# Introduction to Regression

- Sandeep Chaurasia

## Machine Learning

- [Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed. —Arthur Samuel, 1959

- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. —Tom Mitchell, 1997

Your spam filter is a machine learning program that, given examples of spam emails (flagged by users) and examples of regular emails (nonspam, also called “ham”), can learn to flag spam.

The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample).

The part of a machine learning system that learns and makes predictions is called a model.

Neural networks and random forests are examples of models.

In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined;

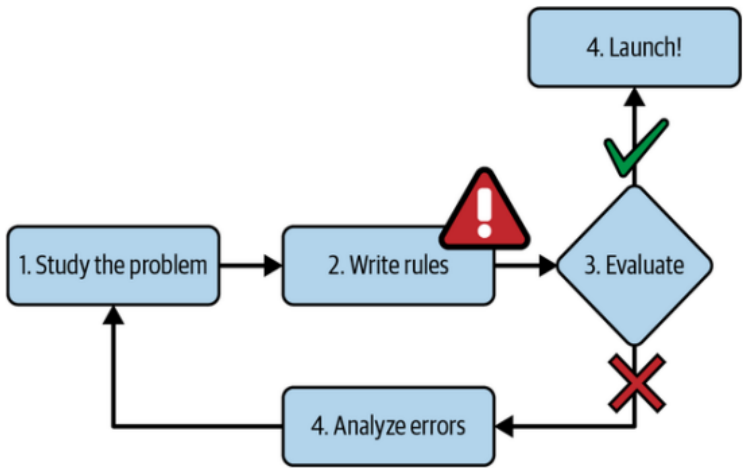


Figure 1-1. The traditional approach

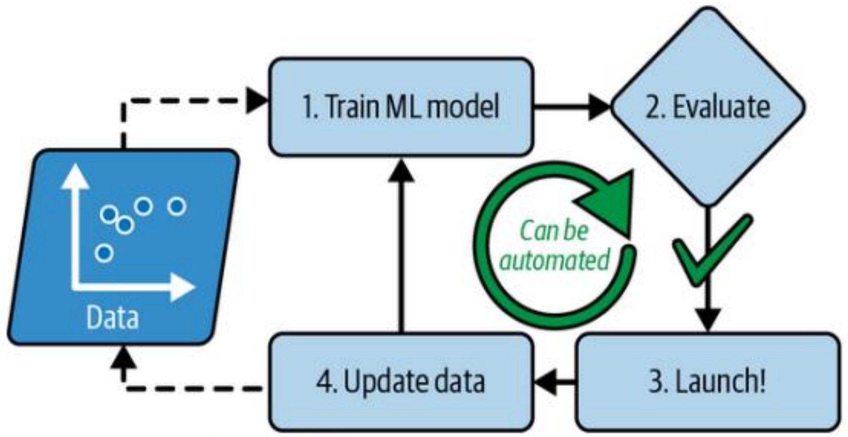


Figure 1-3. Automatically adapting to change

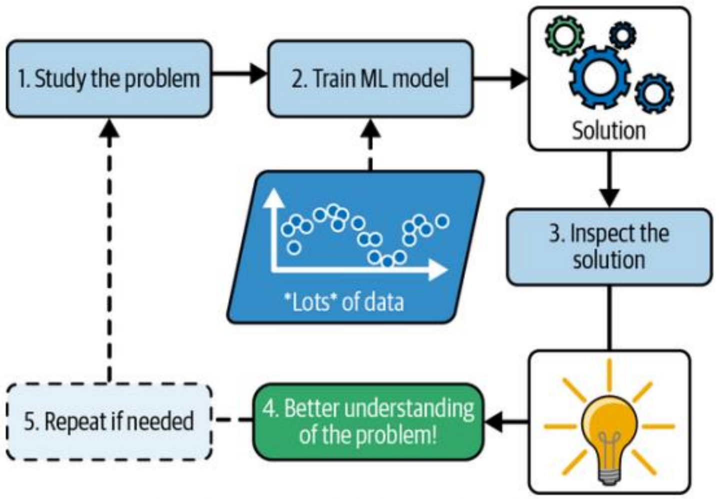


Figure 1-4. Machine learning can help humans learn

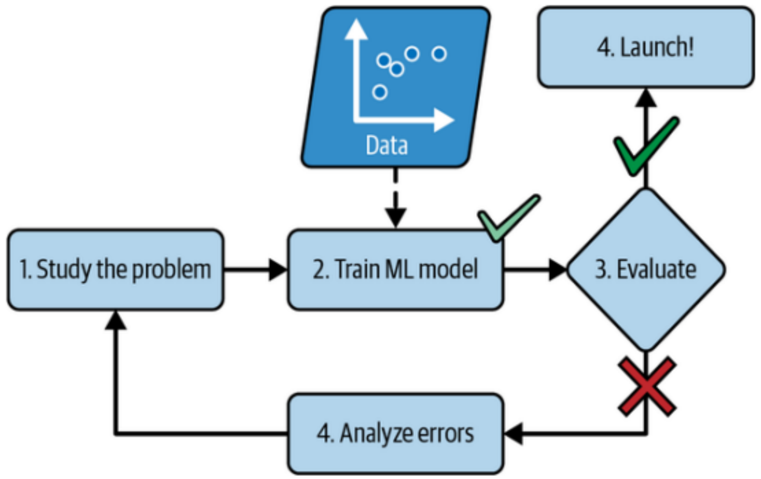


Figure 1-2. The machine learning approach

## Examples of Applications:

- Analyzing images of products on a production line to automatically classify them
- Detecting tumors in brain scans
- Automatically classifying news articles
- Automatically flagging offensive comments on discussion forums
- Summarizing long documents automatically
- Creating a chatbot or a personal assistant.
- Forecasting your company’s revenue next year, based on many performance metrics.
- Making your app react to voice commands
- Detecting credit card fraud
- Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment
- Representing a complex, high-dimensional dataset in a clear and insightful diagram
- Recommending a product that a client may be interested in, based on past purchases
- Building an intelligent bot for a game.

## Types of Machine Learning Systems

- How they are supervised during training (supervised, unsupervised, semi-supervised, self-supervised, and others)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)
- These criteria are not exclusive; you can combine them in any way you like

# Supervised learning

- In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels

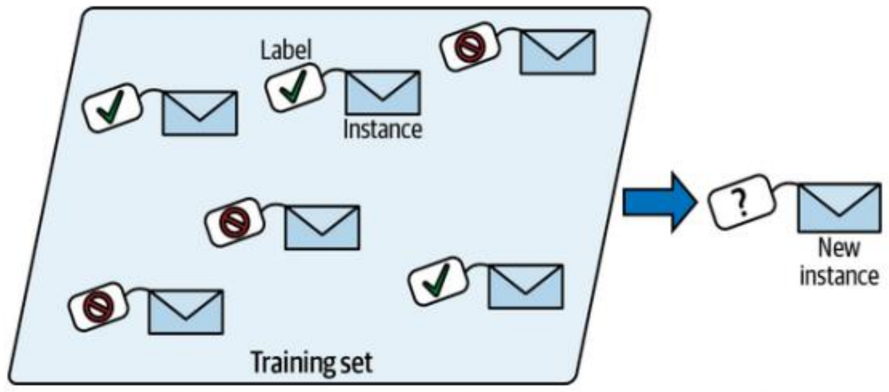


*Figure 1-5. A labeled training set for spam classification (an example of supervised learning)*

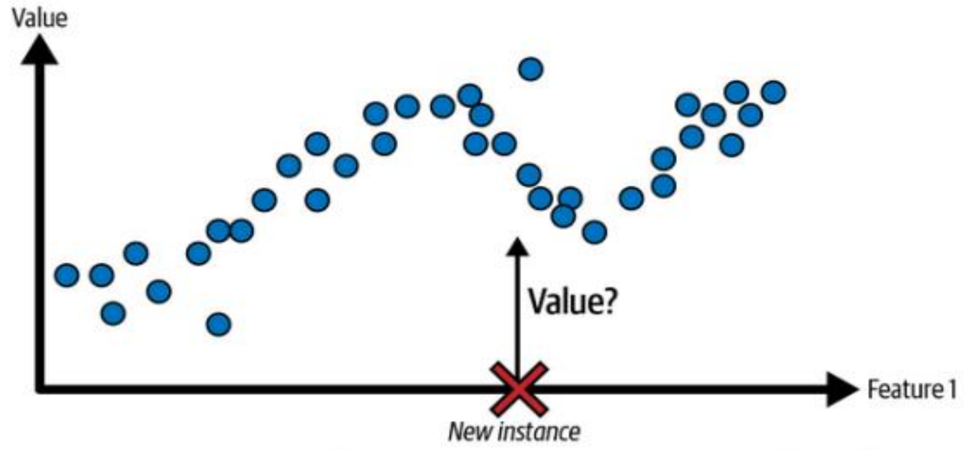


*Figure 1-6. A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)*

A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.

Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.). This sort of task is called regression.

# Unsupervised learning

- In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.
- For example, say you have a lot of data about your blog's visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors.

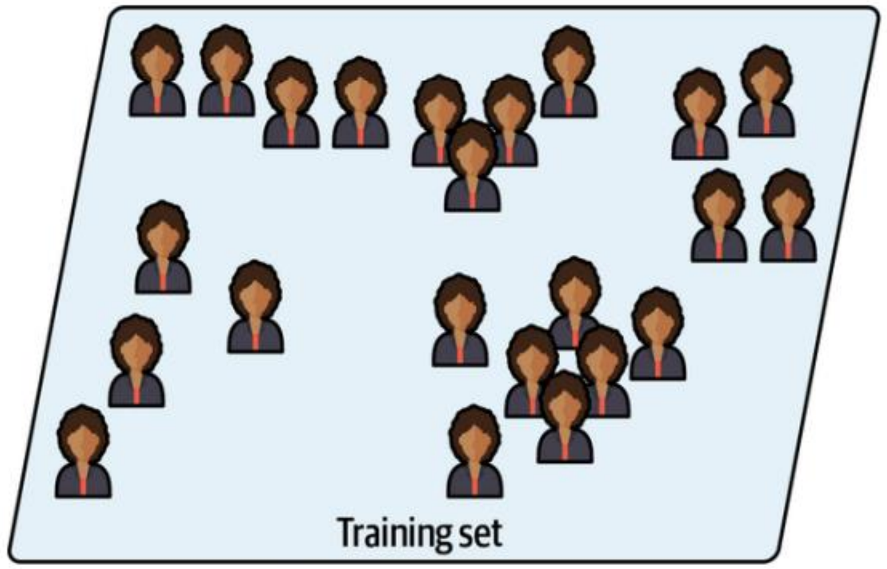


*Figure 1-7. An unlabeled training set for unsupervised learning*

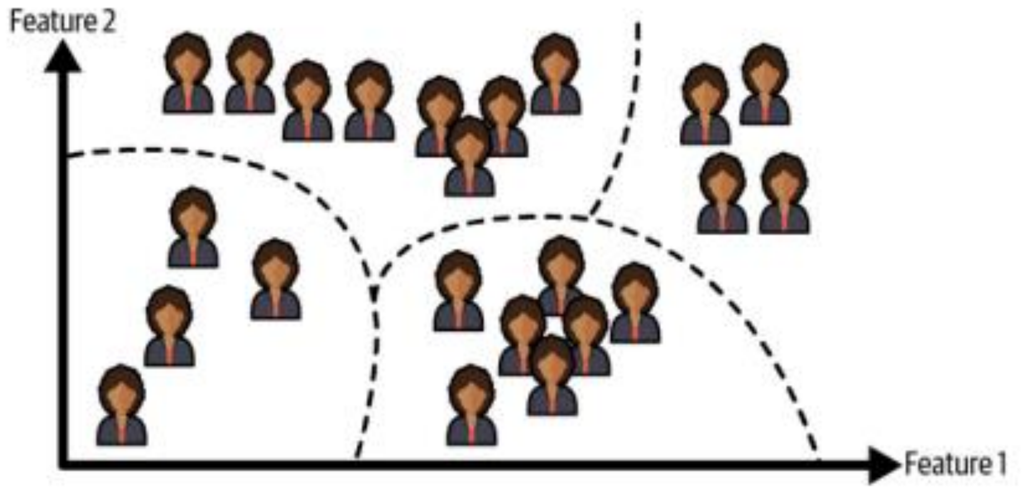


*Figure 1-8. Clustering*

- Another important unsupervised task is anomaly detection—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is shown mostly normal instances during training, so it learns to recognize them; then, when it sees a new instance, it can tell whether it looks like a normal one or whether it is likely an anomaly

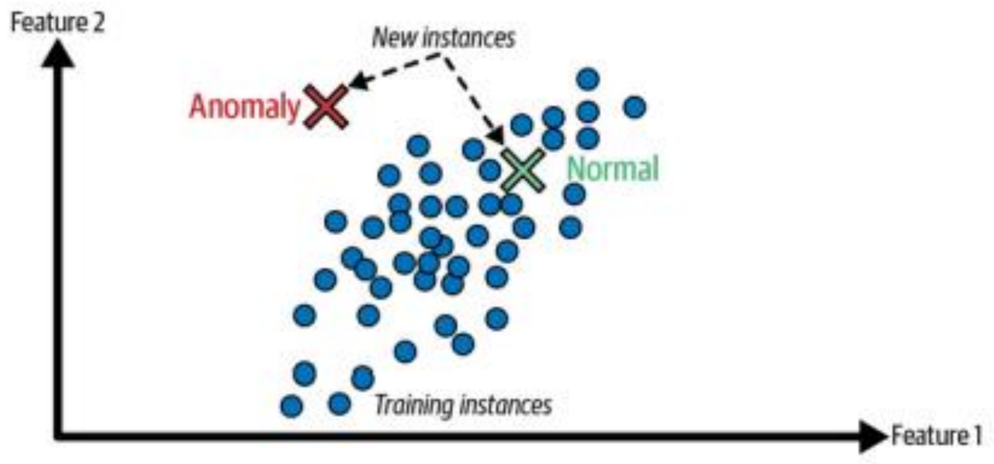


*Figure 1-10. Anomaly detection*

## Semi-supervised learning

- Since labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled. This is called semi supervised learning

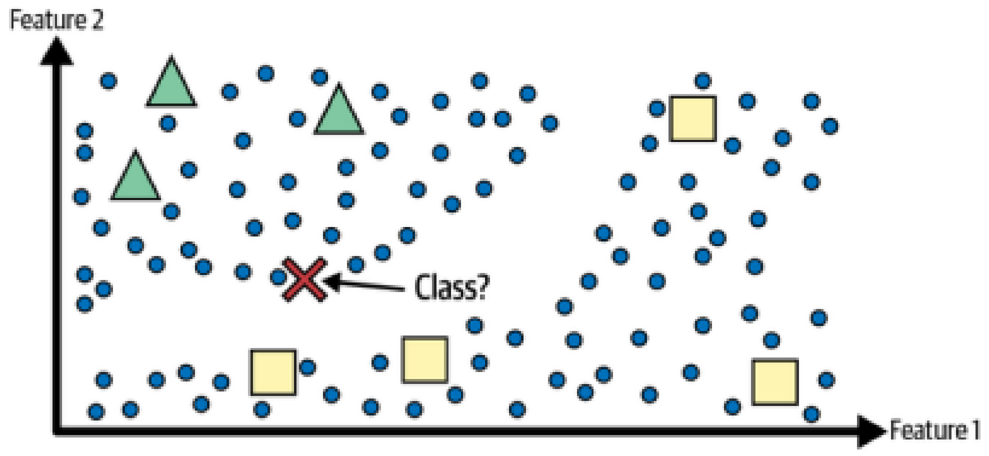


*Figure 1-11. Semi-supervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares*

Some photo-hosting services, such as Google Photos, are good examples of this

# Self-supervised learning

- Another approach to machine learning involves actually generating a fully labeled dataset from a fully unlabeled one. Again, once the whole dataset is labeled, any supervised learning algorithm can be used. This approach is called self-supervised learning.

For example, if you have a large dataset of unlabeled images, you can randomly mask a small part of each image and then train a model to recover the original image. During training, the masked images are used as the inputs to the model, and the original images are used as the labels

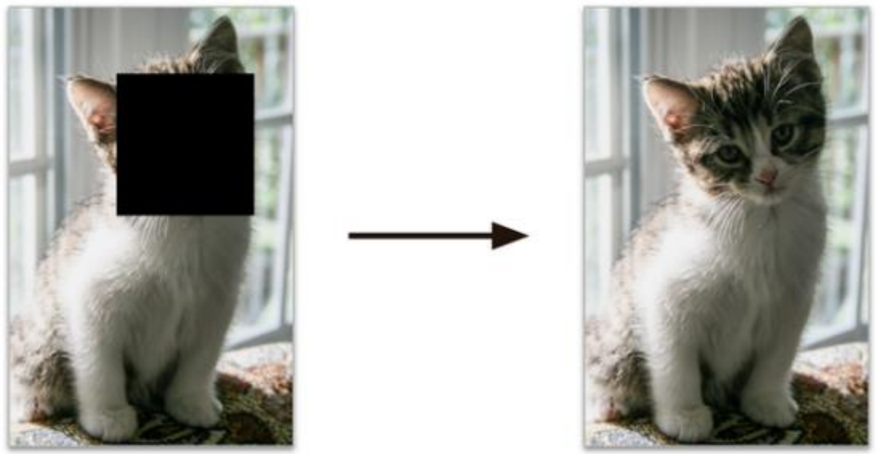


*Figure 1-12. Self-supervised learning example: input (left) and target (right)*

# Reinforcement learning

- The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards. It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

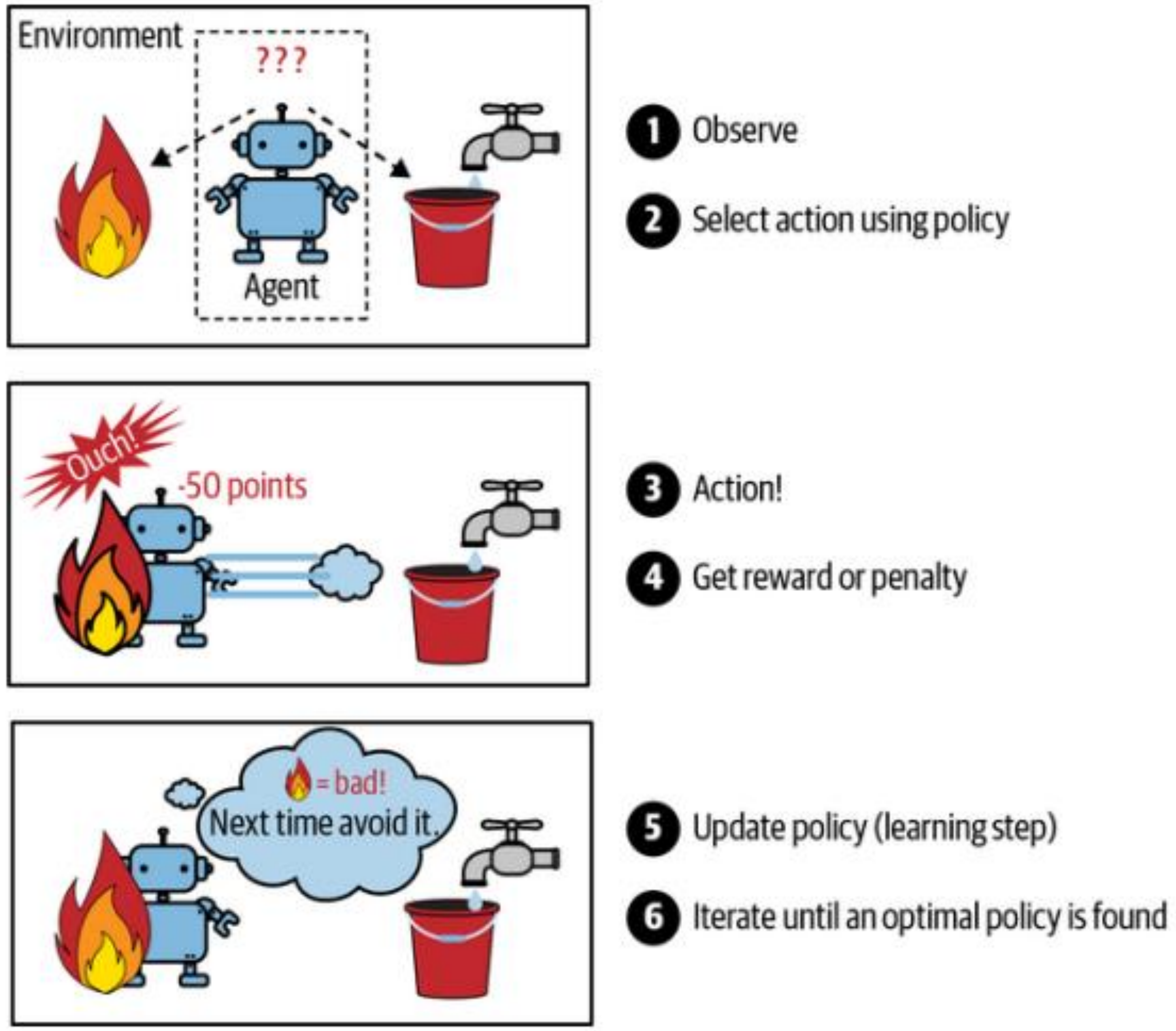

Environment
???
Agent
Ouch!
-50 points
= bad!
Next time avoid it
1 Observe
2 Select action using policy
3 Action!
4 Get reward or penalty
5 Update policy (learning step)
6 Iterate until an optimal policy is found

*Figure 1-13. Reinforcement learning*

## Batch Versus Online Learning

- Batch learning - In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.
- Even a model trained to classify pictures of cats and dogs may need to be retrained regularly, not because cats and dogs will mutate overnight, but because cameras keep changing, along with image formats, sharpness, brightness, and size ratios. Moreover, people may love different breeds next year, or they may decide to dress their pets with tiny hats—who knows?

# Online learning

- In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or in small groups called minibatches. Each learning step is fast and cheap, so the system can learn about new data on the fly

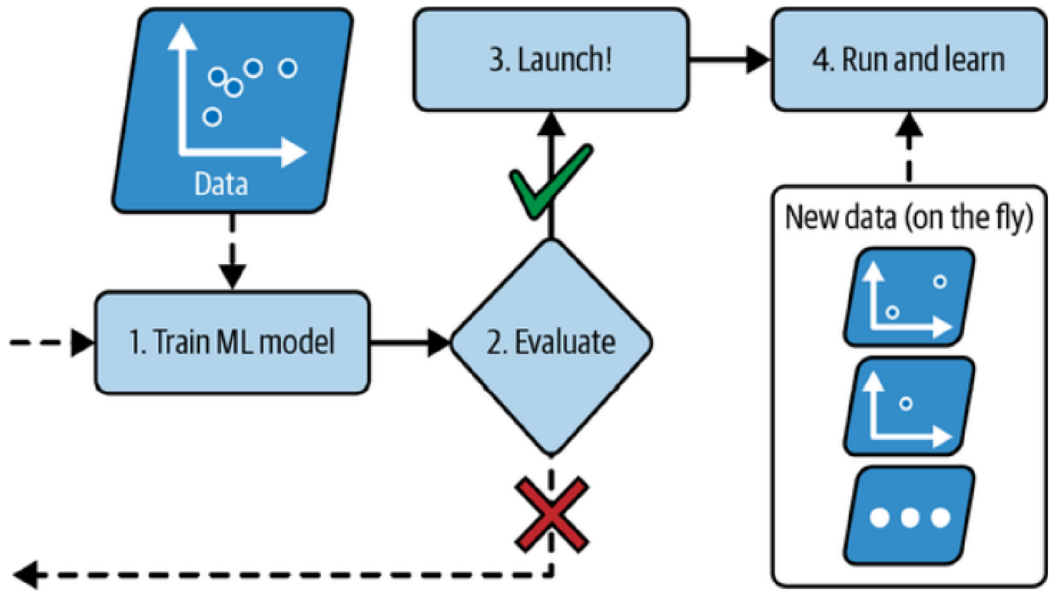


*Figure 1-14. In online learning, a model is trained and launched into production, and then it keeps learning as new data comes in*

## Instance-Based Versus Model-Based Learning

- Instance-based learning: the system learns the examples by heart, then generalizes to new cases by using a similarity measure to compare them to the learned examples (or a subset of them)

A (very basic) similarity measure between two emails could be to count the number of words they have in common. The system would flag an email as spam if it has many words in common with a known spam email

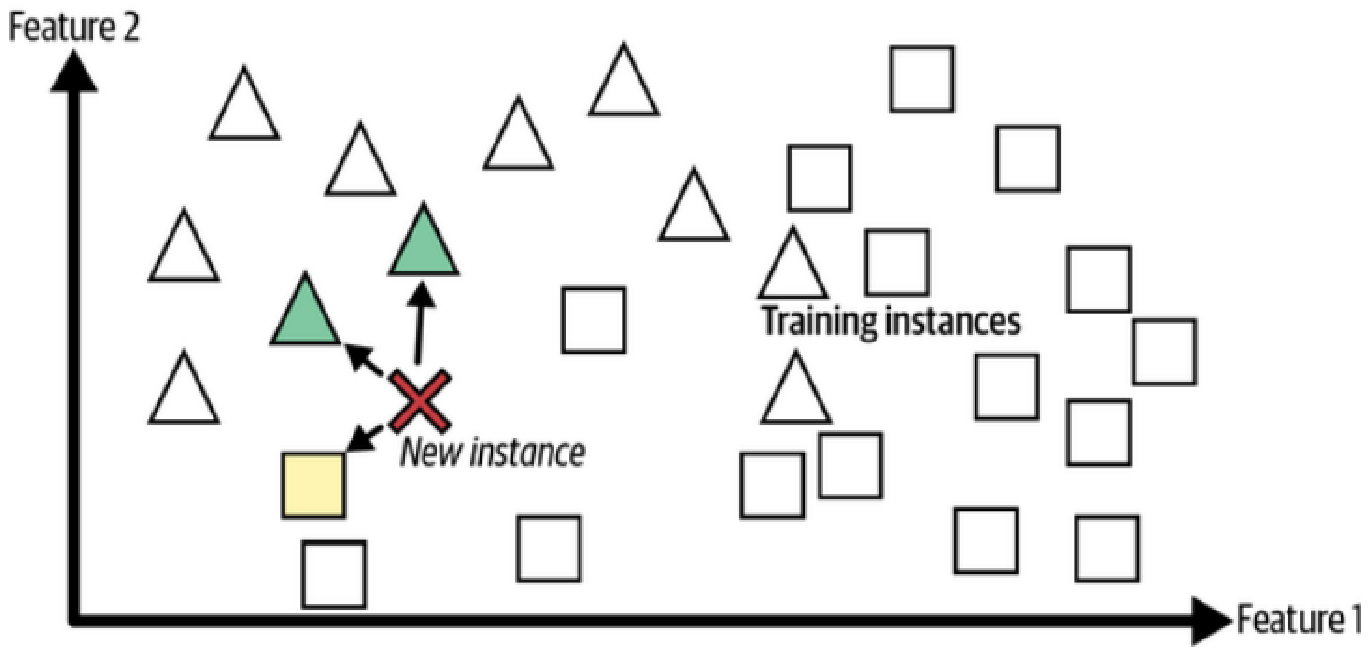


*Figure 1-16. Instance-based learning*

- Another way to generalize from a set of examples is to build a model of these examples and then use that model to make predictions. This is called model-based learning.

# Suppose you want to know if money makes people happy, so you download the Better Life Index data from the OECD's website and World Bank stats about gross domestic product (GDP) per capita. Then you join the tables and sort by GDP per capita

| Country | GDP per capita (USD) | Life satisfaction |
|---|---|---|
| Turkey | 28,384 | 5.5 |
| Hungary | 31,008 | 5.6 |
| France | 42,026 | 6.5 |
| United States | 60,236 | 6.9 |
| New Zealand | 42,404 | 7.3 |
| Australia | 48,698 | 7.3 |
| Denmark | 55,938 | 7.6 |

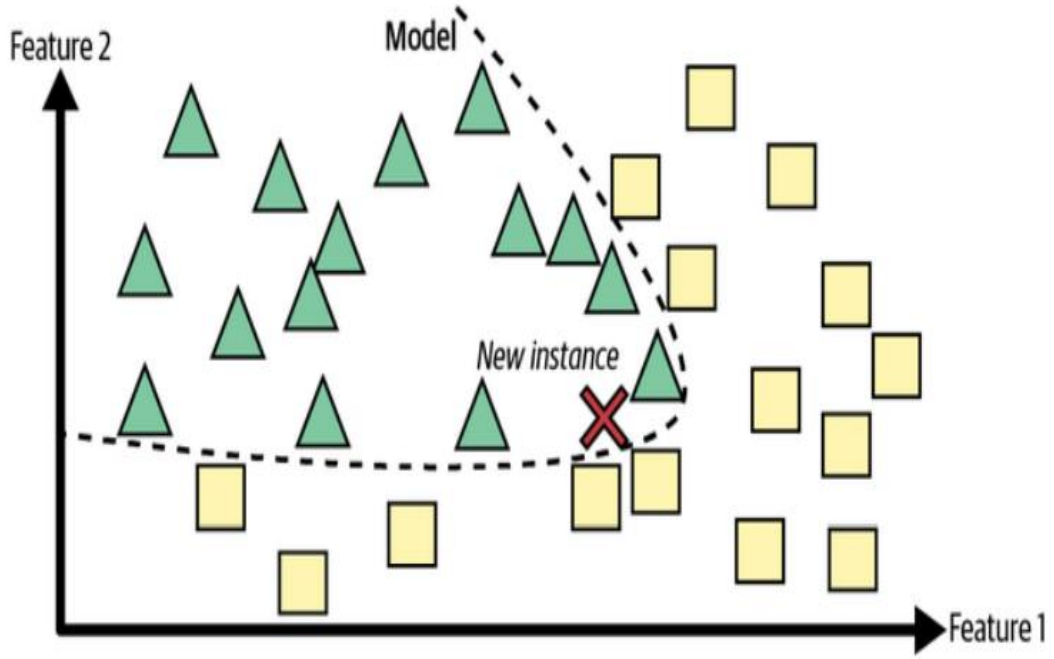
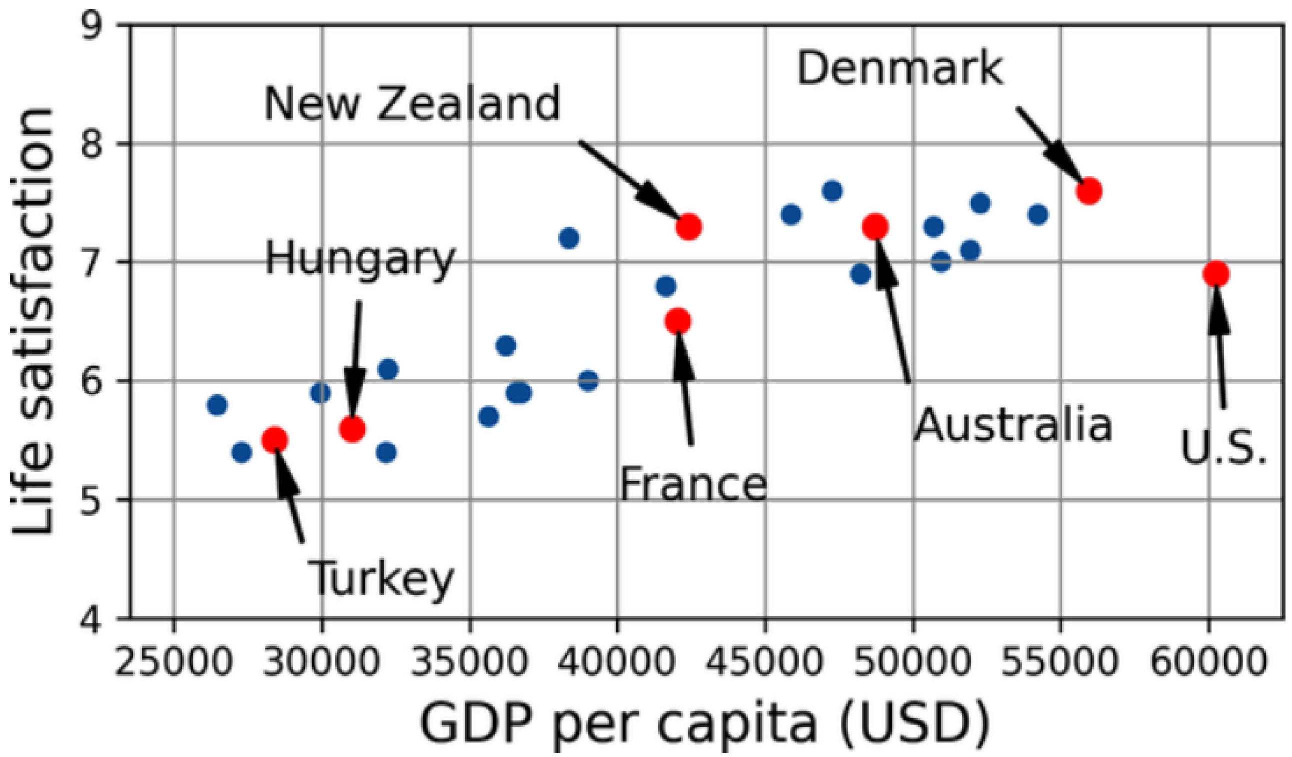


Figure 1-17. Model-based learning

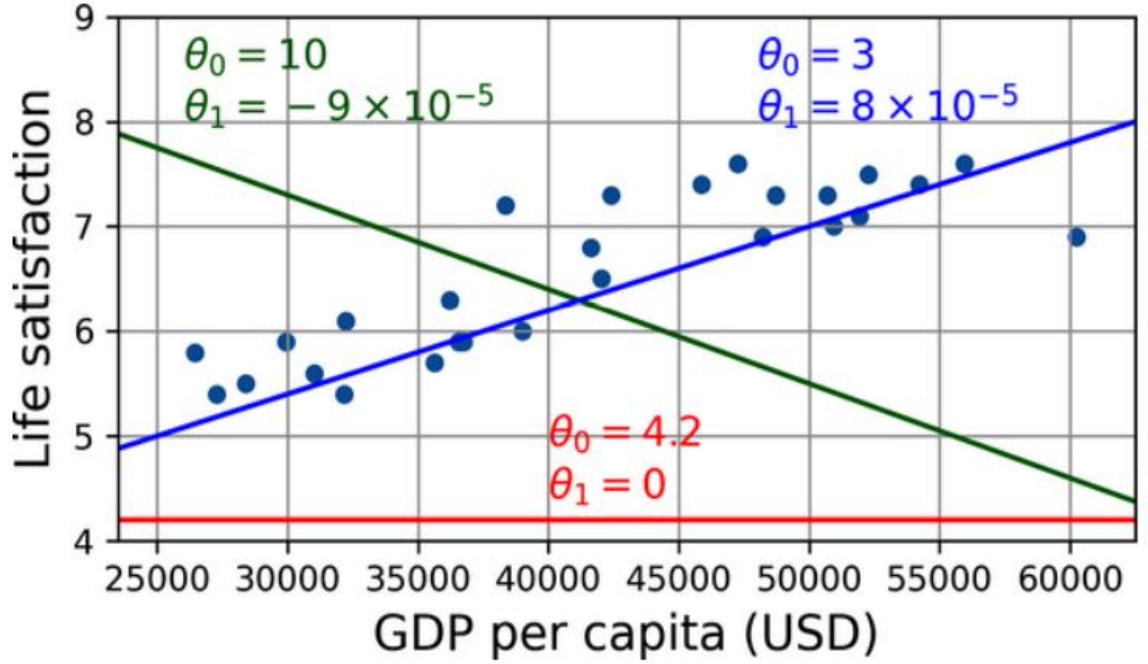


$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{GDP\_per\_capita}$$

This model has two model parameters, θ and θ .

Before you can use your model, you need to define the parameter values $\theta_0$ and $\theta_1$.

The final trained model ready to be used for predictions (e.g., linear regression with one input and one output, using $\theta_0$ = 3.75 and $\theta_1$ = 6.78 × 10 ).

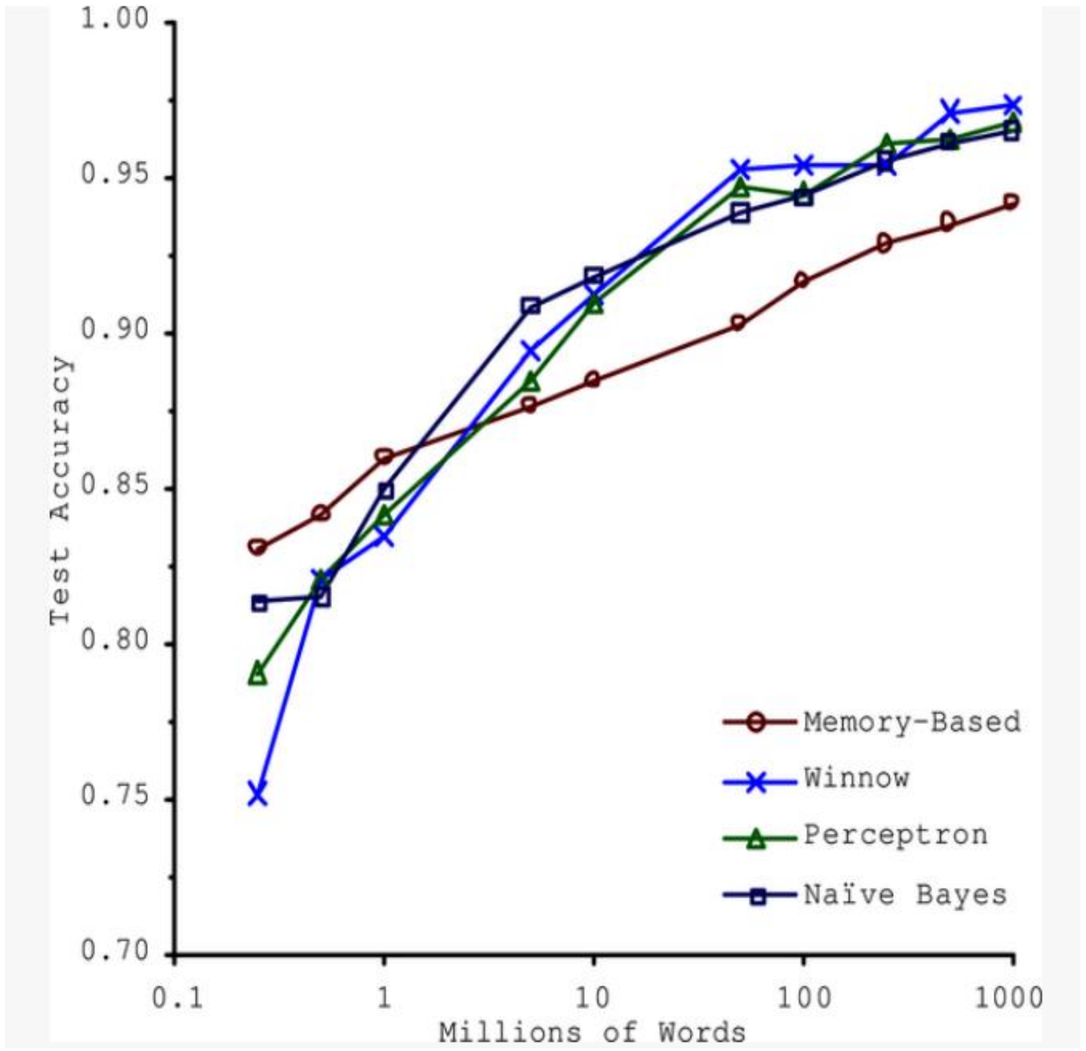


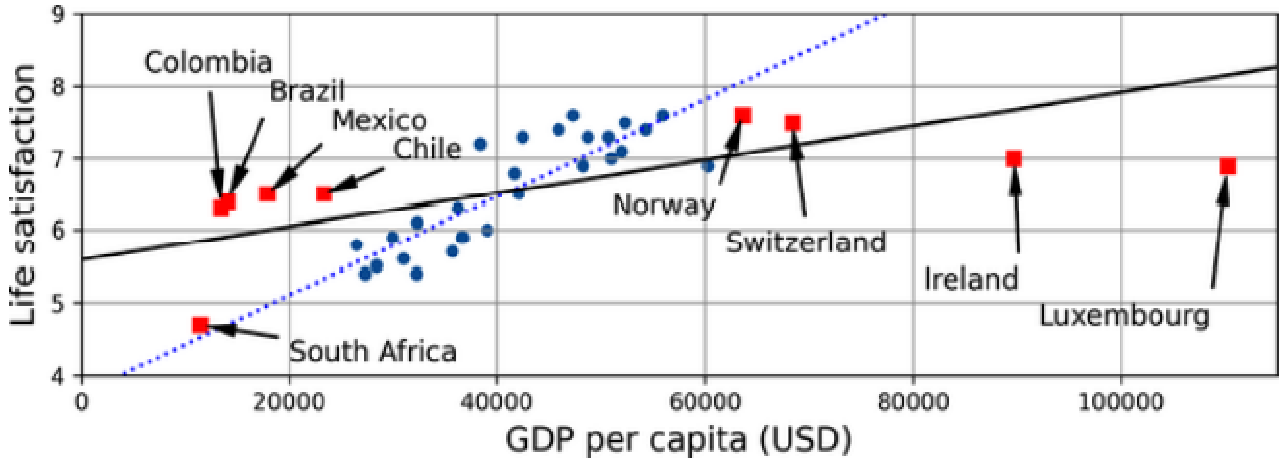
If all went well, your model will make good predictions. If not, you may need to use more attributes (employment rate, health, air pollution, etc.), get more or better-quality training data, or perhaps select a more powerful model (e.g., a polynomial regression model).

- You studied the data.
- You selected a model.
- You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
- Finally, you applied the model to make predictions on new cases (this is called inference), hoping that this model will generalize well.

## Main Challenges of Machine Learning

- Insufficient Quantity of Training Data
- Nonrepresentative Training Data
- Poor-Quality Data
- Irrelevant Features
- Overfitting the Training Data
- Underfitting the Training Data


Test Accuracy
Millions of Words
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.1
1
10
100
1000
Memory-Based
Winnow
Perceptron
Naïve Bayes


Life satisfaction
GDP per capita (USD)
Colombia
Brazil
Mexico
Chile
Norway
Switzerland
Ireland
Luxembourg
South Africa
0
20000
40000
60000
80000
100000
4
5
6
7
8
9

# Linear Regression

@ life_satisfaction = $\theta_0 + \theta_1 \times$ GDP_per_capita.

This model is just a linear function of the input feature GDP_per_capita. $\theta_0$ and $\theta_1$ are the model's parameters.

Linear regression model prediction

$$y\hat{} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- $x_i$ is the i-feature value.
- $\theta_j$ is the j model parameter, including the bias term $\theta_0$ and the feature weights $\theta_1$ , $\theta_2$ , ⋯, $\theta_n$ .

- Linear regression model prediction (vectorized form)

$$\hat{y} = h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta} \cdot x$$

  - $h_{\boldsymbol{\theta}}$ is the hypothesis function, using the model parameters $\boldsymbol{\theta}$.
  - $\boldsymbol{\theta}$ is the model's *parameter vector*, containing the bias term $\theta_0$ and the feature weights $\theta_1$ to $\theta_n$.
  - $\mathbf{x}$ is the instance's *feature vector*, containing $x_0$ to $x_n$, with $x_0$ always equal to 1.
  - $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$.

The MSE of a linear regression hypothesis h on a training set X is calculated using *MSE cost function for a linear regression model*

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left( \boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

## The Normal Equation

- To find the value of θ that minimizes the MSE, there exists a closed-form solution— in other words, a mathematical equation that gives the result directly. This is called the Normal equation

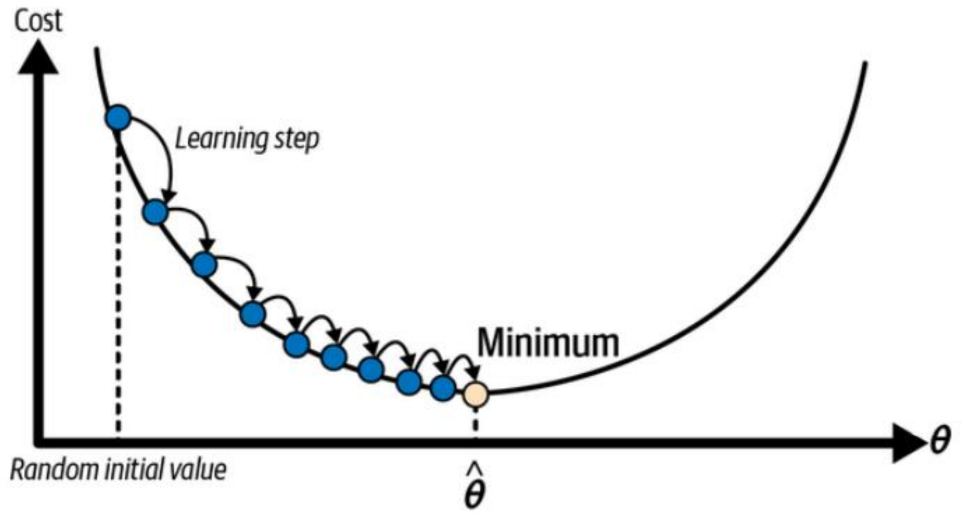
$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^\intercal \mathbf{X}\right)^{-1} \mathbf{X}^\intercal \mathbf{y}$$

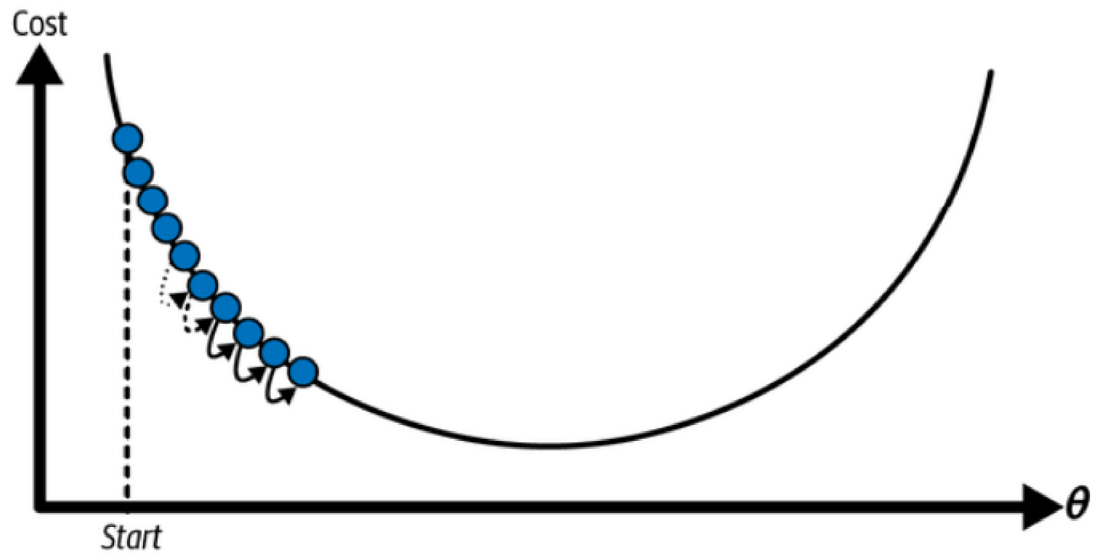
$\boldsymbol{\theta}$ˆ is the value of θ that minimizes the cost function

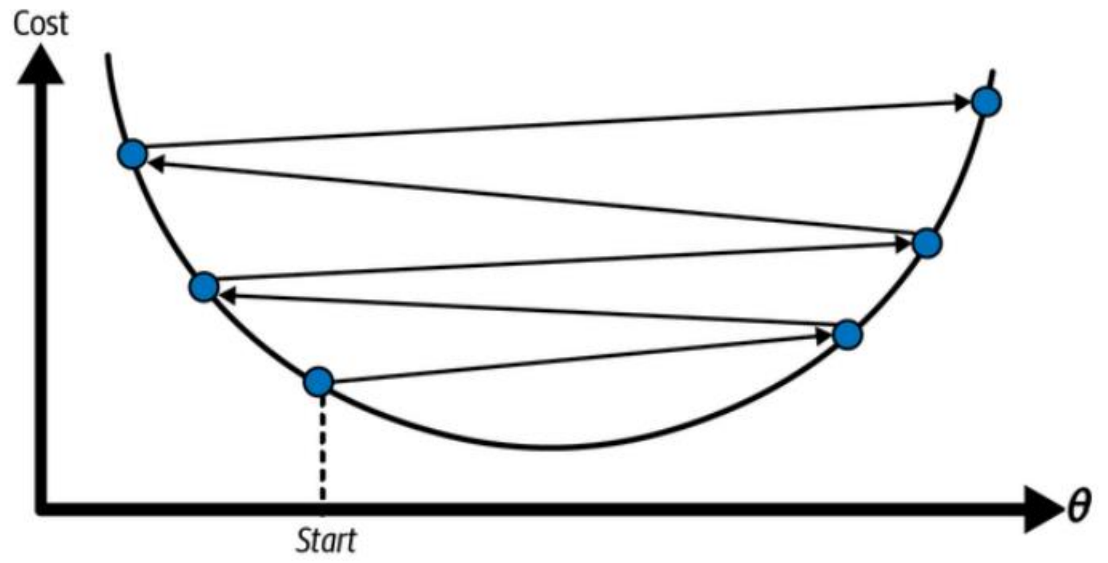
y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$.

# Gradient Descent

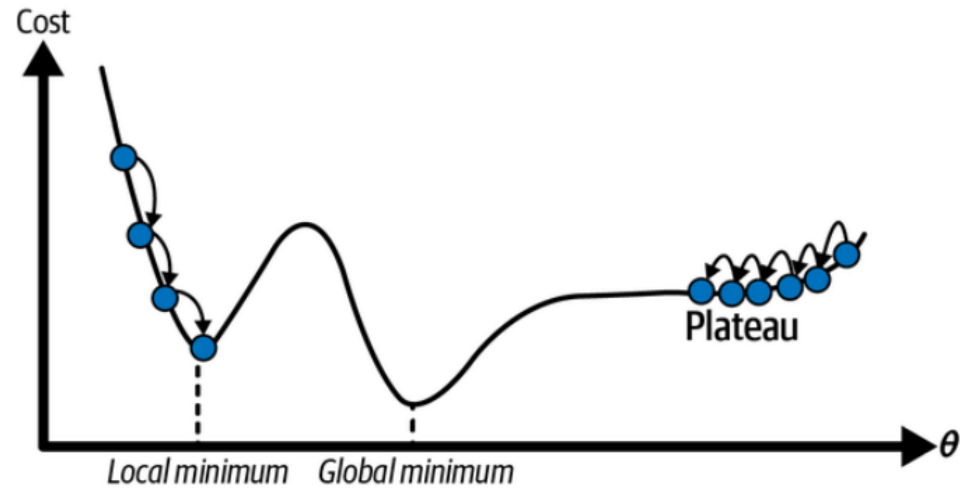
- Gradient descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of gradient descent is to tweak parameters iteratively in order to minimize a cost function.
- In practice, you start by filling θ with random values (this is called random initialization). Then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum.



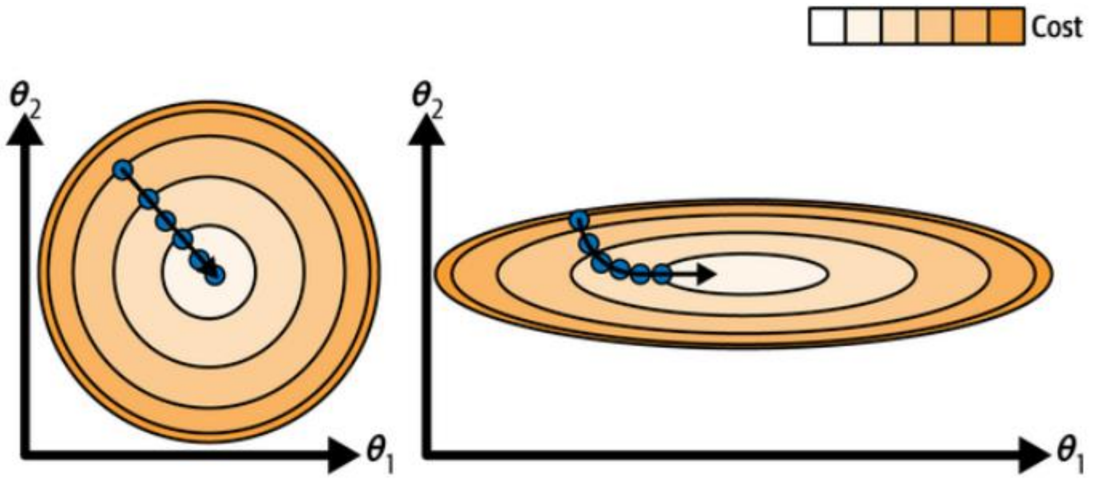




The two main challenges with gradient descent. If the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau. And if you stop too early, you will never reach the global minimum.



The MSE cost function for a linear regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them is never below the curve. This implies that there are no local minima, just one global minimum.



Gradient descent with (left) and without (right) feature scaling

**Batch Gradient Descent**

To implement gradient descent, you need to compute the gradient of the cost function regarding each model parameter θ . In other words, you need to calculate how much the cost function will change if you change $\theta_j$ just a little bit.

Computes the partial derivative of the MSE regarding parameter θ , noted $\partial_j$ MSE(θ) / ∂θ

Partial derivatives of the cost function :

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left( \boldsymbol{\theta}^\intercal \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

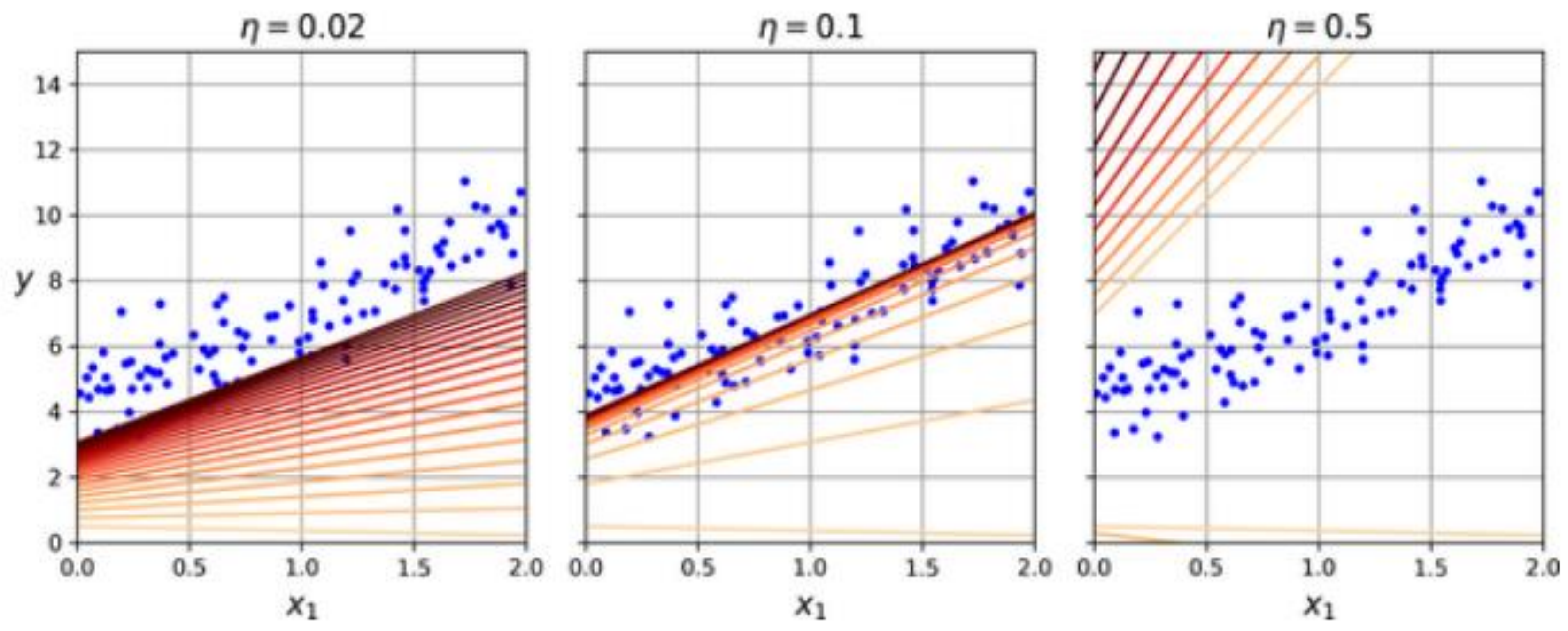
Gradient vector of the cost function

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\intercal (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Gradient descent step

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

learning rate η

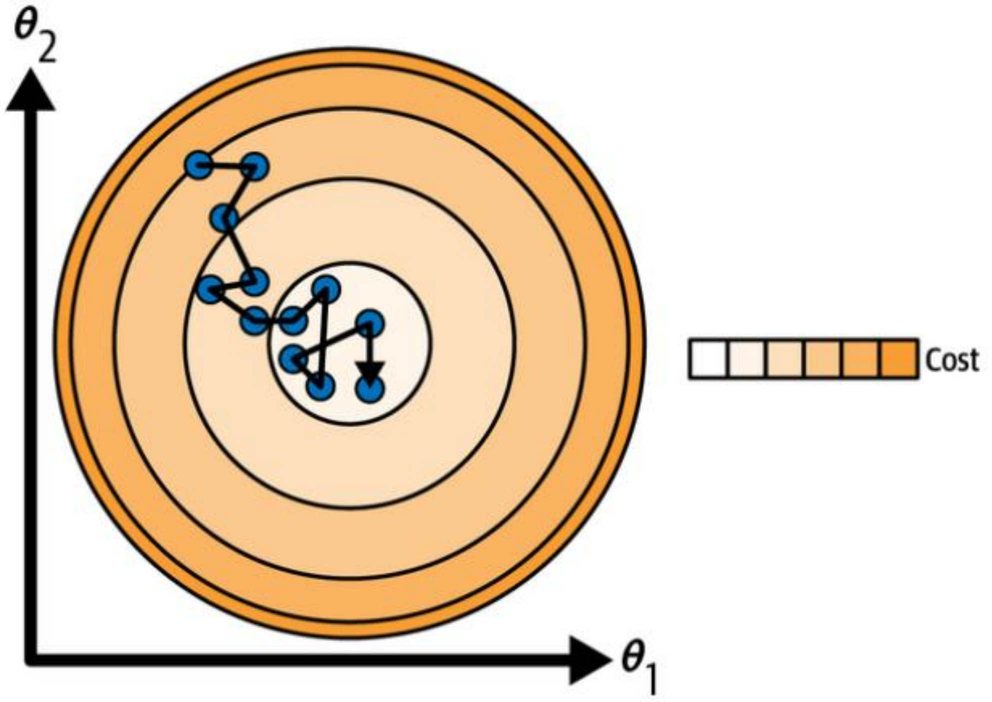


Gradient descent with various learning rates

On the left, the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time. In the middle, the learning rate looks pretty good: in just a few epochs, it has already converged to the solution. On the right, the learning rate is too high: the algorithm diverges, jumping all over the place and actually getting further and further away from the solution at every step.

## Stochastic Gradient Descent

- The main problem with batch gradient descent is the fa that it uses the whole training set to compute the gradien at every step, which makes it very slow when the trainir set is large.
- At the opposite extreme, stochastic gradient descent pic a random instance in the training set at every step ar computes the gradients based only on that single instance
- Working on a single instance at a time makes the algorith much faster because it has very little data to manipulate every iteration.



- It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration
- On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than batch gradient descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down.
- Once the algorithm stops, the final parameter values will be good, but not optimal.

To minimize the cost function, the model needs to have the best value of θ$_1$ and θ$_2$.

Initially model selects θ$_1$ and θ$_2$ values randomly and then iteratively update these value in order to minimize the cost function until it reaches the minimum.

By the time model achieves the minimum cost function, it will have the best θ$_1$ and θ$_2$ values.

Using these finally updated values of θ$_1$ and θ$_2$ in the hypothesis equation of linear equation, the model predicts the value of x in the best manner it can.

**Cost Function**

$$J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[\underset{\substack{\uparrow \\ \text{Predicted Value}}}{h_\Theta(x_i)} - \underset{\substack{\uparrow \\ \text{True Value}}}{y_i}]^2$$

**Gradient Descent**

$$\boxed{\Theta_j = \Theta_j - \underset{\substack{\uparrow \\ \text{Learning Rate}}}{\alpha}\frac{\partial}{\partial \Theta_j}J(\Theta_0, \Theta_1)}$$

**Now,**

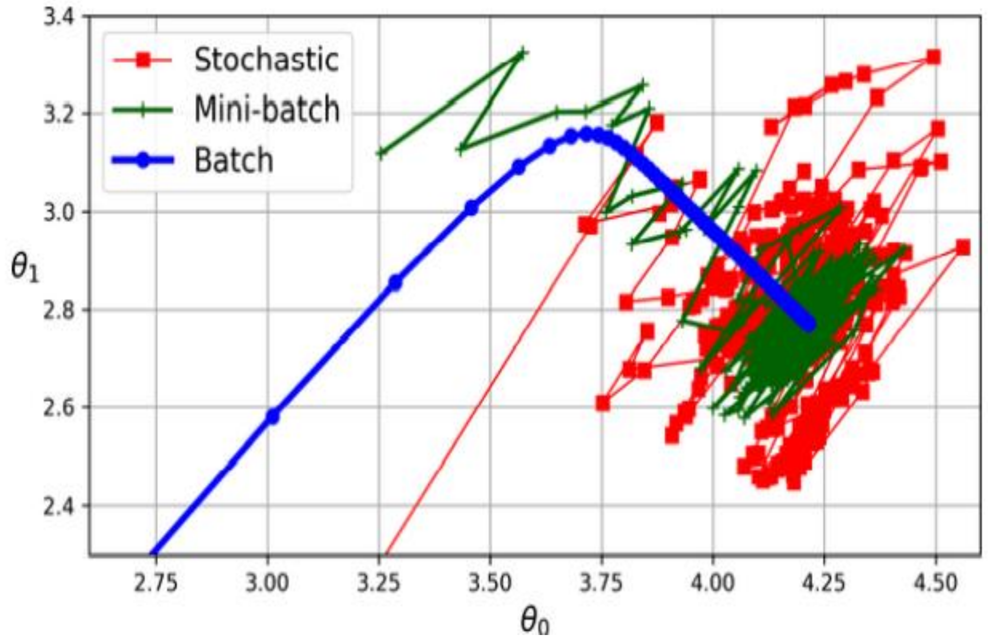
$$\begin{aligned}\frac{\partial}{\partial \Theta}J_\Theta &= \frac{\partial}{\partial \Theta}\frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y]^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x_i) - y)\frac{\partial}{\partial \Theta_j}(\Theta x_i - y) \\ &= \frac{1}{m}(h_\Theta(x_i) - y)x_i\end{aligned}$$

**Therefore,**

$$\boxed{\Theta_j := \Theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h_\Theta(x_i) - y)x_i]}$$

# Mini-Batch Gradient Descent

- Mini-batch GD computes the gradients on small random sets of instances called mini-batches. The main advantage of mini-batch GD over stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.

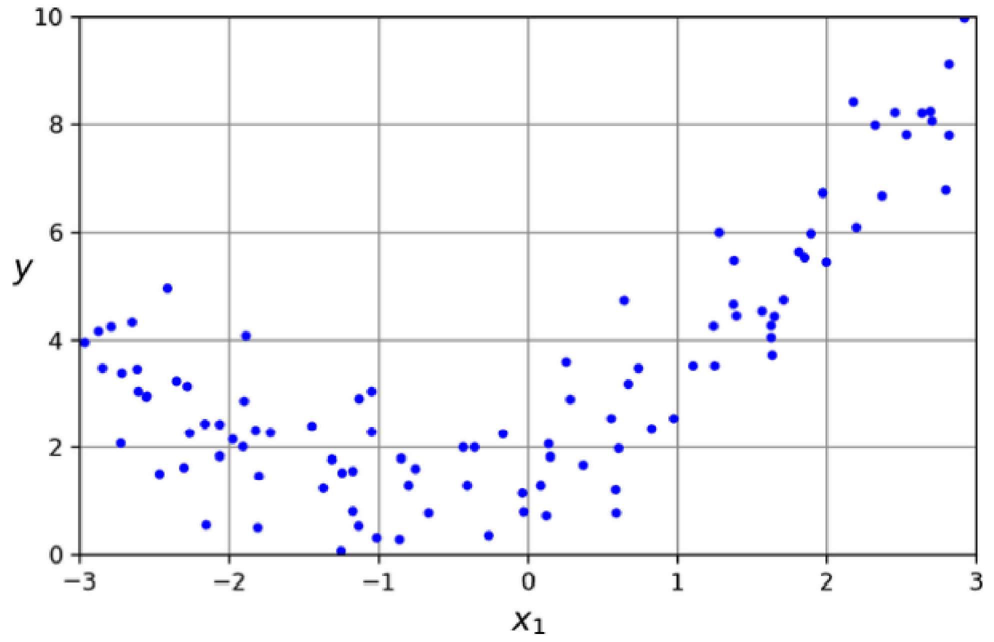


# Polynomial Regression

- A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features. This technique is called polynomial regression.

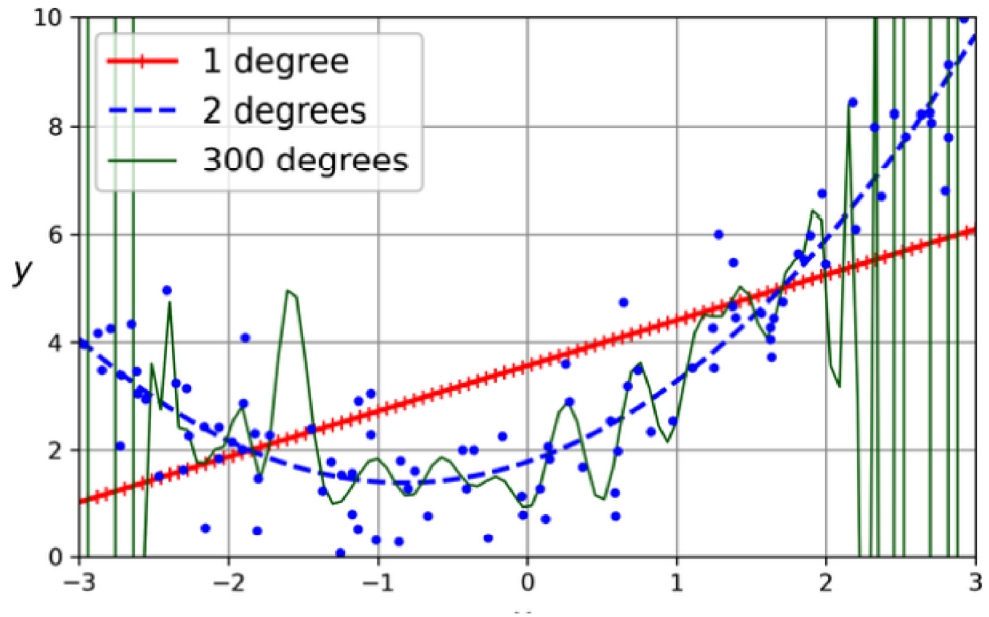
Clearly, a straight line will never fit this data properly. So, let's use Scikit-Learn's PolynomialFeatures class to transform our training data, adding the square (second-degree polynomial) of each feature in the training set as a new feature.

For e.g., if there were two features a and b, PolynomialFeatures with degree=3 would not only add the features $a^2$ , $a^3$ , $b^2$ , and $b^3$ , but also the combinations ab, $a^2b$, and $ab^2$ .



## Learning Curves

- This high-degree polynomial regression model is severely overfitting the training data, while the linear model is underfitting it.
- The model that will generalize best in this case is the quadratic model, which makes sense because the data was generated using a quadratic model.



# THE BIAS/VARIANCE TRADE-OFF

- An important theoretical result of statistics and machine learning is the fact that a model's generalization error can be expressed as the sum of three very different errors:
- **Bias** This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic.
- A high-bias model is most likely to underfit the training data.

- **Variance** This part is due to the model's excessive sensitivity to small variations in the training data.
- A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance and thus overfit the training data.
- **Irreducible error** This part is due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data (e.g., fix the data sources, such as broken sensors, or detect and remove outliers).

Increasing a model's complexity will typically increase its variance and reduce its bias. Conversely, reducing a model's complexity increases its bias and reduces its variance

## Regularized Linear Models

- A simple way to regularize a polynomial model is to reduce the number of polynomial degrees. For a linear model, regularization is typically achieved by constraining the weights of the model. We will now look at ridge regression, lasso regression, and elastic net regression.
- **Ridge Regression:** This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible. Note that the regularization term should only be added to the cost function during training.
- The hyperparameter α controls how much you want to regularize the model. If α = 0, then ridge regression is just linear regression. If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m} \sum_{i=1}^{n} {\theta_i}^2$$

If we define w as the vector of feature weights ($\theta_1$ to $\theta_n$), then the regularization term is equal to $\alpha(\| w \|_2)^2 / m$, where $\| w \|_2$ represents the ℓ2 norm of the weight vector.

- Lasso Regression: Least absolute shrinkage and selection operator regression (usually simply called lasso regression) is another regularized version of linear regression: just like ridge regression, it adds a regularization term to the cost function, but it uses the ℓ1 norm of the weight vector.

- Notice that the ℓ1 norm is multiplied by 2α, whereas the ℓ2 norm was multiplied by α / m in ridge regression.

- These factors were chosen to ensure that the optimal α value is independent from the training set size: different norms lead to different factors

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + 2\alpha \sum_{i=1}^{n} |\theta_i|$$

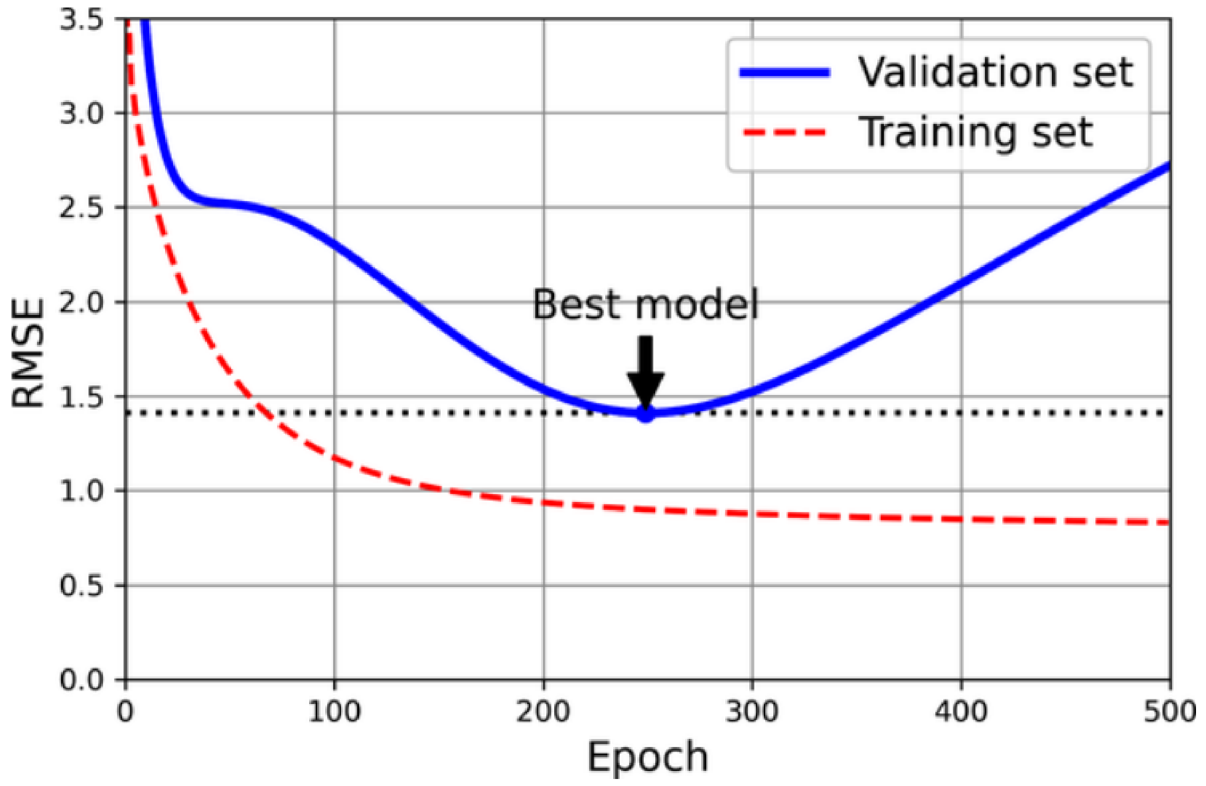
Elastic Net Regression: Elastic net regression is a middle ground between ridge regression and lasso regression. The regularization term is a weighted sum of both ridge and lasso's regularization terms, and you can control the mix ratio r. When r = 0, elastic net is equivalent to ridge regression, and when r = 1, it is equivalent to lasso regression

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\left(2\alpha \sum_{i=1}^{n} |\theta_i|\right) + (1-r)\left(\frac{\alpha}{m} \sum_{i=1}^{n} \theta_i^2\right)$$

Ridge is a good default, but if you suspect that only a few features are useful, you should prefer lasso or elastic net because they tend to reduce the useless features' weights down to zero, as discussed earlier . In general, elastic net is preferred over lasso because lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

# Early Stopping

- A very different way to regularize iterative learning algorithms such as gradient descent is to stop training as soon as the validation error reaches a minimum. This is called early stopping.
- With early stopping you just stop training as soon as the validation error reaches the minimum. It is such a simple and efficient regularization technique that Geoffrey Hinton called it a "beautiful free lunch"



With stochastic and mini-batch gradient descent, the curves are not so smooth, and it may be hard to know whether you have reached the minimum or not. One solution is to stop only after the validation error has been above the minimum for some time then roll back the model parameters to the point where the validation error was at a minimum.