Machine Learning - Exercise 2

Regression

Esra Ceylan, Daniel Fangl, Christian Hatschka

TU Wien

Table of contents

Data Sets

Diamonds Data Set

Several attributes of diamonds, e.g. carat, clarity or measurements has been collected.

- #samples = 53 940
- #attributes = 9
- 6 numeric and 3 categorical attributes
- No missing values
- To predict: price in US Dollars

Concrete Data Set

Data set contains information about the composition as well as the age of cement in days and it's compressive strength.

- #samples = 1 030
- #attributes = 8
- Only numeric attribute
- Has no missing values
- To predict: concrete compressive strength measured in MPa

Life Expectancy Data Set

Data set contains health factors of 193 countries between the years 2000 to 2015.

- #samples = 2 938
- #attributes = 20
- 19 numeric attributes and 1 categorical feature
- Has missing values
- To predict: life expectancy in years

Data Preparation

Data Preparation

1. Categorical data:
    - One-hot encoding
2. Missing values:
    - Most common value
    - Median
3. Scaling:
    - Without
    - Standard-Scaler
    - Min-Max-Scaler

Gradient Descent

Gradient Descent

- $n = \#attributes$
- Weight vector $\vec{\mathbf{w}} = (w_0, ..., w_n)'$
- Minimize error by computing impact of all w_j:

$$\begin{aligned} \min MSE(\vec{\mathbf{w}}) &= \min \frac{1}{\#samples} RSS(\vec{\mathbf{w}}) \\ &= \min \frac{1}{\#samples} \sum_{i=1}^{\#samples} (y_i - (w_0 + \sum_{j=1}^{n} w_j x_{ij}))^2 \end{aligned}$$

Use *MSE* instead of *RSS* as cost function because even for different values for α the weights do not converge for data set with big sample size

Gradient Descent: Pseudocode

Algorithm 1 GD-Regressor(α, *max_iterations*, X_t, Y_t)

for $j = 0$ to n **do** ▷ Initialize weights
    $w_j = 1$
end for
for $k = 1$ to *max_iterations* **do**
    for $j = 0$ to n **do** ▷ Calculate derivative
        Calculate $\frac{\partial MSE}{\partial w_j}(\vec{\mathbf{w}})$
    end for
    for $j = 0$ to n **do** ▷ Update weights
        $w_j = w_j - \alpha \frac{\partial MSE}{\partial w_j}(\vec{\mathbf{w}})$
    end for
end for
return $\vec{\mathbf{w}}$

Calculating Partial Derivative

1. Manually calculated:

$$\frac{\partial MSE}{\partial w_j}(\vec{\mathbf{w}}) = \begin{cases} \frac{-2}{\#samples} \sum_{i=1}^{\#samples} (y_i - (w_0 + \sum_{j=1}^{n} w_j x_{ij})), & j = 0, \\ \frac{-2}{\#samples} \sum_{i=1}^{\#samples} (y_i - (w_0 + \sum_{j=1}^{n} w_j x_{ij})) \cdot x_{ik}, & j \geq 1. \end{cases}$$

   $\rightarrow$ too slow because of huge number of arithmetic operations
   $\sim 2n \cdot \#samples$.

2. In terms of vector-operations:
   $\tilde{X}_t \in \mathbb{R}^{\#samples \times (n+1)}$...insert first column with 1 to training set X_t

$$\left(\frac{\partial MSE}{\partial w_j}(\vec{\mathbf{w}})\right)_{j=0,\ldots,n} = \frac{-2}{\#samples} \tilde{X}_t{}' \cdot \left(Y_t - \tilde{X}_t \cdot \vec{\mathbf{w}}\right)$$

   $\rightarrow$ much faster

Gradient Descent: Pseudocode Prediction

Algorithm 2 GD-Prediction($\vec{\mathbf{w}}$, X_p)

1: $\tilde{X}_p :=$ insert first column with 1 to X_p
2: $Y = \tilde{X}_p \cdot \vec{\mathbf{w}}$
3: **return** Y

Again, using vector operations made the prediction much faster.

Results

- Vector operations way faster than simply summing up
- Although very fast takes more iterations to reach similar results as in sklearn
- Weight initialization has an impact on iterations needed till convergence $\rightarrow$ done differently in sklearn
- For all our data sets best results using Standard-Scaling
- Min-Max-Scaler was not bad, but also not really good
- Without scaling weights get too big $\rightarrow$ do not converge
- In the following plots we used $\alpha = 10^{-2}$ for our implementation of Gradient Descent (best α in our tests) and $\alpha = 10^{-5}$ for the scikitlearn implementation

Performance on Diamonds Data Set

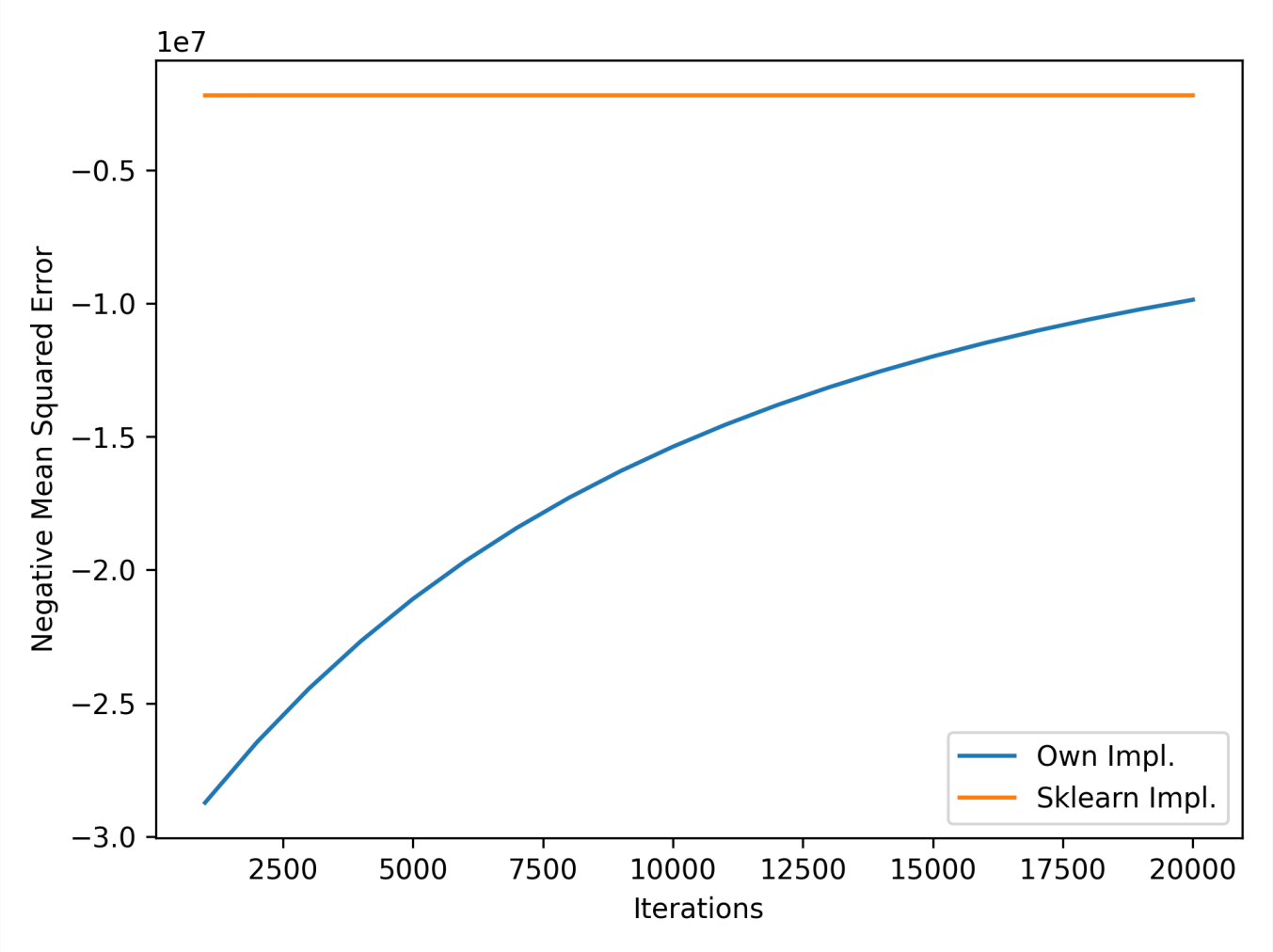

Figure 1: Mean squared error of Gradient Descent on Diamonds Data Set

Performance on Diamonds Data Set

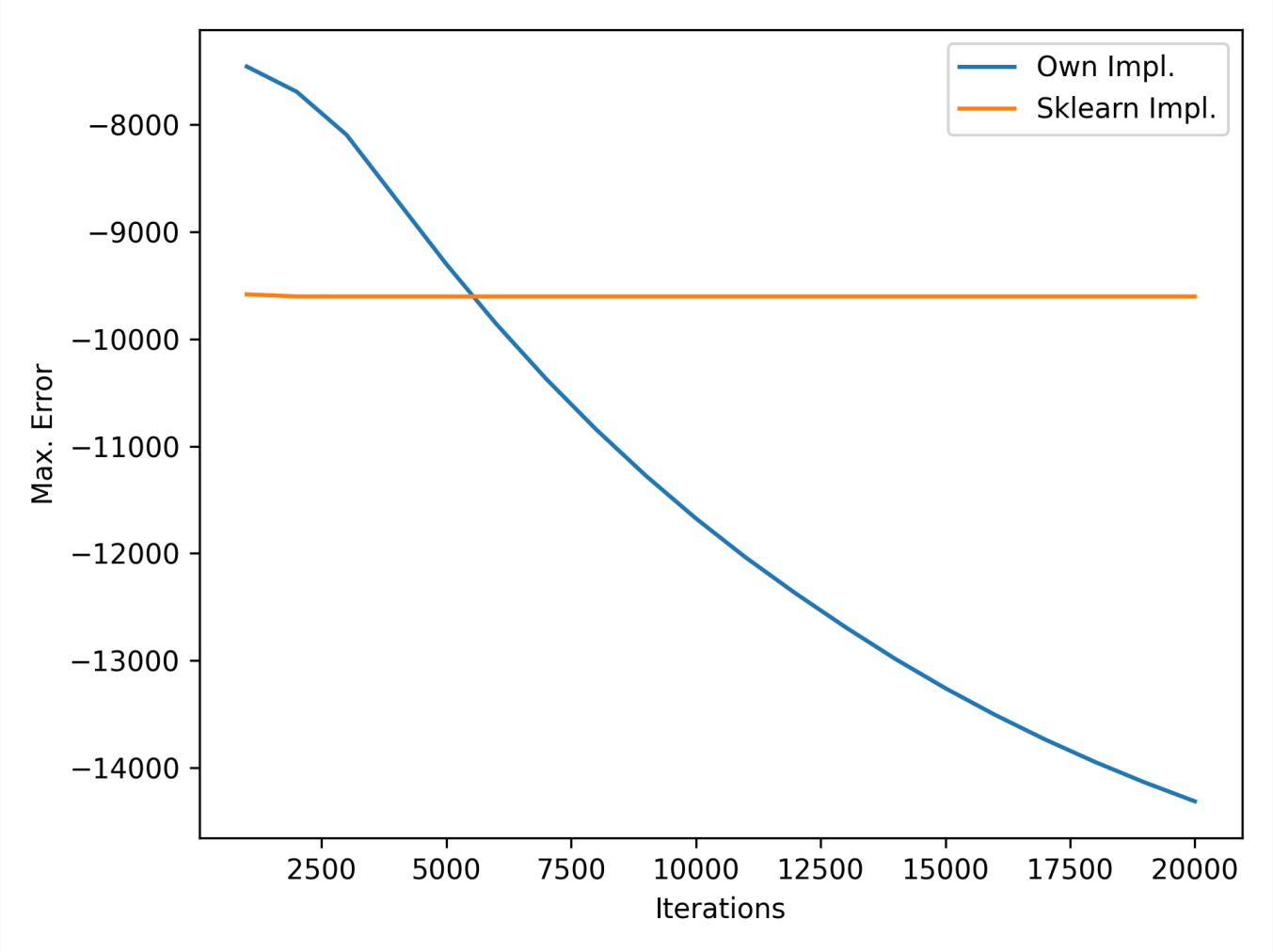

Figure 2: Max. error of Gradient Descent on Diamonds Data Set

Performance on Concrete Data Set

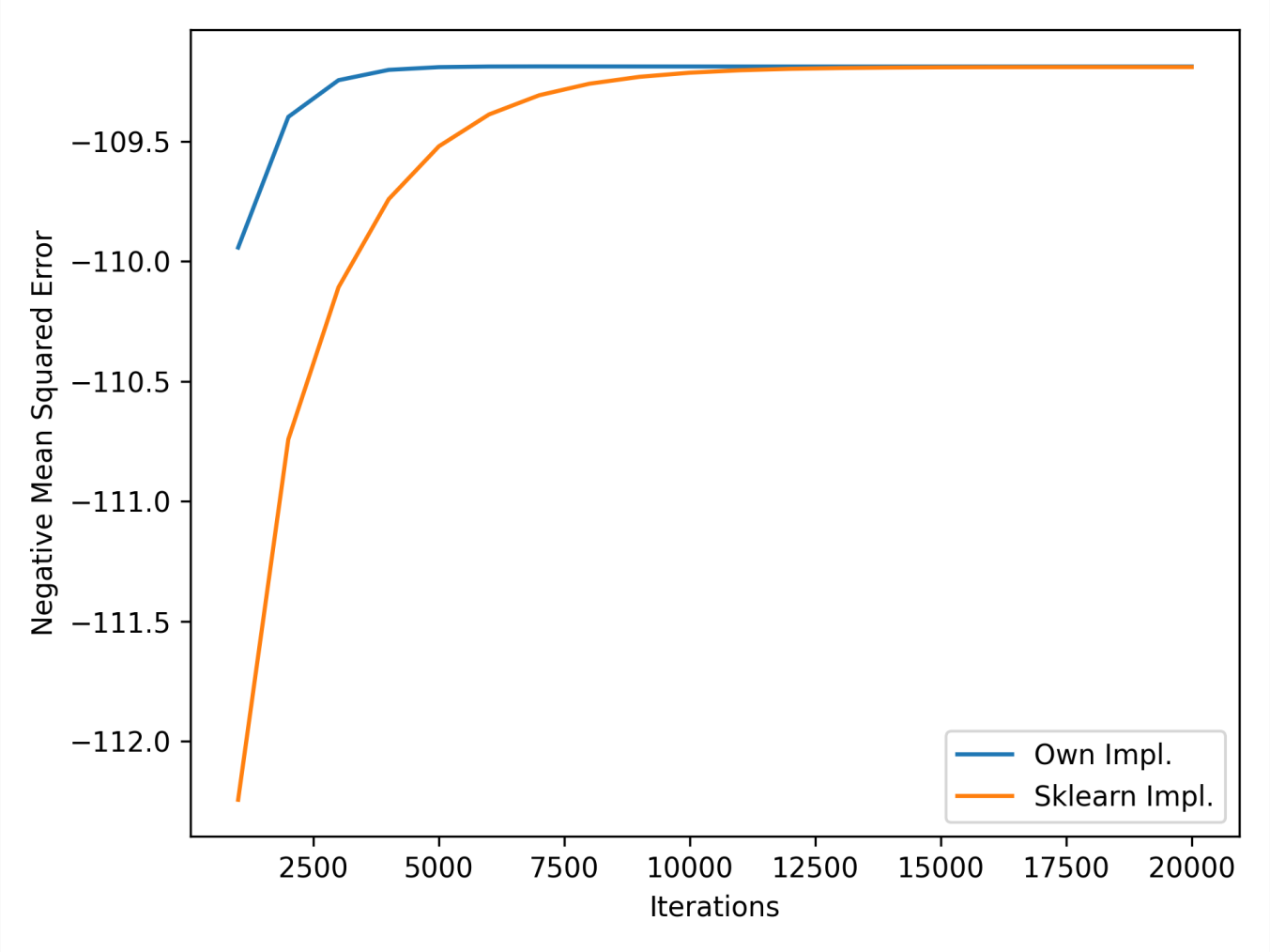

Figure 3: Mean squared error of Gradient Descent on Concrete Set

Performance on Concrete Data Set

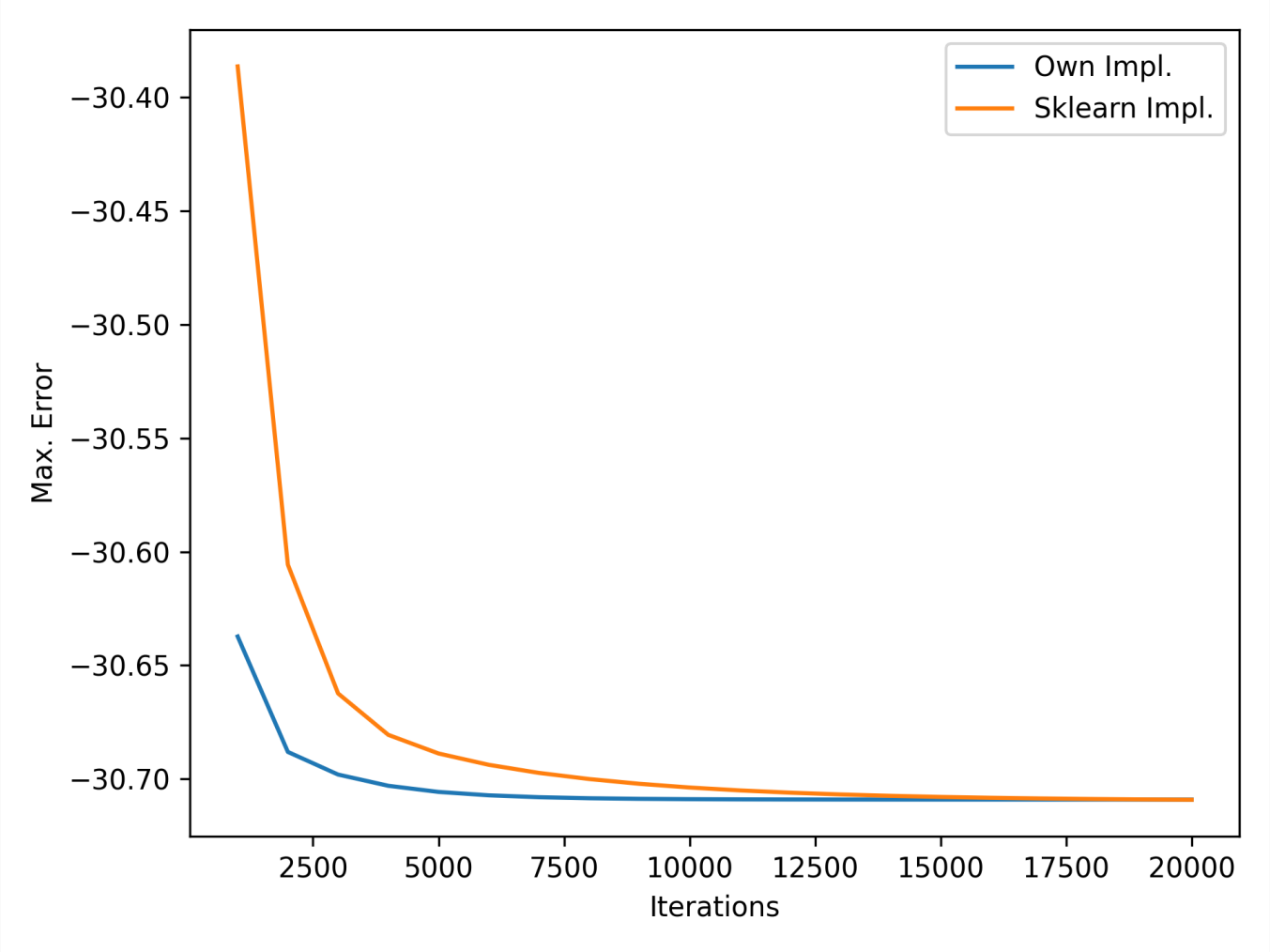

Figure 4: Max. error of Gradient Descent on Concrete Set

Performance on Life Expectancy Data Set

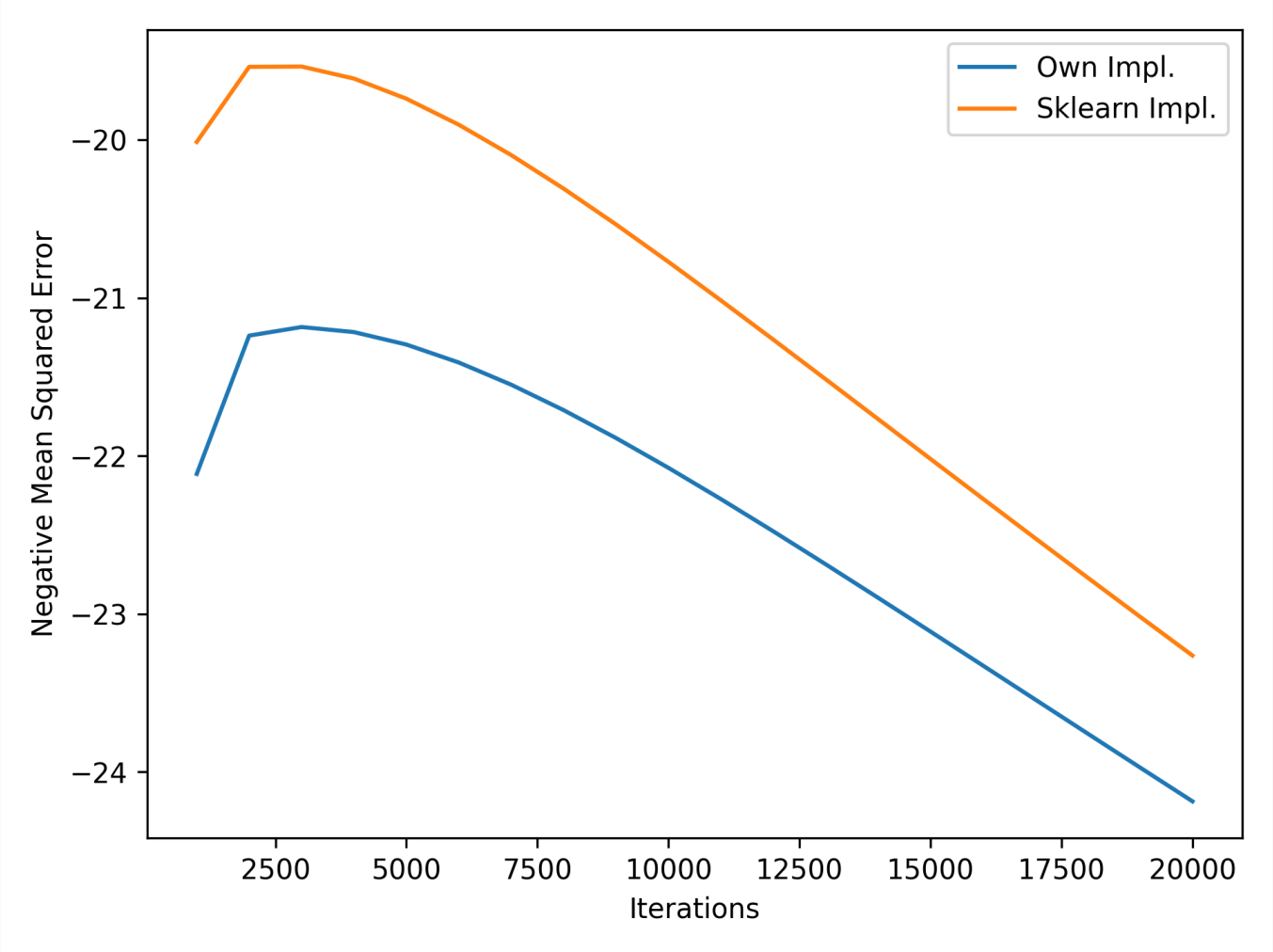

Figure 5: MSE of Gradient Descent on Life Expectancy Data Set

Performance on Life Expectancy Data Set

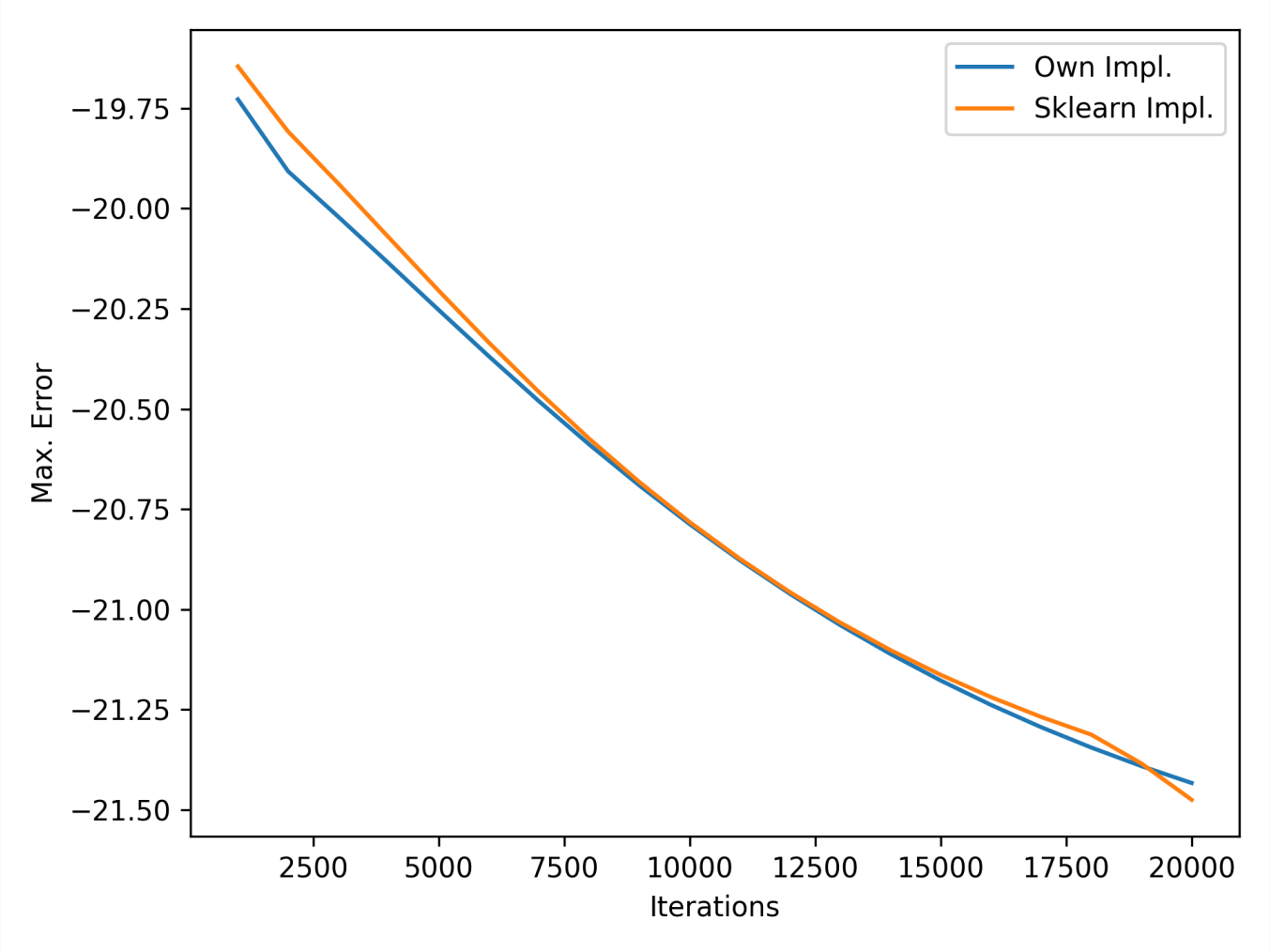

Figure 6: Max. error of Gradient Descent on Life Expectancy Data Set

Runtime on Diamonds Data Set

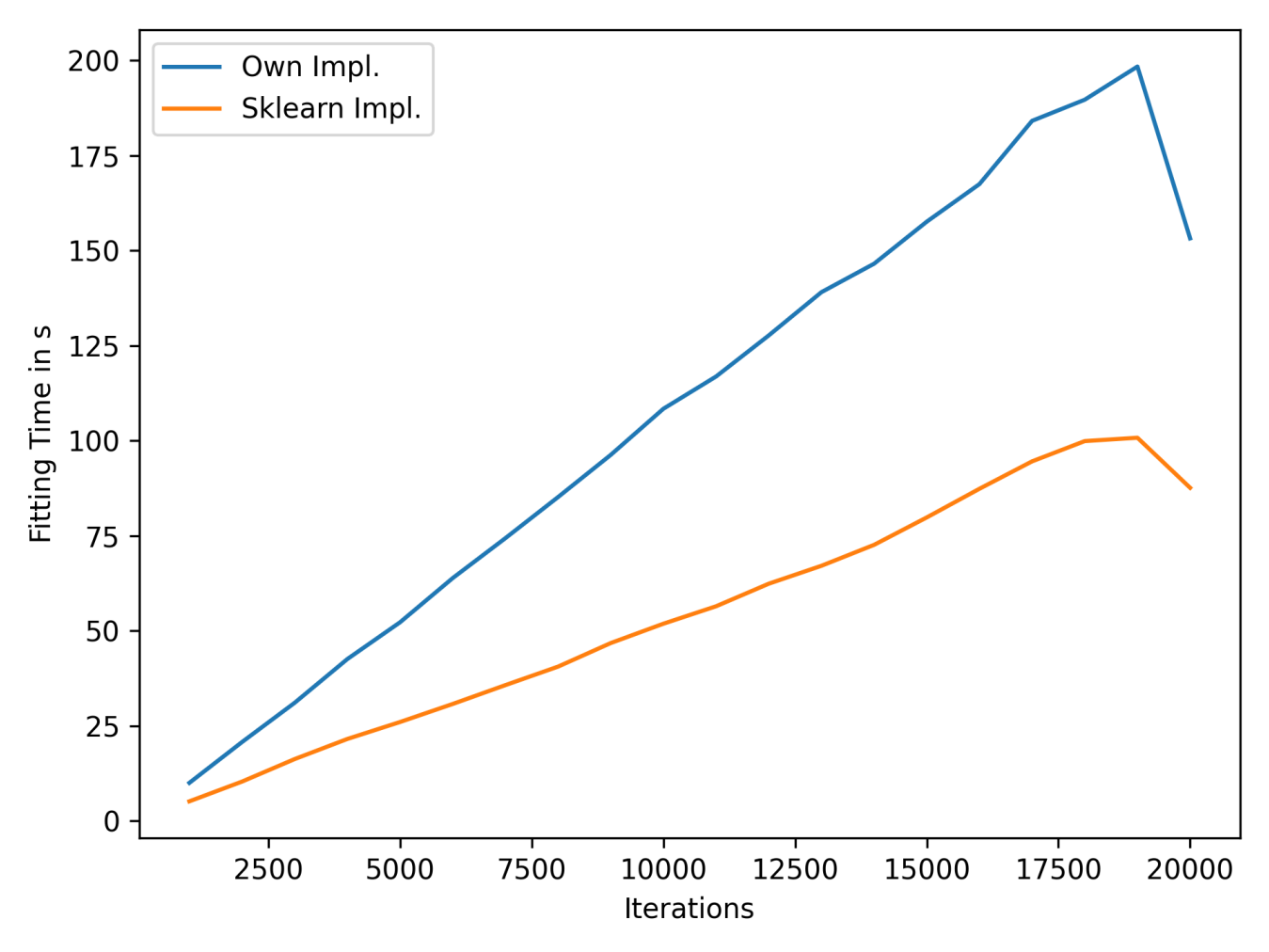

Figure 7: Runtime of Gradient Descent on Diamonds Data Set

Runtime on Concrete Data Set

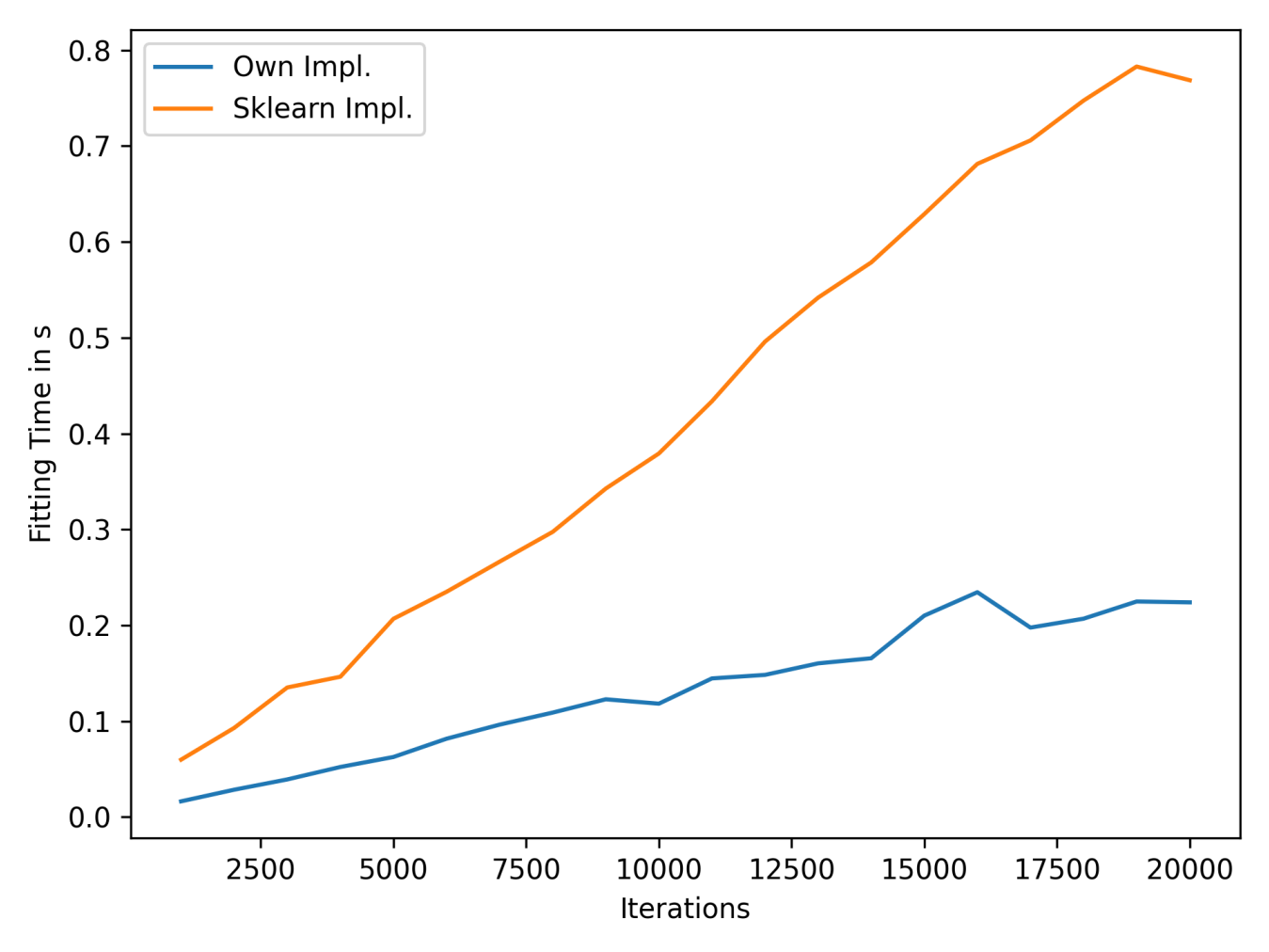

Figure 8: Runtime of Gradient Descent on Concrete Data Set

Runtime on Life Expectancy Data Set

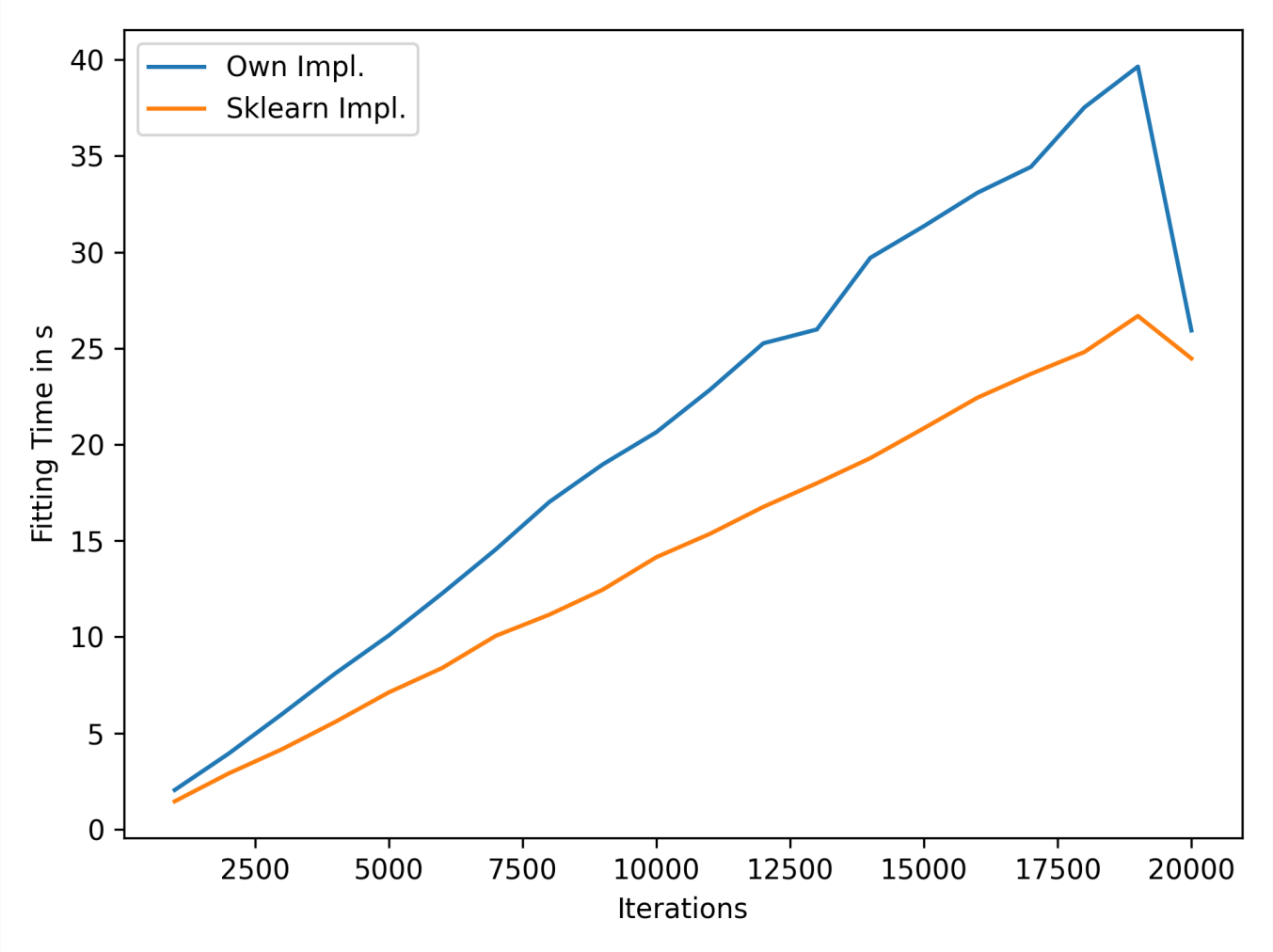

Figure 9: Runtime of Gradient Descent on Life expectancy Data Set

k-NN

k-NN

$n = \#\text{attributes}$

Distance functions (using numpy.linalg.norm in Python):

1. Manhattan distance: $d_1(x_1, x_2) = \sum_{i=1}^{n} |x_{1i} - x_{2i}|$
2. Euclidean distance: $d_2(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$
3. Chebyshev distance: $d_\infty(x_1, x_2) = \max_{i \in \{1,\ldots,n\}} |x_{1i} - x_{2i}|$

Weight:

1. Uniform $\rightarrow$ mean of predicted class of k nearest neighbours
2. Distance based: $\frac{1}{d(x_1,x_2)+\varepsilon}$ (very small $\varepsilon > 0$ to prevent division by 0)
   $\rightarrow$ weighted mean of predicted class of k nearest neighbours

k-NN: Pseudocode

Algorithm 3 k-NN-Regressor(k, distance-fct d, weight, X_t, Y_t, x_p)

neighbors = [ ] ▷ Initialize min-heap
for each row (x,y) in $(X_t,\ Y_t)$ **do** ▷ Find k nearest neighbors
    distance_tuple = $(-d(x, x_p), x, y)$
    if $|\text{queue}| < k$ **then**
        push(neighbors, distance_tuple)
    else
        pushpop(neighbors, distance_tuple)
    end if
end for
if weight='uniform' **then** ▷ Prediction
    prediction = $\frac{1}{k} \sum_{i=1}^{k} \text{neighbors}[i][2]$
else if weight='distance' **then**
    prediction = $\frac{\sum_{i=1}^{k} neighbors[i][0] \cdot neighbors[i][2]}{\sum_{i=1}^{k} neighbors[i][0]}$
end if
return prediction

Results

- Distance function:
  1. For Diamonds and Life Expectancy data set: Manhattan distance best results, but Euclidean distance also pretty good
  2. For Concrete data set: Euclidean distance best, but Manhattan also good
  3. Chebyshev distance generally not optimal for our data sets
- k value:
  1. Not the same optimal k value for all our data sets
- Scaling:
  1. Best results for all data sets obtained with standard-Scaling
  2. Min-Max-Scaler resp. without scaling not really good

Performance on Diamonds Data Set

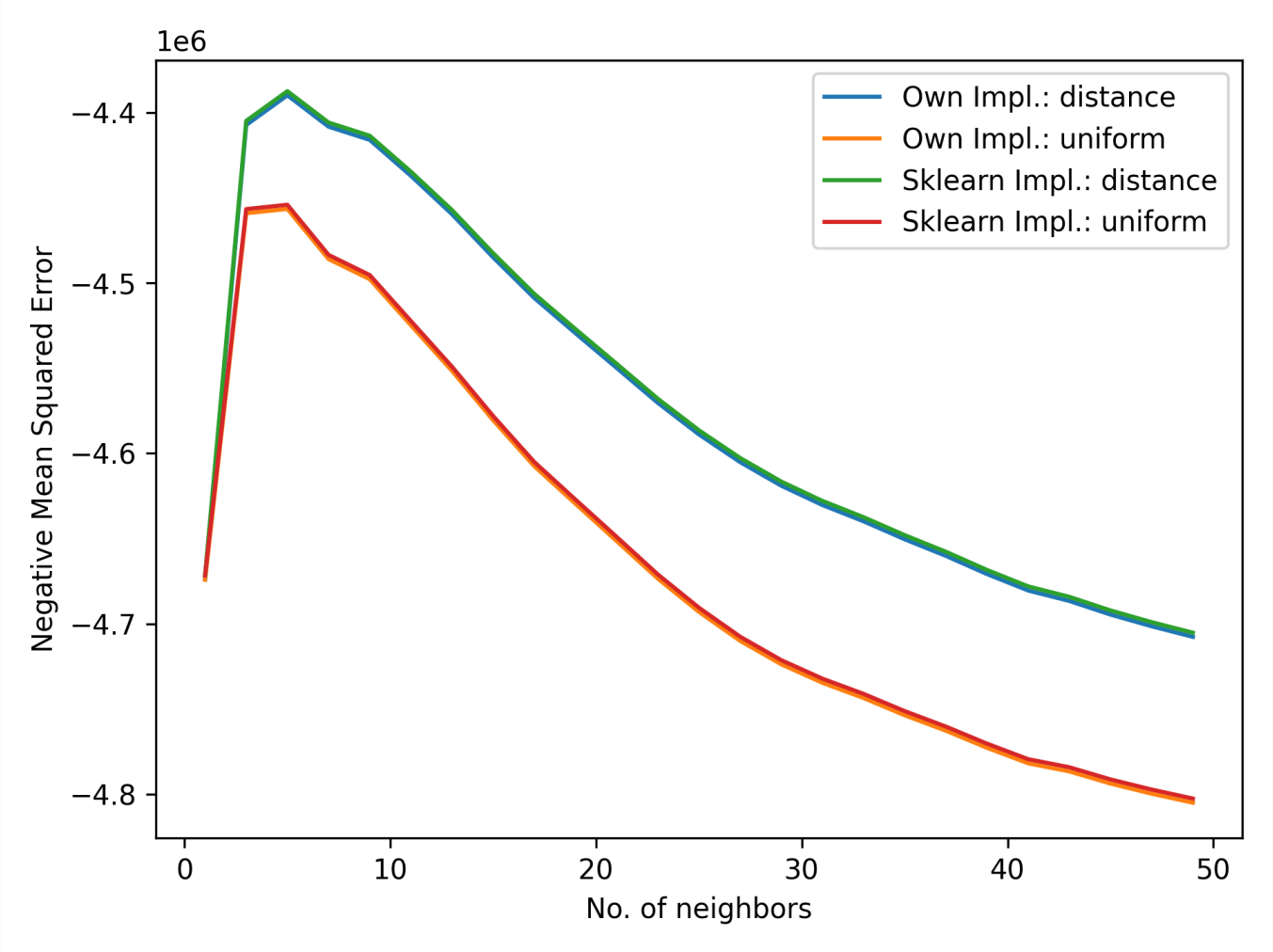

Figure 10: Mean squared error of k-NN on Diamonds Data Set

Performance on Diamonds Data Set

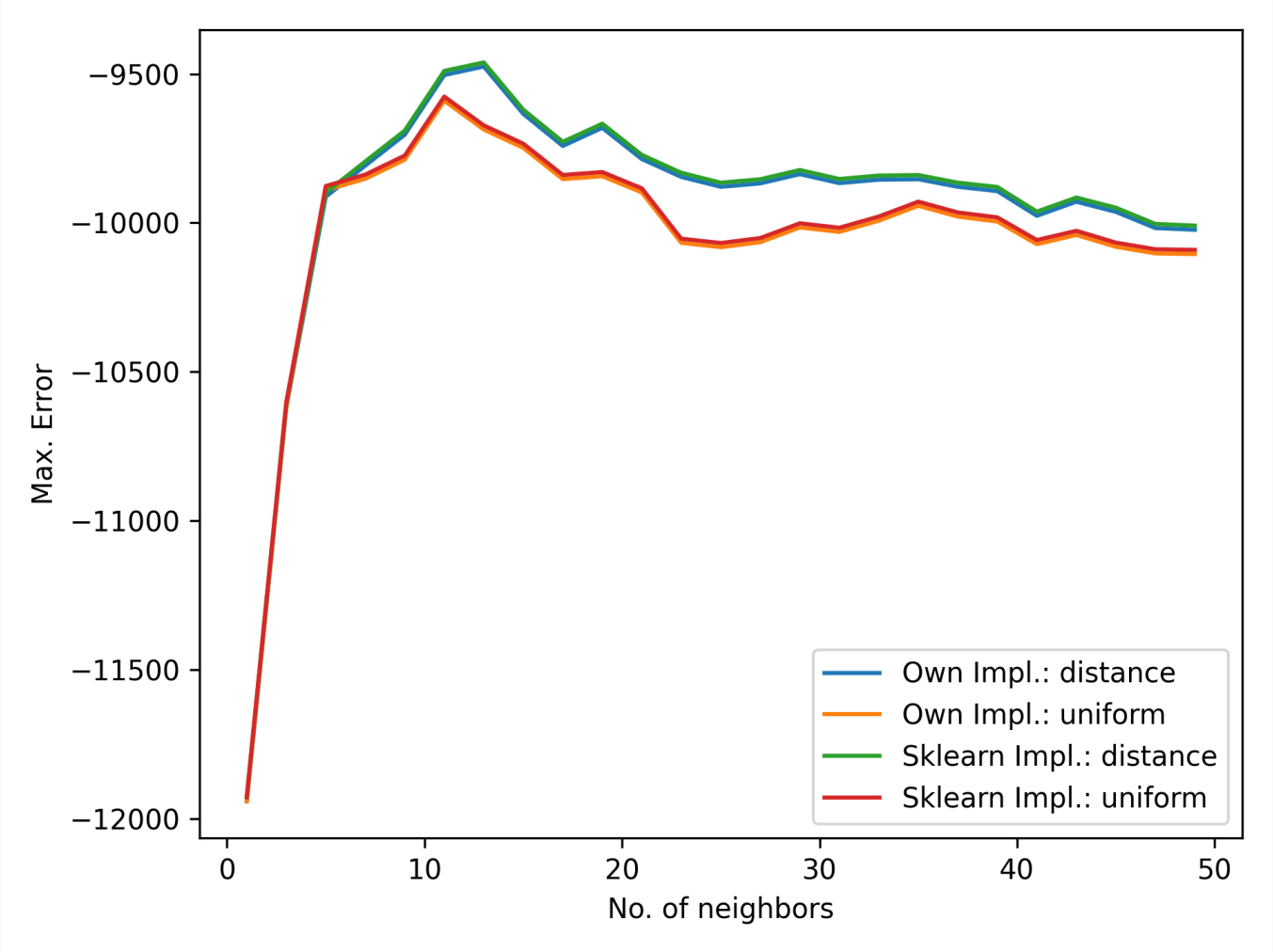

Figure 11: Max. error of k-NN on Diamonds Data Set

Performance on Concrete Data Set

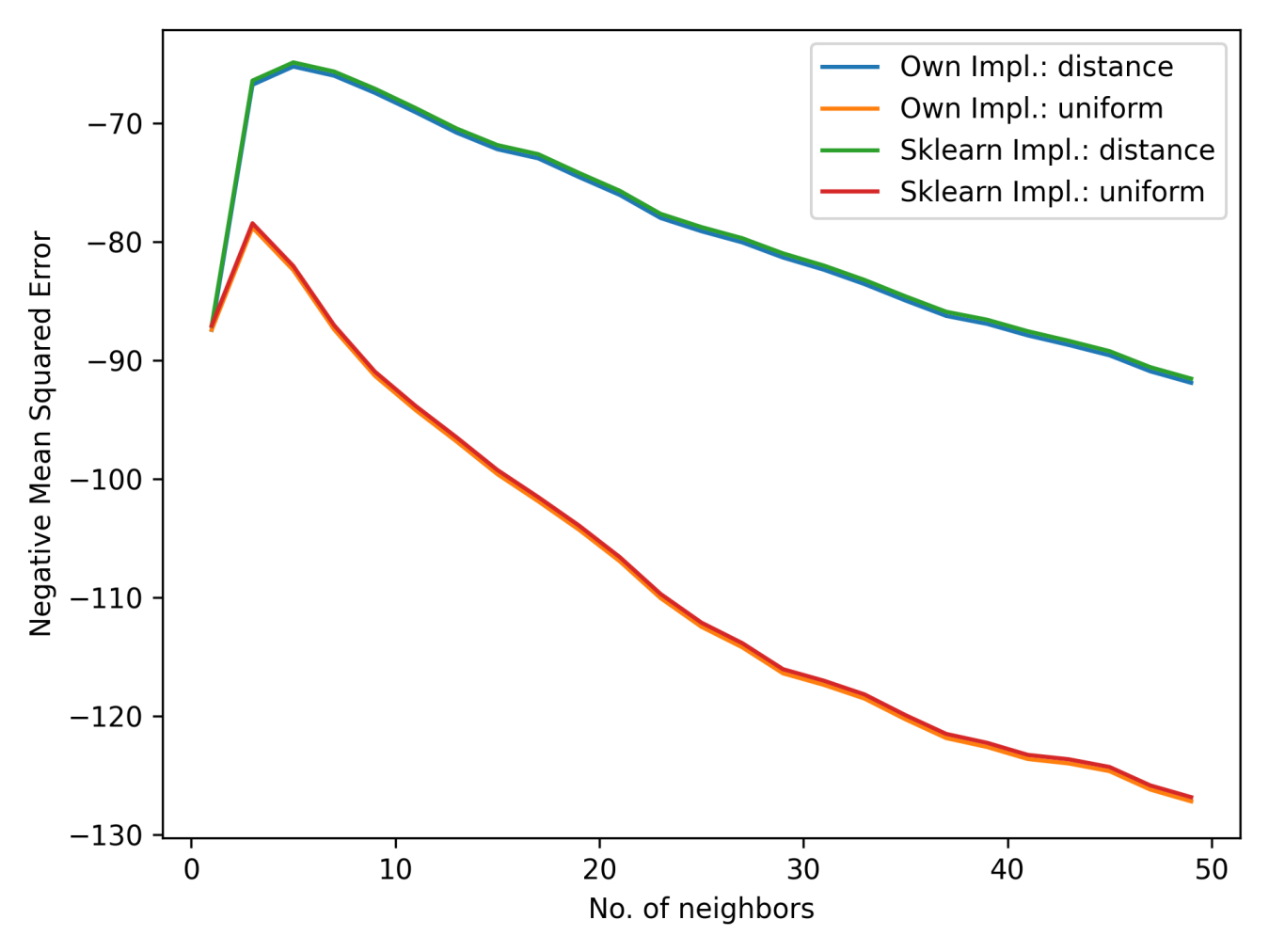

Figure 12: Mean squared error of k-NN on Concrete Data Set

Performance on Concrete Data Set

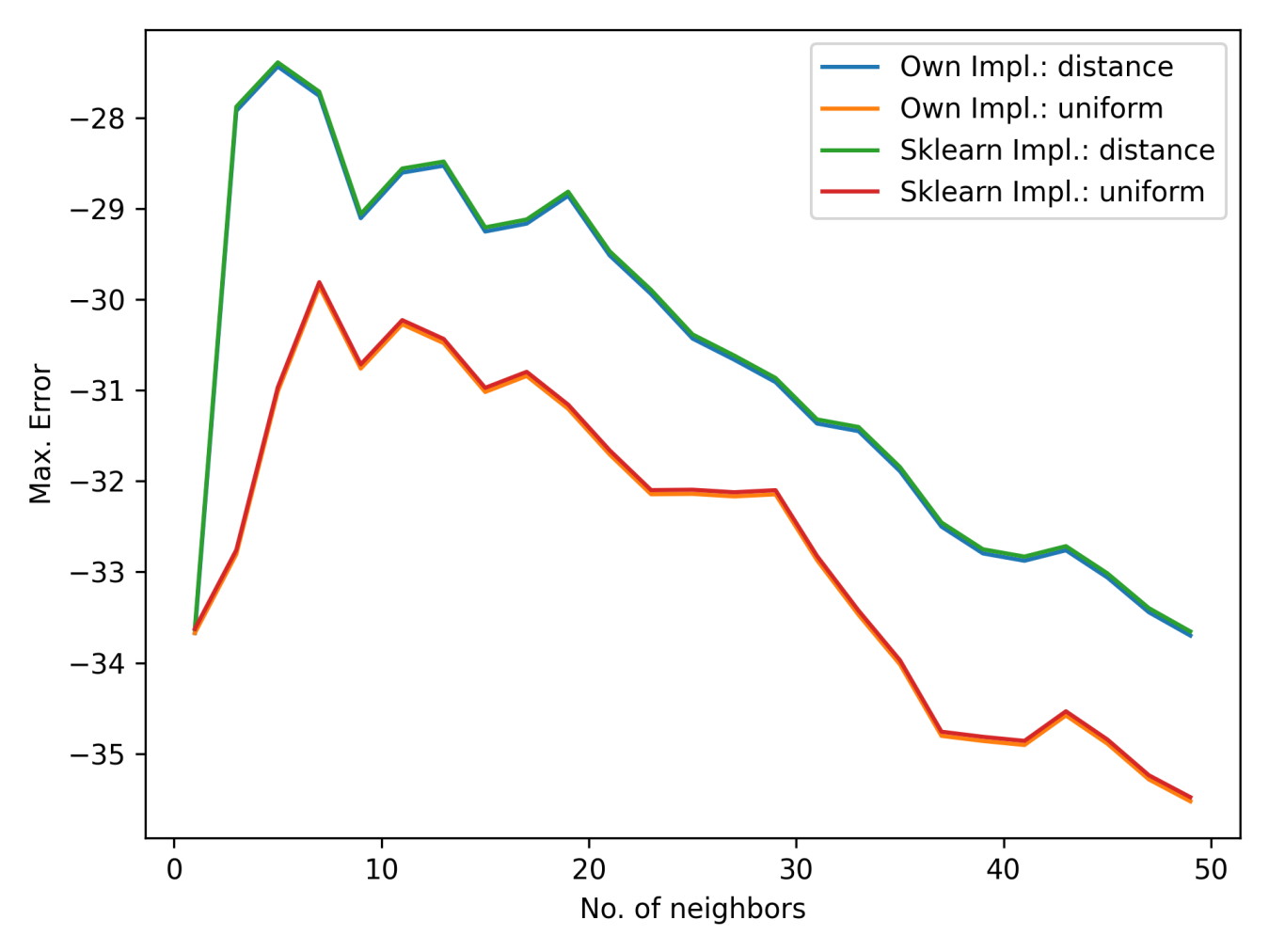

Figure 13: Max. error of k-NN on Concrete Data Set

Performance on Life Expectancy Data Set

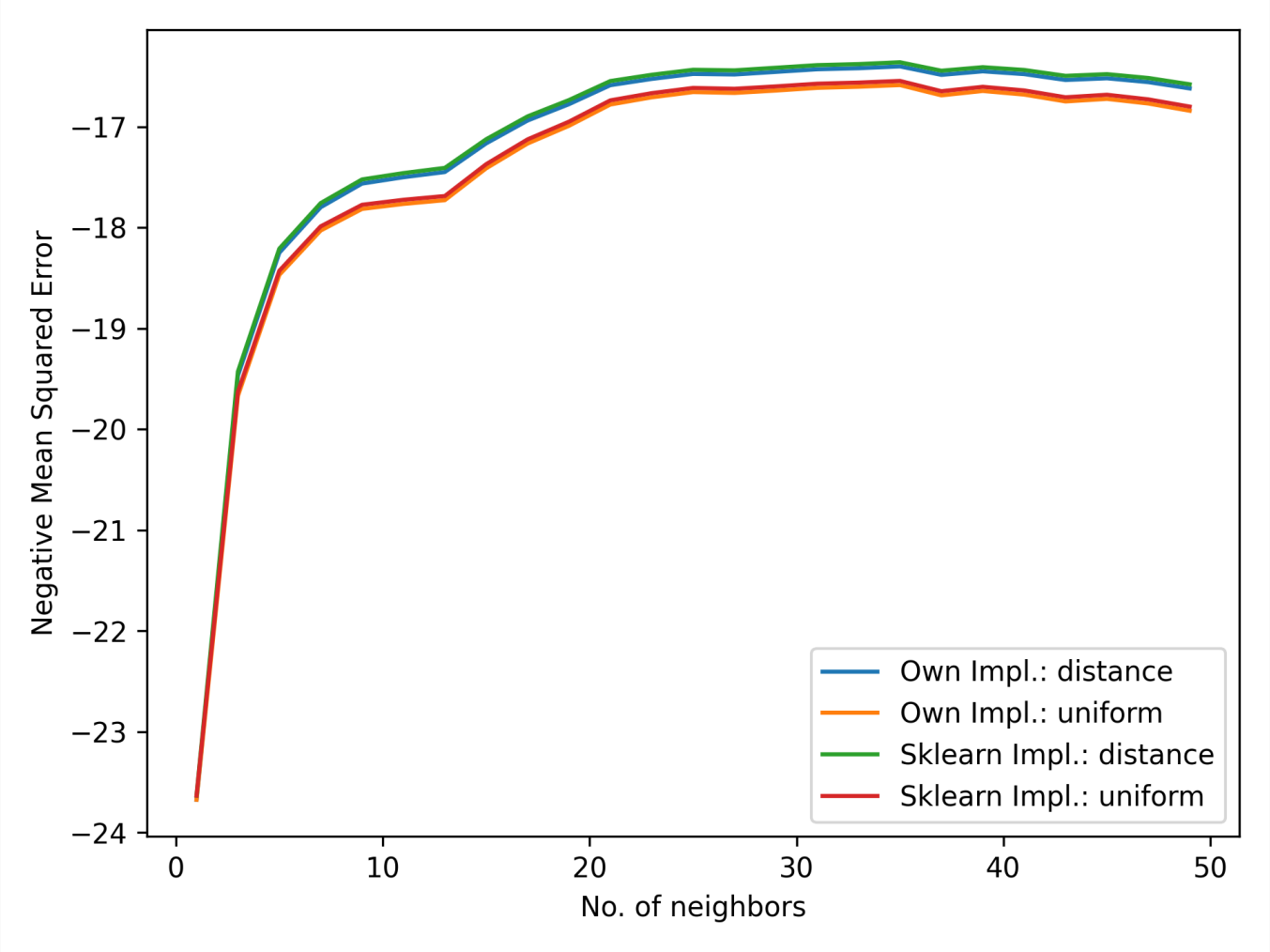

Figure 14: MSE of k-NN on Life Expectancy Data Set

Performance on Life Expectancy Data Set

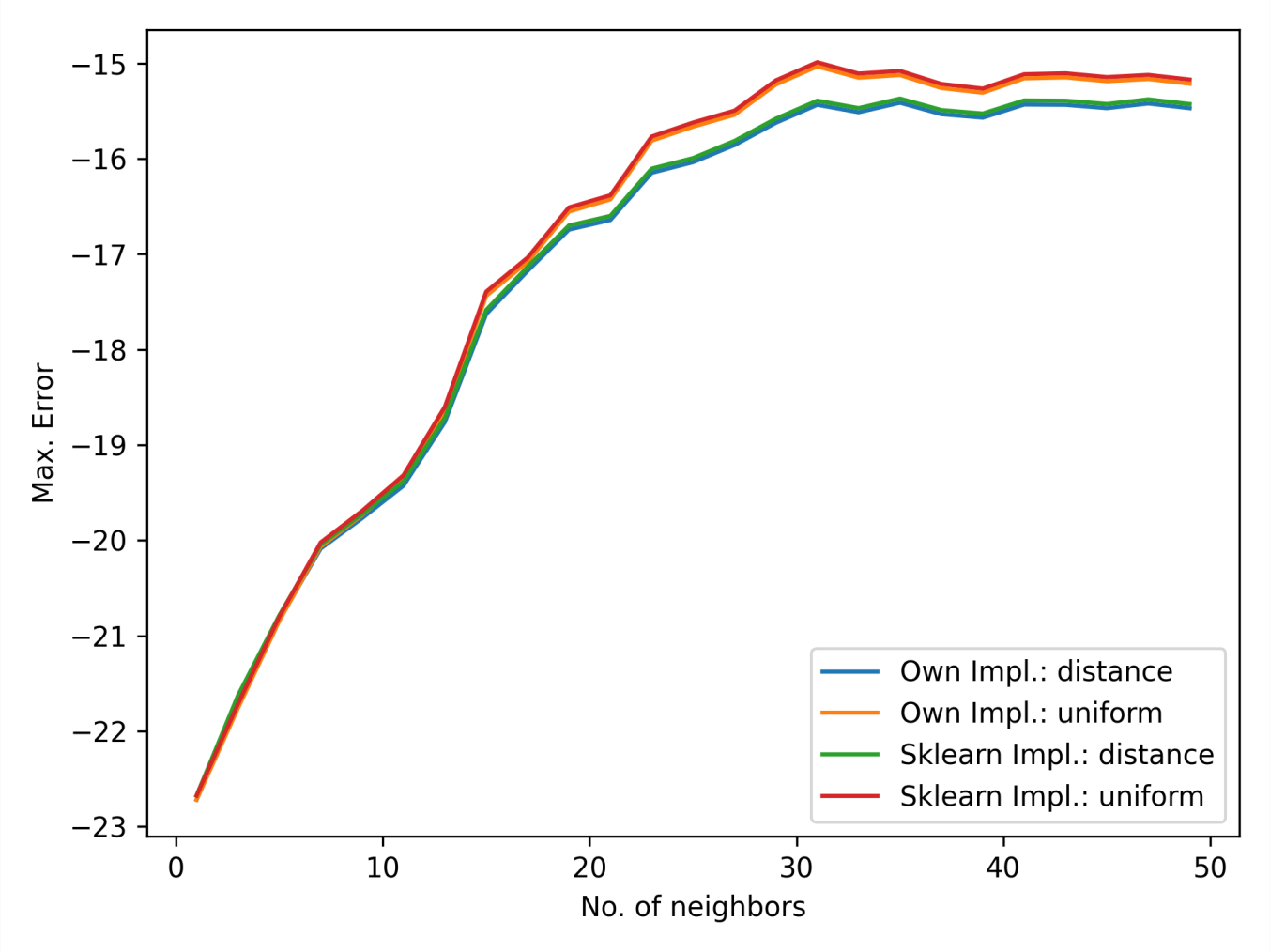

Figure 15: Max. error of k-NN on Life Expectancy Data Set

Performance on Diamonds Data Set

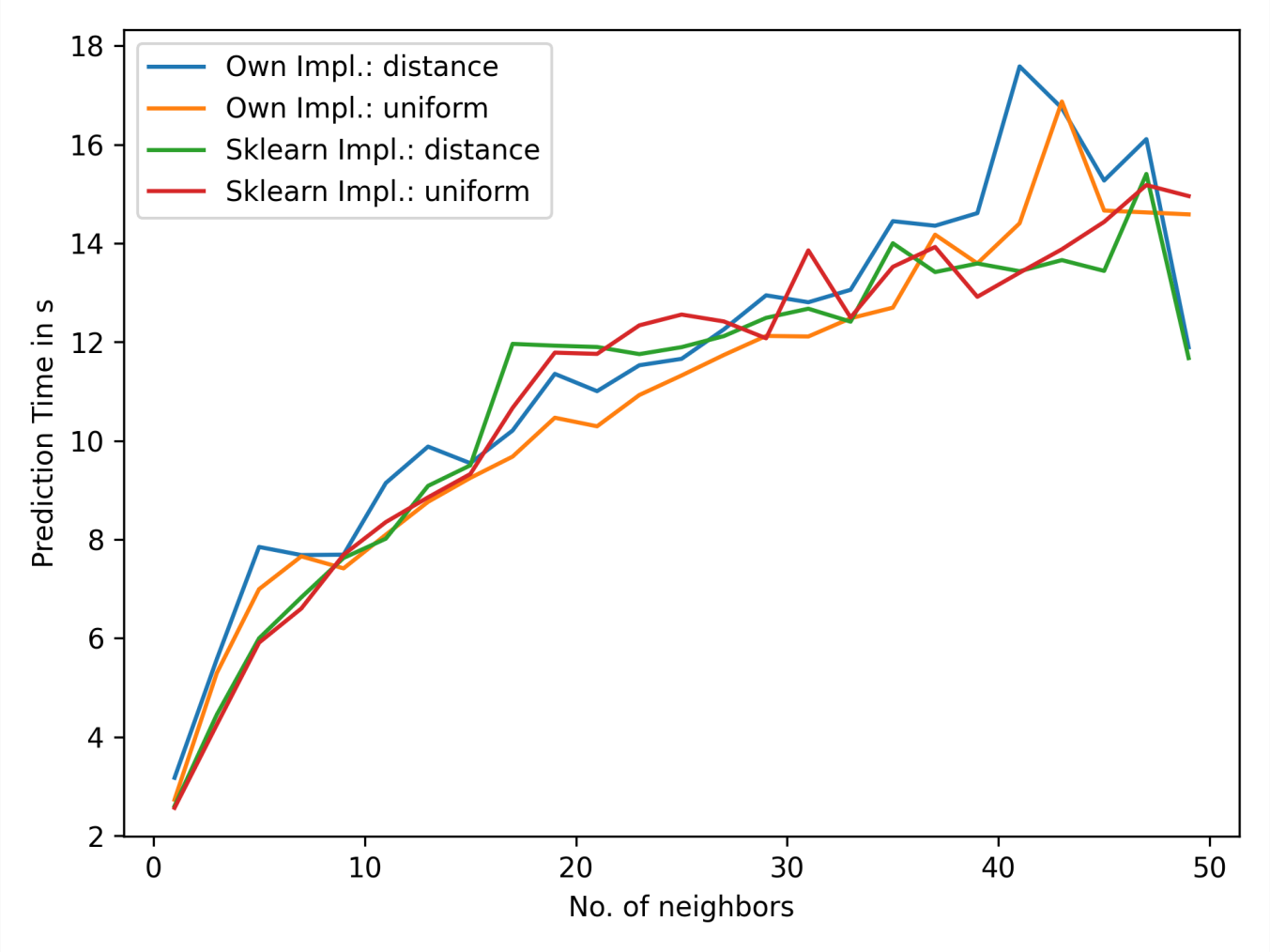

Figure 16: Prediction time of k-NN on Diamonds Data Set

Runtime on Concrete Data Set

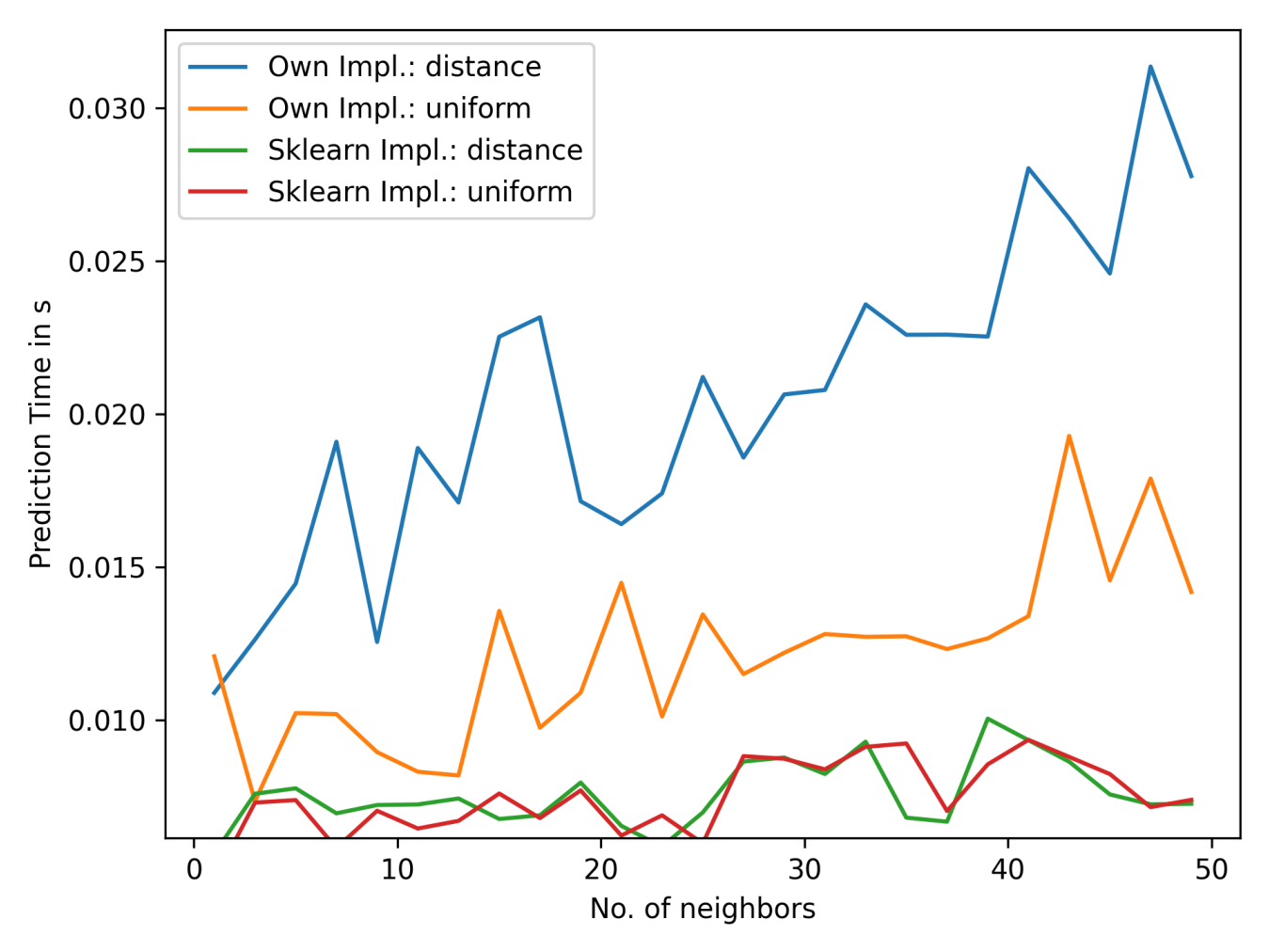

Figure 17: Prediction time of k-NN on Concrete Data Set

Runtime on Life Expectancy Data Set

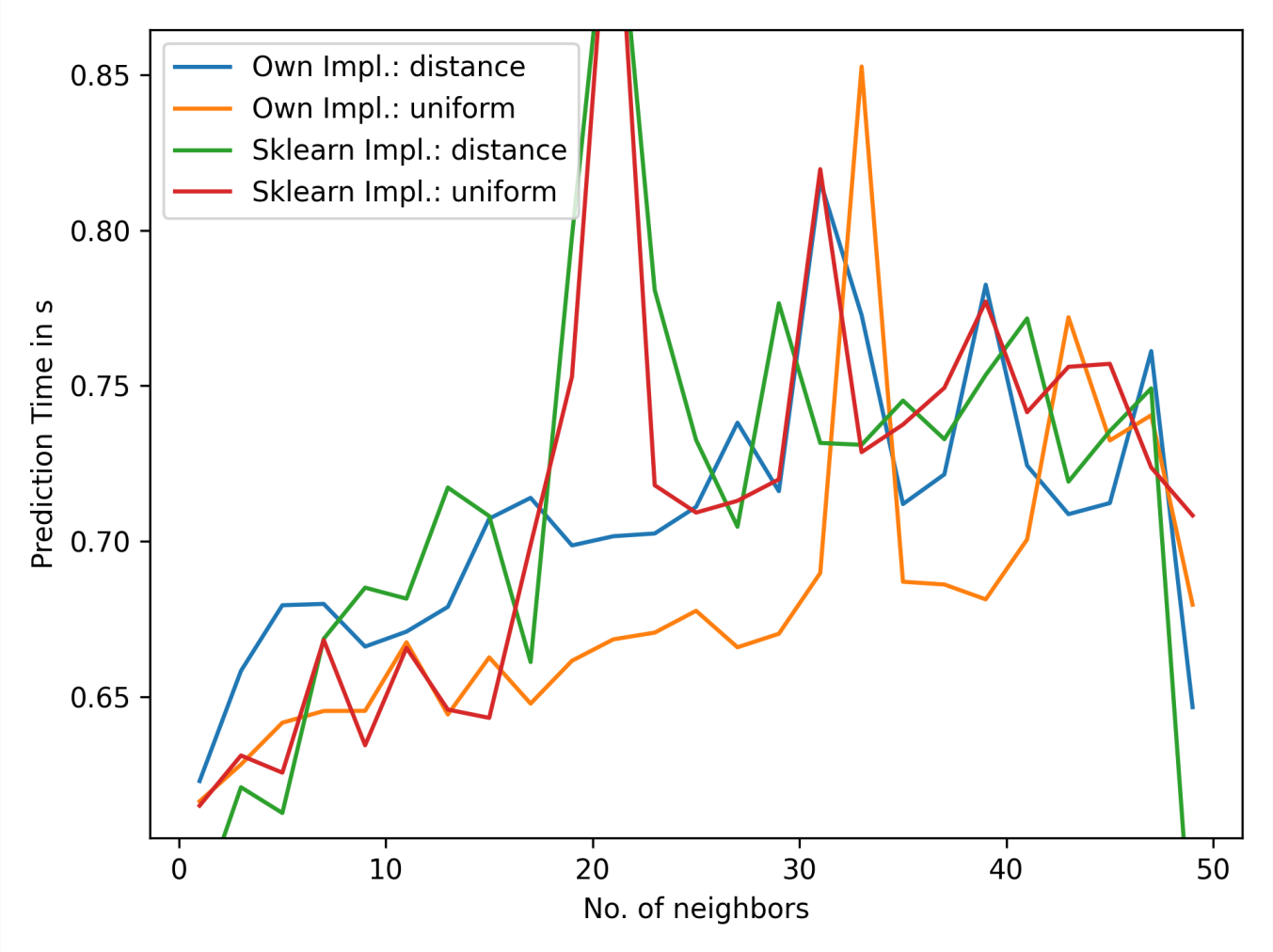

Figure 18: Prediction time of k-NN on Life Expectancy Data Set

Comparison with other algorithms

Comparison with other algorithms

	Diamonds		Concrete		Life Expectancy	
	MSE	Max. err.	MSE	Max. err.	MSE	Max. err.
k-NN	$4.4 \cdot 10^6$	$9.5 \cdot 10^3$	65	27	16	15
GD	$9.9 \cdot 10^6$	$7.5 \cdot 10^3$	109	31	21	20
DT	$3.9 \cdot 10^6$	$8.0 \cdot 10^3$	42	31	14	15
RF	$3.5 \cdot 10^6$	$6.5 \cdot 10^3$	25	23	8	12

k-NN...implemented k-NN

GD...implemented Gradient Descent

DT...Decision tree of sklearn (with default parameters)

RF... Random forest of sklearn (with default parameters)

Conclusion

Conclusion Gradient Descent

- Our implementation was worse than the sklearn implementation regarding the performance metrics, but behaved similar
- Additionally, the runtime of our implementation was higher compared to the sklearn version
- To achieve similar results as in sklearn, a higher number of iterations was needed
- Our algorithm worked the best for $\alpha = 10^{-2}$ in our tests, across all data sets
- Standard-Scaling improved the results

Conclusion k-NN

- Our k-NN implementation achieved the same results as in sklearn, also with an similar runtime
- Different optimal k values for different data sets
- Manhattan distance performed best resp. always really good
- Using the inverse distance as weights yielded better scores than uniform weights
- Standard-Scaling improved the results

References i

Diamonds Data Set: www.openml.org/d/42225

Concrete Data Set: www.kaggle.com/prokaggler/concrete-data

Life Expectancy Data Set:
www.kaggle.com/kumarajarshi/life-expectancy-who