mmiimran/**machine-learning-with-scikit-...**

0 Contributors | 0 Issues | 0 Stars | 0 Forks

What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can learn from data.

Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.

—Arthur Samuel,1959

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

—Tom Mitchell, 1997

- **Imagine we're talking about a spam filter, like the one in your email.**
- **This filter is like a smart program that learns from examples of spam and non-spam emails given by users.**
- **Each example it learns from is called a "training instance."**
- **The filter uses this learning to make predictions about whether new emails are spam or not.**

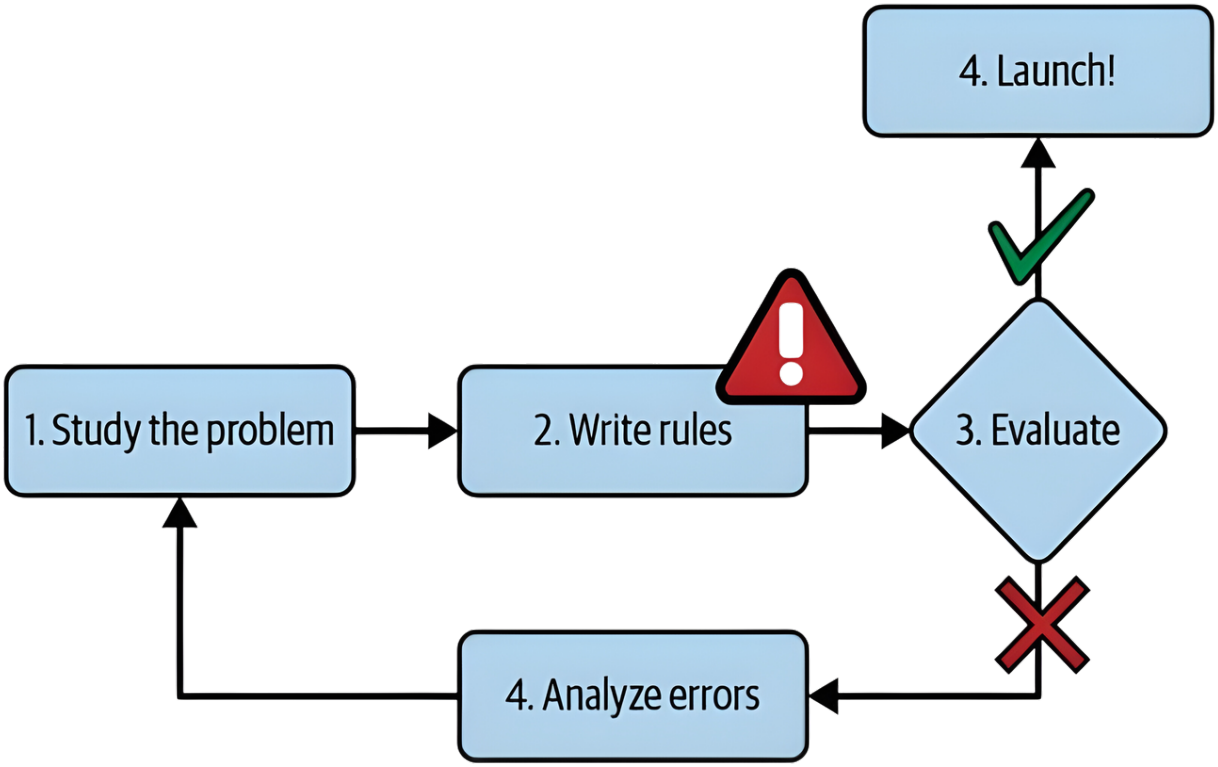
- **It uses different techniques like neural networks or random forests to make these predictions.**
- **The main job of the filter is to recognize and flag spam emails.**
- **The data it learns from is called the "training data," which is like its experience.**
- **We also need a way to measure how well the filter is doing, which we call "performance." We often use accuracy to measure how many emails it correctly classifies.**
- **It's important to understand that simply having a lot of data, like all of Wikipedia, doesn't automatically make a program smart. Without the right learning techniques, it's not really "machine learning."**

Why Use Machine Learning?

Two approaches to spam filter creation: traditional programming and machine learning techniques.

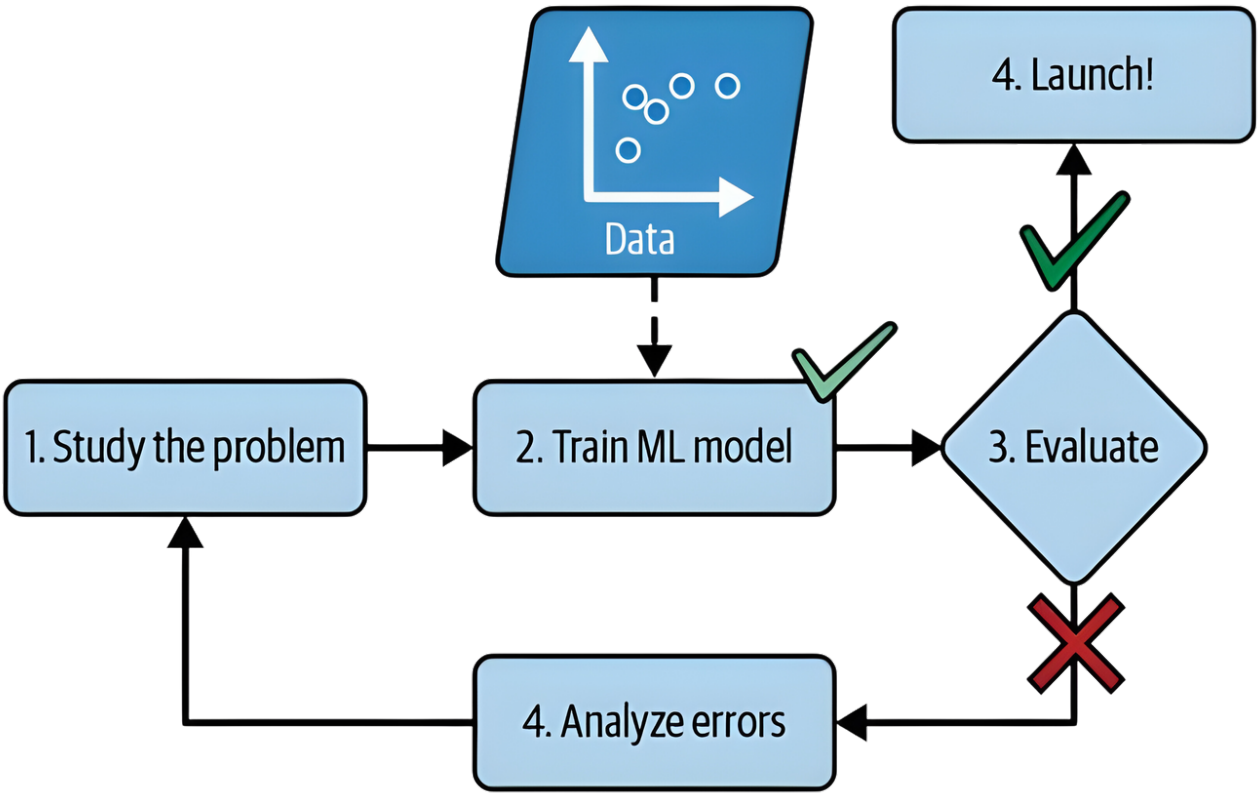
Traditional approach:

- **You'd analyze common patterns in spam, like specific words or phrases, and write detection algorithms for each pattern.**
- **Your program would flag emails as spam if it detected a certain number of these patterns.**
- **This approach results in a long list of complex rules, difficult to maintain and update.**



Machine learning approach:

- **Instead of manually writing rules, the program learns from examples of spam and non-spam emails.**
- **It automatically identifies which words or phrases are good predictors of spam by detecting unusual patterns in spam examples compared to non-spam examples.**
- **This approach results in a shorter, more maintainable program that's likely more accurate.**



Advantages of machine learning:

- **It adapts automatically to new spam patterns, reducing the need for manual updates.**
- **It's effective for complex problems like speech recognition, where traditional approaches fall short.**
- **Machine learning models can be inspected to reveal insights and patterns, aiding in problem understanding and data mining.**
- **In summary, machine learning offers a more efficient and adaptable solution for spam filtering and excels at solving complex problems and discovering hidden patterns in large datasets.**

Examples of Applications

- **Sorting Products on a Production Line**: Machines can analyze images of products and sort them automatically. This task, called image classification, is usually done using techniques like convolutional neural networks (CNNs).

- **Finding Tumors in Brain Scans**: In medical imaging, identifying tumors requires precise classification of each pixel in an image. This is achieved through semantic image segmentation, often using CNNs.

- **Classifying News Articles**: To automatically categorize news articles, we turn to natural language processing (NLP) techniques like text classification. Recurrent neural networks (RNNs) and CNNs are common tools, but transformers are even more effective.

- **Identifying Offensive Comments**: Similar to classifying news articles, spotting offensive comments on forums involves text classification using NLP tools.

- **Summarizing Lengthy Documents**: Text summarization, a branch of NLP, condenses long texts into shorter versions using various NLP techniques.

- **Creating Chatbots or Personal Assistants**: Building chatbots involves understanding natural language (NLU) and answering questions accurately, requiring various NLP components.

- **Revenue Forecasting**: Predicting future revenue based on past performance metrics falls under regression tasks. Models like linear regression, support vector machines, or neural networks can be used.

- **Voice Command Recognition**: Processing voice commands, known as speech recognition, involves analyzing audio samples using techniques like RNNs, CNNs, or transformers.

- **Detecting Credit Card Fraud**: Anomaly detection methods, such as isolation forests or autoencoders, help identify unusual patterns indicating fraud.

- **Client Segmentation for Marketing**: Clustering techniques like k-means or DBSCAN group clients based on purchasing behavior, allowing tailored marketing strategies.

- **Data Visualization**: Representing complex data in understandable diagrams often requires dimensionality reduction techniques to visualize high-dimensional data effectively.

- **Product Recommendations**: Recommender systems analyze past purchases to suggest products. Artificial neural networks are commonly used for this, trained on sequences of past purchases across all clients.

- **Intelligent Game Bots**: Reinforcement learning trains game bots to make decisions that maximize rewards over time within a gaming environment. The famous AlphaGo program was built using reinforcement learning.

These examples demonstrate the wide range of tasks machine learning can handle and the diverse techniques employed for each task.

Types of Machine Learning Systems

Understanding machine learning systems can be simplified by categorizing them based on key criteria:

Supervision during Training: Machine learning systems can be supervised, unsupervised, semi-supervised, self-supervised, or a combination of these. Supervised learning involves learning from labeled data, unsupervised learning deals with unlabeled data, semi-supervised learning uses a small amount of labeled data with a large amount of unlabeled data, and self-supervised learning generates labels from the data itself.

Learning Incrementally: Machine learning systems can learn incrementally on the fly (online learning) or in batches (batch learning). Online learning allows systems to update their knowledge as new data becomes available, while batch learning requires all data to be available upfront for training.

Learning Approach: Machine learning systems can either compare new data points to known data points or detect patterns in the training data to build predictive models. Instance-based learning involves directly comparing data points, while model-based learning involves building a predictive model based on patterns in the data.

These criteria can be combined in various ways. For instance, a spam filter may learn on the fly using a deep neural network model trained with labeled examples of spam and non-spam emails, making it an online, model-based, supervised learning system.

By understanding these criteria, we can better grasp the workings of different machine learning systems and how they are applied in various real-world scenarios.

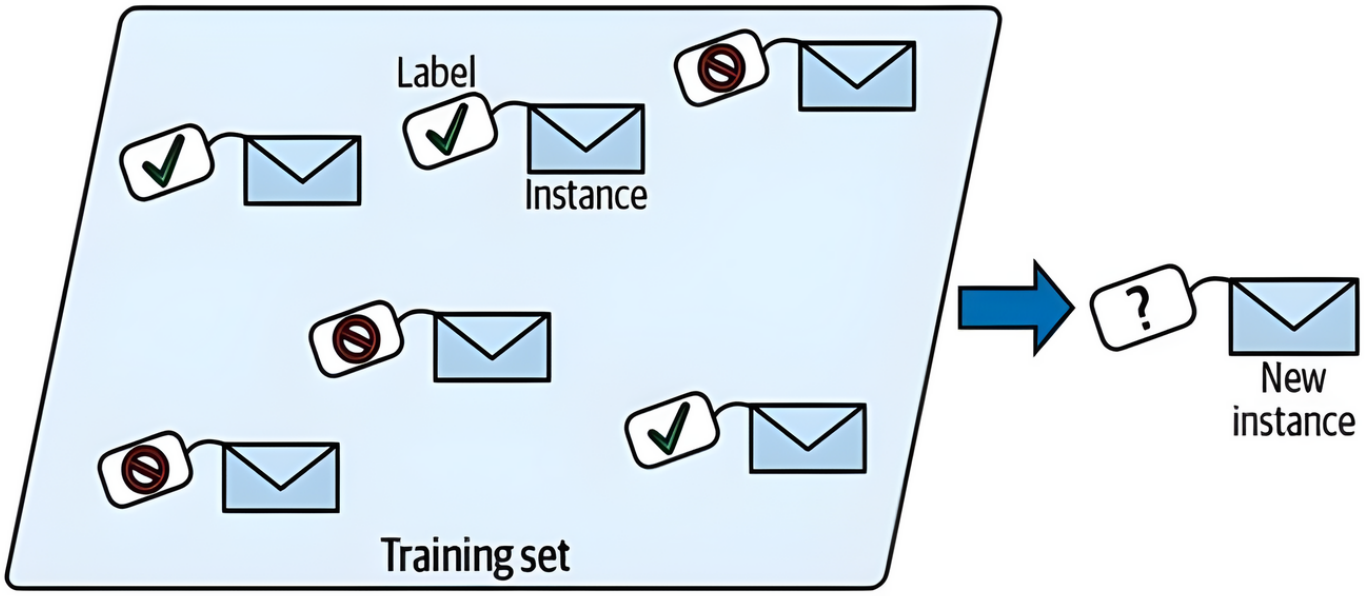
Training Supervision

Machine learning systems are categorized based on the supervision they receive during training. The main types include supervised, unsupervised, self-supervised, semi-supervised, and reinforcement learning.

Supervised learning

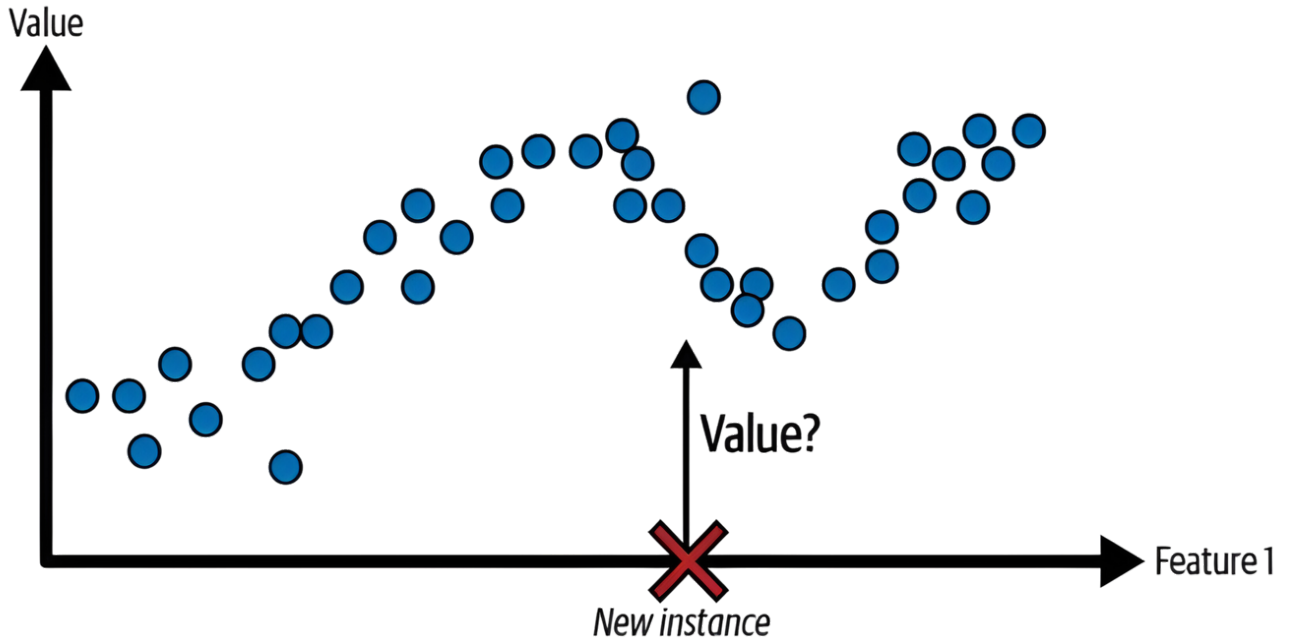
In supervised learning, you provide the algorithm with labeled data, where the labels represent the desired solutions. For example, imagine training a spam filter. You'd give it many example emails labeled as either spam or not spam, so it can learn to recognize new emails. Another common task is

regression, where the algorithm predicts a number, like the price of a car, based on features such as mileage and age. To train it, you need lots of examples of cars with their features and prices.



A labeled training set for spam classification (an example of supervised learning)

It's interesting that some regression models, like logistic regression, can also be used for classification. For instance, logistic regression can tell you the probability of something belonging to a certain class, making it handy for tasks like spam classification.



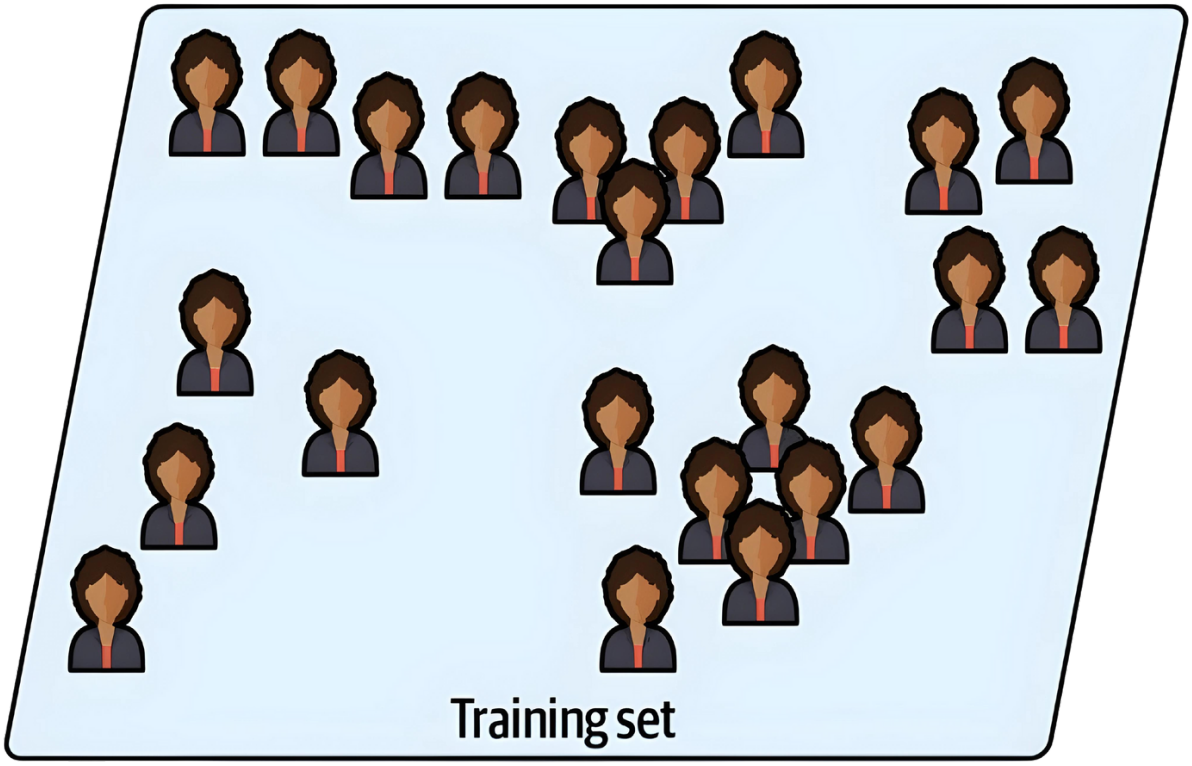
A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)

Note:

In supervised learning, "target" and "label" are often used interchangeably, but "target" is typically associated with regression tasks, while "label" is more common in classification tasks. Additionally, features can also be referred to as predictors or attributes. These terms describe the characteristics of the data used for training machine learning models. They can either refer to specific attributes of individual samples, like the mileage of a car, or describe general relationships across the entire dataset, such as the correlation between mileage and price

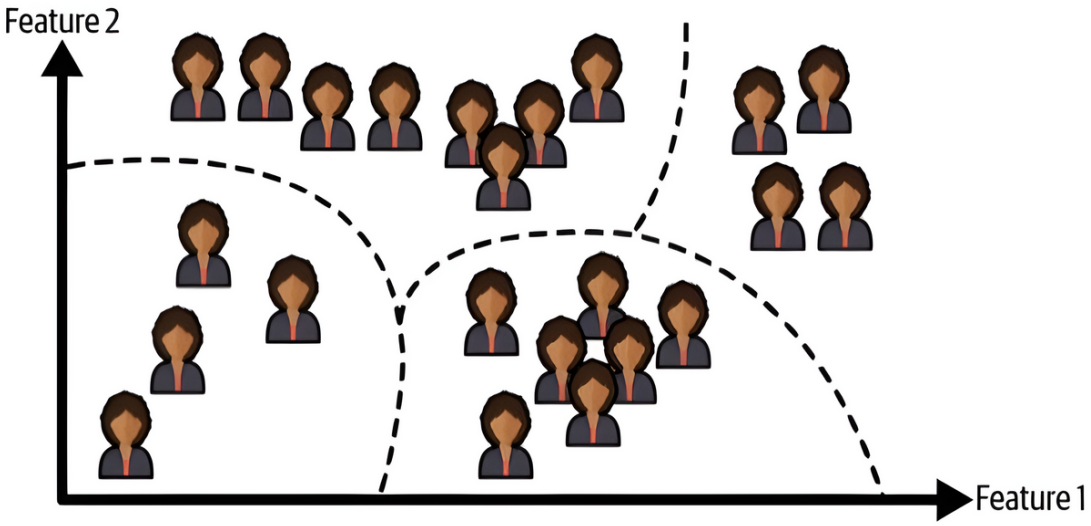
Unsupervised learning

In unsupervised learning, the system learns from data that doesn't have predefined labels. Imagine you have information about visitors to your blog, but you don't know which group each visitor belongs to. You could use clustering algorithms to find groups of similar visitors, like teenagers who like comic books or adults who enjoy sci-fi.



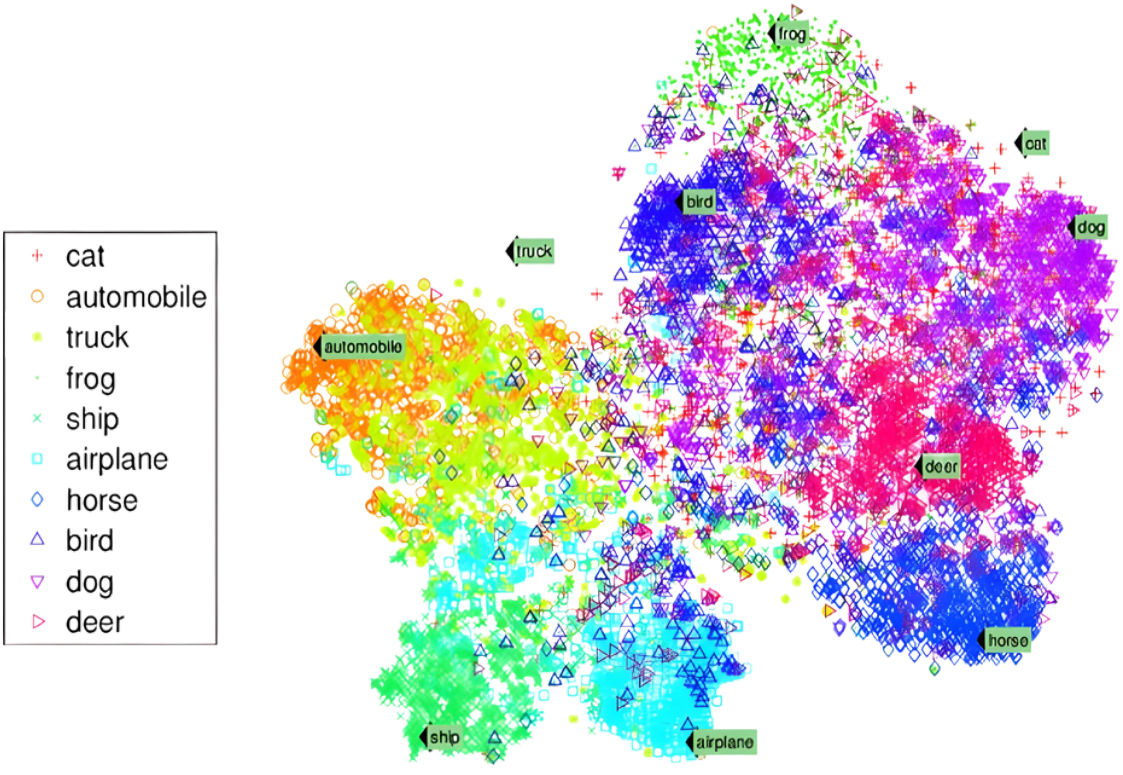
Another example is visualization algorithms that turn complicated, unlabeled data into simple 2D or

3D images. These images help us understand patterns in the data even without labels.



Clustering

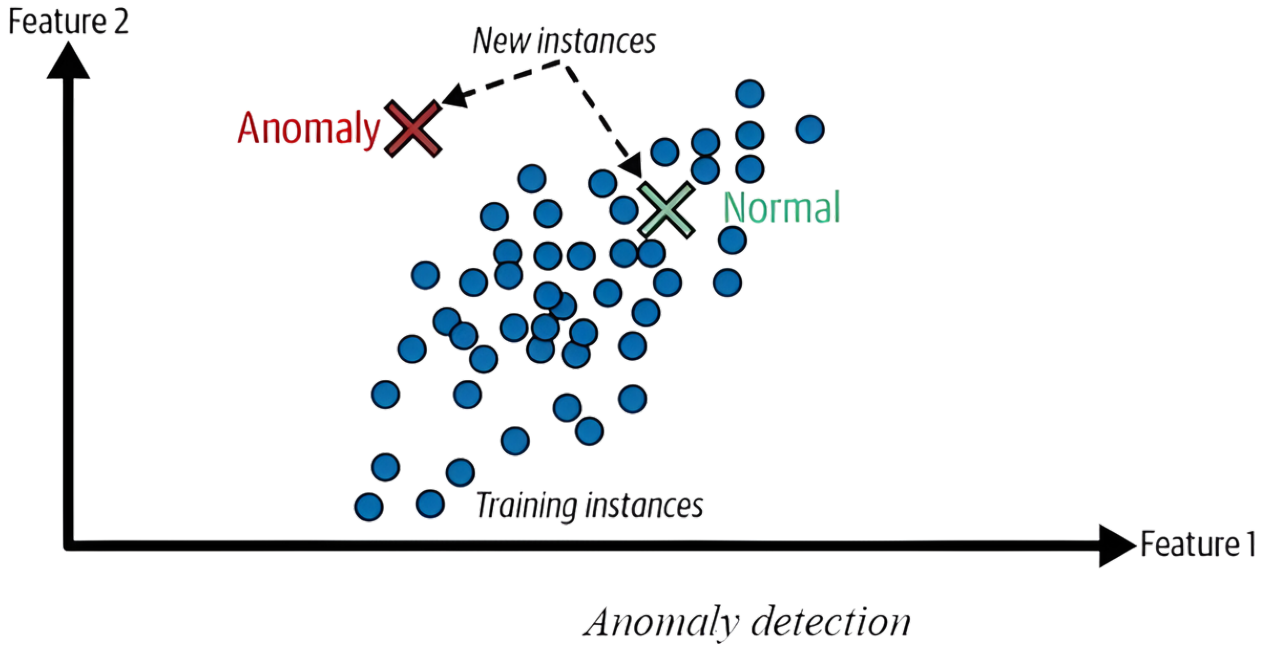
Dimensionality reduction is a similar task where related features are combined to make the data simpler. For example, instead of looking at a car's mileage and age separately, you might combine them to understand wear and tear.



Example of a t-SNE visualization highlighting semantic clusters[2]

Unsupervised learning also includes anomaly detection, where the system learns from normal data to spot unusual things like odd credit card transactions or defects in manufacturing.

Novelty detection is a bit like anomaly detection but focuses on identifying entirely new instances not seen before.



Anomaly detection

Lastly, association rule learning uncovers relationships between different aspects in large datasets. For example, it might find that customers who buy barbecue sauce and potato chips also tend to buy steak at a supermarket.

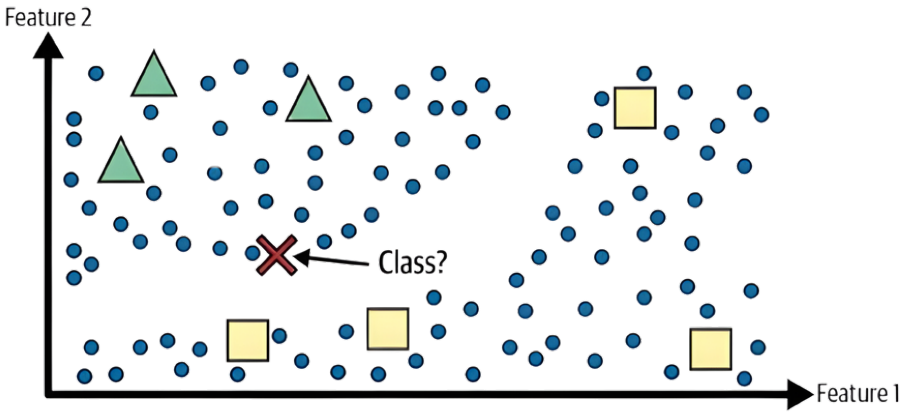
Suggestion

Before you use your data to train a machine learning model, try using a dimensionality reduction algorithm. This helps by making the data simpler, which can speed up the process, save storage space, and sometimes even improve the performance of your model

Semi-supervised learning

Semi-Supervised Learning: When you have lots of unlabeled data and only a few labeled instances, semi-supervised learning comes in handy. For instance, in Google Photos, the

system automatically groups similar photos together (unsupervised part), and you just need to label each person once to name everyone in all photos.



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

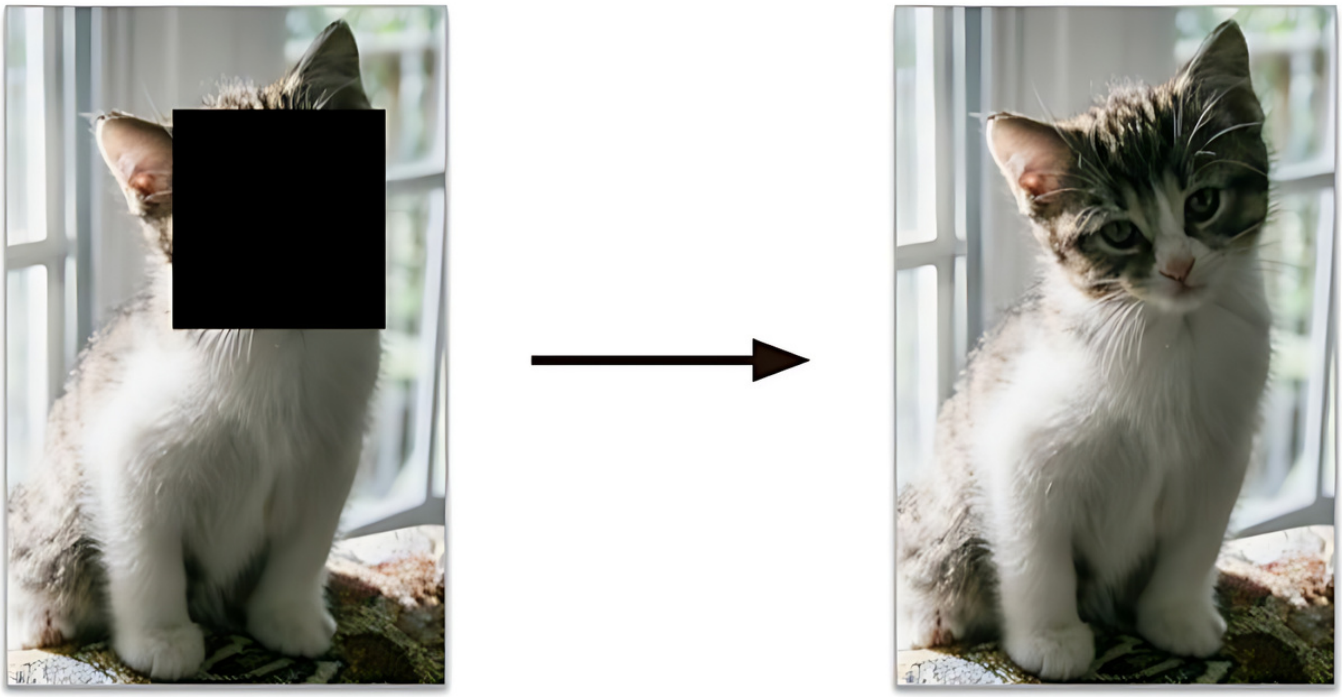
Combination of Algorithms: Most semi-supervised learning methods blend unsupervised and supervised techniques. For example, a clustering algorithm groups similar items, and then each unlabeled item is labeled based on the most common label in its cluster. Once the whole dataset is labeled, any supervised learning algorithm can be applied.

Self-supervised learning

Self-Supervised Learning: This approach creates a fully labeled dataset from an unlabeled one. Once labeled, any supervised learning algorithm can be applied. For example, you can start with a dataset of unlabeled images and train a model to recover the original images by masking a small part of each one.

Training Process: During training, the masked images are used as inputs, and the original images act as labels.

Utility of Resulting Model: The trained model can be useful on its own, like repairing damaged images or removing unwanted objects. However, it's often used as a starting point for a different task.



Self-supervised learning example: input (left) and target (right)

Example of Pet Classification: Let's say you want a model to classify pet species. You can start by training an image-repairing model using self-supervised learning. Once it's good at repairing images, you can tweak it to recognize different pet species based on the repairs it makes.

Fine-Tuning: Finally, the model is fine-tuned on a labeled dataset to learn the mapping between known species and expected labels.

Classification vs. Unsupervised Learning: While self-supervised learning deals with unlabeled datasets, it uses generated labels during training, making it closer to supervised learning. Unsupervised learning typically involves tasks like clustering or anomaly detection, while self-supervised learning focuses on classification and regression.

Note:

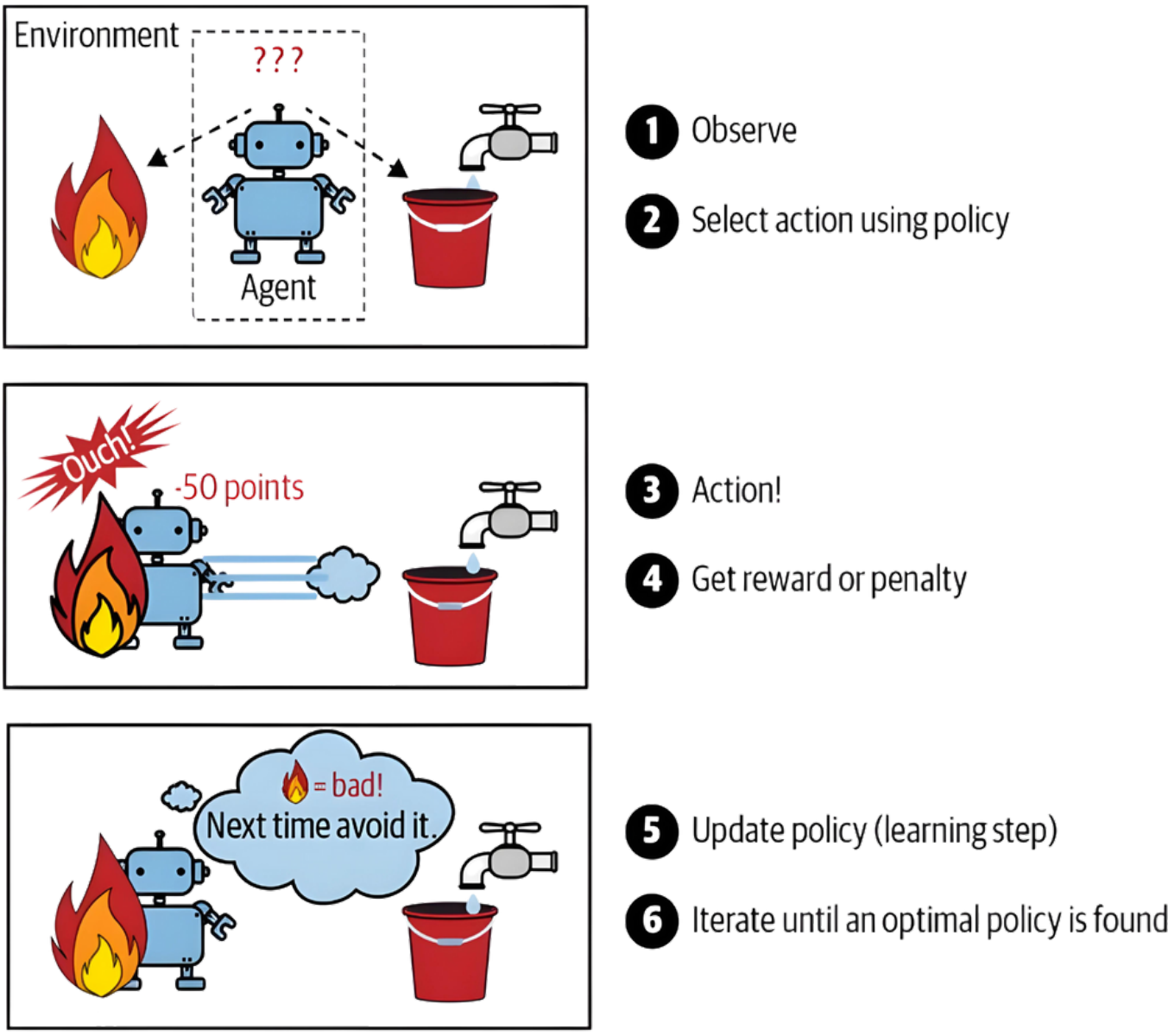
Remember, transferring knowledge from one task to another is called transfer learning. This is a big deal in machine learning, especially with deep neural networks, which are networks with lots of layers. We'll dive deeper into this topic soon!

Conclusion: Self-supervised learning is considered its own category, distinct from

supervised and unsupervised learning.

Reinforcement learning

Reinforcement Learning Basics: Reinforcement learning is different from other types of machine learning. Here, the learning system, called an agent, observes the environment, takes actions, and receives rewards or penalties based on its actions.



Reinforcement learning

Learning Strategy: The agent learns to determine the best strategy, called a policy, to maximize rewards over time. A policy guides the agent on which action to choose in a given

situation.

Example: Robots and AlphaGo: Many robots use reinforcement learning to learn tasks like walking. For example, DeepMind's AlphaGo program learned to play the game of Go by analyzing millions of games and playing against itself. It achieved success by applying the winning policy it had learned.

Offline Learning: During matches against top players like Ke Jie, AlphaGo simply applied the policy it had learned, without further learning. This is known as offline learning.

In simpler terms, reinforcement learning is about learning by taking actions and getting rewards or penalties. Robots and programs like AlphaGo use this method to learn complex tasks and strategies.

Batch Versus Online Learning

Another way we classify machine learning systems is based on whether they can learn from new data as it comes in, bit by bit. This means the system can update its knowledge as it receives more information over time.

Batch learning

Batch Learning:

- The system learns from all available data at once and can't learn incrementally.
- It requires a lot of time and computing resources, often done offline.
- Once trained, the system runs without learning anymore, known as offline learning.
- However, the model's performance can decay over time due to changes in the world, known as model rot or data drift.

Even if a model learns to tell cats from dogs, it might need regular updates. Not because the pets change, but because things like cameras, picture quality, and pet fashion keep changing. Who knows, maybe next year everyone will love a new cat breed or start putting tiny hats on their pets!

Adapting to New Data:

- To update a batch learning system with new data (e.g., new spam types), you need to retrain it from scratch on the entire dataset.
- Fortunately, the process of training, evaluating, and launching a system can be automated, allowing adaptation to changes.

Challenges with Full Data Training:

- Training with the full dataset can take many hours and require significant computing resources.
- Doing this frequently, especially with large datasets, can be costly and resource-intensive.

Limitations of Batch Learning:

- If the data changes rapidly or the system has limited resources (e.g., a smartphone app or a Mars rover), batch learning may not be suitable.
- Carrying large training datasets and consuming extensive resources for frequent training may not be feasible.

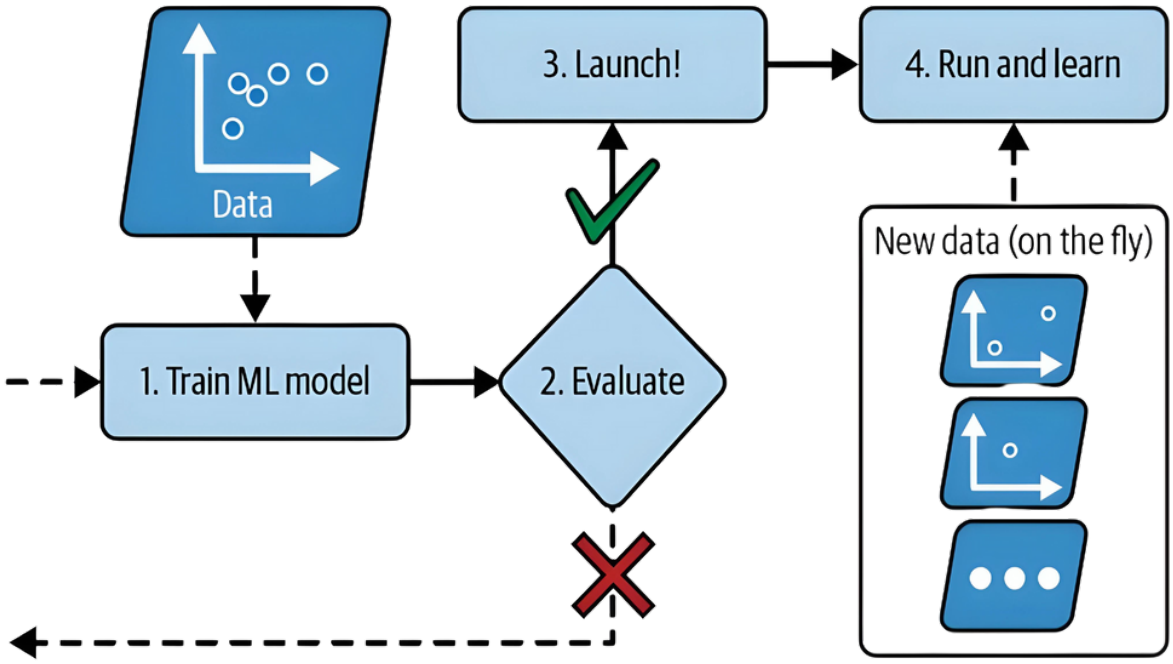
Incremental Learning:

- A better option in such cases is to use algorithms capable of learning incrementally.
- These algorithms can adapt to new data over time, making them more efficient and suitable for systems with changing data or resource constraints.

Online learning

Online Learning:

- Train the system incrementally by feeding data instances sequentially, either one by one or in small groups (mini-batches).
- Each learning step is quick and cost-effective, allowing the system to learn about new data as it arrives.

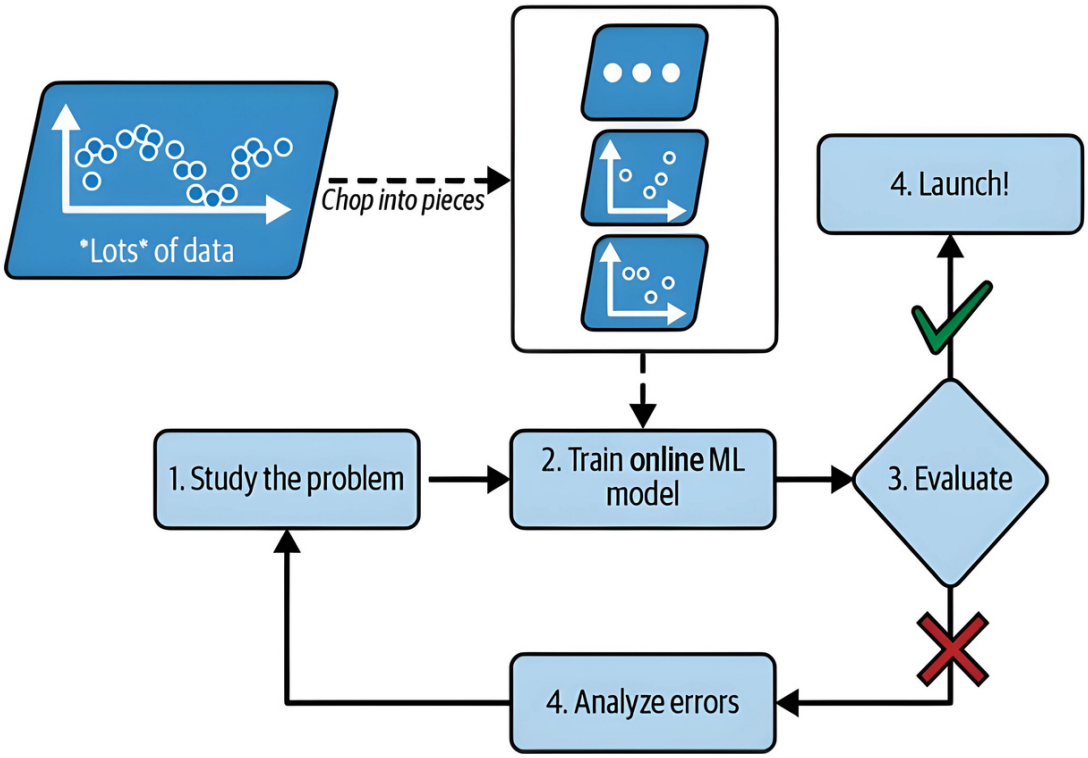


In online learning, a model is trained and launched into production, and then it keeps learning as new data comes in

Benefits of Online Learning:

- Ideal for systems needing rapid adaptation to change, like detecting new patterns in the stock market.
- Great for scenarios with limited computing resources, such as training models on mobile devices.

- Can handle huge datasets that don't fit into a single machine's memory through out-of-core learning, where data is loaded in parts for training.



Using online learning to handle huge datasets

Learning Rate:

- An important parameter in online learning systems is the learning rate, determining how fast the system should adapt to changing data.
- Setting a high learning rate makes the system quickly adapt to new data but may forget old data too fast (not ideal for a spam filter).
- Conversely, setting a low learning rate makes the system learn slowly but be less sensitive to noise or outliers in the data.

Out-of-core learning, often called "online learning," is actually done offline. Instead, think of it as incremental learning, where the model is continuously updated with new data as it arrives. This is helpful for adapting to changing data quickly, like detecting new patterns in the stock market, and is useful for systems with limited resources or massive datasets.

Instance-Based Versus Model-Based Learning

In machine learning, we often categorize systems based on how well they make predictions for new, unseen examples. While it's essential to perform well on training data, the real goal is to generalize and make accurate predictions for new instances. We have two main ways of achieving this: instance-based learning and model-based learning.

Instance-based learning

Learning by Heart:

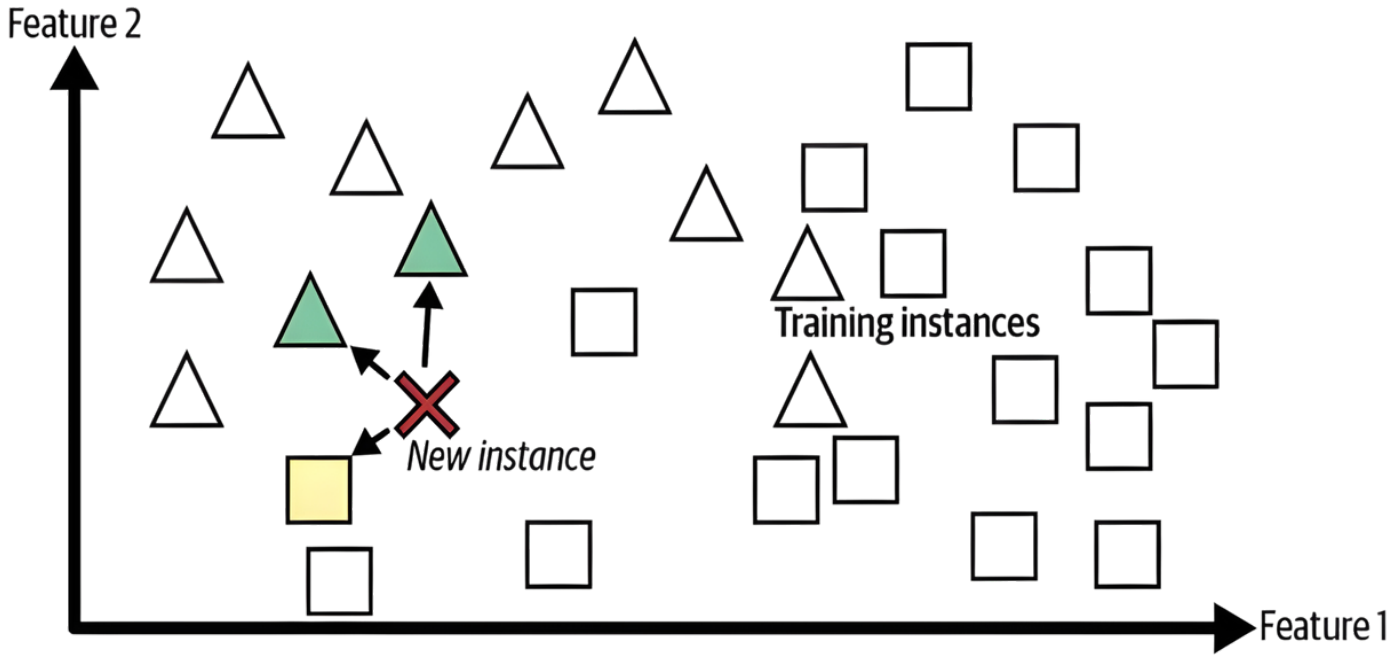
- This is the most basic form of learning where the system memorizes examples. For instance, a spam filter would simply flag emails identical to those previously marked as spam by users.

Improving the Spam Filter:

- Instead of just flagging identical emails, the spam filter can also flag emails very similar to known spam ones.
- To do this, the system needs a way to measure similarity between emails, like counting common words.

Instance-Based Learning:

- This approach is called instance-based learning, where the system memorizes examples and then makes predictions based on similarity to those examples.
- For example, if most similar instances to a new shape are triangles, the system would classify it as a triangle too.

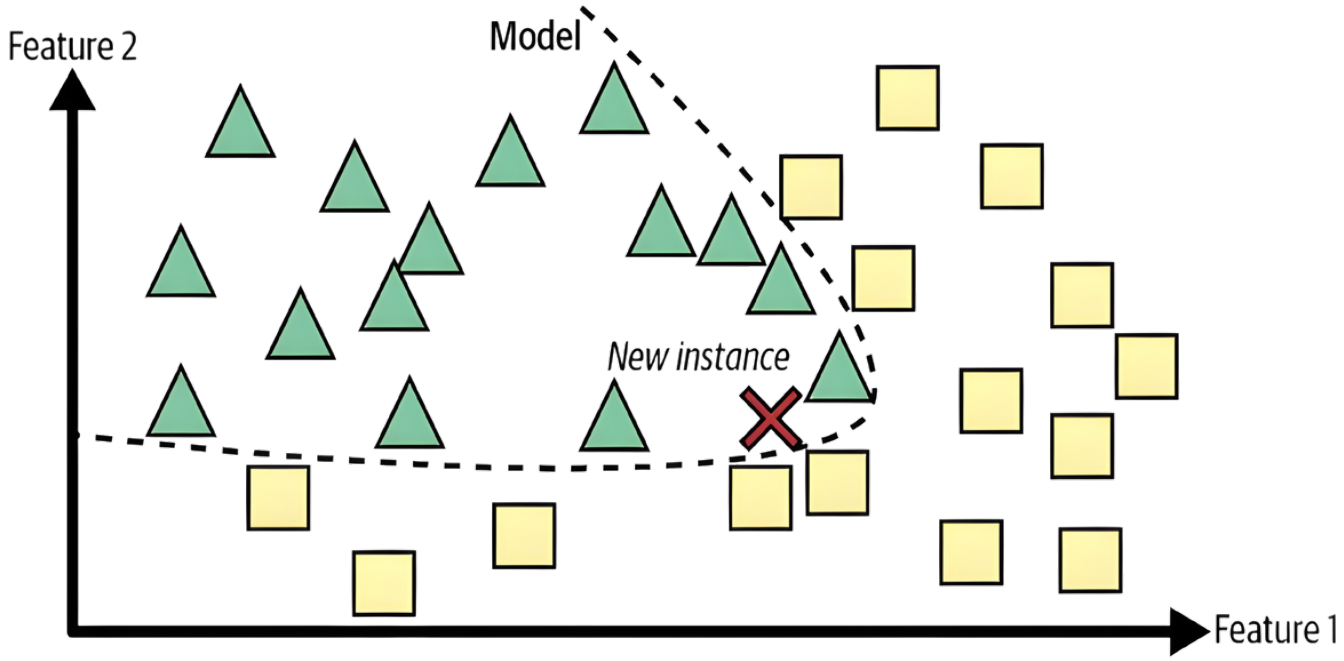


Instance-based learning

Model-based learning and a typical machine learning workflow

Model-Based Learning:

Model-based learning involves building a model from a set of examples and then using that model to make predictions. Let's consider an example: suppose you're curious about whether money influences people's happiness. You collect data from the OECD's Better Life Index and the World Bank on gross domestic product (GDP) per capita. After sorting the data by GDP per capita, you notice a trend: as GDP per capita increases, so does life satisfaction. To model this relationship, you choose a linear function, where life satisfaction is predicted based on GDP per capita.



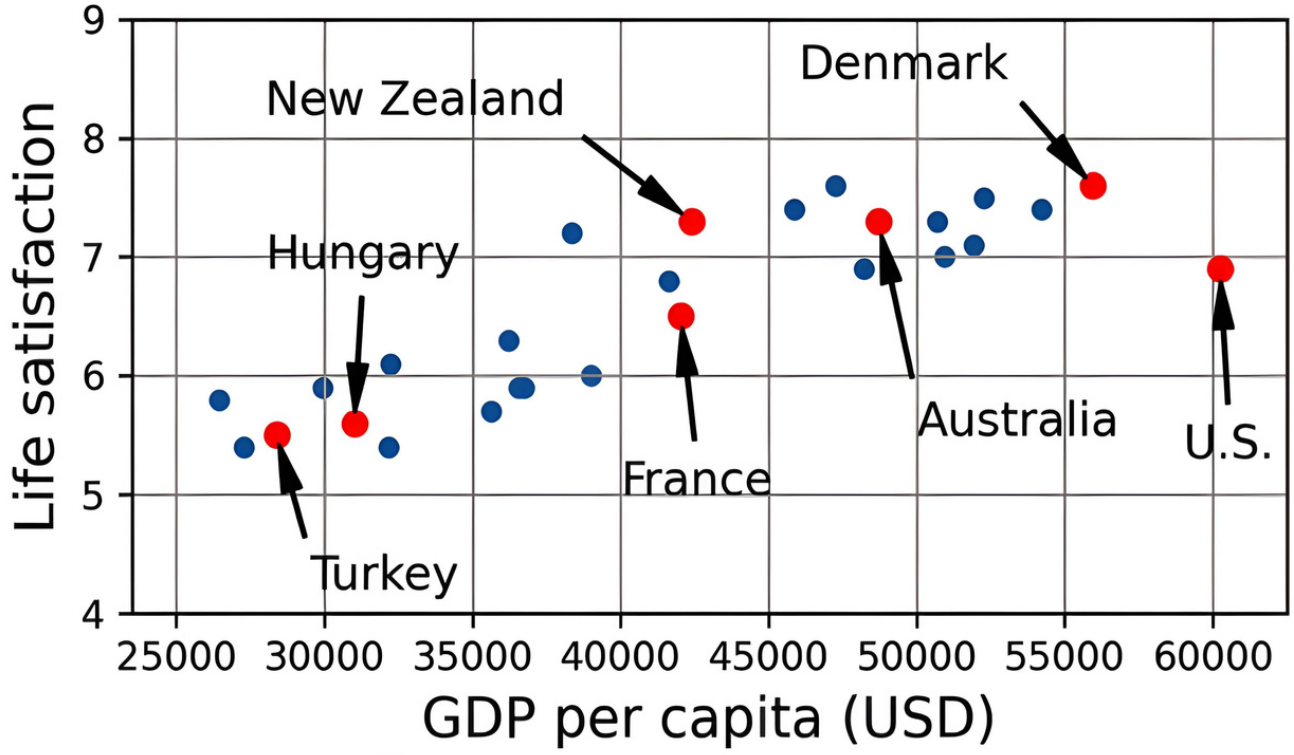
Model-based learning

Model Selection:

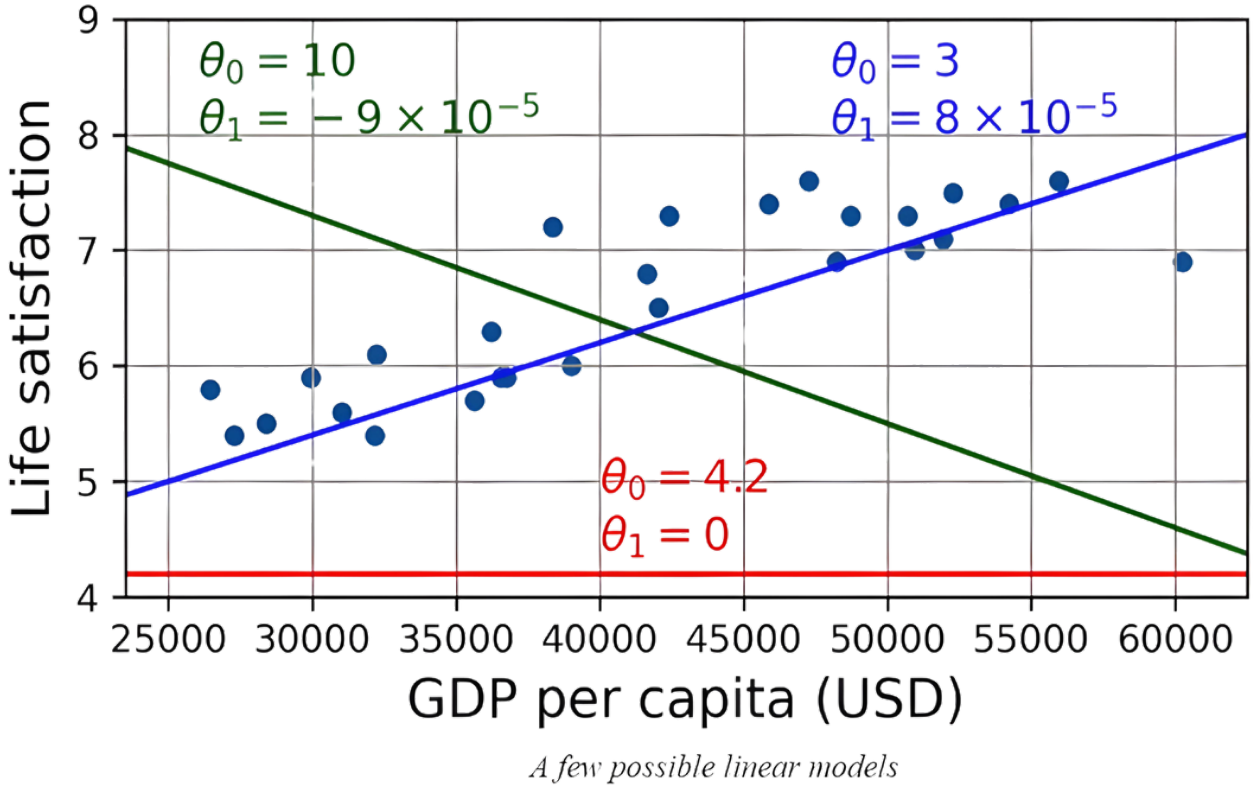
Now, you need to select a specific model. In this case, you opt for a simple linear model where life satisfaction is calculated using the equation: life satisfaction = θ0 + θ1 × GDP_per_capital. The parameters θ0 and θ1 are values that need to be determined to make the model fit the data accurately.

Does money make people happier?

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



Do you see a trend here?



A few possible linear models

Parameter Estimation:

Before using the model, you must find the optimal parameter values θ0 and θ1. To do this, you define a performance measure, typically a cost function that measures the difference between the model's predictions and the actual data. Then, you employ a linear regression algorithm, which adjusts the parameters to minimize this difference. For our example, the algorithm finds that θ0 = 3.75 and θ1 = 6.78 × 10^−5.

Model Validation:

With the optimal parameters, your model fits the training data closely. Now, you can utilize it to make predictions. For instance, if you want to predict the life satisfaction of Cypriots, you input Cyprus's GDP per capita into the model and find a predicted life satisfaction value.

Python Code Example:

To demonstrate this process, below is a Python code snippet that loads the data, separates the inputs and labels, creates a scatterplot for visualization, trains a linear model, and makes predictions. This hands-on example showcases how machine learning algorithms can be implemented in practice to analyze and predict real-world phenomena.

1. **Understanding "Model":**
   - **The term "model" can mean different things, adding to the confusion.**
   - **It can refer to the type of model (like linear regression), the specific architecture of the model (e.g., linear regression with one input and one output), or the final trained model ready for predictions.**
2. **Model Selection:**
   - **Model selection involves choosing the type of model and specifying its architecture fully.**
3. **Training a Model:**
   - **Training a model entails running an algorithm to find the best parameters that allow it to fit the training data optimally.**
   - **The goal is to ensure that the model makes accurate predictions on new data.**

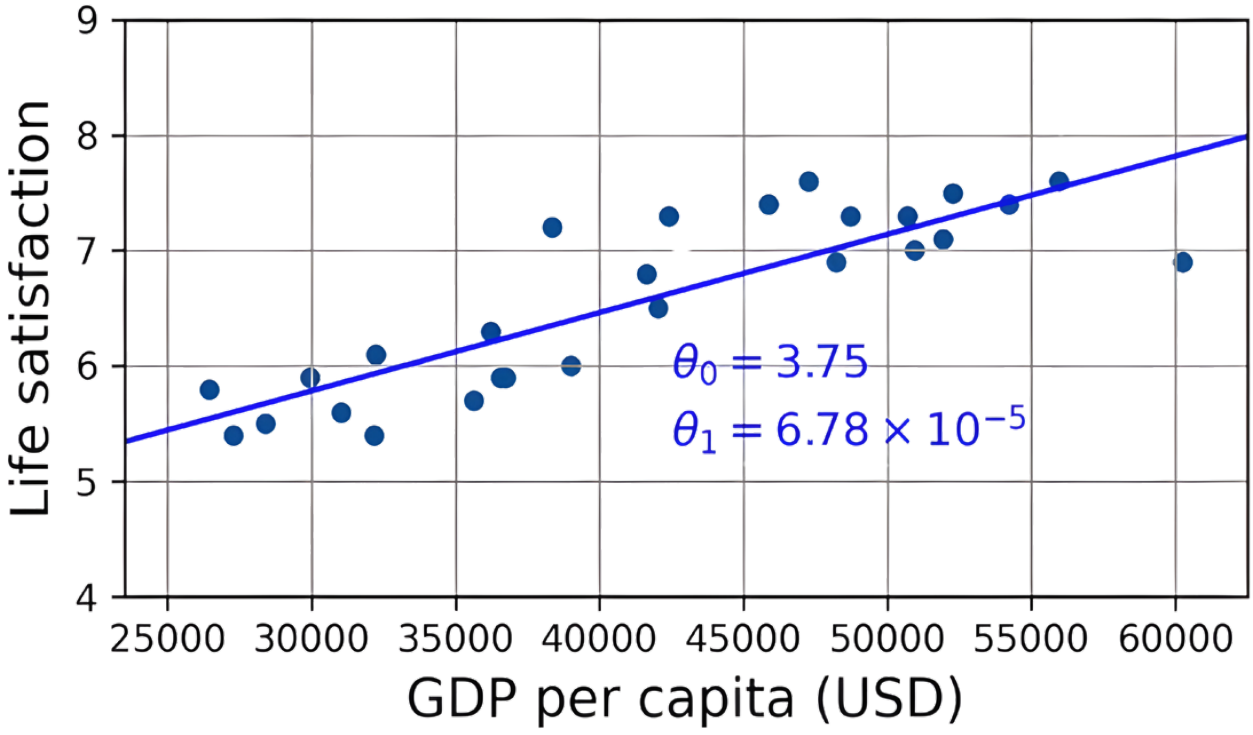
By understanding these distinctions, you'll navigate the world of machine learning with clarity and confidence.

Running the Model:

- Now that your model is trained, it's time to put it to work and make predictions.
- For instance, let's say you're curious about the happiness level of people in Cyprus, but the OECD data doesn't have that information.

Using Your Model for Predictions:

- Fortunately, you can use your trained model to make a good prediction.
- You find Cyprus's GDP per capita, which is $37,655.
- Applying your model to this data, you predict that the life satisfaction in Cyprus is likely to be around 6.30.



The linear model that fits the training data best

Python Code Example:

- To give you a taste of how this works, below is an example of Python code:

```
# Example : Python Code

# Load data, separate inputs X from labels y

# Create a scatterplot for visualization

# Train a linear model and make a prediction



# Your code goes here...
```

Visualization:

- The code loads the data, separates inputs and labels, and creates a scatterplot for visualization.
- Then, it trains a linear model and makes a prediction based on the input data.

Understanding how to make predictions using your model is an exciting step in your journey through machine learning!

Training and running a linear model using Scikit-Learn

```
import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.linear_model import LinearRegression



# Download and prepare the data

data_root = "https://raw.githubusercontent.com/mmiimran/data_set_4_data_science/main/"

lifesat = pd.read_csv(data_root + "lifesat/lifesat.csv")

X = lifesat[["GDP per capita (USD)"]].values

y = lifesat[["Life satisfaction"]].values
```

```
# Visualize the data
lifesat.plot(kind='scatter', grid=True,
        x="GDP per capita (USD)", y="Life satisfaction", color='blue')
plt.axis([23_500, 62_500, 4, 9])
plt.show()

# Select a linear model
model = LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[37_655.2]]  # Cyprus' GDP per capita in 2020
print(model.predict(X_new))  # output: [[6.30165767]]
```

Note

If you used a different learning method called instance-based learning, you'd find that Israel has a similar GDP per capita to Cyprus ($38,341). Since Israelis' life satisfaction is 7.2 according to OECD data, you'd predict a life satisfaction of 7.2 for Cyprus. Looking at the two next-closest countries, Lithuania and Slovenia, with a life satisfaction of 5.9 each, and averaging these values, you'd get 6.33, which is quite close to the prediction made by the model-based approach. This simple method is called k-nearest neighbors regression, where k represents the number of neighbors considered (in this case, k = 3).

To switch from using linear regression to k-nearest neighbors regression in the code, you just need to replace the following lines:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

---

with these two lines:

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=3)
```

This change will allow you to use the k-nearest neighbors algorithm for prediction.

If your model is making accurate predictions, great job! If not, don't worry. You might need to consider adding more attributes like employment rate, health, or air pollution to improve your predictions. Also, getting more or better-quality training data could help, or you might need to choose a more powerful model, like a polynomial regression model.

In summary:

- You studied the data carefully.
- You chose a suitable model for your project.
- You trained the model on the training data, where the learning algorithm searched for the best model parameter values to minimize a cost function.
- Finally, you used the trained model to make predictions on new data (this is called inference), hoping it would generalize well.
- This is a typical workflow for a machine learning project. In Chapter 2, you'll get hands-on experience with a project from start to finish.
- So far, we've covered a lot: you now understand what machine learning is, why it's valuable, common categories of ML systems, and the typical workflow of a project. Now, let's explore some common challenges in learning and how to overcome them.

Main Challenges of Machine Learning

Alright, in simple terms, when you're working on selecting a model and training it with some data, there are two main things that can go wrong: "bad model" and "bad data". Now, let's dive into examples of bad data to understand it better.

Insufficient Quantity of Training Data

Learning with Examples: Just like how a toddler learns what an apple is by seeing and hearing the word "apple", machine learning needs examples to understand things.

Need for Lots of Data: Unlike toddlers, machine learning algorithms need a lot of examples to learn properly. Even for simple tasks, thousands of examples are often needed. For more complex tasks like recognizing images or understanding speech, millions of examples might be necessary, unless parts of an existing model can be reused.

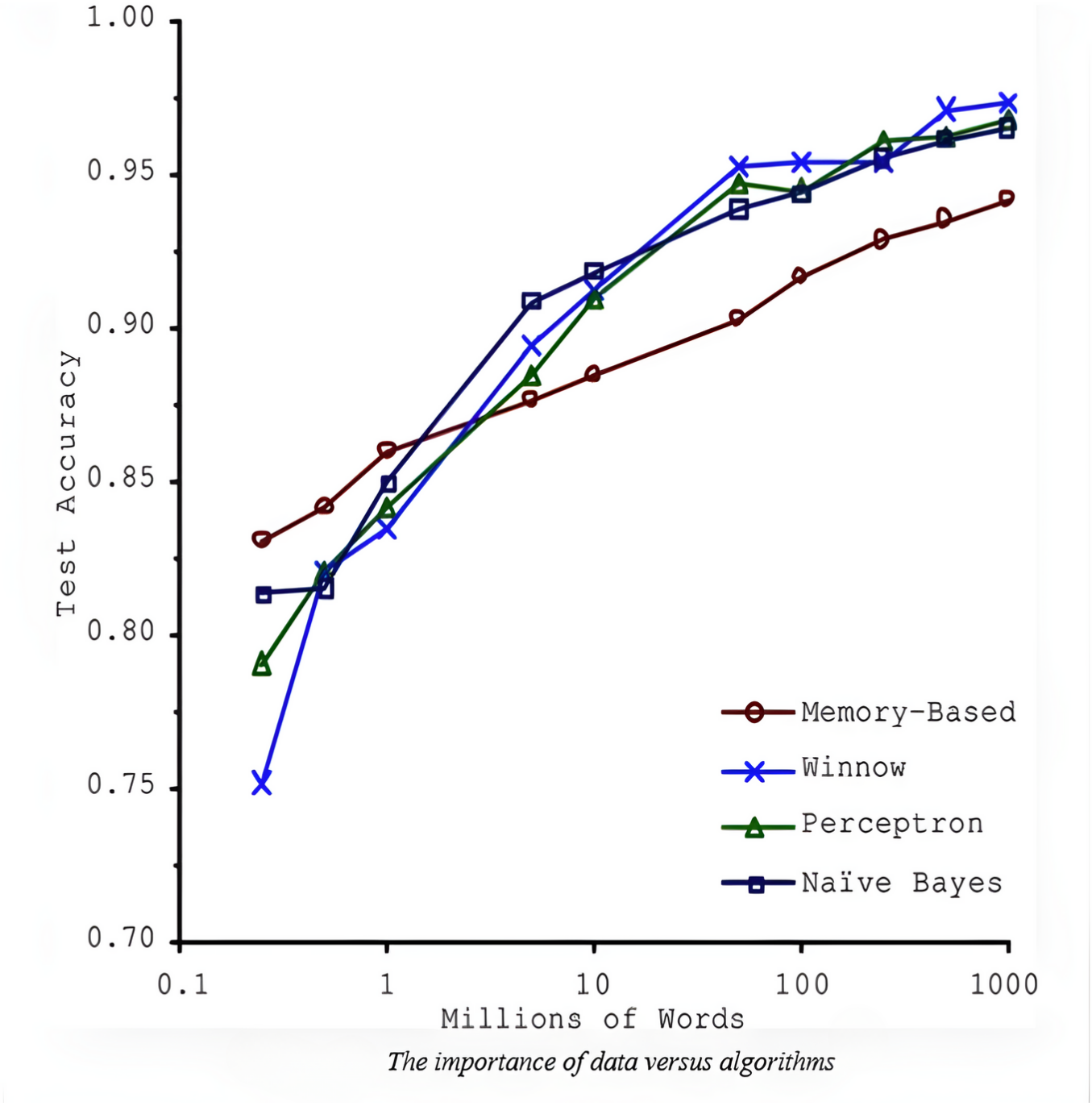
THE UNREASONABLE EFFECTIVENESS OF DATA

Data vs. Algorithms: In a paper from 2001, Microsoft researchers found something surprising: different machine learning methods performed almost equally well on a tricky language problem when given lots of data.

The Big Idea: They suggested that instead of focusing too much on fancy algorithms, we should pay more attention to collecting and preparing good quality data.

Data's Power: This idea became even more famous in 2009 with a paper titled "The Unreasonable Effectiveness of Data" by Peter Norvig and others. They emphasized that having lots of data can often be more important than using complex algorithms.

Reality Check: But remember, getting lots of data isn't always easy or cheap, especially for small or medium-sized projects. So, while data is important, algorithms still have their place in the learning process.



The importance of data versus algorithms

Nonrepresentative Training Data

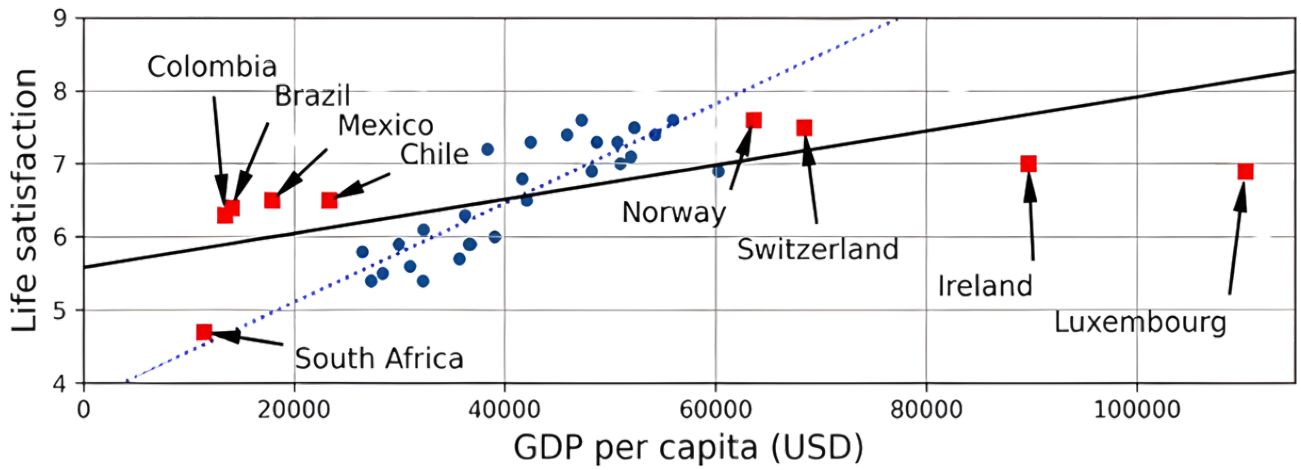
Importance of Representative Data: To make accurate predictions, it's essential that the data used for training represents the new cases you want to predict. This principle applies regardless of whether you're using instance-based learning or model-based learning.

Example with Country Data: Consider the scenario where we trained a linear model to predict life satisfaction based on a country's GDP per capita. However, the initial training data didn't include countries with very low or very high GDPs.

Impact of Missing Data: When we added these missing countries to our data, our model changed significantly. The new data showed that a simple linear model might not work well because it failed to capture certain trends. For instance, very rich countries didn't seem happier, and some poorer countries seemed happier.

Consequences of Non-representative Data: By using a non-representative training set, we ended up with a model that's unlikely to make accurate predictions, especially for very poor and very rich countries.

Challenges in Ensuring Representativeness: It's challenging to ensure that the training set accurately represents the cases you want to generalize to. Even large samples can be biased if the sampling method is flawed, leading to what's known as sampling bias. This highlights the importance of careful data selection and sampling techniques in machine learning.



A more representative training sample

EXAMPLES OF SAMPLING BIAS

1936 US Presidential Election: The Literary Digest conducted a massive poll involving 10 million people during the 1936 US presidential election. Despite receiving 2.4 million responses, their prediction of Landon winning with 57% of the votes was way off. Roosevelt actually won with 62% of the votes. The flaw was in their sampling method:

- They used sources like telephone directories and magazine subscriber lists, which favored wealthier people more likely to vote Republican.
- Less than 25% of those polled responded, creating a nonresponse bias by excluding those who didn't care much about politics or didn't like the Literary Digest.

Sampling Bias in Music Recognition: Imagine building a system to recognize funk music videos. If you use YouTube search results for "funk music" to build your training set, it assumes that the results are representative of all funk music videos on YouTube. However, YouTube's algorithm may bias results toward popular artists or specific genres like "funk carioca," especially if you're in Brazil. This highlights the challenge of obtaining a diverse and representative training set while avoiding biases introduced by sampling methods

Poor-Quality Data

Importance of Clean Training Data:

- Errors, outliers, and noise in your training data can make it challenging for the system to identify underlying patterns, leading to poor performance.
- Spending time cleaning up your training data is often worthwhile, as it significantly impacts the effectiveness of your machine learning models.

- Many data scientists dedicate a substantial portion of their time to data cleaning, highlighting its importance in the machine learning process.

Examples of Data Cleaning:

- **Handling Outliers**:
  - If certain instances in your data stand out as outliers (i.e., data points significantly different from others), consider discarding them or manually correcting errors if possible.
- **Dealing with Missing Values**:
  - In cases where some instances lack certain features (e.g., missing age information for 5% of customers), you have several options:
    - Ignore the attribute altogether.
    - Exclude instances with missing values.
    - Fill in missing values using techniques like median imputation.
    - Train separate models, one with the feature and one without, to assess their impact on model performance.

Cleaning up training data ensures that machine learning models can learn effectively from the available data, leading to more accurate predictions and better overall performance.

Irrelevant Features

Importance of Relevant Features:

- The success of a machine learning project hinges on the quality of its training data.
- To ensure effective learning, the training data should contain relevant features while avoiding irrelevant ones.
- A popular saying in machine learning is "garbage in, garbage out," emphasizing the importance of quality data inputs for accurate predictions.

Feature Engineering:

- Feature engineering is a crucial process in machine learning that involves crafting meaningful features from raw data.
- It consists of several key steps:
  - **Feature Selection**: Choosing the most informative features from the existing set of features to train the model effectively.
  - **Feature Extraction**: Combining or transforming existing features to create new ones that better represent the underlying patterns in the data. Techniques like dimensionality reduction can aid in this process.
  - **Creating New Features**: Gathering additional data or deriving insights to generate new features that enhance the model's predictive power.

Examples of Bad Algorithms:

- After understanding the importance of good data and feature engineering, it's essential to recognize instances of bad algorithms.
- These algorithms may not effectively utilize the available data or may introduce biases that skew the results.
- Exploring examples of bad algorithms will provide insights into common pitfalls to avoid when designing machine learning systems.

Understanding the significance of relevant features, mastering feature engineering techniques, and discerning good algorithms from bad ones are fundamental steps in building robust and accurate machine learning models.

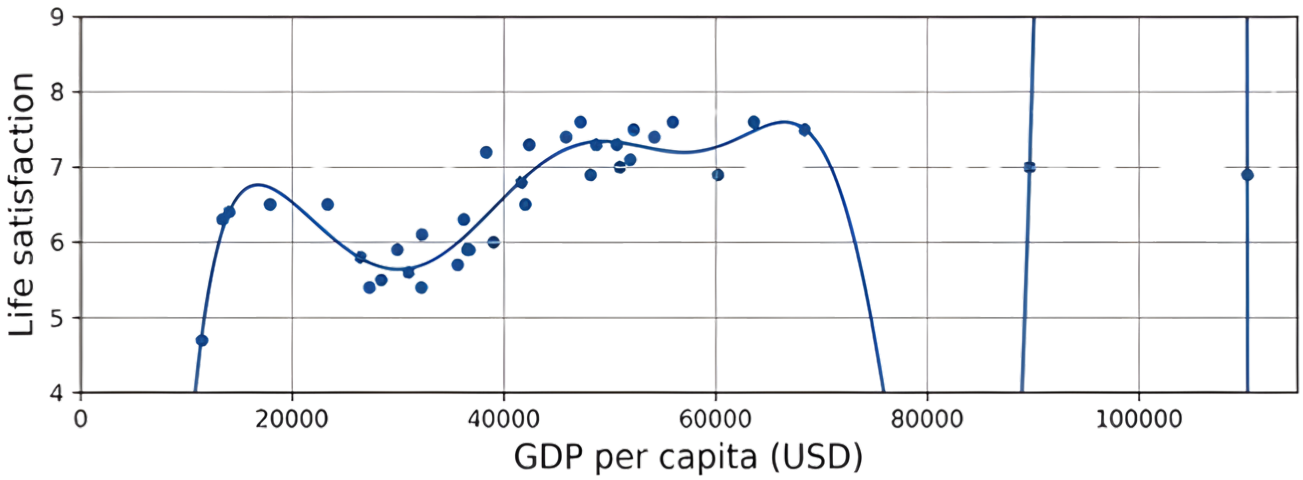
Overfitting the Training Data

Introduction to Overfitting:

- Overgeneralizing, or overfitting, occurs when a model performs well on the training data but fails to generalize to new, unseen data.
- It's akin to judging all taxi drivers in a foreign country as thieves just because one driver ripped you off—an example of human overgeneralization.

Understanding Overfitting with Models:

- Imagine a complex model, like a high-degree polynomial life satisfaction model, which fits the training data extremely well.
- Despite its excellent performance on the training data, this model may not be reliable for making predictions in real-world scenarios.
- Figure 1-23 illustrates this concept, showing a polynomial model that overly fits the training data but may produce unreliable predictions.



Overfitting the training data

Challenges with Complex Models:

- Complex models, such as deep neural networks, are capable of capturing intricate patterns in the data.
- However, they are susceptible to overfitting, especially when the training data is noisy or too small, leading the model to pick up on random patterns in the noise rather than true relationships.
- For instance, if additional irrelevant attributes, like the country's name, are included in the data, a complex model might erroneously associate unrelated features with the target variable.

Risk of Spurious Patterns:

- Complex models may mistakenly identify spurious patterns in the training data, leading to incorrect conclusions.

- For example, a model might incorrectly infer that all countries with a "w" in their name have high life satisfaction scores based on chance occurrences in the training data.
- Without proper safeguards, the model might incorrectly generalize these patterns to unseen data, like Rwanda or Zimbabwe, leading to inaccurate predictions.

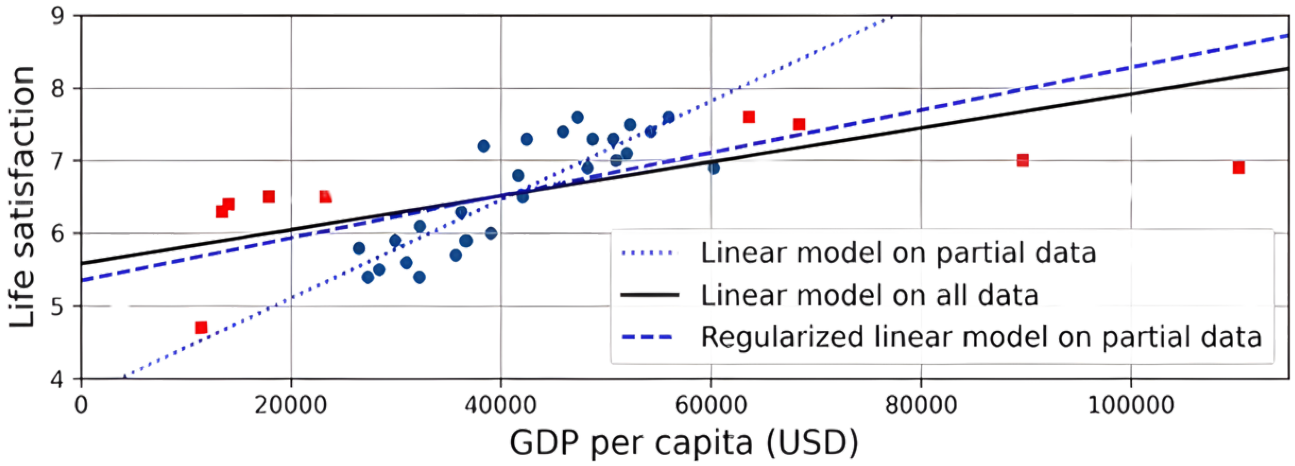
Understanding the concept of overfitting is crucial for building reliable machine learning models. By recognizing the risks associated with overfitting and implementing strategies to mitigate it, data scientists can develop models that generalize well to new data and produce more accurate predictions.

- **Simplify the Model:**
  - **Choose a simpler model with fewer parameters, like a straight line instead of a complex curve. This helps the model generalize better.**
- **Gather More Training Data:**
  - **Increase the amount of training data to provide the model with more examples to learn from. More data can help the model learn the underlying patterns better.**
- **Reduce Noise in the Training Data:**
  - **Clean up the training data by fixing errors and removing outliers or irrelevant information. This ensures that the model focuses on the most important features.**
  -
- **By following these steps, you can avoid overfitting and build more accurate machine learning models.**

- **Regularization and Overfitting:**
  - Regularization is a technique used to make a model simpler and prevent overfitting.
  - Overfitting happens when a model becomes too complex for the available data, resulting in poor generalization to new examples.
- **Reducing Model Complexity:**
  - By constraining a model, such as a linear model, its flexibility is reduced, making it simpler.
  - For instance, setting certain parameters, like θ, to zero limits the model's freedom to fit the data perfectly, resulting in a more generalized model.
- **Finding the Right Balance:**
  - It's essential to strike a balance between fitting the training data accurately and keeping the model simple for better generalization.

- Regularization helps in achieving this balance by controlling the complexity of the model.

- **Effects of Regularization:**
  - Figure 1-24 illustrates the impact of regularization on model complexity.
  - The dashed line represents a model with regularization, which has a smaller slope compared to the original model (solid line).
  - While the regularized model may not fit the training data as closely, it tends to generalize better to new examples, reducing the risk of overfitting.
- **Controlling Regularization:**
  - The level of regularization is controlled by a hyperparameter, which must be set before training and remains constant during training.
  - Adjusting the regularization hyperparameter allows for fine-tuning the model's complexity to achieve optimal performance.



Regularization reduces the risk of overfitting

Understanding regularization is crucial for building machine learning systems that can generalize well to unseen data while avoiding overfitting. Tuning hyperparameters, including regularization, is an integral part of developing robust machine learning models.

Underfitting the Training Data

- **Understanding Underfitting:**
  - Underfitting happens when a model is too simple to capture the underlying patterns in the data, resulting in poor performance.
  - For instance, a linear model may underfit complex data because it lacks the flexibility to capture intricate relationships.
- **Addressing Underfitting:**
  - To mitigate underfitting and improve model performance, several strategies can be employed:

Select a More Powerful Model:

- Choose a model with more parameters or complexity that can better capture the underlying patterns in the data.
- For instance, using a polynomial regression model instead of a linear one may help capture nonlinear relationships.

Improve Feature Engineering:

- Enhance the quality of features provided to the learning algorithm.
- Feature engineering involves selecting or creating informative features that better represent the data and its underlying structure.

Reduce Model Constraints:

- Relax the constraints imposed on the model to allow for more flexibility in fitting the data.
- For instance, decreasing the regularization hyperparameter in regularized models can help reduce constraints and prevent underfitting.

Understanding and addressing underfitting is crucial for developing models that accurately capture the complexity of real-world data and make reliable predictions. By selecting appropriate models, improving feature engineering techniques, and adjusting model constraints, underfitting can be mitigated, leading to better-performing machine learning systems.

Stepping Back

Understanding the Big Picture:

Machine Learning Essence:

- Machine learning teaches machines to improve tasks by learning from data, rather than explicitly coding rules.

Types of ML Systems:

- ML systems vary, including supervised or unsupervised, batch or online, and instance-based or model-based.

Key Components of an ML Project:

Data Gathering:

- ML projects begin with gathering data into a training set, serving as the foundation for learning.

Learning Algorithm:

- The training set is then inputted into a learning algorithm, which adjusts model parameters (if model-based) or learns examples directly (if instance-based).

Factors Affecting System Performance:

Training Set Size:

- Small training sets may result in poor performance, emphasizing the importance of having sufficient data.

Data Quality:

- Data quality significantly impacts system performance; it must be representative, non-noisy, and relevant.

Model Complexity:

- Models should strike a balance—not too simple (underfitting) or too complex (overfitting)—to ensure accurate predictions.

Importance of Model Evaluation:

- After training, evaluating the model's performance is crucial for identifying shortcomings and fine-tuning to enhance predictions on new data.

Understanding these core concepts provides a sturdy foundation for navigating machine learning projects, facilitating effective learning, and constructing robust models capable of making reliable predictions.

Testing and Validating

Understanding Model Generalization:

Trial and Error Approach:

- To assess how well a model generalizes to new cases, it's crucial to test it on actual new cases.
- Putting the model into production and monitoring its performance is one approach, but it's risky if the model performs poorly.

Utilizing Training and Test Sets:

- A safer method involves splitting the data into two sets: the training set and the test set.
- The training set is used to train the model, while the test set is reserved for evaluating its performance.

Generalization Error:

- The error rate on new cases is termed the generalization error or out-of-sample error.
- By evaluating the model on the test set, you obtain an estimate of this error, indicating its performance on unseen instances.

Identifying Overfitting:

- If the training error is low (few mistakes on the training set) but the generalization error is high, it suggests overfitting.
- Overfitting occurs when the model fits too closely to the training data, resulting in poor performance on new data.

Tip

Tip for Dataset Splitting:

When splitting your dataset for training and testing, a common practice is to allocate 80% for training and reserve 20% for testing. Yet, this ratio can vary based on dataset size. For instance, in a dataset comprising 10 million instances, allocating just 1% for testing would result in a test set containing 100,000 instances. This substantial number of instances in the test set is likely adequate for obtaining a reliable estimate of the generalization error. Adjusting the split ratio accordingly ensures a robust evaluation of your model's performance

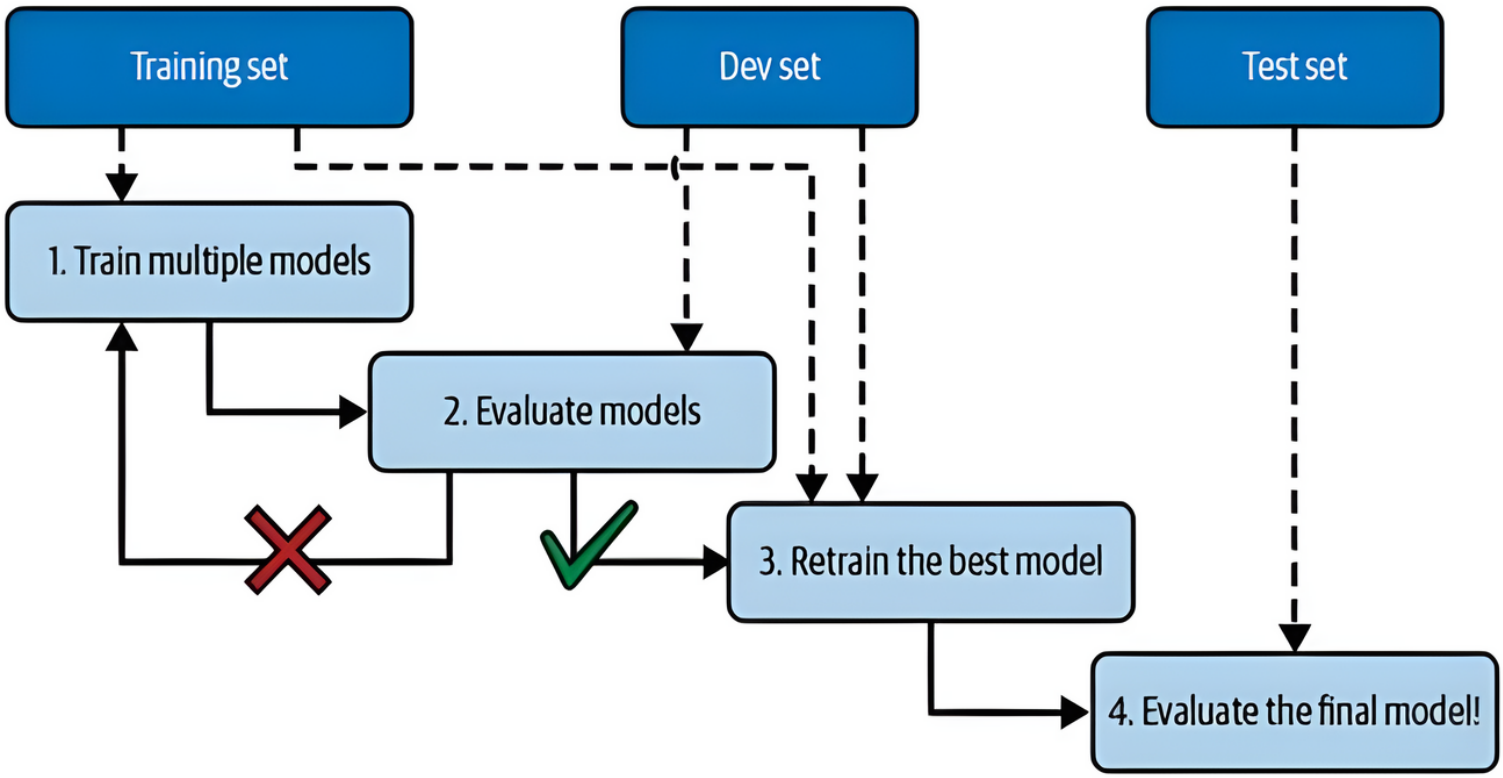
---

Hyperparameter Tuning and Model Selection

- **Comparing Models:**
  - Suppose you're unsure between two models, like a linear one and a polynomial one. To decide, you can train both and assess how well they perform using a test set.
- **Choosing Hyperparameters:**
  - Let's say the linear model performs better, but you want to add regularization to avoid overfitting. How do you pick the regularization value? You could train multiple models, each with a different value for this parameter, and select the one with the lowest generalization error.
- **The Pitfall:**

- Imagine you find the best model based on test set performance, but when put into action, it doesn't work as well. This occurs because the model was optimized specifically for the test set, not for new data.

- **Holdout Validation:**
  - A solution to this is holdout validation. Here, you split a portion of the training set to evaluate multiple models and choose the best one. This separated part becomes the validation set.

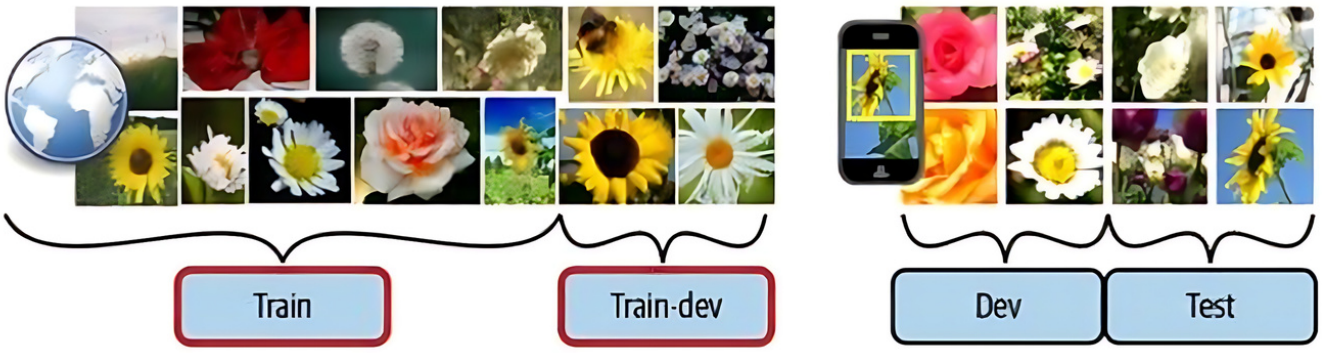


Model selection using holdout validation

- **Final Model Selection:**
  - After selecting the best model using the validation set, you train it on the full training set (including the validation set) to get the final model.
- **Estimating Generalization Error:**
  - Finally, you assess this final model on the test set to estimate its generalization error, which helps understand its performance on new data.
- **Cross-Validation:**
  - If the validation set is too small or large, it can affect model evaluation. Repeated cross-validation solves this by using multiple small validation sets, improving evaluation accuracy, albeit at the cost of increased training time.

Data Mismatch

- **Large Training Data, Representative Test Data:**
  - Sometimes, you might have a lot of training data, but it may not perfectly represent the real-world data you'll encounter. For instance, when building a flower recognition app, you can easily gather millions of flower images online, but they may not resemble the pictures taken with the actual app.
- **Ensuring Representativeness in Validation and Test Sets:**
  - Both the validation and test sets must mirror the real-world data to ensure accurate model evaluation. This means using only representative pictures, splitting them equally between validation and test sets, and ensuring no duplicates appear in both sets.
- **Challenges with Model Performance:**
  - If your model trained on web pictures performs poorly on the validation set, you won't know if it's due to overfitting or the mismatch between web and app pictures.
- **Introducing the Train-Dev Set:**
  - To address this uncertainty, Andrew Ng introduced the train-dev set. This set consists of some training pictures (from the web) held out separately.
- **Diagnosing Model Performance:**
  - After training on the main training set, you evaluate the model on the train-dev set. If it performs poorly, it likely overfits the training set, requiring simplification, regularization, more data, or data cleaning.
- **Addressing Data Mismatch:**
  - If the model performs well on the train-dev set but poorly on the dev set, the issue likely stems from data mismatch. Preprocessing the web images to resemble app pictures and retraining the model can help resolve this.
- **Final Evaluation:**
  - Once the model performs well on both train-dev and dev sets, it undergoes final evaluation on the test set to gauge its performance in real-world scenarios.



When real data is scarce (right), you may use similar abundant data (left) for training and hold out some of it in a train-dev set to evaluate overfitting; the real data is then used to evaluate data mismatch (dev set) and to evaluate the final model's performance (test set)

NO FREE LUNCH THEOREM

- **Model as Simplified Representation:**
  - A model simplifies data by discarding unnecessary details to generalize to new instances effectively.
- **Implicit Assumptions with Model Selection:**
  - When you choose a model type, you implicitly assume characteristics about the data. For instance, opting for a linear model assumes the data follows a linear trend, with deviations treated as noise.
- **No Free Lunch Theorem (NFL):**
  - David Wolpert's 1996 paper introduced the NFL theorem, stating that without assumptions about the data, no model is inherently superior to others. The best model varies based on the dataset, with no prior guarantee of one model performing better.
- **Need for Evaluation:**
  - Since evaluating all possible models isn't feasible, practical approach involves making reasonable assumptions about the data and assessing a few models.
- **Practical Evaluation Approach:**

- For simpler tasks, linear models with different levels of regularization may be evaluated, while complex problems may warrant assessment of various neural networks.


Exercises

1. How would you define machine learning?

2. Can you name four types of applications where it shines?

3. What is a labeled training set?

4. What are the two most common supervised tasks?

5. Can you name four common unsupervised tasks?

6. What type of algorithm would you use to allow a robot to walk in various unknown terrains?

7. What type of algorithm would you use to segment your customers into multiple groups?

8. Would you frame the problem of spam detection as a supervised

learning problem or an unsupervised learning problem?

9. What is an online learning system?

10. What is out-of-core learning?

11. What type of algorithm relies on a similarity measure to make predictions?

12. What is the difference between a model parameter and a model hyperparameter?

13. What do model-based algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?

14. Can you name four of the main challenges in machine learning?

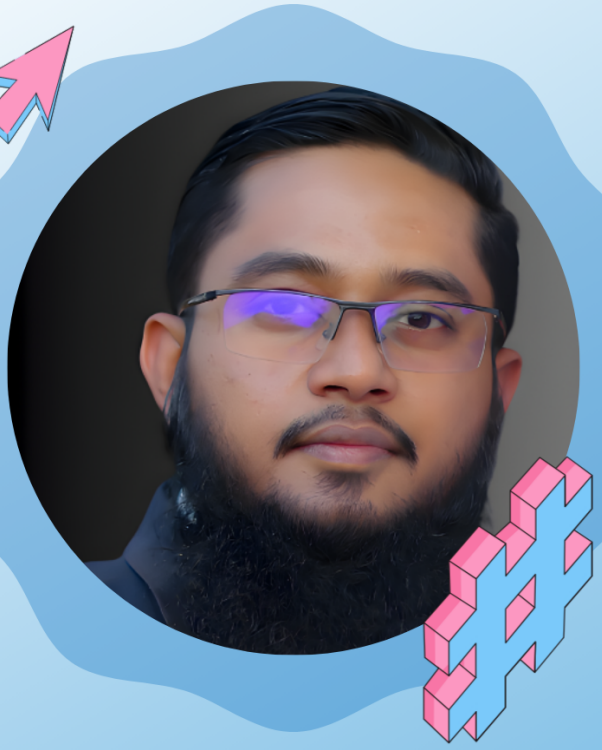
15. If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?

16. What is a test set, and why would you want to use it?

17. What is the purpose of a validation set?

18. What is the train-dev set, when do you need it, and how do you use it?

19. What can go wrong if you tune hyperparameters using the test set?



Thanks from:

m.m.isfam IMRAN

www.go2imran.com

mmiimran

go2imran6@gmail.com

@code_2_learn6666

www.go2imran.com