Machine Learning

CS SCHOLARS - PROGRAMMING

Learning Goals

Identify the three major categories of learning (**supervised**, **unsupervised**, and **reinforcement**) and the three major categories of reasoning (**classification**, **regression**, and **clustering**)

Describe how **training**, **validation**, and **testing** are used to build a model and measure its performance

Use the **scikit-learn** library to run pre-existing machine learning algorithms and generate models based on datasets

Machine Learning and Models

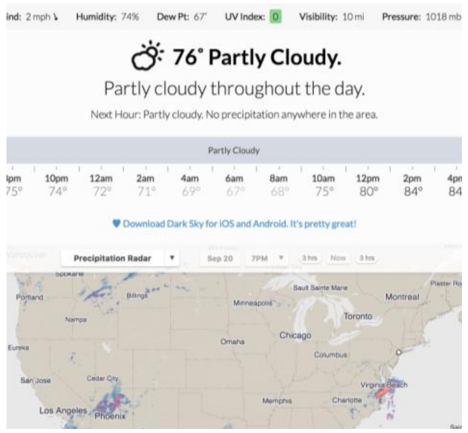
Machine Learning Creates Models

In machine learning we don't need to hand-design a model of the world based on a particular dataset. Instead, we apply a general machine learning algorithm to the data. This identifies patterns in the data and **generates the model automatically**. It's like outsourcing the work to the computer!

The special part of this process is that the programmer doesn't teach the algorithm the pattern to create a specific model; the programmer teaches the algorithm how to find patterns generally. This means that the computer can generate many different models on its own, given the right data.

Machine Learning in Many Contexts

Machine Learning generates models that are used in thousands of contexts across the world, including speech recognition, weather prediction, and medical diagnosis.



ind: 2 mph Humidity: 74% Dew Pt: 67° UV Index: 0 Visibility: 10 mi Pressure: 1018 mb

76° Partly Cloudy.

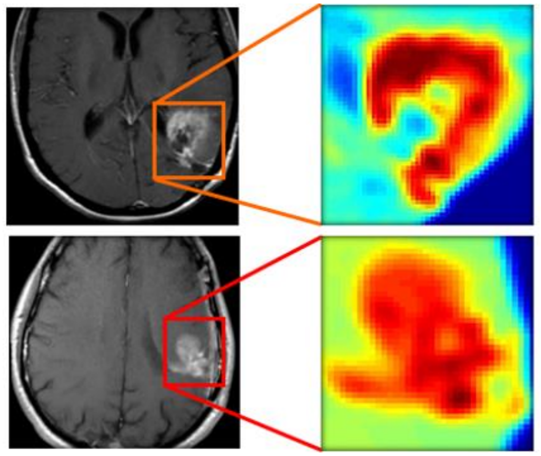
Partly cloudy throughout the day.

Next Hour: Partly cloudy. No precipitation anywhere in the area.

Partly Cloudy

```
pm   10pm  12am  2am  4am  6am  8am  10am  12pm  2pm  4pm
75°  74°   72°   71°  69°  67°  68°  75°   80°   84°  84
```

Download Dark Sky for iOS and Android. It's pretty great!



Models

The **model** generated by a machine learning algorithm is used to perform reasoning tasks.

Once a model has been generated, we can analyze it to gain insights about the original dataset. For example, we might build a model to group open-ended feedback provided in a country-wide survey, then use those groups to identify common trends among answers.

We can also apply a model to new data to make predictions. A model trained on market behavior could make a prediction about which products produced by a business might be most popular in an upcoming quarter.

Models are Based on Data

To train a model using machine learning, we must start with a large amount of **data**. The algorithm will try to find patterns and correlations in that data. A model is only as good as its dataset!

Each data entry must be broken down into a set of **features** so the machine learning algorithm can analyze features separately or in groups. These features can be **categorical** (discrete), or **numerical** (continuous). If you have a table of data, the features would be the **columns**; if you have a set of text, the features would be **words** or **groups of words**.

Learning Categories

Many Machine Learning Algorithms

There are a very large number of machine learning algorithms that computer scientists have designed which all generate models using different approaches. How do we choose which one to use?

Random Forests
Quantile Regression Forests
Expert Systems
XGBoost Gradient Boosting Mach
Boosted Generalized Additive M
Reinforcement Learning
Bagged Decision Trees
Naive Bayes
Manifold Machine Learning
(Multiple) Linear Regression
K-Nearest Neighbours
Boosted Regression Trees
Neural Networks
Generalized Additive Models
Gradient Boosting Machine Mode
Gaussian Mixture Models
Probabilistic Machine Learning
Multi-Agent Systems
Multivariate Adaptive Regressi
Support Vector Regression
Expert Systems
Reduced Error Pruning
Hidden Markov Models
Spatial Cluster Analysis
Bayesian Networks
Recurrent Neural Networks
Maximum Entropy
Regression Trees
(Hierarchical) Clustering
K-means
Logistic Regression
Decision Trees
Support Vector Machines
Convolutional Neural Networks
Model Based Boosting
Genetic Programming Model
Latent Dirichlet Allocation
Boosted Decision Trees

We can break algorithms into groups based on how they **learn** and on how they perform **reasoning**.

Three Ways to Learn

There are three primary ways that machine learning algorithms learn patterns from data. Which type you use depends on what type of problem you're trying to solve.

The three categories are **supervised learning**, **unsupervised learning**, and **reinforcement learning**.

Supervised Learning

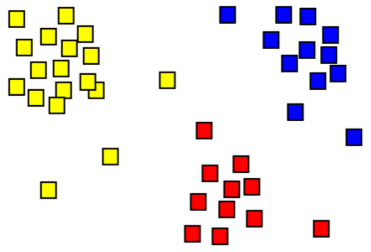
Supervised learning algorithms train models to predict outputs based on inputs. A data entry in the dataset is the input, and a **label** or a **score** is the output.

Example: to train a model to predict a person's favorite ice cream based on other desert preferences, each data entry (set of desert preferences) must be accompanied by a label (that person's favorite ice cream).

Once the model is fully trained, it can be used to make predictions about unlabeled data.

This is like when you learn by completing a homework assignment. You're trying to predict a correct answer (label) which is known by the teacher. This means the teacher (ML algorithm) can check your work. Later, you'll apply your knowledge to new problems where the answer is unknown.

Unsupervised Learning



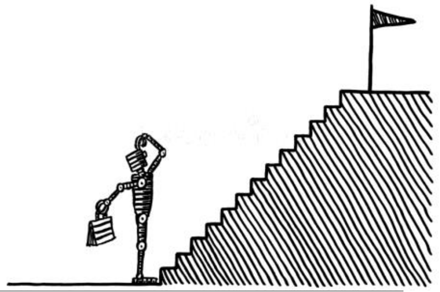
Unsupervised learning algorithms train models that infer the natural structure of a dataset. Which data entries are most similar to each other?

Example: identify common categories of favorite ice cream flavors by grouping together people who have similar desert preferences.

Unsupervised learning is often used when the data is **not** labeled – no true answer is known. That means that a human being will need to look over the model's results to see if they make sense or seem random.

This is like how researchers come up with theories to explain how the world works. Theories are based on data and may make sense, but might be changed of disproven later on.

Reinforcement Learning



Reinforcement learning algorithms train models that help an **artificial intelligence agent** approach an overall goal. Usually this involves taking a sequence of actions to bring the agent closer to the endpoint and giving feedback on those actions.

Example: train a robot to make ice cream by constantly changing the temperature and speed of an ice cream machine and measuring the consistency of the product until it's just right.

This is like playing a guessing game where the person with the secret word tells you if your guesses are hot or cold. You adjust your guesses and questions based on the person's feedback until you've gotten to the correct word.

Demo: Supervised Learning

We can build an image-recognition model using supervised learning by labeling a bunch of pictures as belonging to specific classes.

Try it out here: https://teachablemachine.withgoogle.com/train/image

Reasoning Categories

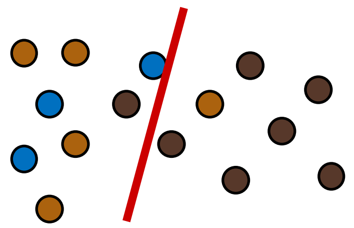
Many Ways to Reason

Once you've identified what type of learning you need to do, there are still lots of options for what type of algorithm you can use! One way to narrow the options down further is to ask what type of **reasoning** the algorithm will need to do.

For supervised learning, common reasoning approaches include **classification** and **regression** algorithms.

For unsupervised learning, common reasoning approaches include **clustering**.

Classification Algorithms

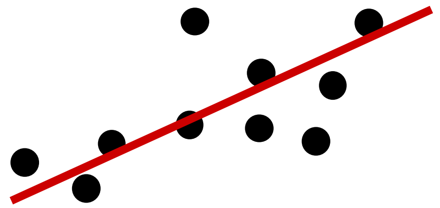


A **classification** algorithm takes labeled data of any type and produces a **model** (called a classifier) which determines which **category** a given data entry most likely belongs to. In other words, it produces a **categorical** (discrete) result.

For example, a classifier might take in data about an email (the sender, title, message content) and classify it as SPAM or NOT SPAM

Classification is commonly used in image recognition. Examples of classification algorithms include Naive Bayes, Decision Trees, and Support Vector Machines (SVMs).

Regression Algorithms

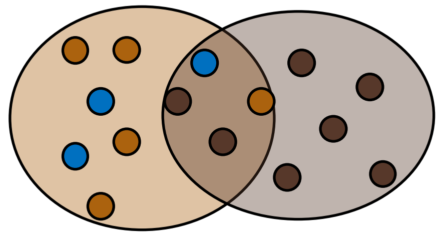


A **regression** algorithm takes numerically-labeled data and produces a function that maps data entries to **numerical** (continuous) results. Instead of predicting a specific category, it gives each data entry a **score**.

For example, a regression algorithm might take information about a house (GPS location, previous sell date, previous sell price, average price of nearby houses) and use it to predict its market value (a price).

Regression algorithms are used to predict things like stock market prices and weather temperatures. A linear regression is one kind of regression algorithm.

Clustering Algorithms



A **clustering** algorithm takes a set of data and produces a model that breaks the data into some number of **clusters** by identifying data entries that are similar to each other. We don't actually know the true categories or scores of the provided data, so it's hard to 'test' the resulting model, as we aren't sure what the actual result should look like.

For example, a clustering algorithm could recommend a new set of plant species by finding similarities between plant features in a given region.

Clustering is used in consumer analysis, data compression, and bioinformatics. K-means clustering is one kind of clustering algorithm.

Activity: Choose How to Reason

When you want to apply machine learning to a problem, investigate the type of problem and feature types to choose a method of reasoning.

You do: what combination of learning category + reasoning category would you use to...

... predict someone's favorite sport based on their job?

... investigate potential groupings of people based on their favorite restaurants?

... estimate how many meters a person walks per day based on their demographics?

Many Other Algorithms

Classification, regression, and clustering are just the most common types of reasoning algorithms. There are plenty of others that we don't have time to discuss in this class.

To make a model, though, you just need one algorithm- the best fit for the problem you're trying to solve.

Let's dive a bit deeper into how we can apply machine learning with supervised algorithms specifically.

Training a Model

ML Process: Decide, Train, then Test

To apply machine learning to a supervised task, follow a simple process

First: **decide** which learning algorithm you'll use, what it should produce and which features you'll train on. Make sure the algorithm matches the feature types!

Second: **train** a model created by the algorithm by providing it with the data. The algorithm will 'learn' from the data the same way a student learns by going over worked examples.

Third: **test** the model on a different set of data. This helps determine how accurate the model actually is.

Training Identifies Key Features

During the training process the algorithm identifies which features contribute the most to the underlying pattern. It does this by picking features that reduce the **error** that results from running the model on known data.

It's rare for a machine learning algorithm to identify a single feature that can definitively be used to answer a question. Usually, the algorithm uses a **combination of several features** which are weighted based on how well they correlate with the correct answer.

The algorithm needs to learn from a **lot** of examples to get a good sense of what the real pattern is.

Example: Is It a Dog?

Let's use a very simplified example to demonstrate how a machine learning algorithm might learn which features best match a label.

We want to train a model to recognize pictures of dogs. What kinds of features might it identify?

(The features we describe here are far too high-level, but the general idea is about right).

Example: Is It a Dog?

Guess: ?

Answer: ✓

Prior knowledge:

???

Updated knowledge:

-Has Pointy Ears

-Has Fur

-Screen in background

Example: Is It a Dog?

Guess: ✓?   Answer: ✓

Prior knowledge:

-Has Pointy Ears

-Has Fur

-Screen in background

Updated knowledge:

-Has Pointy Ears

-Has Fur

-Grey/Black Coloring

Example: Is It a Dog?

Guess: ✓ Answer: ✗

Prior knowledge:

-Has Pointy Ears

-Has Fur

-Grey/Black Coloring

Updated knowledge:

???

Testing a Model

Training Data, Validation Data, Testing Data

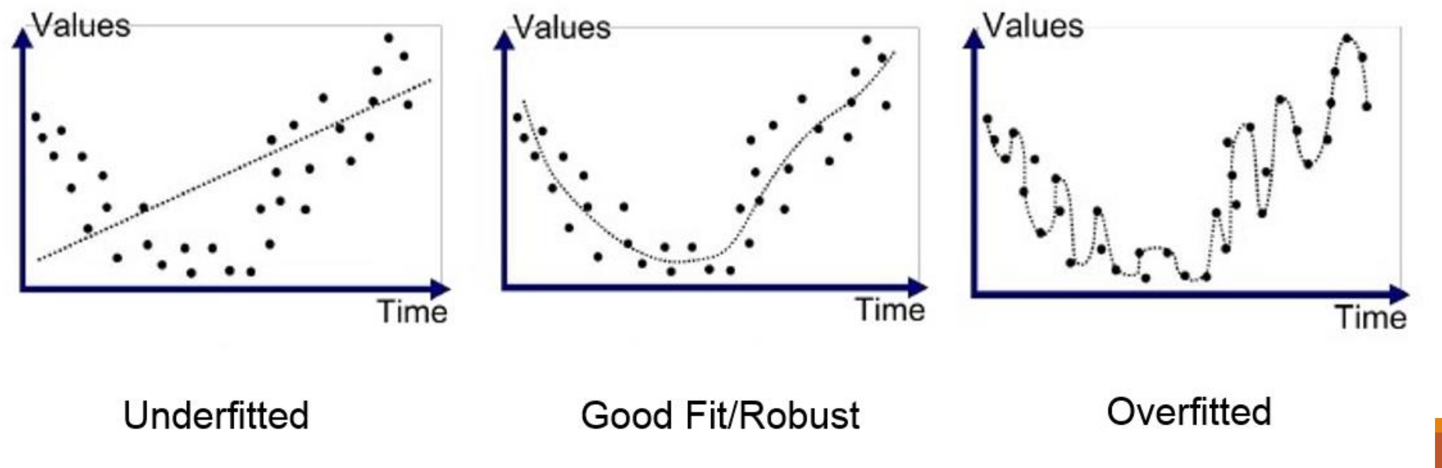
Once we've trained a model, we can use that model to make predictions about data. That means we don't want the model to **only** work on the data we provided originally - we want it to work on future data too.

At the beginning of the process, we separate our data into two sets: **training** data vs. **testing** data. When building algorithms professionally, you often want to create a third data set as well: the **validation** data set. This data set is used to help make the training work as well as possible before the final test is done.

Too Much Data Can Cause Overfitting

The **training data** is normally composed of the majority (maybe 70%) of the available dataset. This data is run through the machine learning algorithm to produce the model. The more training data there is, the more **accurate** the algorithm's model becomes.

This can go wrong if the algorithm over-optimizes the model. For example, it might identify a pattern that only exists within the training data, not in the general population – the pattern might just be noise. This is called **overfitting**. Overfitting can result in a model performing very well on training data, but poorly on new data gathered later on.



Validation Data Identifies Overfitting

To detect and remove parameters in the model that cause overfitting, you can use **validation data**. This is a subset of the data (maybe 15%) that is **not** used when training the model. Instead, it will be used to **validate** the model during training.

The programmer repeatedly evaluates the model produced by the algorithm on the validation data to see how accurate it is. They can then modify certain parameters that are used by the algorithm to see which combination produces a model that works well on the training data *and* the validation data.

This is similar to how a teacher may provide students a set of practice problems to learn on, then provide a practice quiz with different problems where the students can test their knowledge.

Testing Data Provides Final Results

When the programmer thinks they've achieved an optimal model, the **testing data** is used to determine how accurate that model actually is. This is a portion of the data (maybe 15%) that was set aside at the beginning and never used during the training or validation process.

Unlike the validation data, which is evaluated multiple times, the model is run on the test data only **once**. We measure how close the predicted results are to the actual results. That score is the accuracy of the model.

You cannot train on your testing data if you want a fair test of the model!!!

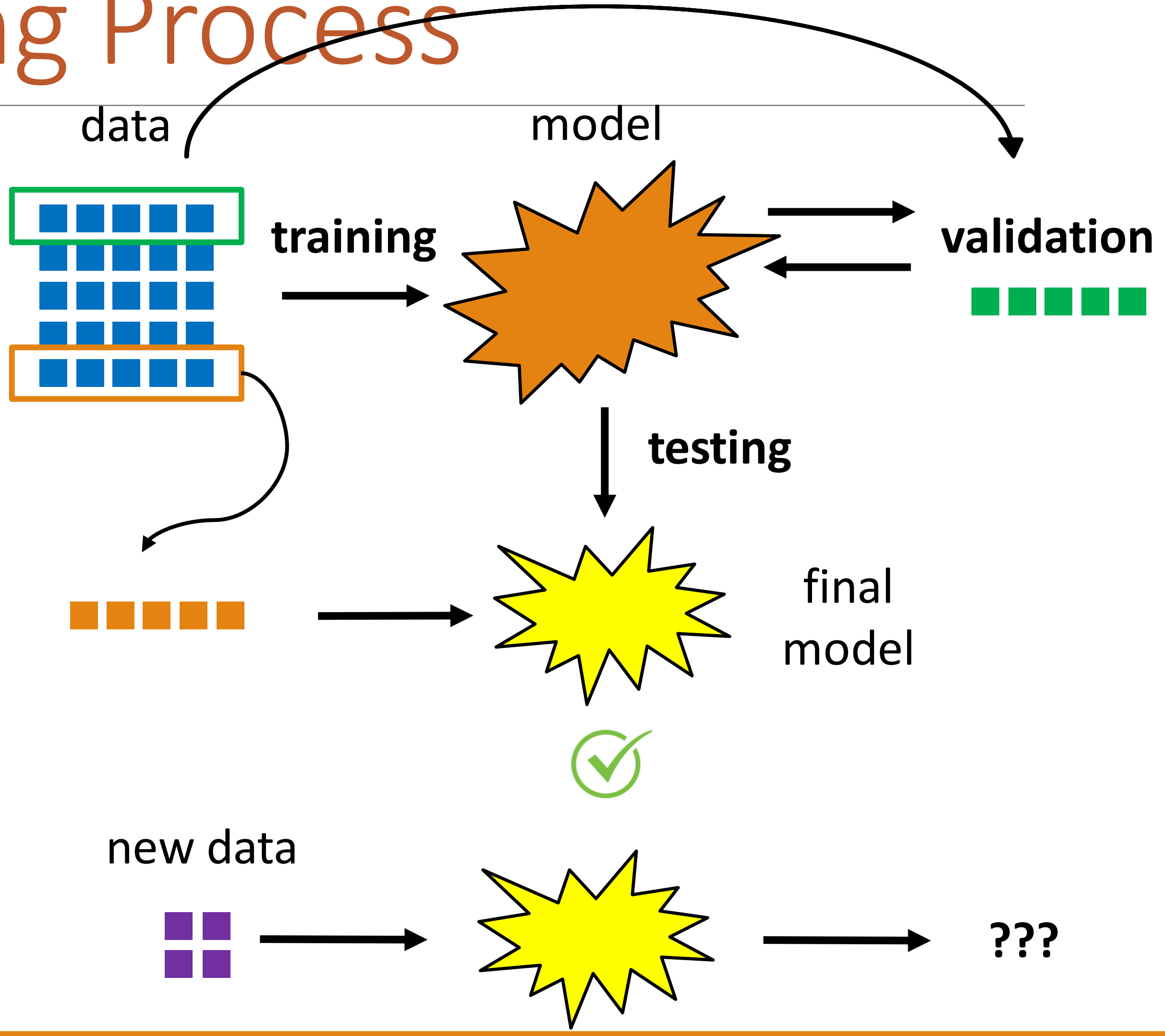
Example: Bad Training Process

What happens if we use our validation and test data to train as well?

The algorithm will get the opportunity to observe patterns in the additional data. It will optimize the model to include those patterns, even through validation.

When the model is tested, it will of course be accurate because the model was optimized to notice the correct patterns.

But if we try to use the model on new, unlabeled data later on, the patterns may no longer be valid. We don't know for sure because all the labeled data was used for training.



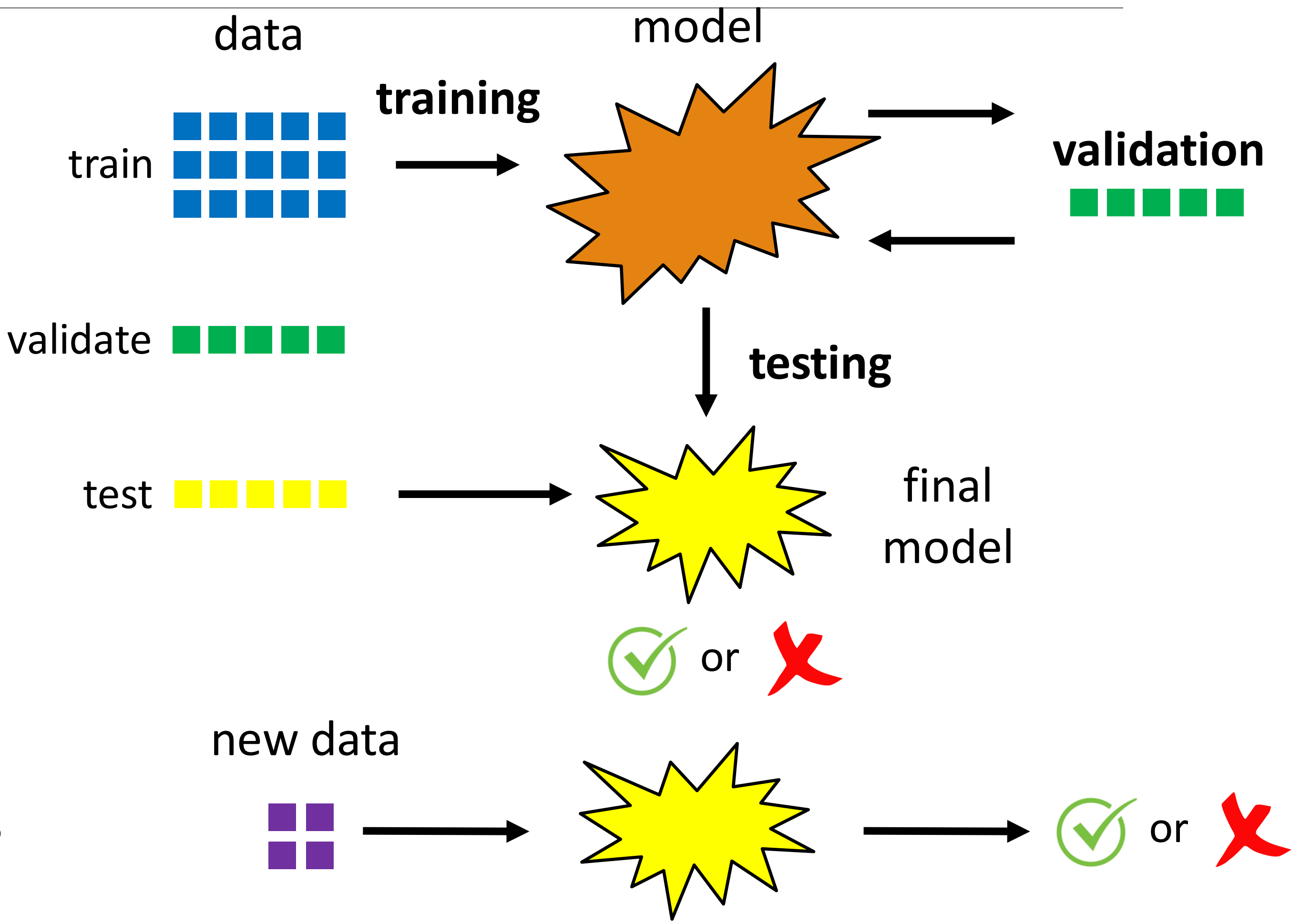
Example: Good Training Process

A better process: split the data into training, validation, and testing sets.

We'll train on **only** the training set and repeatedly test on the validation set. This should remove some of the overfitting from the training data.

When we're done, we'll test on the test set **once**. That produces our final result. It might be good, or it might be bad; it depends on how the model turned out.

However, the new data should have about the same accuracy as the test data, since the model never saw the test data before. There's less uncertainty.



Real-World Example: ChatGPT

Let's consider the popular chatbot ChatGPT. How was this system trained?

ChatGPT was trained to generate sequences of text using a HUGE dataset based on the internet and a large corpus of books. This taught the chatbot which words most often go together and allowed it to **generate new text**.

ChatGPT was trained in how to communicate using an advanced algorithm that combined **supervised learning** and **reinforcement learning**.

- For supervised learning, the programmers created example conversations with human and chatbot participants to serve as training data
- For reinforcement learning, ChatGPT generated several possible responses and humans ranked them
- Even now, ChatGPT is still being trained! When you thumbs-up or thumbs down a response, that's treated as a positive/negative indicator by the reinforcement learning system.

Machine Learning with scikit-learn

Real Example: Favorite Ice Cream Model

Now let's try building a real model with machine learning!

We have a dataset of people's favorite ice cream flavors (1st, 2nd, and 3rd). Our goal is to predict a person's 3rd favorite ice cream based on their 1st and 2nd preferences.

There are a ton of possible flavors, so we'll **hand code** the data to group similar flavors together and reduce the number of options to predict. This results in 6 possible categories: chocolate, coffee/tea, cookie, fruit, vanilla, and other.

Our approach is **supervised** (we already know what people's favorites are) and predicts a **categorical** result (an ice cream flavor). We need to use a classification algorithm.

Machine Learning Libraries

You can implement a machine learning algorithm yourself, but it isn't usually necessary. There are lots of great **external libraries** that have already implemented machine learning algorithms for you. All you have to do is plug in data!

We'll use the library **scikit-learn** here. You can install it in Thonny by going to Manage Packages and searching 'sklearn', or by running `pip` with:

```
pip install sklearn
```

Naive Bayes Classifier

Naive Bayes is a classification algorithm. The classifier it creates uses **conditional probabilities** to predict categories.

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{B}|A) \cdot P(A)}{P(B)}$$

When Naive Bayes is given a new data entry X, it uses the data it has already collected to determine the **probability** that X belongs to each possible category.

For each category C, it checks how **probable** it is that each feature in X occurs with C by checking how often it occurred in past data entries. By multiplying these probabilities together, it can form a general probability that the data entry belongs in that category.

$$P(C|X) = P(X_1|C) * P(X_2|C) * P(X_3|C) * P(C)$$

The category with the highest probability wins!

Naive Bayes Example

Say we've collected the following data:

- #1: Chocolate
  #2: Cookie
  #3: Chocolate
- #1: Chocolate
  #2: Coffee/Tea
  #3: Vanilla
- #1: Cookie
  #2: Cookie
  #3: Vanilla
- #1: Coffee/Tea
  #2: Vanilla
  #3: Chocolate
- #1: Cookie
  #2: Vanilla
  #3: Chocolate
- #1: Coffee/Tea
  #2: Vanilla
  #3: Vanilla

If we encounter a new data point where the #1 favorite is Chocolate and the #2 favorite is Vanilla, we'd calculate the odds of each class as follows:

- P(#3 Chocolate | #1 Chocolate, #2 Vanilla) =
  P(#1 Chocolate | #3 Chocolate) * P(#2 Vanilla | #3 Chocolate) * P(#3 Chocolate) =
  1/3 * 2/3 * 1/2 = 1/9

- P(#3 Vanilla | #1 Chocolate, #2 Vanilla) =
  P(#1 Chocolate | #3 Vanilla) * P(#2 Vanilla | #3 Vanilla) * P(#3 Vanilla) =
  1/3 * 1/3 * 1/2 = 1/18

Chocolate has a higher probability than Vanilla, so Naive Bayes would predict that classification for the entry.

Reading The Data

First, we need to read the data and reformat it to just include the parts we need.

```
f = open("all-icecream.csv", "r")
text = f.read()
f.close()
```

Columns 7-9 contain the coded groups; extract them from the file.

```
lines = text.split("\n")
data = []
for i in range(1, len(lines)):
    lines[i] = lines[i].split(",")
    if len(lines[i]) >= 10:
        data.append(lines[i][7:10])
```

Naive Bayes in sklearn

In `sklearn`, we can employ Naive Bayes with the function `sklearn.naive_bayes.CategoricalNB()`.

This will create an 'empty' model. We can then train that model on data using `model.fit(data, labels)`

Once we've trained the model, we can run it on new data using `model.predict(newData)`.

Rearrange Data

However, we'll need to rearrange our data to make it fit into `sklearn`'s expectations! `CategoricalNB` wants integers, not strings. We can solve this by mapping each category to a number.

0: coffee/tea	1: chocolate	2: cookie
3: other	4: vanilla	5: fruit

```
groupNames = []
for i in range(len(data)):
    for j in range(len(data[i])):
        if data[i][j] not in groupNames:
            groupNames.append(data[i][j])
        data[i][j] = groupNames.index(data[i][j])
```

Setting Aside Test Data

Before training, we need to split our data into two groups: **training** data and **testing** data. That will make it possible for us to check our work later on.

Let's randomize the order of the data and split off 20% to test on. Next, separate the features (#1 and #2 preferences) from the label (#3 preference) that we're trying to predict.

```
import random
random.shuffle(data)
testSize = len(data) // 5
testData = data[:testSize]
trainData = data[testSize:]

features = []
labels = []
for i in range(len(trainData)):
    features.append(trainData[i][:2])
    labels.append(trainData[i][2])
```

Training the Model

Once the data is organized, training the model is easy. Just generate a new instance of the model using `CategoricalNB`, then call the `fit` method on that model using the data.

```
from sklearn.naive_bayes import CategoricalNB


model = CategoricalNB()

model.fit(features, labels)
```

We can use the trained model to make predictions on new data points. For example, let's say we want to predict someone's 3rd favorite if their 1st favorite is chocolate and 2nd favorite is cookie. Create a list of the two category numbers (1 and 2), put that in another list, and call the `predict` method on the model.

```
print(model.predict([ [1, 2] ])) # [1]
```

Perhaps the function returns 1, which corresponds to the chocolate category. A reasonable guess!

Testing the Model

To test our model in the code example, we just need to call the `score` method on the model using the test data we set aside earlier.

The result will be an accuracy between [0, 1]. We could get an accuracy of 25.6% - not great, but also not terrible considering the amount of variety in the possible answers.

Note that we had to set aside the test data first for this to work! If we trained on the test data, we'd get a faulty result.

```
testFeatures = []

testLabels = []

for i in range(len(testData)):

    testFeatures.append(testData[i][:2])

    testLabels.append(testData[i][2])


accuracy = model.score(testFeatures, testLabels)

print(accuracy) # 0.25595238095238093
```

Learning Goals

Identify the three major categories of learning (**supervised**, **unsupervised**, and **reinforcement**) and the three major categories of reasoning (**classification**, **regression**, and **clustering**)

Describe how **training**, **validation**, and **testing** are used to build a model and measure its performance

Use the **scikit-learn** library to run pre-existing machine learning algorithms and generate models based on datasets