Optimizing the training process:
Underfitting, overfitting, testing, and regularization

- Let's say that we have to study for a test.
- Several things could go wrong during our study process.
- Maybe we didn't study enough. There's no way to fix that, and we'll likely perform poorly in our test. ---------- Underfitting
- What if we studied a lot but in the wrong way. For example, instead of focusing on learning, we decided to memorize the entire textbook word for word. Will we do well in our test? It's likely that we won't, because we simply memorized everything without learning. ----------Overfitting
- The best option, of course, would be to study for the exam properly and in a way that enables us to answer new questions that we haven't seen before on the topic. ----------Generalization

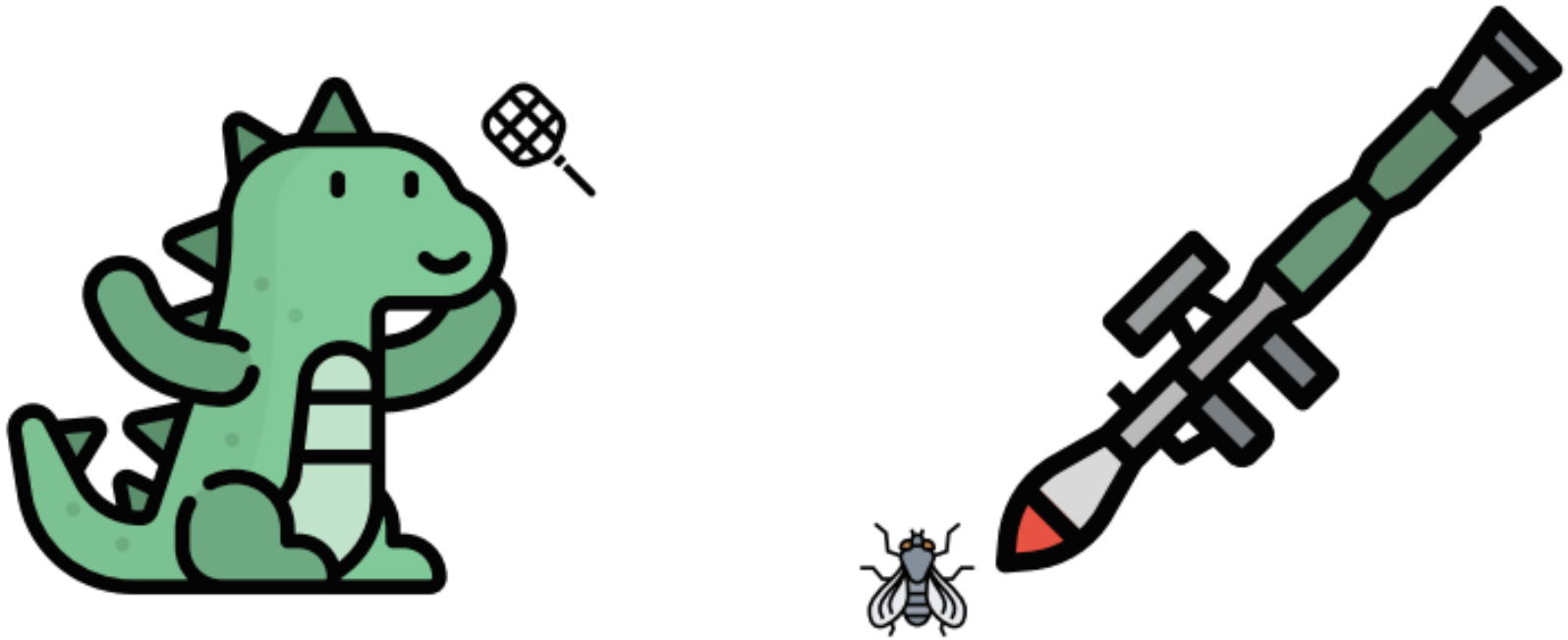

Figure 4.1 Underfitting and overfitting are two problems that can occur when training our machine learning model. Left: Underfitting happens when we oversimplify the problem at hand, and we try to solve it using a simple solution, such as trying to kill Godzilla using a fly swatter. Right: Overfitting happens when we overcomplicate the solution to a problem and try to solve it with an exceedingly complicated solution, such as trying to kill a fly using a bazooka.

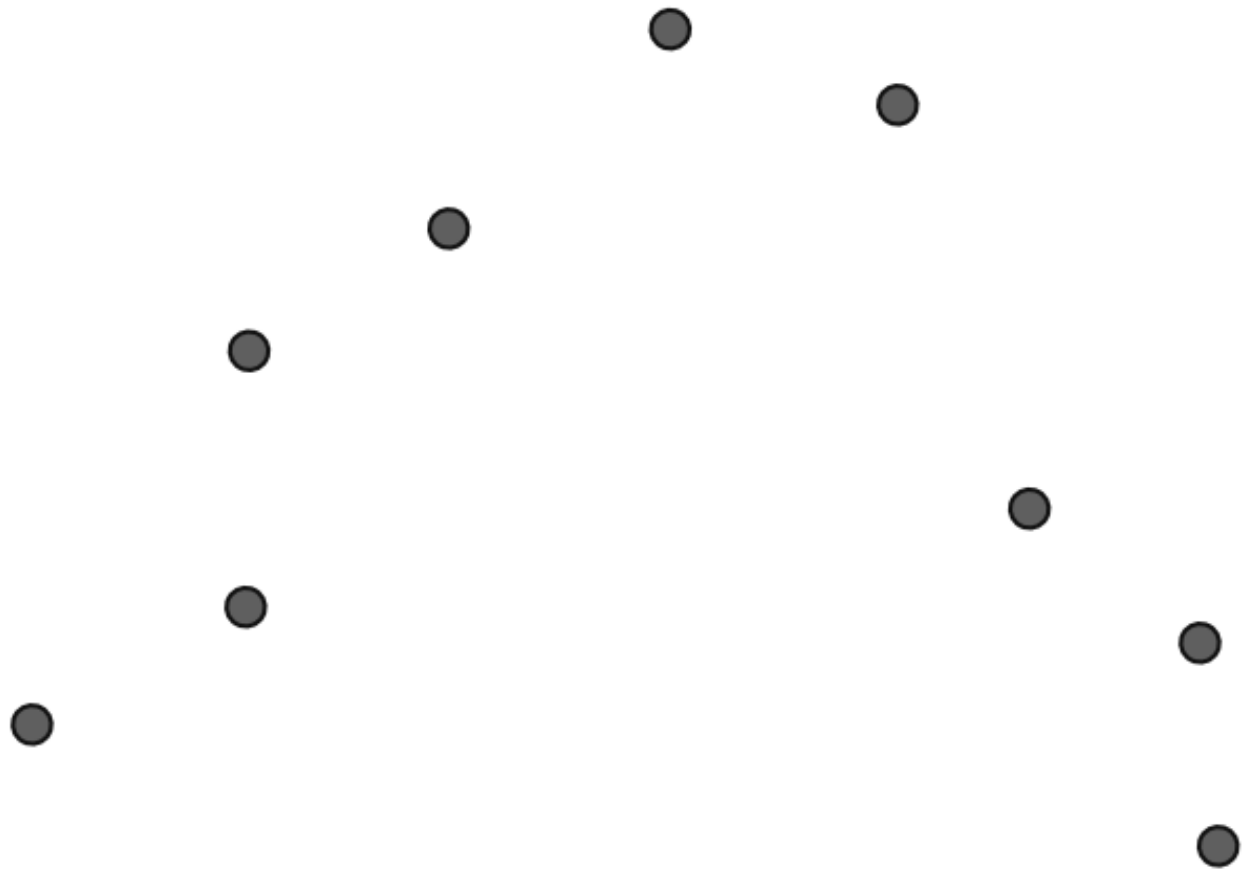

Figure 4.2 In this dataset, we train some models and exhibit training problems such as underfitting and overfitting. If you were to fit a polynomial regression model to this dataset, what type of polynomial would you use: a line, a parabola, or something else?

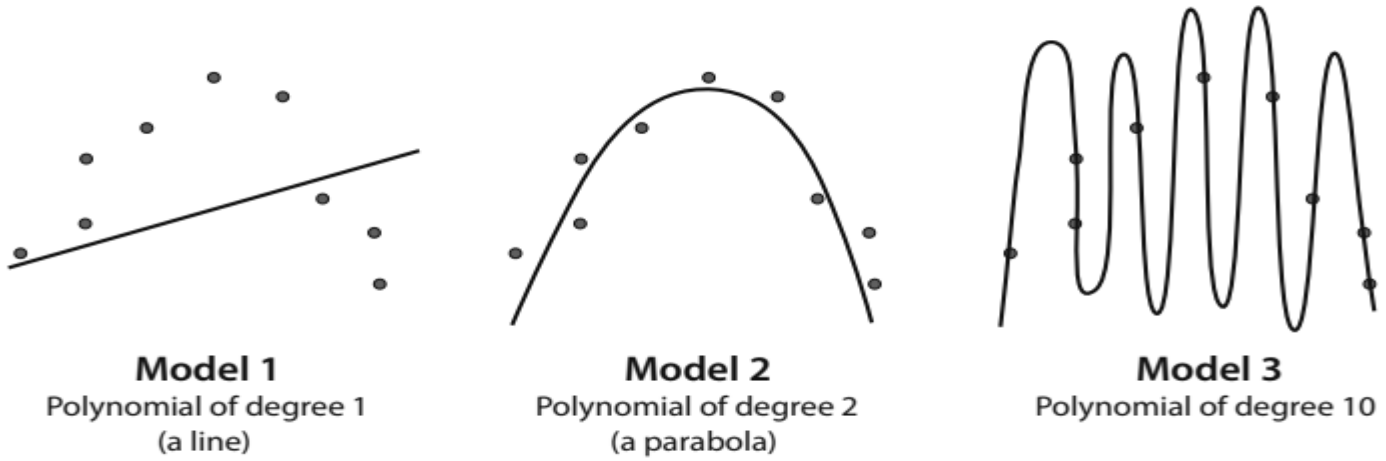

Figure 4.3 Fitting three models to the same dataset. Model 1 is a polynomial of degree 1, which is a line. Model 2 is a polynomial of degree 2, or a quadratic. Model 3 is a polynomial of degree 10. Which one looks like the best fit?

- Notice that model 1 is too simple, because it is a line trying to fit a quadratic dataset. There is no way we'll find a good line to fit this dataset, because the dataset simply does not look like a line. Therefore, model 1 is a clear example of underfitting.
- Model 2, in contrast, fits the data pretty well. This model neither overfits nor underfits.
- Model 3 fits the data extremely well, but it completely misses the point. The data is meant to look like a parabola with a bit of noise, and the model draws a very complicated polynomial of degree 10 that manages to go through each one of the points but doesn't capture the essence of the data.

  Model 3 is a clear example of overfitting.

How do we get the computer to pick the right model?

By testing

- Testing a model consists of picking a small set of the points in the dataset and choosing to use them not for training the model but for testing the model's performance. This set of points is called the *testing set*.
- The remaining set of points (the majority), which we use for training the model, is called the *training set*.
- Once we've trained the model on the training set, we use the testing set to evaluate the model.
- In this way, we make sure that the model is good at generalizing to unseen data, as opposed to memorizing the training set.

- Going back to the exam analogy, let's imagine training and testing this way.
- Let's say that the book we are studying for in the exam has 100 questions at the end.
- We pick 80 of them to train, which means we study them carefully, look up the answers, and learn them.
- Then we use the remaining 20 questions to test ourselves—we try to answer them without looking at the book, as in an exam setting.

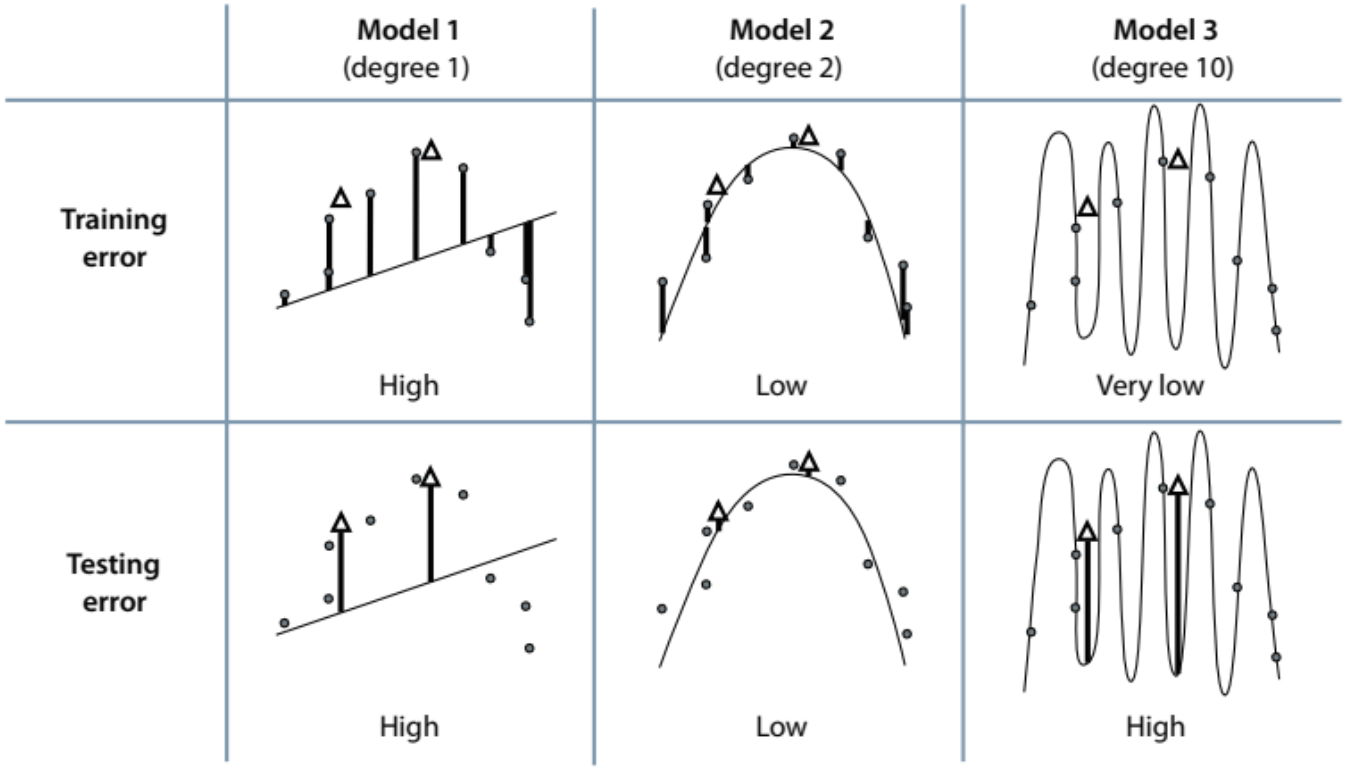

Figure 4.4 We can use this table to decide how complex we want our model. The columns represent the three models of degree 1, 2, and 10. The columns represent the training and the testing error. The solid circles are the training set, and the white triangles are the testing set. The errors at each point can be seen as the vertical lines from the point to the curve. The error of each model is the mean absolute error given by the average of these vertical lengths. Notice that the training error goes down as the complexity of the model increases. However, the testing error goes down and then back up as the complexity increases. From this table, we conclude that out of these three models, the best one is model 2, because it gives us a low testing error.

- Looking at the top row we can see that model 1 has a large training error, model 2 has a small training error, and model 3 has a tiny training error (zero, in fact). Thus, model 3 does the best job on the training set.

- Model 1 still has a large testing error,meaning that this is simply a bad model, underperforming with the training and the testing set: it underfits.

- Model 2 has a small testing error, which means it is a good model, because it fits both the training and the testing set well.

- Model 3, however, produces a large testing error. Because it did such a terrible job fitting the testing set, yet such a good job fitting the training set, we conclude that model 3 overfits.

How do we pick the testing set, and how big should it be?

- A portion of the dataset is picked randomly (or based on some features in case of temporal data) as the test set.
- In practice, 10-20% of the data is kept as the test set.

Can we use our testing data for training the model? No.

golden rule Thou shalt never use your testing data for training.

- We broke the golden rule in the previous example.
- Recall that we had three polynomial regression models: one of degree 1, one of degree 2, and one of degree 10, and we didn't know which one to pick.
- We used our training data to train the three models, and then we used the testing data to decide which model to pick.
- We are not supposed to use the testing data to train our model or to make any decisions on the model or its hyperparameters.

Solution: Validation Set

We break our dataset into the following three sets:

- **Training set:** for training all our models
- **Validation set:** for making decisions on which model to use
- **Testing set:** for checking how well our model did

it is common to use a 60-20-20 split or an 80-10-10 split—in other words,

60% training, 20% validation, 20% testing, or

80%training, 10% validation, 10% testing.

A numerical way to decide how complex our model should be:

The model complexity graph

- Imagine that we have a different and much more complex dataset, and we are trying to build a polynomial regression model to fit it. We want to decide the degree of our model among the numbers between 0 and 10 (inclusive).
- the way to decide which model to use is to pick the one that has the smallest validation error.

- However, plotting the training and validation errors can give us some valuable information and help us examine trends.

The model complexity graph

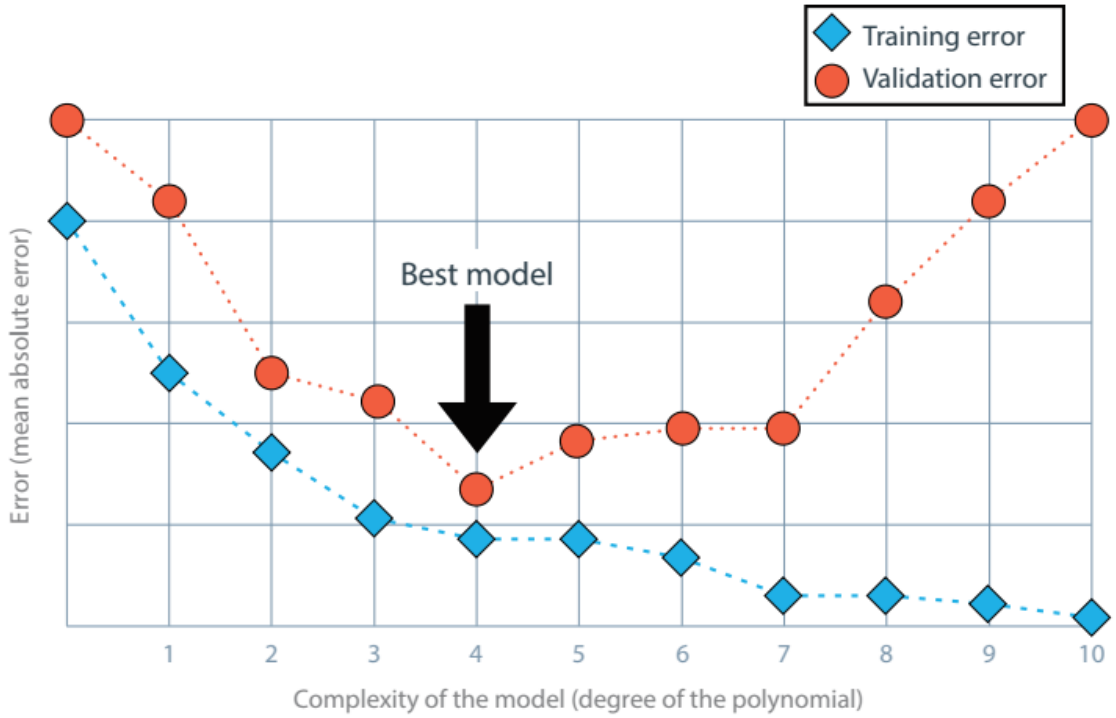

Figure 4.5 The model complexity graph is an effective tool to help us determine the ideal complexity of a model to avoid underfitting and overfitting. In this model complexity graph, the horizontal axis represents the degree of several polynomial regression models, from 0 to 10 (i.e., the complexity of the model). The vertical axis represents the error, which in this case is given by the mean absolute error. Notice that the training error starts large and decreases as we move to the right. This is because the more complex our model is, the better it can fit the training data. The validation error, however, starts large, then decreases, and then increases again—very simple models can't fit our data well (they underfit), whereas very complex models fit our training data but not our validation data because they overfit. A happy point in the middle is where our model neither underfits or overfits, and we can find it using the model complexity graph.

Another alternative to avoiding overfitting: Regularization

- simple models tend to underfit and complex models tend to overfit
- in the previous methods, we tested several models and selected the one that best balanced performance and complexity.
- In contrast, when we use regularization, we don't need to train several models. We simply train the model once, but during the training, we try to not only improve the model's performance but also reduce its complexity.

Another alternative to avoiding overfitting: Regularization

- Performance (in mL of water leaked)
  Roofer 1: 1000 mL water
  Roofer 2: 1 mL water
  Roofer 3: 0 mL water
- Complexity (in price)
  Roofer 1: $1
  Roofer 2: $100
  Roofer 3: $100,000
- Performance + complexity
  Roofer 1: 1001
  Roofer 2: 101
  Roofer 3: 100,000

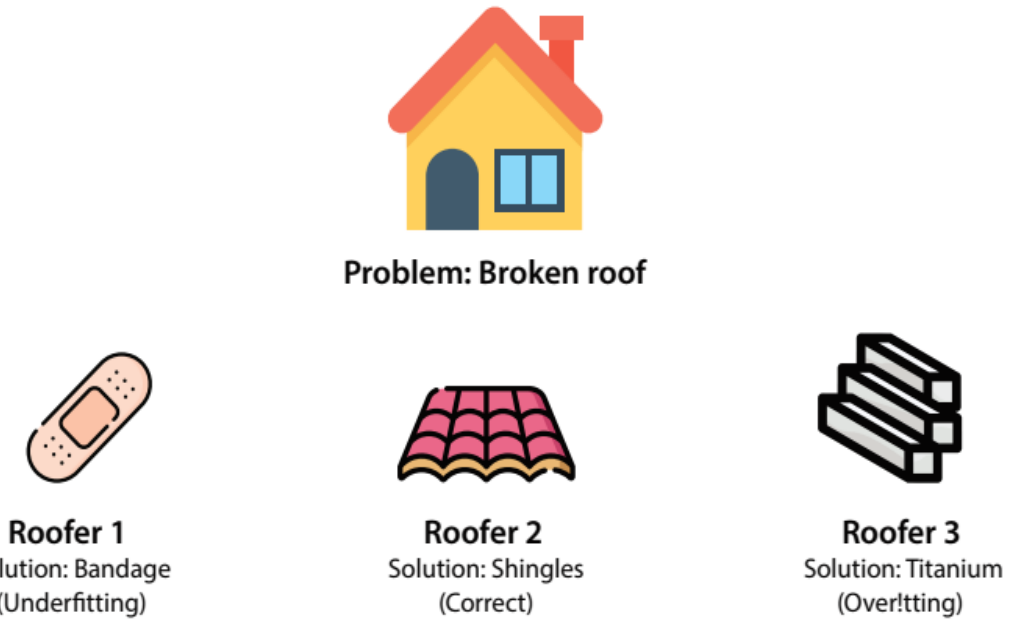

Figure 4.6 An analogy for underfitting and overfitting. Our problem consists of a broken roof. We have three roofers who can fix it. Roofer 1 comes with a bandage, roofer 2 comes with roofing shingles, and roofer 3 comes with a block of titanium. Roofer 1 oversimplified the problem, so they represent underfitting. Roofer 2 used a good solution. Roofer 3 overcomplicated the solution, so they represent overfitting.

Now it is clear that roofer 2 is the best one, which means that optimizing performance and complexity at the same time yields good results that are also as simple as possible. This is what regularization is about: measuring performance and complexity with two different error functions, and adding them to get a more robust error function.

Regularization- Measuring how complex a model is: L1 and L2 norm

- Notice that a model with more coefficients, or coefficients of higher value, tends to be more complex. Therefore, any formula that matches this will work, such as the following:
  • The sum of the absolute values of the coefficients
  • The sum of the squares of the coefficients
  The first one is called the *L1 norm*, and the second one is called the *L2 norm*.

- **Model 1:** $\hat{y} = 2x + 3$
- **Model 2:** $\hat{y} = -x^2 + 6x - 2$
- **Model 3:** $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10$

The L1 and L2 norms are calculated as follows:

L1 norm:

- **Model 1:** $|2| = 2$
- **Model 2:** $|-1| + |6| = 7$
- **Model 3:** $|1| + |4| + |-9| + |3| + |-14| + |-2| + |-9| + |1| + |6| = 49$

L2 norm:

- **Model 1:** $2^2 = 2$
- **Model 2:** $(-1)^2 + 6^2 = 37$
- **Model 3:** $1^2 + 4^2 + (-9)^2 + 3^2 + (-14)^2 + (-2)^2 + (-9)^2 + 1^2 + 6^2 = 425$

Regularization- Modifying the error function

- in the roofer analogy, our goal was to find a roofer that provided both good quality and low complexity. We did this by minimizing the sum of two numbers: the measure of quality and the measure of complexity. Regularization consists of applying the same principle to our machine learning model.
- **regression error** A measure of the quality of the model. In this case, it can be the absolute or square errors
- **regularization term** A measure of the complexity of the model. It can be the L1 or the L2 norm of the model.
- Error = Regression error + λ Regularization term
- λ is the regularization hyperparameter.
- Lasso regression error = Regression error + λ L1 norm
- Ridge regression error = Regression error + λ L2 norm

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 + \lambda ||\mathbf{w}||_2^2$$

Regularization- Effects of L1 and L2 regularization

- A quick rule of thumb to use when deciding if we want to use L1 or L2 regularization follows:
  - if we have too many features and we'd like to get rid of most of them, L1 regularization is perfect for that.
  - If we have only few features and believe they are all relevant, then L2 regularization is what we need, because it won't get rid of our useful features.