Restricted

50.039 Theory and Practice of Deep Learning

W1-S2 Introduction and Machine Learning Reminders

Matthieu De Mari

Restricted

Restricted

About this week (Week 1)

1. What are the **typical concepts of Machine Learning** to be used as a starting point for this course?
2. What are the **different families of problems** in Deep Learning?
3. What is the **typical structure of a Deep Learning problem**?
4. What is **linear regression** and how to implement it?
5. What is the **gradient descent algorithm** and how is it used to **train Machine Learning models**?
6. What is **polynomial regression** and how to implement it?
7. What is **regularization** and how to implement it in **Ridge regression**?

Restricted

Restricted

About this week (Week 1)

8. What is **overfitting** and why is it bad?
9. What is **underfitting** and why is it bad?
10. What is **generalization** and how to evaluate it?
11. What is a **train-test split** and why is it related to **generalization**?
12. What is a **sigmoid** function? What is a **logistic** function?
13. How to perform **binary classification** using a **logistic regressor** and how is it related to linear regression?

Restricted

Restricted

About this week (Week 1)

14. What are **Neural Networks** and how do they relate to the **biology of a human brain**?
15. What is a **Neuron** in a Neural Network and how does it relate to linear/logistic regression?
16. What is the **difference** between a **shallow** and a **deep neural network**?
17. How to **implement a shallow Neural Network** manually and define a **forward propagation** method for it?
18. How to **train a shallow Neural Network** using **backpropagation**? How to define **backward propagation** and **trainer** functions?

Restricted

Restricted

Extension of multi-parameter Linear Regression

Let us now consider that the inputs consist of more than one parameter, e.g. each sample x_i consists of:

- The surface x_i^1,
- The distance to closest MRT x_i^2,
- Etc.

The linear regression can be simply transposed.

S$ 1,680,000 Negotiable

3 3 1184 sqft S$ 1,418.92 psf

Details

Property Type Executive Condominium For Sale	Floor Size 1184 sqft
Developer Tampines EC Pte Ltd	PSF S$ 1,418.92 psf
Furnishing Partially Furnished	Floor Level High Floor
Tenure 99-year Leasehold	TOP January, 2016

Restricted

Restricted

Extension of multi-parameter Linear Regression

Definition (Multi-parameter Linear Regression):

Multi-parameter Linear Regression is a model, which **assumes that there is a linear relationship between inputs and outputs.**

It therefore consists of several trainable parameters $(a_1, a_2, \dots a_K, b)$, to be freely chosen.

These will connect any input $x_i = \left(x_i^1, x_i^2, \dots, x_i^K\right)$ to its respective output y_i, with the following equation:

$$y_i \approx \sum_{k=1}^{K} a_k \, x_i^k \; + \; b$$

Restricted

Restricted

Extension of multi-parameter Linear Regression

The MSE loss then becomes:

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=1}^{K} a_k x_i^k + b - y_i\right)^2$$

And the optimization problem related to training is then:

$$(a_1^*, \ldots, a_K^*, b^*) = \arg\min_{a_1,\ldots,a_K,b} L$$

Restricted

Restricted

Extension of multi-parameter Linear Regression

We could then try to compute the new normal equation for this problem (good practice for your optimization skills, try it out!)

Or, and it is often preferable, we could then define gradient descent update rules with respect to every trainable parameter $(\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots \boldsymbol{a_K}, \boldsymbol{b})$ like before.

However, we leave this implementation for students who would like to practice. (Or it might be the homework for a later week!).

Restricted

Restricted

From Linear to Polynomial Regression

Definition (Polynomial Regression with degree K):

Polynomial Regression is a model, which **assumes that there is a polynomial relationship between inputs and outputs,** which can be defined as a **polynomial function of degree K.**

It consists of several trainable parameters $(a_1, a_2, \dots a_K, b)$, to be freely chosen.

These will connect any input x_i (e.g. the surface only) to its respective output y_i, with the same sort of polynomial equation with degree K:

$$y_i \approx \sum_{k=1}^{K} a_k \, (x_i)^k \; + \; b$$

Restricted

Restricted

The hyperplane concept

Definition (Hyperplanes in Linear Algebra):

A **hyperplane** is a subspace of one dimension less than the ambient space.

In the case of linear regression with one input feature and one output feature, the ambient space is of dimension 2 (that is 1+1).

The Linear Regression model equation, is defined as

$$y = ax + b$$

This is the equation of a line, which is a 1D subspace, and therefore a hyperplane of the 2D ambient space.

The same happens with the polynomial regression, but at a higher degree instead of just 2D/1D.

Restricted

Restricted

From Linear to Polynomial Regression

We will follow the same steps as before with linear regression, starting with a mock dataset.

- This will be different compared than what has been implemented in notebooks 1 and 2, as we will generate prices y_i as a polynomial function of the surfaces x_i.
- We will assume that the function $y = f(x)$, giving the price of an apartment with surface x, is defined as a polynomial function with degree 3.

$$y = f(x) = 100000 + 14373x + 3x^3$$

- In addition, we will add a random noise to the final pricing, with a random +/- 10% drift as before.

Restricted

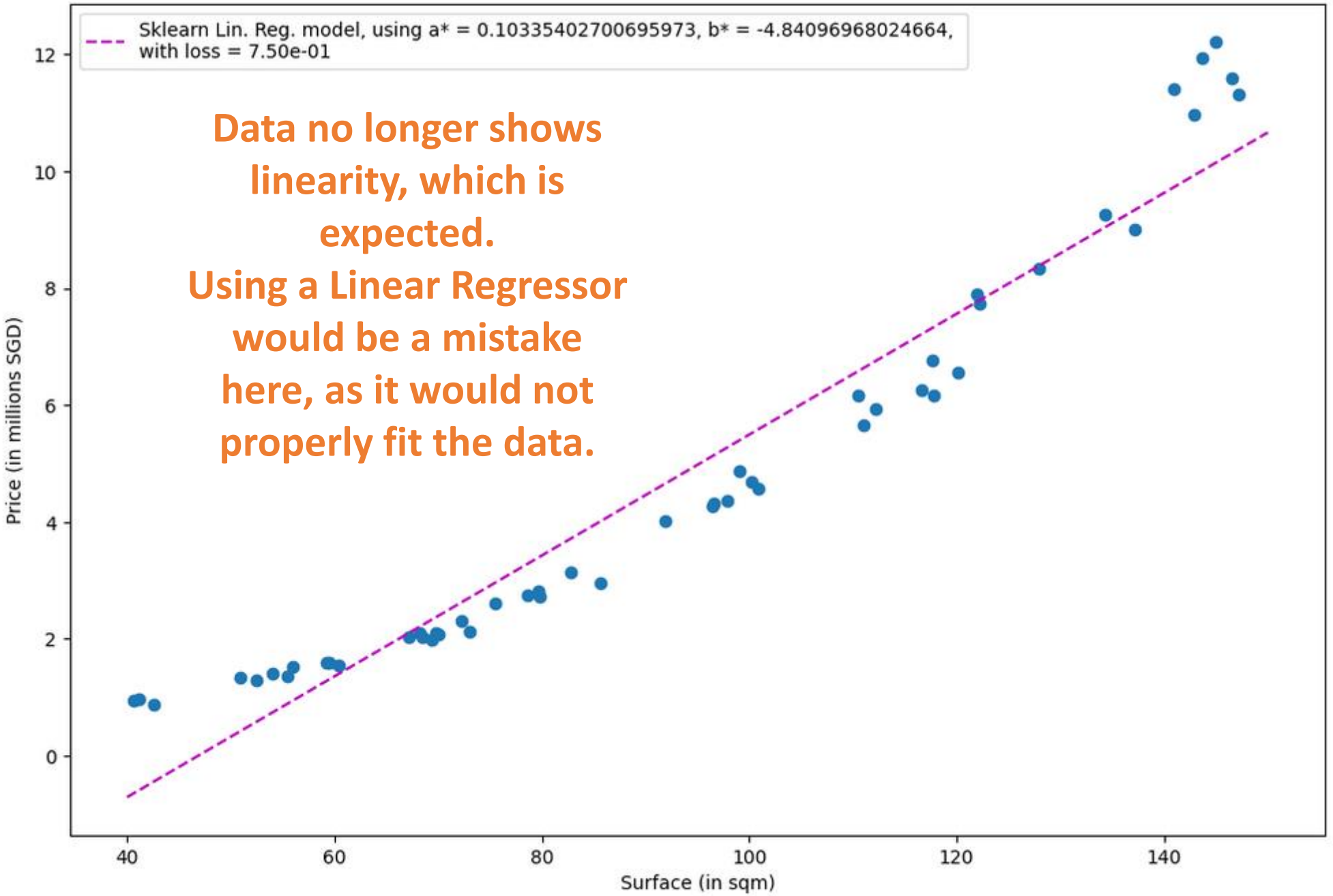
Restricted

From Linear to Polynomial Regression

```python
# All helper functions
def surface(min_surf, max_surf):
    return round(np.random.uniform(min_surf, max_surf), 2)
def price(surface):
    # Note: this has changed and is now a polynomial function.
    return round((100000 + 14373*surface + 3*surface**3)*(1 + np.random.uniform(-0.1, 0.1)))/1000000
def generate_datasets(n_points, min_surf, max_surf):
    x = np.array([surface(min_surf, max_surf) for _ in range(n_points)])
    y = np.array([price(i) for i in x])
    return x, y
def linreg_matplotlib(a, b, min_surf, max_surf, n_points = 50):
    x = np.linspace(min_surf, max_surf, n_points)
    y = a*x + b
    return x, y
def loss_mse(a, b, x, y):
    val =  np.sum((y - (a*x + b))**2)/x.shape[0]
    return '{:.2e}'.format(val)
```

Restricted

Restricted



Restricted

Restricted

Polynomial Regression with sklearn

Sklearn treats polynomial regression as a **multi-parameter linear regression**, with **polynomial features**.

Polynomial features: reworking the inputs so that

$$\tilde{X} = \begin{pmatrix} x_1 & \cdots & (x_1)^K \\ \vdots & \ddots & \vdots \\ x_N & \cdots & (x_N)^K \end{pmatrix}$$

And $\tilde{x}_{i,j} = (x_i)^j$

```
1 # Preparing polynomial features for our dataset
2 n_degree = 3
3 sk_poly = PolynomialFeatures(degree = n_degree, include_bias = False)
4 sk_poly_inputs = sk_poly.fit_transform(sk_inputs.reshape(-1, 1))
5 print(sk_poly_inputs)
```

```
[[5.24800000e+01 2.75415040e+03 1.44537813e+05]
 [1.47190000e+02 2.16648961e+04 3.18885606e+06]
 [1.20160000e+02 1.44384256e+04 1.73492122e+06]
 [7.86600000e+01 6.18739560e+03 4.86700538e+05]
 [1.17840000e+02 1.38862656e+04 1.63635754e+06]
 [1.27960000e+02 1.63737616e+04 2.09518653e+06]
 [1.11010000e+02 1.23232201e+04 1.36800066e+06]
 [8.56100000e+01 7.32907210e+03 6.27441862e+05]
 [1.17660000e+02 1.38438756e+04 1.62887040e+06]
 [6.71300000e+01 4.50643690e+03 3.02517109e+05]
 [6.81600000e+01 4.64578560e+03 3.16656746e+05]
 [4.26400000e+01 1.81816960e+03 7.75267517e+04]
 [5.08600000e+01 2.58673960e+03 1.31561576e+05]
 [7.30500000e+01 5.33630250e+03 3.89816898e+05]
 [1.10490000e+02 1.22080401e+04 1.34886635e+06]
 [7.54400000e+01 5.69119360e+03 4.29343645e+05]
 [6.04000000e+01 3.64816000e+03 2.20348864e+05]
```

Restricted

Restricted

Polynomial Regression with sklearn

We then proceed normally, assuming $\tilde{X}$ are our new inputs.

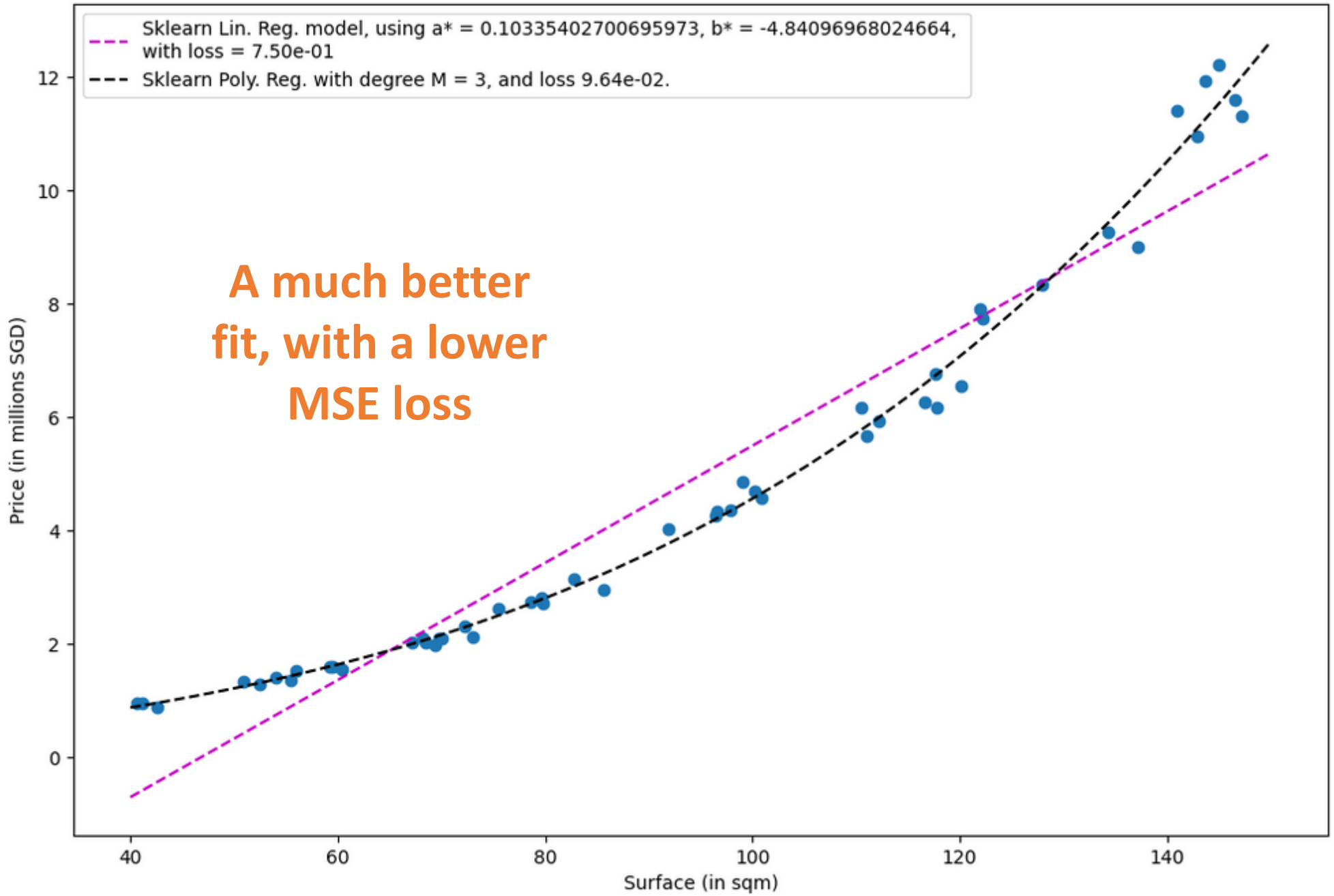
Sklearn then automatically adjusts and produces the right number of parameters for the model.

```
1  # Training a Polynomial Regressor
2  poly_reg_model = LinearRegression()
3  poly_reg_model.fit(sk_poly_inputs, sk_outputs)
4  a_sk_poly = poly_reg_model.coef_
5  b_sk_poly = poly_reg_model.intercept_
6  print(a_sk_poly, b_sk_poly)
```

```
[ 2.38010878e-02 -1.30213791e-04  3.58102988e-06] -0.09785310239196843
```

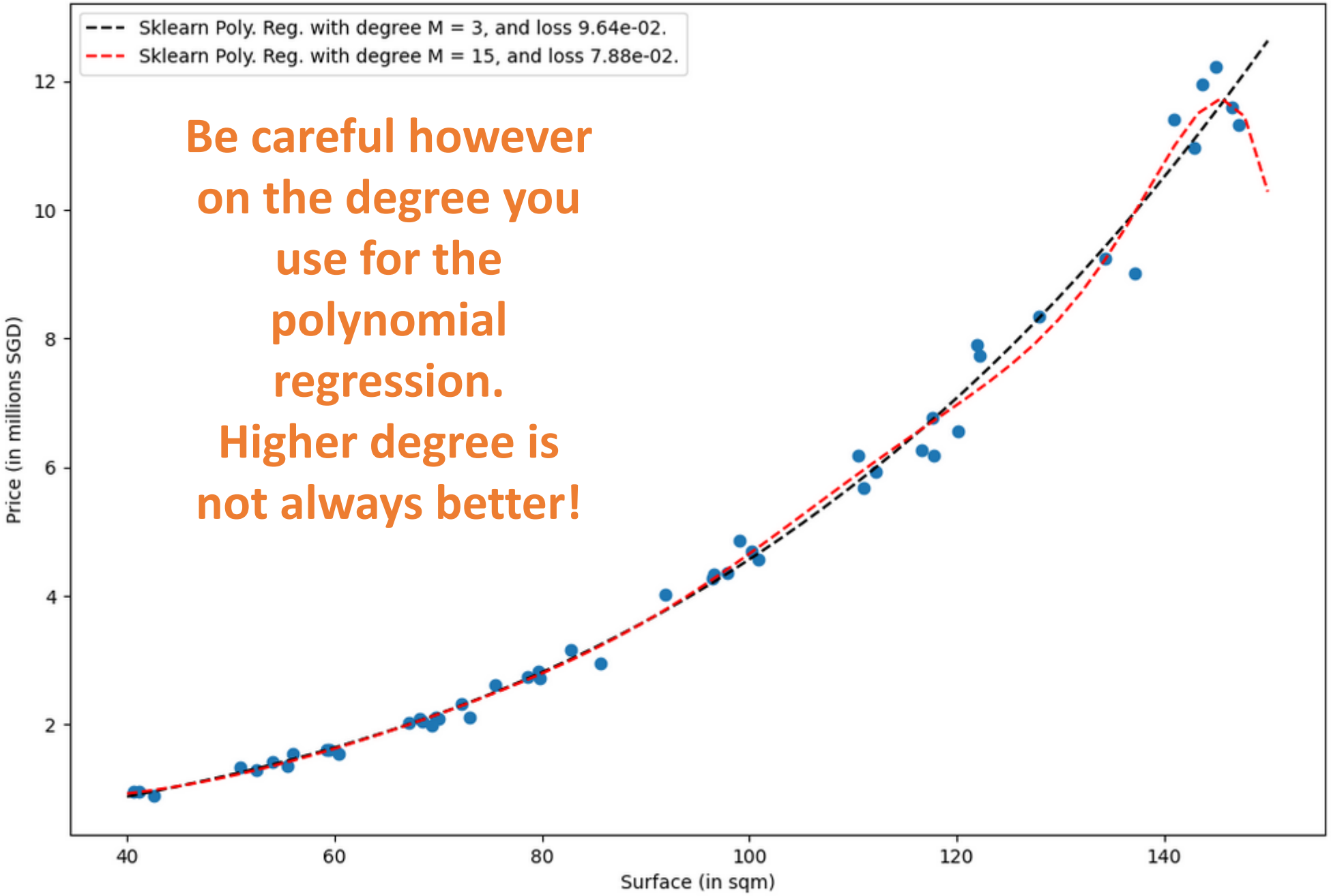
Restricted

Restricted



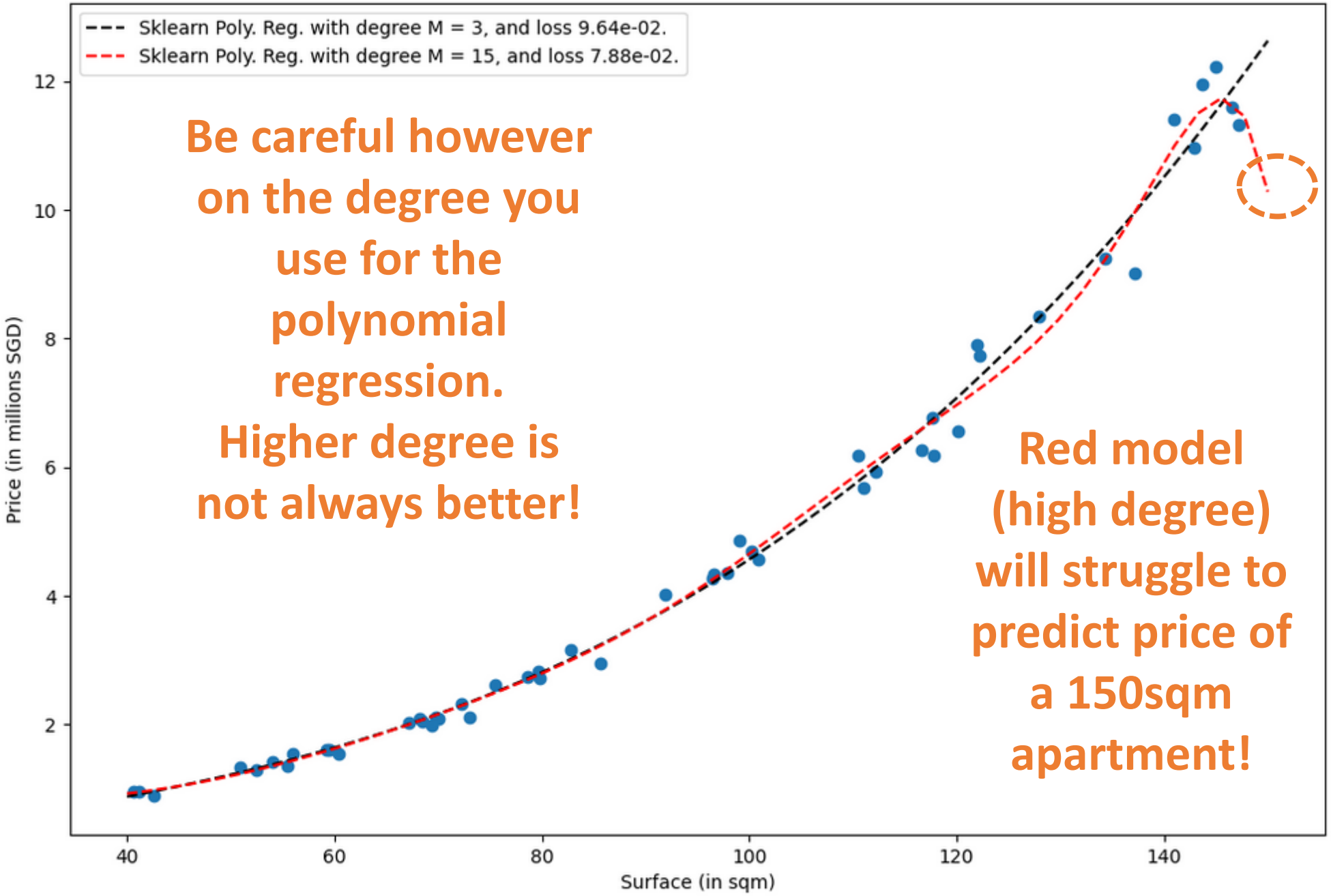
Restricted

Restricted



Restricted

Restricted



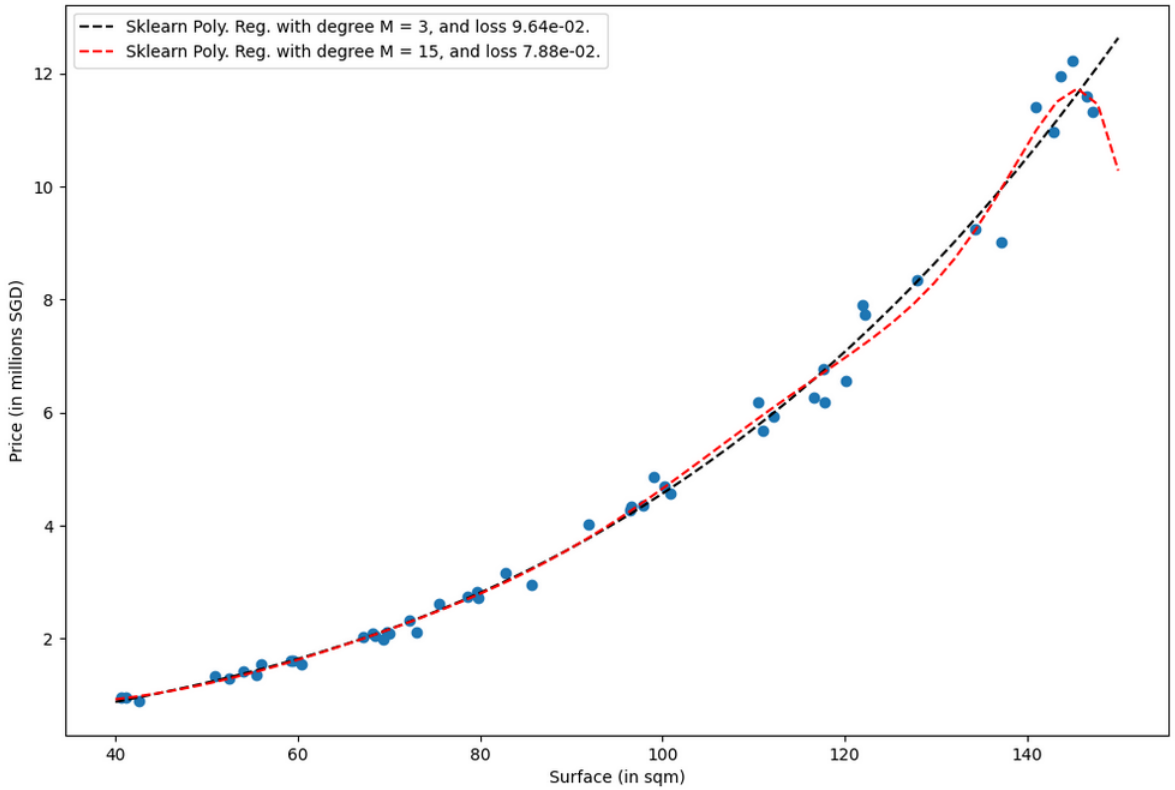
Restricted

Restricted

Overfitting vs. degree of polynomial

- The red curve shows what happens when using a degree K that is too high compared to the relationship connecting inputs and outputs (it was 3).

- This phenomenon is called **overfitting**.



Restricted

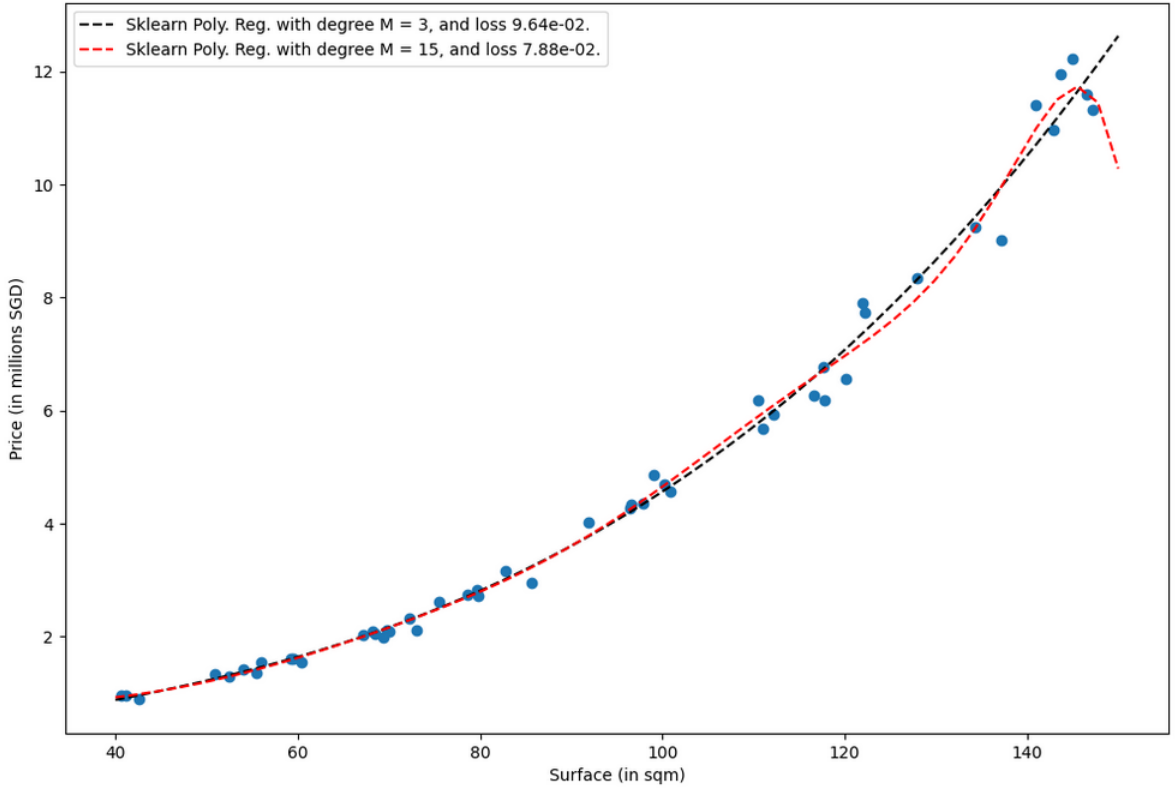
Restricted

Overfitting vs. degree of polynomial

Definition (Overfitting):

Overfitting is a phenomenon that occurs in machine learning when a model is trained too well on the training data.

As a result, it performs poorly on new, unseen data (here an apartment with 150sqm surface, which was not present in the training dataset).



Restricted

Restricted

Overfitting vs. degree of polynomial

Definition (Overfitting):

Overfitting is a phenomenon that occurs in machine learning when a model is trained too well on the training data.

As a result, it performs poorly on new, unseen data (here an apartment with 150sqm surface, which was not present in the training dataset).

This typically happens when

- a model is trained using too many features in the dataset,
- or has imbalance in the data.

It becomes too complex for the task at hand, resulting in poor **generalization** to new data.

In other words, it memorizes the noise in the training data rather than learning the correct underlying pattern.

Restricted

Restricted

Overfitting vs. degree of polynomial

Definition (Overfitting):

Overfitting is a phenomenon that occurs in machine learning when a model is trained too well on the training data.

As a result, it performs poorly on new, unseen data (here an apartment with 150sqm surface, which was not present in the training dataset).

It also typically happens if the model complexity is too high compared to the dataset complexity.

Typically, the model complexity here is the polynomial degree we use: the higher the polynomial degree, the higher the complexity of the model.

Restricted

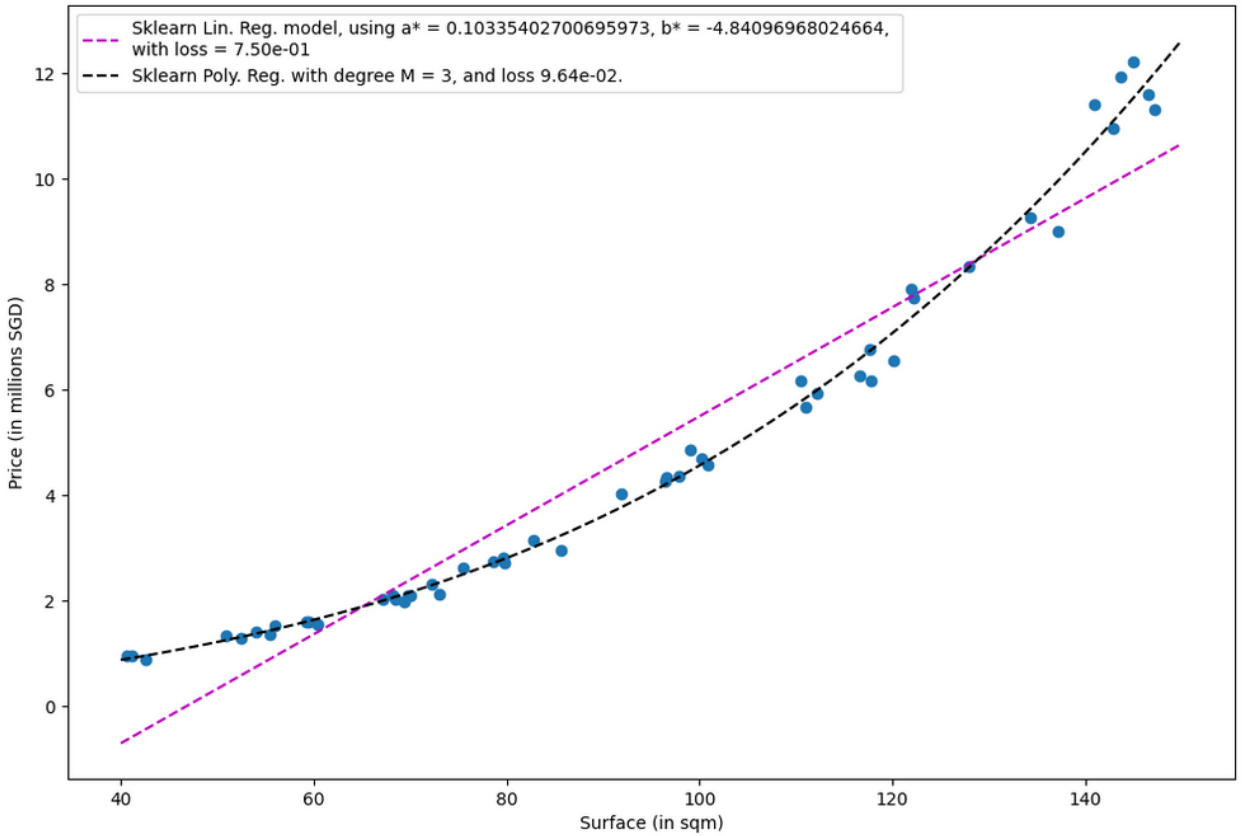
Restricted

Underfitting vs. degree of polynomial

Definition (Underfitting):

Underfitting is a phenomenon that occurs in machine learning when a model is not capable enough to capture the underlying pattern of the data.

For instance, here, Linear Regression (which is Polynomial Regression with $K = 1$) could not fit the data well.



Restricted

Restricted

Underfitting vs. degree of polynomial

Definition (Underfitting):

Underfitting is a phenomenon that occurs in machine learning when a model is not capable enough to capture the underlying pattern of the data.

For instance, here, Linear Regression (which is Polynomial Regression with $K = 1$) could not fit the data well.

As the opposite of overfitting, underfitting will typically happen when a model

- is trained with too few features
- or too little data,
- or when the model is not powerful enough to capture the complexity of the task.

Restricted

Restricted

Generalization

Definition (Generalization):

Generalization is considered the **"holy grail" or ultimate goal of any machine learning model**.

Even though the model was trained on seen data we collected, the purpose of a machine learning model is to **generalize**, i.e. to **make accurate predictions on new, unseen data**.

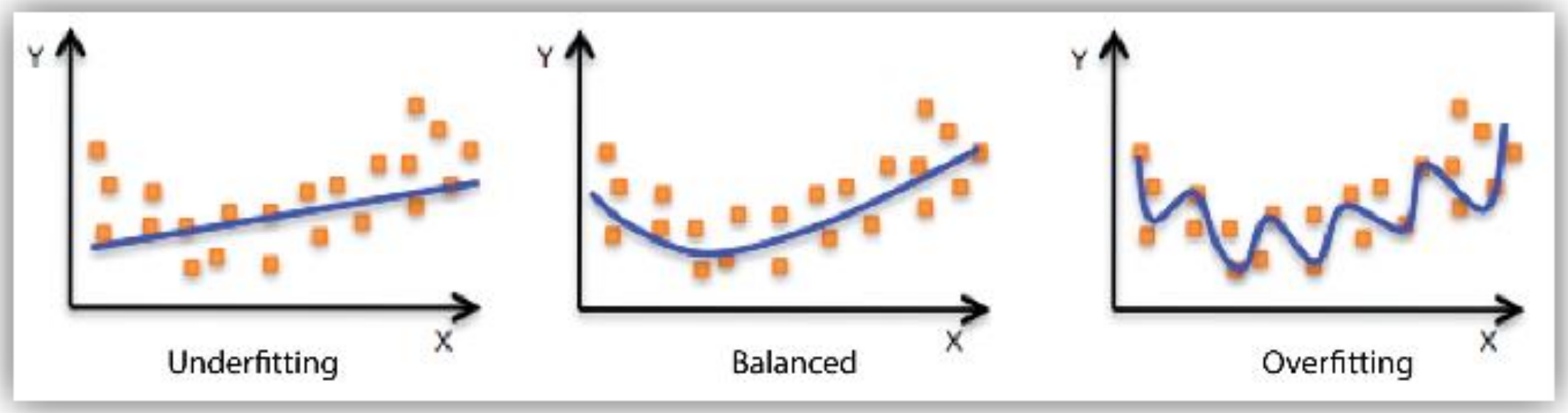
If a model can make accurate predictions on data it has not seen before, it is said to have **generalized well**, and can then be used in the real world for solving the problem it was designed to solve.

Restricted

Restricted

Generalization vs. Overfitting/Underfitting

- In general, having a model that is **underfitting** or **overfitting** will lead to poor **generalization**.
- We would very much prefer to fit the data “just right”.



Restricted

Restricted

While we are at it…

In general, we want to:

- Train the model on the data we collected for this task,
- Confirm that the model can generalize and make accurate predictions on new, unseen data, after training.

Problem: We would prefer to test the model before releasing it in the wild (what if it is wrong?)

Problem #2: Testing its generalization capabilities on the same data that was used for training would NOT confirm generalization.

Solution: Use some of our seen data to train the model, and some of the data (that has not been used for training and is therefore unseen to the model yet) for evaluating its generalization.

Restricted

Restricted

Train and test split

Definition (Train and Test Split):

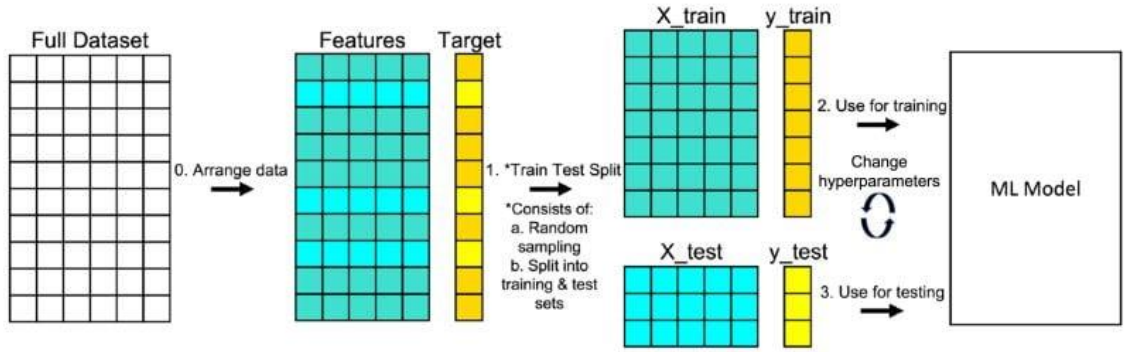
Train and test split is a method used in machine learning to divide a dataset into two subsets:

- The **training dataset** is used to train the model.
- The **test dataset** is used to evaluate the model performance after training.

The idea is to use a portion of the dataset for training and a separate portion for testing so that the model can be evaluated on unseen data.

The typical split is to use 80% of the data for training and 20% for testing.

This can however vary depending on the specific use case and the size of the dataset.



Restricted

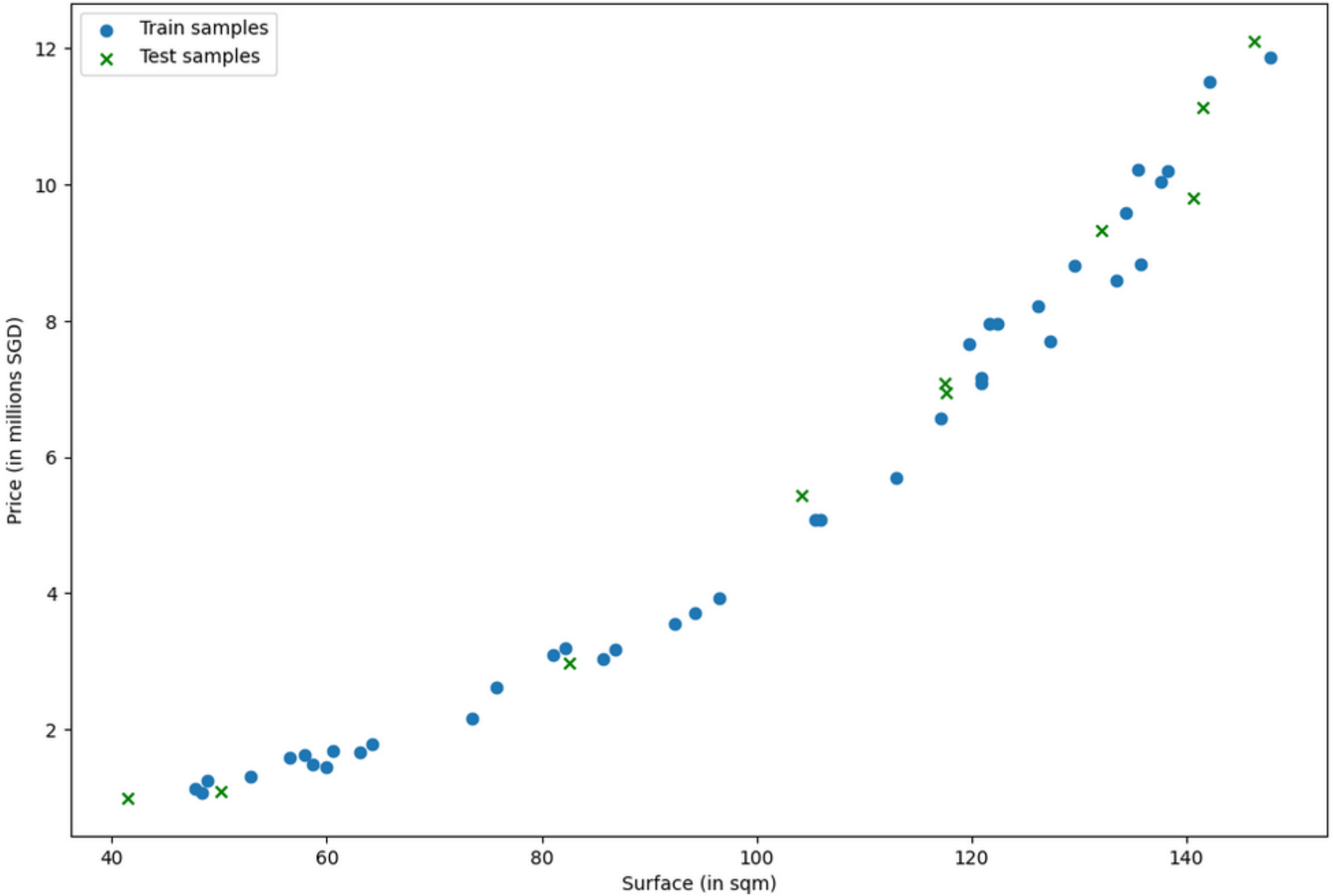
Restricted

Train and test split

```
# 80% of the samples will be used for training,
# and the remaining 20% will be used to evaluate generalization/overfitting.
ratio_train = 0.8
split_index = int(n_points*ratio_train)
# Training inputs and outputs
train_inputs = inputs[:split_index]
train_outputs = outputs[:split_index]
# Testing inputs and outputs
test_inputs = inputs[split_index:]
test_outputs = outputs[split_index:]
# Display
print(train_inputs)
print(train_outputs)
print(test_inputs)
print(test_outputs)
```

```
[ 86.83 129.6  120.89 135.48  82.17 147.74 138.25  63.07 121.6  112.95
 137.55 134.38 122.42 135.72  60.54  75.81  81.02 127.31  56.62  58.69
  48.93  73.57 126.16  57.92  47.77 117.12  59.91 105.88  85.68  96.49
  64.27 119.81 133.44 142.18 120.95  92.42  94.22 105.4   48.36  52.92]
[ 3.171633  8.805667  7.166722 10.224944  3.195151 11.87205  10.206911
  1.676214  7.954078  5.696169 10.049157  9.588631  7.968783  8.827882
  1.693317  2.6239    3.094831  7.700582  1.597057  1.489182  1.247223
  2.166055  8.212138  1.627403  1.12722   6.560241  1.450404  5.072051
  3.031968  3.938955  1.782147  7.652644  8.58658  11.518957  7.088832
  3.563134  3.702923  5.087156  1.071143  1.308982]
[146.31 104.17  50.17  41.5  132.06 140.63 117.5   82.57 117.63 141.56]
[12.111945  5.438864  1.088438  0.998857  9.333922  9.801835  7.095279
  2.982004  6.953392 11.129658]
```

Restricted



Restricted

Restricted

Training the Polynomial Regressor, again

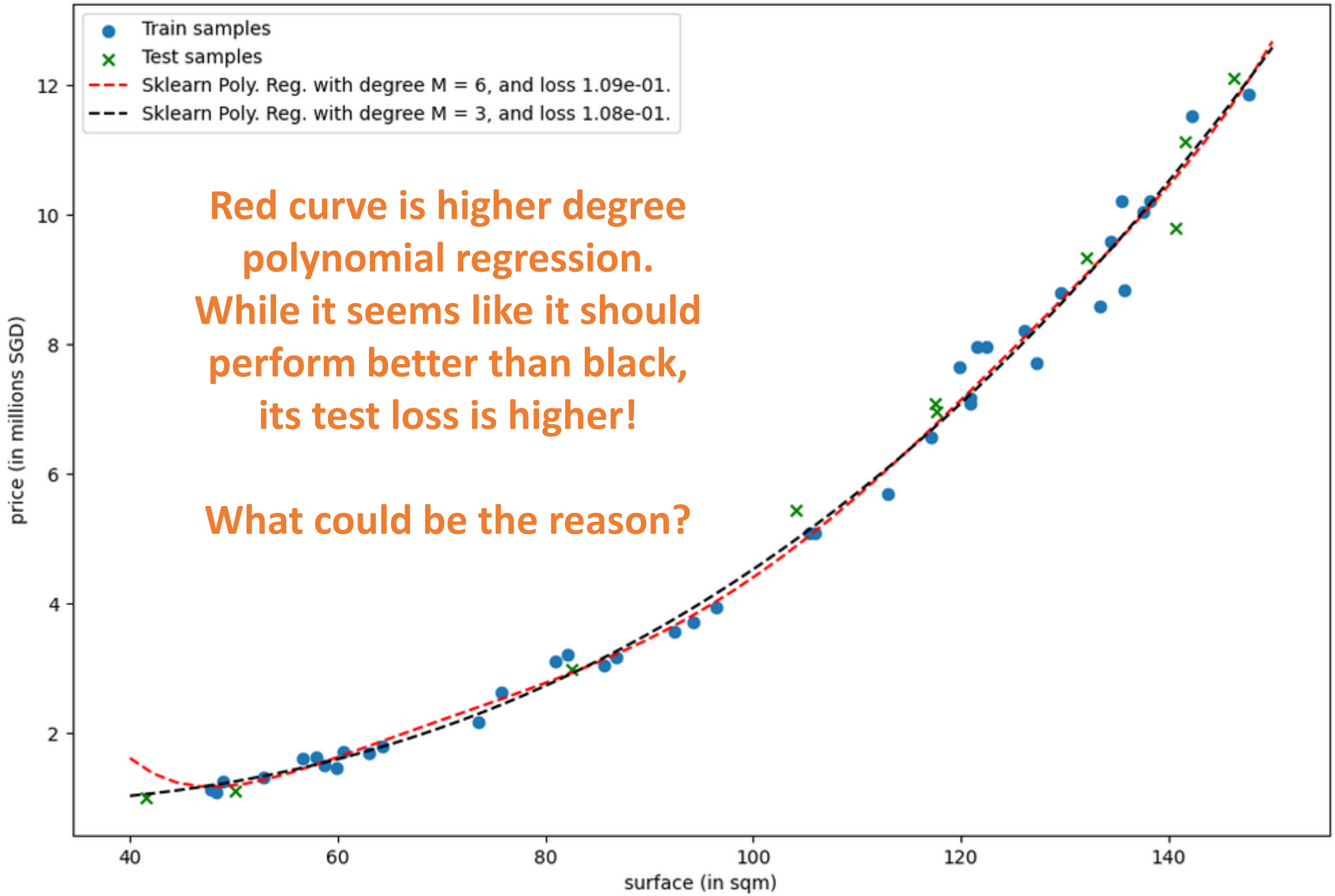
- This time, we only train the polynomial regressor on the training data.
- After training, evaluate loss using the test data.

```
# Training a Polynomial Regressor
n_degree = 6
sk_inputs = np.array(train_inputs).reshape(-1, 1)
sk_outputs = np.array(train_outputs)
sk_poly = PolynomialFeatures(degree = n_degree, include_bias = False)
sk_poly_inputs = sk_poly.fit_transform(sk_inputs.reshape(-1, 1))
poly_reg_model = LinearRegression()
poly_reg_model.fit(sk_poly_inputs, sk_outputs)
a_sk = poly_reg_model.coef_
b_sk = poly_reg_model.intercept_
print(a_sk, b_sk)
```

```
[-5.31103644e+00  1.55941662e-01 -2.34706297e-03  1.92370550e-05
 -8.12548986e-08  1.38734052e-10] 73.25495081585393
```

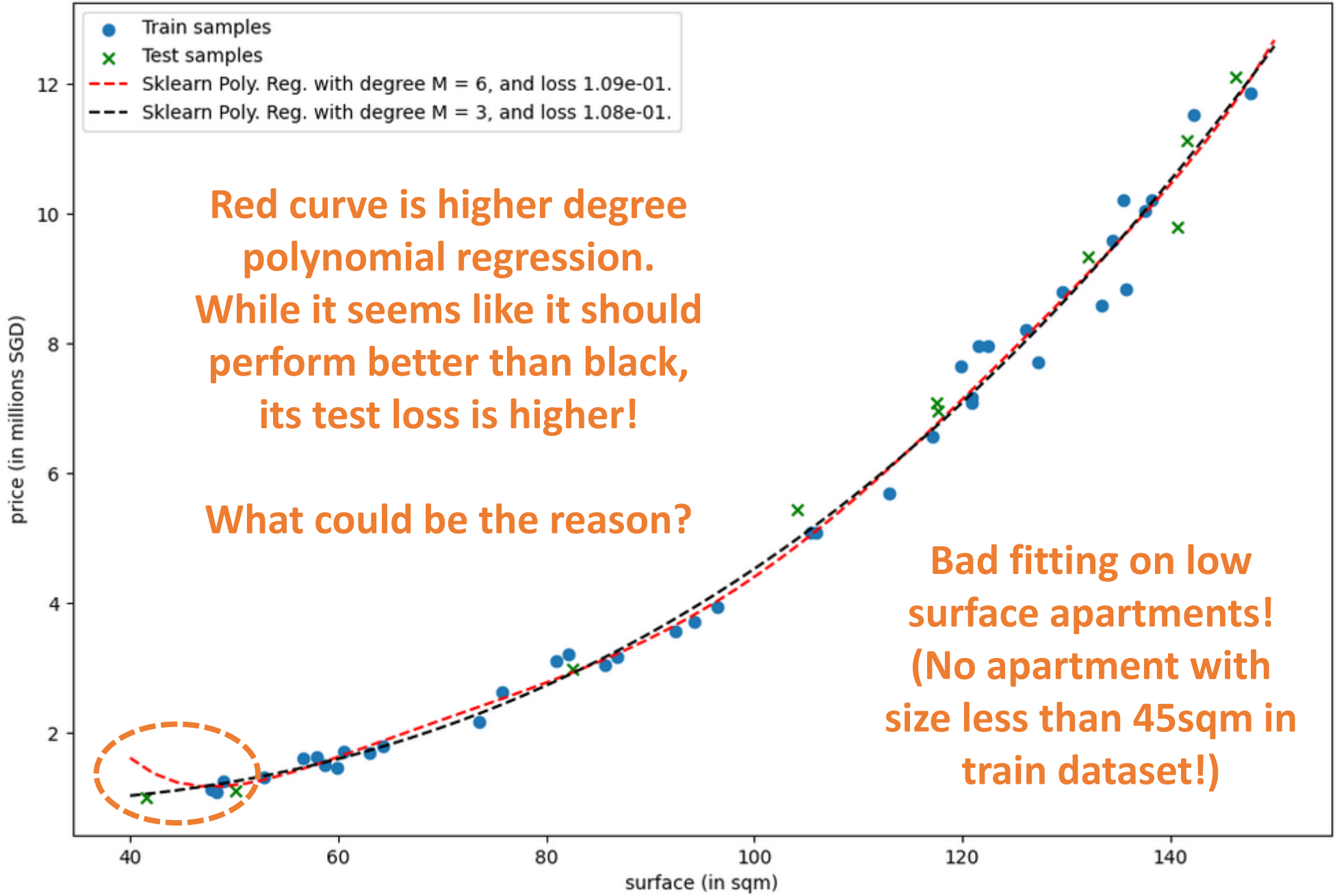
Restricted

Restricted



Restricted

Restricted



Restricted

Restricted

Spotting over/underfitting

A typical way to identify overfitting or underfitting is to look at two loss values: the loss calculated using samples from the training set and the one calculated using samples from the test set.

Observation: In general, the train loss has a smaller value to the test one, but they should remain on similar orders of magnitude.

If not, then there might be underfitting or overfitting.

```
1 loss_train = loss_mse_poly(poly_reg_model, n_degree, train_inputs, train_outputs)
2 print(loss_train)
3 loss_test = loss_mse_poly(poly_reg_model, n_degree, test_inputs, test_outputs)
4 print(loss_test)
```

```
5.46e-02
7.99e-01
```

Restricted

Generalization vs. Train/Test Distributions

Definition (The need for train and test distribution similarity):

In order to generalize well, it is important that the **train and test set are following similar distributions**. Mathematically speaking, we want:

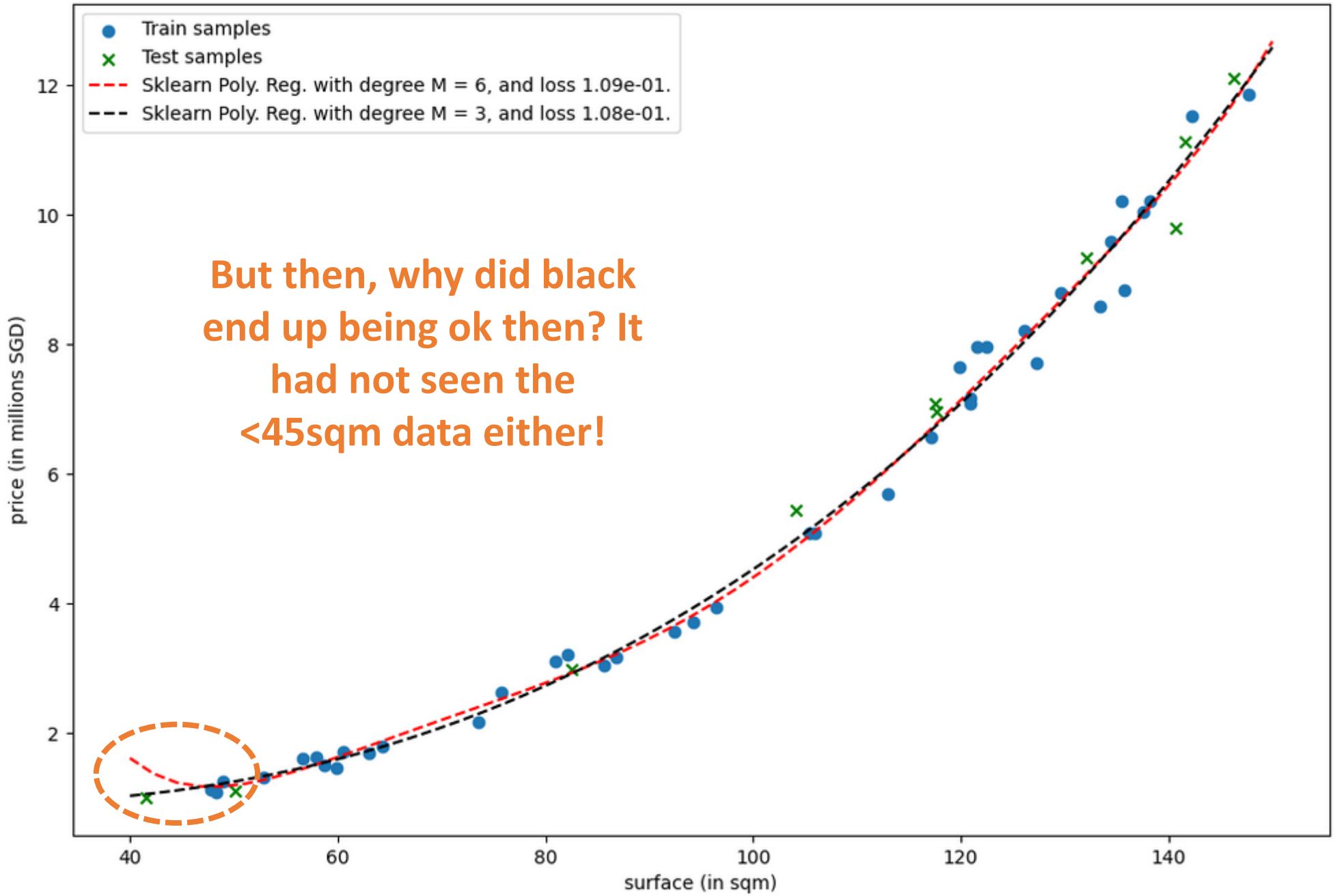
$$P_{train} \approx P_{test}$$

Here, we ended up having a problem, as our training dataset did not contain any apartments with size lower than 45sqm.

I mean, you would not train a model on Singaporean apartments and use it to predict the price of apartments in Kuala Lumpur, right?

Restricted

Restricted



Restricted

Restricted

Regularization

Definition (Regularization):

Regularization consists of techniques, whose purpose is to help the model to avoid overfitting.

It can typically be done by making sure the model is not too complex for the data, but many other approaches exist.

In general,

- **Scale model and dataset complexity:** the more complex the data is (in terms of features), the more complex the model can be.
- **Scale model and dataset size:** the more data you have (in terms of quantity of individual samples in dataset), the more complex the model can be.

Restricted

Restricted

Regularization

Definition (Regularization):

Regularization consists of techniques, whose purpose is to help the model to avoid overfitting.

It can typically be done by making sure the model is not too complex for the data, but many other approaches exist.

A common approach consists of **adding a penalty term to the loss function**.

The penalty term, also known as a **regularization term**, discourages the model from assigning too much weight to any one feature, i.e. making a certain parameter a_k too prominent (and that is often the reason leading to overfitting).

Restricted

Restricted

Regularization

Definition (Ridge Regression):

The **Ridge Regressor is a polynomial regressor,** like described earlier.

Its only difference is that **the loss function includes an additional regularization term $\lambda\, R(a, b)$.**

This regularization term consists of the **sum of the squared values for the trainable parameters**, i.e.

$$\lambda\, R(a,b) = \frac{\lambda}{2N}\left(\sum_{k}^{M} (a_k)^2 \; + \; (b)^2\right)$$

Why does it work?

- This **regularization term encourages the model to assign low values to the model parameters**.
- This will, in turn, lead to less overfitting.
- Indeed, overfitting often occurs when high values are assigned to coefficients for high degrees (i.e. coefficients a_k with high k values).

Restricted

Restricted

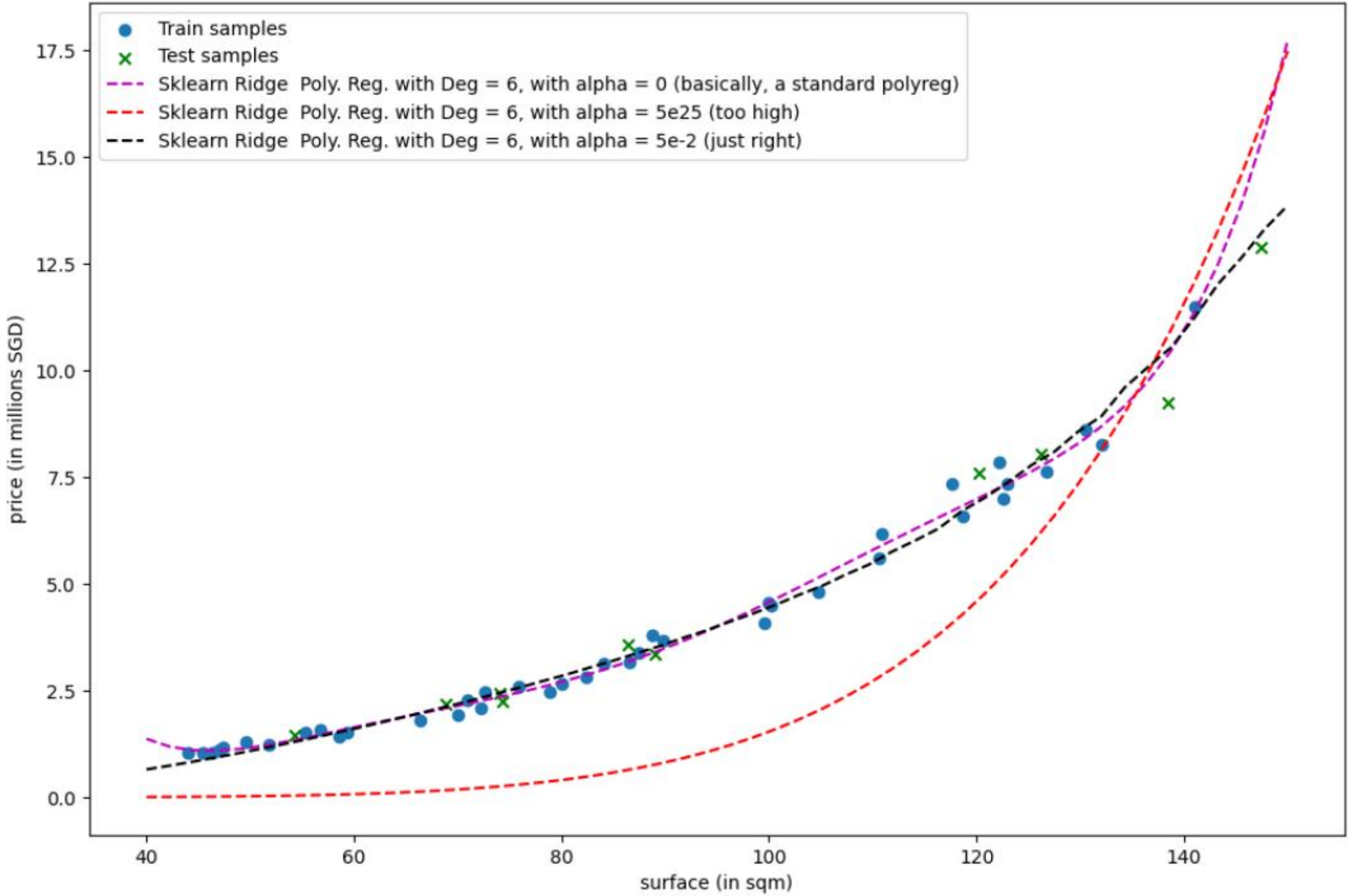
Regularization and hyperparameter λ

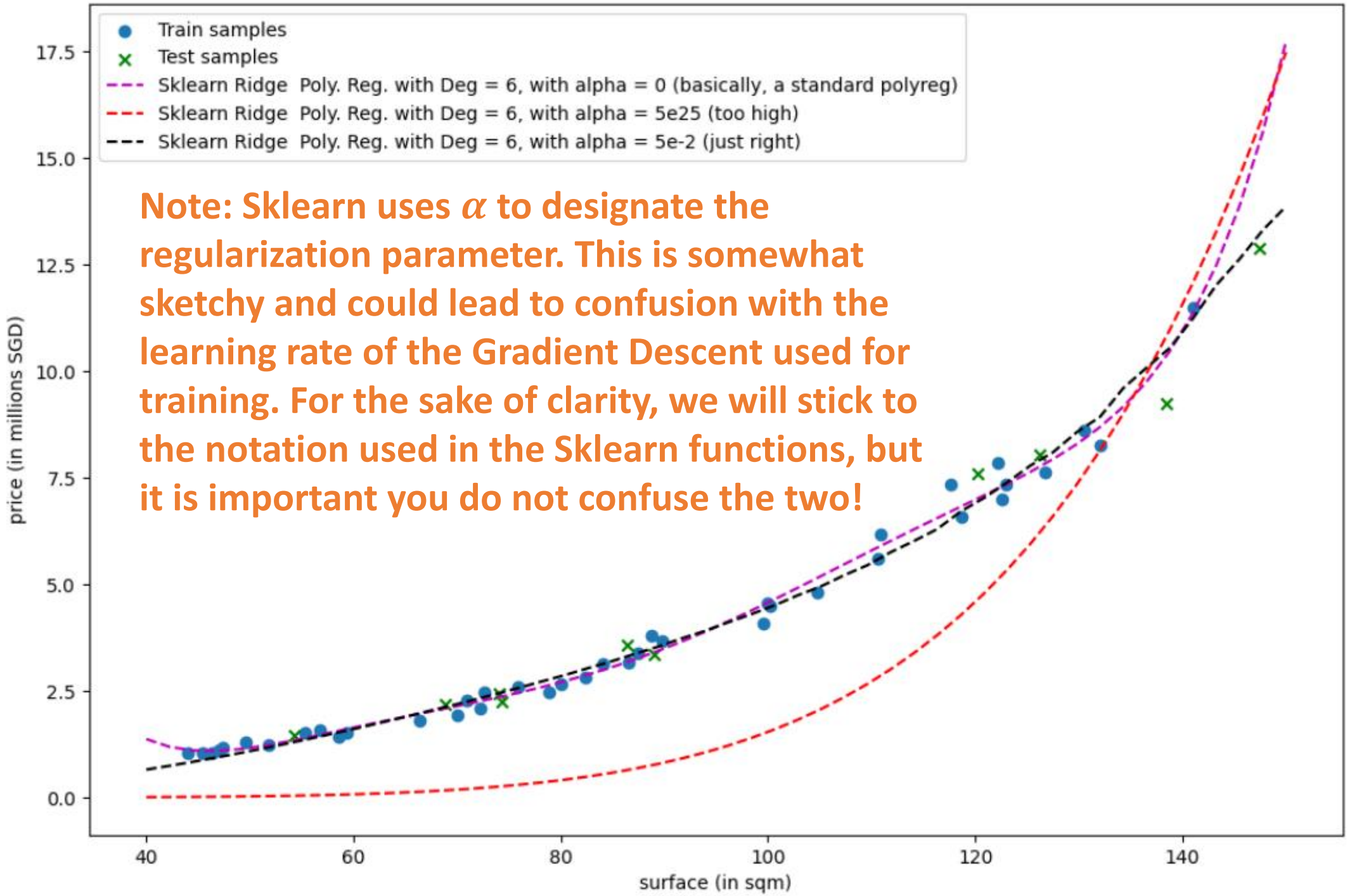
Just like the learning rate α in the GD algorithm, the parameter λ is **manually decided** and is used to weight the importance of the regularization term compared to the MSE.

- If the **value for λ is small**, the loss function will be **mostly MSE** and our model will therefore suffer from the same **overfitting** problems as earlier.
- If the **value of λ is too high** however, the model will **mostly ignore the MSE** part of the loss function, which will lead to a model not fitting (or **underfitting**) the data.

We say that λ is a **hyperparameter**, and choosing the right value to use is commonly referred to as **hyperparameter tuning**.

Restricted


Train samples
Test samples
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 0 (basically, a standard polyreg)
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 5e25 (too high)
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 5e-2 (just right)
price (in millions SGD)
surface (in sqm)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
40
60
80
100
120
140


Train samples
Test samples
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 0 (basically, a standard polyreg)
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 5e25 (too high)
Sklearn Ridge Poly. Reg. with Deg = 6, with alpha = 5e-2 (just right)
Note: Sklearn uses α to designate the regularization parameter. This is somewhat sketchy and could lead to confusion with the learning rate of the Gradient Descent used for training. For the sake of clarity, we will stick to the notation used in the Sklearn functions, but it is important you do not confuse the two!
price (in millions SGD)
surface (in sqm)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
40
60
80
100
120
140

Restricted

Regularization with other things than L2?

Possible to use the sum of **absolute** values (or what we call a L1 regularization), making a **Lasso Regressor** instead of a **Ridge Regressor**.

Why use L1 regularization over L2 regularization?

- L1 regularization results in some weights being exactly equal to zero, which can be used for feature selection.
- L2 regularization will not produce sparse models, and it will only shrink the weights.

Regularization Term	Method
L2 norm: $\|\beta\|_2$	ridge regression
L1 norm: $\|\beta\|_1$	LASSO

Restricted

Restricted

Regularization with other things than L2?

Also possible to use the sum of **absolute** values **AND squared** values as regularization, i.e. combining the **L1** and **L2** regularizations together.

Essentially getting the best of both worlds, and making an **Elastic Regressor** instead! (MSE + L1 Reg + L2 Reg)

$$ElasticNet = \sum_{i=1}^{n} (y_i - y(x_i))^2 + \alpha \sum_{j=1}^{p} |w_j| + \alpha \sum_{j=1}^{p} (w_i)^2$$

MSE L1 L2

Restricted