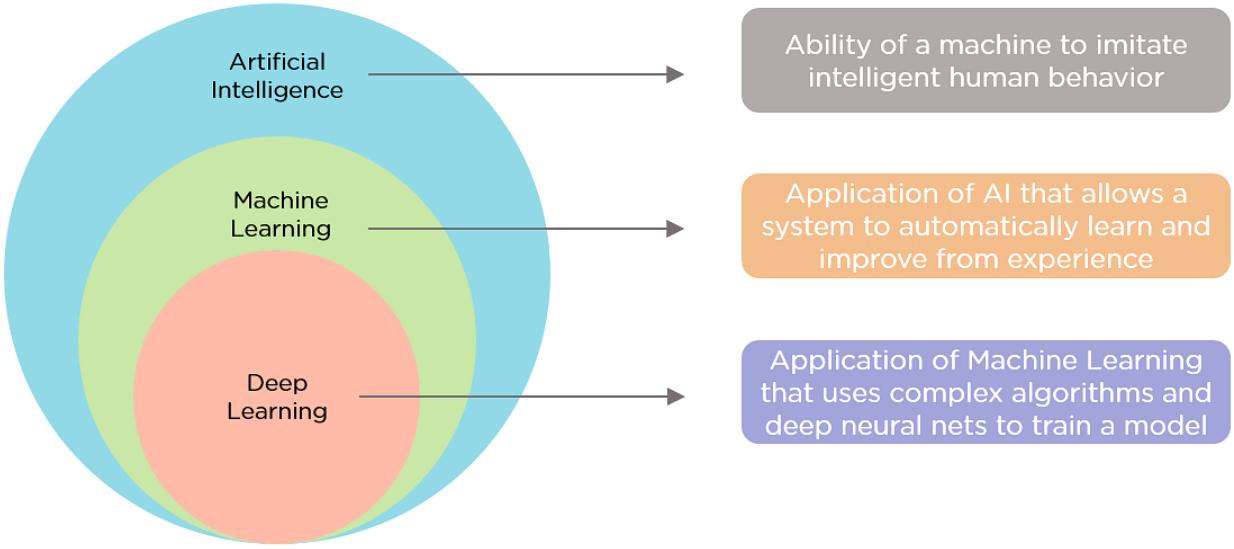
Artificial Intelligence and its subsets

19 April 2024 15:13

While artificial intelligence (AI), machine learning (ML), deep learning and neural networks are related technologies, the terms are often used interchangeably, which frequently leads to confusion about their differences.

- *Artificial intelligence is the overarching system.*
- *Machine learning is a subset of AI.*
- *Deep learning is a subfield of machine learning, and neural networks make up the backbone of deep learning algorithms. It's the number of node layers, or depth, of neural networks that distinguishes a single neural network from a deep learning algorithm, which must have more than three.*



What is AI? - The concept of creating smart intelligent machines.

- Artificial intelligence, or AI, is technology that enables computers and machines to simulate human intelligence and problem -solving capabilities.
- On its own or combined with other technologies (e.g., sensors, geolocation, robotics) AI can perform tasks that would otherwi se require human intelligence or intervention.
- AI focuses on 3 major aspects:
  - Learning
  - Reasoning
  - Self-Correction

  to obtain the maximum efficiency possible. These are called **cognitive functions (any mental/intellectual activity).**

Three main categories of AI are:

- Artificial Narrow Intelligence (ANI)
- Artificial General Intelligence (AGI)
- Artificial Super Intelligence (ASI)

ANI is considered "weak" AI, whereas the other two types are classified as "strong" AI. We define weak AI by its ability to complete a specific task, like winning a chess game or identifying a particular individual in a series of photos. Natural language processing (NLP) and computer vision, which let companies automate tasks and underpin chatbots and virtual assistants such as Siri and Alexa, are examples of ANI. Computer vision is a factor in the development of self-driving cars.

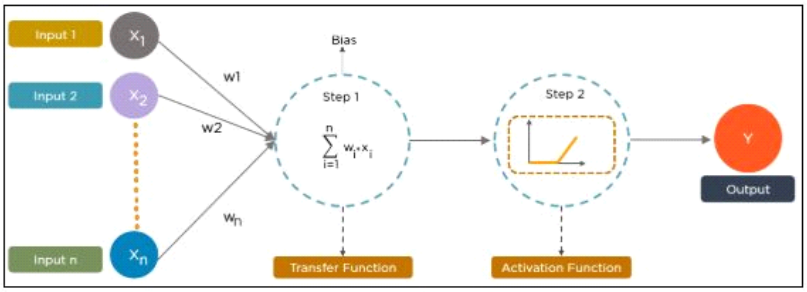
Stronger forms of AI, like AGI and ASI, incorporate human behaviors more prominently, such as the ability to interpret tone and emotion. Strong AI is defined by its ability compared to humans. Artificial General Intelligence (AGI) would perform on par with another human, while Artificial Super Intelligence (ASI)—also known as superintelligence—would surpass a human's intelligence and ability. Neither form of Strong AI exists yet, but research in this field is ongoing.

Working:

Here are the steps:

1. Calculate the weighted sums.
2. The calculated sum of weights is passed as input to the activation function.
3. The activation function takes the "weighted sum of input" as the input to the function, adds a bias, and decides whether the neuron should be fired or not.
4. The output layer gives the predicted output.

5. The model output is compared with the actual output. After training the neural network, the model uses the backpropagation me thod to improve the performance of the network. The cost function helps to reduce the error rate.



Here are some common examples:

- **Speech recognition:** speech recognition systems use deep learning algorithms to recognize and classify images and speech. These systems are used in a variety of applications, such as self-driving cars, security systems, and medical imaging.
- **Personalized recommendations:** E-commerce sites and streaming services like Amazon and Netflix use AI algorithms to analyze users' browsing and viewing history to recommend products and content that they are likely to be interested in.
- **Predictive maintenance:** AI-powered predictive maintenance systems analyze data from sensors and other sources to predict when equipment is likely to fail, helping to reduce downtime and maintenance costs.
- **Medical diagnosis:** AI-powered medical diagnosis systems analyze medical images and other patient data to help doctors make more accurate diagnoses and treatment plans.
- **Autonomous vehicles:** Self-driving cars and other autonomous vehicles use AI algorithms and sensors to analyze their environment and make decisions about speed, direction, and other factors.
- **Virtual Personal Assistants (VPA) like Siri or Alexa –** these use natural language processing to understand and respond to user requests, such as playing music, setting reminders, and answering questions.
- **Autonomous vehicles –** self-driving cars use AI to analyze sensor data, such as cameras and lidar, to make decisions about navigation, obstacle avoidance, and route planning.
- **Fraud detection –** financial institutions use AI to analyze transactions and detect patterns that are indicative of fraud, such as unusual spend ing patterns or transactions from unfamiliar locations.
- **Image recognition –** AI is used in applications such as photo organization, security systems, and autonomous robots to identify objects, people, and scenes in images.
- **Natural language processing –** AI is used in chatbots and language translation systems to understand and generate human-like text.
- **Predictive analytics –** AI is used in industries such as healthcare and marketing to analyze large amounts of data and make predictions about future events, such as disease outbreaks or consumer behavior.
- **Game-playing AI –** AI algorithms have been developed to play games such as chess, Go, and poker at a superhuman level, by analyzing game data and making predictions about the outcomes of moves.

Machine Learning — An Approach to Achieve Artificial Intelligence

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that involves the use of algorithms and statistical models to allow a computer system to "learn" from data and improve its performance over time, without being explicitly programmed to do so.
Machine learning algorithms improve performance over time as they are trained—exposed to more data. Machine learning models are the output, or what the program learns from running an algorithm on training data. The more data used, the better the model will get.

Here are some examples of Machine Learning:

- **Image recognition:** Machine learning algorithms are used in image recognition systems to classify images based on their contents. These systems are used in a variety of applications, such as self-driving cars, security systems, and medical imaging.
- **Speech recognition:** Machine learning algorithms are used in speech recognition systems to transcribe speech and identify the words spoken. These systems are used in virtual assistants like Siri and Alexa, as well as in call centers and other applications.
- **Natural language processing (NLP):** Machine learning algorithms are used in NLP systems to understand and generate human language. These systems are used in chatbots, virtual assistants, and other applications that involve natural language interactions.
- **Recommendation systems:** Machine learning algorithms are used in recommendation systems to analyze user data and recommend products or services that are likely to be of interest. These systems are used in e-commerce sites, streaming services, and other applications.
- **Sentiment analysis:** Machine learning algorithms are used in sentiment analysis systems to classify the sentiment of text or speech as positive, negative, or neutral. These systems are used in social media monitoring and other applications.
- **Predictive maintenance:** Machine learning algorithms are used in predictive maintenance systems to analyze data from sensors and other sources to predict when equipment is likely to fail, helping to reduce downtime and maintenance costs.
- **Spam filters in email –** ML algorithms analyze email content and metadata to identify and flag messages that are likely to be spam.
- **Recommendation systems –** ML algorithms are used in e-commerce websites and streaming services to make personalized recommendations to users based on their browsing and purchase history.
- **Predictive maintenance –** ML algorithms are used in manufacturing to predict when machinery is likely to fail, allowing for proactive maintenance and reducing downtime.
- **Credit risk assessment –** ML algorithms are used by financial institutions to assess the credit risk of loan applicants, by analyzing data such as their income, employment history, and credit score.
- **Customer segmentation –** ML algorithms are used in marketing to segment customers into different groups based on their characteristics and

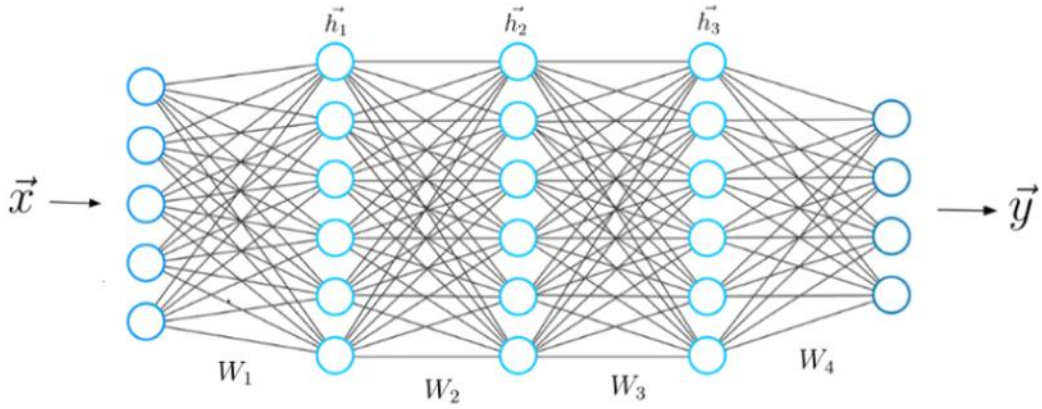
behavior, allowing for targeted advertising and promotions.

- **Fraud detection –** ML algorithms are used in financial transactions to detect patterns of behavior that are indicative of fraud, such as unusual spending patterns or transactions from unfamiliar locations.
- **Speech recognition –** ML algorithms are used to transcribe spoken words into text, allowing for voice-controlled interfaces and dictation software.

Deep Learning — A Technique for Implementing Machine Learning

Deep learning is a subfield of artificial intelligence that aims at mimicking the function of the human brain using neural networks.

Since deep learning algorithms also require data in order to learn and solve problems, we can also call it a subfield of machine learning. The terms machine learning and deep learning are often treated as synonymous. However, these systems have different capabilities.

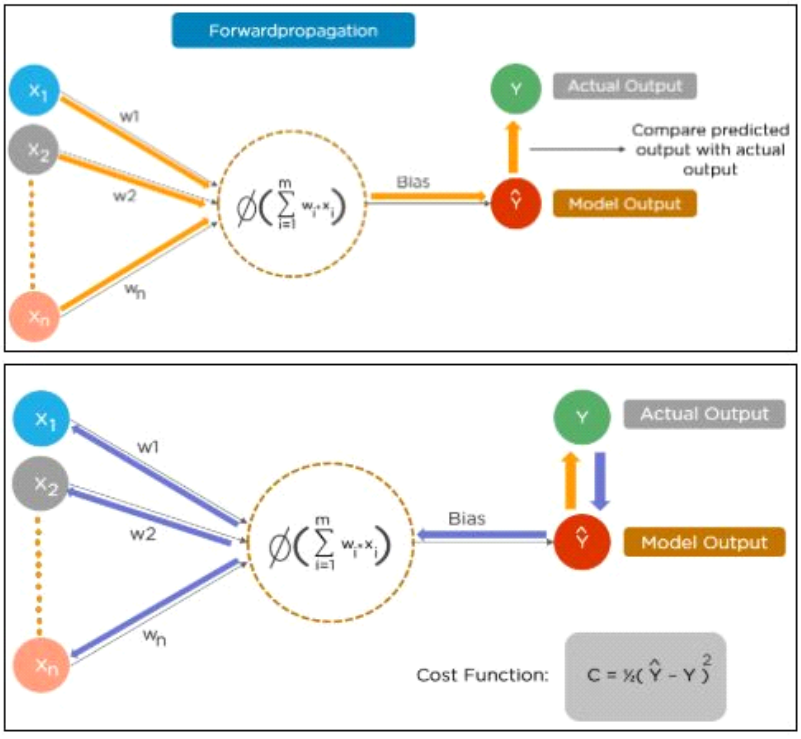


Unlike machine learning, deep learning uses a multi-layered structure of algorithms called the neural network.

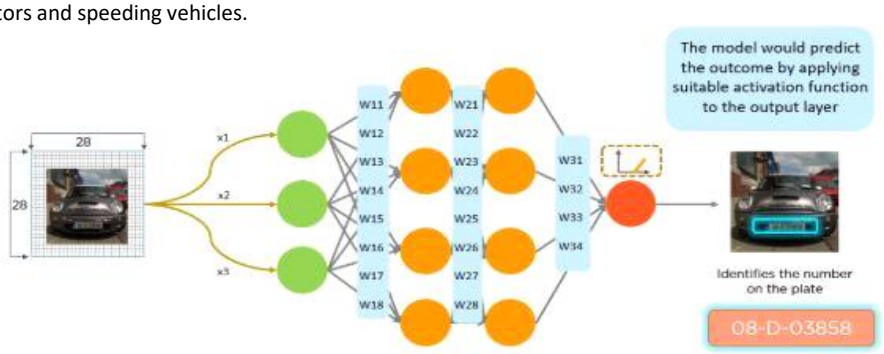
Artificial neural networks have unique capabilities that enable deep learning models to solve tasks that machine learning models could never solve.

All recent advances in intelligence are due to deep learning. Without deep learning we would not have self-driving cars, chatbots or personal assistants like Alexa and Siri. Google Translate would remain primitive and Netflix would have no idea which movies or TV series to suggest.

Neural networks are inspired by our understanding of the biology of our brains – all those interconnections between the neurons. But, unlike a biological brain where any neuron can connect to any other neuron within a certain physical distance, these artificial neural networks have discrete layers, connections, and directions of data propagation.
You might, for example, take an image, chop it up into a bunch of tiles that are inputted into the first layer of the neural network. In the first layer individual neurons, then passes the data to a second layer. The second layer of neurons does its task, and so on, until the final layer and the final output is produced.



In the following example, deep learning and neural networks are used to identify the number on a license plate. This technique is used by many countries to identify rules violators and speeding vehicles.



Deep learning automates much of the feature extraction piece of the process, eliminating some of the manual human intervention required. It also enables the use of large data sets, earning the title of scalable machine learning. That capability is exciting as we explore the use of ***unstructured data*** *further, particularly since over 80% of an organization's data is estimated to be unstructured.*

Here are some examples of Deep Learning:

- **Image and video recognition:** Deep learning algorithms are used in image and video recognition systems to classify and analyze visual data. These systems are used in self-driving cars, security systems, and medical imaging.
- **Generative models:** Deep learning algorithms are used in generative models to create new content based on existing data. These systems are used in image and video generation, text generation, and other applications.
- **Autonomous vehicles:** Deep learning algorithms are used in self-driving cars and other autonomous vehicles to analyze sensor data and make decisions about speed, direction, and other factors.
- **Image classification –** Deep Learning algorithms are used to recognize objects and scenes in images, such as recognizing faces in photos or identifying items in an image for an e-commerce website.
- **Speech recognition –** Deep Learning algorithms are used to transcribe spoken words into text, allowing for voice-controlled interfaces and dictation software.
- **Natural language processing –** Deep Learning algorithms are used for tasks such as sentiment analysis, language translation, and text generation.
- **Recommender systems –** Deep Learning algorithms are used in recommendation systems to make personalized recommendations based on users' behavior and preferences.
- **Fraud detection –** Deep Learning algorithms are used in financial transactions to detect patterns of behavior that are indicative of fraud, such as unusual spending patterns or transactions from unfamiliar locations.
- **Game-playing AI –** Deep Learning algorithms have been used to develop game-playing AI that can compete at a superhuman level, such as the AlphaGo AI that defeated the world champion in the game of Go.
- **Time series forecasting –** Deep Learning algorithms are used to forecast future values in time series data, such as stock prices, energy consumption, and weather patterns.

What is Machine Learning?

24 September 2022 18:31

Machine learning (ML) is a field of computer science that gives computers the ability to learn without being explicitly programmed. Traditionally, computers rely on following a set of instructions to complete a task. Machine learning takes a different approach. Instead of explicit instructions, machines are given data and a goal. The machine then learns from the data to achieve the goal.

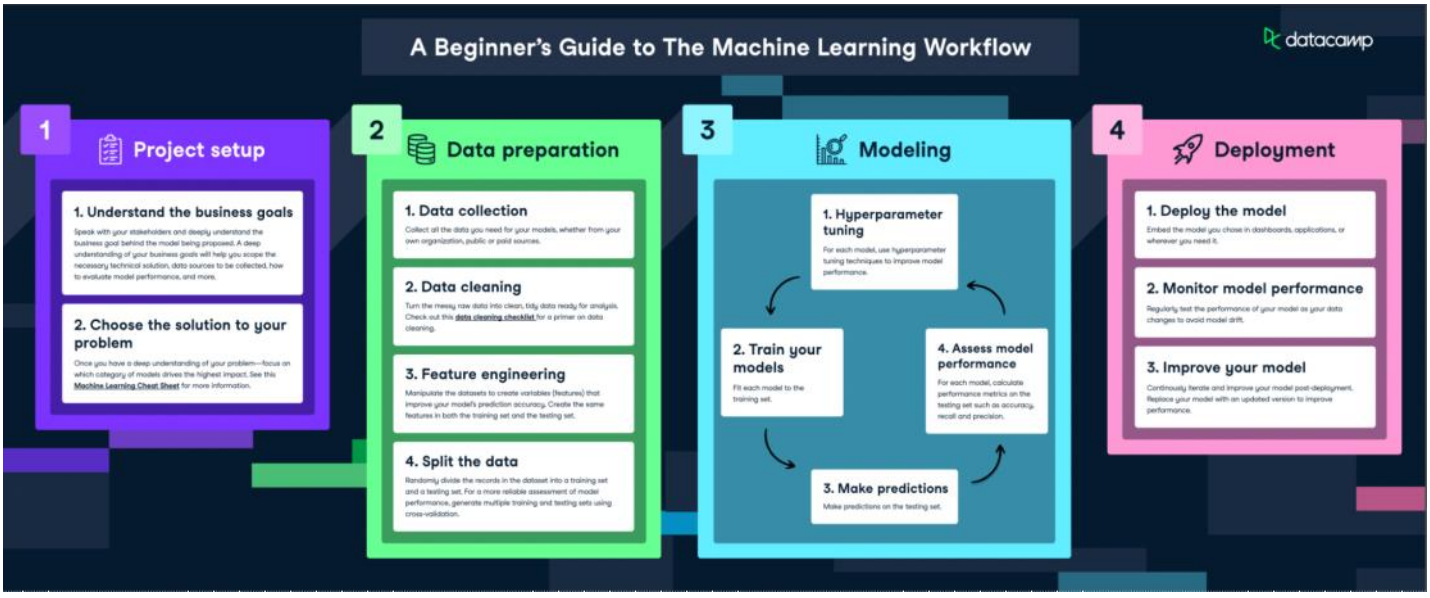
Arthur Samuel **wrote a checkers playing program** in the 1950s**.** The amazing thing about this program was that Arthur Samuel himself wasn't a very good checkers player. What he did was he had programmed the computer to play maybe tens of thousands of games against itself.
By watching what sorts of board positions tended to wins, and what positions tended to losses, the checkers playing program learned over time what are good or bad board positions. By trying to get a good position and avoid bad positions this program learned to get better and better at playing checkers because the computer had the patience to play tens of thousands of games against itself.

It was able to get so much checkers playing experience that eventually it became a better checkers player than also, Samuel himself.

The two main types of machine learning are:
- supervised learning
- unsupervised learning.

Of these two, supervised learning is the type of machine learning that is used most in many real-world applications and has seen the most rapid advancements and innovation. In this specialization, which has three courses in total, the first and second courses will focus on supervised learning, and the third will focus on unsupervised learning, recommender systems, and reinforcement learning.
By far, the most used types of learning algorithms today are supervised learning, unsupervised learning, and recommender systems.

Machine Learning Methodology



Step 1: Data collection

The first step in the machine learning process is data collection. Data is the lifeblood of machine learning - the quality and quantity of your data can directly impact your model's performance. Data can be collected from various sources such as databases, text files, images, audio files, or even scraped from the web.
Once collected, the data needs to be prepared for machine learning. This process involves organizing the data in a suitable format, such as a CSV file or a database, and ensuring that the data is relevant to the problem you're trying to solve.

Step 2: Data preprocessing

Data preprocessing is a crucial step in the machine learning process. It involves cleaning the data (removing duplicates, correcting errors), handling missing data (either by removing it or filling it in), and normalizing the data (scaling the data to a standard format).
Preprocessing improves the quality of your data and ensures that your machine learning model can interpret it correctly. This step can significantly improve the accuracy of your model.

Step 3: Choosing the right model

Once the data is prepared, the next step is to choose a machine learning model. There are many types of models to choose from, including linear regression, decision trees, and neural networks. The choice of model depends on the nature of your data and the problem you're trying to solve.

Factors to consider when choosing a model include the size and type of your data, the complexity of the problem, and the computational resources available.

Step 4: Training the model

After choosing a model, the next step is to train it using the prepared data. Training involves feeding the data into the model and allowing it to adjust its internal parameters to better predict the output.
During training, it's important to avoid overfitting (where the model performs well on the training data but poorly on new data) and underfitting (where the model performs poorly on both the training data and new data).

Step 5: Evaluating the model

Once the model is trained, it's important to evaluate its performance before deploying it. This involves testing the model on new data it hasn't seen during training.
Common metrics for evaluating a model's performance include accuracy (for classification problems), precision and recall (for binary classification problems), and mean squared error (for regression problems).

Step 6: Hyperparameter tuning and optimization

After evaluating the model, you may need to adjust its hyperparameters to improve its performance. This process is known as parameter tuning or hyperparameter optimization.
Techniques for hyperparameter tuning include grid search (where you try out different combinations of parameters) and cross validation (where you divide your data into subsets and train your model on each subset to ensure it performs well on different data).

Step 7: Predictions and deployment

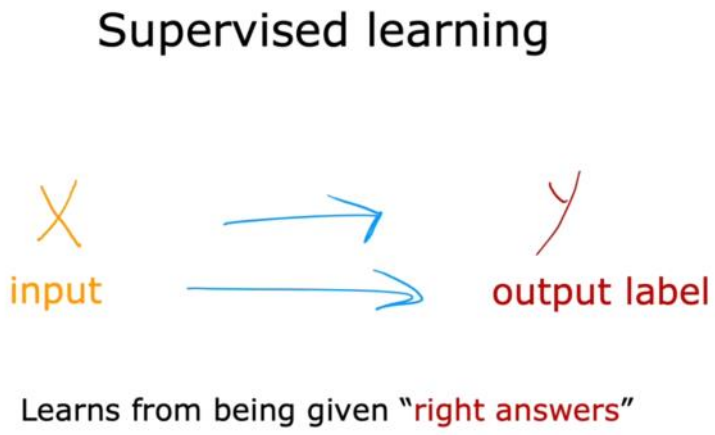
Once the model is trained and optimized, it's ready to make predictions on new data. This process involves feeding new data into the model and using the model's output for decision-making or further analysis.
Deploying the model involves integrating it into a production environment where it can process real-world data and provide real-time insights. This process is often known as MLOps.

What is Supervised Learning?

24 September 2022 19:02

Supervised learning is a subcategory of machine learning and artificial intelligence that is defined by its use of labeled data sets to train algorithms to classify data or predict outcomes accurately.
As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. It can be used to build highly accurate machine learning models.

Machine learning is creating tremendous economic value today. Roughly 99 percent of the economic value created by machine learning today is through one type of machine learning, which is called supervised learning.



Supervised learning, refers to algorithms that learn x to y or input to output mappings. The key characteristic of supervised learning is that you give your learning algorithm has examples to learn from. That includes the right answers, whereby right answer, I mean, the correct label y for a given input x, and is by seeing correct pairs of input x and desired output label y that the learning algorithm eventually learns to take just the input alone without the output label and gives a reasonably accurate prediction or guess of the output.

How supervised learning works

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

- Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
- Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Let us take some examples:

Input (X)	Output (Y)	Application
email	spam? (0/1)	spam filtering
audio	text transcripts	speech recognition
English	Spanish	machine translation
ad, user info	click? (0/1)	online advertising
image, radar info	position of other cars	self-driving car
image of phone	defect? (0/1)	visual inspection

Let's dive more deeply into one specific example. Say you want to predict housing prices based on the size of the house. You've collected some data and say you plot the data and it looks like this -->

Here on the horizontal axis is the size of the house in square feet. Here on the vertical axis is the price of the house in, say, thousands of dollars. With this data, let's say a friend wants to know what's the price for their 750 square foot house. How can the learning algorithm help you?

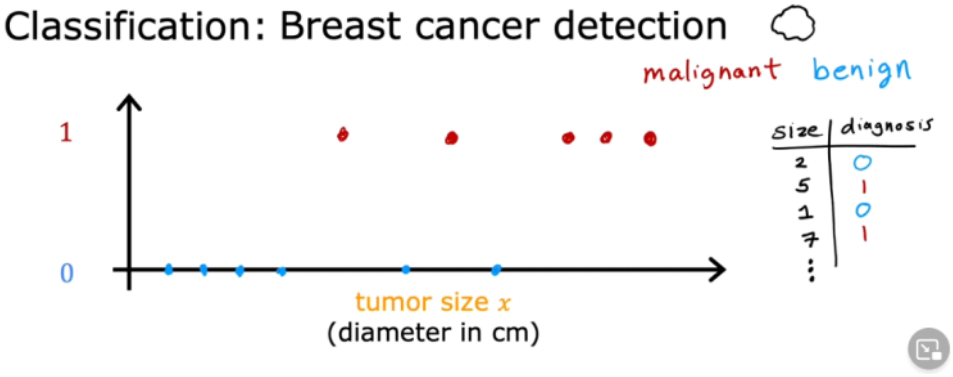
One thing a learning algorithm might be able to do is say, for the straight line to the data and reading off the straight line, it looks like your friend's house could be sold for maybe about, I don't know, $150,000.

But fitting a straight line isn't the only learning algorithm you can use. There are others that could work better for this application. For example, routed and fitting a straight line,
you might decide that it's better to fit a curve, a function that's slightly more complicated or more complex than a straight line. If you do that and make a prediction here, then it looks like, well, your friend's house could be sold for closer to $200,000.

What you've seen in this slide is an example of supervised learning. Because we gave the algorithm a dataset in which the so-called right answer, that is the label or the correct price y is given for every house on the plot. The task of the learning algorithm is to produce more of these right answers, specifically predicting what is the likely price for other houses like your friend's house.

To define a little bit more terminology, this housing price prediction is the particular type of supervised learning called **regression**. ***By regression, we're trying to predict a number from infinitely many possible numbers*** such as the house prices in our example, which could be 150,000 or 70,000 or 183,000 or any other number in between.

There's a second major type of supervised learning algorithm called a classification algorithm. Take breast cancer detection as an example of a classification problem.



Say you're building a machine learning system so that doctors can have a diagnostic tool to detect breast cancer. This is important because early detection could potentially save a patient's life. Using a patient's medical records your machine learning system tries to figure out if a tumor (that is a lump) is malignant, meaning cancerous or dangerous. Or if that tumor is benign, meaning that it's just a lump that isn't cancerous and isn't that dangerous?

So maybe your dataset has tumors of various sizes. And these tumors are labeled as either:

→ **benign,** in this example with a 0

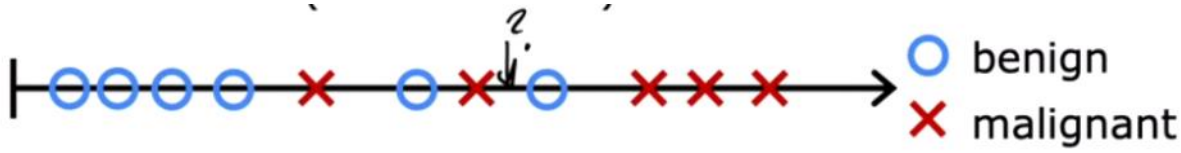
or

→ **malignant,** which will be designated in this example with a 1.

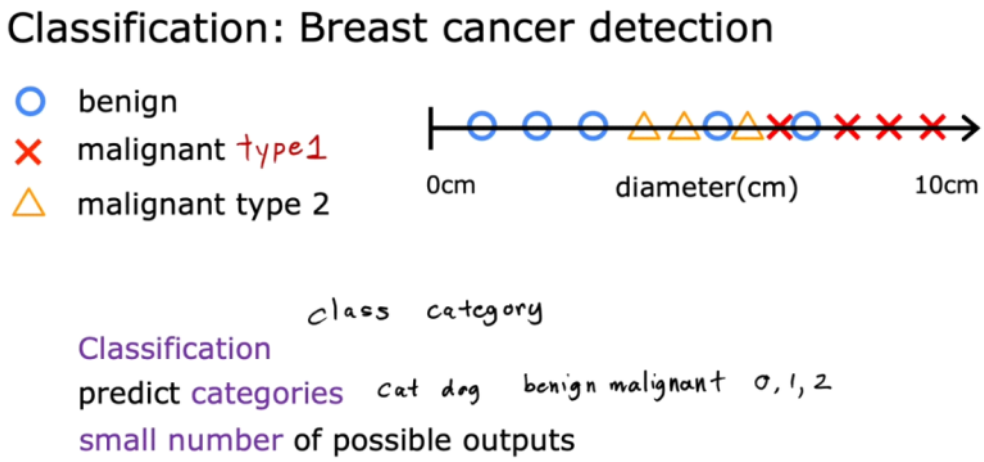
You can then plot your data on a graph like this where the horizontal axis represents the size of the tumor and the vertical axis takes on only two values 0 or 1 depending on whether the tumor is benign, 0 or malignant 1.
One reason that this is different from regression is that ***we're trying to predict only a small number of possible outputs or categories.***
In this case **two possible outputs 0 or 1**, benign or malignant. ***This is different from regression which tries to predict any number, all of the infinitely many number of possible numbers.***

And so the fact that there are only two possible outputs is what makes this classification.

Because there are only two possible outputs or two possible categories in this example, you can also plot this data set on a line like this.
And if new patients walks in for a diagnosis and they have a lump that is this size, then the question is, will your system classify this tumor as benign or malignant?

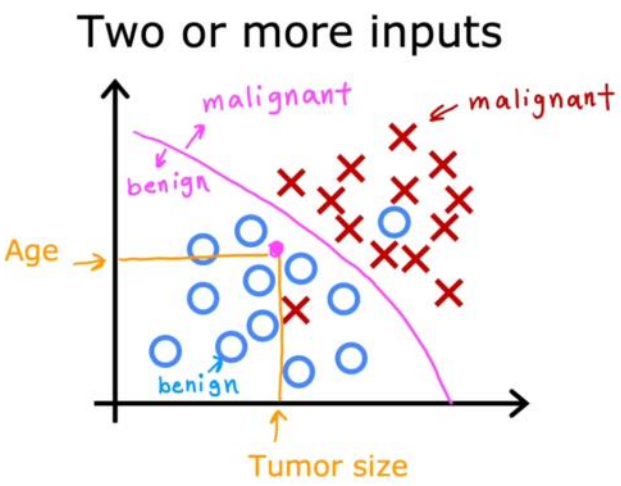


It turns out that in classification problems you can also have more than two possible output categories. Maybe you're learning algorithm can output multiple types of cancer diagnosis if it turns out to be malignant. So let's call two different types of cancer type 1 and type 2. In this case the average would have three possible output categories it could predict. And by the way in classification, the terms output classes and output categories are often used interchangeably.



Classification algorithms predict categories. Categories don't have to be numbers. It could be non numeric for example, it can predict whether a picture is that of a cat or a dog. And it can predict if a tumor is benign or malignant. Categories can also be numbers like 0, 1 or 0, 1, 2. But what makes classification different from regression when you're interpreting the numbers is that classification predicts a small finite limited set of possible output categories such as 0, 1 and 2 but not all possible numbers in between like 0.5 or 1.7.

In the example of supervised learning that we've been looking at, we had only one input value the size of the tumor. **But you can also use more than one input value to predict an output**. Here's an example:



Instead of just knowing the tumor size, say you also have each patient's age in years. Your new data set now has two inputs, age and tumor size. When a new patient comes in, the doctor can measure the patient's tumor size and also record the patient's age. And so given this, how can we predict if this patient's tumor is benign or malignant? Well, given the dataset like this, what the learning algorithm might do is find some boundary that separates out the malignant tumors from the benign ones. So the learning algorithm has to decide how to fit a boundary line through this data. The boundary line found by the learning algorithm would help the doctor with the diagnosis. In this case the tumor is more likely to be benign. From this example we have seen how to inputs the patient's age and tumor size can be used. In other machine learning problems often many more input values are required.

Supervised learning

Learns from being given "right answers"

Regression	Classification
Predict a number	predict categories
infinitely many possible outputs	small number of possible outputs

Supervised Learning Algorithms

21 April 2024 09:05

Various algorithms and computations techniques are used in supervised machine learning processes. Below are brief explanation s of some of the most commonly used learning methods, typically calculated through use of Python:

- **Neural networks:** Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer.
- **Naive bayes:** Naive Bayes is classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.
- **Linear regression:** Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.
- **Logistic regression:** While linear regression is leveraged when dependent variables are continuous, logistic regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no." While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.
- **Support vector machines (SVM):** A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression. That said, it is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.
- **K-nearest neighbor:** K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average. Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.
- **Random forest:** Random forest is another flexible supervised machine learning algorithm used for both classification and regression purposes. The "forest" references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

Unsupervised Learning

21 April 2024 11:51

Unsupervised learning uses machine learning (ML) algorithms to analyze and cluster **unlabeled data** sets. These algorithms discover hidden patterns or data groupings without the need for human intervention.
Unsupervised learning's ability to discover similarities and differences in information make it the ideal solution for explor atory data analysis, cross-selling strategies, customer segmentation and image recognition.

How does unsupervised learning work?

As the name suggests, unsupervised learning uses self-learning algorithms—they learn without any labels or prior training. Instead, the model is given raw, unlabeled data and has to infer its own rules and structure the information based on similarities, differences, and patt erns without explicit instructions on how to work with each piece of data.
Unsupervised learning algorithms are better suited for more complex processing tasks, such as organizing large datasets into clusters. They are useful for identifying previously undetected patterns in data and can help identify features useful for categorizing data.
Imagine that you have a large dataset about weather. An unsupervised learning algorithm will go through the data and identify patterns in the data points. For instance, it might group data by temperature or similar weather patterns.
While the algorithm itself does not understand these patterns based on any previous information you provided, you can then go through the data groupings and attempt to classify them based on your understanding of the dataset. For instance, you might recognize that the different temperature groups represent all four seasons or that the weather patterns are separated into different types of weather, such as rain, s leet, or snow.

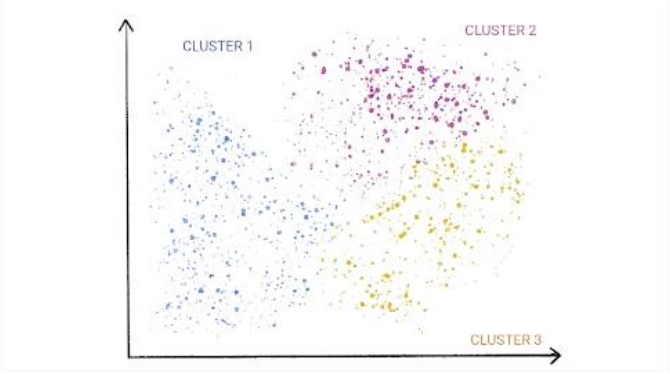


Figure 1. An ML model clustering similar data points.

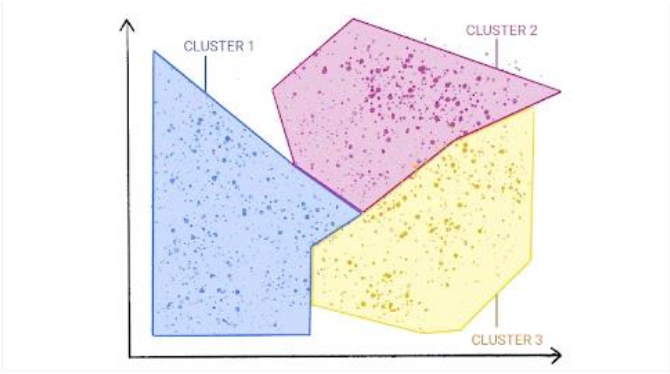
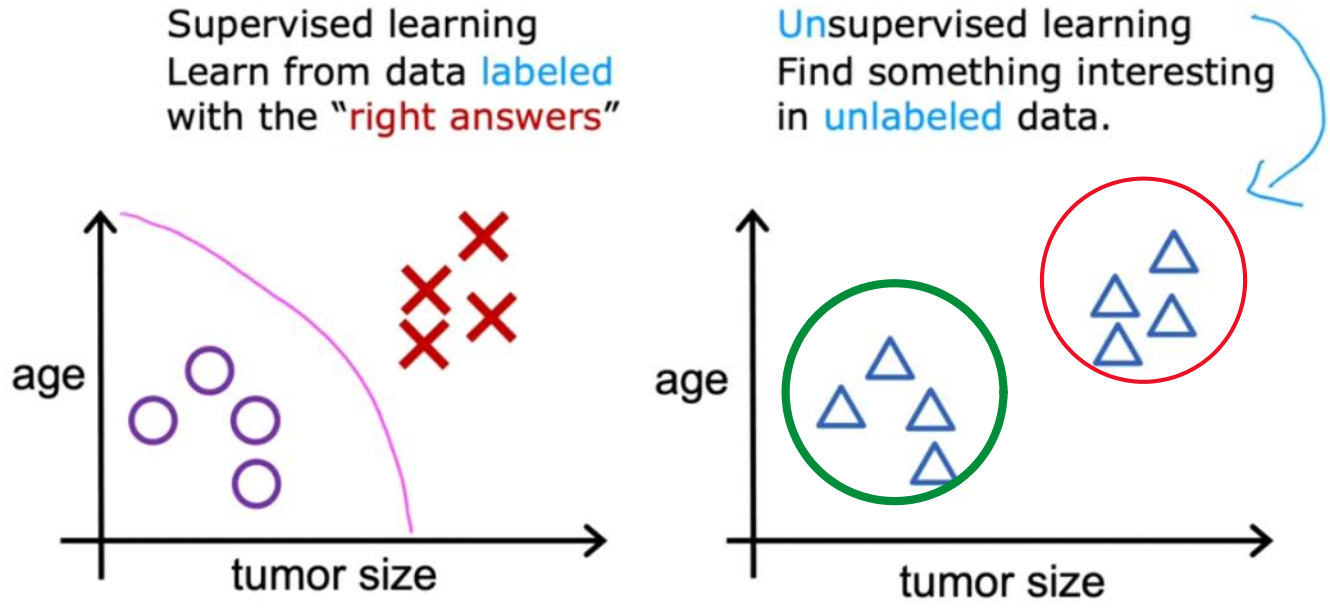


Figure 2. Groups of clusters with natural demarcations.

Imagine a classification problem in supervised learning. Each data point has a corresponding label, such as "benign" or "mali gnant" for cancer diagnosis.
In unsupervised learning, **the data has no labels.** For instance, you might have patient data with tumor size and age, but not whether the tumor is benign or malignant.



Unsupervised Learning Applications

- **Clustering:** Unsupervised learning algorithms like clustering can group unlabeled data into different clusters. This is used in various applications like:
    - **Google News:** Clustering news articles based on similar words, like grouping articles about pandas together.
    - **DNA Microarray Data:** Clustering individuals based on their genetic similarities.
    - **Market Segmentation:** Grouping customers into different segments for targeted marketing.

Formal Definition of Unsupervised Learning:

- In supervised learning, data has both inputs (x) and labels (y).
- In unsupervised learning, data only has inputs (x) and no labels (y).
- The algorithm must find patterns or structures within the data itself.

Other Unsupervised Learning Techniques:

- **Anomaly Detection:** This identifies unusual events, useful for fraud detection in finance and many other applications.
- **Dimensionality Reduction:** This compresses a large dataset into a smaller one while preserving information.

Supervised vs. Unsupervised Learning Examples:

- **Spam Filtering:** Classifying emails as spam or non-spam (supervised learning with labeled data).
- **News Story Clustering:** Grouping news articles based on content (unsupervised learning with clustering).
- **Market Segmentation:** Grouping customers automatically (unsupervised learning).
- **Diagnosing Diabetes:** Classifying patients as diabetic or non-diabetic (supervised learning similar to cancer diagnosis).

Regression Model

25 September 2022 23:45

Linear Regression just means fitting a straight line to your data. It's probably the most widely used learning algorithm in t he world today. As you get familiar with linear regression, many of the concepts you see here will also apply to other machine learning models that you' ll see later in this specialization. Let's start with a problem that you can
address using linear regression.
Say you want to predict the price of a house based on the size of the house. We're going to use a dataset on house sizes and prices from Portland, a city in the United States.

Here we have a graph where the horizontal axis is the size of the house in square feet, and the vertical axis is the price of a house in thousands of dollars. Here each data point is a house with the size and the price that it most recently was sold for.

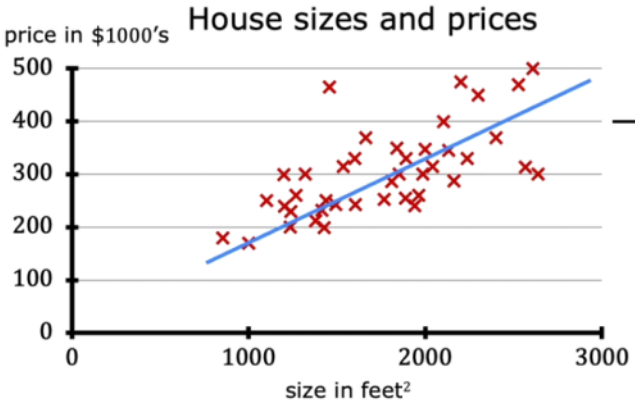
Now, let's say you're a real estate agent in Portland and you're helping a client to sell her house. She is asking you, how m uch do you think I can get for this house? This dataset might help you estimate the price she could get for it. You start by measuring the size of the house , and it turns out that the house is 1250 square feet. How much do you think this house could sell for? One thing you could do this, you can build a line ar regression model from this dataset.

Your model will fit a straight line to the data, which might look like this. Based on this straight line fit to the data, you can see that the house is 1250 square feet, it will intersect the best fit line over here, and if you trace that to the vertical axis on the left, you can s ee the price is maybe around here, say about $220,000. This is an example of what's called a supervised learning model.

We call this supervised learning because you are first training a model by giving a data that has right answers because you g et the model examples of houses with both the size of the house, as well as the price that the model should predict for each house. Well, here are the prices, that is, the right answers are given for every house in the dataset. This linear regression model is a particular type of supervised learning mo del.

It's called regression model because it predicts numbers as the output like prices in dollars. Any supervised learning model that predicts a number such as 220,000 or 1.5 or negative 33.2 is addressing what's called a regression problem. Linear regression is one example of a re gression model. But there are other models for addressing regression problems too.

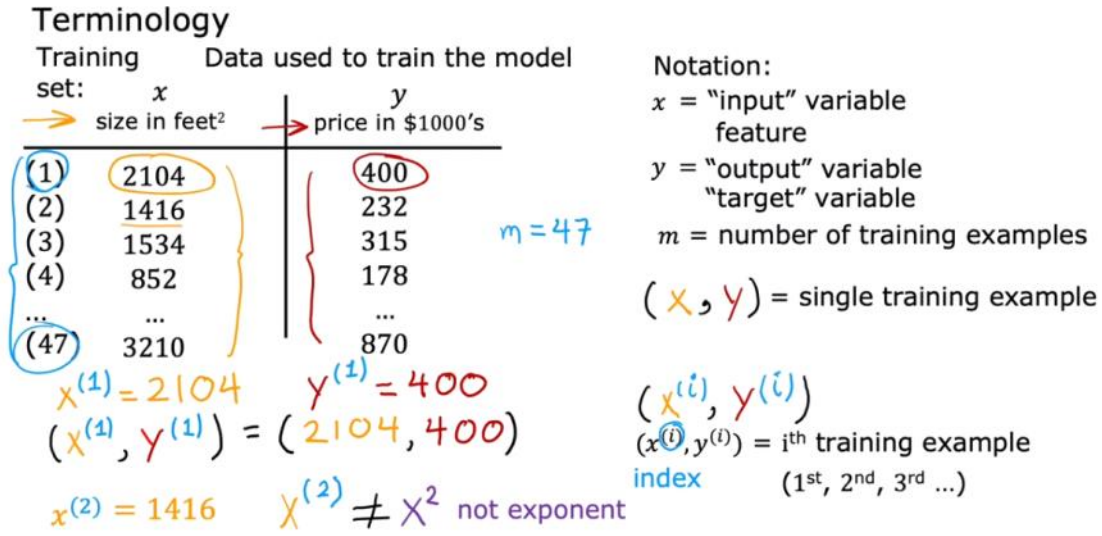
In addition to visualizing this data as a plot here on the left, there's one other way of looking at the data that would be u seful, and that's a data table here on the right.



Data table

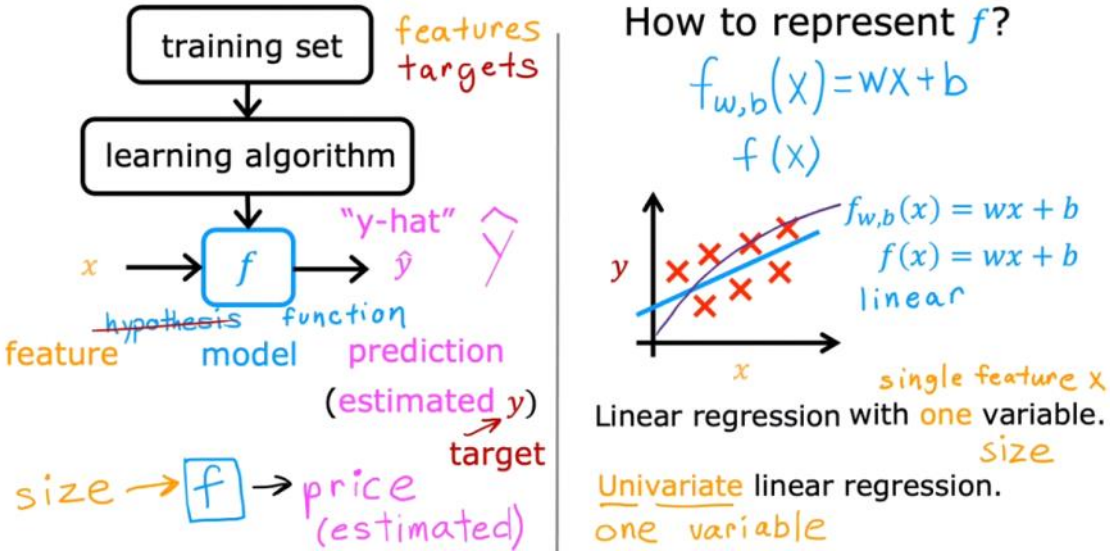
size in feet²	price in $1000's
2104	400
1416	232
1534	315
852	178
...	...
3210	870

The data comprises a set of inputs. Left column is the size of the house, or input. It also has outputs, which is the right c olumn showing prices in $1000s. You're trying to predict the price, which is this column here. Notice that the horizontal and vertical axes correspond to the se two columns, the size and the price. If you have, say, 47 rows in this data table, then there are 47 of these little crosses on the plot of the left, e ach cross corresponding to one row of the table.



Machine Learning Notation

- Training set: the dataset used to train the model (your client's house is not included).
- Input variable (feature): denoted by lowercase x (e.g., size of the house, x = 2104 square feet for the first house).
- Output variable (target variable): denoted by lowercase y (e.g., price of the house, y = $400,000 for the first house).
- Number of training examples: denoted by m (e.g., m = 47 for a dataset with 47 houses).
- Single training example: denoted by (x, y) (e.g., (2104, 400) for the first house).
- Specific training example (i-th example): denoted by x^(i), y^(i) (e.g., x^(1) = 2104, y^(1) = 400 for the first house, i = 1).
    - The superscript (i) is an index, not exponentiation (x^i is not x to the power of i). It refers to the row number (i) in the table.



Training the Model

1. The training set is fed to the learning algorithm.
2. The algorithm produces a function (f) that estimates the output target (y-hat) for a new input (x).

- f(x) is the model, also called the hypothesis.
- x is the input feature.
- y-hat is the predicted value of the output target (y).

Model Predictions

- y-hat is an estimate and may not be the actual true value (y).
- The model's goal is to minimize the difference between y-hat and the actual y value.

Example: Predicting House Prices

- In the house price prediction example, the model (f) takes the size of the house (x) as input and outputs an estimated price (y-hat).

Linear Regression Model

- This video focuses on a linear regression model, which is a supervised learning model that represents the function (f) as a straight line.
- f(x) = w * x + b, where w and b are numbers that determine the slope and y-intercept of the line.

Visualizing the Model

- The training set is plotted on a graph with the input feature (x) on the horizontal axis and the output target (y) on the vertical axis.
- The best-fit line is the linear function f(x) = w * x + b.

Why a Linear Function?

- Linear functions (straight lines) are relatively simple and easy to work with, making them a good foundation for understanding more complex models.

Univariate Linear Regression

- This specific model is called linear regression with one variable because it uses a single input variable (x), the size of the house.
- Another name for this model is univariate linear regression (uni = one, variate = variable).

Cost Function

26 September 2022 19:34

When working with linear regression, we aim to find the best line that fits the training data. ***The cost function measures the difference between the predicted values of the model and the actual target values***. By minimizing this cost function, we can determine the optimal values for the model's parameters and improve its performance.

Training set

features size in feet² (x)	targets price \$1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

Model: $f_{w,b}(x) = wx + b$

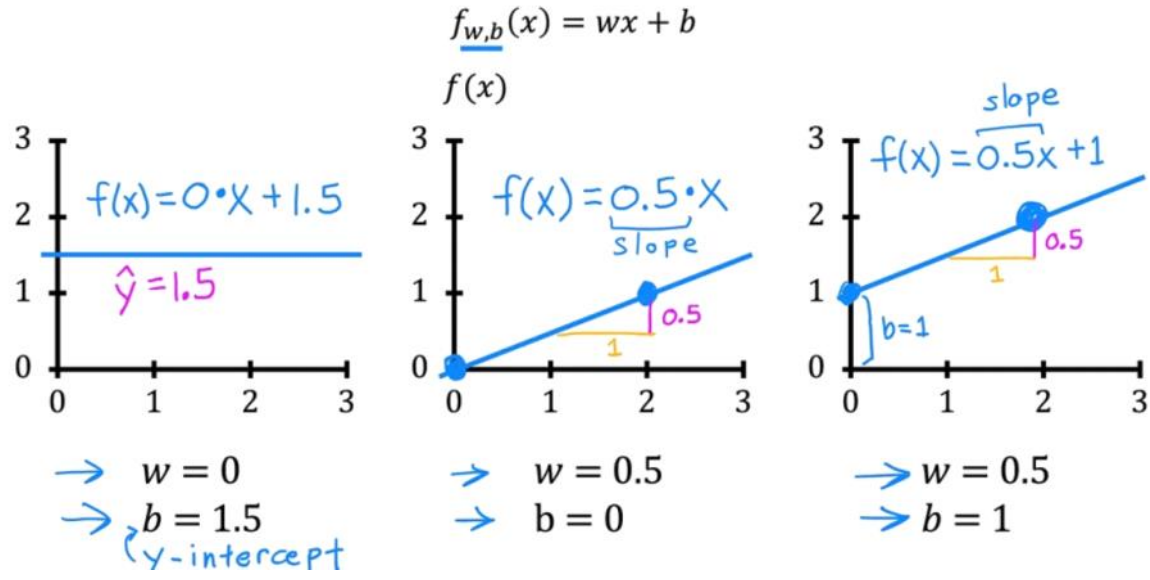
w, b: parameters
coefficients
weights

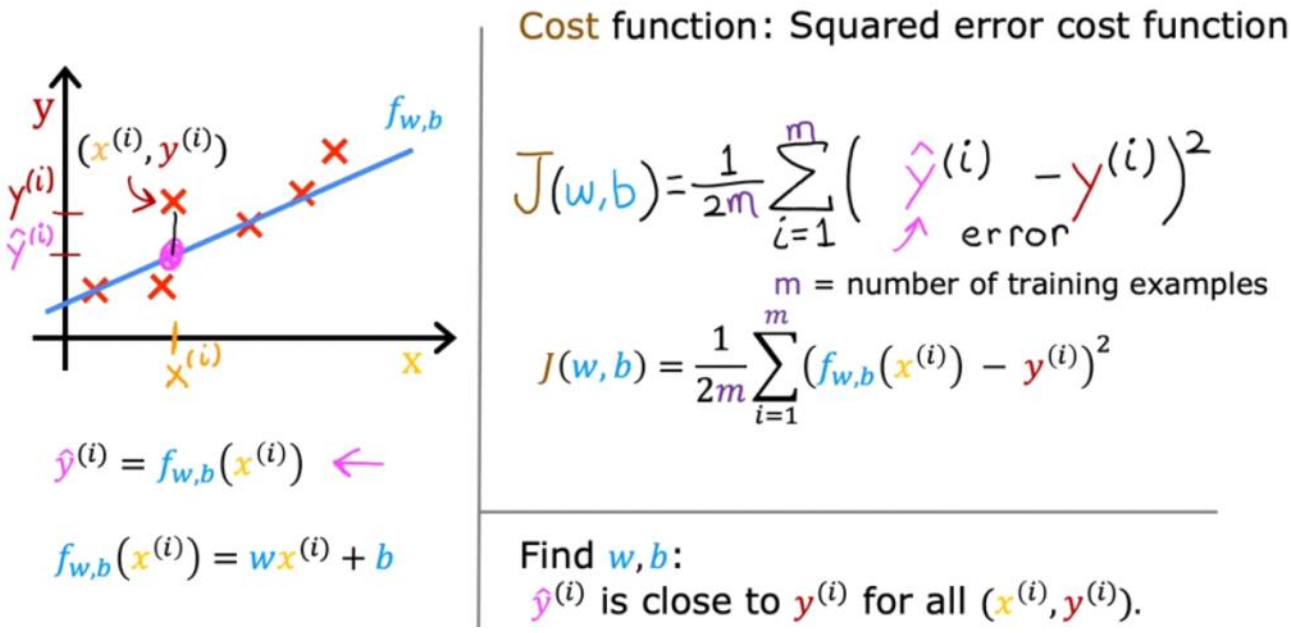
The Linear Regression Model

Before delving into the cost function, let's briefly revisit the linear regression model. It represents a line that fits the training data using parameters called coefficients or weights. The model's equation is :

$$f(w, b, x) = w * x + b$$

where: w and b are the adjustable parameters
The goal is to choose the best values for w and b so that the line generated by the model closely matches the training data.





The line wx+b contains predicted values of y. In order to find a line that fits the data points, we use something called a cost function.

Cost function ***measures the performance of a machine learning model for given data****. Cost function quantifies the error between predicted and expected values and present that error in the form of a single real number.*

The equation for cost with one variable is given by:

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}\left[f_{w,b}(x^{(i)}) - y^{(i)}\right]^2$$

where:

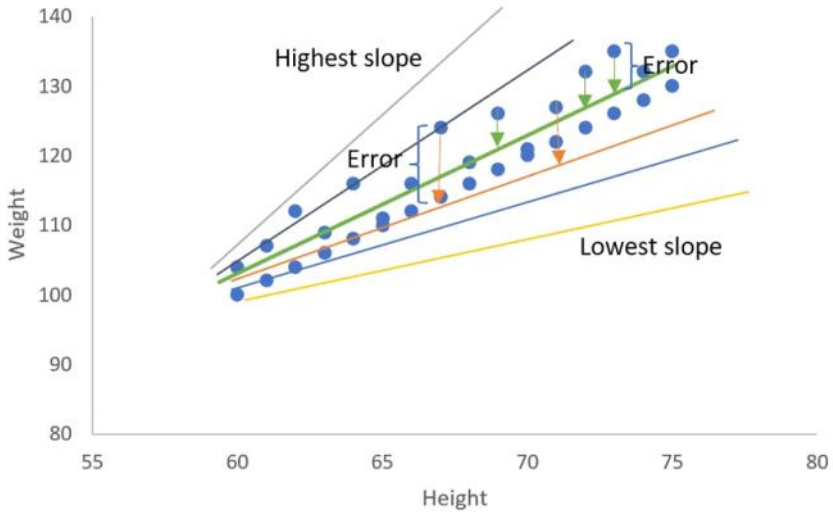
$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

In the equations:

- ***f_{w,b}(x^{(i)})*** *represents our prediction for the example i using the parameters w and b.*
- ***\frac{1}{2m} \sum\limits_{i=0}^{m-1} \left[f_{w,b}(x^{(i)}) — y^{(i)} \right]²*** *denotes the squared difference between the target value and the prediction.*
- *These differences are summed over all the m examples and divided by 2m to calculate the cost J(w,b).*

Intuition Behind the Cost Function

The cost function, denoted as J(w, b), measures how well the model's predictions align with the true target values. It calculates the squared error between the predicted value (f(w, b, x)) and the actual target value (y). The cost function is defined as the sum of squared errors across all training examples, and its value indicates how well the model fits the data.



Error: Difference between actual and predicted. (Image by Author)

Suppose the actual weight and predicted weights are as follows:

Actual (Y)	Predicted (Y')
100	96
102	97
104	105
106	109
108	113
110	115

Actual vs Predicted Weights

We can calculate the MSE as follows:

$$= \frac{1}{6}\left((100-96)^2 + (102-97)^2 + (104-105)^2 + (106-109)^2 + (108-113)^2 + (110-115)^2\right)$$

$$= \frac{1}{6}(16 + 25 + 1 + 9 + 25 + 25)$$

$$= \frac{1}{6}(101) = 16.8$$

Role of the Cost Function

The primary objective in training a linear regression model is to minimize the cost function. By finding the values of w and b that result in a small cost function, we achieve a model that accurately predicts the target values. Minimizing the cost function involves adjusting the parameters iteratively until convergence, using techniques such as **gradient descent**.

In order to implement linear regression the first key step is first to define something called a cost function. The cost function will tell us how well the model is doing so that we can try to get it to do better.

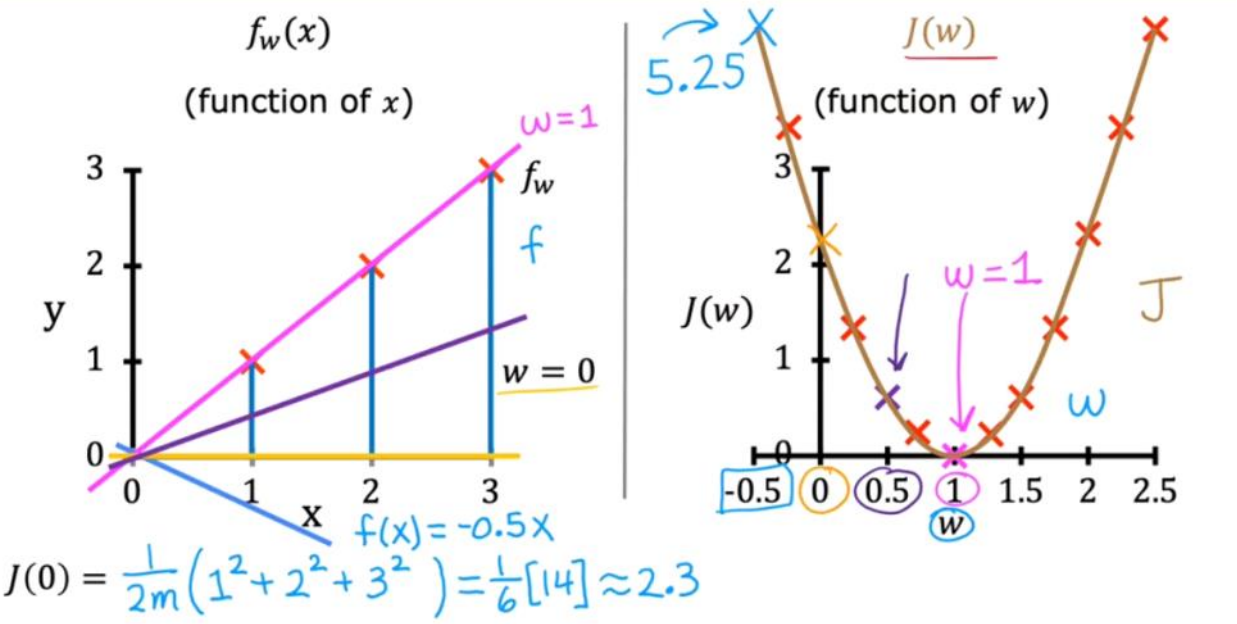
Visualizing the Cost Function

To gain a better understanding, let's visualize how the cost function changes with different values of w. We simplify the model by setting b to 0, resulting in the function f(x) = w * x. The cost function J(w) depends only on w and measures the squared difference between f(x) and y for each training example.

Example: Exploring the Cost Function

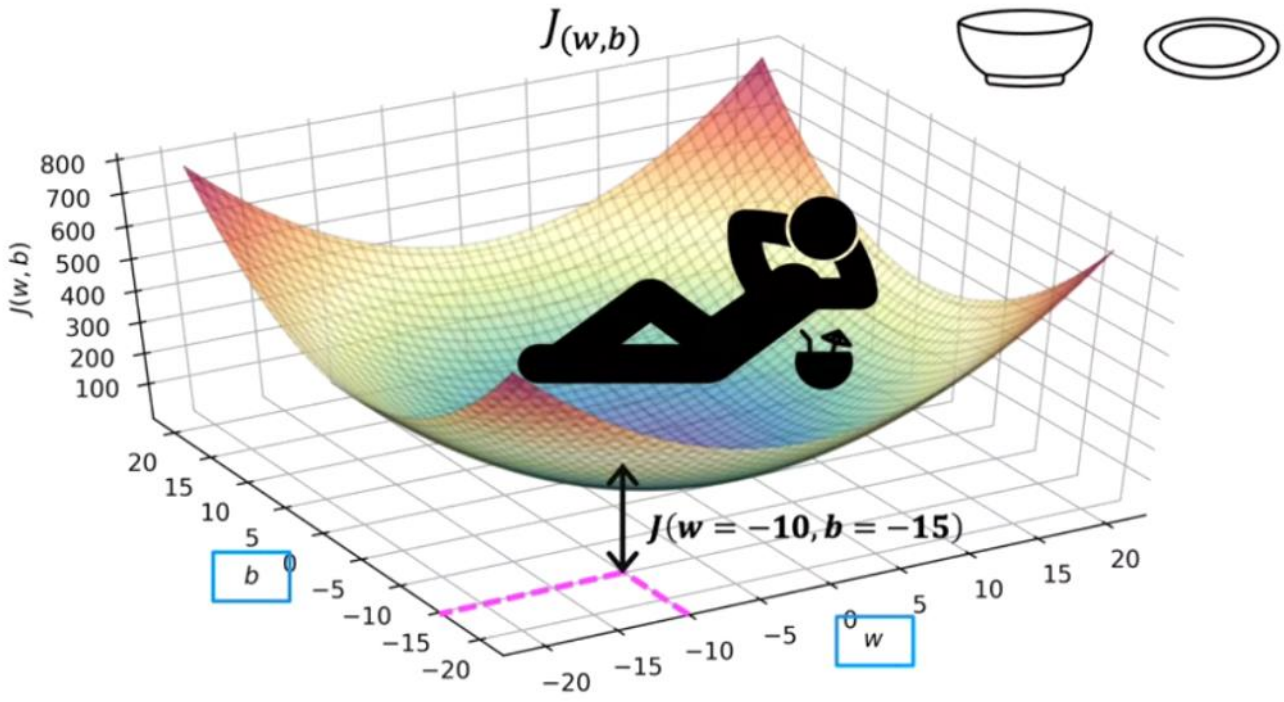
Suppose we have a training set with three points (1, 1), (2, 2), and (3, 3). We plot the function f(x) = w * x for different values of w and calculate the corresponding cost function J(w).

Let's look at graphs of the model f of x, and the cost function J:



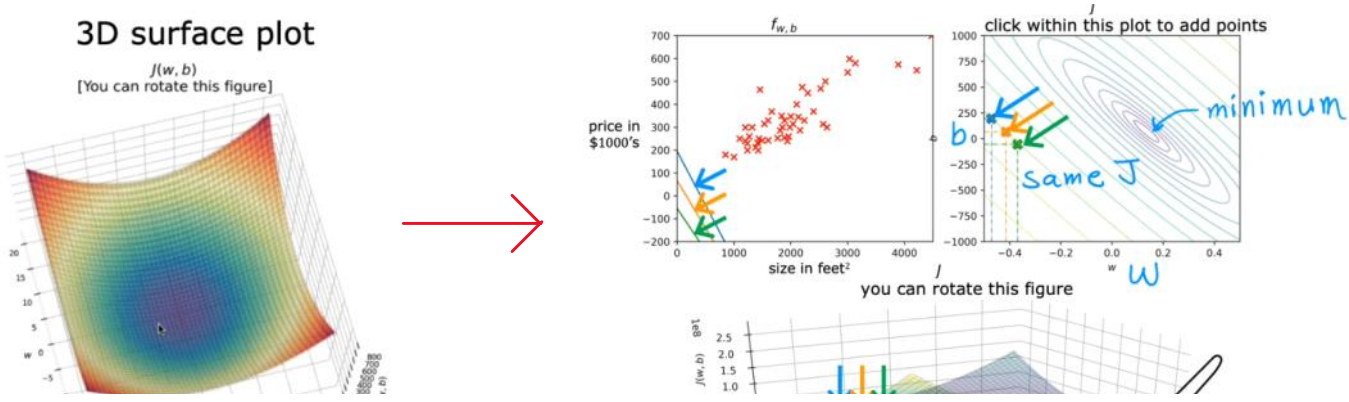
In this example we are assuming the right value of w=1. Therefore the value of J(w) (cost function) is minimum at w=1.

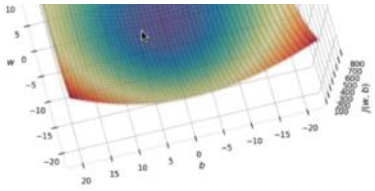
This was the case for a single parameter w. The graph of J(w,b) would be a lot more complex:

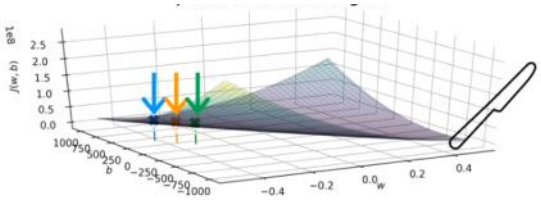


3D SURFACE PLOT OF COST FUNCTION FOR TWO PARAMETERS W AND B

There is another way to plot the cost function with two parameters, using a **contour plot:**







TOP VIEW OF A SURFACE PLOT

- The two axes on this contour plots are b, on the vertical axis, and w on the horizontal axis. What each of these ovals, also called ellipses, shows, is the center points on the 3D surface which are at the exact same height. In other words, the set of points which have the same value for the cost function J.
- To get the contour plots, **you take the 3D surface at the bottom and you use a knife to slice it horizontally.** You take horizontal slices of that 3D surface and get all the points, they're at the same height. Therefore, each horizontal slice ends up being shown as one of th ese ellipses or one of these ovals.
- Concretely, if you take all of these three points have the same value for the cost function J, even though they have differen t values for w and b.
- In the figure on the upper left, you see also that these three points correspond to different functions, f, all three of whic h are actually pretty bad for predicting housing prices in this case.
- The centre of the ellipses is where the best set of parameters lie.

Visualizing Cost Function

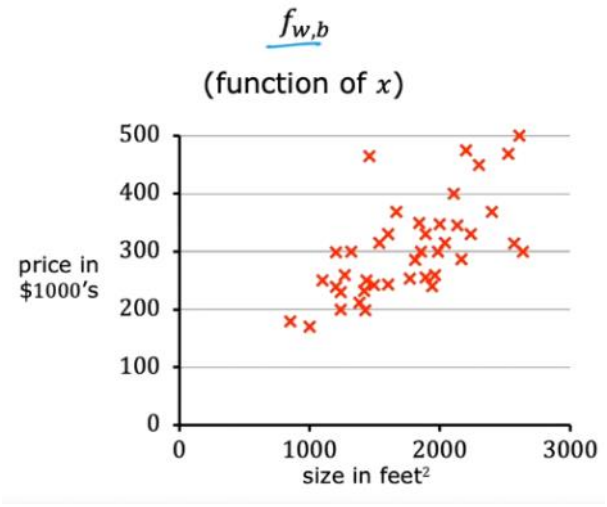
28 September 2022 14:56

Here is what we've seen so far.

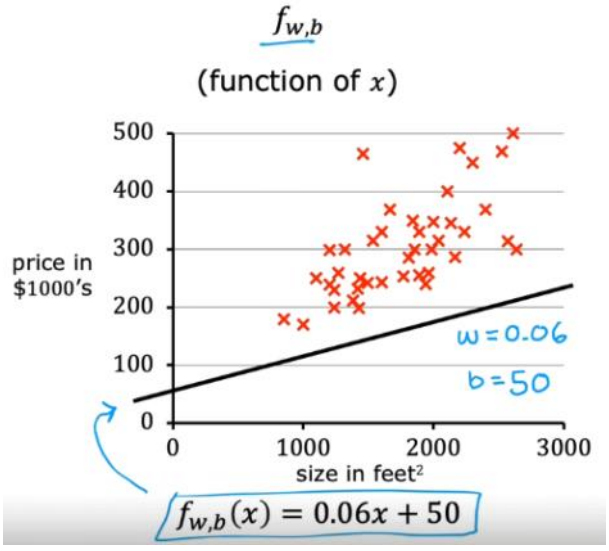
Model	$f_{w,b}(x) = wx + b$
Parameters	w, b
Cost Function	$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$
Objective	$\underset{w,b}{\text{minimize}}\, J(w,b)$

Before, we had temporarily set b to zero in order to simplify the visualizations. Now, let's go back to the original model with both parameters w and b without setting b to be equal to 0.

Here's a training set of house sizes and prices.



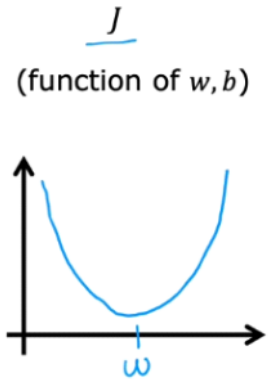
Let's say you pick one possible function of x, like this one.



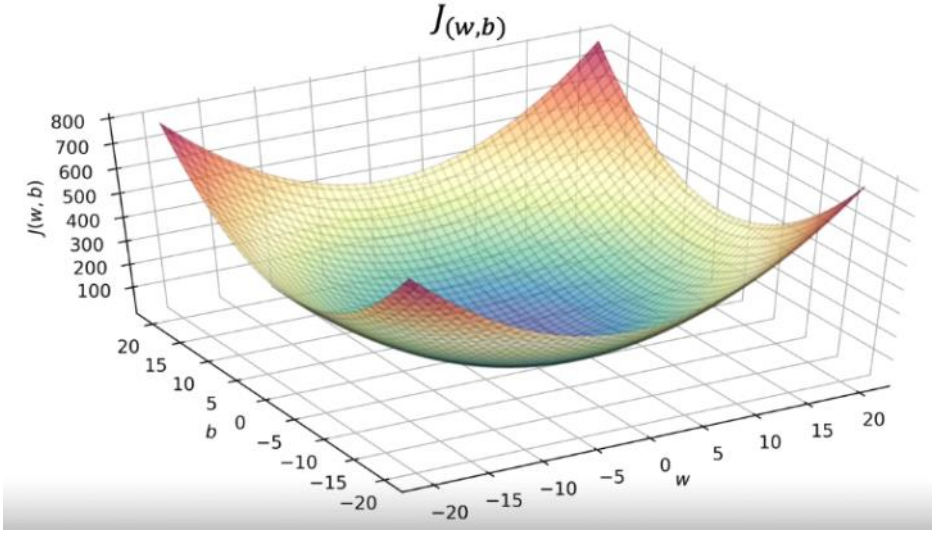
Note that this is not a particularly good model for this training set, is actually a pretty bad model. It seems to consistently underestimate housing prices.

Given these values for w and b let's look at what the cost function J of w and b may look like.

Recall what we saw last time was when you had only w, because we temporarily set b to zero to simplify things, but then we had come up with a plot of the cost function that look like this as a function of w only. When we had only one parameter, w, the cost function had this U-shaped (parabolic) curve, shaped a bit like a soup bowl.



Now, in this housing price example that we have on this slide, we have two parameters, w and b. The plots becomes a little more complex.



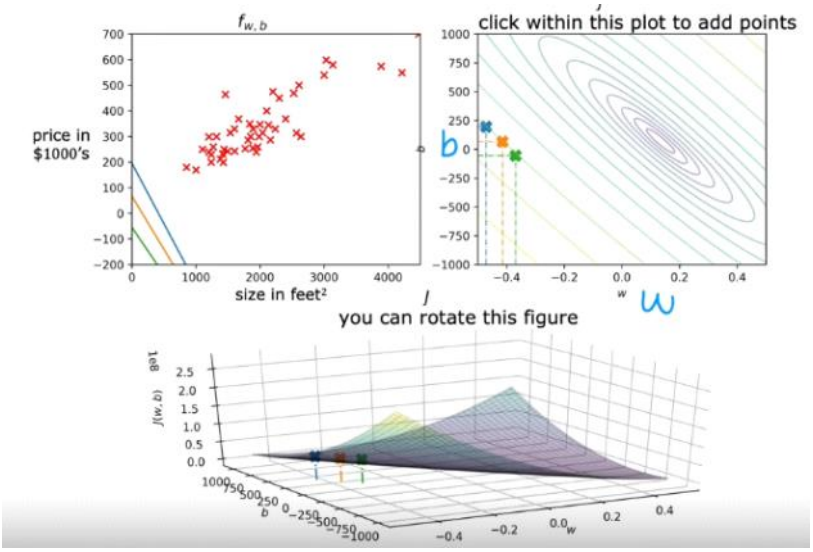
It turns out that the cost function also has a similar shape like a soup bowl, except in three dimensions instead of two.
This is a 3D surface plot.

As you vary w and b, which are the two parameters of the model, you get different values for the cost function J of w, and b. This is a lot like the U-shaped curve
you saw in the last video, except instead of having one parameter w as input for the j, you now have two parameters, w and b as inputs into this soup bowl or this hammock-shaped function J.

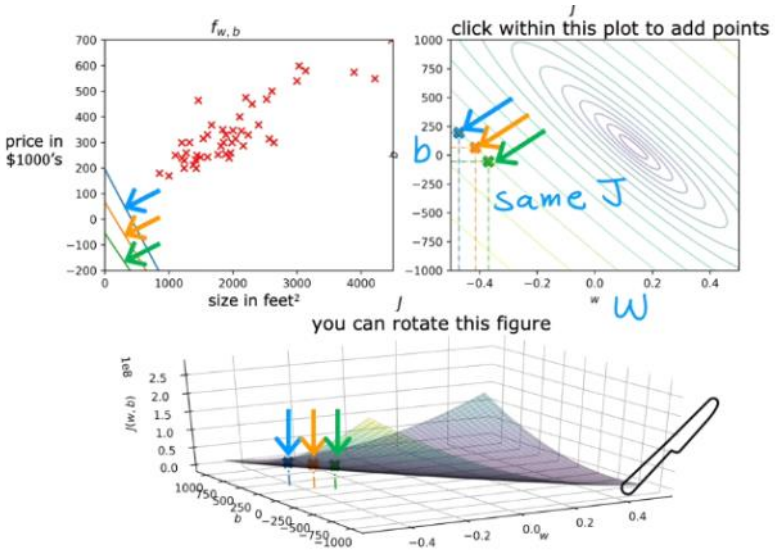
Any single point on this surface represents some particular choice of w and b.

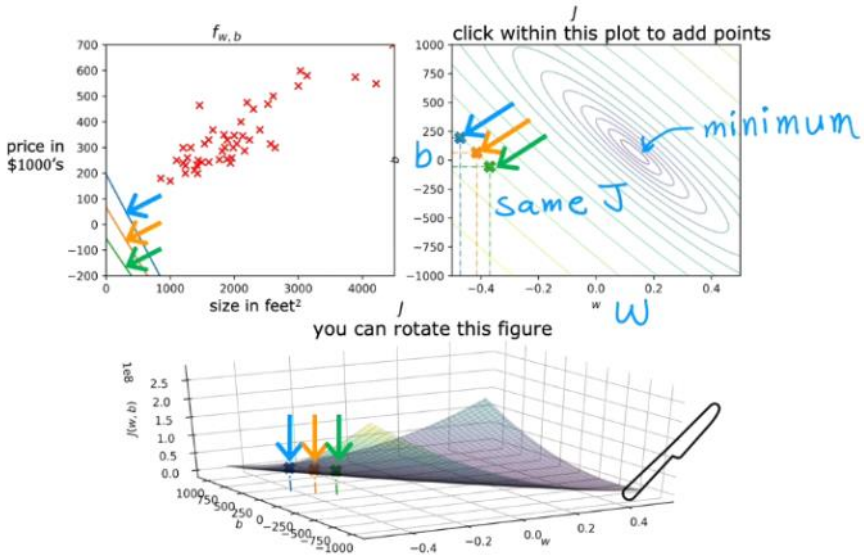
For example, if w was minus 10 and b was minus 15, then the height of the surface above this point is the value of J when w is minus 10 and b is minus 15.
Now, in order to look even more closely at specific points, there's another way of plotting the cost function J that would be useful for visualization, which is, rather than using these 3D-surface plots, we take this exact same function J and plot it using something called a **contour plot.**

Contour Plot: A contour plot is **a graphical technique for representing a 3-dimensional surface by plotting constant z slices, called contours, on a 2-dimensional format**. That is, given a value for z, lines are drawn for connecting the (x,y) coordinates where that z value occurs.

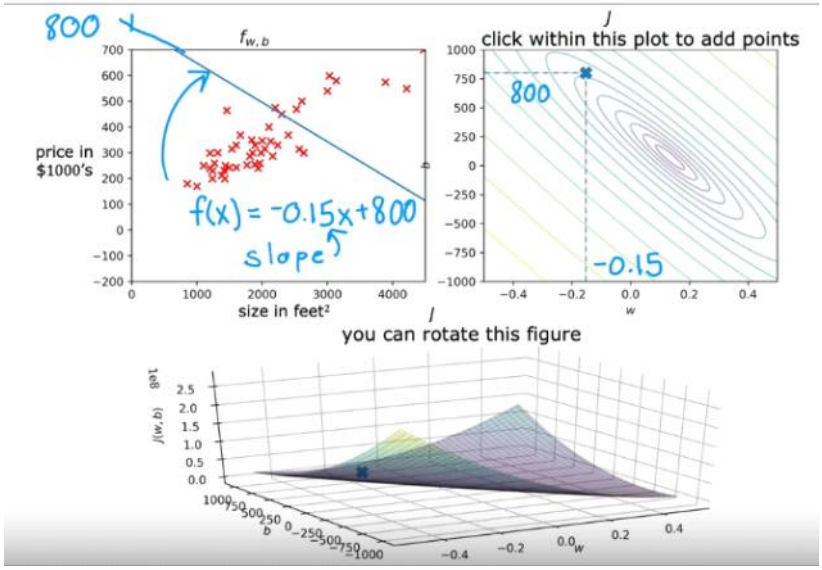


The two axes on this contour plots are b, on the vertical axis, and w on the horizontal axis. What each of these ovals, also called ellipses, shows, is the center points on the 3D surface which are at the exact same height. In other words, the set of points which have the same value for the cost function J.



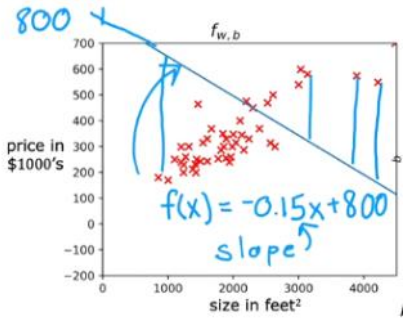


Let's look at some more visualizations of w and b. Here's one example.

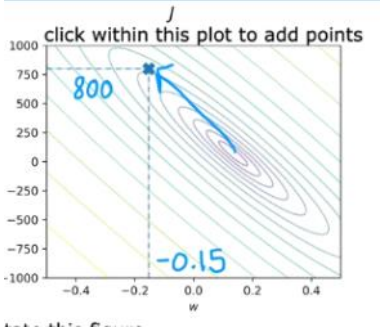


For this point, w equals about negative 0.15 and b equals about 800. This point corresponds to one pair of values for w and b that use a particular cost j. In fact, this booklet pair of values for w and b corresponds to this function f of x, which is this line you can see on the left. This line intersects the vertical axis at 800 because b equals 800 and the slope of the line is negative 0.15, because w equals negative 0.15.

Now, if you look at the data points in the training set, you may notice that this line is not a good fit to the data. For this function f of x, with these values of w and b, many of the predictions for the value of y are quite far from the actual target value of y that is in the training data.



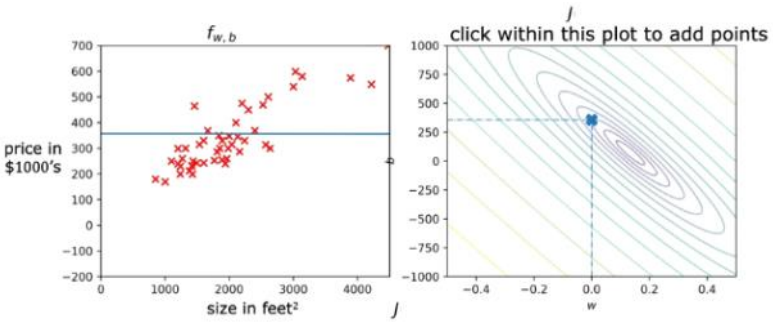
For this function f of x, with these values of w and b, many of the predictions for the value of y are quite far from the actual target value of y that is in the training data.



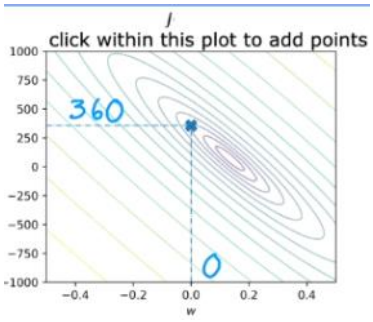
Because this line is not a good fit, if you look at the graph of j, the cost of this line is out here, which is pretty far from the minimum. There's a pretty high cost because this choice of w and b is just not that good a fit to the training set.

Now, let's look at another example with a different choice of w and b.

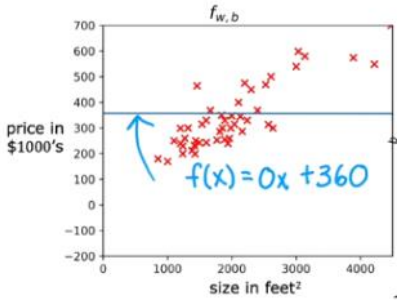
Now, here's another function that is still not a great fit for the data, but maybe slightly less bad.



This points here represents the cost for this booklet pair of w and b that creates that line.

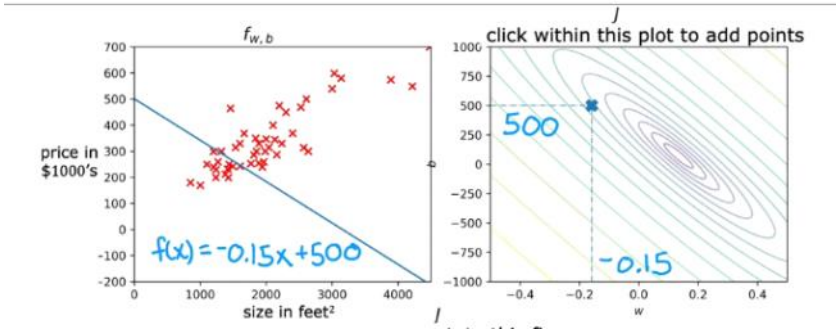


The value of w is equal to 0 and the value b is about 360.



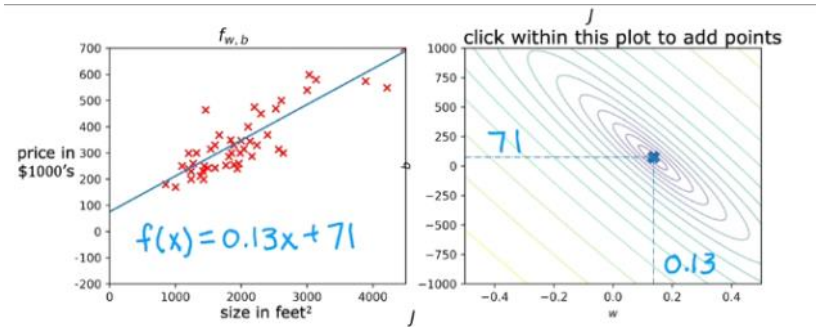
This pair of parameters corresponds to this function, which is a flat line, because f of x equals 0 times x plus 360.

Let's look at yet another example.



Here's one more choice for w and b, and with these values, you end up with this line f of x. Again, not a great fit to the data, is actually further away from the minimum
compared to the previous example. Remember that the minimum is at the center of that smallest ellipse.

Last example, if you look at f of x on the left, this looks like a pretty good fit to the training set.

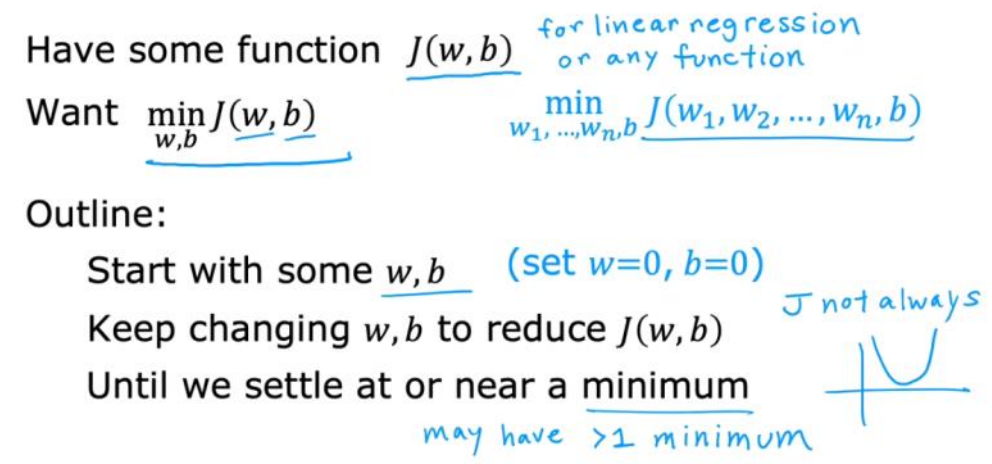


You can see on the right, this point representing the cost is very close to the center of the smaller ellipse, it's not quite exactly the minimum,
but it's pretty close. For this value of w and b, you get to this line, f of x.
You can see that if you measure the vertical distances between the data points and the predicted values on the straight line, you'd get the error for each data point.
The sum of squared errors for all of these data points is pretty close to the minimum possible sum of squared errors among all possible straight line fits.

Gradient Descent

28 September 2022 23:06

It would be nice if we had a more systematic way to find the values of w and b, that results in the smallest possible cost, j of w, b. It turns out there's an algorithm called gradient descent that you can use to do that.
Gradient descent is used all over the place in machine learning, not just for linear regression, but for training for example some of the most advanced neural network models, also called deep learning models.



You have the cost function j of w, b right here that you want to minimize.
In the example we've seen so far, this is a cost function for linear regression, ***but it turns out that gradient descent is an algorithm that you can use to try to minimize any function, not just a cost function for linear regression.***

Just to make this discussion on gradient descent more general, it turns out that gradient descent applies to more general functions, including other cost functions that work with models that have more than two parameters.
For instance, if you have a cost function J as a function of w_1, w_2 up to w_n and b, your objective is to minimize j over the parameters w_1 to w_n and b.

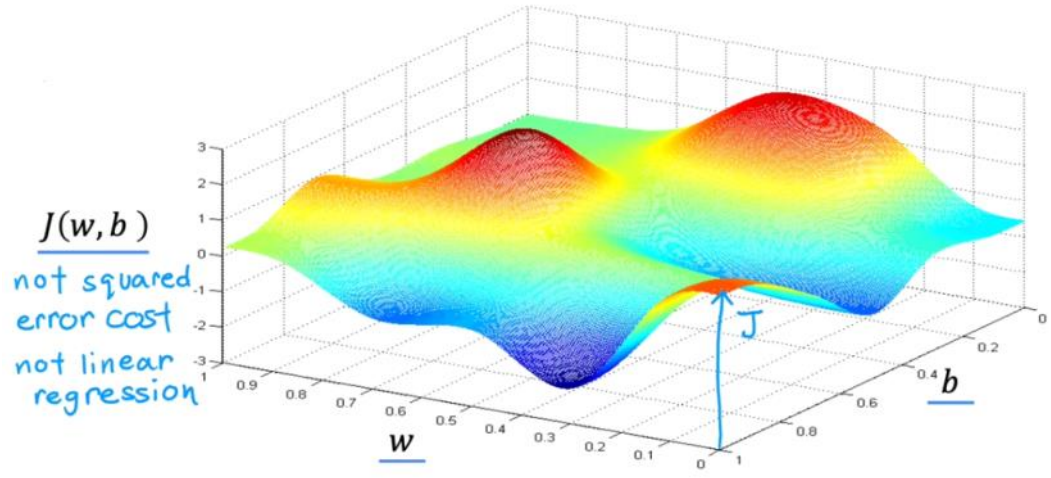
In other words, you want to pick values for w_1 through w_n and b, that gives you the smallest possible value of j.

It turns out that gradient descent is an algorithm that you can apply to try to minimize this cost function j as well.

What you're going to do is just to start off with some initial guesses for w and b. In linear regression, it won't matter too much what the initial value are, so a common choice is to set them both to 0.

For example, you can set w to 0 and b to 0 as the initial guess. With the gradient descent algorithm, what you're going to do is, you'll keep on changing the parameters w and b a bit every time to try to reduce the cost j of w, b until hopefully j settles at or near a minimum.

Let's take a look at an example of a more complex surface plot j to see what gradient is doing.

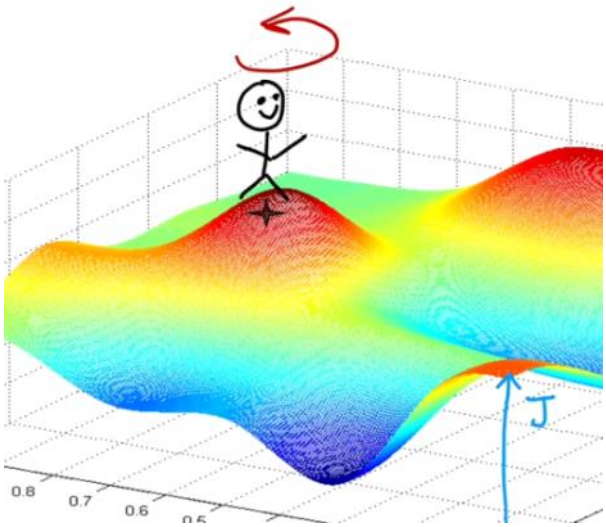


This function is not a squared error cost function. For linear regression with the squared error cost function, you always end up with a bow shape or a hammock shape. But this is a type of cost function you might get if you're training a neural network model. Notice the axes, that is w and b on the bottom axis. For different values of w and b, you get different points on this surface, j of w, b, where the height of the surface at some point is the

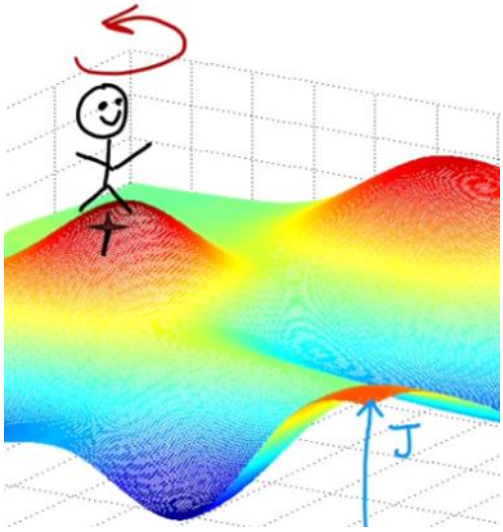
value of the cost function.

I'd like you to imagine if you will, that you are physically standing at this point on the hill. If it helps you to relax, imagine that there's lots of really nice green grass and butterflies and flowers is a really nice hill.
Your goal is to start up here and get to the bottom of one of these valleys as efficiently as possible.

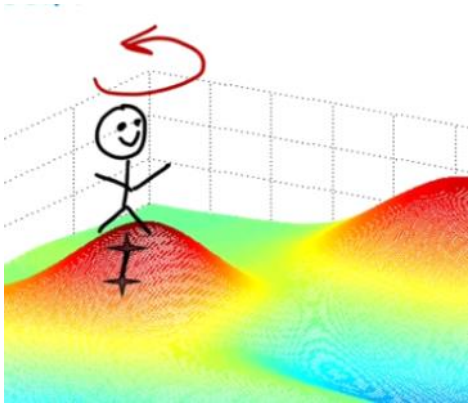
What the gradient descent algorithm does is, you're going to spin around 360 degrees and look around and ask yourself, if I were to take a tiny little baby step in one direction, and I want to go downhill as quickly as possible to or one of these valleys. What direction do I choose to take that baby step?



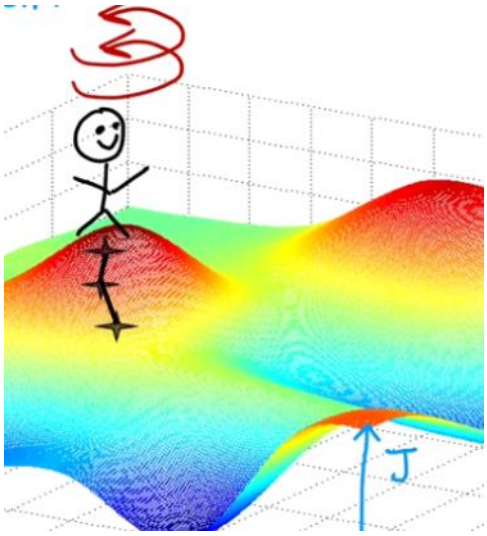
Well, if you want to walk down this hill as efficiently as possible, it turns out that if you're standing at this point in the hill and you look around, you will notice that the best direction to take your next step downhill is roughly that direction. Mathematically, this is the direction of steepest descent.



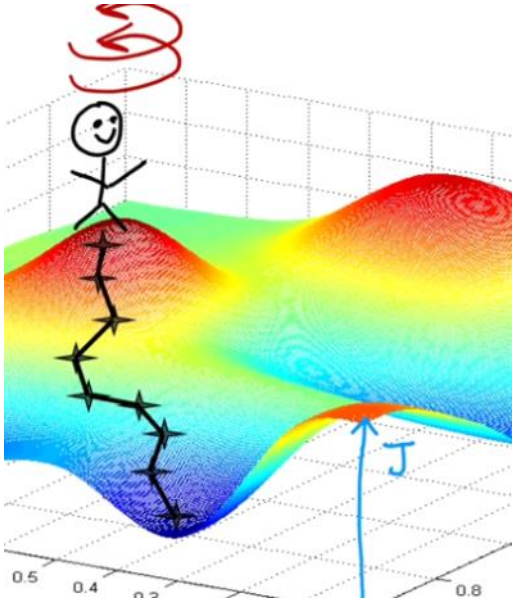
It means that when you take a tiny baby little step, this takes you downhill faster than a tiny little baby step you could have taken in any other direction.
After taking this first step, you're now at this point on the hill over here.



Now let's repeat the process.
Standing at this new point, you're going to again spin around 360 degrees and ask yourself, in what direction will I take the next little baby step in order to move downhill? If you do that and take another step, you end up moving a bit in that direction and you can keep going.

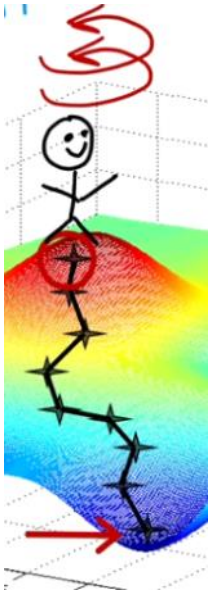


From this new point, you can again look around and decide what direction would take you downhill most quickly.
Take another step, another step, and so on, until you find yourself at the bottom of this valley, at this local minimum, right here.

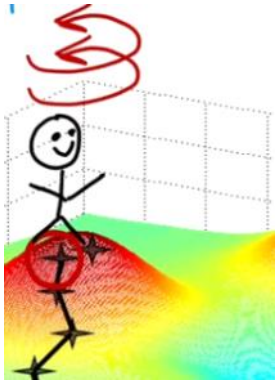


What you just did was go through multiple steps of gradient descent. It turns out, gradient descent has an interesting property.
Remember that you can choose a starting point at the surface by choosing starting values for the parameters w and b.

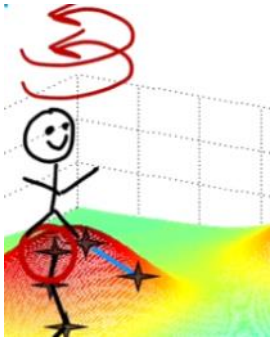
When you perform gradient descent a moment ago, you had started at this point over here.



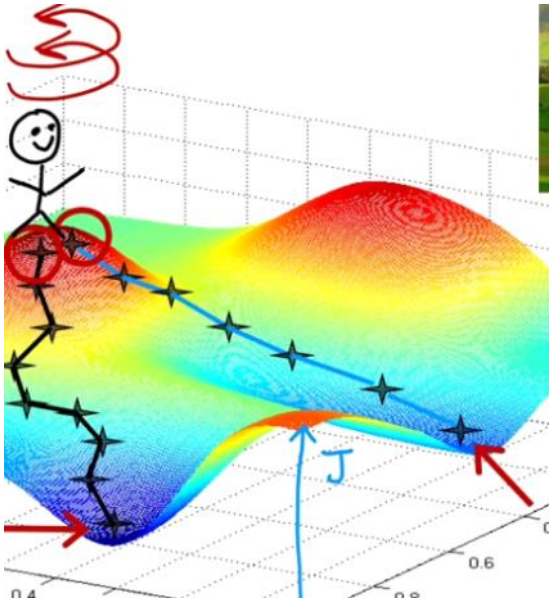
Now, imagine if you try gradient descent again, but this time you choose a different starting point by choosing parameters that place your starting point just a couple of steps to the right over here.



If you then repeat the gradient descent process, which means you look around, take a little step in the direction of steepest ascent so you end up here.



Then you again look around, take another step, and so on. If you were to run gradient descent this second time, starting just a couple steps in the right of where we did it the first time, then you end up in a totally different valley.



This different minimum over here on the right. The bottoms of both the first and the second valleys are called local minima.
Because if you start going down the first valley, gradient descent won't lead you to the second valley, and the same is true if you started going down the second valley, you stay in that second minimum and not find your way into the first local minimum.

What is Gradient Descent?

- Gradient descent is an iterative algorithm used to minimize a cost function (J(w, b)).
- It's applicable to various machine learning models, including linear regression and neural networks.
- The goal is to find the values for parameters (w and b) that minimize the cost function.

Generalizing Gradient Descent

- While this video focuses on linear regression, gradient descent works with cost functions beyond linear regression.
- It can handle functions with more than two parameters (w_1, w_2, ..., w_n, and b).

Steps of Gradient Descent

1. **Initialize parameters (w and b):** Choose initial guess values for w and b (e.g., both set to 0).
2. **Iterate:**
    - Update parameters w and b in each iteration to decrease the cost function (J(w, b)).
    - Continue iterating until the cost function reaches a minimum (or near a minimum).

Challenges with Cost Functions

- Some cost functions (J) may have multiple minima (valleys) on the surface plot.
- Gradient descent might converge to a local minimum instead of the global minimum (the absolute lowest point).

Visualizing Gradient Descent

Imagine a hilly landscape where the height represents the cost function value (J(w, b)). Valleys represent minima.

1. **Starting point:** You start at a point on the hill (initial guess for w and b).
2. **Steepest descent:** You take a tiny step downhill in the direction of steepest descent (negative gradient).
3. **Repeat:** You keep taking small steps downhill, eventually reaching a local minimum (bottom of a valley).

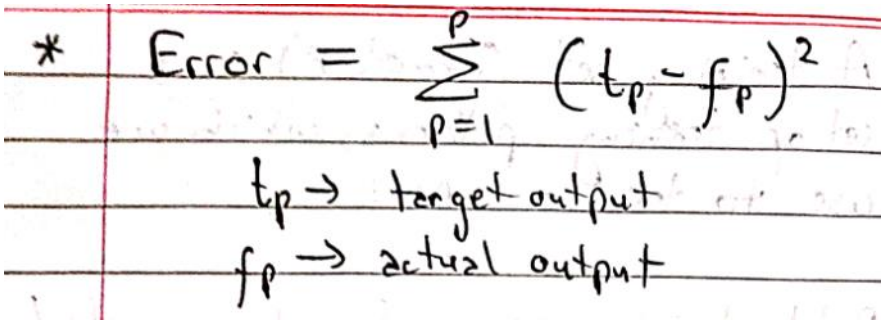
Impact of Starting Point

The starting point for w and b can influence the final minimum reached by gradient descent. Different starting points may lead to different local minima.

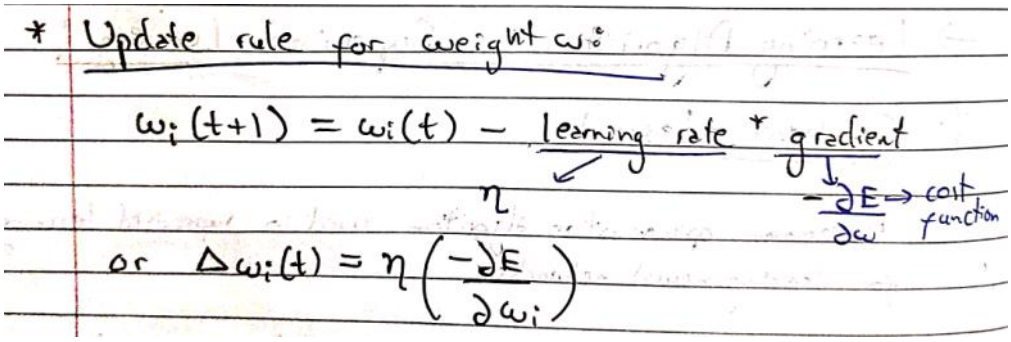
Local Minima vs. Global Minima

- Local minimum: A valley where the cost function is lower than its immediate surroundings.
- Global minimum: The absolute lowest point on the cost function surface.
- Gradient descent may not always find the global minimum, but it can effectively reach a local minimum.

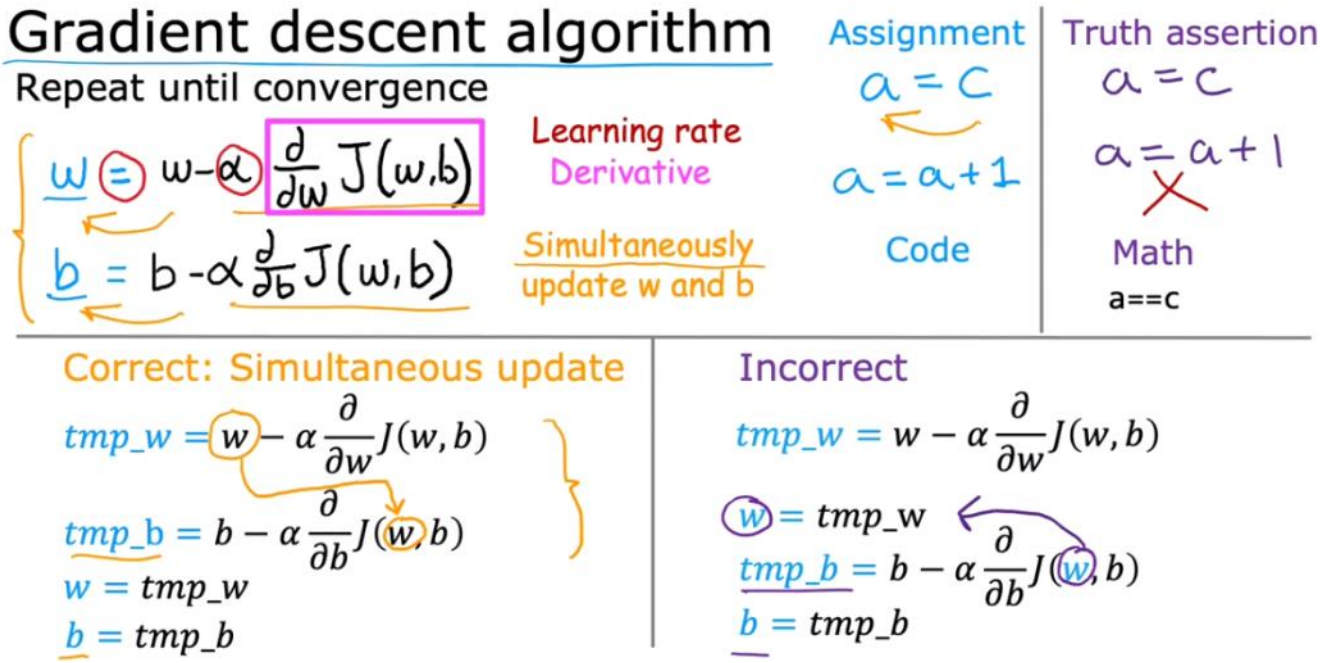
IMPLEMENTING GRADIENT DESCENT:



Cost Function



GRADIENT DESCENT



Alpha --> Learning rate, it controls how big of a step you wish to take down to determine minimum value of J

Gradient Descent Equation

The equation for gradient descent updates the parameters (w and b) of the model iteratively.

- w = w - α * ∂J(w, b) / ∂w (update for parameter w)
- b = b - α * ∂J(w, b) / ∂b (update for parameter b)

Explanation of the Equation

- w: weight parameter of the model
- b: bias parameter of the model (usually set to 1)
- α (alpha): learning rate, a small positive number that controls the step size during updates (typically between 0 and 1)
- ∂J(w, b) / ∂w: partial derivative of the cost function (J) with respect to w (measures how much the cost changes in the direction of w)
- ∂J(w, b) / ∂b: partial derivative of the cost function (J) with respect to b (measures how much the cost changes in the direction of b)

Assignment Operator vs. Truth Assertion

- The equal sign (=) in this context is the assignment operator, not a truth assertion.
- Assigning a value to a variable: a = c (stores the value of c in variable a)
- Updating a variable: a = a + 1 (increments the value of a by 1)

Learning Rate (α)

- Alpha controls the step size taken downhill during gradient descent.
- A larger alpha leads to larger steps (faster but potentially unstable).
- A smaller alpha leads to smaller steps (slower but more stable).

Derivative Term

- The partial derivatives (dJ/dw and dJ/db) indicate the direction of the steepest descent on the cost function surface.
- We won't delve into the details of calculus for now, but you'll gain the necessary intuition in this and the next video.

Simultaneous Updates

- Both w and b are updated simultaneously in each iteration.
- This is crucial for gradient descent to work correctly.
- Incorrect implementation (updating w first, then b) leads to a different algorithm with different properties.

$$J(w)$$

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

WHY IS THERE A NEGATIVE SYMBOL ON THE TERM CONTAINING DERIVATIVE OF COST FUNCTION?

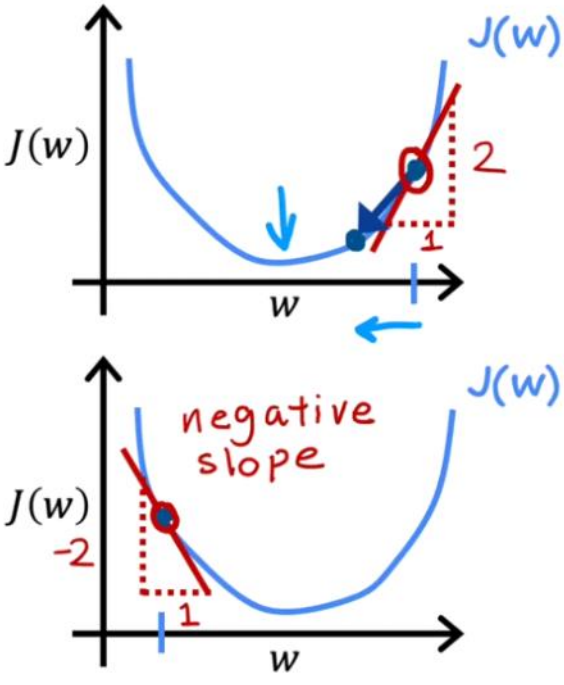
The negative sign in the formula for gradient descent is there to ensure the algorithm moves **downhill** on the cost function surface, ultimately reaching a minimum. Here's a breakdown:

1. Gradient Points Downhill:
The gradient of a function points in the direction of the **steepest ascent**. In gradient descent, we want to move in the opposite direction, which is **steepest descent**. The negative sign flips the direction of the gradient, ensuring we move downhill.

2. Intuition with Derivatives:
Imagine the cost function visualized as a hilly landscape. The derivative at a point represents the slope of the tangent line touching the hill at that point.

- **Positive Derivative:** If the slope is positive (slanted upwards), the negative sign ensures we subtract a positive value from the parameter (w), effectively moving us **left** (downhill) on the graph.
- **Negative Derivative:** If the slope is negative (slanted downwards), the negative sign flips the negative derivative to positive. Subtracting a negative number (adding a positive) moves us **right** (still downhill) on the graph.

3. Without the Negative Sign:
Without the negative sign, the update would be based on the raw gradient direction. This could lead us **uphill** depending on the starting position and the slope at that point.



$$w = w - \alpha \frac{d}{dw} J(w)$$

$$> 0$$

$$w = w - \alpha \cdot (\text{positive number})$$

$$\frac{d}{dw} J(w) < 0$$

$$w = w - \alpha \cdot (\text{negative number})$$

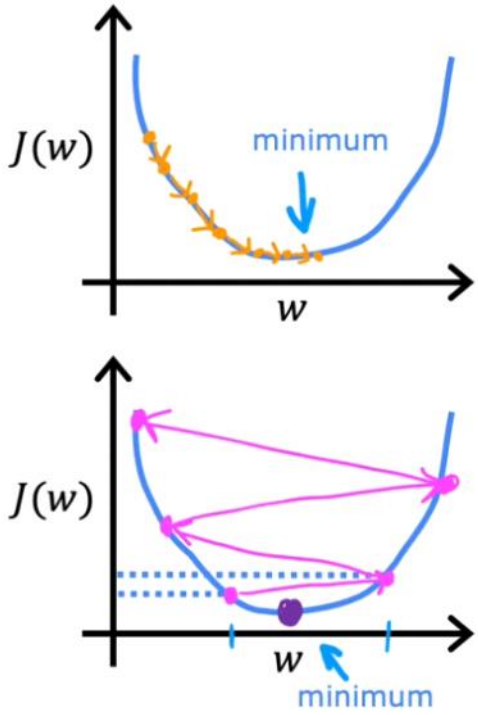
LEARNING RATE:

$$w = w - \alpha \frac{d}{dw} J(w)$$

If α is too small...
Gradient descent may be slow.

If α is too large...

Gradient descent may:
- Overshoot, never reach minimum
- Fail to converge, diverge



Impact of Learning Rate

- Alpha determines the step size taken during each update in gradient descent.
- A well-chosen alpha leads to efficient convergence towards the minimum cost.
- Improper alpha values can hinder performance.

Learning Rate Too Small (Slow Progress)

1. **Scenario:** Imagine a cost function J(w) plotted as a curve.
2. **Effect:** A very small learning rate results in minuscule steps during updates.
3. **Consequence:** Gradient descent crawls towards the minimum cost at a slow pace, requiring many iterations.

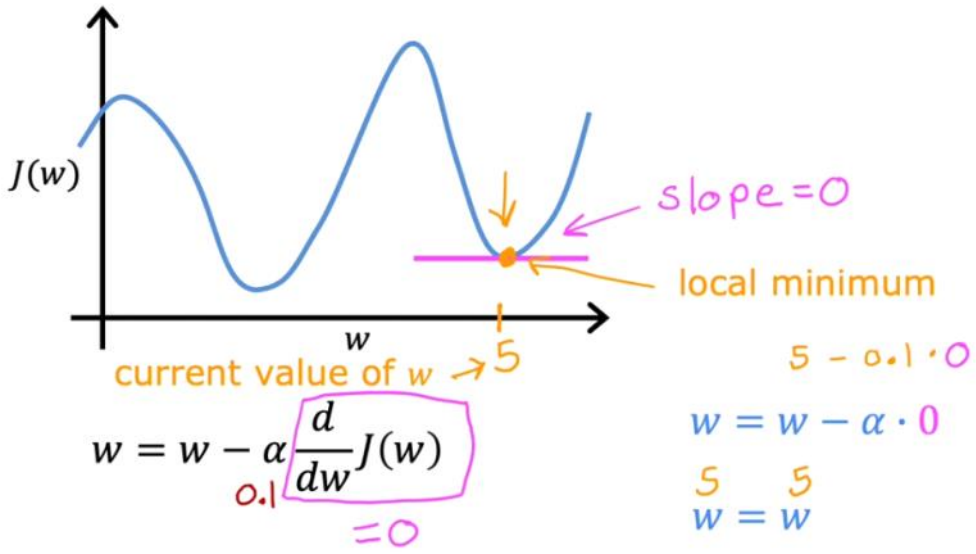
Visualizing Slow Progress:

- The graph shows the cost function (J) on the vertical axis and the parameter (w) on the horizontal axis.
- The starting point is marked on the curve.
- Tiny steps are depicted as the algorithm progresses towards the minimum.

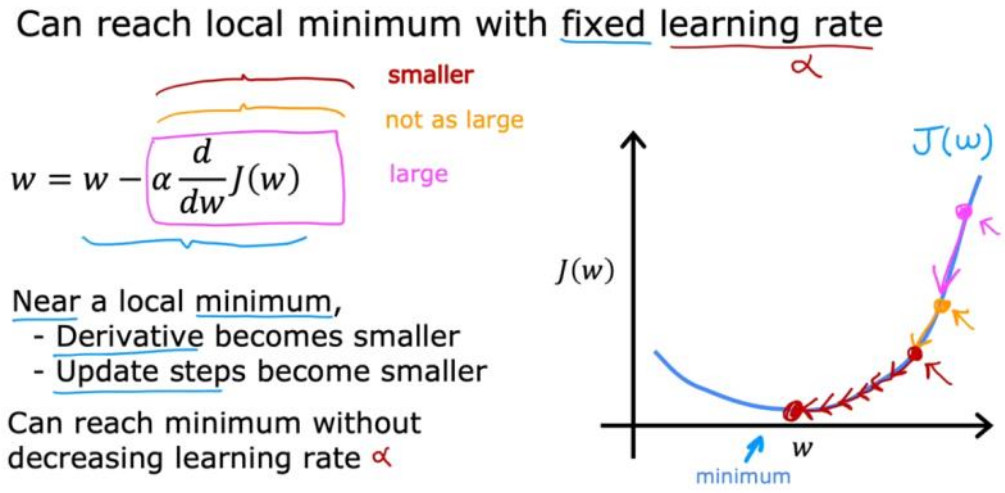
Learning Rate Too Large (Overshooting)

1. **Scenario:** A large learning rate causes significant jumps during updates.
2. **Effect:** The algorithm might jump right past the minimum and oscillate back and forth, never converging.
3. **Consequence:** Gradient descent might diverge or fail to reach the minimum efficiently.

Now what if w is already at a local minimum of cost function?



No change in w.



Visualizing Overshooting:

- Imagine starting near the minimum.
- A large learning rate step overshoots the minimum, landing on the other side of the curve.
- Subsequent steps cause the algorithm to bounce around, failing to settle at the minimum.

Local Minima and Stopping Conditions

- Gradient descent can get stuck in local minima (valleys) on the cost function.
- When the derivative (slope) at a local minimum is zero, the update becomes zero, effectively stopping further movement.

Gradient Descent for Linear Regression

29 September 2022 01:18

Here's the linear regression model. To the right is the squared error cost function. Below is the gradient descent algorithm.

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

}

DERIVATION:

(Optional)

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w}\frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{\partial}{\partial w}\frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)} + b - y^{(i)}\right)2x^{(i)} = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial}{\partial b}\frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{\partial}{\partial b}\frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)} + b - y^{(i)}\right)2 = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

no $x^{(i)}$

Gradient descent algorithm

$\frac{\partial}{\partial w} J(w,b)$

repeat until convergence {

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) \; x^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

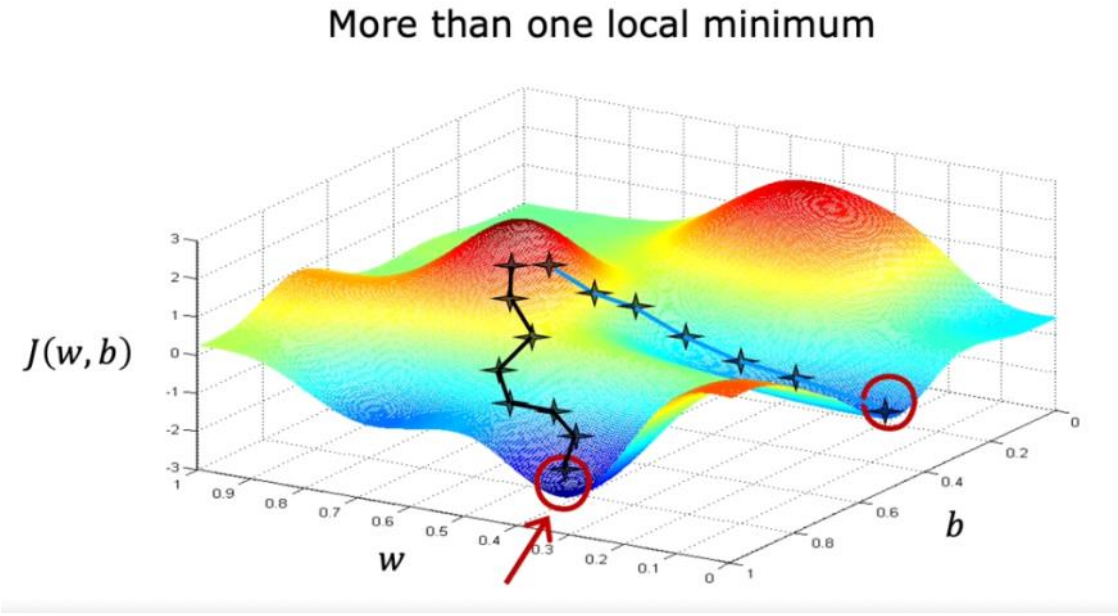
}

Update w and b simultaneously

$\frac{\partial}{\partial b} J(w,b)$

$f_{w,b}(x^{(i)}) = wx^{(i)} + b$

PROBLEM WITH GRADIENT DESCENT: More than one minima. Ideally, we would want a global minima (lowest possible value).



But it turns out when you're using a squared error cost function with linear regression, the cost function does not and will never have multiple local minima.
It has a single global minimum because of this bowl-shape.

Key Components

- **Linear Regression Model:** Predicts a target value (y) based on a linear equation (y = wx + b) where:
  - w: weight parameter
  - b: bias parameter
  - x: input feature
- **Squared Error Cost Function:** Measures how well the model fits the training data by calculating the sum of squared errors between predicted and actual values.
- **Gradient Descent Algorithm:** Iteratively updates the model parameters (w and b) to minimize the cost function.

Deriving the Update Rules

The video acknowledges that the update rules (derivatives) for gradient descent can be derived using calculus. It provides the formulas but assures viewers that understanding the derivation is not essential for implementation.

Optional Calculus Step (Skippable Slide):

This slide (optional) shows the detailed mathematical derivation of the update rules for w and b using partial derivatives.

Gradient Descent Algorithm for Linear Regression:

The algorithm iteratively updates w and b using the calculated derivatives until convergence:
w = w - α * (1/m) * Σ(predicted_i - actual_i) * x_i
b = b - α * (1/m) * Σ(predicted_i - actual_i)

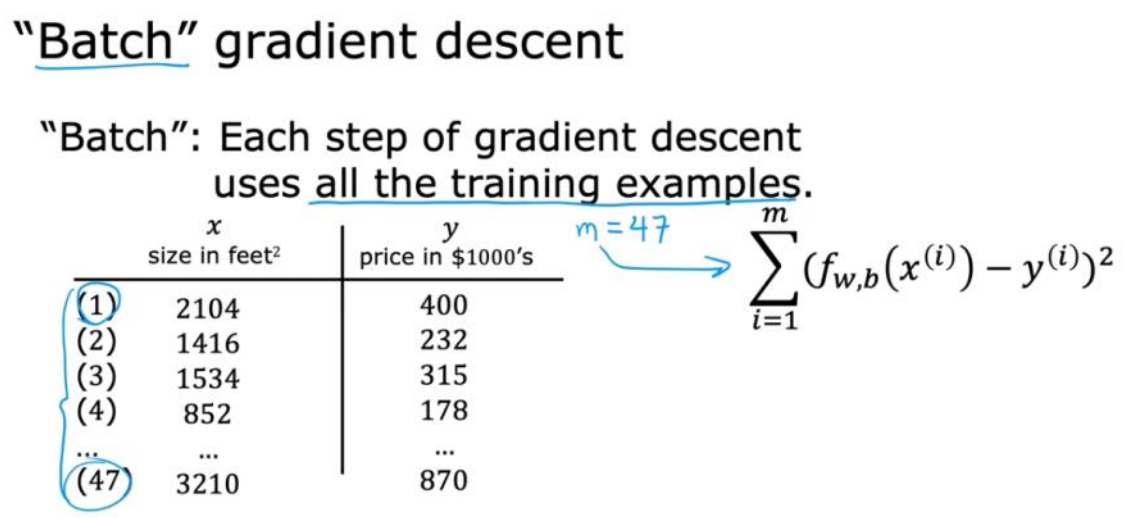
- α (alpha): learning rate (controls step size)
- m: number of training examples
- Σ: summation over all training examples (i = 1 to m)
- predicted_i: predicted value for training example i
- actual_i: actual value for training example i
- x_i: input feature for training example i

Benefits of Squared Error Cost Function with Linear Regression:

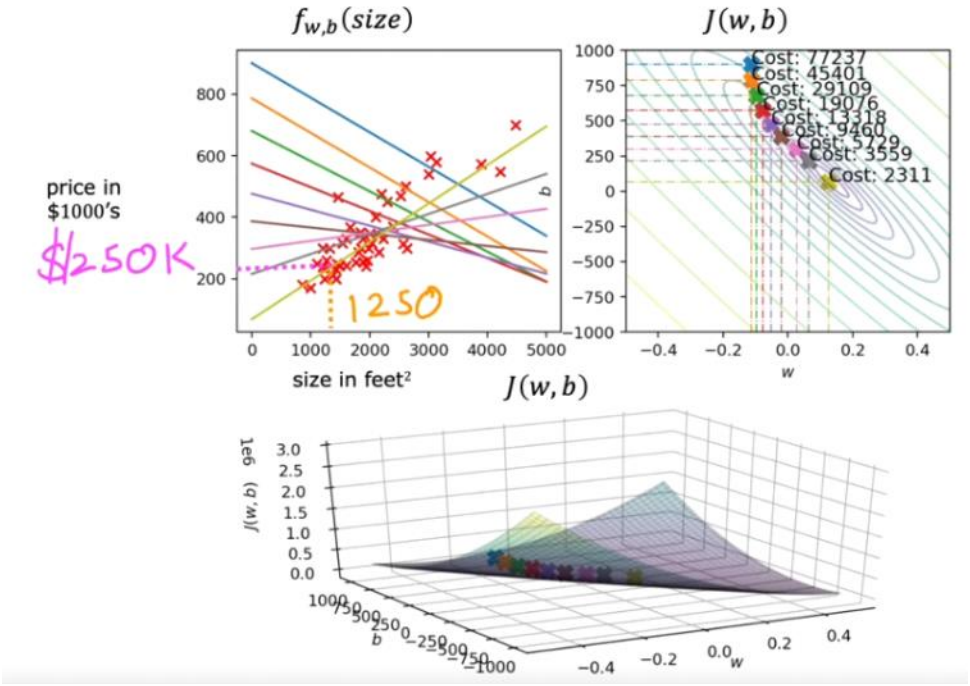
- The squared error cost function, when used with linear regression, guarantees a single global minimum (bowl-shaped convex function).
- Gradient descent with a suitable learning rate will always converge to this global minimum, leading to the best possible model fit for the training data.

The technical term for this is that this cost function is a convex function. Informally, a convex function is of bowl-shaped function and it cannot have any local minima other than the single global minimum.
When you implement gradient descent on a convex function, one nice property is that so long as you're learning rate is chosen appropriately, it will always converge to the global minimum.

RUNNING GRADIENT DESCENT ON LINEAR REGRESSION:



	x size in feet²	y price in $1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870


$f_{w,b}(size)$
price in $1000's
$250K
1250
size in feet²
$J(w,b)$
Cost: 77237
Cost: 45401
Cost: 29109
Cost: 19076
Cost: 13318
Cost: 9460
Cost: 5729
Cost: 3559
Cost: 2311
w
b
$J(w,b)$
1e6
$J(w,b)$

Multiple Features

18 December 2022 10:17

Single Feature vs. Multiple Features

- Previously, linear regression used a single feature (e.g., house size) to predict an output (e.g., price).
- The model equation was f(x) = wx + b, where x is the single feature and w and b are the parameters.

one feature →

Size in feet² (x)	Price ($) in 1000's ($y$) ←
2104	400
1416	232
1534	315
852	178
...	...

$$f_{w,b}(x) = wx + b$$

Multiple Features for Richer Data

- Now, we consider multiple features for prediction (e.g., size, bedrooms, floors, age).
- New notation:
  - X_1, X_2, ..., X_n represent n features.
  - **X^i denotes a vector containing all features for the i^{th} training example.**
  - $X^{(i)}_{[j]}$ refers to the specific j^{th} feature in the i^{th} training example.

Multiple features (variables)

Size in feet² X_1	Number of bedrooms X_2	Number of floors X_3	Age of home in years X_4	Price ($) in $1000's
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$j = 1 \ldots 4$

$n = 4$

$i = 2$

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = features of i^{th} training example

$x_j^{(i)}$ = value of feature j in i^{th} training example

$\vec{x}^{(2)} = [1416 \ 3 \ 2 \ 40]$

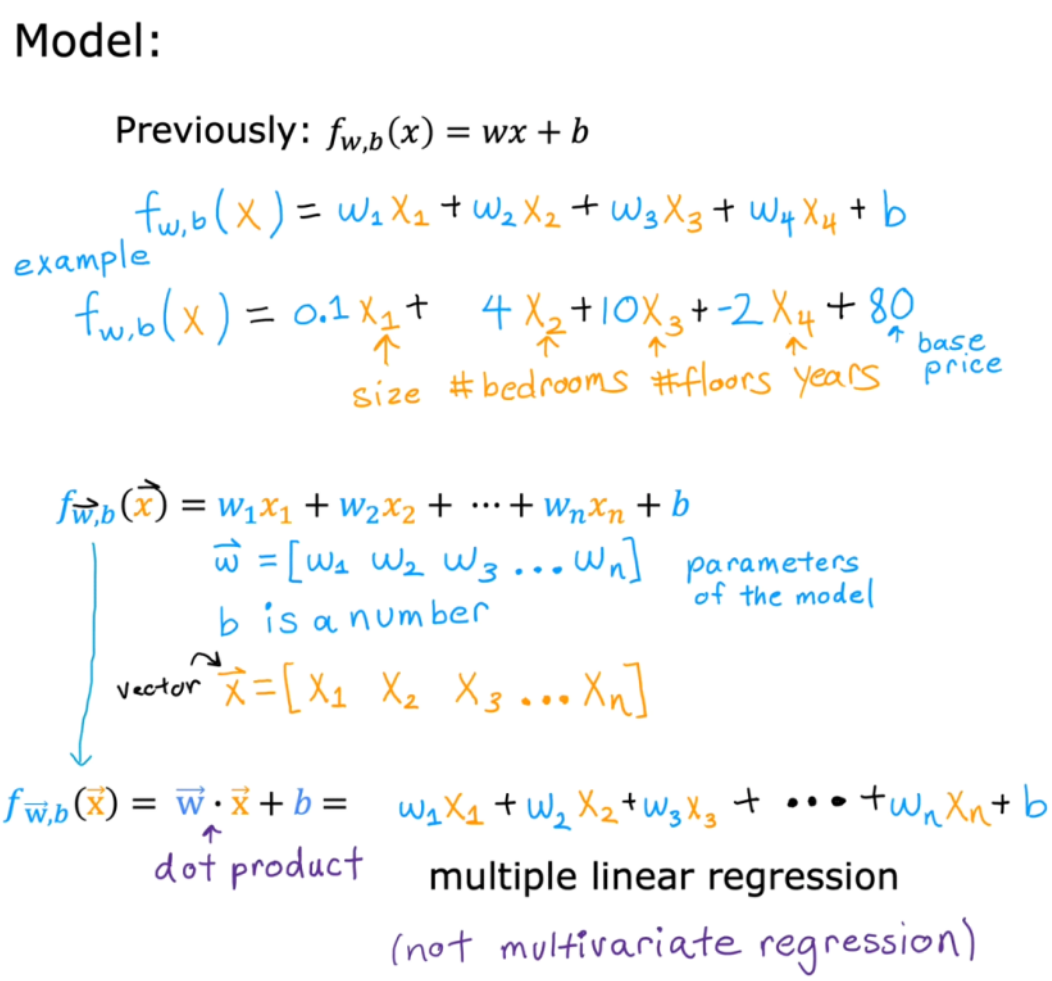
$x_3^{(2)} = 2$

Model with Multiple Features

The model equation is modified to accommodate multiple features:

- f(X) = w_1X_1 + w_2X_2 + ... + w_nX_n + b
- Each w_i corresponds to the weight/coefficient of the i-th feature.

- A positive w_i indicates a positive correlation (increases prediction with higher feature value).
- A negative w_i indicates a negative correlation (decreases prediction with higher feature value).
- **The bias term (b) represents the base prediction when all features are zero.**



Example: Predicting House Price

- A possible model for house price prediction:
  - f(X) = 0.1X_1 (size) + 4X_2 (bedrooms) + 10X_3 (floors) - 2X_4 (age) + 80 (base price)
- Interpretation of parameters:
  - Every additional square foot increases the price by $100.
  - Each additional bedroom increases the price by $4,000.
  - Each additional floor increases the price by $10,000.
  - Each year of age decreases the price by $2,000 (assuming negative impact on value).

Model in Compact Form with Vectors

- We can define vectors for weights (w) and features (X) for a more concise representation.
- The dot product (·) is used between w and X to efficiently calculate the linear combination.
- f(X) = w · X + b

Multiple Linear Regression vs. Univariate Regression

- This model with multiple features is called **multiple linear regression**.
- It contrasts with **univariate regression** that uses only one feature.

Vectorization

27 April 2024 10:47

Benefits of Vectorization

- **Shorter Code:** Vectorization reduces complex calculations to a single line of code, improving readability and maintainability.
- **Enhanced Efficiency:** It leverages modern numerical libraries and hardware for significantly faster execution, especially for large datasets.

Understanding Vectorization

Consider a scenario with parameters w (vector with 3 numbers) and features x (another vector with 3 numbers). Here, n represents the number of features (n = 3).

Indexing in Linear Algebra vs. Python

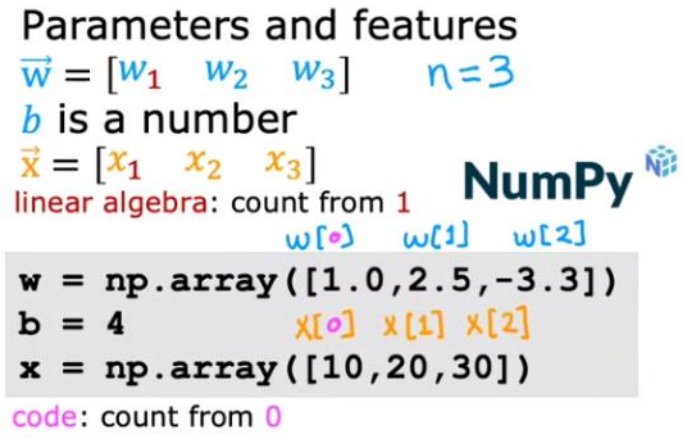
- Linear Algebra: Indexing starts from 1 (w1, x1).
- Python (NumPy): Indexing starts from 0 (w[0], x[0]).

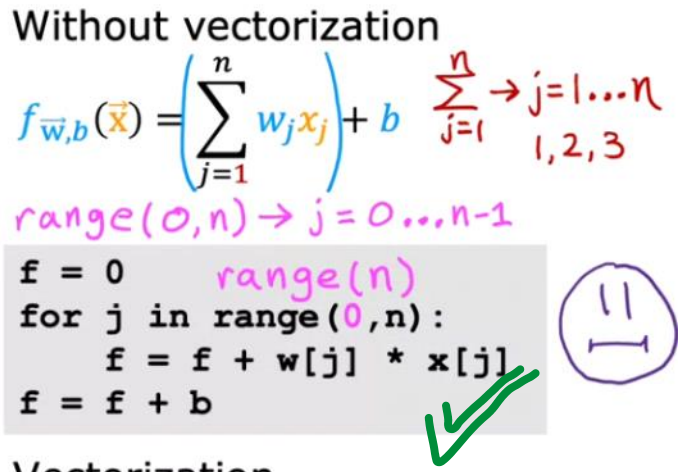
Non-Vectorized Implementations

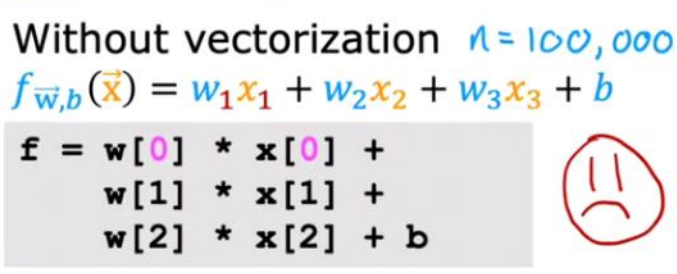
1. **Manual Multiplication:**
   Inefficient for large datasets (n = 100, 100,000).
2. **For Loop Implementation:**
   Improves readability but doesn't utilize vectorization.

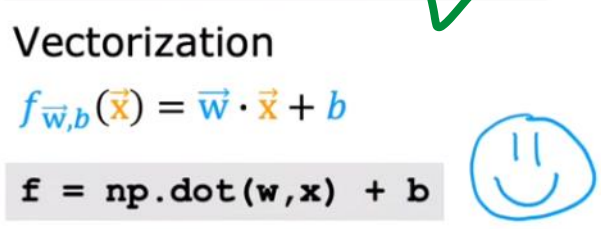
Vectorized Implementation using NumPy

- **Single Line of Code:** fp = np.dot(w, x) + b
  np.dot efficiently calculates the dot product of w and x.







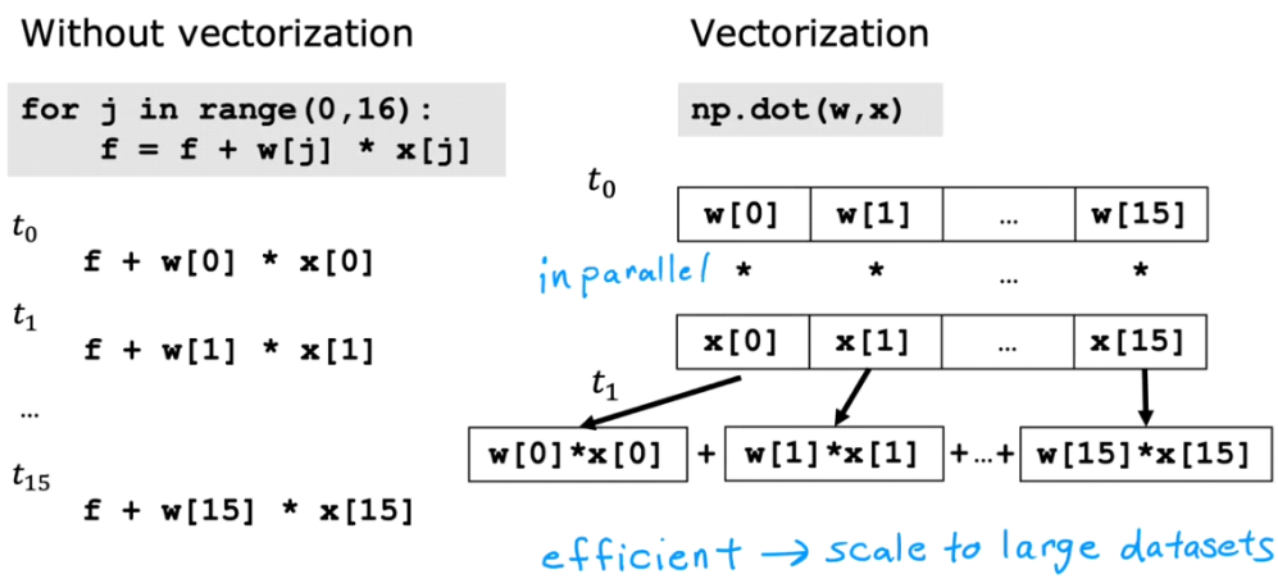


Advantages of NumPy's dot Function

- **Vectorized Operation:** Handles calculations element-wise between vectors.
- **Hardware Acceleration:** Utilizes parallel processing capabilities of CPUs and GPUs for faster execution.

The Allure of Vectorization

- **Condensed Code:** Vectorization transforms complex calculations into a single line, enhancing readability and maintainability.
- **Exceptional Efficiency:** It leverages modern numerical libraries and hardware for dramatic performance gains, especially with large datasets.



Understanding Vectorization in Action

Consider a for loop implementing a non-vectorized algorithm. Let j range from 0 to 15. This loop performs operations sequentially:

- At time t0, it processes values at index 0.
- In subsequent steps (t1, t2, ..., t15), it calculates values for indices 1 to 15, one at a time.

In contrast, NumPy's vectorized functions operate differently:

1. **Simultaneous Processing:** The computer retrieves all values from w and x vectors and performs element-wise multiplication **in parallel**.
2. **Efficient Summation:** It utilizes specialized hardware to efficiently sum the 16 resulting products, avoiding multiple additions.

This parallel approach translates to significantly faster execution, especially for large datasets encountered in machine learning.

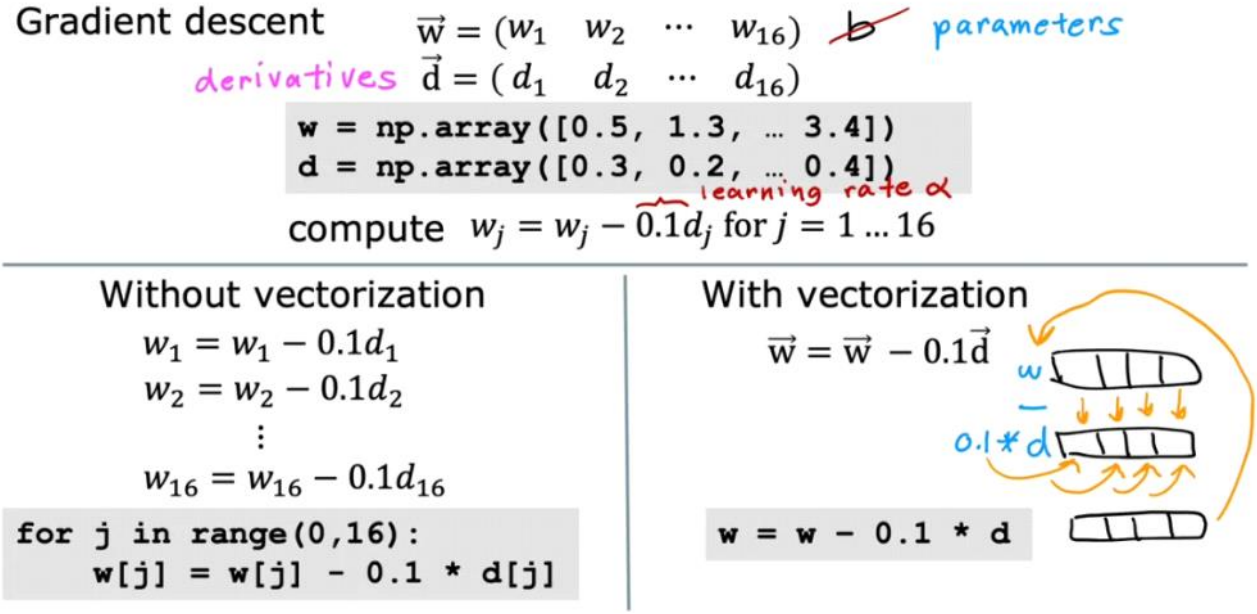
Imagine a scenario with 16 features (w1 to w16) and a parameter b. You need to calculate 16 derivative terms for these weights.

Non-Vectorized Implementation:

This approach would involve a for loop iterating from 1 to 16, updating each weight (w_j) by subtracting the learning rate times the corresponding derivative (d_j).

Vectorized Implementation with NumPy:

Vectorization allows the computer's parallel processing hardware to perform all 16 subtractions simultaneously. In code, this translates to a simple assignment: w = w - 0.1 * d.



Gradient Descent for Multiple Linear Regression

27 April 2024 13:53

Multiple Linear Regression in Vector Notation

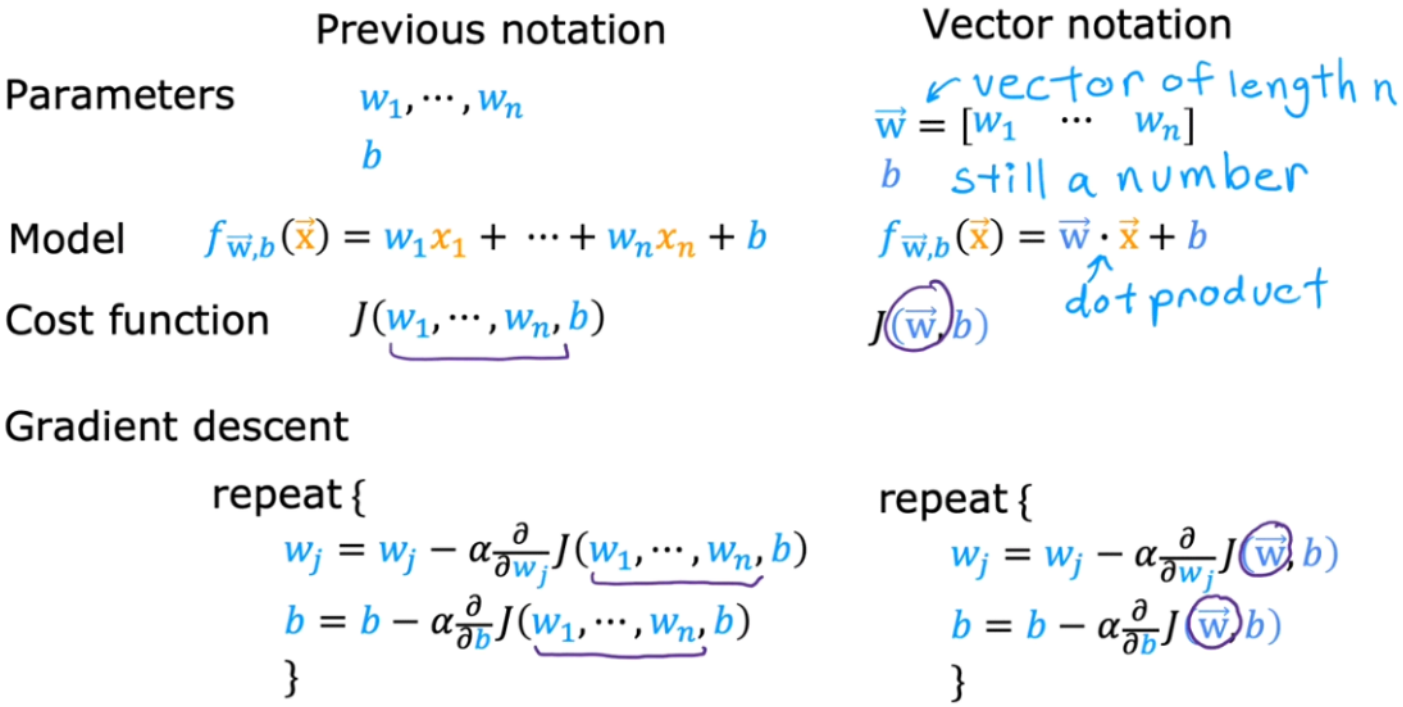
- We have parameters w_1 to w_n and b.
- Instead of separate parameters, we define a vector w of length n to represent all ws.
- The model is written as f_w(x) = w · x + b (dot product between w and x).

Cost Function with Vectorization

- The cost function J is defined for w and b.
- It represents the function we want to minimize to find optimal parameters.

Gradient Descent Update Rule

- We iteratively update each parameter w_j using: w_j := w_j - α ∂J(w, b) / ∂w_j
  - α is the learning rate.
  - ∂J(w, b) / ∂w_j is the partial derivative of J with respect to w_j.



Gradient Descent with Multiple Features

- The update rule for w_j becomes: w_j := w_j - α Σ(f(x^(i)) - y^(i)) * x^(i)_j
  - x^(i) is the i-th training example (feature vector).
  - y^(i) is the target value for the i-th example.
  - The summation iterates over all training examples.

Gradient descent

One feature

repeat {

$$w = w - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$$

$$\frac{\partial}{\partial w}J(w,b)$$

$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)$$

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$j = 1$

$$w_1 = w_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)x_1^{(i)}$$

$$\frac{\partial}{\partial w_1}J(\vec{w},b)$$

$\vdots$

$j = n$

$$w_n = w_n - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)$$

simultaneously update w_j (for $j = 1, \cdots, n$) and b

}

An alternative to gradient descent

→ Normal equation

- Only for linear regression
- Solve for w, b without iterations

Disadvantages

- Doesn't generalize to other learning algorithms.
- Slow when number of features is large (> 10,000)

What you need to know

- Normal equation method may be used in machine learning libraries that implement linear regression.
- Gradient descent is the recommended method for finding parameters w,b

Key Points

- Vectorization allows efficient calculations for multiple features.
- Gradient descent updates each parameter w_j based on the derivative of the cost function.
- The normal equation (alternative method) is not generally recommended due to limitations with other algorithms and large datasets.

Feature Scaling

28 April 2024 13:02

Impact of Feature Scale on Parameter Values

The relationship between feature scale and parameter values is crucial. Consider a model predicting house prices based on size (x1) and bedrooms (x2). Here's an example:

- **Feature Ranges:**
  - x1 (Size): 300 - 2000 sq ft (large range)
  - x2 (Bedrooms): 0 - 5 (small range)
- **Sample Training Example:**
  - Size: 2000 sq ft
  - Bedrooms: 5
  - Price: $500,000

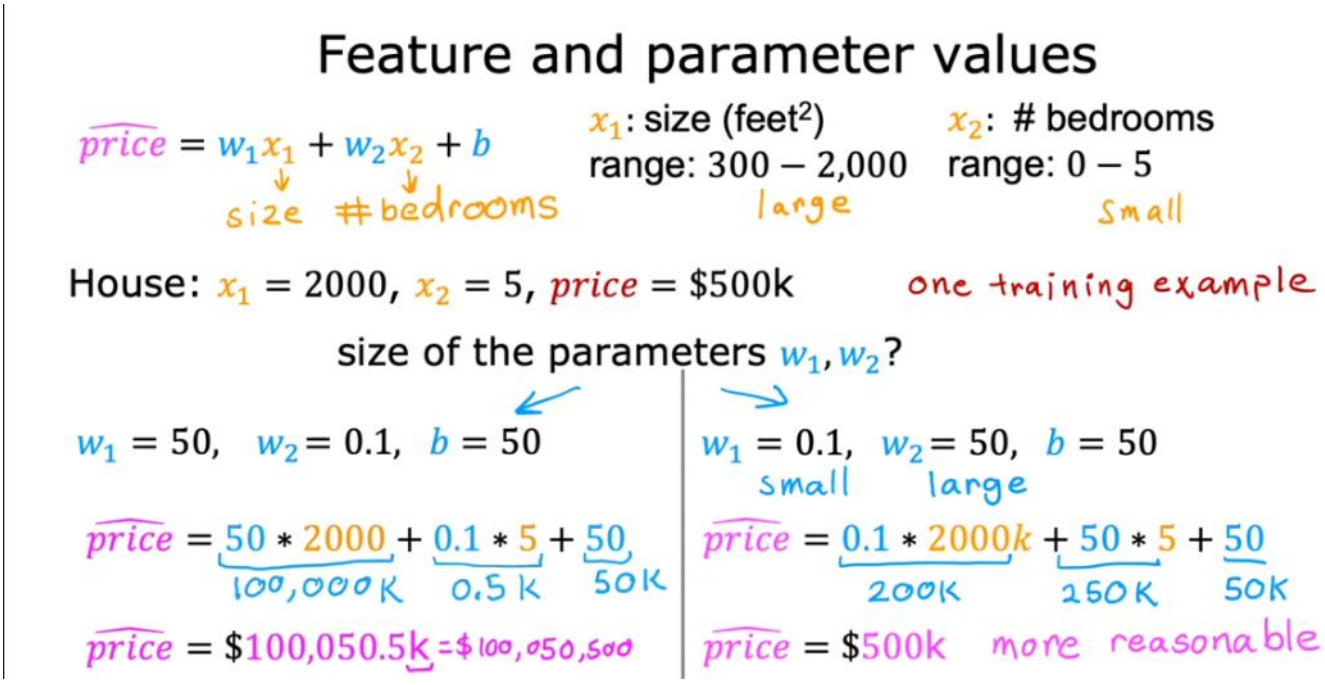
Let's analyze two possible parameter choices (w1, w2, and bias b) for this example:

Choice 1 (w1 = 50, w2 = 0.1, b = 50):

Predicted Price = (100,000 + 0.5 + 50) thousands of dollars ≈ $100,100,000 (incorrect)

Choice 2 (w1 = 0.1, w2 = 50, b = 50):

Predicted Price = (0.1 * 2000) + (50 * 5) + 50 thousands of dollars = $500,000 (correct)



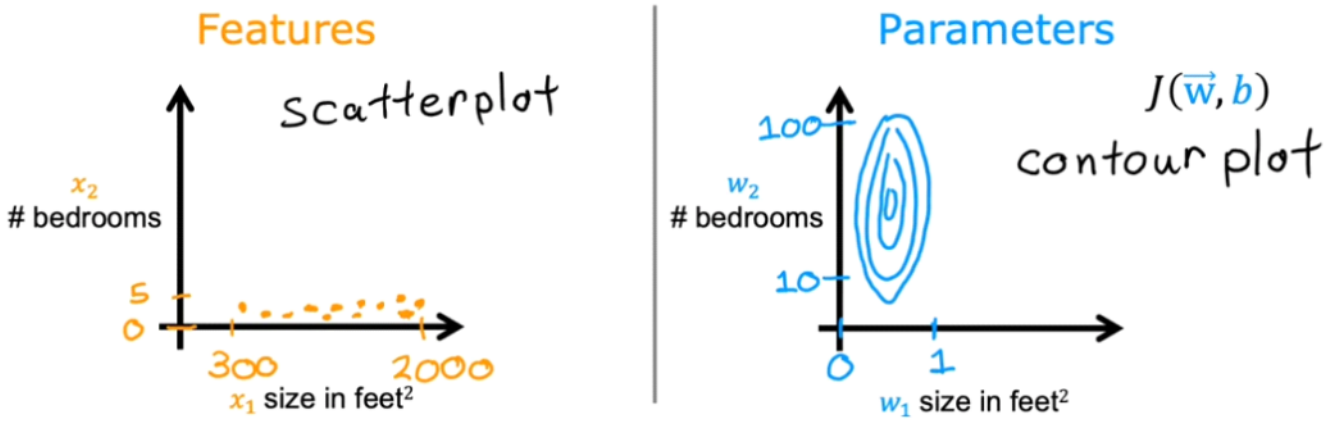
This demonstrates that features with large value ranges tend to have smaller parameter values in a well-performing model, and vice versa.

Feature Scale and Gradient Descent

Imagine a scatter plot where x1 (size) is on the horizontal axis and x2 (bedrooms) is on the vertical axis.

Feature size and parameter size

	size of feature x_j	size of parameter w_j
size in feet^2	⟷	⟷
#bedrooms	⟷	⟷



The horizontal axis, x1x1, represents the size in square feet, while the vertical axis represents the number of bedrooms. When plotting the training data, it becomes apparent that x1x1spans a much broader range of values compared to the number of bedrooms.
Moving on to the contour plot of the cost function, you'll likely observe that the horizontal axis has a narrower range, typically between zero and one, whereas the vertical axis encompasses larger values, often between 10 and 100. Consequently, the contours on the plot assume an elliptical or oval shape, with one side being shorter and the other longer.
This characteristic arises because a minute alteration to w1 can exert a substantial influence on the estimated price and consequently on the cost function J. Since w1 is typically multiplied by a large value—the size in square feet—a small change in w1w1can lead to a significant shift in the prediction and subsequently in the cost J.
Conversely, w2 typically requires a more considerable adjustment to induce a noticeable change in predictions. Therefore, minor variations in w2 yield less pronounced alterations in the cost function.

This shape arises because small changes in w1 (due to a large x1 scale) significantly impact the predicted price and cost function (J). Conversely, larger changes in w2 (due to a small x2 scale) are needed to affect the cost function.

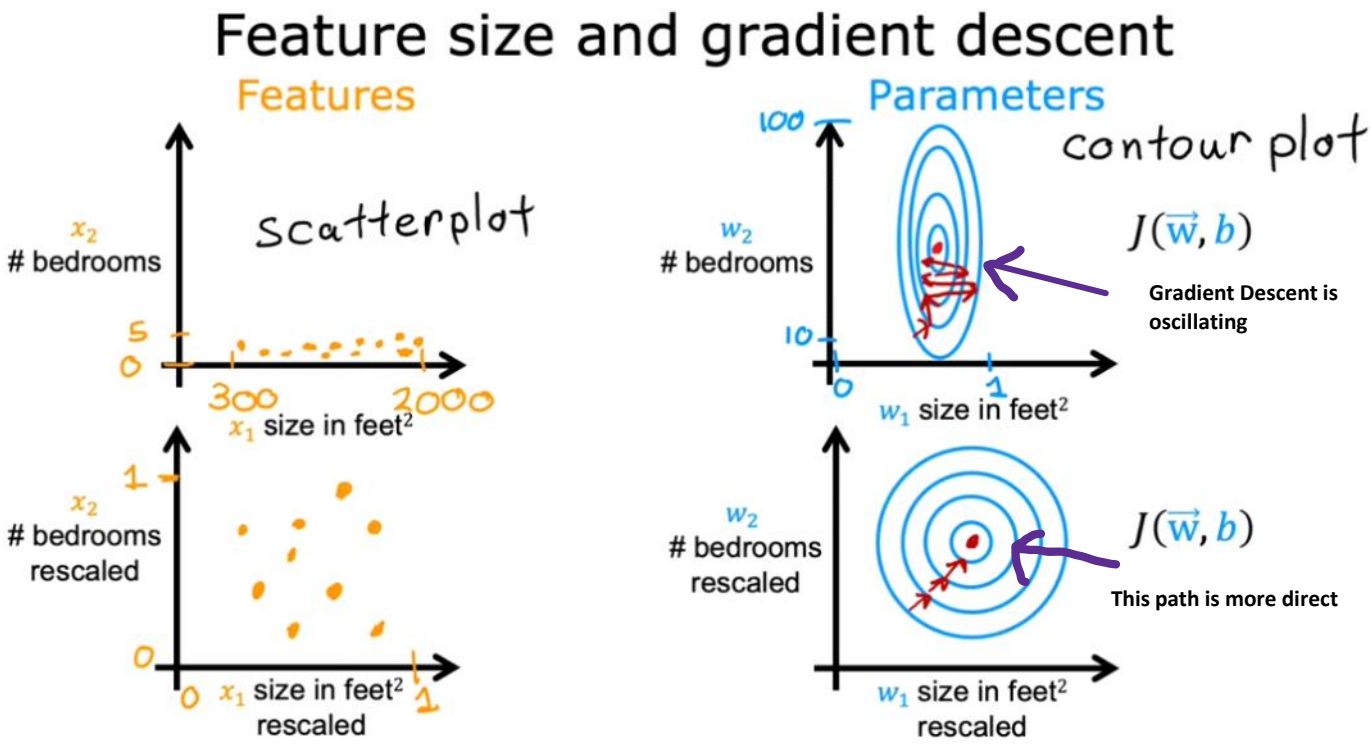
Challenges with Unscaled Features

Running gradient descent with unscaled features can lead to:

- **Slow convergence:** Gradient descent may oscillate for a long time before reaching the minimum.

Solution: Feature Scaling

Feature scaling involves transforming the training data so that all features have comparable ranges.
For example, scaling x1 and x2 to a range of 0 to 1.



This transformation results in:

- Rescaled data points with similar scales on both axes.
- More circular contours in the cost function plot.
- Gradient descent finding a more direct path to the minimum.

Benefits of Feature Scaling

- **Faster Gradient Descent Convergence:** Gradient descent converges significantly faster with scaled features.

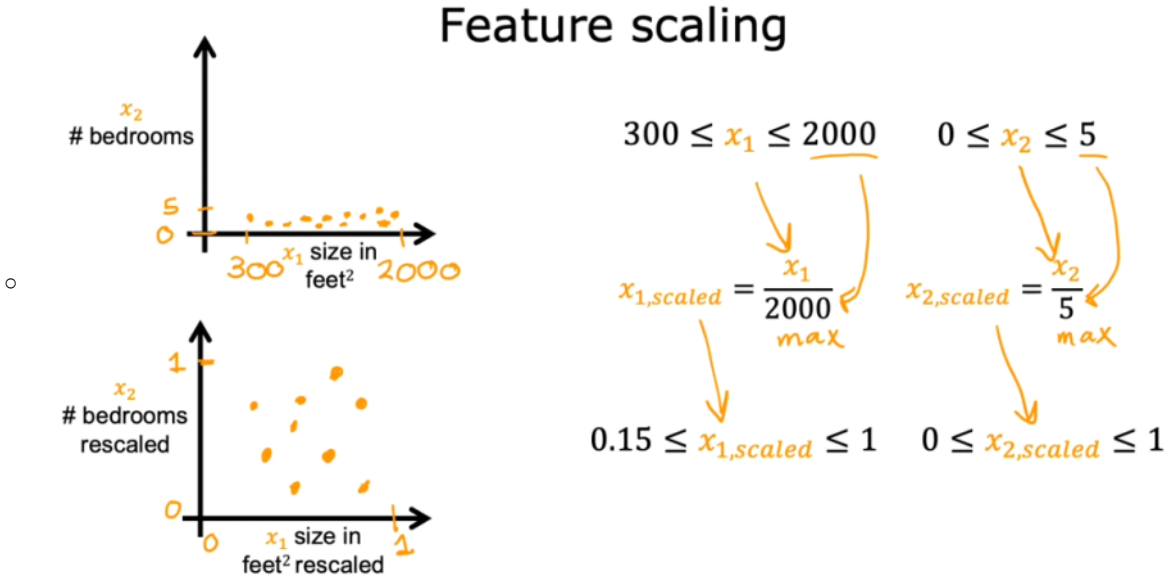
What is Feature Scaling?

Feature scaling is a technique for transforming features in a dataset so that they have a comparable range. This is important for machine learning algorithms, especially those that rely on gradient descent optimization.

Feature Scaling Techniques

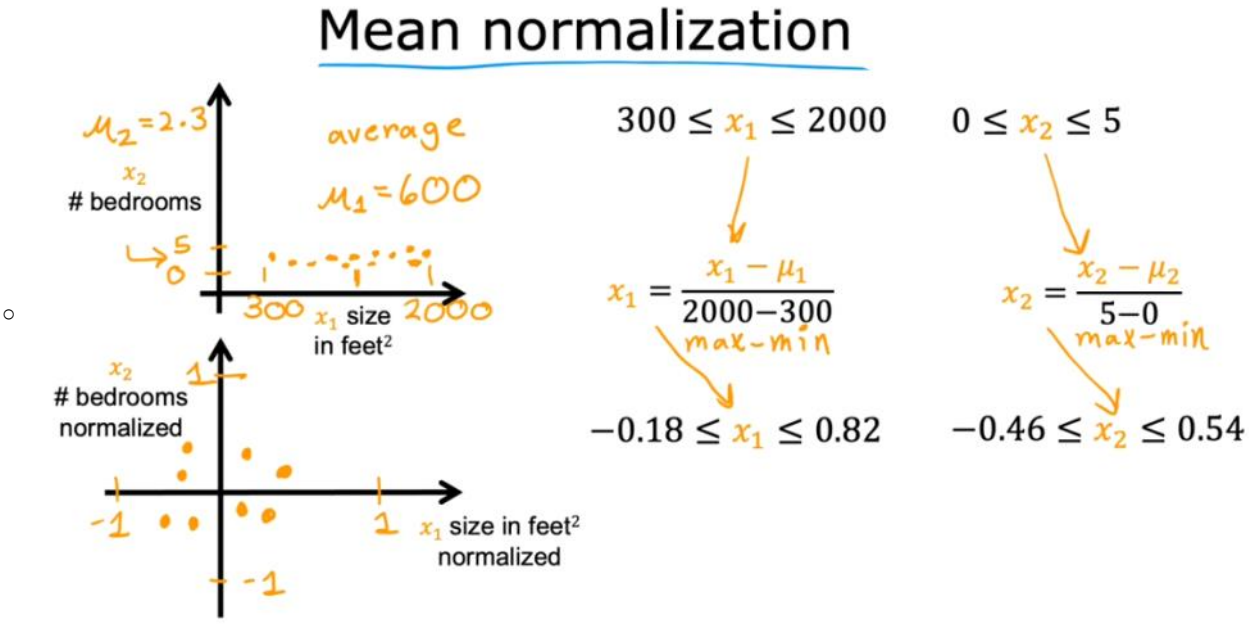
There are three common feature scaling techniques:

1. **Min-Max Scaling:**
    - Divide each feature value by the maximum value in its range.
    - Typical new range: 0 to 1.



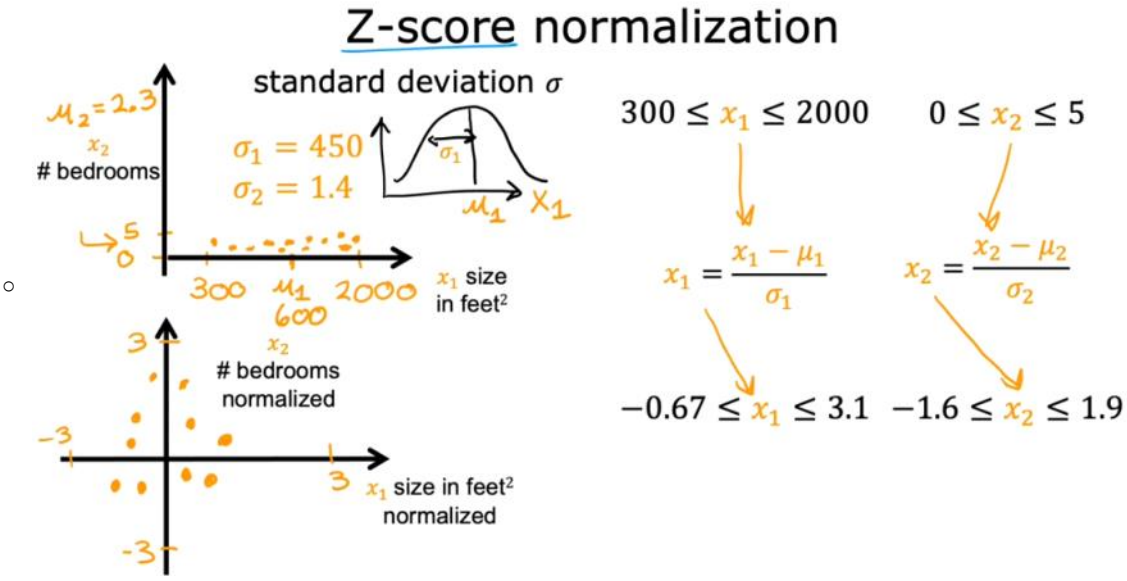
2. **Mean Normalization:**

- Subtract the mean (average) of each feature from its values.
- Divide by the difference between the maximum and minimum values in the feature's range.
- Typical new range: Centered around zero, with both positive and negative values.



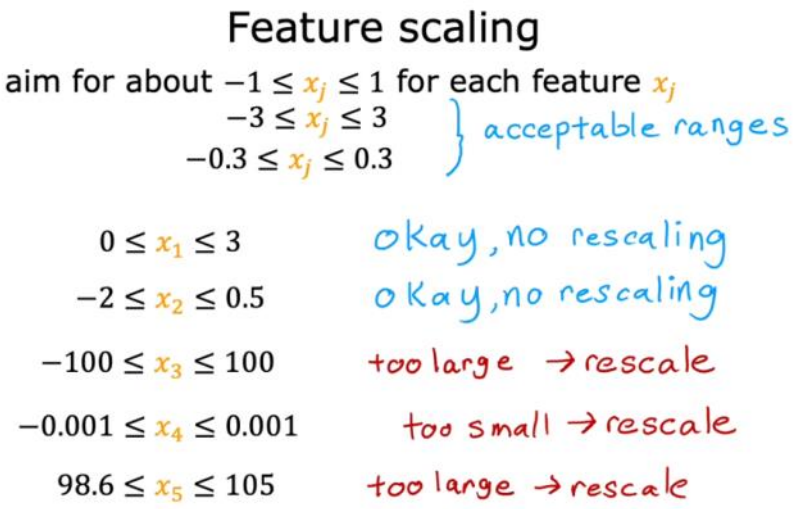
3. Z-score Normalization:

- Subtract the mean of each feature from its values.
- Divide by the standard deviation of each feature.
- New range: Often between negative one and plus one, but can vary.



Rule of Thumb for Feature Scaling

Aim for features to range from negative one to plus one for each feature, but these values can be flexible.



However, these values can vary. Ranges like -3 to +3 or -0.3 to +0.3 are also acceptable.
For instance, if x1 ranges from 0 to 3, it's not an issue; rescaling is optional.
Similarly, if x2 ranges from -2 to +0.5, it's acceptable, though rescaling is optional.

When to Rescale

- Rescale features with significantly different ranges (e.g., -100 to 100 vs. 0 to 3).
- Rescale features with very small values (e.g., -0.001 to 0.001).

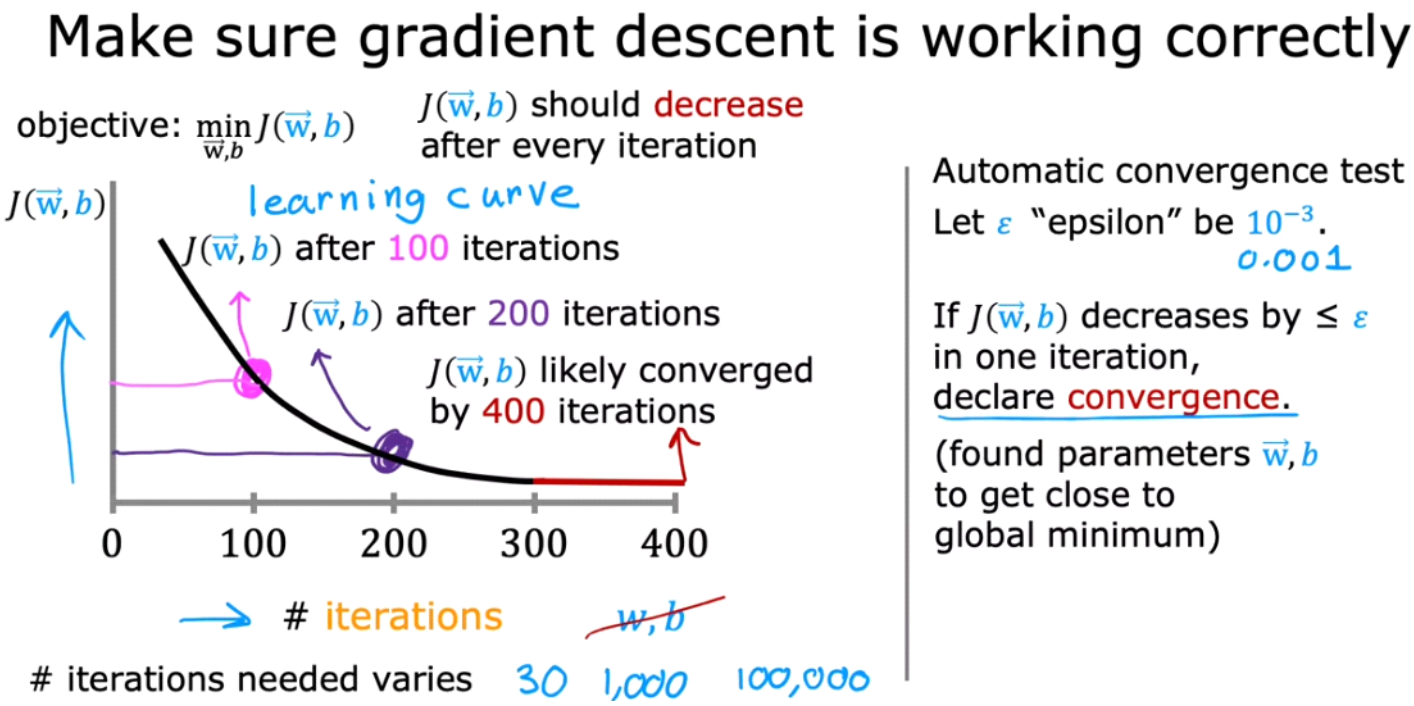
Gradient Descent Convergence

Monitoring Cost Function (J)

- Plot the cost function (J) calculated on the training set after each gradient descent iteration.
- The horizontal axis represents the number of iterations.

Learning Curve

- This plot is a type of learning curve used in machine learning.
- Each point on the curve represents the cost (J) after a specific number of iterations.



Ideal Gradient Descent Behavior

- The cost (J) should decrease after every iteration.
- If J ever increases, it indicates a potentially poorly chosen learning rate (alpha) or a bug in the code.
- The curve should flatten out as iterations progress, signifying convergence (gradient descent has found a minimum).

Number of Iterations for Convergence

- The number of iterations for convergence can vary significantly between applications.
- A learning curve helps you identify when to stop training your specific model.

Automatic Convergence Test

- Epsilon (ε) represents a small number (e.g., 0.001).
- Convergence can be declared if the cost (J) decrease is less than epsilon on one iteration.
- Choosing the right epsilon threshold can be challenging.
- **It's generally better to rely on observing the learning curve rather than automatic tests.**

Solid Learning Curve as an Indicator

- A solid learning curve with a clear flattening can indicate successful gradient descent.

Choosing Learning Rate

29 April 2024 19:01

Impact of Learning Rate

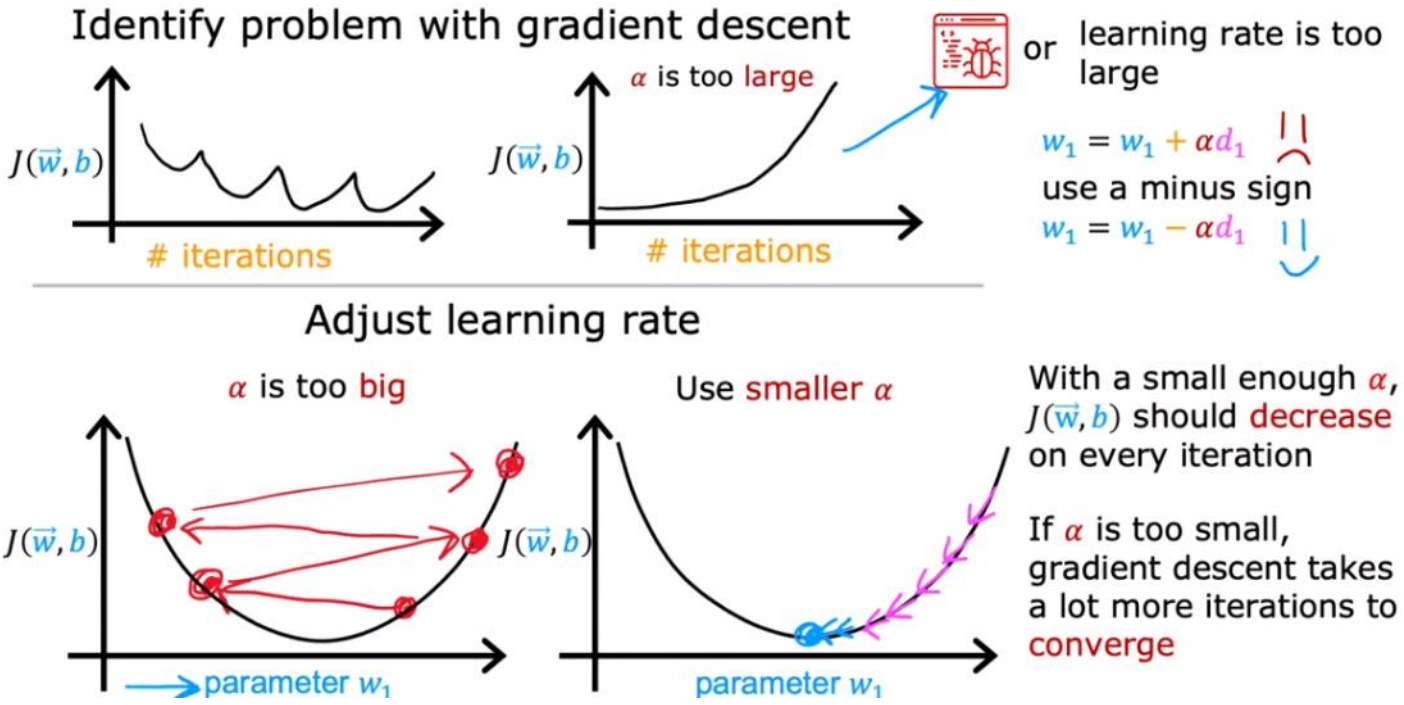
- **Too small:** Slow convergence (many iterations needed)
- **Too large:**
  - Cost function (J) oscillates (increases and decreases)
  - Cost function (J) consistently increases (possible bug in code or very large learning rate)

Learning Rate and Cost Function Behavior

- A large learning rate can cause overshooting the minimum during updates, leading to oscillating cost (J).
- A smaller learning rate can ensure more consistent decrease in cost (J) towards the minimum.
- Consistently increasing cost (J) can also indicate a bug (e.g., missing minus sign in gradient update) or a very large learning rate.

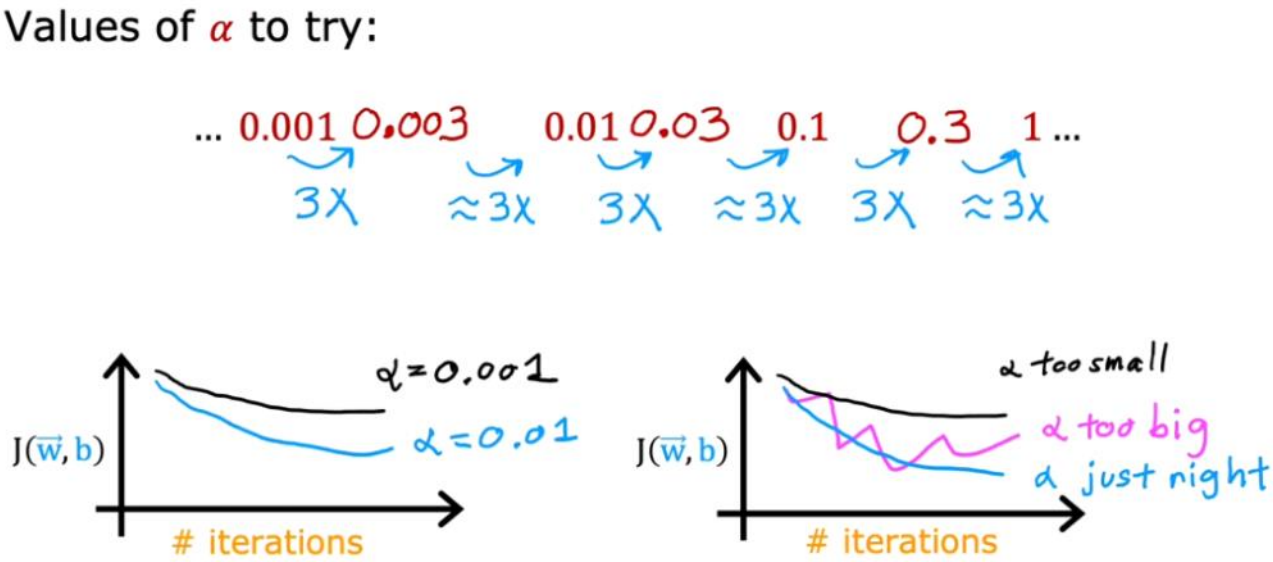
Debugging Gradient Descent with Learning Rate

- Set a very small learning rate.
- If the cost (J) still doesn't decrease on every iteration, there's likely a bug in the code.



Choosing a Good Learning Rate in Practice

1. Try a range of learning rates (e.g., 0.001, 0.01, 0.1).
2. Run gradient descent for a few iterations for each learning rate.
3. Plot the cost function (J) vs. number of iterations.
4. Choose the learning rate that leads to:
   - Rapid decrease in cost (J)
   - Consistent decrease in cost (J)



Example Approach

- Start with a small learning rate (e.g., 0.001).
- Gradually increase the learning rate by a factor of 3 (e.g., 0.003, 0.01).
- Continue until you find a learning rate that's both:
  - Too small (slow convergence)
  - Too large (oscillating or increasing cost)
- Choose the largest reasonable learning rate you found (or slightly smaller).

Learning Rate Selection Summary

- Experimenting with different learning rates is essential.
- This video provides intuition for choosing a good learning rate for gradient descent.

Feature Engineering

30 April 2024 00:28

Feature Engineering: Choosing Powerful Features

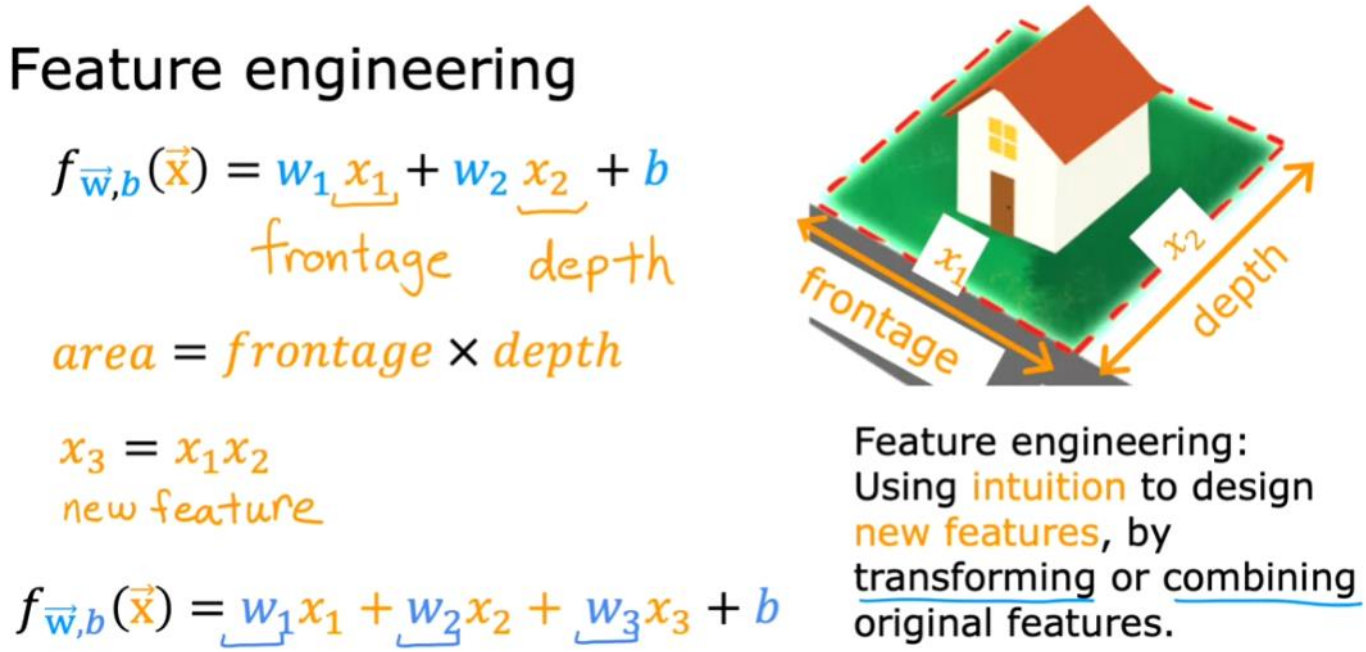
The choice of features significantly impacts a learning algorithm's performance. Choosing the right features is often critical for success. This video explores feature engineering for selecting or creating effective features.

Example: Predicting House Prices

Imagine predicting house prices using two features:

- x1: Lot width (frontage)
- x2: Lot depth (assuming a rectangular plot)

A basic model might use these features directly:
f(x) = w1 * x1 + w2 * x2 + b
(f(x) is the predicted price, w1 and w2 are weights, and b is the bias)



Feature Engineering for Improved Performance

We can potentially create a more effective model by considering the area (x3) of the lot:
x3 = x1 * x2 (area)
A new model can then utilize all three features:
f_w,b(x) = w1 * x1 + w2 * x2 + w3 * x3 + b
This allows the model to learn the relative importance of frontage, depth, and area in predicting price.

Impact of Feature Engineering

Creating new features through feature engineering is a powerful technique. It leverages your knowledge or intuition about the problem to:

- Design new features
- Transform or combine existing features

This can significantly improve the performance of your learning algorithm by making it easier to learn accurate predictions.

Importance of Feature Selection

Feature engineering goes beyond simply using all available features. Sometimes, carefully selecting the most relevant features can be more effective than using all of them.

Polynomial Regression

30 April 2024 00:37

Limitations of Straight Lines

Linear regression models data using straight lines. However, some datasets may not be well-represented by straight lines.

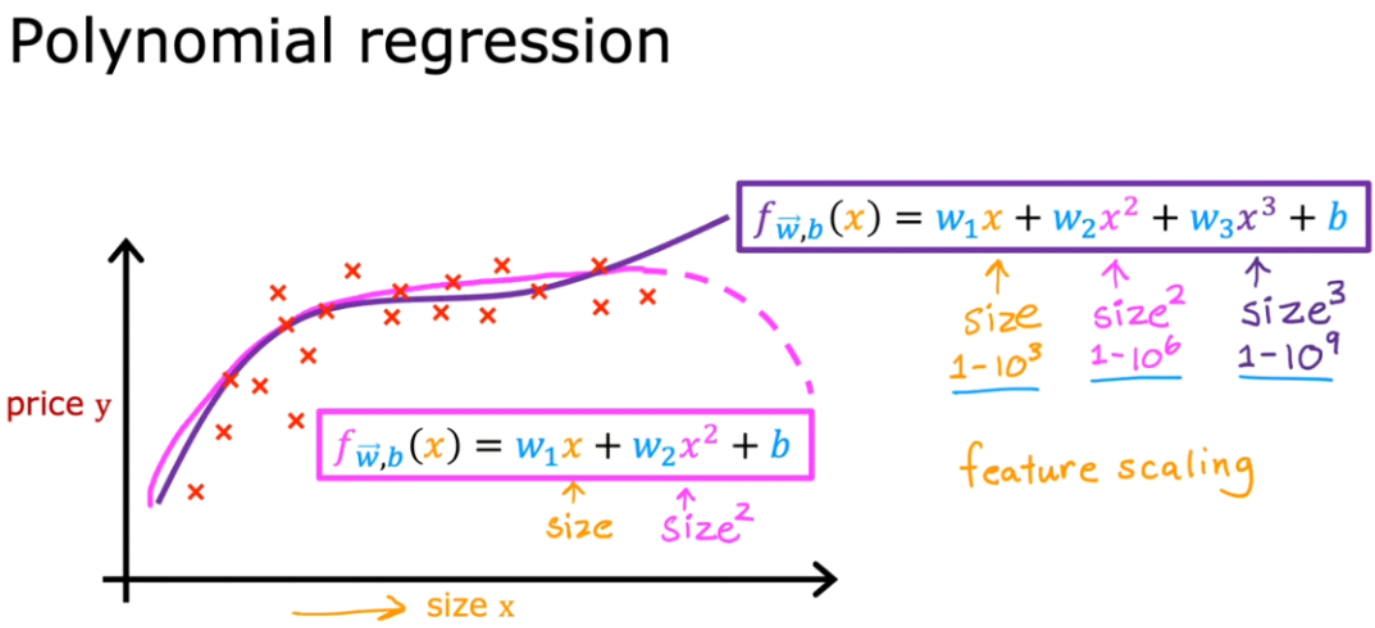
Polynomial Regression for Curve Fitting

- Involves raising existing features to different powers (e.g., x^2, x^3) to create new features.
- The new model can then include these additional features to capture non-linear relationships.

Example: Predicting House Prices

Consider a dataset where house price (y) is related to house size (x). A straight line might not accurately model this data.

- **Quadratic Model (x and x^2):** This captures a curve initially increasing but eventually decreasing. However, this might not be realistic for house prices.
- **Cubic Model (x, x^2, and x^3):** This creates a curve that increases and then keeps increasing as size increases, potentially a better fit.



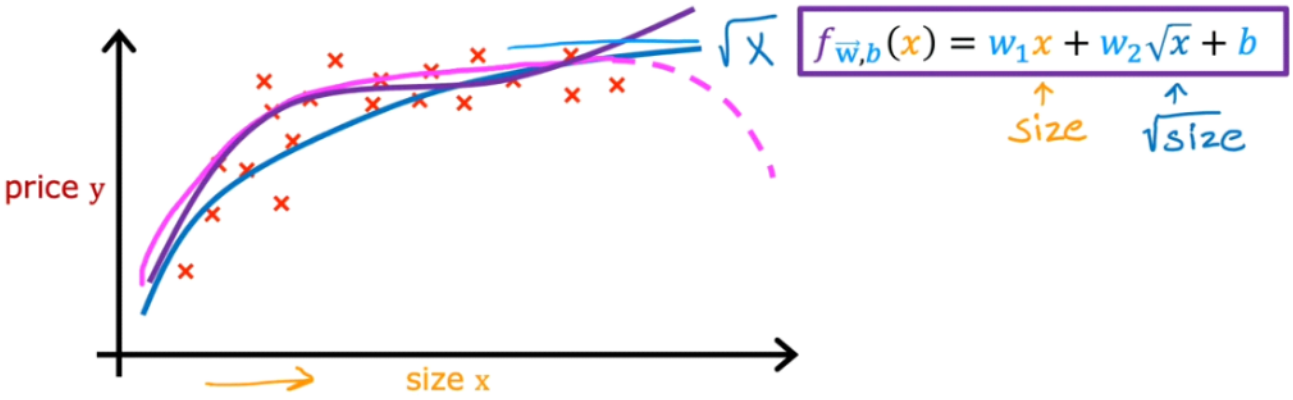
Impact of Feature Scaling

- Feature scaling becomes crucial as higher-order terms (x^2, x^3) can have significantly larger ranges than the original feature (x).
- Gradient descent requires features to be in comparable ranges for effective optimization.

Alternative Feature Choices

- Polynomial regression offers flexibility in choosing feature transformations.
- Square root (√x) is another alternative to x^2 and x^3, suitable for curves that never flatten out or decrease.

Choice of features



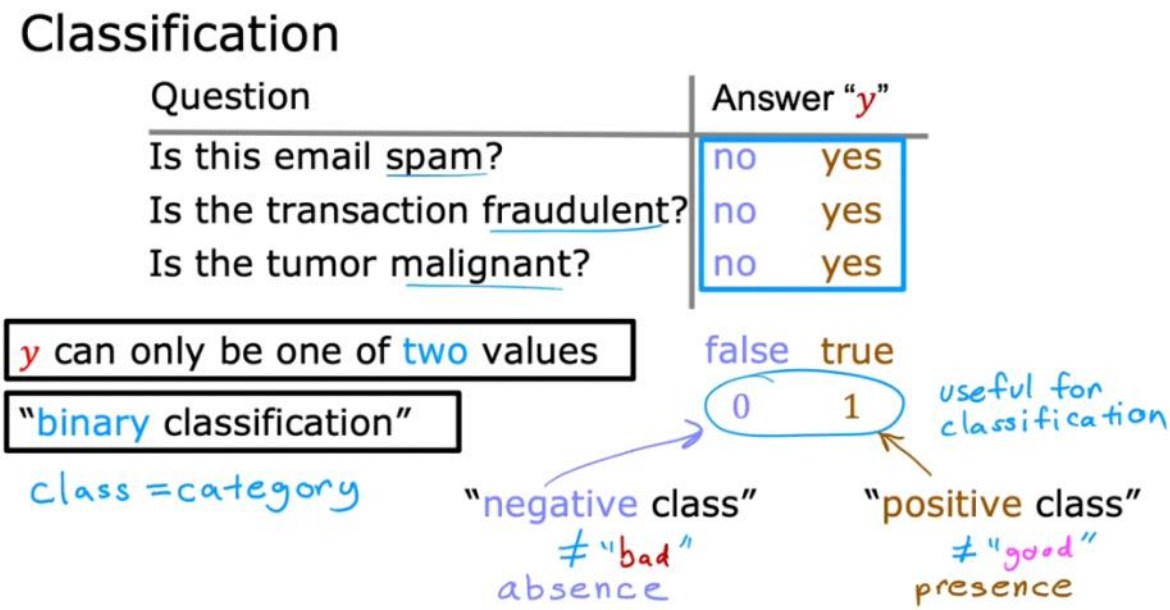
Benefits of Feature Engineering

- Feature engineering allows creating new features that can significantly improve model performance by capturing complex relati onships in the data.

Classification

04 May 2024 23:37

Classification vs. Regression

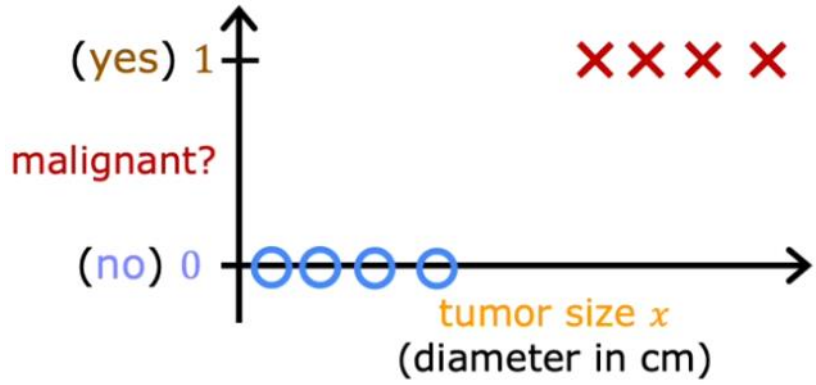
- Classification: Output variable (y) can take only a few possible values (e.g., spam/not spam, fraudulent/legitimate).
- 

- Regression: Output variable (y) can take on any number within a continuous range.

Why Linear Regression is Unsuitable for Classification

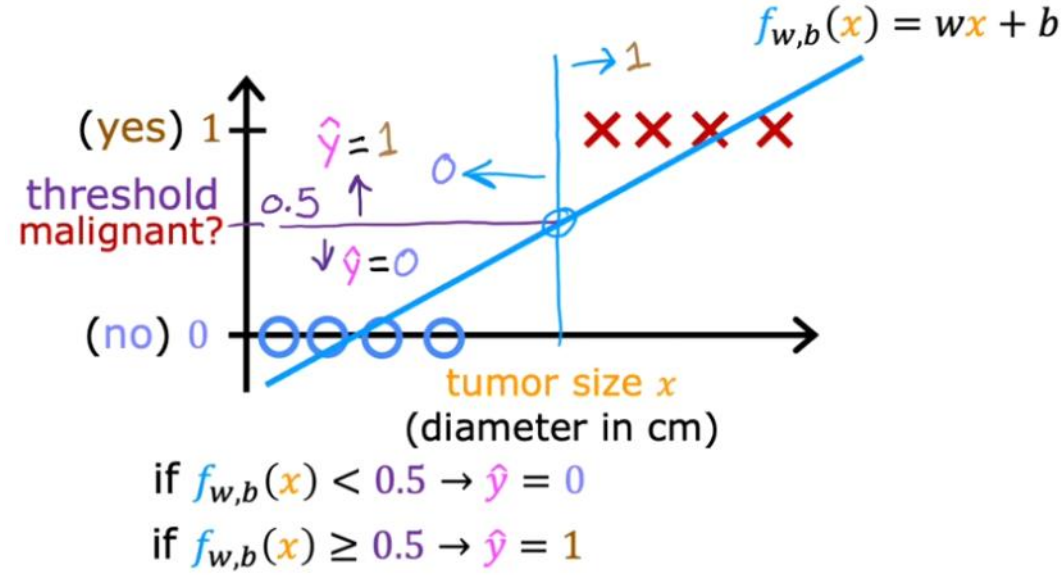
- Linear regression predicts a continuous range of values, not discrete classes (0 or 1).
- Classification requires predicting categories (e.g., spam/not spam).

Example: Malignant Tumor Classification

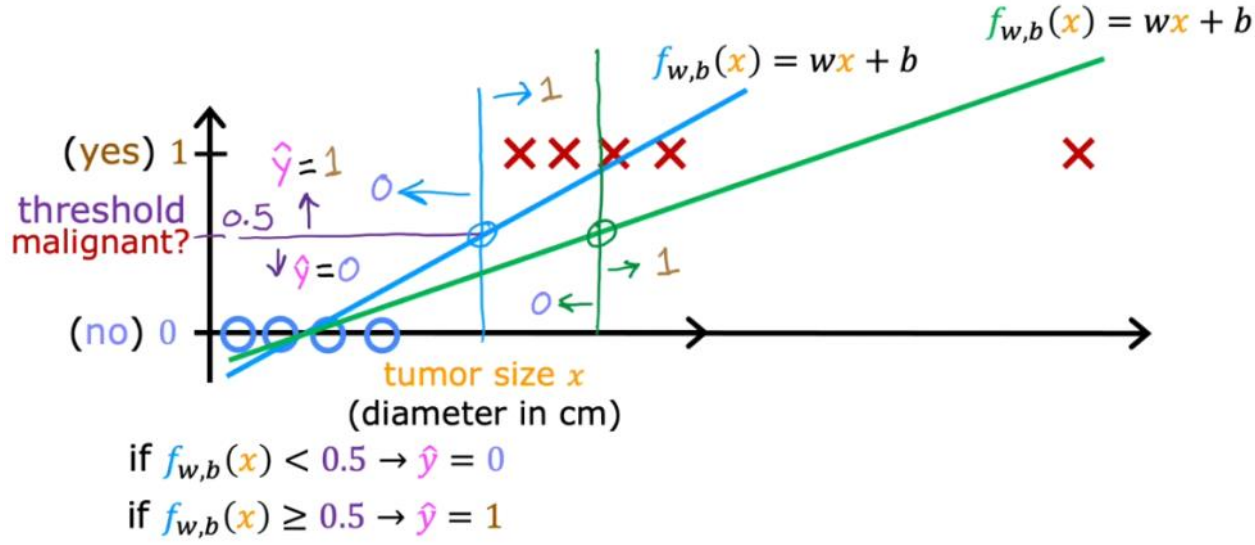
- Classify tumors as malignant (1) or benign (0) based on size.
- Fitting a straight line (linear regression) might not be ideal.
- 

Challenges with Linear Regression

- Threshold Selection: Assigning classes (0 or 1) based on a threshold on the predicted value (e.g., below 0.5 is benign).

- 

- Sensitivity to New Data: The decision boundary (classification line) can shift significantly with new data points, leading to incorrect classifications.

- 

Logistic Regression: The Solution

- Logistic regression outputs values between 0 and 1, making it suitable for classification tasks.
- The decision boundary is less sensitive to changes in data and provides a more robust classification model.

Terminology

- Negative class (0): The class assigned a value of 0 (e.g., not spam, benign).
- Positive class (1): The class assigned a value of 1 (e.g., spam, malignant).

Logistic Regression

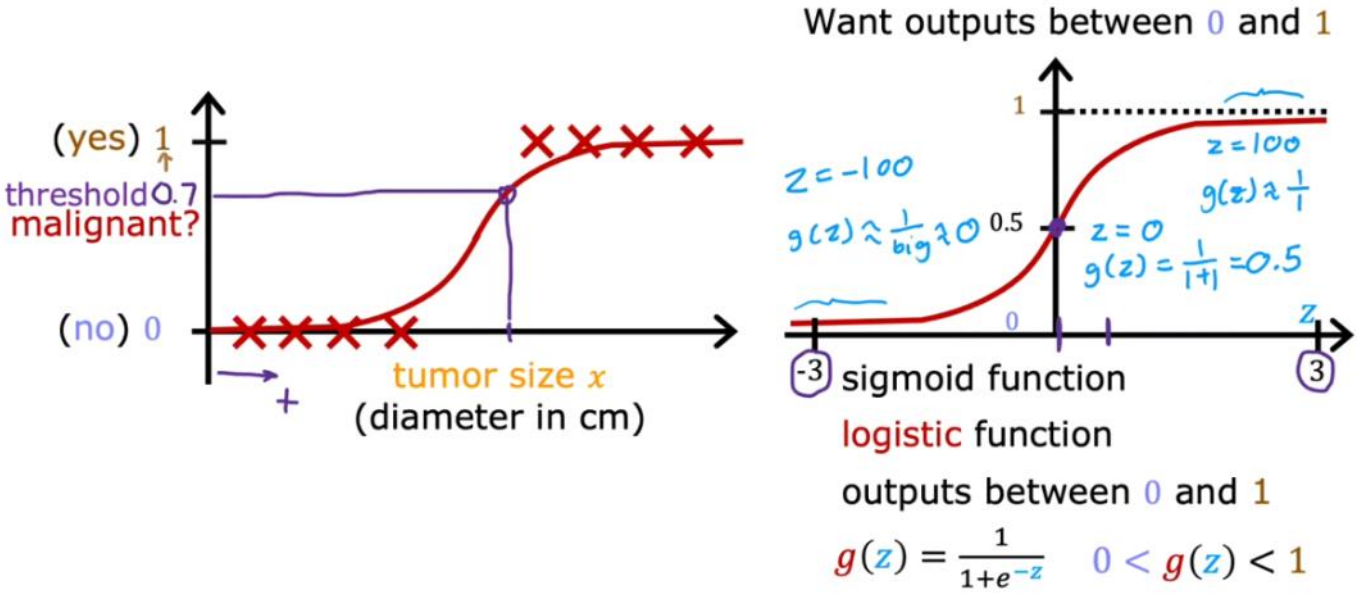
06 May 2024 15:21

Beyond Linear Regression for Classification

- Linear regression predicts continuous values, not discrete classes (0 or 1) for classification problems.
- Logistic regression addresses this limitation by using the sigmoid function to output probabilities between 0 and 1.

Sigmoid Function: The Core of Logistic Regression

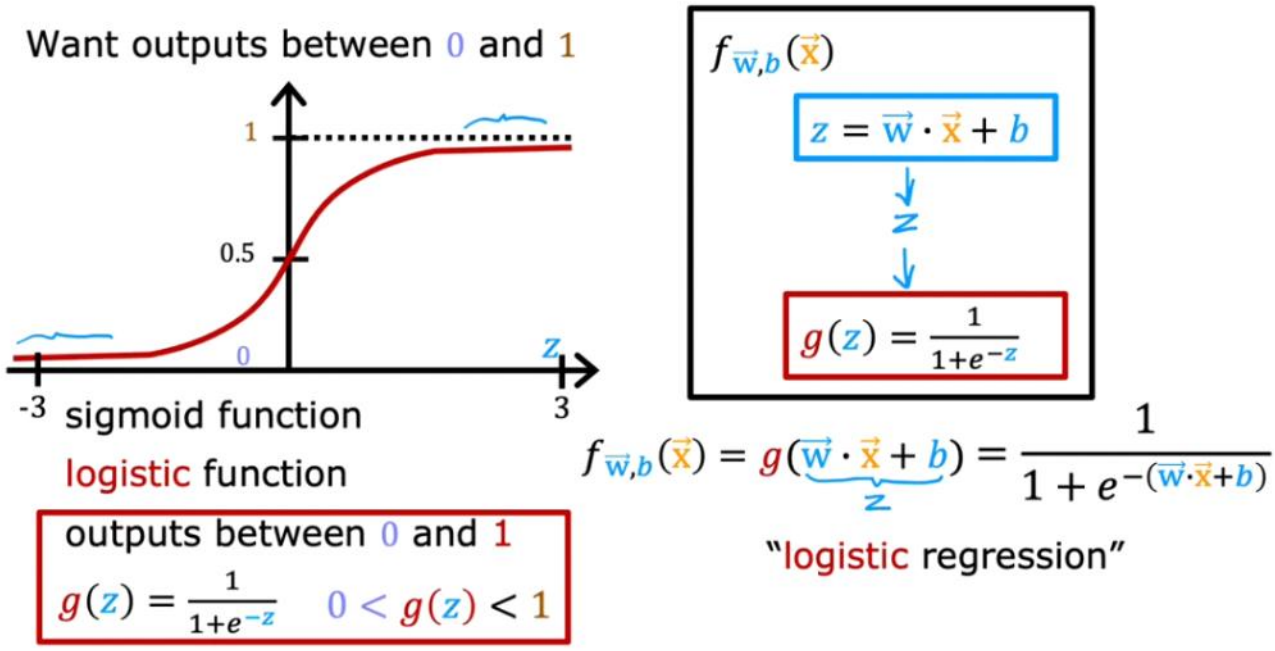
- The sigmoid function (S-shaped curve) maps input values to probabilities between 0 and 1.



- Larger input values result in probabilities closer to 1 (more likely to be class 1).
- Smaller input values result in probabilities closer to 0 (more likely to be class 0).

Logistic Regression Model

- Combines a linear function (w * x + b) with the sigmoid function (g(z)) to produce a classification model (f(x)).
  - w and b are parameters learned during the training process.
  - z is an intermediate value calculated before applying the sigmoid function.
- f(x) outputs a value between 0 and 1, representing the probability of the label y being 1 given the input x.

- 

Interpreting Logistic Regression Output

- The output (f(x)) represents the probability (between 0 and 1) that the class label (y) is 1 for a given input (x).
  - Example: f(x) = 0.7 signifies a 70% chance of y being 1 (positive class).

Logistic Regression Notation (Optional)

- f(x) = p(y = 1 | x; w, b): Probability of y being 1 given input x, with parameters w and b.
- The semicolon (;) separates the fixed input features (x) from the model parameters (w and b).

Decision Boundary

06 May 2024 19:11

Logistic Regression Model Recap

The logistic regression model outputs are computed in two steps:

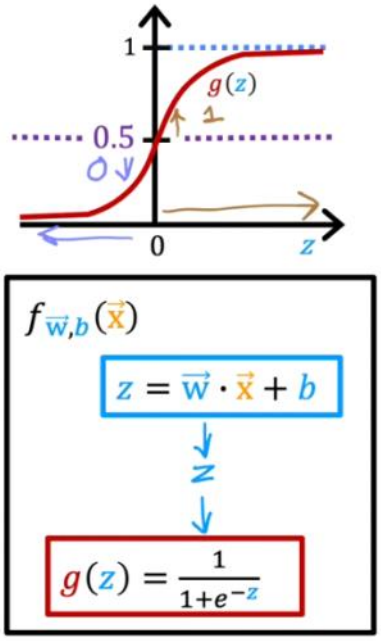
1. **Compute z:** z = w.x + b (w represents weights, x represents features, and b represents bias)
2. **Apply the sigmoid function (g) to z:** f(x) = g(z)

The sigmoid function outputs a value between 0 and 1, which can be interpreted as the probability of y being equal to 1 given x and parameters w and b.

Classifying with a Threshold

To predict whether y is 0 or 1, a threshold is often set on f(x). A common choice is 0.5:

- If f(x) >= 0.5, predict y = 1
- If f(x) < 0.5, predict y = 0



$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$$

$$= P(y = 1 | x; \vec{w}, b) \quad 0.7 \quad 0.3$$

0 or 1? threshold

Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

$g(z) \geq 0.5$

$z \geq 0$

$\vec{w} \cdot \vec{x} + b \geq 0$ $\vec{w} \cdot \vec{x} + b < 0$

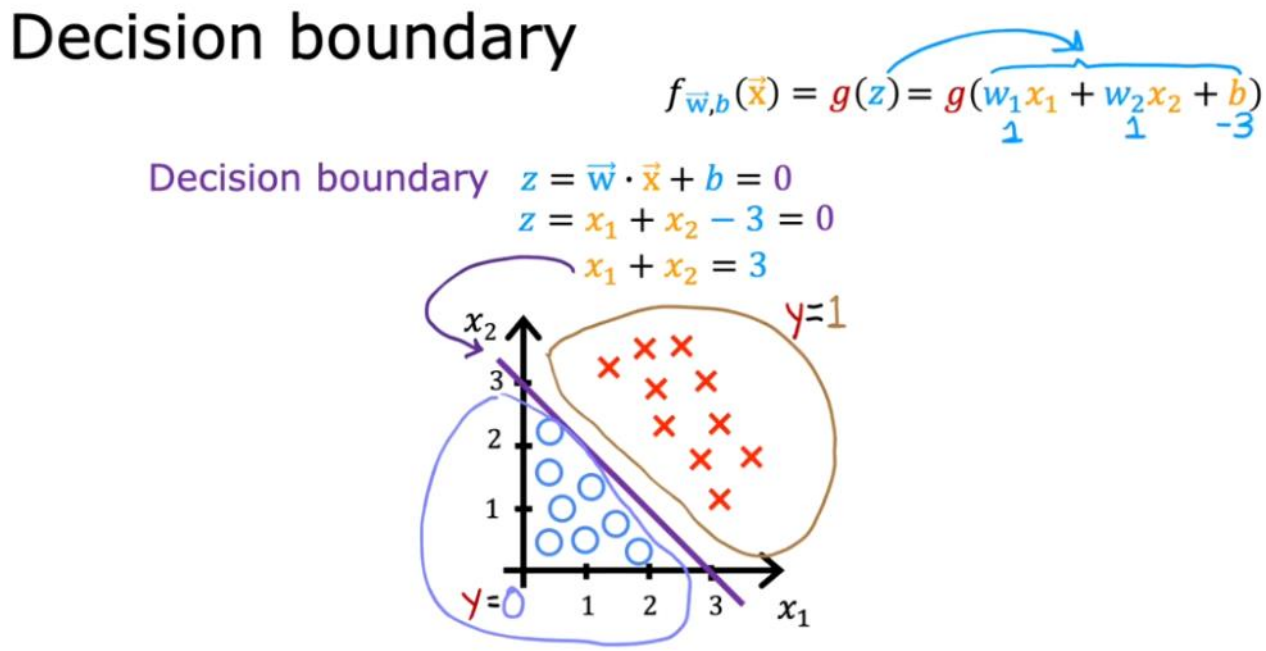
$\hat{y} = 1$ $\hat{y} = 0$

Decision Boundary

The decision boundary is the line where f(x) is equal to the threshold (usually 0.5). This line separates the data points into two regions where the model predicts different classes.

Visualizing the Decision Boundary in 2D

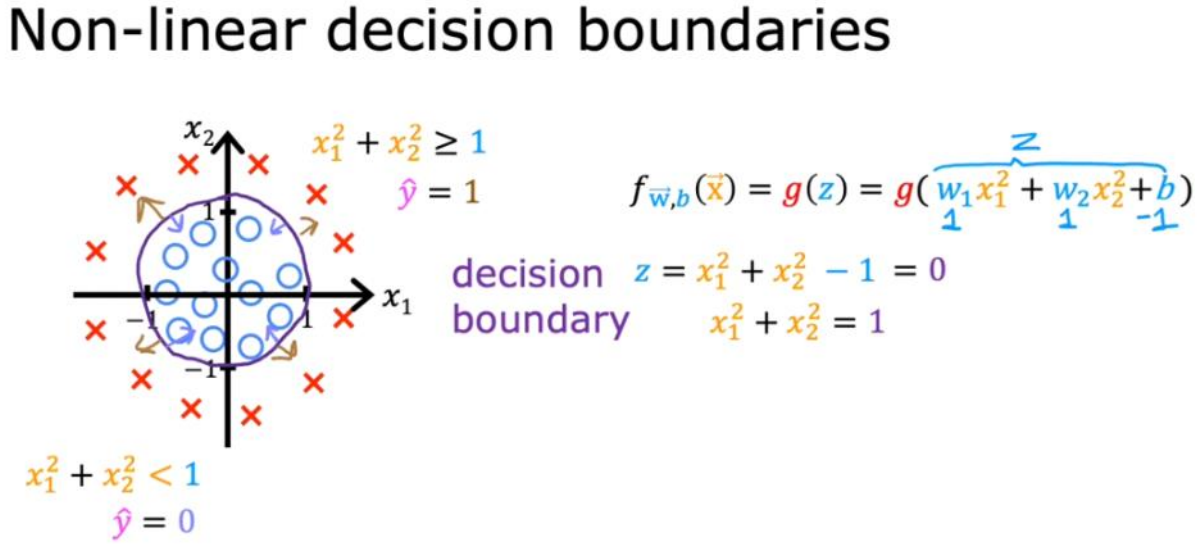
Consider a classification problem with two features, x1 and x2. The decision boundary can be visualized as a line that separates the positive examples (red crosses) from the negative examples (blue circles). The equation of the decision boundary depends on the values of the weights (w1 and w2) and the bias (b).



For example, with w1 = 1, w2 = 1, and b = -3, the decision boundary becomes the line x1 + x2 = 3. Points to the right of the line are predicted as class 1, and points to the left are predicted as class 0.

Decision Boundaries with Polynomial Features

Logistic regression can also handle more complex decision boundaries by using polynomial features. These features are created by raising the existing features to higher powers (e.g., x1^2, x2^2).

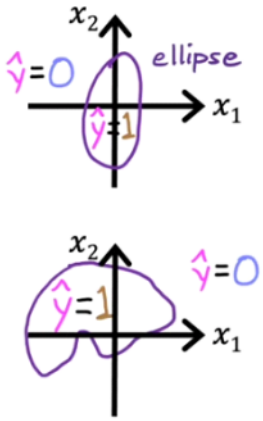


Using polynomial features, the decision boundary can become a curve or even a more intricate shape, depending on the chosen weights and bias.

Higher-Order Polynomials for Even More Complex Boundaries

By including even higher-order polynomial terms, logistic regression can learn even more complex decision boundaries. This allows the model to fit more intricate data patterns.

Non-linear decision boundaries



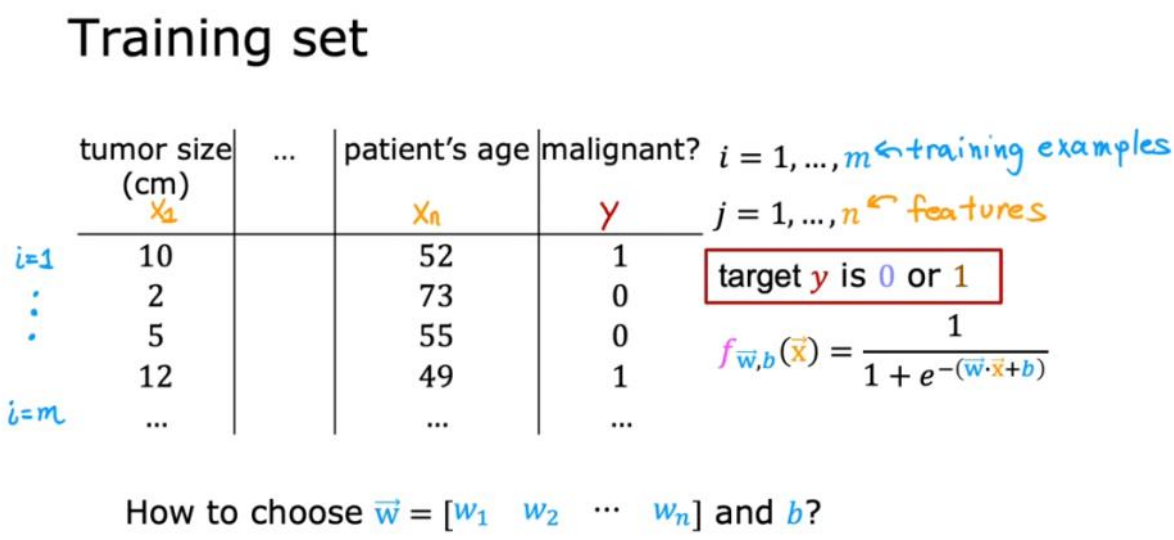
$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2 + w_5x_2^2 + w_6x_1^3 + \cdots + b)$$

Cost Function for Logistic Regression

07 May 2024 08:36

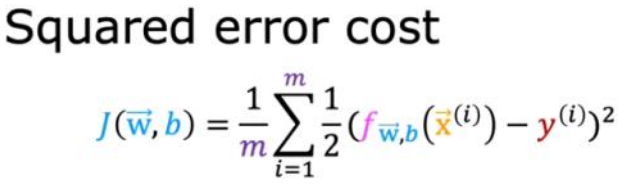
Squared Error Not Ideal for Logistic Regression

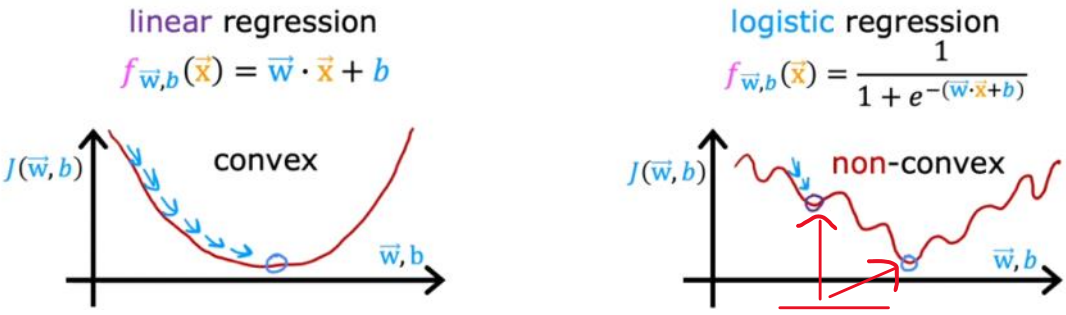
- This is what a dataset for logistic regression looks like:



How would you determine its parameters w and b?

- The squared error cost function, suitable for linear regression, is not ideal for logistic regression.
- Logistic regression outputs probabilities between 0 and 1, leading to a non-convex cost function with multiple local minima.

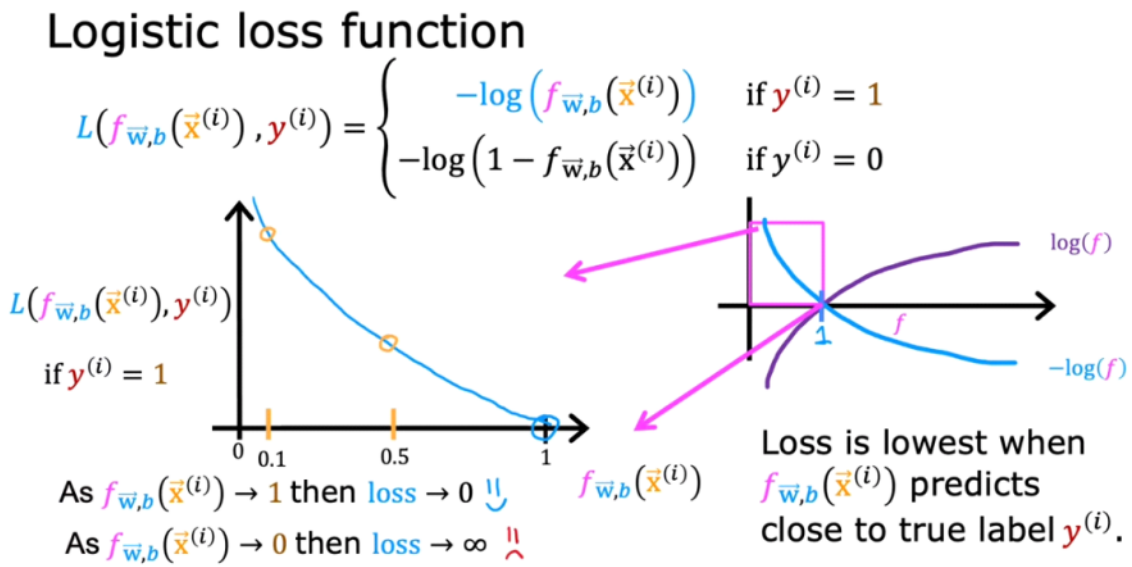


- 

- Gradient descent struggles with non-convex cost functions, potentially **getting stuck in local minima.**

A New Cost Function for Logistic Regression

- We introduce a different cost function specifically designed for logistic regression.



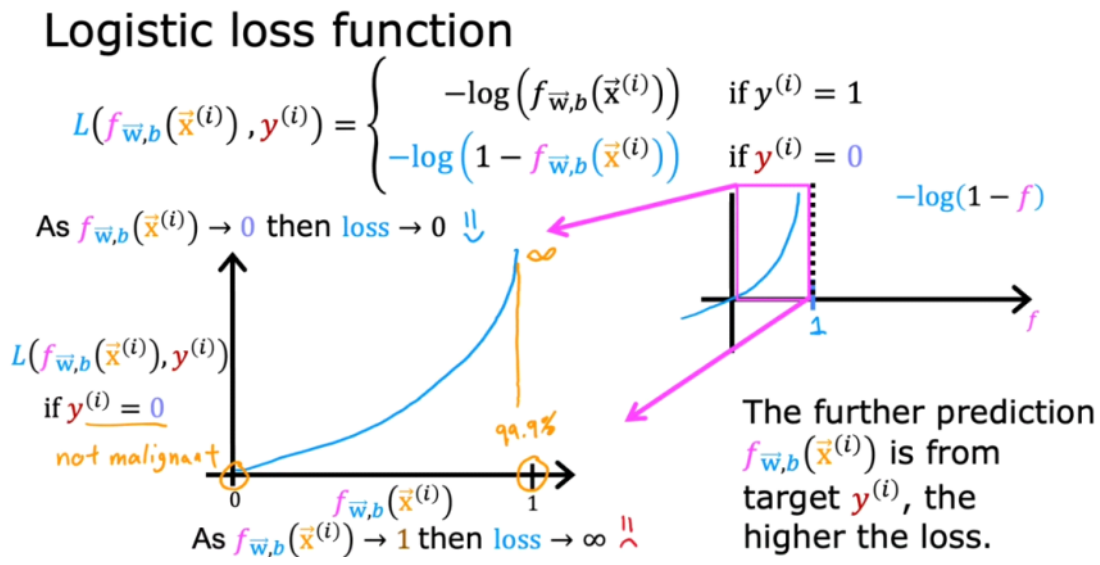
- This cost function utilizes a loss function to measure performance on a single training example.
  - The loss function considers the true label (y) and the model's prediction (f(x)).

Logistic Regression Loss Function

- The loss function depends on the true label (y):
  - y = 1 (positive class): Loss is negative log(f(x)) (penalizes low predicted probabilities).
  - y = 0 (negative class): Loss is negative log(1 - f(x)) (penalizes high predicted probabilities)
- This loss function incentivizes the model to make predictions closer to the true labels.

Visualizing the Loss Function

- When y = 1 (positive class):
  - Loss is very low when f(x) is close to 1 (correct prediction).
  - Loss increases as f(x) deviates from 1 (incorrect predictions).
- When y = 0 (negative class):
  - Loss is very low when f(x) is close to 0 (correct prediction).
  - Loss increases as f(x) approaches 1 (incorrect predictions).



Overall Cost Function

- The total cost function is the average of the loss function across all training examples.
  - Minimizing this cost function leads to better model parameters (w and b).
- This choice of loss function ensures a convex cost function.
  - Gradient descent can reliably find the global minimum, leading to optimal parameters.

Cost

- $$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)$$ (cost; loss)

$$= \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

convex → can reach a global minimum

find w, b that minimize cost J

Simplified Loss Function for Logistic Regression

Simplifying the Loss Function

- We can exploit the binary nature of y (0 or 1) to simplify the loss function.
- The new form combines both cases (y = 1 and y = 0) into a single expression:
  - Loss = -y * log(f(x)) - (1 - y) * log(1 - f(x))

Equivalence of Loss Functions

- Simplified loss function

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

- When y = 1, the second term becomes zero, and the equation reduces to -log(f(x)).

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

($y^{(i)} = 1$; $(1 - y^{(i)}) = (1 - 1) = 0$)

if $y^{(i)} = 1$:

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -1 \log\left(f(\vec{x})\right)$$

- When y = 0, the first term becomes zero, and the equation reduces to -log(1 - f(x)).

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

$y^{(i)} \to 0$

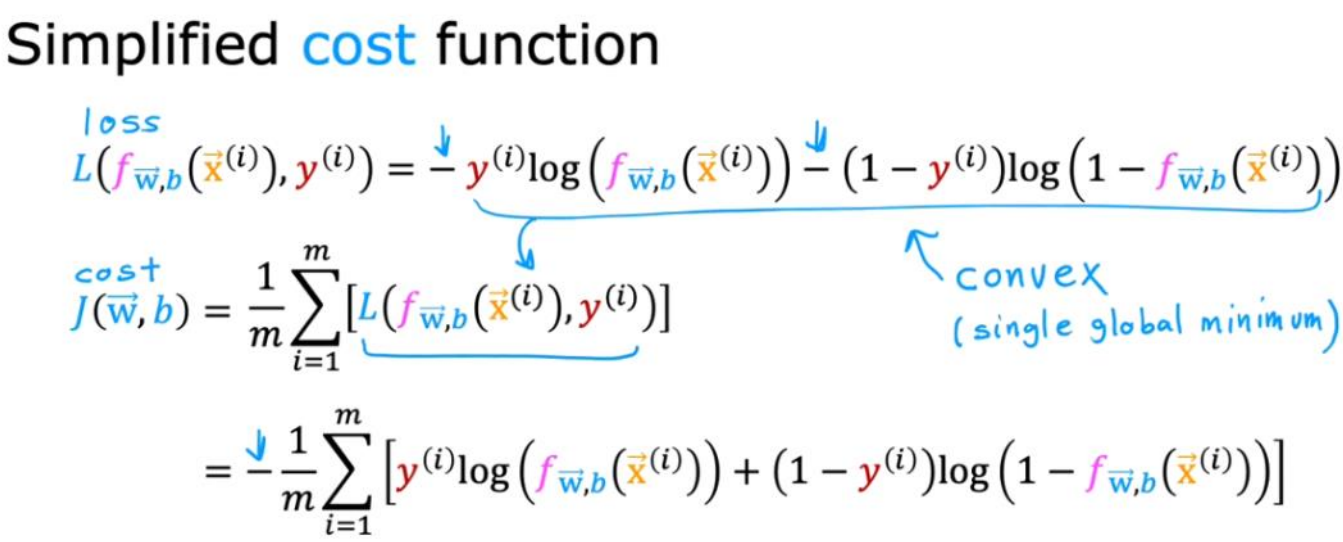
if $y^{(i)} = 0$:

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -(1 - 0)\log(1 - f(\vec{x}))$$

Cost Function with Simplified Loss

- The cost function (J) is the average loss across all m training examples:

    - J = (1/m) * sum(loss_i), where i goes from 1 to m (number of examples)
- We substitute the simplified loss function into the cost function:
    - J = (1/m) * sum(-y_i * log(f(x_i)) - (1 - y_i) * log(1 - f(x_i)))
- After rearranging terms, we arrive at the commonly used cost function for logistic regression.



Justification for the Cost Function

- This specific cost function is derived from a statistical principle called maximum likelihood estimation.
- It aims to efficiently find optimal parameters for the model.
- Convexity: An important property of this cost function (covered in previous video).

Gradient Descent for Logistic Regression

07 May 2024 09:41

Goal: Minimizing the Cost Function

- We aim to find w and b that minimize the cost function J(w, b).
- Gradient descent is a common optimization technique for this purpose.

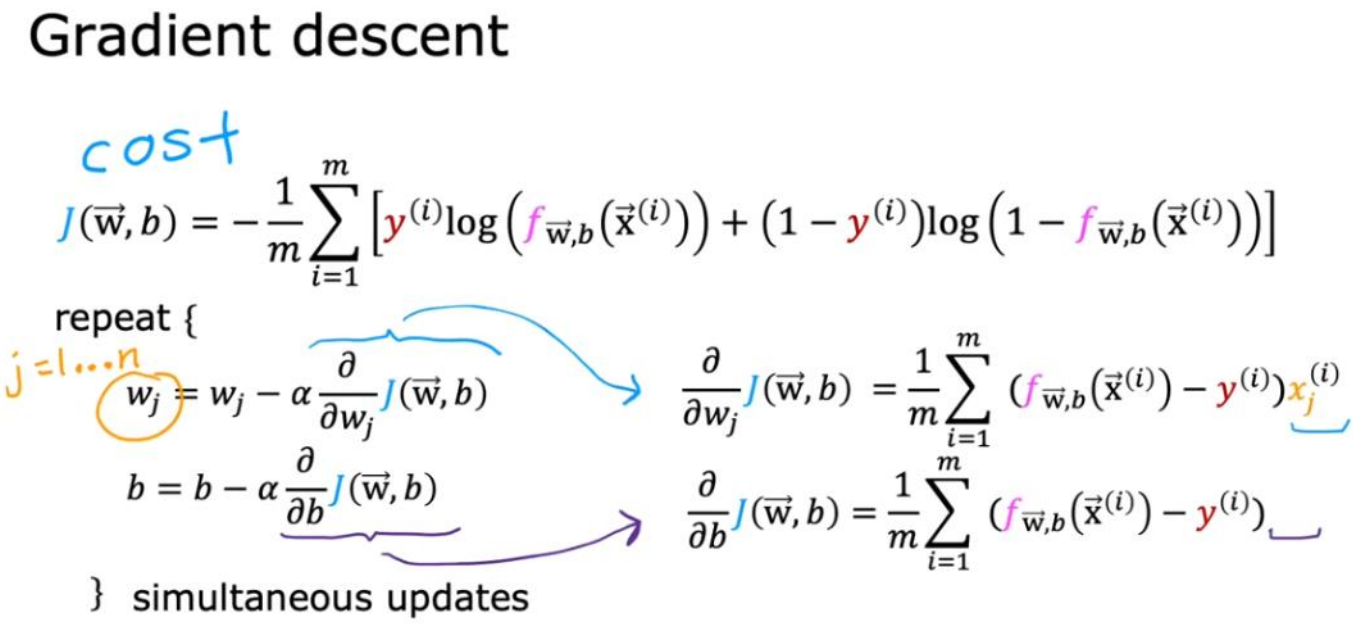
Logistic Regression Prediction

After finding optimal parameters, the model can predict the probability (y = 1) for new data points (e.g., patients).

Gradient Descent Algorithm

Here's the gradient descent algorithm for updating parameters:

- w_j = w_j - α * ∂J/∂w_j (update w_j)
- b = b - α * ∂J/∂b (update b)
- α: learning rate
- ∂J/∂w_j, ∂J/∂b: derivatives of J w.r.t. w_j and b



Derivatives of the Cost Function

- ∂J/∂w_j = (1/m) * Σ(f(x^(i)) - y^(i)) * x_j^(i) (sum over training examples)
  - f(x^(i)): predicted probability for training example i
  - y^(i): true label for training example i
  - x_j^(i): j-th feature value in training example I

- ∂J/∂b = (1/m) * Σ(f(x^(i)) - y^(i)) (similar to ∂J/∂w_j but without x_j^(i))

Simultaneous Updates

We perform simultaneous updates for all parameters (w_j and b) at each iteration.

Comparison with Linear Regression

The update equations resemble linear regression, but the key difference lies in f(x).

- Logistic regression: f(x) is the sigmoid function applied to (wx + b).
- Linear regression: f(x) is simply (wx + b).

Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right) \right]$$

} simultaneous updates

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

Linear regression $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Logistic regression $f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$

Overfitting and underfitting problem

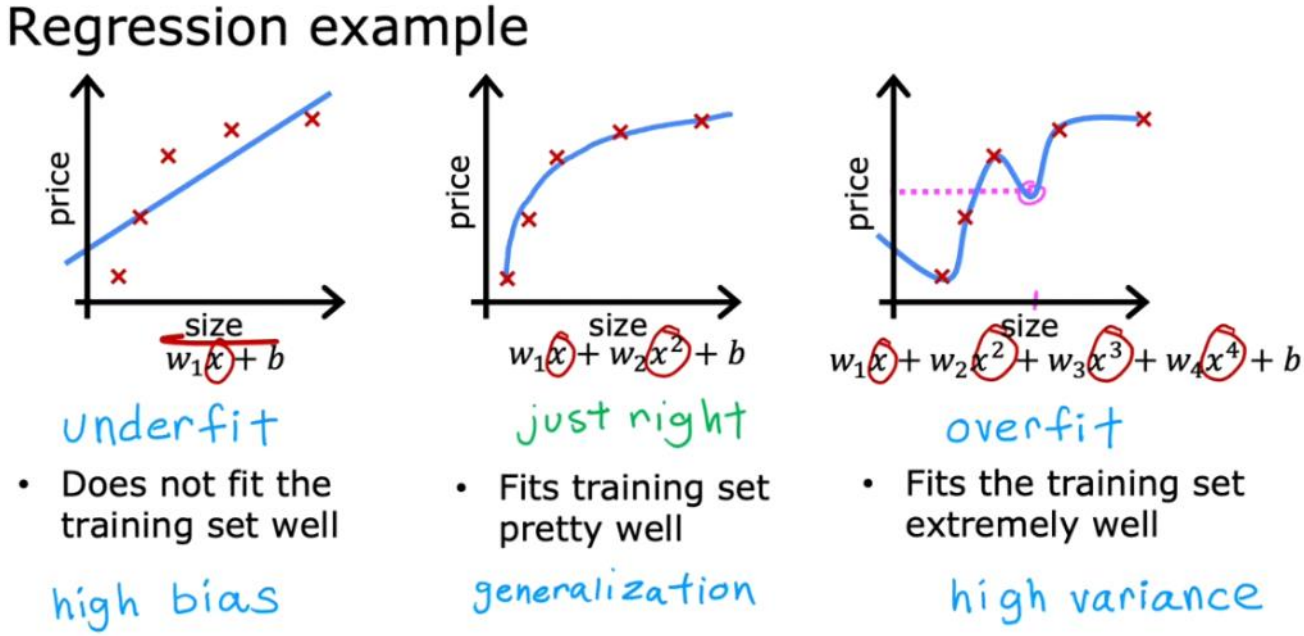
07 May 2024 12:48

Overfitting vs. Underfitting

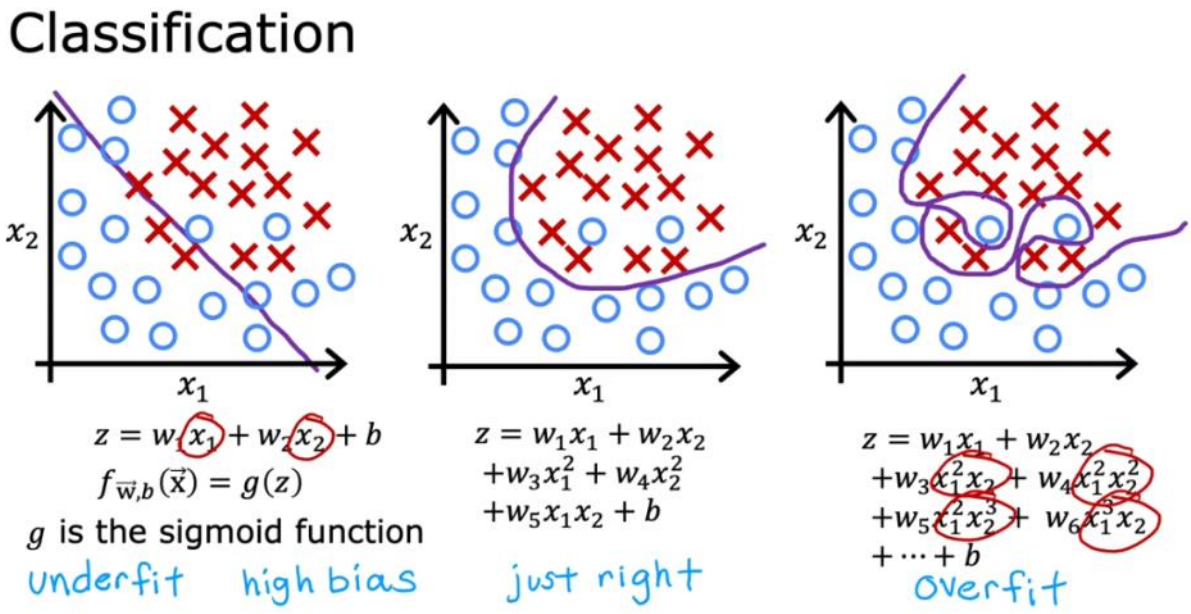
- **Overfitting:** A model fits the training data too closely, capturing noise or irrelevant details.
  - Leads to poor performance on unseen data (generalization).
  - High variance.

- **Underfitting:** A model is too simple and cannot capture the underlying pattern in the data.
  - High bias.

Example: Predicting Housing Prices

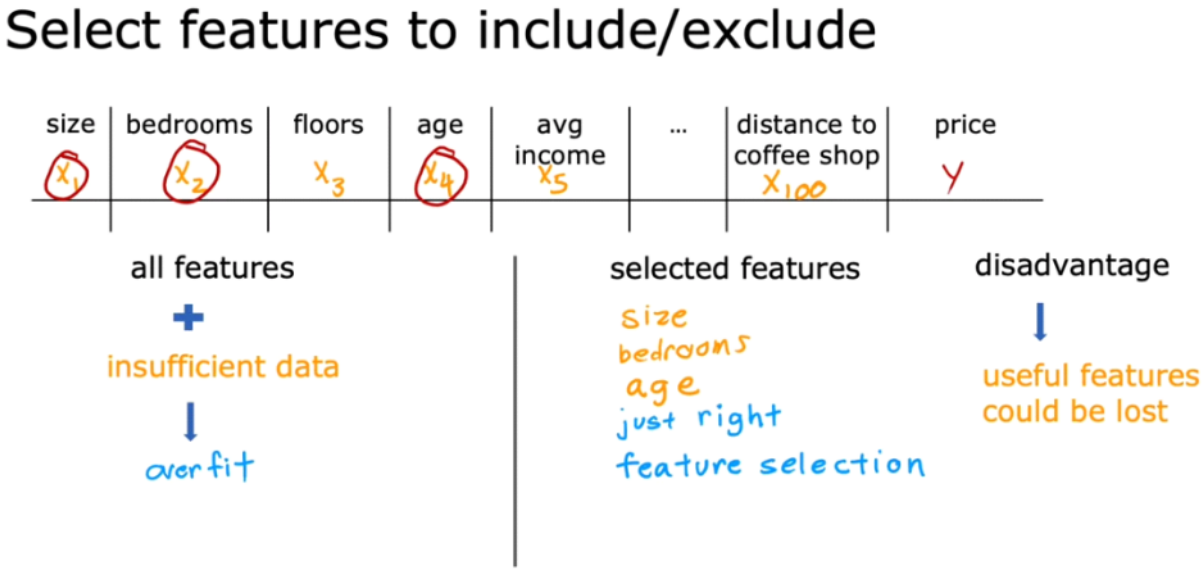
- We want to predict house prices based on size (linear regression).
- 

- **Underfitting (High Bias):** A straight line might not capture the non-linear price increase with size.
- **Overfitting (High Variance):** A high-order polynomial might fit the training data perfectly but become erratic for unseen sizes.
- **Just Right:** A moderate-complexity model (e.g., quadratic) might capture the trend without overfitting.



Techniques to Address Overfitting

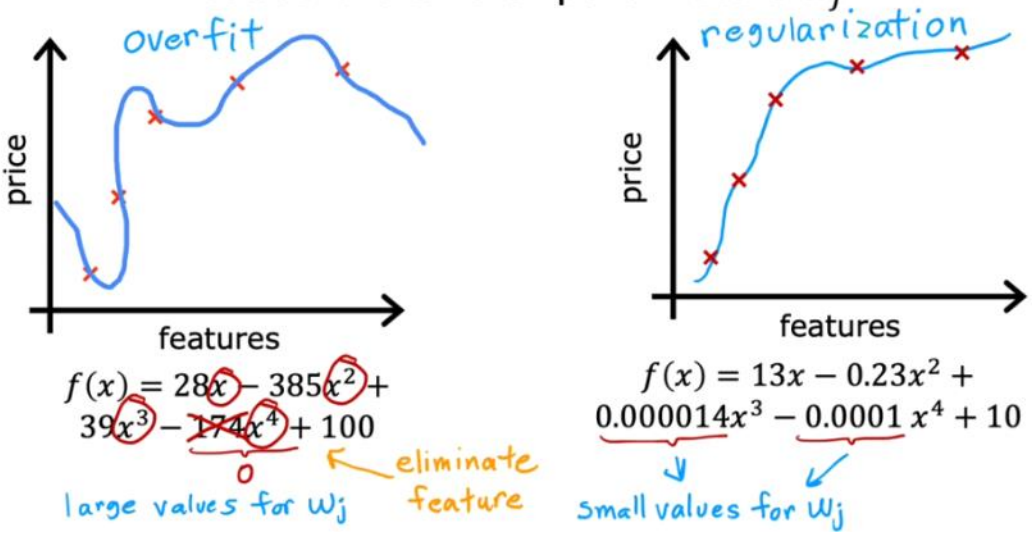
1. **Collect More Training Data**
    - Having more data allows the model to learn a less "wiggly" function that generalizes better.
    - 

    - This approach might not always be feasible (e.g., limited data availability).
2. **Feature Selection**
    - Reduce overfitting by using a subset of the most relevant features.
    - 

    - Intuition-based selection: Choose features believed to be most influential for prediction.
    - Feature selection algorithms (covered later) can automate this process.
    - Disadvantage: Throws away potentially useful information by discarding features.
3. **Regularization**
    - Encourages the learning algorithm to reduce the impact of parameters (weights) without eliminating them completely.

Regularization

Reduce the size of parameters w_j

- 

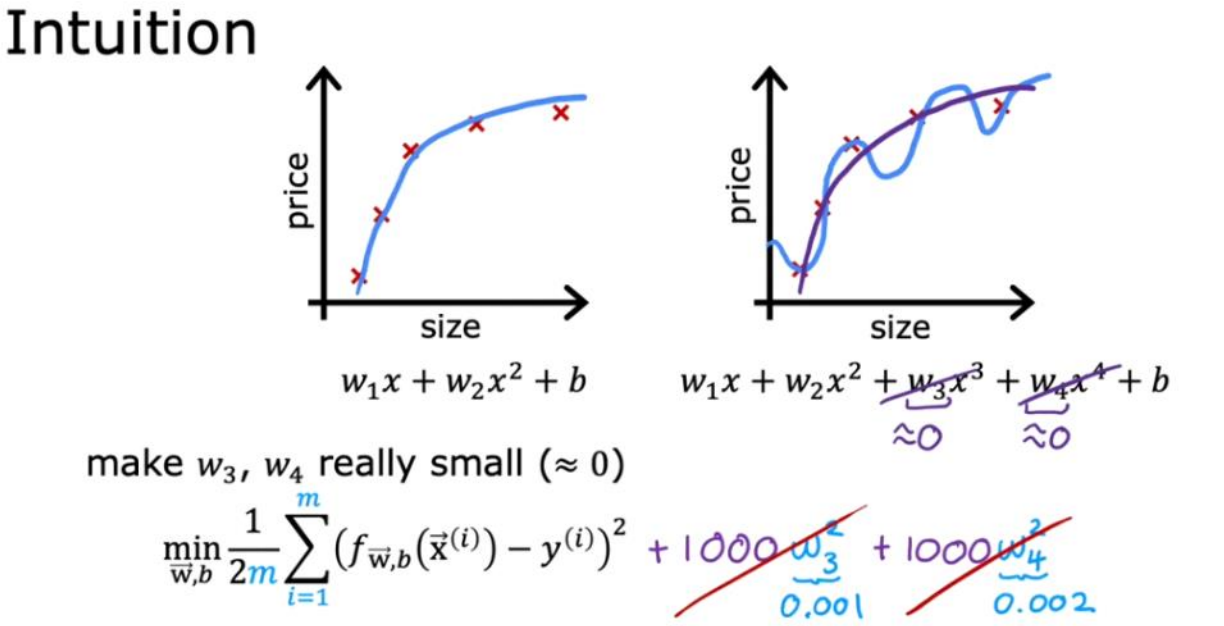
  - Allows keeping all features while preventing them from having an overly significant effect, which can cause overfitting.
  - Commonly used technique, especially for training neural networks (covered later).

Regularization

08 May 2024 00:08

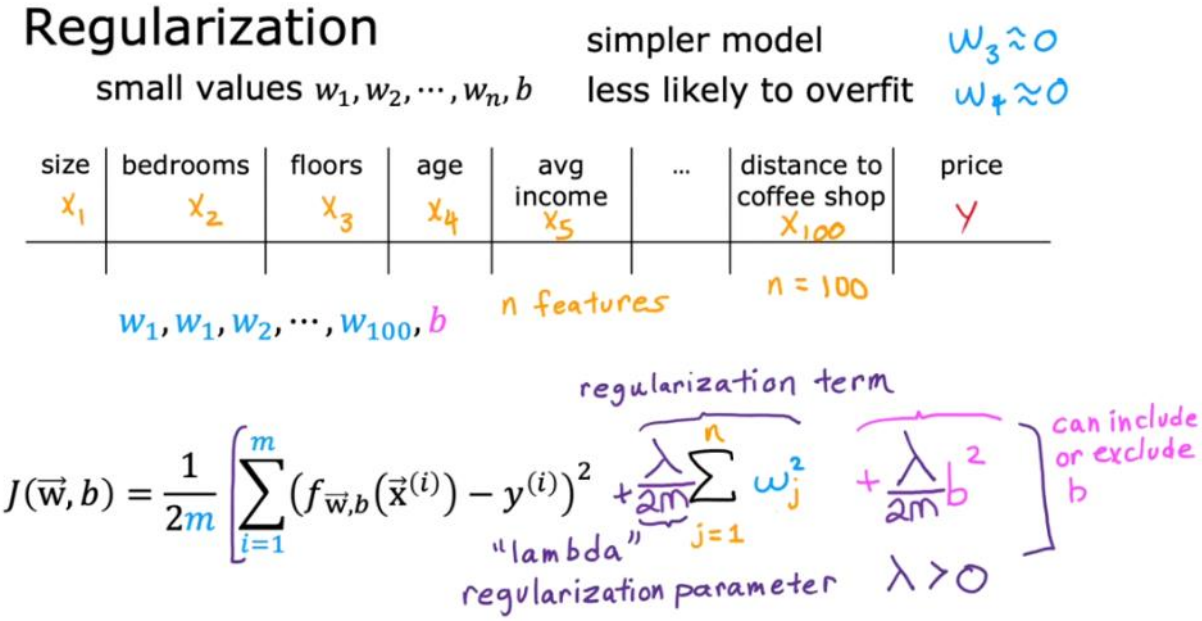
Regularization: The Core Idea

- Regularization aims to make the parameter values (weights denoted by W1 to Wn) smaller to reduce overfitting.
- Smaller parameter values lead to a simpler model, similar to having fewer features, making it less prone to overfitting.



Modifying the Cost Function

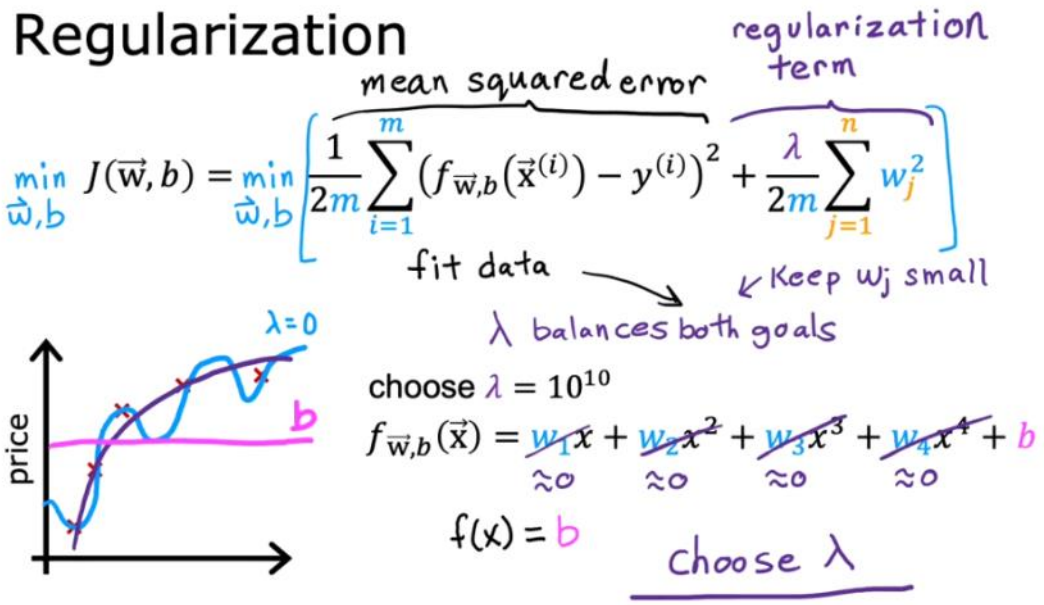
- We can modify the cost function (used for linear regression in this example) to penalize models with large parameter values.
- Consider a cost function with an additional term: lambda * sum(W_j^2) (lambda is a regularization parameter).



- Minimizing this modified cost function penalizes models with large W_j values (because the squared term becomes large).
- This effectively reduces the influence of features associated with large W_j values, potentially reducing overfitting.

Impact of the Regularization Parameter (Lambda)

- **Lambda = 0:** No regularization, potentially leading to overfitting (highly wiggly function).
- **Lambda = very large:** Heavy emphasis on the regularization term, forcing all Wj values close to 0 (underfitting - horizontal line).



- The ideal lambda value balances minimizing the mean squared error (fitting the training data) and keeping the parameters small (reducing overfitting).

Regularized Cost Function

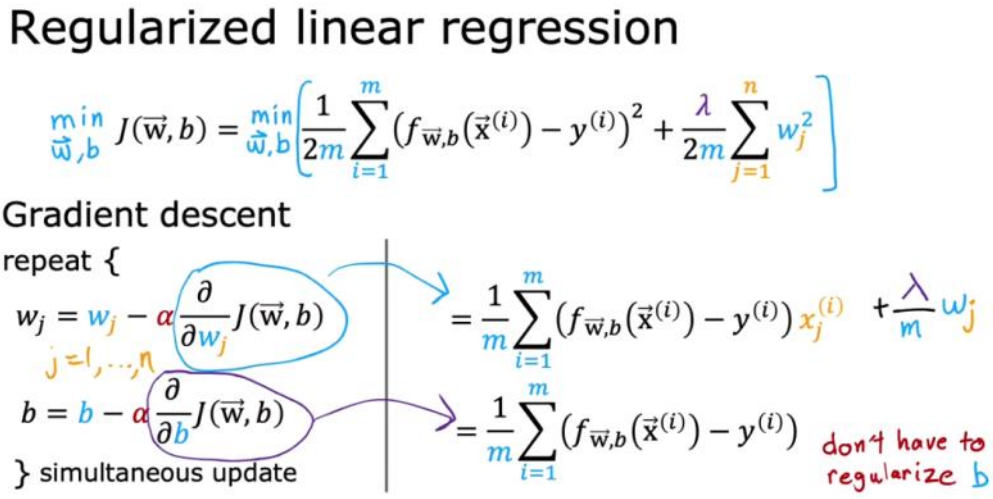
$J(w, b) = (1/2m) * \text{sum}(h(x^{(i)}) - y^{(i)})^2 + (\text{lambda}/m) * \text{sum}(w_j^2)$

Here,

- J(w, b) is the cost function.
- w and b are the parameters (weights and bias) to be optimized.
- m is the number of training examples.
- $h(x^{(i)})$ is the predicted value for the i^{th} training example.
- $y^{(i)}$ is the actual value for the i^{th} training example.
- lambda is the regularization parameter controlling the trade-off between fitting the data and keeping the parameters small.

Gradient Descent Updates

The standard gradient descent updates are modified slightly for regularized linear regression:



Here, alpha is the learning rate. The key difference is the additional term (alpha * lambda/m) * w_j in the w_j update, which shrinks the weight values during each iteration.

Impact of Regularization Parameter (Lambda)

- A small lambda value has a weaker regularization effect, similar to unregularized linear regression.
- A large lambda value shrinks the weights more aggressively, potentially leading to underfitting.

Intuition Behind Regularization

- We can rewrite the w_j update as: $w_j := w_j * (1 - \text{alpha} * \text{lambda}/m) - (\text{alpha}/m) * \text{sum}(h(x^{(i)}) - y^{(i)}) * x_j^{(i)}$
- The first term $w_j * (1 - \text{alpha} * \text{lambda}/m)$ shrinks w_j by a factor slightly less than 1 in each iteration (due to a small lambda and alpha)
- This shrinkage discourages large parameter values, leading to a simpler model less prone to overfitting.

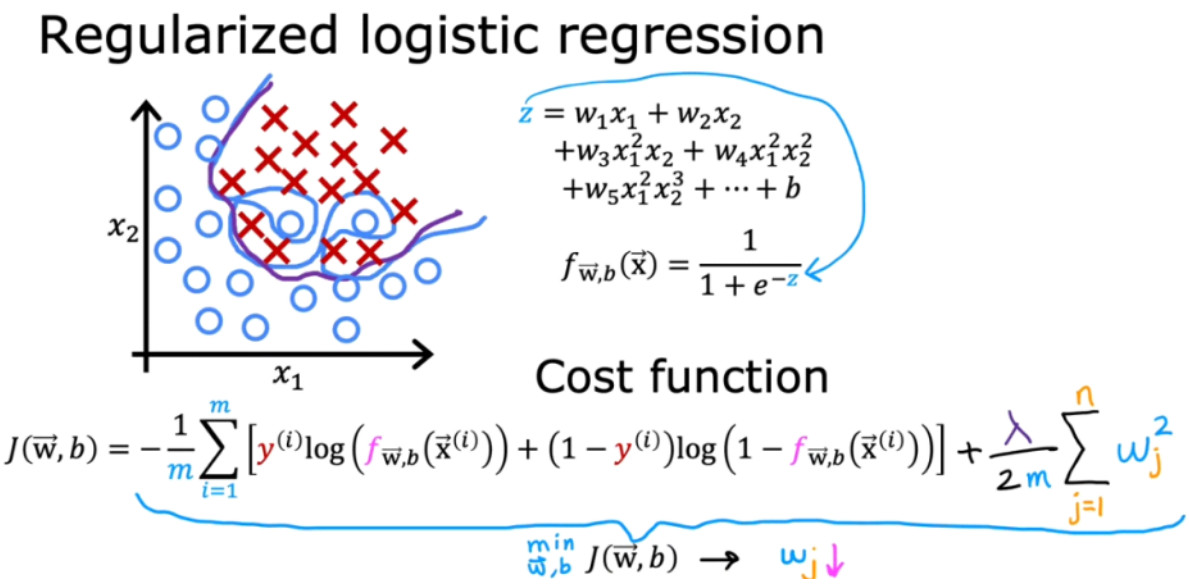
Regularization for Logistic Regression

Logistic Regression and Overfitting

Logistic regression can be prone to overfitting, especially when using high-order polynomial features. This leads to a complex decision boundary that may not generalize well to unseen data.

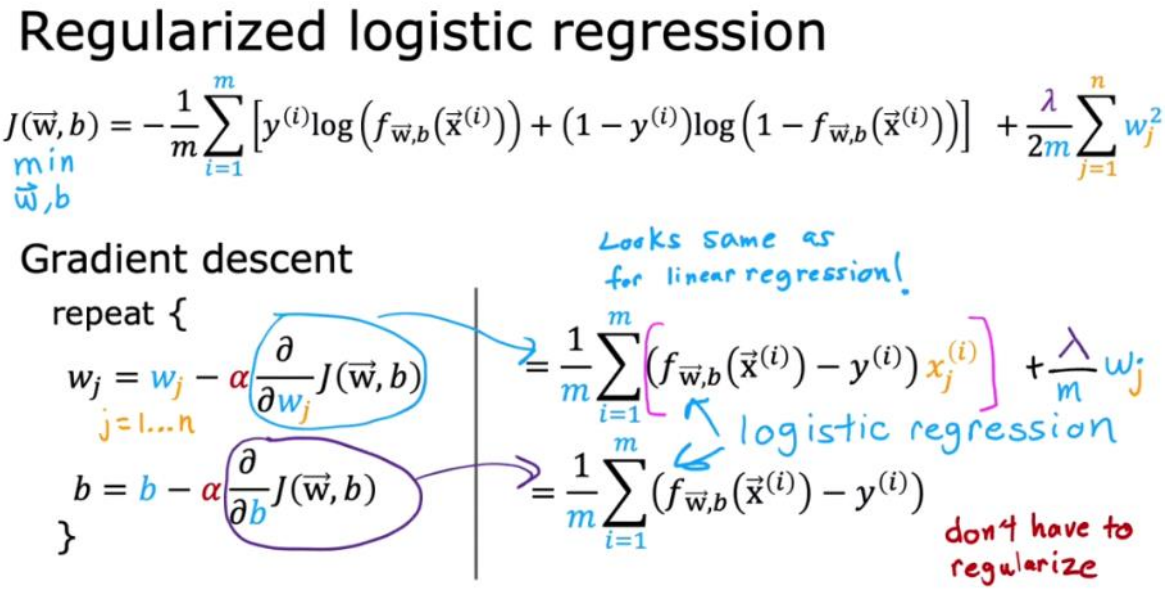
Regularization for Logistic Regression

Similar to linear regression, regularization can be applied to logistic regression by modifying the cost function:



Gradient Descent Updates

The gradient descent updates are modified slightly for regularized logistic regression:



The key difference is the additional term (alpha * lambda/m) * w_j in the w_j update, which shrinks the weight values during each iteration. This helps prevent the weights from becoming too large and reduces overfitting.

Impact of Regularization

Regularization with a larger lambda value shrinks the weights more aggressively, leading to a simpler decision boundary that may generalize better to unseen data.