Can a Machine Tell Pokmon Types Apart?

SBA MODA Workshop A Gentle Introduction to Machine Learning

Dr. Harshvardhan

School of Business Administration
American University of Sharjah

Spring 2026

Who am I?

Dr. Harshvardhan (you can call me Dr. Harsh or Prof. Harsh)

Assistant Professor of Information Systems & Analytics
School of Business Administration, AUS

My Teaching:

- ISA 201 Introduction to MIS
- ISA 383 Python for Business Analytics
- BDA 625 Data Mining & Machine Learning

Before AUS:

- PhD, University of Tennessee, Knoxville
- MBA, IIM Indore
- Strategic Planning & Modeling, HP

Research: machine learning, LLMs, applied analytics for business problems.

Outside work:
Interested in coffee, coding, reading and writing.

harshvardhan@aus.edu

www.harsh17.in

Welcome!

Today we will build, together, a small piece of machine learning.

- No prior coding experience required
- No prior machine-learning experience required
- Bring curiosity, a laptop, and a browser

Our goal: teach a computer to look at a Pokmon and guess whether it is Fire, Water, or Grass.



Part 1

What *is* Machine Learning?

Artificial Intelligence is already in your pocket

AI = computers doing things that look intelligent.

Examples you use every day:

- Gmail filtering spam
- Netflix recommending shows
- Face unlock on your phone
- Google Translate
- ChatGPT

Examples in business:

- Bank fraud detection
- Credit-score prediction
- Demand forecasting
- Chatbots in customer service
- Pricing optimisation

Machine Learning is how modern AI learns

The old way: programming

Write rules by hand:

```
IF email has ‘‘free money’’
AND lots of !!!
THEN it is spam
```

Brittle. Spammers change tricks. New rules needed every week.

The new way: machine learning

Show the computer 100,000 emails labelled *spam* / *not spam*.

It *learns* the pattern itself.

Adapts as spammers evolve.

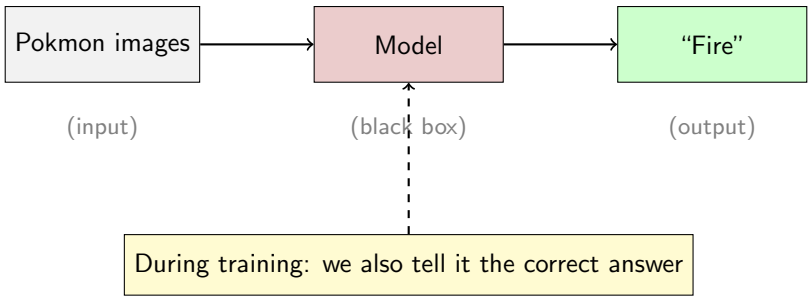
What is “data” anyway?

Data is just **numbers organised in a table**.

Pokmon	Avg R	Avg G	Avg B
Charmander	240	200	150
Squirtle	180	210	240
Bulbasaur	200	230	180

Each **row** is one example.
Each **column** is one feature.

Supervised learning, in one picture



The model adjusts itself to match the labels we provide.

The most important idea: train vs. test

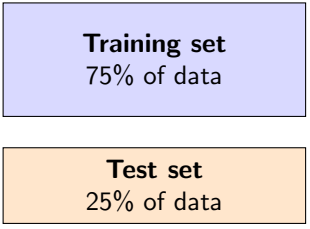
Analogy: studying for an exam.

You study from a textbook (the *training set*).

Then you take the exam, where the questions are *new* (the *test set*).

If we let the model peek at the exam, of course it gets 100%. That tells us nothing.

Always hold out unseen data for the final test.



Part 2

How a Computer Sees an Image

Pictures are secretly grids of numbers

What you see

What the computer sees
(10×10 patch, RED channel)

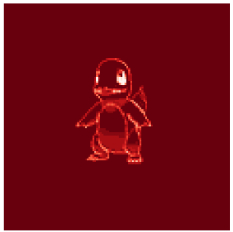
221	221	221	221	238	187	153	255	255	255
221	221	221	221	238	255	187	187	255	255
238	238	238	255	255	255	255	170	221	255
255	255	255	255	255	255	255	221	153	255
255	255	255	255	255	255	255	255	187	255
255	255	255	255	255	255	255	255	238	221
255	255	255	255	255	255	255	255	255	187
255	255	255	255	255	255	255	255	255	170
255	255	255	255	255	255	255	255	255	187
255	255	255	255	255	255	255	255	255	238

Each pixel is a brightness value from 0 (black) to 255 (full colour).

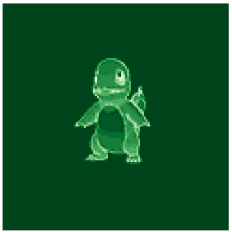
Three colour channels: Red, Green, Blue

Original

Red channel



Green channel



Blue channel

Stack the three together $\rightarrow$ full-colour image.

One image = a LOT of numbers

$$120 \times 120 \times 3 = \textbf{43,200 numbers}$$

(height width colour channels)

That's a lot of numbers per Pokmon.

A simple model with so many inputs would either:

- train painfully slowly, or
- just memorise the training images and fail on new ones.

We need to summarise.

Part 3

Features: turning images into a few numbers

A feature is a summary number

Idea: replace the 43,200 pixel values by a small list of *descriptive* numbers.

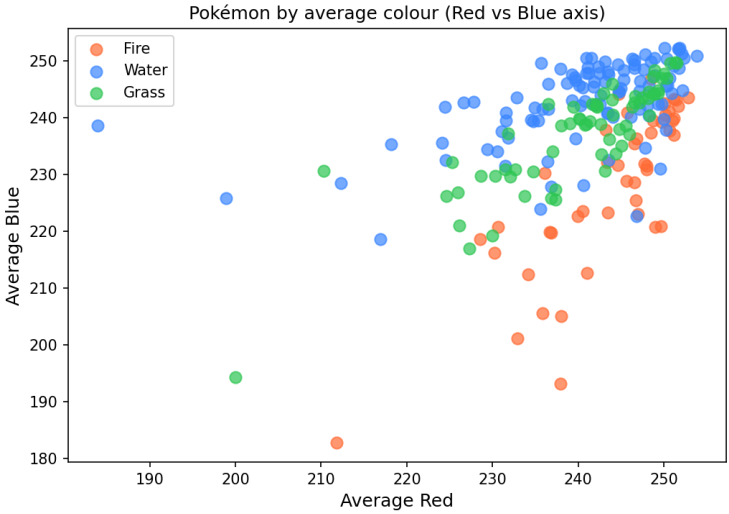
For Pokmon, the simplest meaningful summary is the **average** amount of red, green, and blue across all pixels.

Charmander
R=251 G=248 B=243

Squirtle
R=244 G=248 B=248

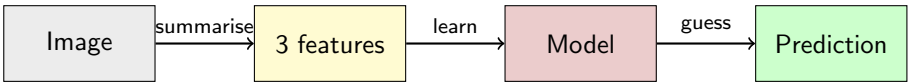
Bulbasaur
R=244 G=249 B=246

Why this works for Pokmon



Just two of the three colour averages already separate the types nicely.

The complete pipeline

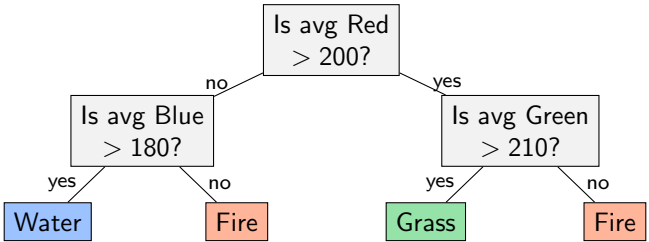


Feature engineering is the art of choosing those summary numbers wisely.

Part 4

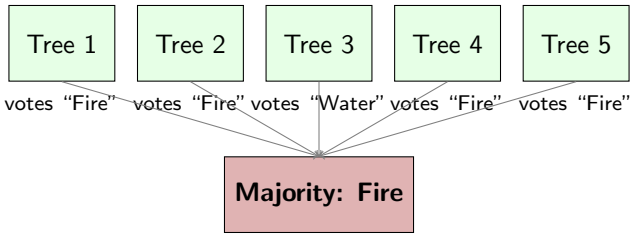
Two kinds of models we will try

Decision tree: a flowchart of yes/no questions



The computer figures out the questions and the thresholds itself.

Random Forest: many trees voting



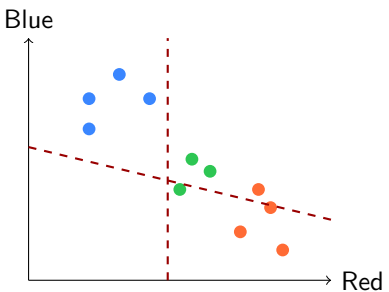
- Build many decision trees, each on a different slice of the data
- Each tree votes on a new Pokmon
- The majority answer wins
- Almost always more accurate than a single tree

Logistic Regression: draw straight lines between classes

Imagine plotting every Pokmon as a dot in feature space (e.g. Red on x-axis, Blue on y-axis).

Logistic Regression *draws straight lines* that separate the classes.

Despite the name, it does **classification**, not regression.



Trees versus lines

Random Forest

- Many flexible trees
- Captures complex non-linear patterns
- Needs more data + features
- Harder to interpret

Logistic Regression

- Single straight-line model
- Best when classes *are* roughly linearly separable
- Fast and simple
- Easy to explain

Which one wins for Pokmon? Let's find out in the notebook.

Part 5

How do we know if it worked?

Accuracy: the simplest score

$$\text{Accuracy} = \textbf{number correct} \;/\; \text{total}$$

Out of 55 test Pokmon:

- 42 right $\rightarrow$ **76% accuracy**
- 30 right $\rightarrow$ **55% accuracy**

Easy to compute. Easy to explain.

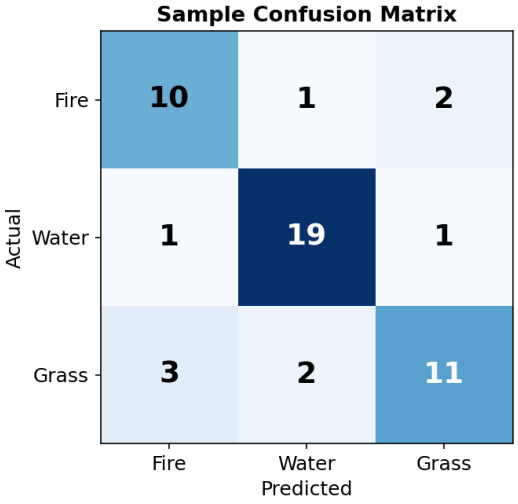
Accuracy hides where the model failed

A model with 76% accuracy got 24% wrong. **Where** did the mistakes come from?

- Did it confuse Fire with Water? (the colours are far apart)
- Did it confuse Grass with Water? (some plants are blue-ish)
- Did it always predict "Water" because that's the most common type?

A single number can't tell us. We need a **confusion matrix**.

Confusion matrix



- **Diagonal** = correct predictions
- **Off-diagonal** = mistakes
- **Rows** = the actual type
- **Columns** = what the model predicted

Lets us see exactly which classes the model gets confused between.

Part 6

Tuning the model

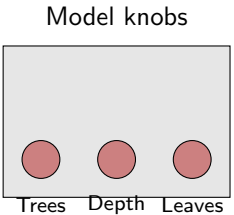
Models have knobs: hyperparameters

Random Forest knobs:

- `n_estimators` how many trees?
- `max_depth` how deep can a tree grow?
- `min_samples_leaf` how few examples in a leaf?

Logistic Regression knobs:

- `C` how strongly to penalise complex boundaries?
- `max_iter` how long to keep optimising?

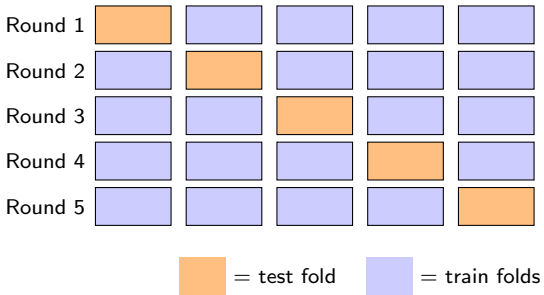


Different settings → different performance.

Cross-validation: a fairer test during training

Split the training data into 5 chunks. For each setting:

1. Train on 4 chunks, test on 1
2. Repeat 5 times, rotating which chunk is held out
3. Average the 5 scores



GridSearchCV: try every combination automatically

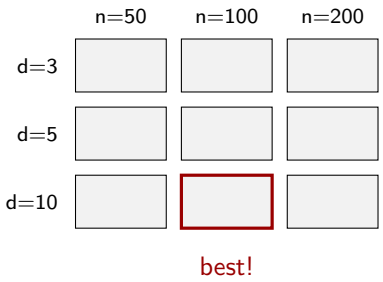
We give the computer a **grid** of values:

`n_estimators`	50, 100, 200
`max_depth`	3, 5, 10

That's $3 \times 3 = 9$ combinations.

For each one, it runs 5-fold cross-validation:
$9 \times 5 = 45$ small experiments.

It returns the best-scoring combination.



Part 7

Two big lessons before we open the notebook

Lesson 1 Tuning is useful but not magic

Hyperparameter tuning often gives you a small bump.

Sometimes it gives you nothing at all.

Occasionally it makes things worse.

Don't expect tuning to rescue a poorly-chosen model.

Lesson 2 Picking the right model often matters more

A simple model that fits the problem
beats a complex model that doesn't.

Logistic Regression >
Random Forest for our Pokmon problem.

Always try a simple baseline first.

Time to code!

Open the notebook in Google Colab
and we will run through it together.

`github.com/harshvardhaniimi/pokemon-workshop`

What you have learned today

1. AI & ML are everywhere; ML learns from data instead of explicit rules
2. Always split your data into a training set and a test set
3. An image is just a grid of numbers (R, G, B per pixel)
4. Features are summary numbers that describe each example
5. Decision trees, Random Forests, and Logistic Regression are three classic model families
6. Accuracy + confusion matrix tell you *how well* and *where* a model fails
7. GridSearchCV with cross-validation tunes hyperparameters honestly

Most important takeaway: picking the right model that fits your problem matters more than tuning a fancy one.

Bonus material on PCA is at the end of the notebook try it at home.

Questions?

harshvardhan@aus.edu

School of Business Administration
American University of Sharjah