Part A.I. Linear Regression

Our github repository for this assignment(Group11) :https://github.com/BrokenYouth0707/Machine_Learning_Task1

Dataset

x	x
27	442
26	442
20	457
31	431
14	471
16	463
25	434
7	487
6	480
10	480
15	470
31	430
6	489
28	436

26	444
16	464
28	447
24	430
11	475
24	445

Z-score Standardization

We used z-score standardization to normalize our data. Z-score standardization puts all features to the same scale by setting their mean to zero and standard deviation to one. By doing so, it preserves the original shape of the data, as the transformation only shifts and rescales the values without changing their distribution and internal relationships. This standardization is useful because many models—such as k-NN, linear regression and logistic regression—assume that the input data are centered and scaled. Moreover, it is easy to interpret: a z-score of 1 just indicates that a value lies one standard deviation above the mean. Z-score is defined as:

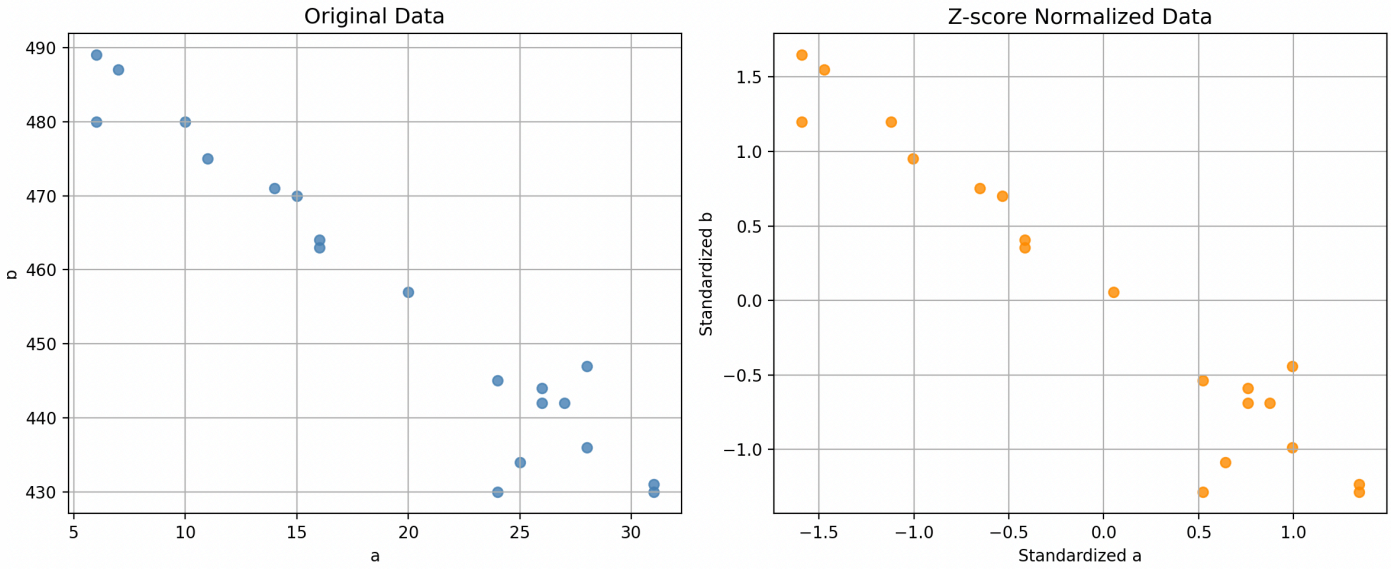
$$z_i = \frac{x_i - \mu}{\sigma}, \tag{1}$$

where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{2}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{3}$$

We comparted the data through pictures:



x	y
0.8744979860368924	-0.6881427304417103
0.7571157060319403	-0.6881427304417103
0.05282202600222832	0.057138205054725624
1.3440271060567004	-1.2346820831390966
-0.6514716540274837	0.7527337448513991
-0.4167070940175797	0.3552505792533
0.6397334260269883	-1.0856258960398095
-1.4731476140621478	1.5477000760475974
-1.5905298940670998	1.1999023061492606
-1.1210007740472918	1.1999023061492606
-0.5340893740225318	0.7030483491516367
1.3440271060567004	-1.284367478838859
-1.5905298940670998	1.6470708674471222
0.9918802660418444	-0.9862551046402847
0.7571157060319403	-0.5887719390421855

-0.4167070940175797	0.4049359749530624
0.9918802660418444	-0.4397157519428983
0.5223511460220364	-1.284367478838859
-1.0036184940423398	0.9514753276504487
0.5223511460220364	-0.5390865433424231

Least Square

For one demintional feature, we can simply use:

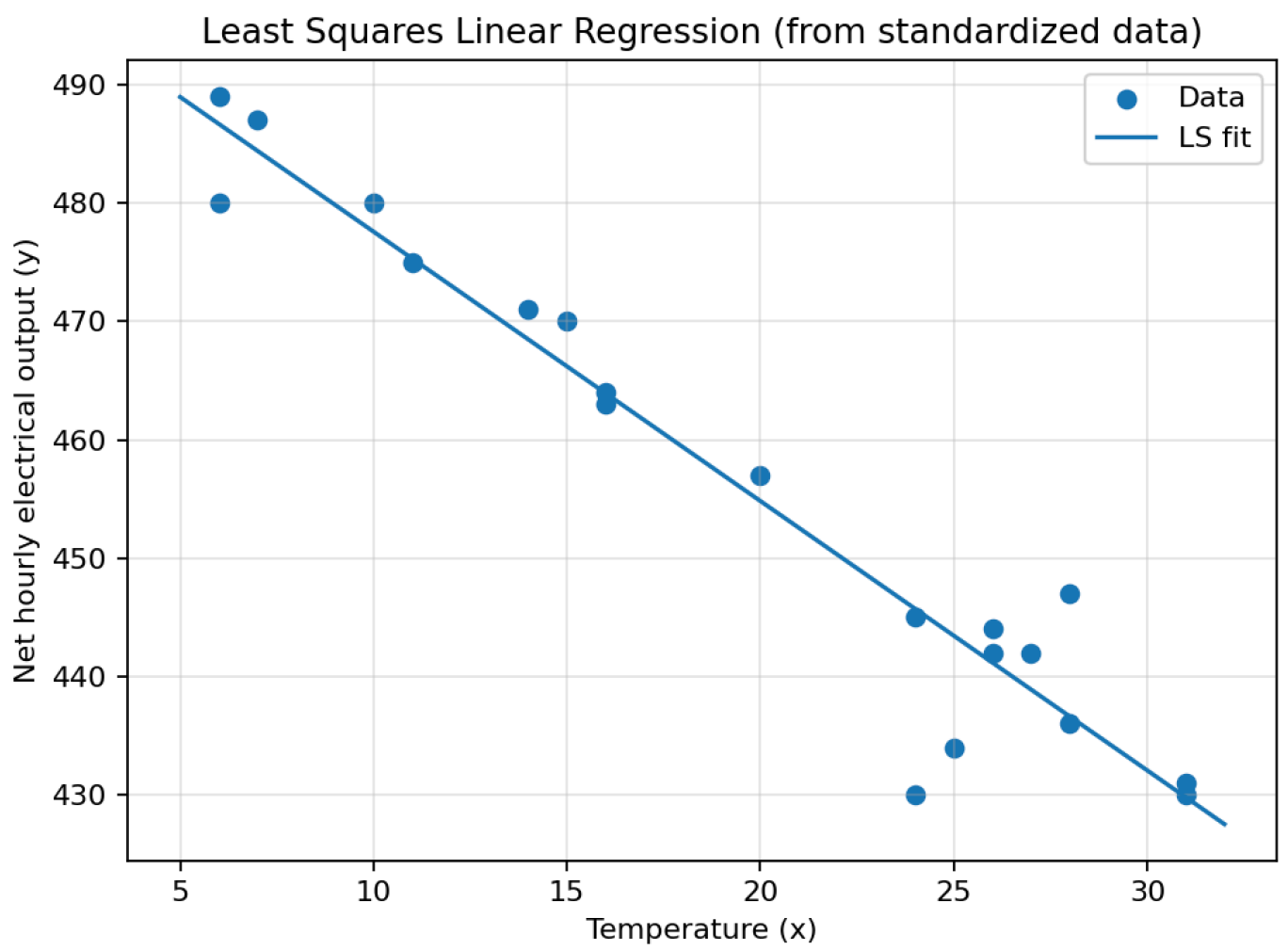
$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4}$$

to caculate its slope, and then using it to calculate the intercept:

$$b = \bar{y} - m\bar{x}. \tag{5}$$

Finially, we can get the equation of the line:

$$y = -2.275898x + 500.343812. \tag{6}$$



Linear regression with Gradient Descent

We define the hypothesis function as

$$h_\theta(x_i) = \theta_0 + \theta_1 x_i, \quad i = 1, \ldots, m \tag{7}$$

where m is the numbers of training samples.

We choose the **Mean Squared Error** as the cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{8}$$

Reason

MSE is smooth and differentiable, allowing gradient-based optimization and it measures the average squared difference between predicted and true values, and the square term penalizes large errors more strongly, helping the algorithm converge more stably.

Paritial Derivative

Partial derivative with respect to θ_0:

$$\begin{aligned} \frac{\partial J}{\partial \theta_0} &= \frac{1}{2m} \sum_{i=1}^{m} 2(h_\theta(x_i) - y_i) \frac{\partial h_\theta(x_i)}{\partial \theta_0} \\ &= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \end{aligned} \tag{9}$$

Partial derivative with respect to θ_1:

$$\begin{aligned} \frac{\partial J}{\partial \theta_1} &= \frac{1}{2m} \sum_{i=1}^{m} 2(h_\theta(x_i) - y_i) \frac{\partial h_\theta(x_i)}{\partial \theta_1} \\ &= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_i \end{aligned} \tag{10}$$

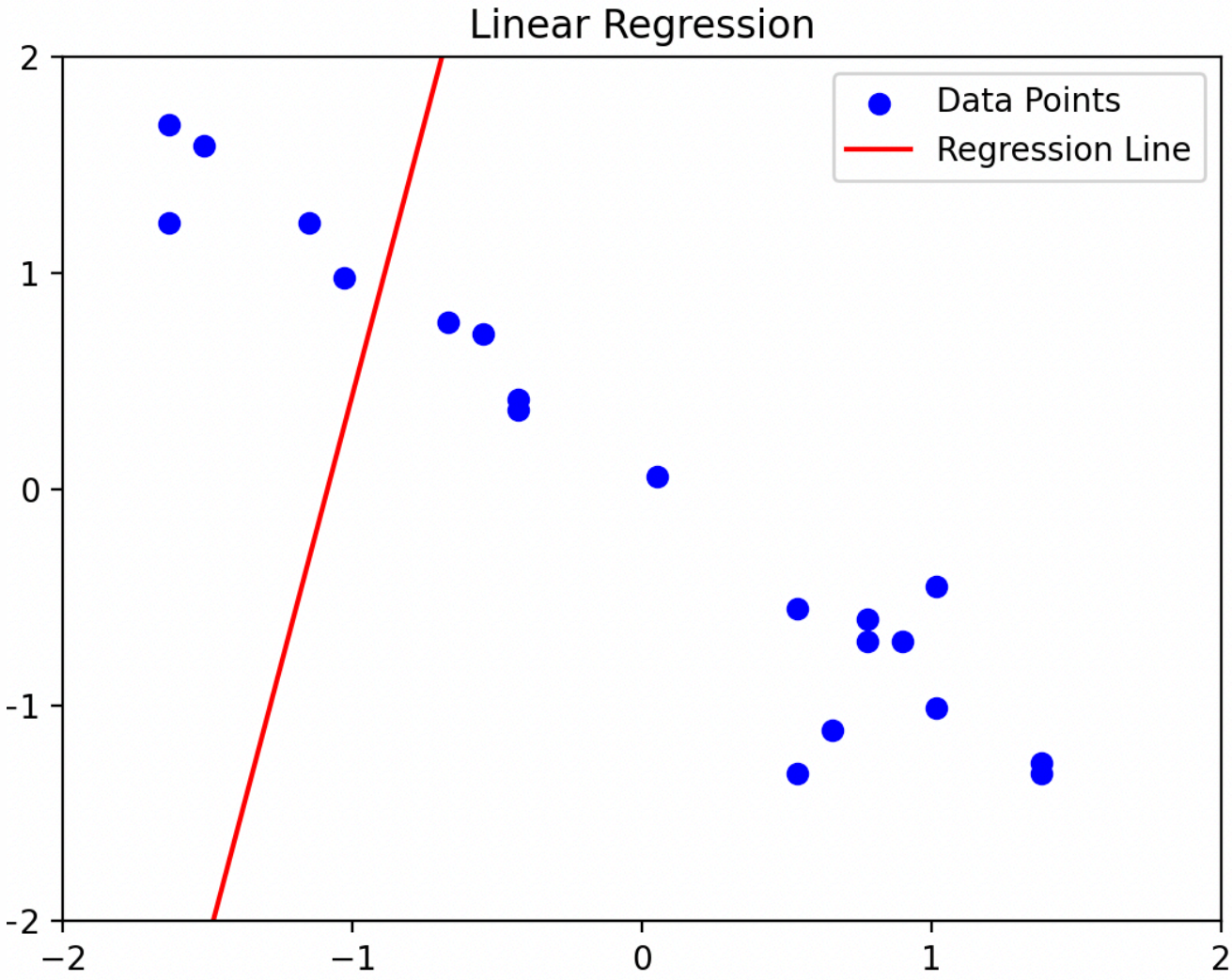
The gradient descent update rules are given by:

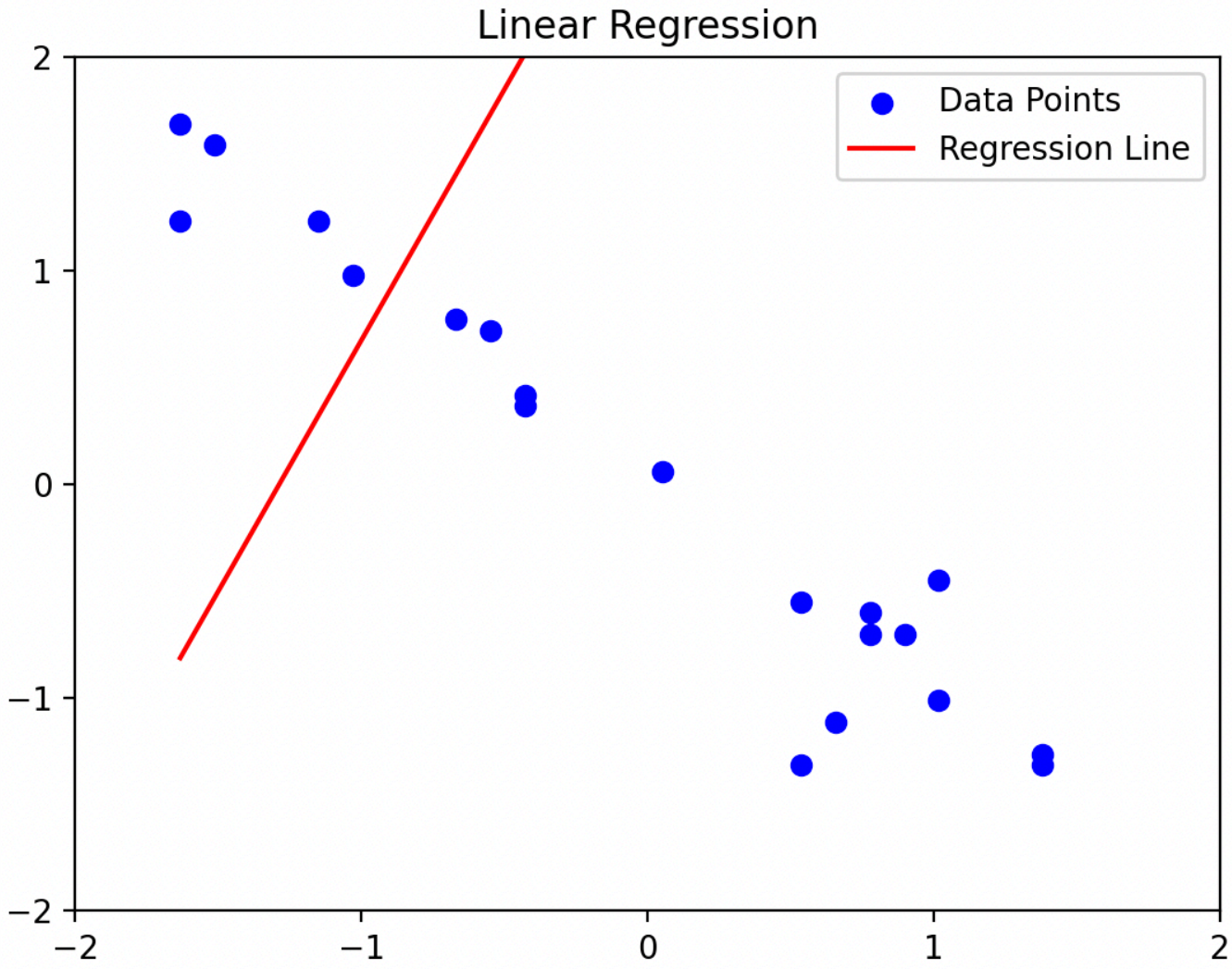
$$\theta_0 \leftarrow \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \tag{11}$$

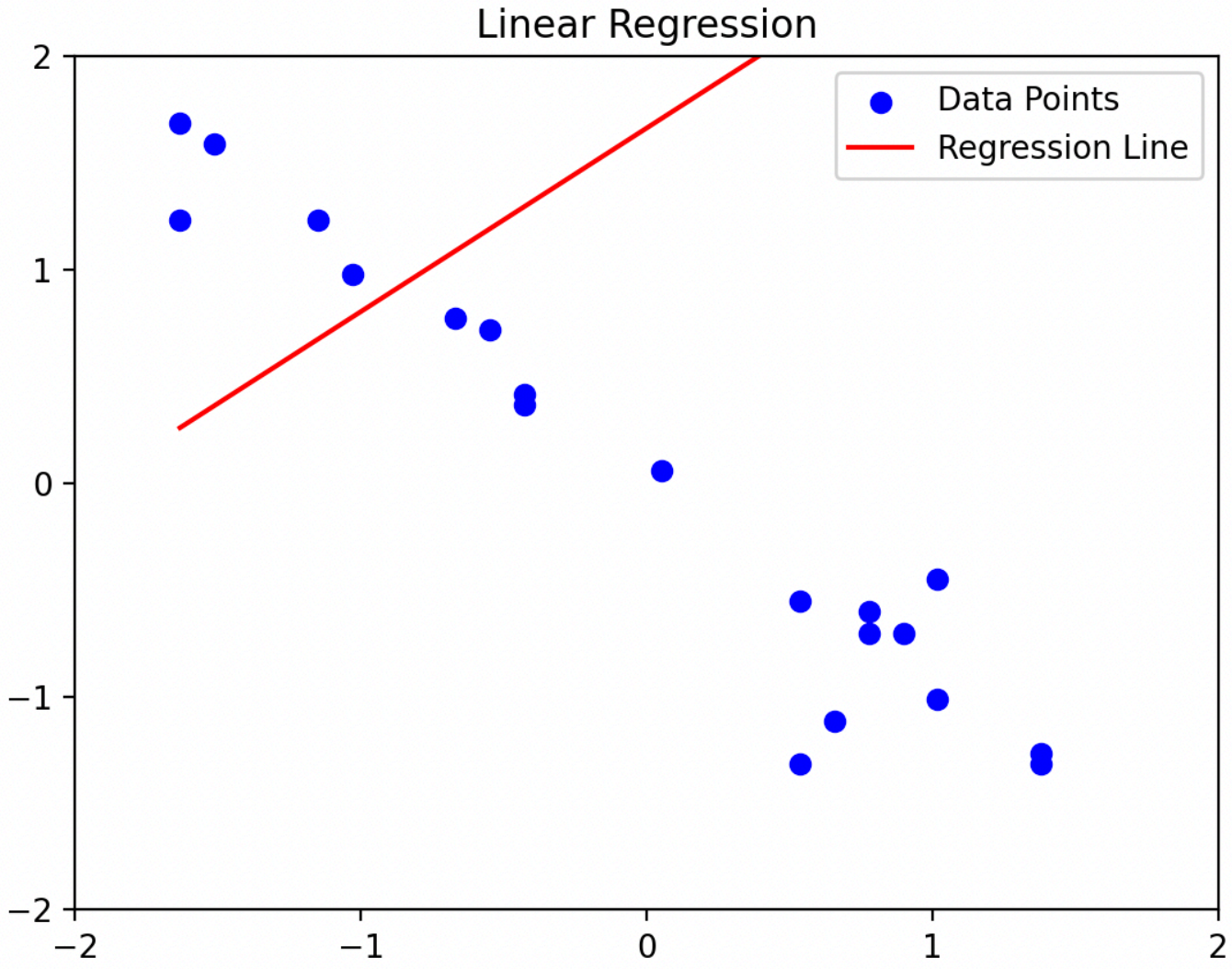
$$\theta_1 \leftarrow \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_i \tag{12}$$

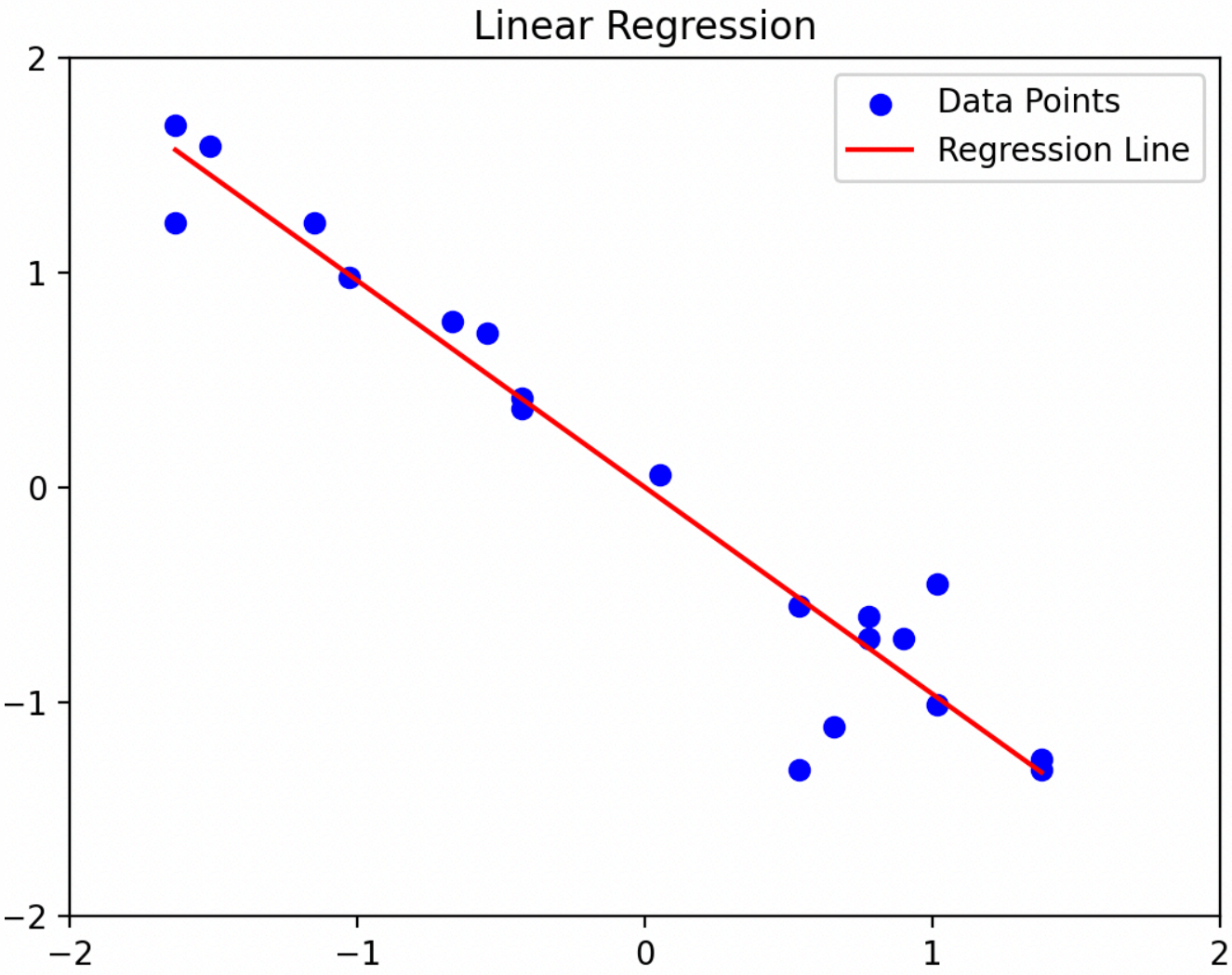
We started with $(\theta_0, \theta_1) = (10, 10)$ and a learning rate of 0.9; after the first, second, and third iterations, the parameters were updated to $(5.5, 5.07)$, $(3.03, 2.35)$, and $(1.66, 0.86)$ respectively, and finally converged to about $(0, -0.96)$.

Here are the regression lines after the first, second, and third iterations, as well as the final converged line:

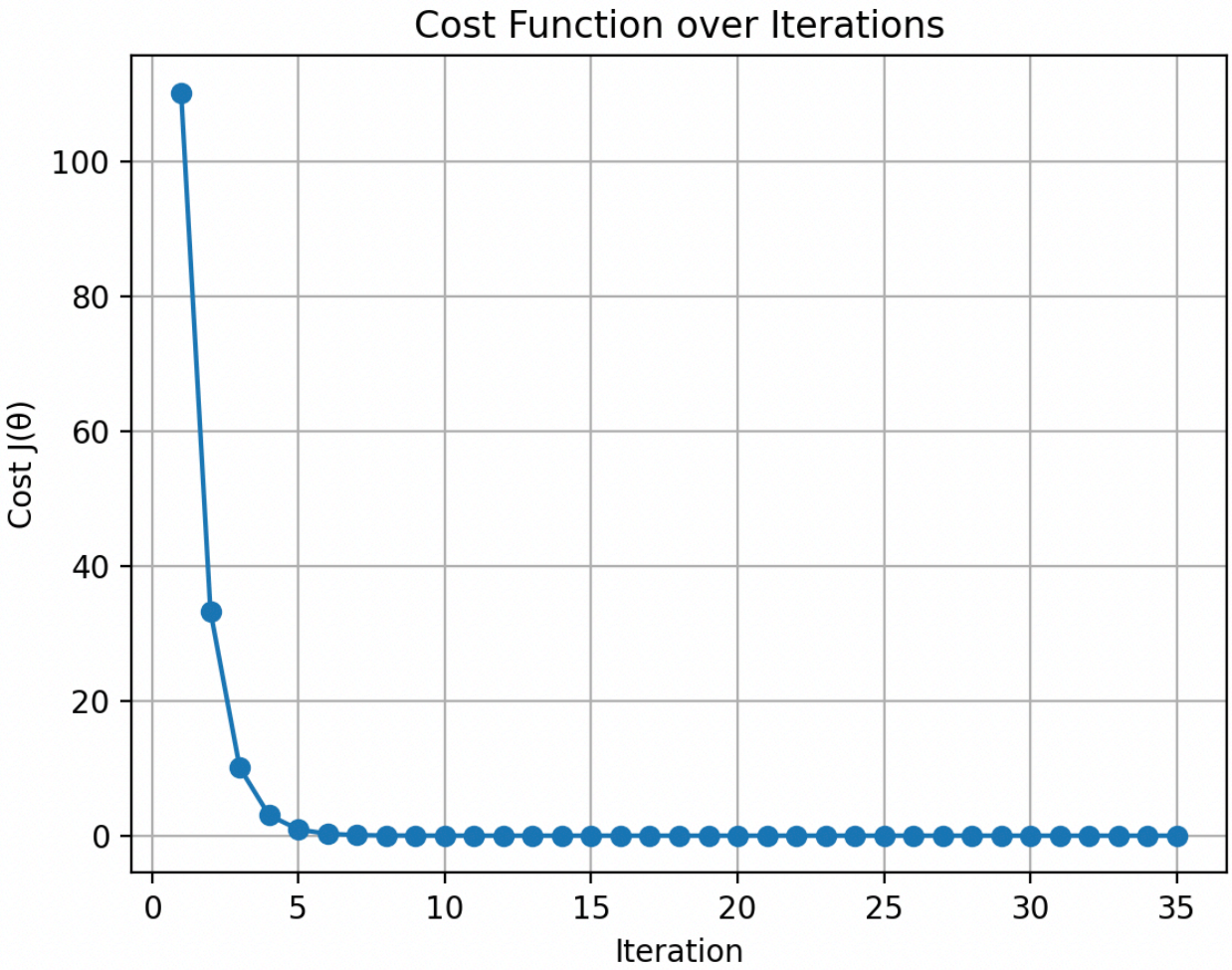

Linear Regression
Data Points
Regression Line
2
1
0
-1
-2
−2
−1
0
1
2


Linear Regression
Data Points
Regression Line
2
1
0
−1
−2
−2
−1
0
1
2


Linear Regression
Data Points
Regression Line
2
1
0
−1
−2
−2
−1
0
1
2



Here is the picture for the cost:



Comparision

Similarities:

Both methods aim to find the optimal parameters for a linear regression model.

Differences:

- Least Squares: Computes the analytical (closed-form) solution in one step without iteration. However, when the feature dimension is large, calculating the inverse matrix can be very slow. It generally has low bias but high variance.
- Gradient Descent: An iterative method that gradually approaches the optimal solution. It is more suitable for large-scale datasets or high-dimensional features. The results are affected by the choice of learning rate and initial values. It typically has low variance but high bias.

Performance:

Least Squares performs well on small to medium-sized datasets with relatively few features. It provide an exact solution efficiently. By contrast, Gradient Descent performs better on large-scale or high-dimensional datasets, in which computing the matrix inverse in Least Squares may be computationally expensive. However, Gradient Descent may require more time to converge and its performance depends on proper hyperparameter tuning.

PartA.II. Polynomial Regression

(1) Partial Derivatives of the Cost Function

Given the cost function:

$$J(\theta) = \frac{1}{4n} \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)}) \right)^4 \tag{13}$$

where the hypothesis function is:

$$h_\theta(x^{(i)}) = \theta_2 (x^{(i)})^2 + \theta_1 x^{(i)} + \theta_0 \tag{14}$$

We define the error term as:

$$\varepsilon^{(i)} = y^{(i)} - h_\theta(x^{(i)}) \tag{15}$$

Thus, the cost function can be rewritten as:

$$J(\theta) = \frac{1}{4n} \sum_{i=1}^{n} (\varepsilon^{(i)})^4 \tag{16}$$

Derivatives with respect to each parameter:

With respect to θ_2:

$$\frac{\partial J}{\partial \theta_2} = -\frac{1}{n} \sum_{i=1}^{n} (\varepsilon^{(i)})^3 (x^{(i)})^2 \tag{17}$$

With respect to θ_1:

$$\frac{\partial J}{\partial \theta_1} = -\frac{1}{n} \sum_{i=1}^{n} (\varepsilon^{(i)})^3 x^{(i)} \tag{18}$$

With respect to θ_0:

$$\frac{\partial J}{\partial \theta_0} = -\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 \tag{19}$$

(2) Parameter Update Rules

We use gradient descent to update the parameters.
The general update rule is:

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j} \tag{20}$$

where:

- α is the learning rate,
- $\frac{\partial J}{\partial \theta_j}$ is the gradient for parameter θ_j.

Specifically, in out problem, the update rules for each parameter are:

$$\theta_2 := \theta_2 - \alpha \left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 (x^{(i)})^2 \right) \tag{21}$$

$$\theta_1 := \theta_1 - \alpha \left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 x^{(i)} \right) \tag{22}$$

$$\theta_0 := \theta_0 - \alpha \left(-\frac{1}{n}\sum_{i=1}^{n}(\varepsilon^{(i)})^3 \right) \tag{23}$$

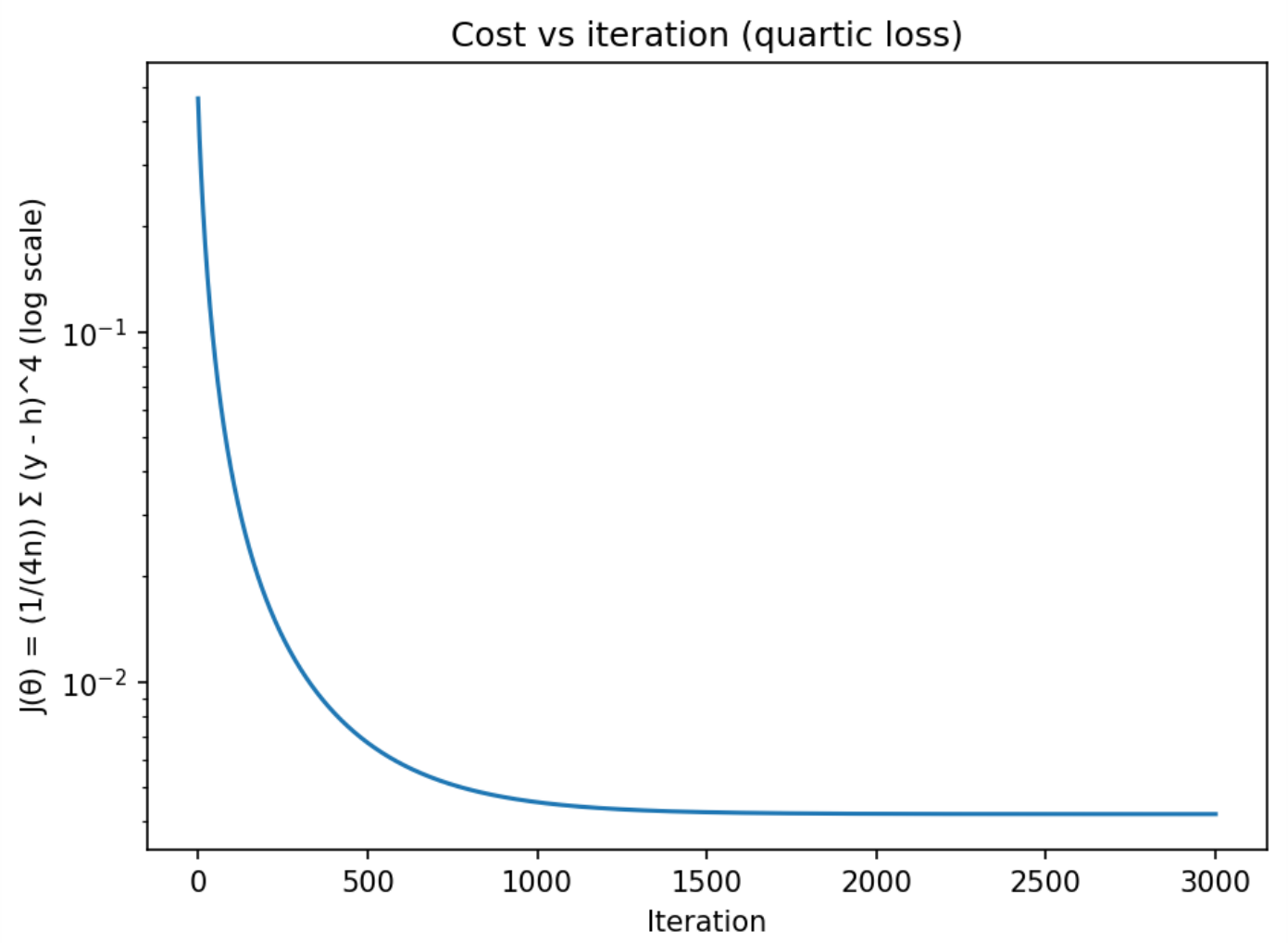
(3) Iterations

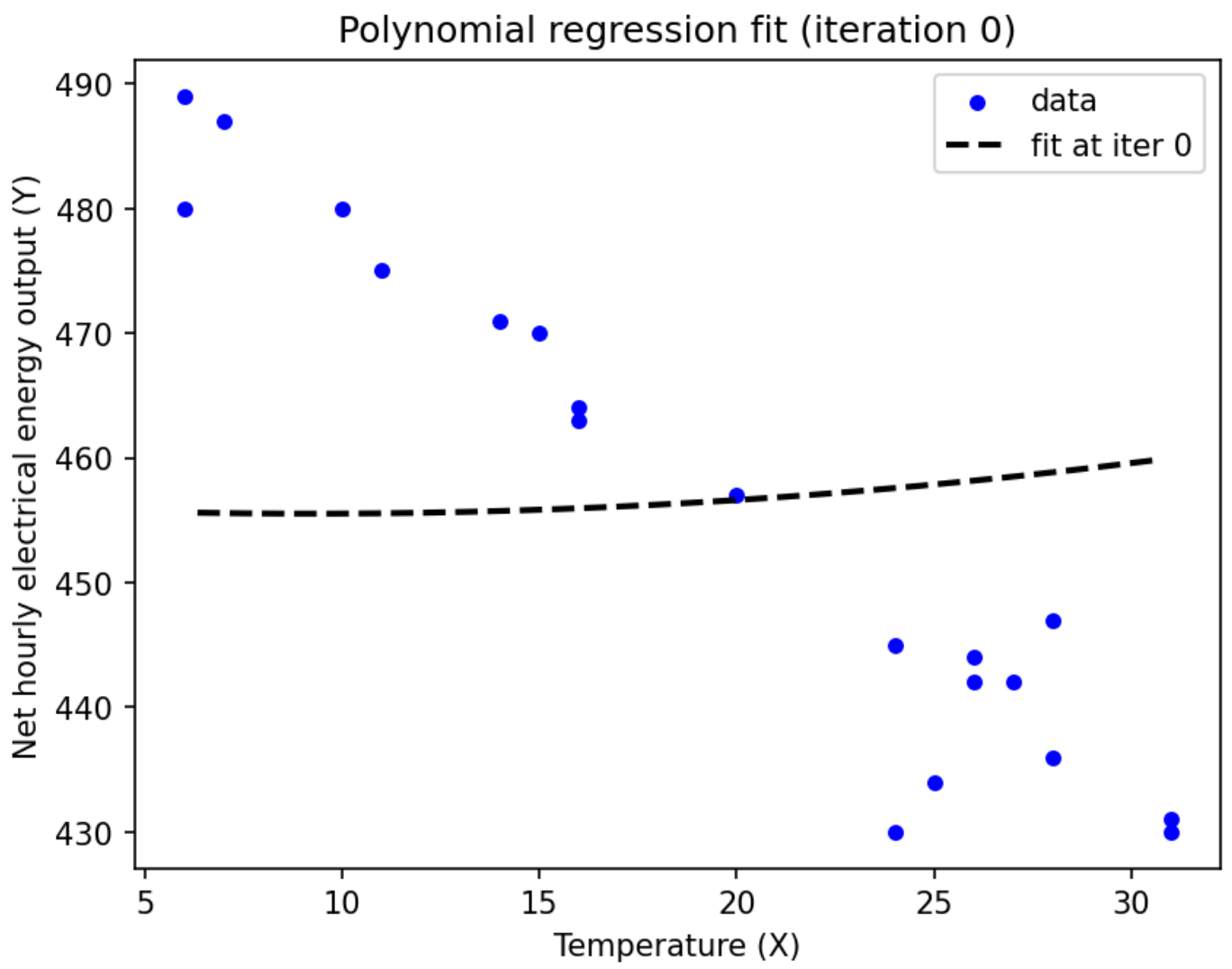
Parameter evolution and Cost during training(learning rate $\alpha = 0.01$):

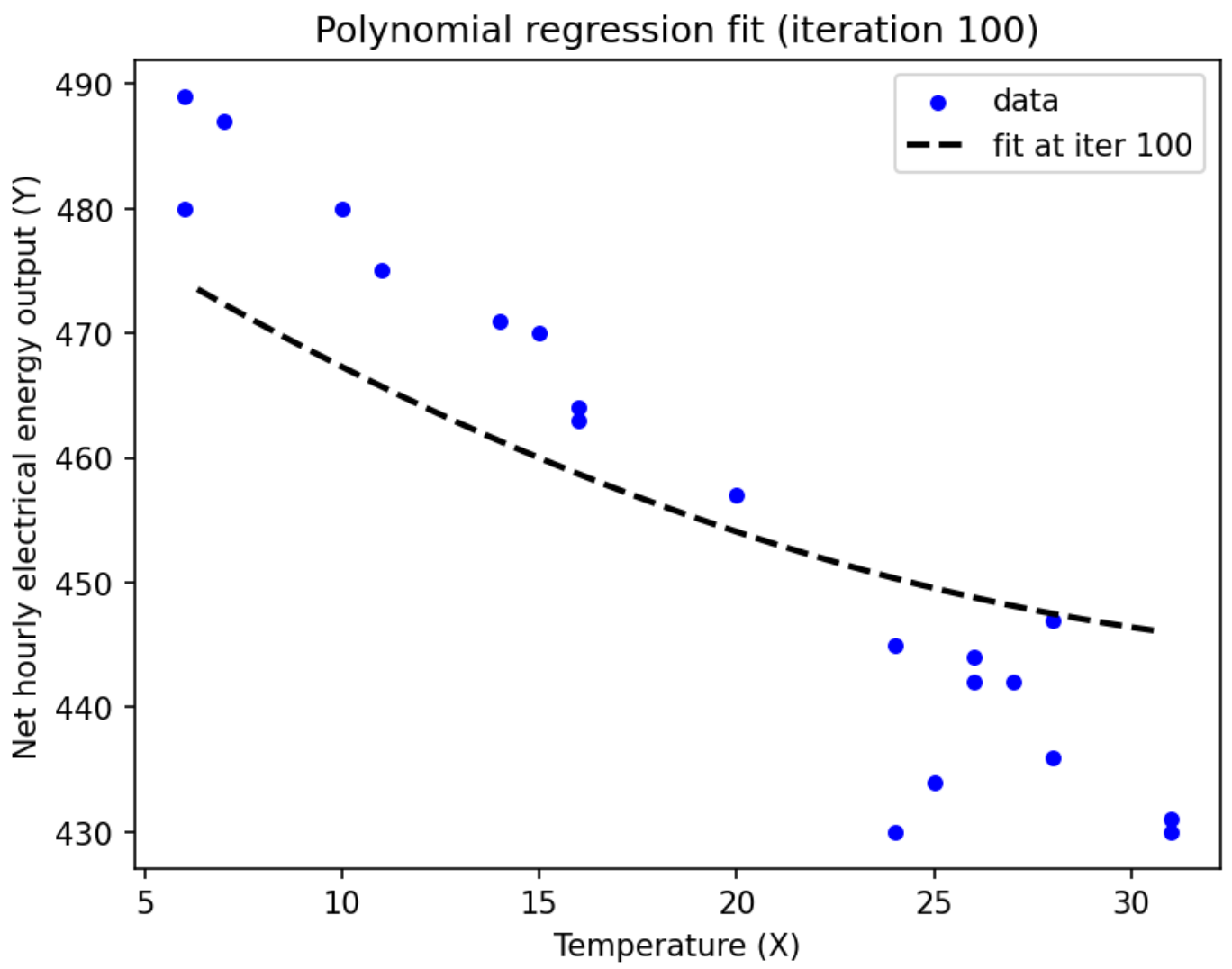
Iteration	θ0	θ1	θ2	Cost (J)
0	0.03455842	0.08216181	0.03304371	0.4643122
100	-0.06609397	-0.4526044	0.09791056	0.03924249
200	-0.1092821	-0.5906548	0.1053334	0.01732243
500	-0.1683796	-0.7493620	0.1199282	0.006708472
1000	-0.2047500	-0.8408934	0.1345493	0.004537533

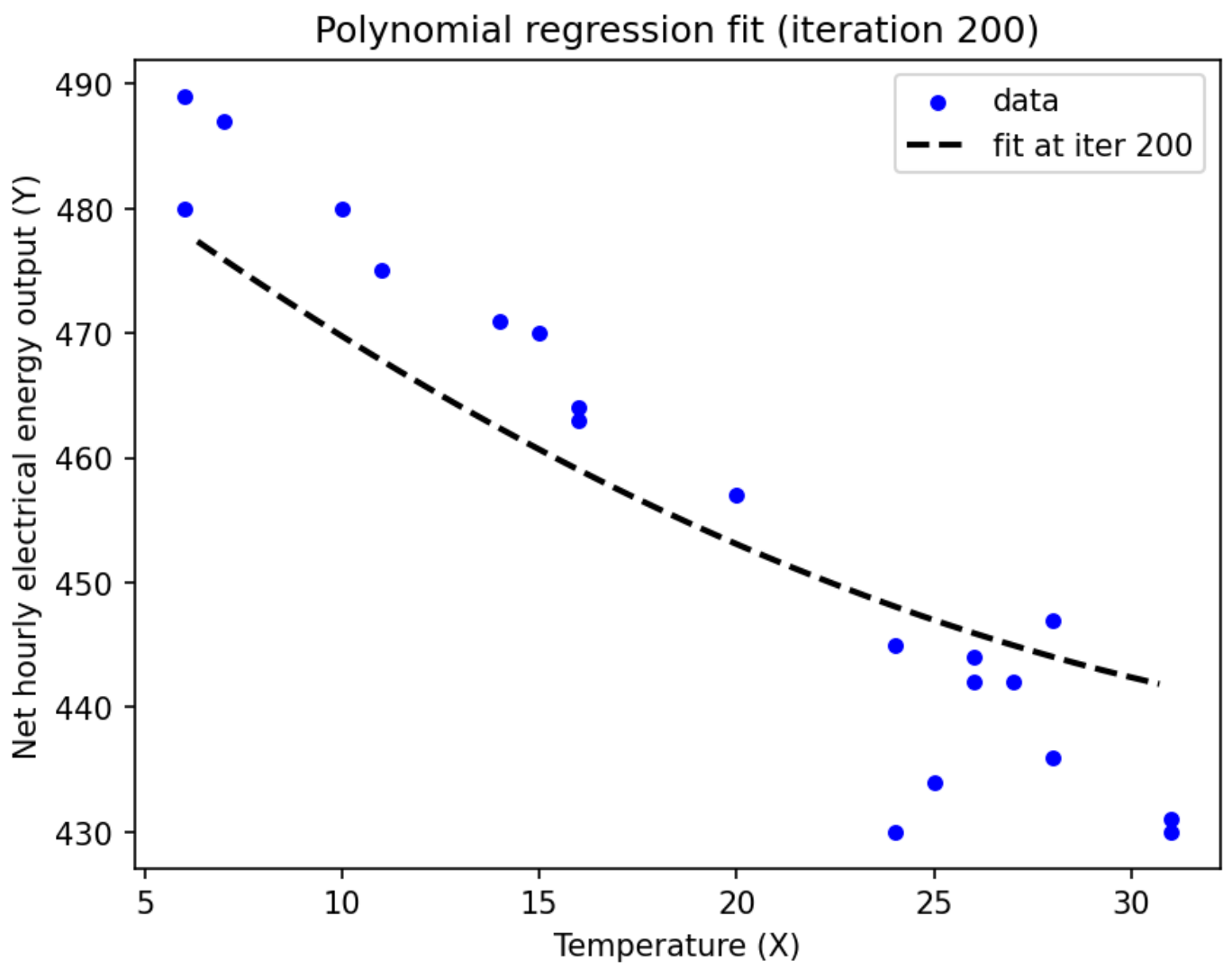
2000	-0.2221575	-0.8875152	0.1457154	0.004204270
3000	-0.2249141	-0.8943075	0.1478593	0.004196765

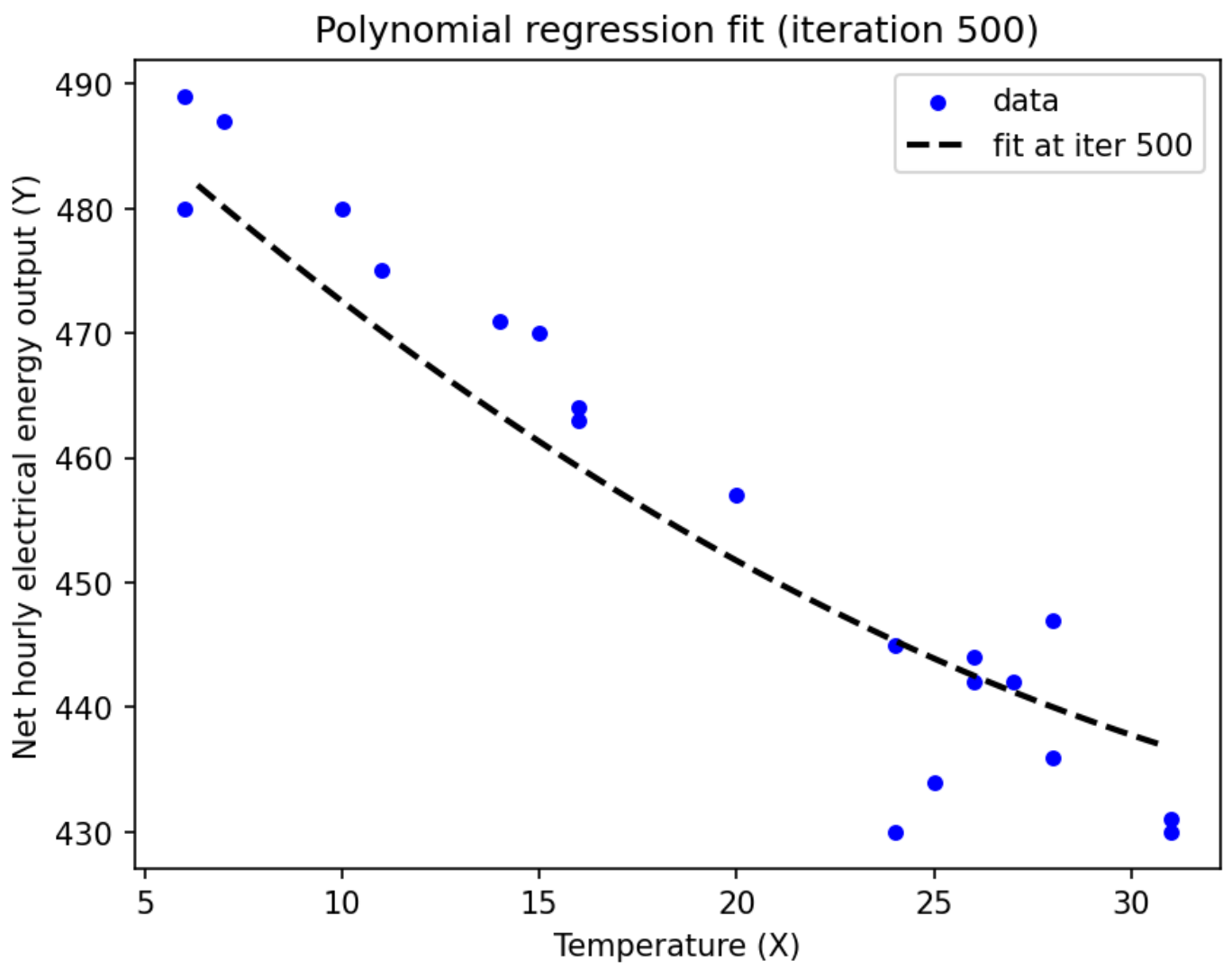
The first image illustrates the decrease of the cost function over iterations, while the others display the fitting results at different stages. It is obvious that the model becomes stable and shows good convergence after about 1,000 iterations.

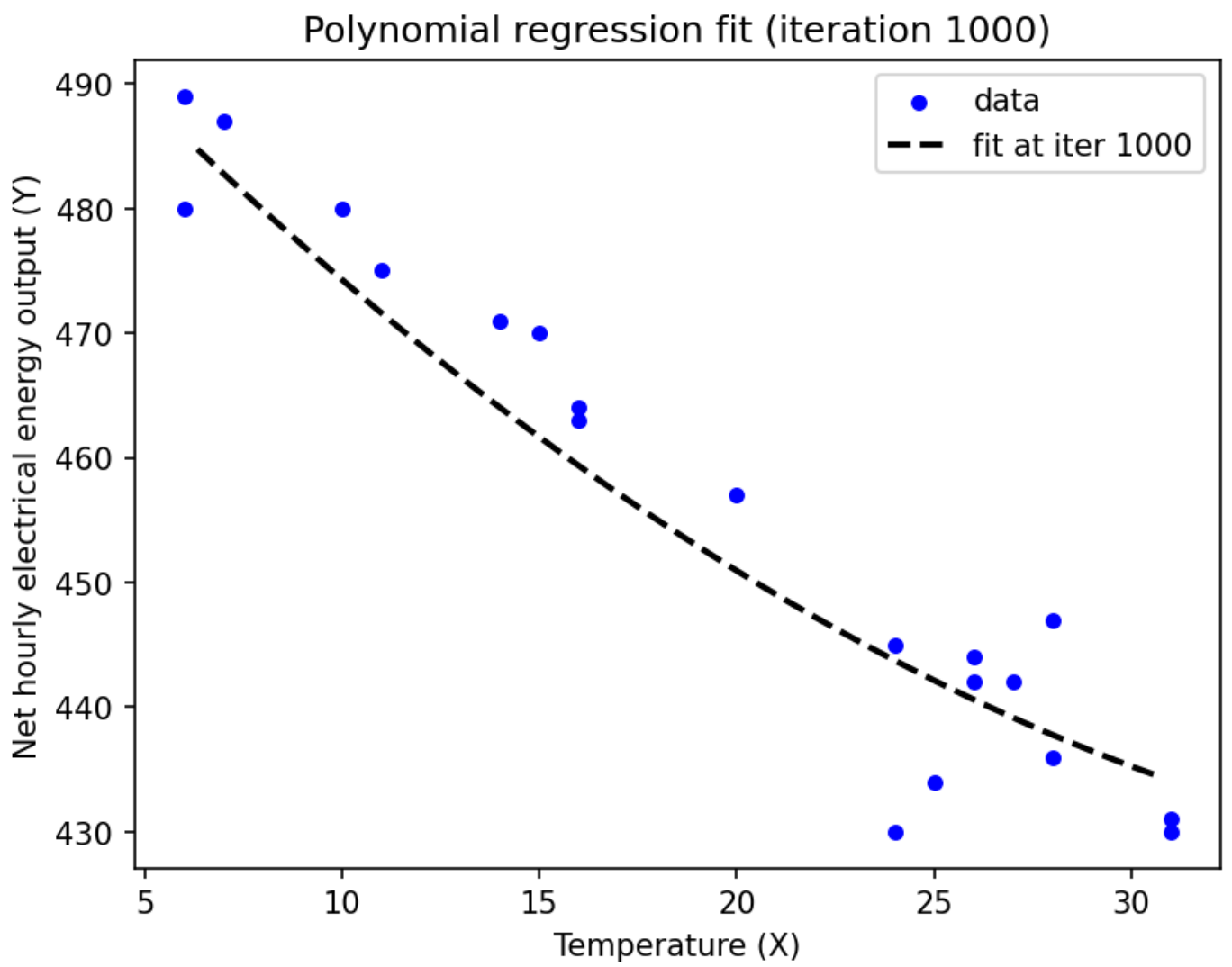


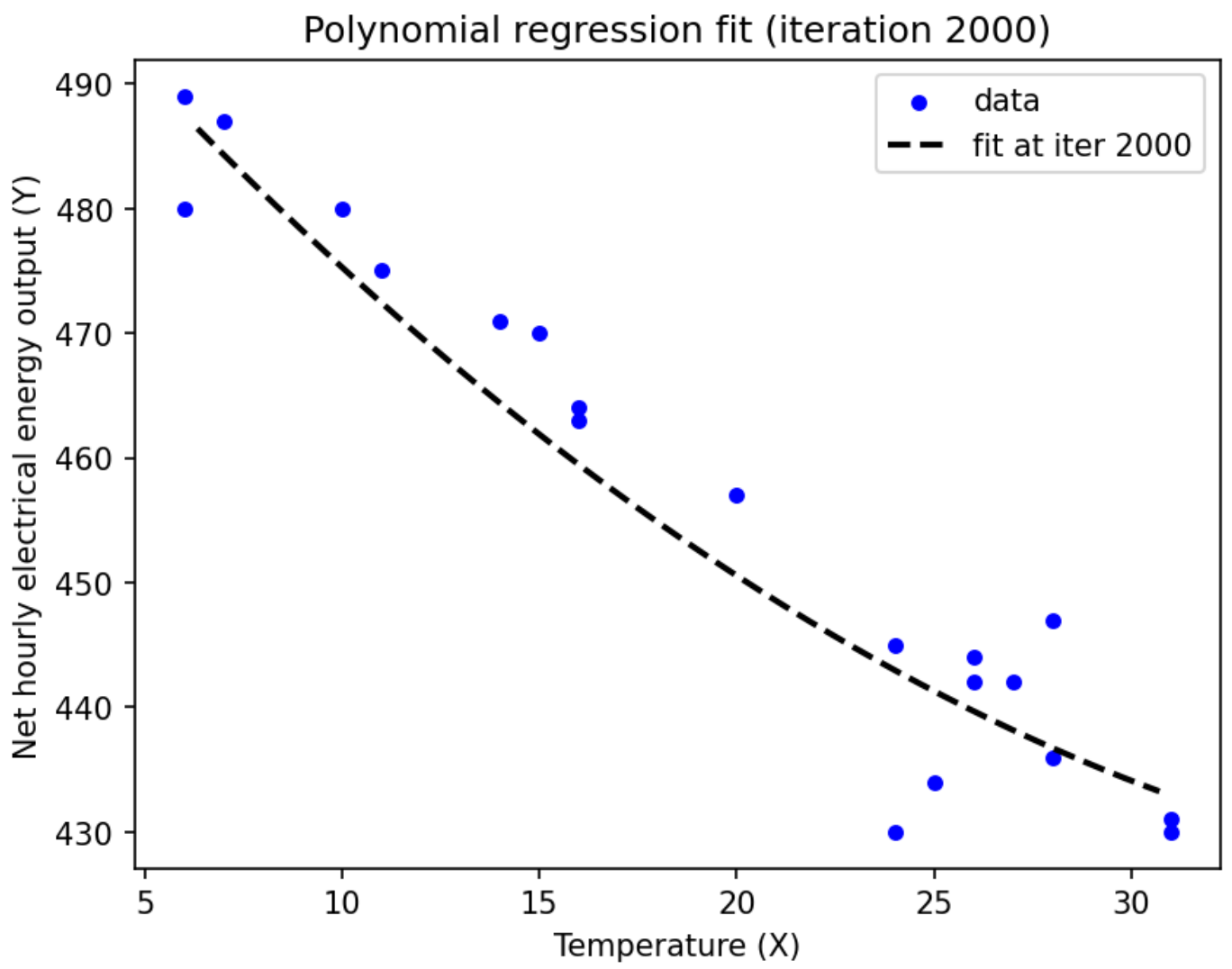

Polynomial regression fit (iteration 0)
data
fit at iter 0
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30

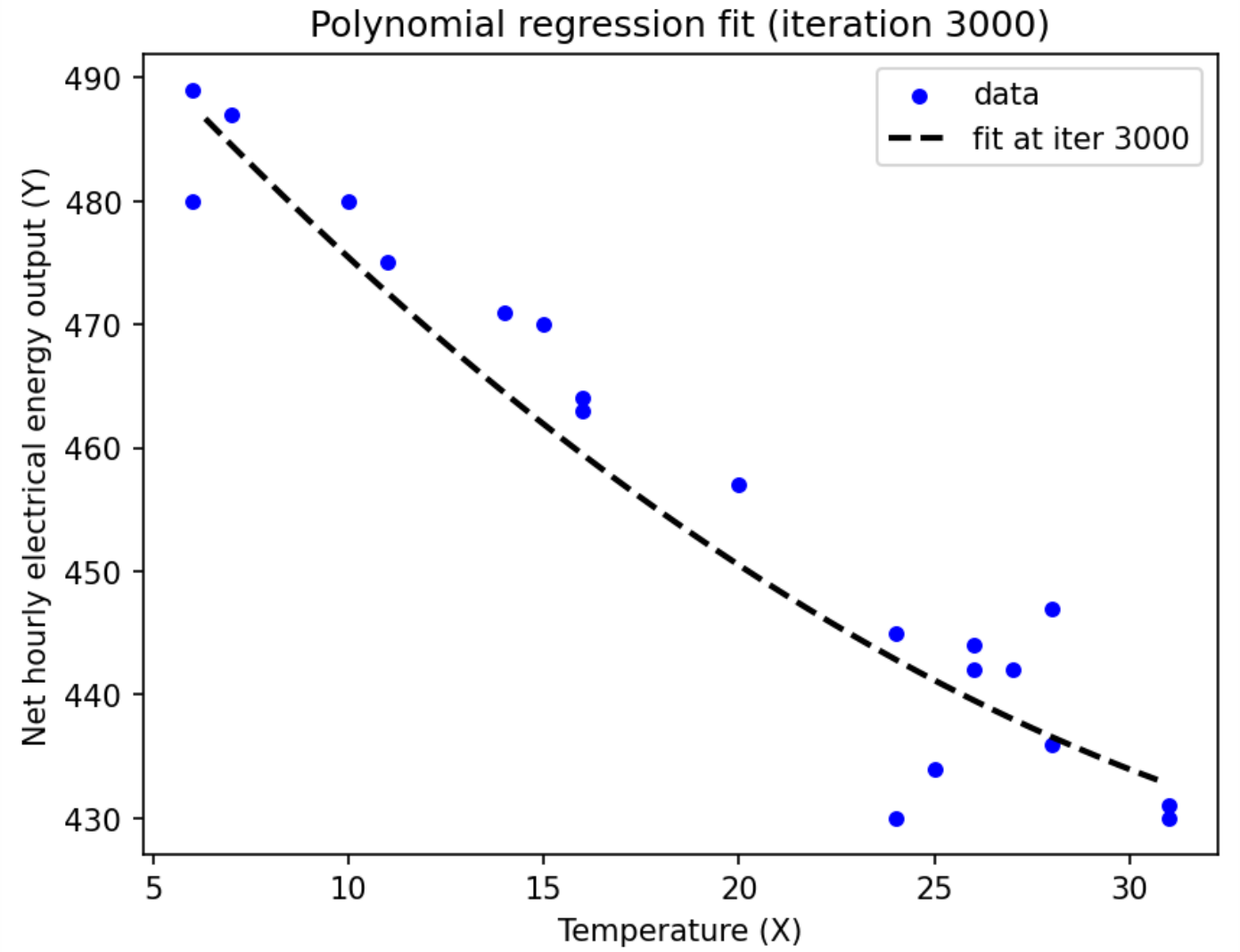

Polynomial regression fit (iteration 100)
data
fit at iter 100
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 200)
data
fit at iter 200
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 500)
data
fit at iter 500
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 1000)
data
fit at iter 1000
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30


Polynomial regression fit (iteration 2000)
data
fit at iter 2000
Net hourly electrical energy output (Y)
Temperature (X)
490
480
470
460
450
440
430
5
10
15
20
25
30



Similarities

The two are very consistent in their core principles and objectives: their goal is the same: regression algorithms within supervised learning. They aim to find a model (a line or a curve) that minimizes error on the training data. This error is typically measured by Mean Squared Error or the Sum of Squared Errors .

They have the same mathematical principle : The term "linear" in "linear regression" refers to linearity with respect to the model parameters (coefficients β), not necessarily linearity with respect to the features.

Univariate linear regression model:

$$y = \beta_0 + \beta_1 x \tag{24}$$

This model is linear in the parameters β_0 and β_1.

For Second-order polynomial regression model:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{25}$$

The model remains linear in the parameters $\beta_0, \beta_1, \beta_2$.

In addition, they have the same approach to get $h(\vec{x})$: Least Square and Gradient Descent.

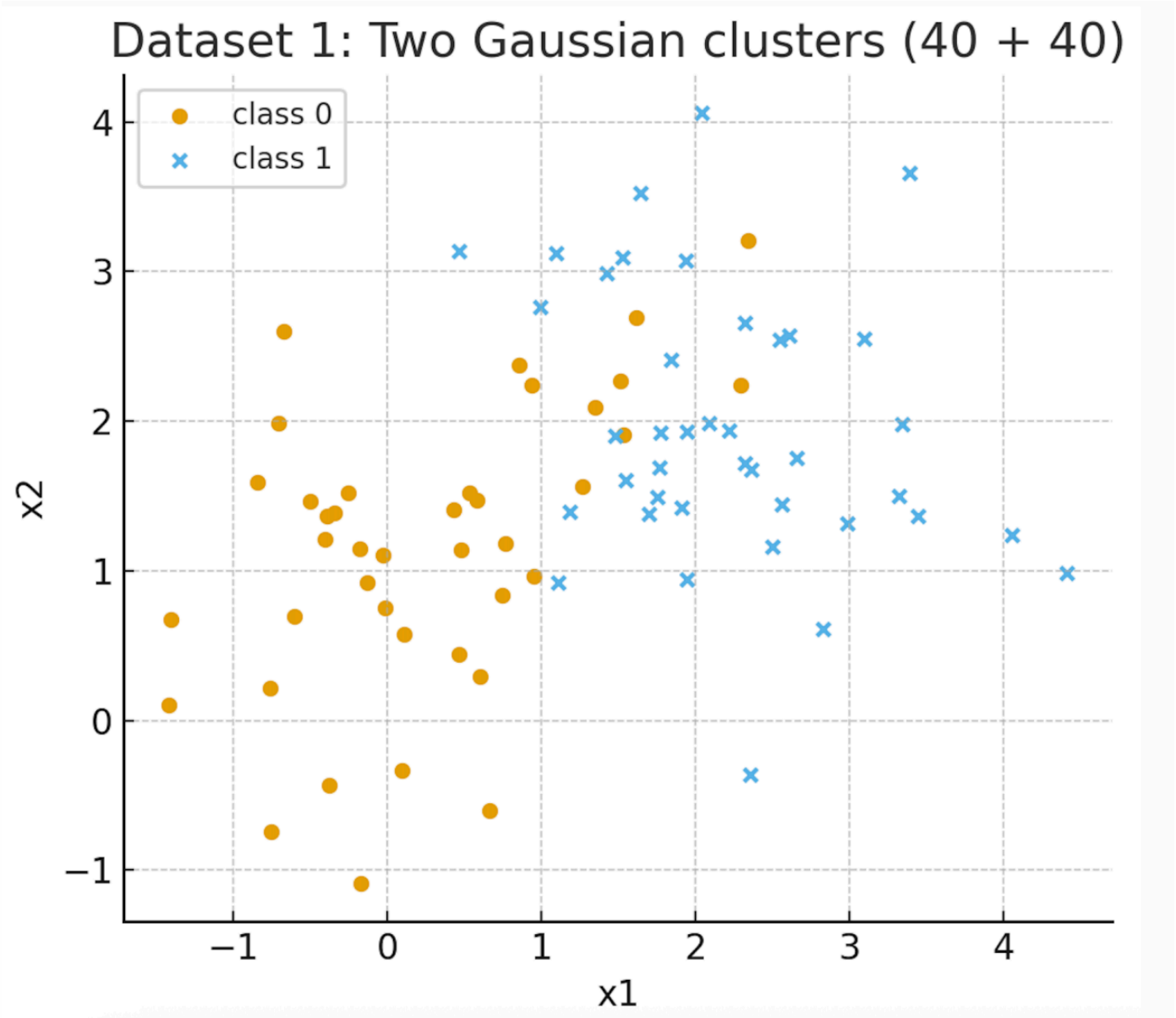
Differences

Views	**Linear Regression**	**Polynomial Regression**
Assumed Relationship	Assumes a linear relationship between features (X) and target (Y).	Assumes a non-linear relationship between features (X) and target (Y).
Model Form	$y = \beta_0 + \beta_1 x + \ldots \beta_n x_n$	$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$
Graphical Representation	A straight line.	A curve.
Model Complexity	Low. The model is simple.	High (depends on the degree n).
Bias	High Bias. It will underfit if the relationship is non-linear.	Low Bias. Can fit the training data better.
Variance	Low Variance. Not sensitive to small changes in training data.	High Variance. Prone to overfitting, especially with a high degree n.

Part.B

Dateset1

Our Dataset1 contains 80 samples, in which there are 40 samples with `y=0`, and 40 samples with `y=1`. It was generated through Gaussian clusters, and then stored them on the disk in the Numpy `.npy` format.



feature1	feature2	label
-0.1784637424485630	1.1442869652435100	0
0.7495904145465280	0.8352596500090170	0
0.9548472585837160	0.9631888877436280	0
-0.844555153601184	1.5936922615224200	0
0.7645416621333130	1.1807868743362400	0
-0.6060513743793620	0.6931615422735390	0

0.4679612044212450	0.44341087447755100	0
-0.38952443129654300	1.3673269879717600	0
1.3531627422439500	2.0961189147860400	0
2.2954452393098200	2.2375129860406800	0
1.6211827703905800	2.692674773323640	0
0.8584072176711300	2.375768390096750	0
-0.015044196592351700	0.7551524591177790	0
2.3445433562370900	3.2071710125134500	0
-0.028297874922390200	1.102774000298400	0
1.5146839634063200	2.268743328404230	0
1.2676392923050300	1.5662363932232000	0
-0.37367728045284100	-0.4342697042485070	0
-0.4992238039806130	1.4626238439138400	0
0.5358164393793030	1.5182463055524700	0
-0.12674976384362300	0.9226719075256420	0
0.9402238057078760	2.239144666991400	0
-0.17395691182432000	-1.086805174004270	0
-0.7622062694262290	0.21487420350894700	0
-0.6723670613360700	2.6036668507314100	0
0.0992803767020507	-0.33202666327168500	0
-0.40253307882607700	1.2077397337406500	0
-0.25383330388293200	1.5178275522556900	0
-0.3428805600759970	1.390229501683550	0
-0.7557852676418120	-0.7405821016333690	0
0.11182648751603800	0.575035566912016	0

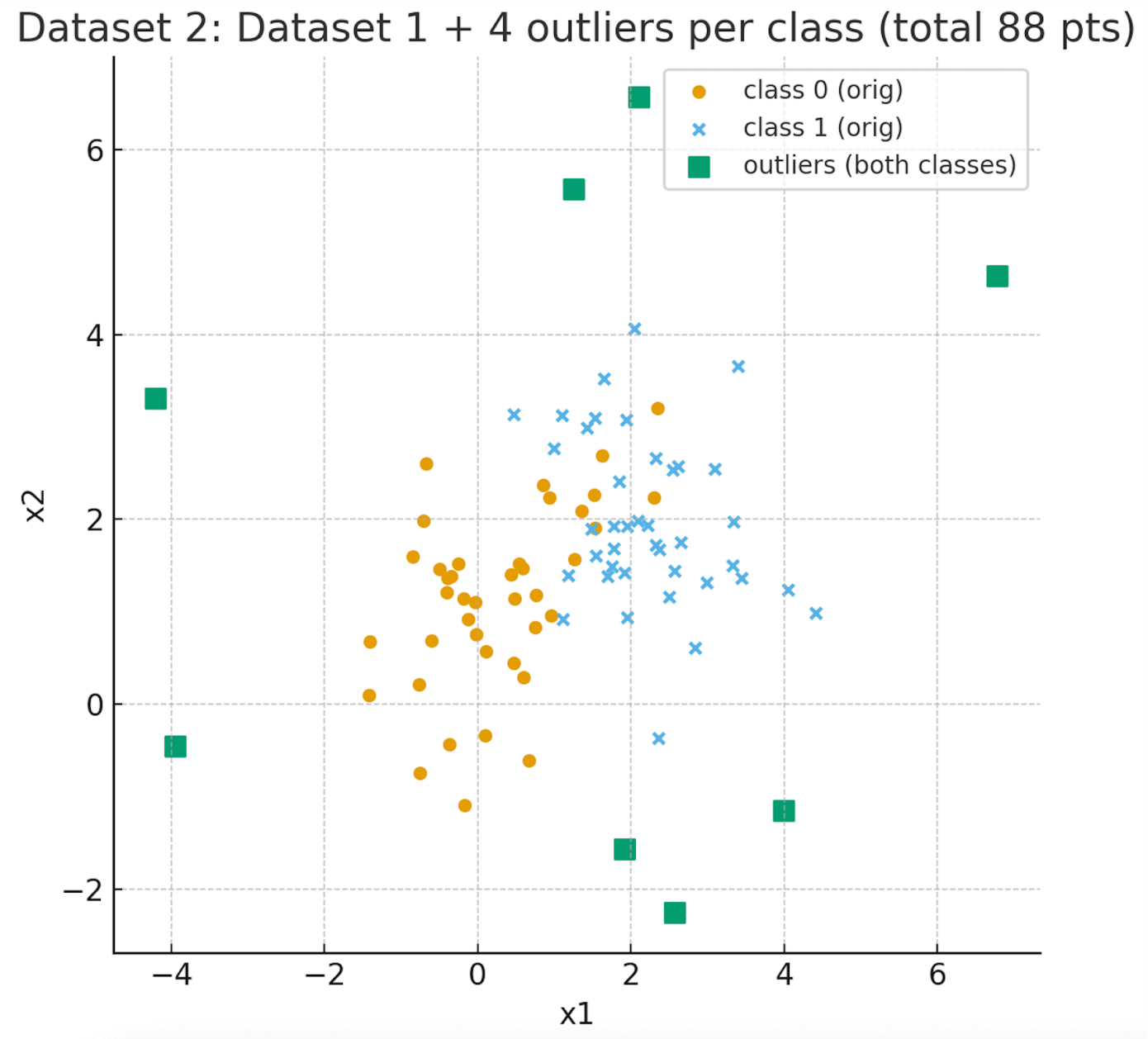
0.6028934873446810	0.2969368325952510	0
0.5825533001319400	1.4728781626219000	0
-1.4033171544297800	0.6758468701916590	0
1.5367375776874200	1.9116828788417700	0
0.6632460452284320	-0.6051033525085560	0
0.43037655679380700	1.4070964194832200	0
-1.4207150621212900	0.10232869920551200	0
0.4821740550021290	1.1415694756345600	0
-0.7038135814186100	1.9874836356974500	0
2.365425000359470	1.6752555184148400	1
1.8442848471742100	2.412817635111410	1
1.6999646192922200	1.3818533601854700	1
1.9468627139564000	1.9282849049632900	1
1.6462100988309400	3.5265346386500900	1
2.6148847490220600	2.575954342809810	1
3.0964579405066000	2.548434755164580	1
2.5494869201246000	2.540780390152160	1
3.4458500897709200	1.3643472759038100	1
2.356956830283130	-0.3651905149064520	1
1.9130727540939500	1.4221185407926000	1
3.3957523372572100	3.660279521929390	1
2.5014266626223600	1.1603133298665500	1
2.3260686410432800	2.657272552181530	1
2.2203930908157500	1.9359468564766400	1
1.9412377657722000	3.0739252463348600	1

2.986531798810 27	1.3197796015372600	1
1.5477093176871800	1.6046076514939800	1
1.4829371227873700	1.9019150833319300	1
1.5303247849070700	3.0977352544605200	1
1.7783417203563500	1.9223441017293200	1
1.1137436516352600	0.9224910573188100	1
3.326828784619190	1.5024890554777100	1
4.416965968383810	0.9847547432431060	1
4.055440519456520	1.239944551884510	1
3.342518840978810	1.9791958400798500	1
1.185219949178080	1.3963724267780000	1
2.0458295371032300	4.060021374084250	1
2.0910025593215600	1.9848953366663100	1
1.7519809742859800	1.4942032372672300	1
0.9931118402254740	2.764529763431160	1
1.7719505488366900	1.6894084885642800	1
1.9488438761179100	0.9393313362738170	1
1.0953588808686800	3.1223222437971200	1
1.4291320323571100	2.9866338433261900	1
2.5665764159532500	1.4468843716195500	1
2.6563813793534000	1.7530359538025100	1
2.325113718566920	1.7200141213939100	1
2.8341059548307400	0.611221069074144	1
0.46934924506403100	3.1364369028536300	1

Dataset2

We added four outliers for each type (`y=0`and `y=1`) based on dataset1, repsecitvely. Hence, there are 88 samples on dataset2. The outliers are ploted on the pircutre below with green sqaure.

feature1	feature2	label
2.565142301299080	-2.2456412656282300	0
-3.951047159005270	-0.44764262073455400	0
1.2461230358784400	5.573061064777870	0
-4.209301764620870	3.313574892076170	0
1.9117748740417800	-1.5620015479615600	1
2.0987181302286900	6.5653514457518900	1
6.78426287967574	4.630375747339830	1
3.9908467663642600	-1.1429863003365900	1



Spliting Data

The dataset Dataset1 and Dataset2 were split using function `train_test_split` form `sklearn.model_selection`. Three critical parameters were used in this function:

- test_size=0.3, which means 30% samples in a dataset were used as its own testset.
- `stratify=type`, which is important because it guarantees that 30% of the samples are proportionally selected from both classes and used as the test set.
- random_state=42, which controls the randomness and ensures the reproductility of the traning process.

The split data of Dataset1

The table below is the test set of Dataset1:

feature1	feature2	label
0.6632460452284320	-0.6051033525085560	0
4.055440519456520	1.239944551884510	1
-0.3428805600759970	1.390229501683550	0
2.045829537103230	4.060021374084250	1
-0.1267497638436230	0.9226719075256420	0
-0.1739569118243200	-1.086805174004270	0
0.4693492450640310	3.1364369028536300	1
1.9468627139564	1.928284904963290	1
0.5825533001319400	1.4728781626219000	0
1.5477093176871800	1.6046076514939800	1
2.3260686410432800	2.657272552181530	1
2.8341059548307400	0.611221069074144	1
0.4821740550021290	1.1415694756345600	0
1.7783417203563500	1.9223441017293200	1
-0.0282978749223901	1.102774000298400	0
3.342518840978810	1.979195840079850	1
1.8442848471742100	2.412817635111410	1
-0.3895244312965430	1.3673269879717600	0
-0.4992238039806130	1.4626238439138400	0
3.445850089770920	1.3643472759038100	1
1.6211827703905800	2.692674773323640	0

-0.844555153601184	1.5936922615224200	0
-0.7622062694262290	0.2148742035089470	0
1.5303247849070700	3.0977352544605200	1

The table below is the training set of Dataset1:

feature1	feature2	label
2.98653179881027	1.3197796015372600	1
-0.402533078826077	1.207739733740650	0
1.6462100988309400	3.526534638650090	1
2.365425000359470	1.6752555184148400	1
1.113743651635260	0.9224910573188100	1
2.344543356237090	3.2071710125134500	0
2.0910025593215600	1.9848953366663100	1
-0.0150441965923516	0.7551524591177790	0
2.6148847490220600	2.575954342809810	1
2.2203930908157500	1.935946856476640	1
1.948843876117910	0.9393313362738170	1
2.325113718566920	1.7200141213939100	1
2.5494869201246000	2.540780390152160	1
-1.4033171544297800	0.6758468701916590	0
1.771950548836690	1.6894084885642800	1
1.2676392923050300	1.5662363932232000	0
2.2954452393098200	2.2375129860406800	0

0.4303765567938070	1.4070964194832200	0
1.4291320323571100	2.9866338433261900	1
0.8584072176711300	2.375768390096750	0
1.9412377657722000	3.0739252463348600	1
2.5014266626223600	1.160313329866550	1
0.4679612044212450	0.4434108744775510	0
0.7645416621333130	1.1807868743362400	0
1.9130727540939500	1.4221185407926000	1
0.111826487516038	0.575035566912016	0
1.5146839634063200	2.268743328404230	0
1.4829371227873700	1.9019150833319300	1
-0.3736772804528410	-0.4342697042485070	0
-0.1784637424485630	1.1442869652435100	0
0.0992803767020507	-0.332026663271685	0
1.185219949178080	1.3963724267780000	1
1.0953588808686800	3.1223222437971200	1
3.3957523372572100	3.660279521929390	1
1.6999646192922200	1.3818533601854700	1
2.5665764159532500	1.4468843716195500	1
-0.7038135814186100	1.9874836356974500	0
-1.4207150621212900	0.1023286992055120	0
0.993111840225474	2.764529763431160	1
0.5358164393793030	1.5182463055524700	0
3.096457940506600	2.548434755164580	1
3.326828784619190	1.5024890554777100	1

4.416965968383810	0.9847547432431060	1
0.9402238057078760	2.239144666991400	0
0.9548472585837160	0.9631888877436280	0
-0.7557852676418120	-0.7405821016333690	0
-0.6723670613360700	2.6036668507314100	0
1.7519809742859800	1.4942032372672300	1
1.3531627422439500	2.0961189147860400	0
-0.6060513743793620	0.6931615422735390	0
2.356956830283130	-0.3651905149064520	1
0.7495904145465280	0.8352596500090170	0
0.6028934873446810	0.2969368325952510	0
-0.2538333038829320	1.5178275522556900	0
1.5367375776874200	1.9116828788417700	0
2.6563813793534000	1.7530359538025100	1

2. The split data of Dataset2

The table below is the test set of Dataset2:

feature1	feature2	label
2.5494869201246000	2.540780390152160	1
0.9548472585837160	0.9631888877436280	0
-0.1739569118243200	-1.086805174004270	0
2.98653179881027	1.3197796015372600	1
3.445850089770920	1.3643472759038100	1

-0.3895244312965430	1.3673269879717600	0
2.565142301299080	-2.2456412656282300	0
1.9468627139564	1.928284904963290	1
2.045829537103230	4.060021374084250	1
2.0987181302286900	6.5653514457518900	1
1.8442848471742100	2.412817635111410	1
0.4303765567938070	1.4070964194832200	0
-0.7622062694262290	0.2148742035089470	0
-0.4992238039806130	1.4626238439138400	0
-0.3428805600759970	1.390229501683550	0
1.5303247849070700	3.0977352544605200	1
0.4821740550021290	1.1415694756345600	0
0.9402238057078760	2.239144666991400	0
3.096457940506600	2.548434755164580	1
4.055440519456520	1.239944551884510	1
2.3260686410432800	2.657272552181530	1
1.5367375776874200	1.9116828788417700	0
-0.1267497638436230	0.9226719075256420	0
3.9908467663642600	-1.1429863003365900	1
-0.0282978749223901	1.102774000298400	0
1.6211827703905800	2.692674773323640	0
3.342518840978810	1.979195840079850	1

The table below is the traning set of Dataset2:

feature1	**feature2**	**label**

1.5146839634063200	2.268743328404230	0
2.2954452393098200	2.2375129860406800	0
0.0992803767020507	-0.332026663271685	0
-3.951047159005270	-0.44764262073455400	0
-4.209301764620870	3.313574892076170	0
2.365425000359470	1.6752555184148400	1
0.4693492450640310	3.1364369028536300	1
-0.7557852676418120	-0.7405821016333690	0
4.416965968383810	0.9847547432431060	1
2.325113718566920	1.7200141213939100	1
2.356956830283130	-0.3651905149064520	1
0.5825533001319400	1.4728781626219000	0
1.9130727540939500	1.4221185407926000	1
-0.844555153601184	1.5936922615224200	0
1.2676392923050300	1.5662363932232000	0
2.6148847490220600	2.575954342809810	1
-1.4207150621212900	0.1023286992055120	0
0.5358164393793030	1.5182463055524700	0
1.113743651635260	0.9224910573188100	1
1.4829371227873700	1.9019150833319300	1
0.7495904145465280	0.8352596500090170	0
6.78426287967574	4.630375747339830	1
-0.6060513743793620	0.6931615422735390	0
0.6028934873446810	0.2969368325952510	0
2.8341059548307400	0.611221069074144	1

3.326828784619190	1.5024890554777100	1
-0.4025330788260770	1.207739733740650	0
1.6462100988309400	3.526534638650090	1
2.2203930908157500	1.935946856476640	1
1.6999646192922200	1.3818533601854700	1
-0.2538333038829320	1.5178275522556900	0
2.344543356237090	3.2071710125134500	0
1.948843876117910	0.9393313362738170	1
-0.6723670613360700	2.6036668507314100	0
2.5665764159532500	1.4468843716195500	1
3.3957523372572100	3.660279521929390	1
0.993111840225474	2.764529763431160	1
0.8584072176711300	2.375768390096750	0
0.6632460452284320	-0.6051033525085560	0
1.185219949178080	1.3963724267780000	1
1.7783417203563500	1.9223441017293200	1
2.5014266626223600	1.160313329866550	1
1.0953588808686800	3.1223222437971200	1
1.2461230358784400	5.573061064777870	0
0.4679612044212450	0.4434108744775510	0
2.6563813793534000	1.7530359538025100	1
-0.0150441965923516	0.7551524591177790	0
1.7519809742859800	1.4942032372672300	1
2.0910025593215600	1.9848953366663100	1
-0.1784637424485630	1.1442869652435100	0

1.3531627422439500	2.0961189147860400	0
0.7645416621333130	1.1807868743362400	0
0.111826487516038	0.575035566912016	0
-0.7038135814186100	1.9874836356974500	0
1.771950548836690	1.6894084885642800	1
-0.3736772804528410	-0.4342697042485070	0
-1.4033171544297800	0.6758468701916590	0
1.9117748740417800	-1.5620015479615600	1
1.9412377657722000	3.0739252463348600	1
1.5477093176871800	1.6046076514939800	1
1.4291320323571100	2.9866338433261900	1

The four models:k-NN, Naive Bayes, Decision Trees, and Random Forests

1. Naive Bayes

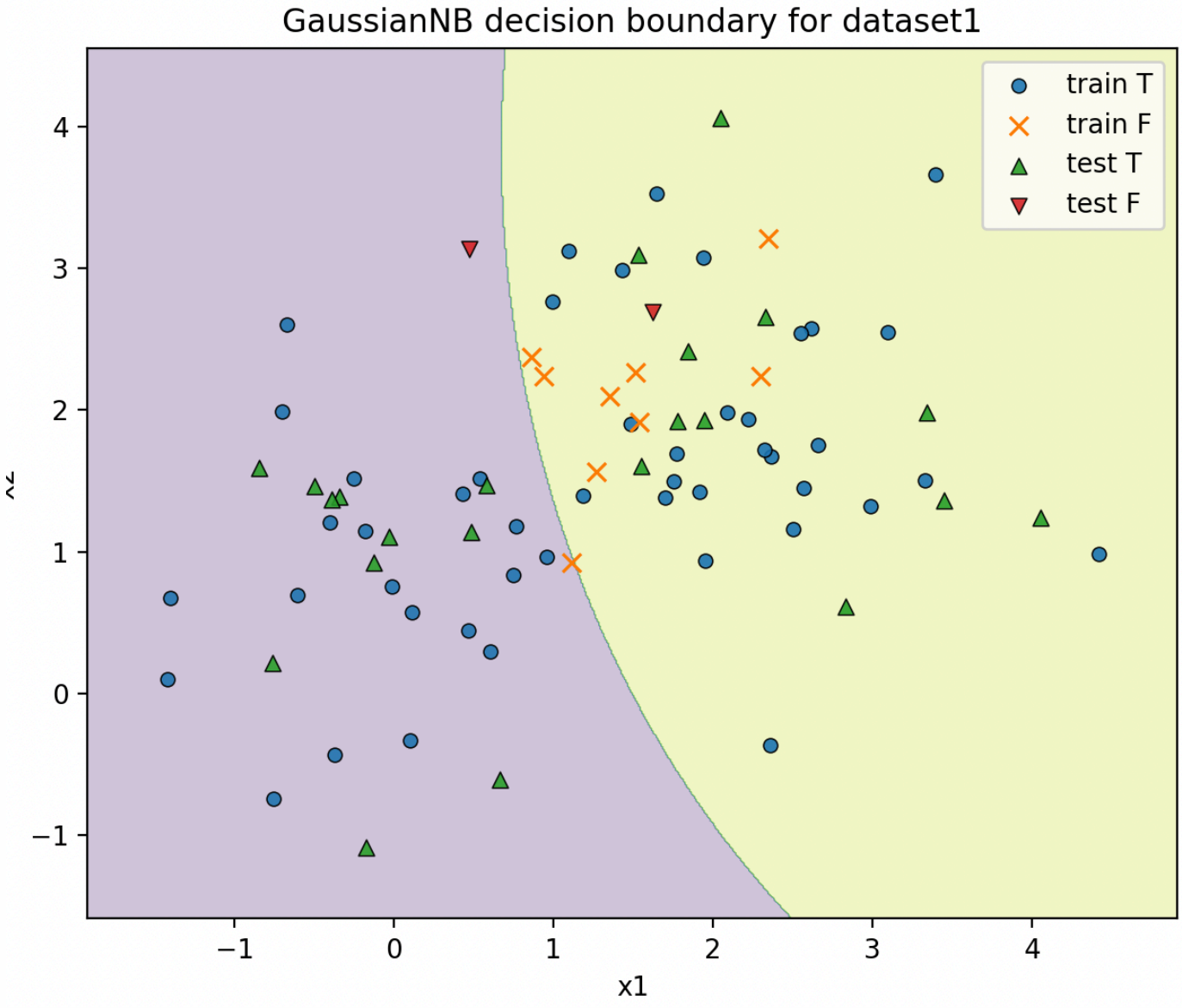
The Accuracy:

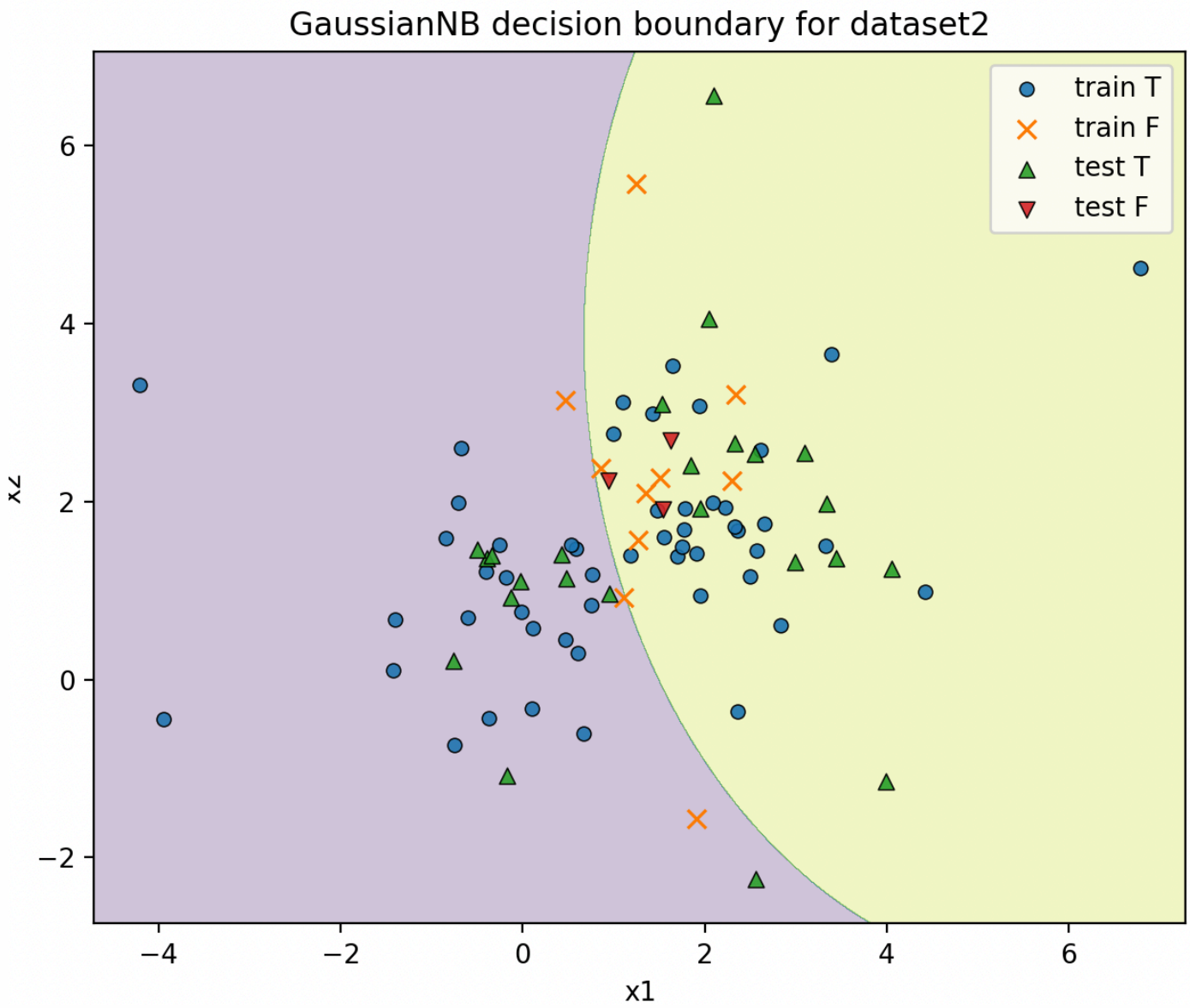
Dataset	Testing set	Training set
Dataset1	91.6%	85.7%
Dataset2	92.6%	83.6%

The Confusion Matrix:

Dataset	Testing set	Training set
Dataset1	$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$	$\begin{bmatrix} 22 & 6 \\ 2 & 26 \end{bmatrix}$
Dataset2	$\begin{bmatrix} 12 & 2 \\ 0 & 13 \end{bmatrix}$	$\begin{bmatrix} 24 & 6 \\ 4 & 27 \end{bmatrix}$

Visualization of the results:





2. Decision Tree

The Accuracy:

Dataset	Testing set	Training set
Dataset1	91.6%	100%
Dataset2	77.8%	100%

The Confusion Matrix:

Dataset	Testing set	Training set
Dataset1		

	$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$	$\begin{bmatrix} 28 & 0 \\ 0 & 28 \end{bmatrix}$
Dataset2	$\begin{bmatrix} 10 & 4 \\ 2 & 11 \end{bmatrix}$	$\begin{bmatrix} 30 & 0 \\ 0 & 31 \end{bmatrix}$

3. Random Forest

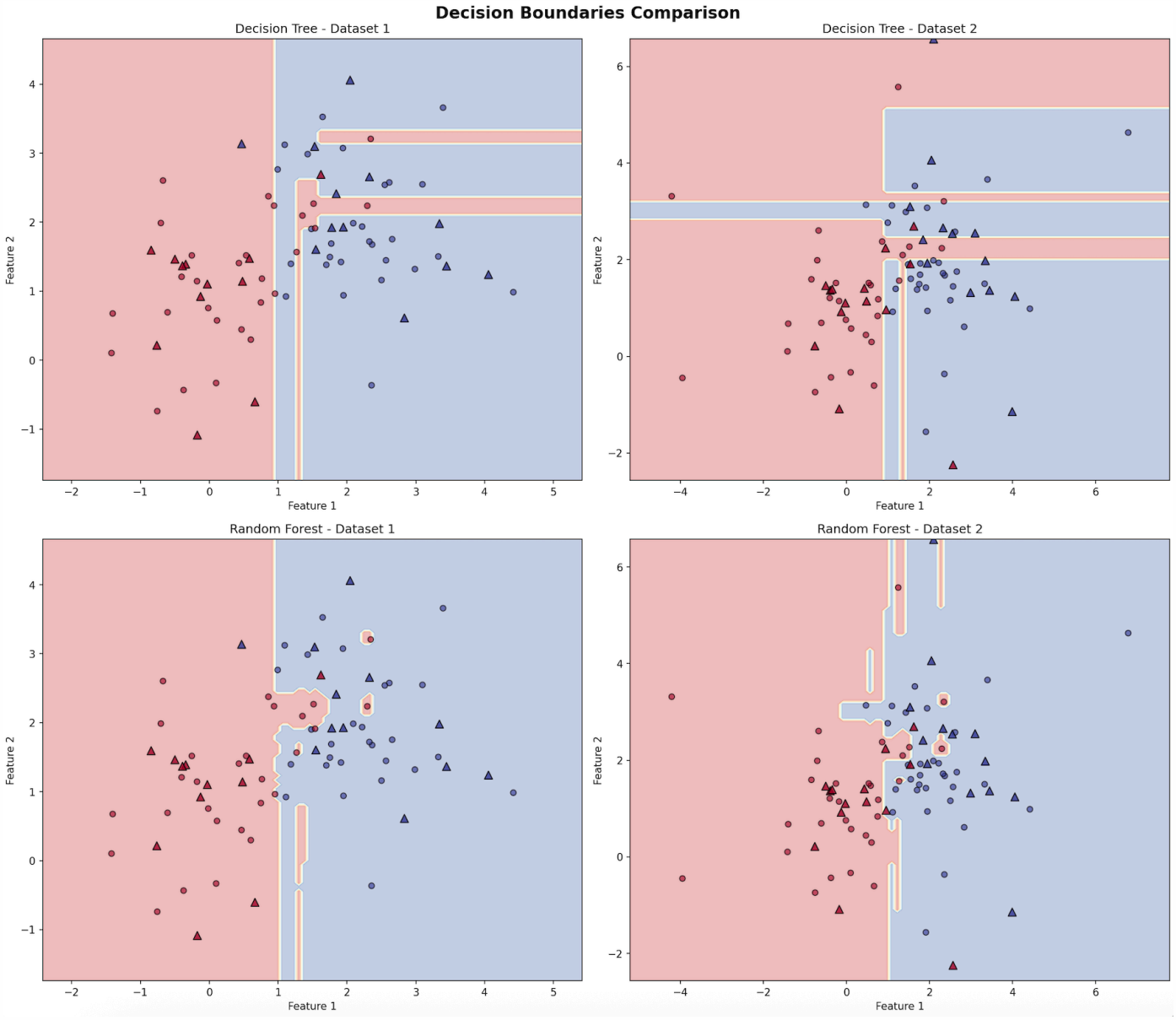
The Accuracy:

Dataset	Testing set	Training set
Dataset1	91.6%	100%
Dataset2	85.2%	100%

The Confusion Matrix:

Dataset	Testing set	Training set
Dataset1	$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$	$\begin{bmatrix} 28 & 0 \\ 0 & 28 \end{bmatrix}$
Dataset2	$\begin{bmatrix} 10 & 4 \\ 0 & 13 \end{bmatrix}$	$\begin{bmatrix} 30 & 0 \\ 0 & 31 \end{bmatrix}$

The visualization of results obtained from Decsion Tree and Random Forest:



4. K-NN

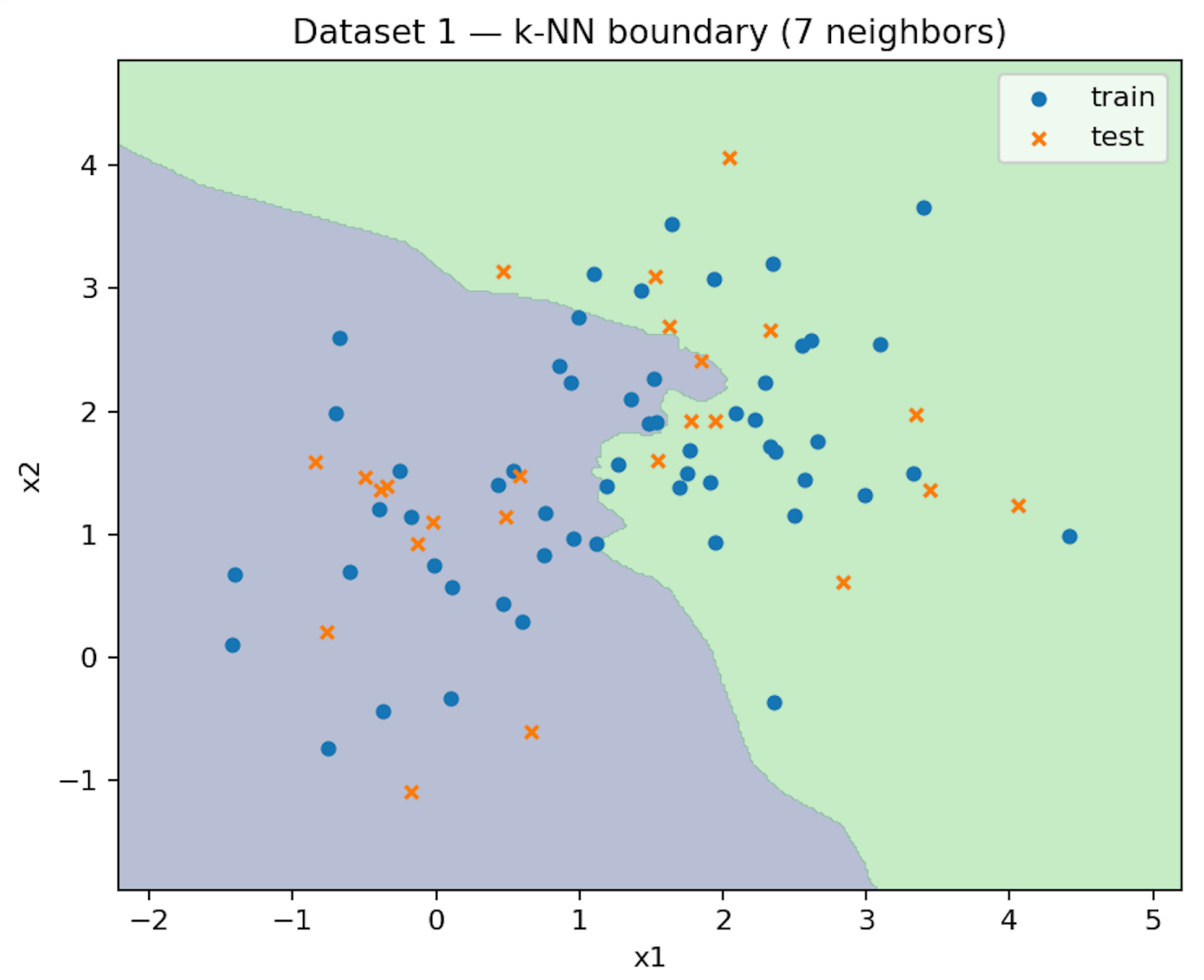
The Accuracy:

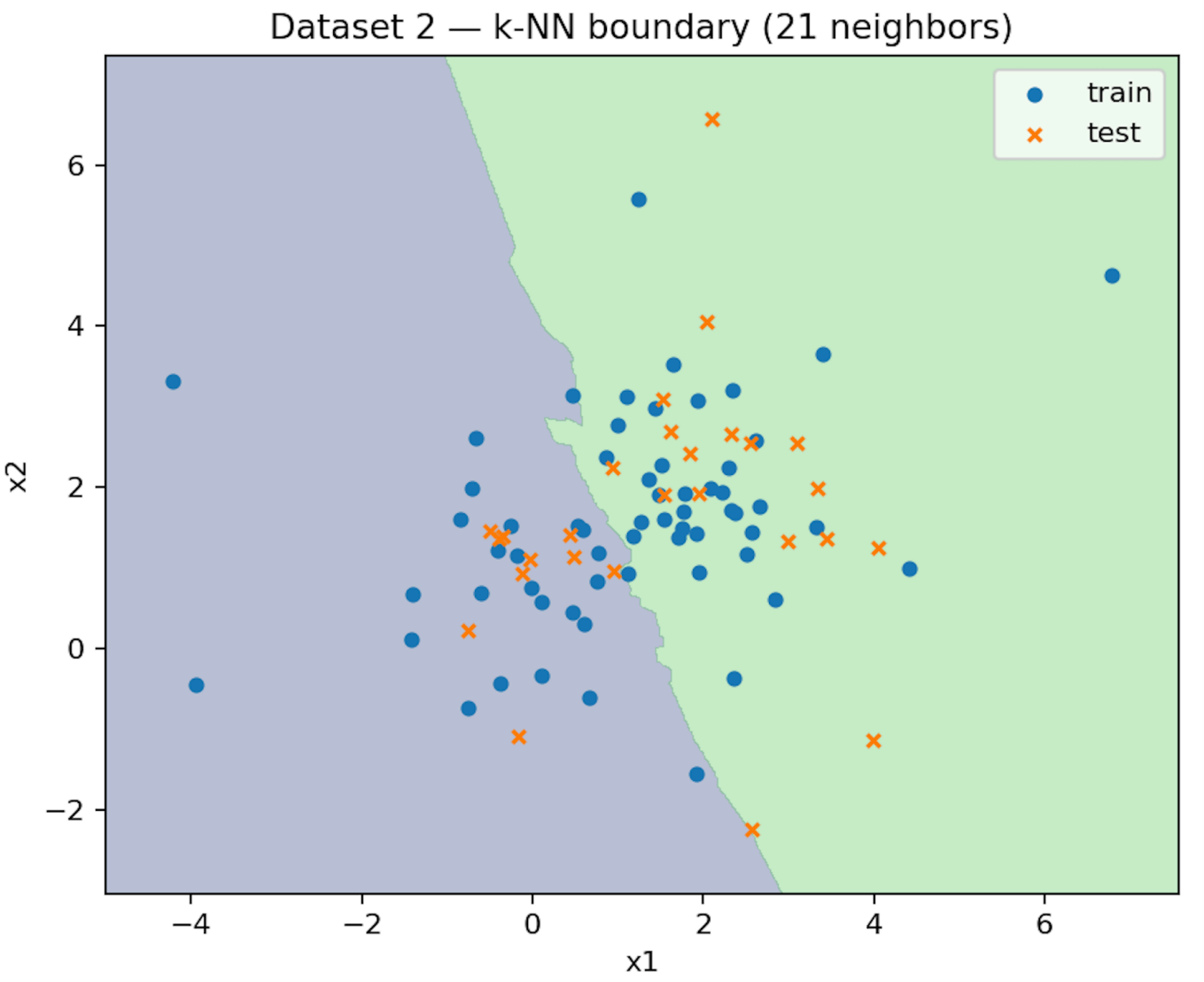
Dataset	Testing set	Training set
Dataset1	91.7%	91%
Dataset2	85.2%	85.2%

The Confusion Matrix:

Dataset	Testing set	Training set
Dataset1	$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$	$\begin{bmatrix} 25 & 3 \\ 2 & 26 \end{bmatrix}$
Dataset2	$\begin{bmatrix} 10 & 4 \\ 0 & 13 \end{bmatrix}$	$\begin{bmatrix} 23 & 7 \\ 2 & 29 \end{bmatrix}$

The visualization of results obtained from Decsion Tree and Random Forest:





Discussion about the results

We discussed the results from the **discrimination** and **generalization** perspective.

In general, Naive Bayes shows best performance acorss the datasets among the models. By contrast, Decision Tree performs the worst.

For this dataset, Naive Bayes model achieves the best performance on both perspectives, as they has highest test-set accurcay, with 91.6% and 92.6%, respectively. As both classes have equal numbers of samples, it is enough to use accuracy alone to evaluate their discrimination ability. This model's generalization ability is also excellent on the datasets. We can see this from the accuracy differences between testing sets and training sets: -5.9% and -9.0%.

The ability to discriminate of Random Forest is also strong. For Dataset1, its accuracy is 91.7%, the same as that of Naive Bayes. However, it only obtains 85.2% accuracy on Dataset2. In addition, this model slightly suffers from overfitting on Dataset2.

The discrimination ability of k-NN is almost smiliar to that of Random forest, but the other performance is better than Random forest's, because a reasonable accuracy differences.

Fininally, there is a large accuracy gap of 22.2% between the traning set and testing set on Dataset2, and the accuracy on training set is 100%. It is obvious that this model overfits Dataset2. In addtion, its discrimination performance on Dataset2 is also the poorest, because of a test accuracy with only 77.8%.

Similarities and dissimilarities

1. Similarities

These four models mainly have two similarites:

1. Supervised Learning: All of them are supervised learning, which means they learn from labeled data.
2. Classification Task: Their only goal is to perform classification, which means assigning new, unseen data points to one or more predefined classes.

2. Disimilarities

The main differences among them are their principles. Hence, this cause other important differcences: **Data-Model fit, Decision Boundary, Generalization perfomance and Cost.**

- **Data-Model fit**: What kind of data is suitable for this model?
    - k-NN: Thise model is suitable for data where distances in the feature space are meaningful.

    - Naive Bayes: This model expects the features to be as independent as possible.
    - Decesion Tree: This model expects the data to be divided into pure subsets through features, and does not require independence or linear relationships among them.
    - Random Forest: This model expects the data to be separable through different combinations of features, which provide complementary information.
- **Decision Boundary**:
    - k-NN: Its Decision Boundary is highly nonlinear and locally adaptive and entirely determined by the positions of the training samples.
    - Naive Bayes: The decision boundary is usually smooth and approximately linear or quadratic.
    - Decesion Tree: The decision boundary of a decision tree consists of horizontal and vertical lines parallel to the coordinate axes, forming a staircase-like pattern.
    - Random Forest: The decision boundary can be highly complex locally, but smoother and more generalizable globally than that of a single decision tree.
- **Cost on Traning and Predition**: Aussme that d is the demention of the features, n is the number of samples and T is the number of trees in the forest.
    -

Model	Training Cost	Prediction Cost
k-NN	**Low** — $O(1)$	**High** — $O(n \cdot d)$
Naive Bayes	**Low** — $O(n \cdot d)$	**Low** — $O(d)$
Decision Tree	**Medium** — $O(n \cdot d \log n)$	**Low** — $O(\text{tree depth})$
Random Forest	**High** — $O(T \cdot n \cdot d \log n)$	**Medium** — $O(T \cdot \text{tree depth})$

- **Generalization perfomance** :
    -

Model	Bias–Variance Trade-off	Sensitivity to Noise/Outliers	Typical Generalization
k-NN	**Low bias, high variance**	**Sensitive**: strongly affected by noisy neighbors, outliers,	**Moderate**: strong on low-dimensional data with clear local structure;

		and feature scaling or distance metric	degrades in high-dimensional settings
Naive Bayes	**High bias, low variance**	**Relatively stable**	**Moderately favorable**: often excellent on high-dimensional sparse data (text); drops when features are strongly correlated
Decision Tree	**Low bias, high variance**	**Sensitive**	**Moderate** when well-regularized; but prone to overfitting without it
Random Forest	**Variance greatly reduced**; bias similar to or slightly higher than a single tree	**Very Stable**: relatively insensitive to noise or outliers	**High**: typically strong generalization performance

Part C

UL task: Clustering

CI.1

Motivation:The goal of clustering is to group data samples into clusters such that samples within the same cluster are more similar to each other than to those in different clusters. This task is suitable for exploring the intrinsic structure of unlabeled data and identifying natural groupings or patterns within it.

Metrics:

1. **Silhouette Coefficien**t: It measures how similar each sample is to its own cluster

compared to other clusters, based solely on distance information. It ranges from −1 to 1, where higher values indicate that samples are well matched to their own cluster and poorly matched to neighboring clusters.
Motivation: This metric is unsupervised, which means it does not require ground-truth labels, making it suitable for internal evaluation when only feature similarity is known. It helps quantify cluster compactness and separation.

2. **Adjusted Rand Index (ARI)**: It compares the clustering assignments with ground-truth labels, adjusting for chance. It evaluates how consistent the predicted clusters are with the true categories, ranging from −1 (poor agreement) to 1 (perfect match). 0 means the clustering result is random.
Motivation: This metric is supervised, which means it can be used when reference labels are available. It provides an objective measure of clustering accuracy.

CI.2 Datasets (Design Choice & Rationale)

Olivetti Faces (real-world, images):
400 images (64×64 → 4096-D). High-D, small-n; benefits from PCA and is a realistic testbed for face embeddings.

Model fit:

- **HDBSCAN:** after PCA (≈100–200), can find identity pockets and mark outliers (`min_cluster_size` tunes strictness).
- **OPTICS:** density ordering shows structure; reasonable on PCA features.
- **Spectral (ours):** with `affinity='nearest_neighbors'` often recovers ≈40 subjects on PCA features; captures non-linear separations.

(We initially considered the generated Two-Moons dataset to illustrate nonconvex structure, but we ultimately report only Olivetti for a purely real-world evaluation.)

CI.3

Model 1: HDBSCAN

Motivation: HDBSCAN is an extension of the DBSCAN algorithm that combines density-based clustering with a hierarchical framework. Unlike algorithms that require a fixed number of clusters (e.g., K-means) or are sensitive to global density thresholds, HDBSCAN can automatically discover clusters of variable densities and identify noise points without manual tuning of the number of clusters.

Key characteristics:

1. No need to predefine the number of clusters
2. Robust to noise and outliers
3. Ability to handle clusters of varying density
4. Scalability and efficiency with complexity $O(nlogn)$

Model2: Spectral Clustering

Motivation:Spectral converts clustering into graph partitioning. It builds a k-nearest-neighbors (kNN) similarity graph and clusters in the space of the graph Laplacian's leading eigenvectors, which captures non-linear structure (pose/lighting/expression) better than centroid-based models.

Key Characteristics:

1. Handles non-linear cluster geometry via a neighborhood graph.
2. Less biased toward spherical clusters than k-means/GMM; tolerates shape/size imbalance.
3. Requires `n_clusters` — useful here because prior knowledge is ≈40 subjects.

CI.4

How HDBSCAN works?

HDBSCAN extends the classical DBSCAN algorithm by transforming the problem of clustering into one of hierarchical density analysis.
Step 1 – Mutual Reachability Distance: HDBSCAN starts by defining a mutual reachability distance between each two points in the dataset,

$$d_{mreach}(a,b) = \max\{d_c(a), d_c(b), d(a,b)\}, \tag{26}$$

while core distance is defined as:

$$d_c(x) = distance_{x-k}(x, k = min_{samples}). \tag{27}$$

Step 2 – Minimum Spanning Tree (MST): Using these mutual reachability distances, HDBSCAN constructs a minimum spanning tree (MST) of the data points. The MST encodes the connectivity structure of the dataset in terms of density-based reachability.
Step 3 – Hierarchical Cluster Tree Construction: The MST is then transformed into a hierarchical cluster tree by progressively removing edges in order of decreasing mutual

reachability distance (i.e., increasing density). Each edge removal may cause clusters to split into smaller subclusters. This process naturally produces a hierarchy of clusters at multiple density levels.

Step 4 – Condensed Tree and Cluster Stability Analysis: HDBSCAN condenses this hierarchy tree by pruning small clusters below a minimum cluster size and calculating a stability score for each cluster. This score measures how long a cluster persists across different density thresholds, and the longer the persistence, the more stable and meaningful the cluster is.

$$stability(C) = \sum_{x \in C} (\lambda^C_{leave}(x) - \lambda^C_{join}(x)) \tag{28}$$

Step 5 – Cluster Selection via EOM (Excess of Mass): The Excess of Mass (EOM) algorithm selects the final flat clustering by maximizing total cluster stability (densest and most persistent) under the constraint that clusters do not overlap.

Optimization method: EOM is a stability-based optimization method, aiming to maximize overall cluster persistence across different density levels. This corresponds to an optimization problem of selecting clusters that maximize total stability mass (EOM). HDBSCAN optimizes cluster selection in a hierarchical density space, rather than through iterative parameter updates.

$$\text{maximize} \sum_{C_i \in S} stability(C_i) \tag{29}$$

How Spectral Clustering Works?

(**Method, Optimization, Implementation**)

Step 1 – Affinity Graph:
Build sparse kNN graph **W** (edges between nearest neighbors).

Step 2 – Normalized Laplacian:

$$L_{sym} = I - D^{-1/2} W D^{-1/2} \tag{30}$$

with $D_{ii} = \sum_j W_{ij}$.

Step 3 – Spectral Embedding:
Take the *k* eigenvectors of L_{sym} with smallest eigenvalues; row-normalize to get $U \in \mathbb{R}^{n \times k}$.

Step 4 – Discretization:
Run k-means on rows of **U** to obtain *k* clusters.

Optimization view:
Spectral is the relaxed solution of Normalized Cuts

$$Ncut(A_1, \ldots, A_k) = \sum_{r=1}^{k} \frac{cut(A_r, \bar{A}_r)}{assoc(A_r, V)} \tag{31}$$

The exact minimization is NP-hard; the spectral relaxation reduces it to an eigenproblem on the Laplacian, then k-means discretizes the relaxed solution.

Our implementation (scikit-learn):

Preprocessing: $PCA(100, whiten = True)$ on flattened images (denoise + stable distances).

Model: $SpectralClustering(affinity =' nearest\_neighbors', assign\_labels =' kmeans')$

Swept hyper-params:

$n\_clusters \in 30, 35, 38, 40, 42, 45, 50, n\_neighbors \in 8, 10, 12, 15, 20, random\_state = 42.$

CI.5

Results for HDBSCAN:

min_cluster_size	min_samples	Clusters	Clustered Samples	Clustered	Silhouette	ARI
3	2	68	341	85.25%	0.2305	0.3957
5	2	39	293	73.25%	0.1982	0.2508
5	3	30	236	59.00%	0.2259	0.1201

We applied **HDBSCAN** on the **Olivetti Faces** dataset (**400 samples**, **40 ground-truth subjects**) to study how its clustering behavior changes with different hyperparameters. Overall, **HDBSCAN** produced varying results depending on its **density parameters**. Smaller *min_cluster_size* values (e.g., *3*) yielded **more clusters** and **higher ARI** (≈ 0.40), while larger values produced **fewer, more stable clusters** but **lower coverage** and **ARI**.
Silhouette scores (≈ 0.2) remained moderate, suggesting reasonable but imperfect separation in the **high-dimensional facial feature space**.
The **ARI <** 0.4 indicates that purely density-based methods still struggle with **high-dimensional, facial image data**.

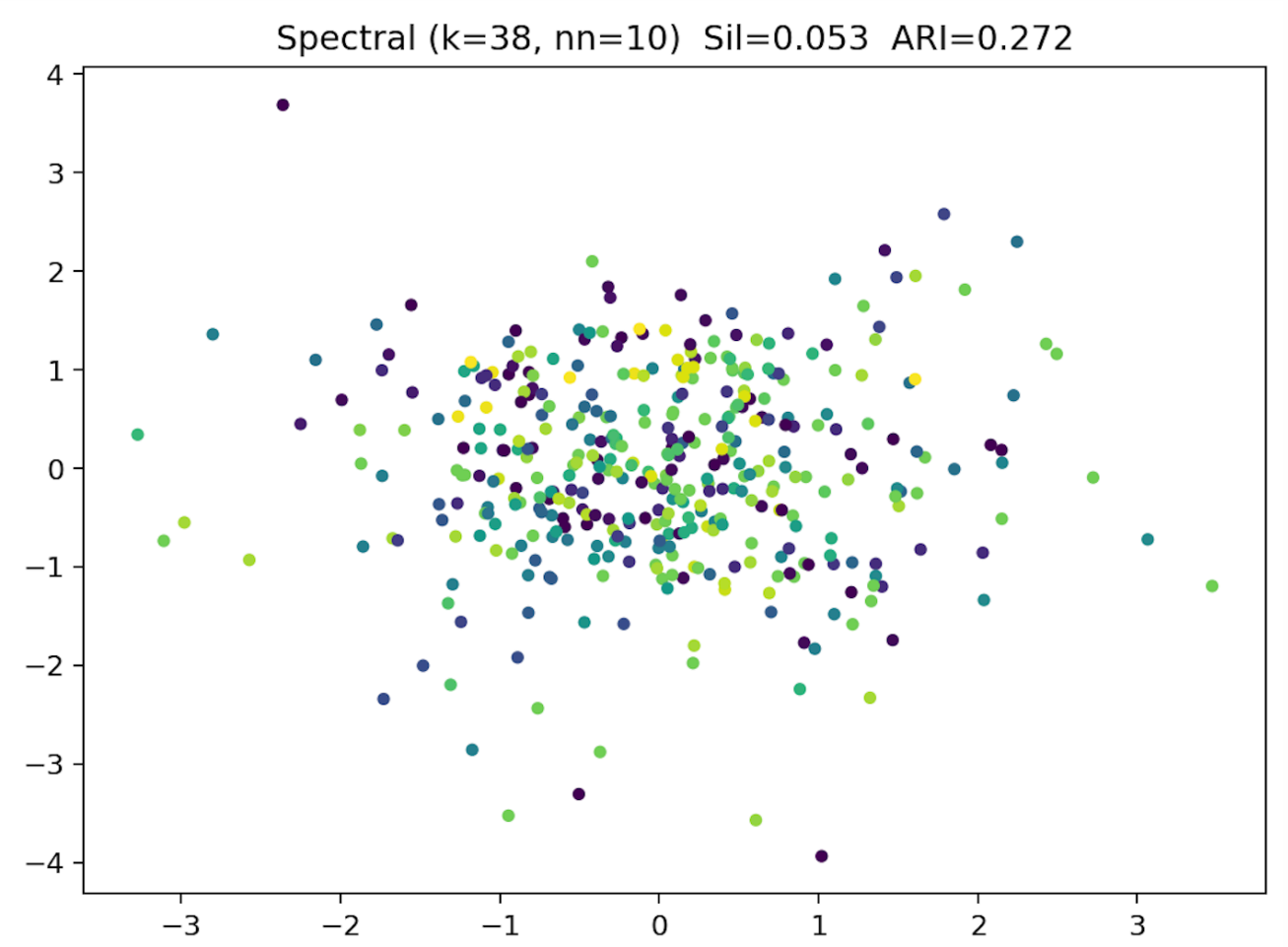
We tried to use **PCA** to reduce the dimension of the raw data and extract features. However, the **clustering performance did not improve**: both **Silhouette** and **ARI** scores remained similar to or worse than those obtained from the raw data. This outcome suggests that **PCA failed to capture the semantic features** necessary for distinguishing different identities. As a *linear method*, **PCA** preserves **global variance** but not the **non-linear manifold structure** of facial images. Consequently, the resulting **lower-dimensional representation** does not provide a more meaningful structure for **density-based clustering methods** such as **HDBSCAN**.

Results for Spectral Clustering:

Input: n=400, 40 ground-truth subjects
Features used: $PCA(100) \rightarrow X \in R^{100 \cdot 400}$

Parameters: `n_clusters=38`, `n_neighbors=10`
Evaluation metrics:

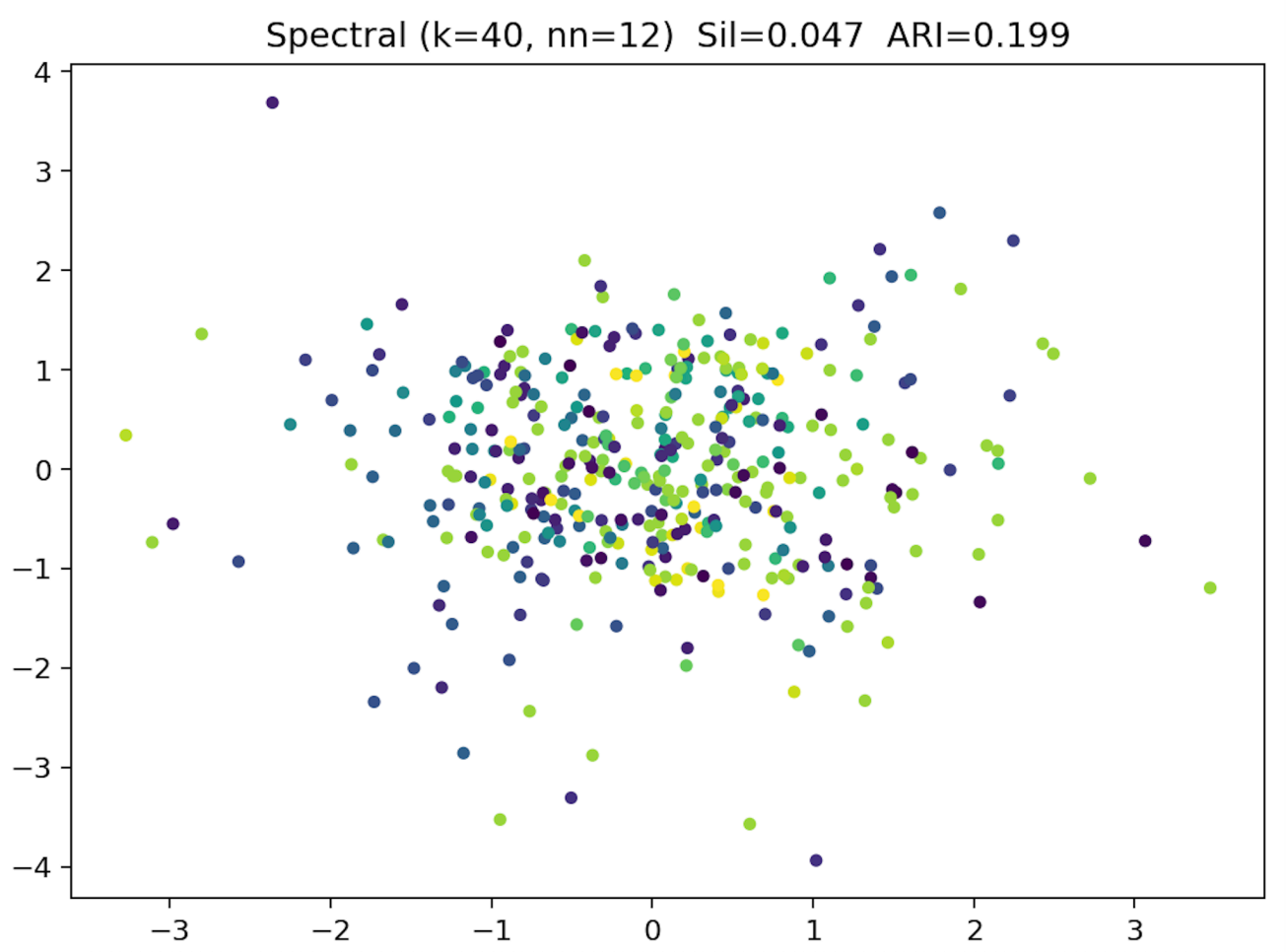
- `silhouette_score: 0.053`
- `adjusted_rand_score: 0.272` *(best ARI in our sweep)*



Parameters: `n_clusters=40`, `n_neighbors=12`

Evaluation metrics:

- `silhouette_score: 0.047`
- `adjusted_rand_score: 0.199`



- **Best setting by ARI:** (k=38, nn=10) slightly under-specifies the nominal 40 identities, avoiding over-splitting and yielding the best identity consistency (ARI ≈ 0.272).
- **Silhouette magnitudes:** low (~0.05) across settings ⇒ clusters are partly overlapping in PCA(100) with Euclidean distance; identity separation exists but is weak in this space.
- **Effect of n_neighbors:** Too small fragments the graph; too large over-smooths and merges identities. The sweet spot is 10–12 neighbors.
- **Full assignment:** Spectral labels every sample (no "noise"), which simplifies Silhouette reporting and contrasts with density methods that may leave hard cases as noise.

CI.6

HDBSCAN Conclusion:

In summary, the results suggest that **density-based clustering methods**, such as **HDBSCAN**, are **not well-suited** for **raw, high-dimensional facial image data**. The **pixel space** of face images lacks **distinct density boundaries** and exhibits **strong non-linear and continuous variations**, which violate the **core assumptions** of density-based clustering.

To make **HDBSCAN** more effective for this type of data, it is necessary to first obtain a **more meaningful representation** through **non-linear dimensionality reduction** or **deep feature extraction**. These approaches can transform the **original pixel data** into a **lower-dimensional manifold** where samples from the same identity are **closer together**, thereby providing a **feature space** that better reflects **semantic similarity** and allows **HDBSCAN** to form **more coherent clusters**.

Spectral Clustering Conclusion:

Using **Spectral Clustering** with a **k-NN affinity** on **PCA(100, whiten=True)** features of the *Olivetti faces*, we obtained moderate identity consistency:
the best setting in our sweep (e.g., `n_clusters=38`, `n_neighbors=10`) delivered **ARI ≈ 0.27** with **Silhouette ≈ 0.05**.

This indicates that Spectral can recover a meaningful portion of the underlying identity structure without supervision, but separability remains limited in this feature space.

We also observed a consistent trade-off:

- Slightly underspecifying the number of clusters relative to the 40 subjects avoided overfragmentation and improved ARI.
- Larger k marginally increased Silhouette but split identities.

Interpretation

Low Silhouette across settings suggests overlapping clusters under Euclidean distances in PCA(100)—i.e., the geometry is still "blurry" for identity.

Nonetheless, Spectral's non-linear partitioning (via the graph Laplacian eigenvectors) captures local structure better than centroid baselines would on the same features and gives a clean, full assignment of all samples (useful for internal metrics).

Limitations

1. **Requires n_clusters**
   Spectral needs k ahead of time. While we justified it via the dataset prior (~40 subjects) and a sweep, the choice still influences outcomes.

2. **Sensitivity to n_neighbors**
   Too small fragments the graph; too large oversmooths and merges identities. Performance varies with this single knob.
3. **Feature space is linear PCA**
   PCA preserves variance, not identity separability. Important, low-variance identity cues may be discarded; Euclidean distances in PCA space may not reflect semantic similarity.
4. **Full assignment (no noise class)**
   Spectral cannot label "hard" images as noise the way density methods can, which can depress Silhouette when clusters overlap.
5. **Scale & generalization**
   Results are on a small dataset (n=400). Out-of-sample assignment isn't native (requires a transform strategy like Nyström); scalability depends on sparse eigensolvers and kNN graph construction.

Possible Future Improvements

Better Representations

- **More expressive features:** Replace PCA with learned embeddings (e.g., a pretrained face CNN or a small autoencoder/contrastive encoder). Then run Spectral on those embeddings.
- **Non-linear DR before Spectral:** Try UMAP (`n_neighbors`≈15, `n_components`≈30) → Spectral; UMAP often improves neighborhood faithfulness for faces.
- **PCA tuning:** Re-run with 150–200 PCs and no whitening as an ablation; on some runs this preserves more identity signal for graph construction.

Graph & Objective Variants

- **Affinity choices:** Compare *nearest_neighbors* vs *RBF/heat kernel affinity*; tune σ via local scaling (self-tuning spectral clustering) to adapt to density variation.
- **Laplacian variants:** Try random-walk Laplacian and check stability; inspect the eigengap to guide *k*.

Model Selection & Robustness

- **Selection protocol:** Choose *(k, n_neighbors)* via Silhouette (primary) and report ARI post-hoc; include per-cluster Silhouette and cluster size balance to detect over-fragmentation.

- **Stability checks:** Repeat with multiple seeds for the k-means discretization; report variance of metrics.
- **Baselines:** Add k-means/GMM on the same features as reference points; include Davies–Bouldin or Calinski–Harabasz for completeness.

Scaling & Deployment

- **Approximate kNN & Nyström:** For larger n, use ANN graphs (e.g., FAISS) and Nyström out-of-sample extensions to map new images to clusters.
- **Noise handling hybrid:** If "unclusterable" images are problematic, consider a hybrid: Spectral to get coarse partitions, then HDBSCAN within clusters to allow noise labeling.

Reporting Polish

- Include a small hyperparameter grid table (top 5 rows by ARI and by Silhouette).
- Add the best-config scatter (2D PCA colored by spectral labels).
- Optionally, include an eigengap plot to justify the k region.