observe? **Describe here your observations:**

2.1.4 Acitivity 2

Find your optimal solution for the dataset 7, we will evaluate the goodness of your model on the test set

[57]:
```
fig, ax = plt.subplots()
ax.plot(Xtr_7, ytr_7, 'o')
ax.set_xlabel("X", fontsize=16)
ax.set_ylabel("y", fontsize=16)
```

[57]:
```
Text(0, 0.5, 'y')
```



[58]:
```
# Insert your code here
```