AI - Classical and Machine Learning

CS 246

Objectives

- Students will be able to:
  - enumerate the main categories of AI
  - understand basic terminology (so they can carry on a coherent conversation during a job interview)
  - speak, from personal experience, about the advantages and limitations of AI (so they can carry on a coherent conversation during a job interview)
  - explain the basics of how machine learning works
  - describe how neural networks work

The ACM's Take on AI

Generative AI is expected to have substantial impact on several aspects of the software process, including (but not limited to) development of new code, comprehension of complex logging and debugging artifacts, static analysis, and code reviewing. The most understandable and visible change is likely to be in the development of (rote) new code—assistance technologies like GitHub's Copilot and other advanced auto-complete mechanisms can meaningfully improve development time, to a point. Effective use of such tools requires a deeper investment in design and code comprehension, while potentially decreasing the need for hands-on programming time. Similar advances in static analysis and code review are anticipated to have a meaningful impact on code quality and clarity, ideally also reducing the impact of implicit bias by increasing consistency and quality of comments and diagnostics

-- ACM Curricular Guidelines, p450

Symbolic AI

- Dominant form of AI from the 1950s-1980s
- Core idea: intelligence can be achieved by manipulating symbols according to logical rules (rather like how you might solve a math problem)
- Relies on knowledge bases filled with facts and rules
- An inference engine is used to draw conclusions and solve problems

Symbolic AI - An Example

Knowledge Base (Rules):

- IF engine_turns_over = NO AND lights_dim = YES THEN battery_dead = YES
- IF battery_dead = YES THEN problem = "Replace battery"
- IF engine_turns_over = YES AND fuel_gauge = EMPTY THEN problem = "Add fuel"
- IF engine_turns_over = YES AND fuel_gauge = NOT_EMPTY THEN problem = "Check spark plugs"

Facts (Input):

- engine_turns_over = NO
- lights_dim = YES
- fuel_gauge = HALF_FULL

Reasoning Process:

1. System checks: engine_turns_over = NO AND lights_dim = YES
2. Rule 1 matches → concludes battery_dead = YES
3. System checks: battery_dead = YES
4. Rule 2 matches → concludes problem = "Replace battery"

Output: "Replace battery"

Pros and Cons of Symbolic AI

- Pros
  - Easy to interpret and explain to others
  - Works well with domains with clear rules
  - Logic can be verified and debugged manually
- Cons
  - If you forget a rule, it fails completely
  - Manually writing rules doesn’t scale well to messy domains
  - It can’t improve from experience
  - Struggles with ambiguity or uncertainty (e.g., natural language, pattern recognition)
- Still in use in some quarters (games, simulations, planning systems for robotics) but largely overshadowed

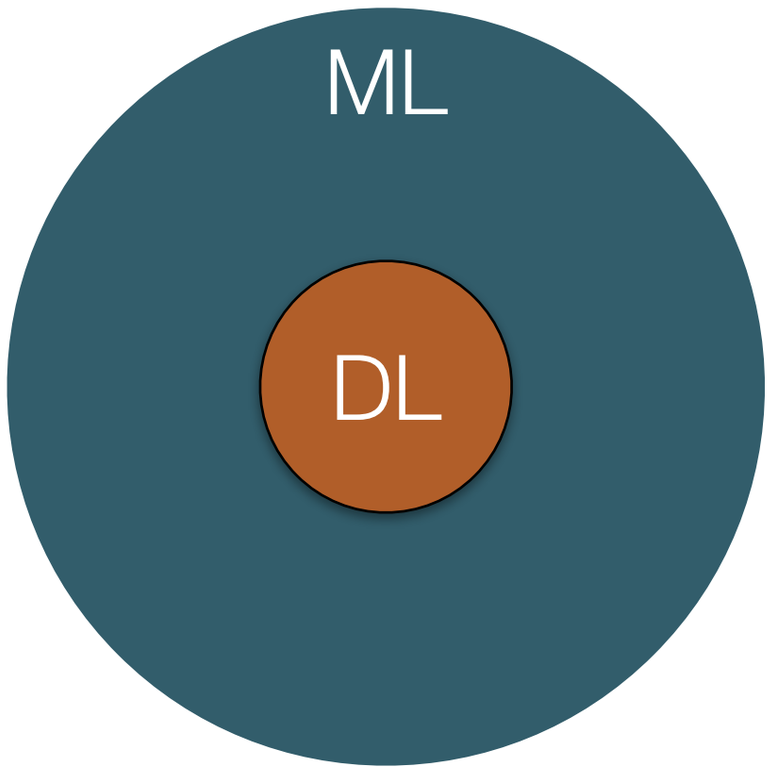
Examples of Symbolic AI

- MYCIN, diagnosed infections and recommend antibiotics
- PROSPECTOR, evaluated sites for potential mineral deposits
- INTERNIST-1, medical diagnosis
- TurboTax (still in use) applies rules to calculate taxes

Machine Learning

- In Machine Learning (ML), new knowledge is acquired by examining *data*, rather than applying facts to a hardcoded set of rules and an inference engine

- It is a broad field that encompasses what everyone know thinks of as AI -- **Deep Learning**



- In classical ML (that is, ML ∩ DL'), models are used to
  - predict discrete values (**classification**)
  - predict continuous values (**regression**)
  - group data (**clustering**)
- The first two are examples of supervised learning, the latter is unsupervised (not discussed here, but read the zyBook for details)

George Box: "All models are wrong, but some are useful"

Machine Learning Models

- A Machine Learning model is a **function** that takes in various inputs (called **features**) and produces an output - a discrete value (called a **label**), or a continuous value (called a **target**)
- Discrete Output: Predicting Vehicle Preferences
    - f(age, income, gender, education) might return a label, "car", "truck", "suv", "ev", "van", depending on their choice in vehicle.
- Discrete Output: Sinking of the Titanic
    - f(pass_class, gender, age, fare) might return a label, "died" or "lived", depending on whether they died or survived.
- Continuous Output: Real estate
    - f(area, lot_size, num_floors) might return a target, the predicted price of a house in Melbourne, Australia (or wherever)
- The functions will be based on the value of various parameters: the goal of ML is to determine values such that the model is as accurate as possible

Supervised Learning

1. Gather a dataset - where we know both the features (inputs) and the label or target (outputs) that the model *should* produce

2. Split the dataset into a training dataset (typically 70-80%) and a test dataset (20-30%)

3. Select the type of model we want to train.

4. Train the model by using the training dataset's features and labels

    1. Compute f(features) for each example

    2. Compare the prediction to the true label/target

5. Adjust the model's parameters to reduce prediction error

6. Repeat training and adjustment (optimization) until we are satisfied with the model's performance.

7. Run the now-trained model on the test set to see how well it generalizes to unseen data

    - This avoids "overfitting" -- where the model basically "memorizes" training data, rather than recognize general patterns

Predicting Discrete Values - an Example

- Suppose we want to predict how likely it was that a person would buy a particular type of vehicle based on various features
- The training dataset is shown at left. The features are age, income, gender, education (a, i, g, e). We *know* what vehicle they own, so this is *labeled* training data (and the label is the vehicle type)
- We perform steps 4 and 5, and based on the values of the parameters, get the center results
- We then tweak the parameters, rerun the model, and see if we can do better (results at right)
- We repeat the process over and over -- computers never tire -- until the error rate is acceptable
- Finally we run the model on some test data, not part of the training dataset

Person	Age	Income ($K)	Gender	Education	Vehicle
A	25	30	Male	High School	Car
B	42	80	Female	Bachelor's	SUV
C	36	60	Male	Master's	Truck
D	29	45	Female	Associate's	EV
E	54	100	Male	High School	Truck
F	39	75	Female	Bachelor's	SUV
G	31	55	Male	Bachelor's	Car
H	22	25	Female	High School	EV
I	47	90	Male	Master's	Van
J	33	68	Female	Master's	Car

Training Dataset

Person	Age	Income ($K)	Gender	Education	Vehicle	Predicted Vehicle - 1
A	25	30	Male	High School	Car	Car
B	42	80	Female	Bachelor's	SUV	Car
C	36	60	Male	Master's	Truck	EV
D	29	45	Female	Associate's	EV	EV
E	54	100	Male	High School	Truck	Truck
F	39	75	Female	Bachelor's	SUV	Van
G	31	55	Male	Bachelor's	Car	SUV
H	22	25	Female	High School	EV	EV
I	47	90	Male	Master's	Van	EV
J	33	68	Female	Master's	Car	Van

Iteration 1

Person	Age	Income ($K)	Gender	Education	Vehicle	Predicted Vehicle - 2
A	25	30	Male	High School	Car	Car
B	42	80	Female	Bachelor's	SUV	Car
C	36	60	Male	Master's	Truck	EV
D	29	45	Female	Associate's	EV	EV
E	54	100	Male	High School	Truck	Truck
F	39	75	Female	Bachelor's	SUV	SUV
G	31	55	Male	Bachelor's	Car	SUV
H	22	25	Female	High School	EV	EV
I	47	90	Male	Master's	Van	Van
J	33	68	Female	Master's	Car	Van

Iteration 2

Output Mechanisms for Classification Models

- All classification models involve internal computations,, producing scores, that then have to be turned into a final prediction. There are two approache
- **Argmax: Score-based classification**
  - Score is computed for each class: $f(a, i, g, e) = (s_{car}, s_{SUV}, s_{Truck}, s_{EV}) = (s_1, s_2, s_3, s_4)$
  - Pick the class with the largest score
  - e.g., (s_{Car}: 2.1, s_{SUV}: 0.7, s_{Truck}: 3.4, S_{EV}: 1.0) ==> Truck
  - Mathematically, an ML person would say argmax(f(a, i, g, e)) = argmax (2.1, 0.7, 3.4, 1.0) = 3.4==> Truck
- Models internally compute these
- **Softmax: Probability-based classification**
  - Take the row scores (aka logins) and run them through the softmax function
  - Transforms arbitrary scores into numbers between 0 and 1 that sum to 1, representing probabilities
  - Softmax formula: $P(class\ i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$
  - Mathematically, an ML person would say softmax(f(a, i, g, e)) = softmax(2.1, 0.7, 3.4, 1.0) = (0.190, 0.047, 0.699, 0.063) ==> Truck, with probability 69.9%
- Aside: we were calculating f(features), in this case f(age, income, gender, education), and we normally think of f() as producing a single number. But in this case it produces *4* -- a score (or probability) for each class an object could belong to

Calculating Scores

Let the features be:

- a = age
- i = income (in $K)
- g = gender (Male = 1, Female = 0)
- e = education (HS = 1, Associate = 2, BSc = 3, MSc = 4)
- $s_1 = S_{\text{Car}} = 0.1a + 0.05i - 0.2g + 0.3e - 10$
- $s_2 = S_{\textbf{SUV}} = 0.05a + 0.1i + 0.1g + 0.2e - 15$
- $s_3 = S_{\textbf{Truck}} = 0.02a + 0.15i + 0.4g + 0.1e - 20$
- $s_4 = S_{\text{EV}} = 0.2a + 0.05i - 0.1g + 0.4e - 25$
- Predicted vehicle = arg max (S_{Car}, S_{SUV}, S_{Truck}, S_{EV})

Calculating Scores, Cont'd

Person A: age 25, income 30, male (1), high school (1)

Compute scores:

Car: 0.1(25)+0.05(30)-0.2(1)+0.3(1)-10 = 2.5 + 1.5 - 0.2 + 0.3 - 10 = -5.9

SUV: 0.05(25)+0.1(30)+0.1(1)+0.2(1)-15 = 1.25 + 3 + 0.1 + 0.2 - 15 = -10.45

Truck: 0.02(25)+0.15(30)+0.4(1)+0.1(1)-20 = 0.5 + 4.5 + 0.4 + 0.1 - 20 = -14.5

EV: 0.2(25)+0.05(30)-0.1(1)+0.4(1)-25 = 5 + 1.5 - 0.1 + 0.4 - 25 = -18.2

Predicted: Car has the highest score → **Car**.

The Confusion Matrix

- Its name notwithstanding, a confusion matrix isn't 😀
- It simply summarizes the results of running the model on trained data
- The rows show the labels in the trained data
- The columns show the predicted labels from the model
- Frequencies of each pair are shown in the matrix proper

Predicted

Actual

	Car	EV	SUV	Truck	Van
Car	46	1	3	0	0
EV	0	12	0	1	2
SUV	4	3	22	3	0
Truck	0	1	2	54	0
Van	2	3	1	1	62

Machine Learning Models: Where the Scores Come From

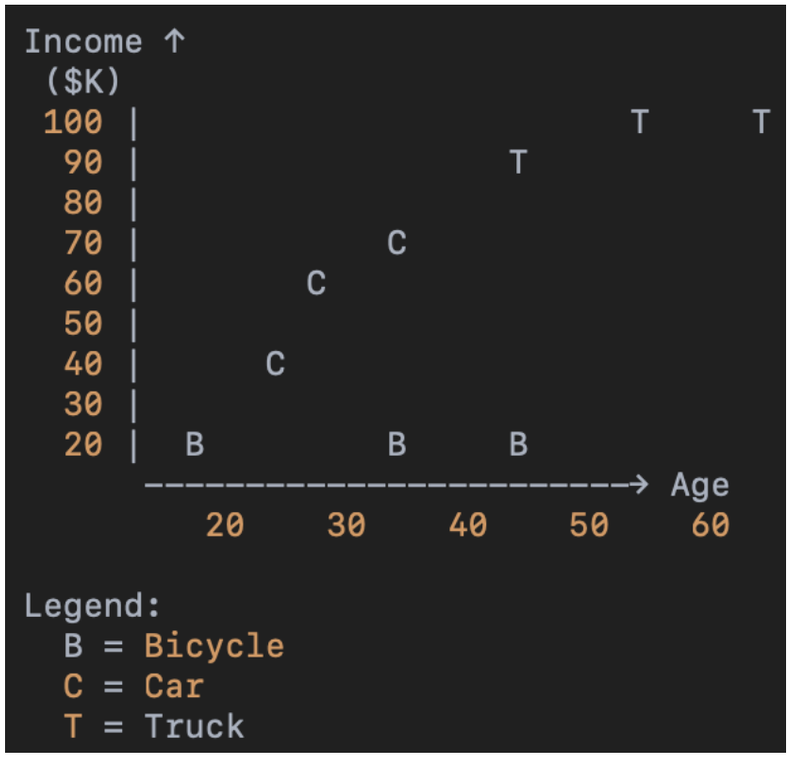
- There are numerous ML models - including:
  - Nearest neighbors - similarity to known examples
  - Decision trees - flowcharts of decisions
  - Random forests - the wisdom of crowds
  - Support Vector Machines -n optimal boundary/margin

Nearest Neighbor Models

- An *extremely* simple model, used for classification
- The training dataset really is the model
- Given an entity with a new set of features, simply find the one closest to it in the training data, and use its label

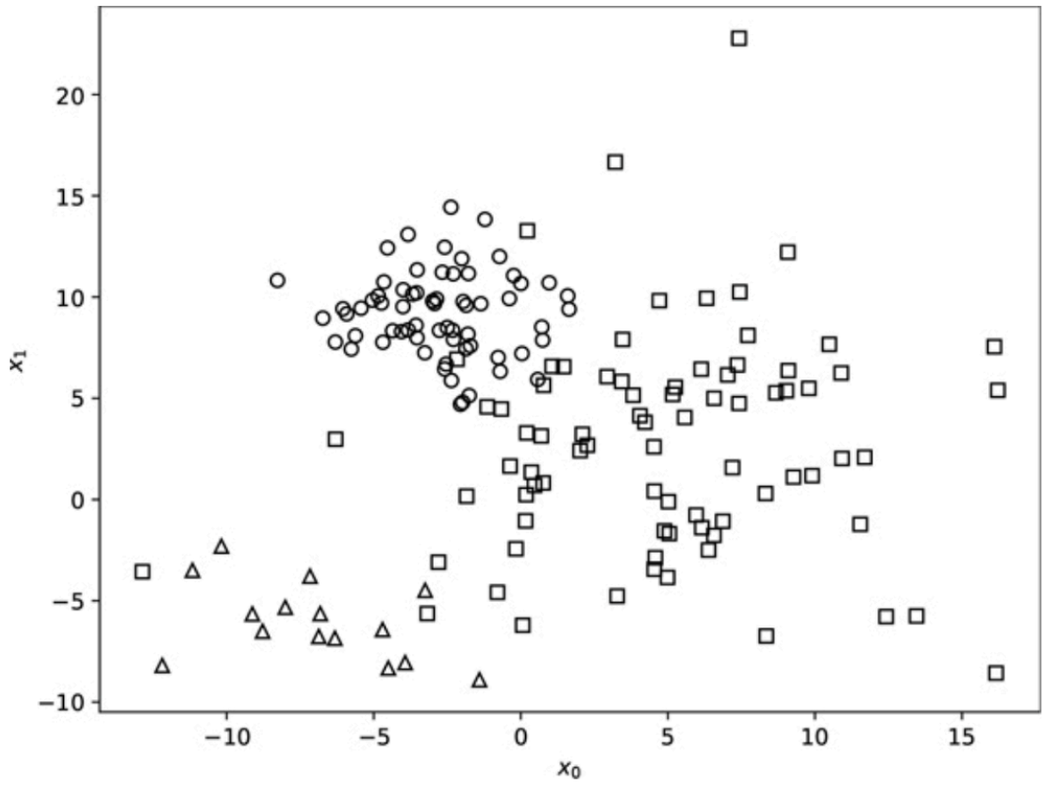
- e.g., A 35 year old makes $75K. What vehicle are they likely to buy?

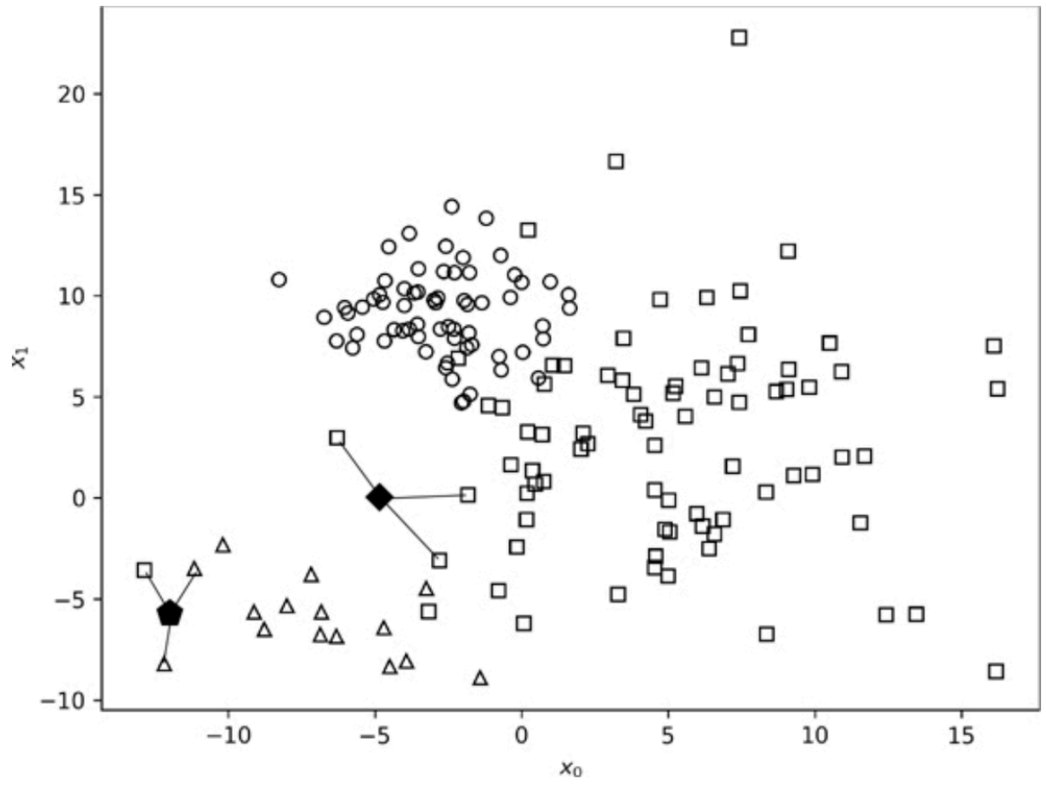
Person	Age	Income ($K)	Vehicle
A	22	20	Bicycle
B	45	90	Truck
C	30	60	Car
D	28	25	Bicycle
E	55	100	Truck
F	38	70	Car
G	27	45	Car
H	23	22	Bicycle
I	52	85	Truck
J	…	…	…



k-Nearest Neighbor Models

- There's a distinct risk just picking the nearest point.
- The K-NN picks the k nearest neighbors, and uses the most frequent as a result
- Q: How do you measure "nearest"?





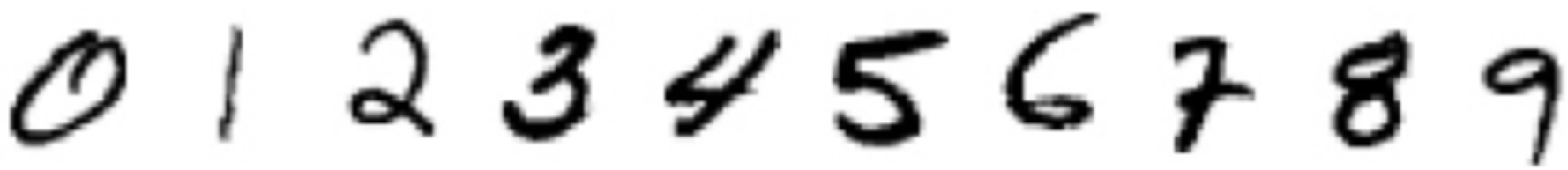
K-NN and the MNIST Digits Dataset

- The MNIST dataset consists of 28x28 pixel images of a large number of handwritten digits.

- We can collapse the image into a vector (1D array) of 784 elements - each element being a value between 0-255

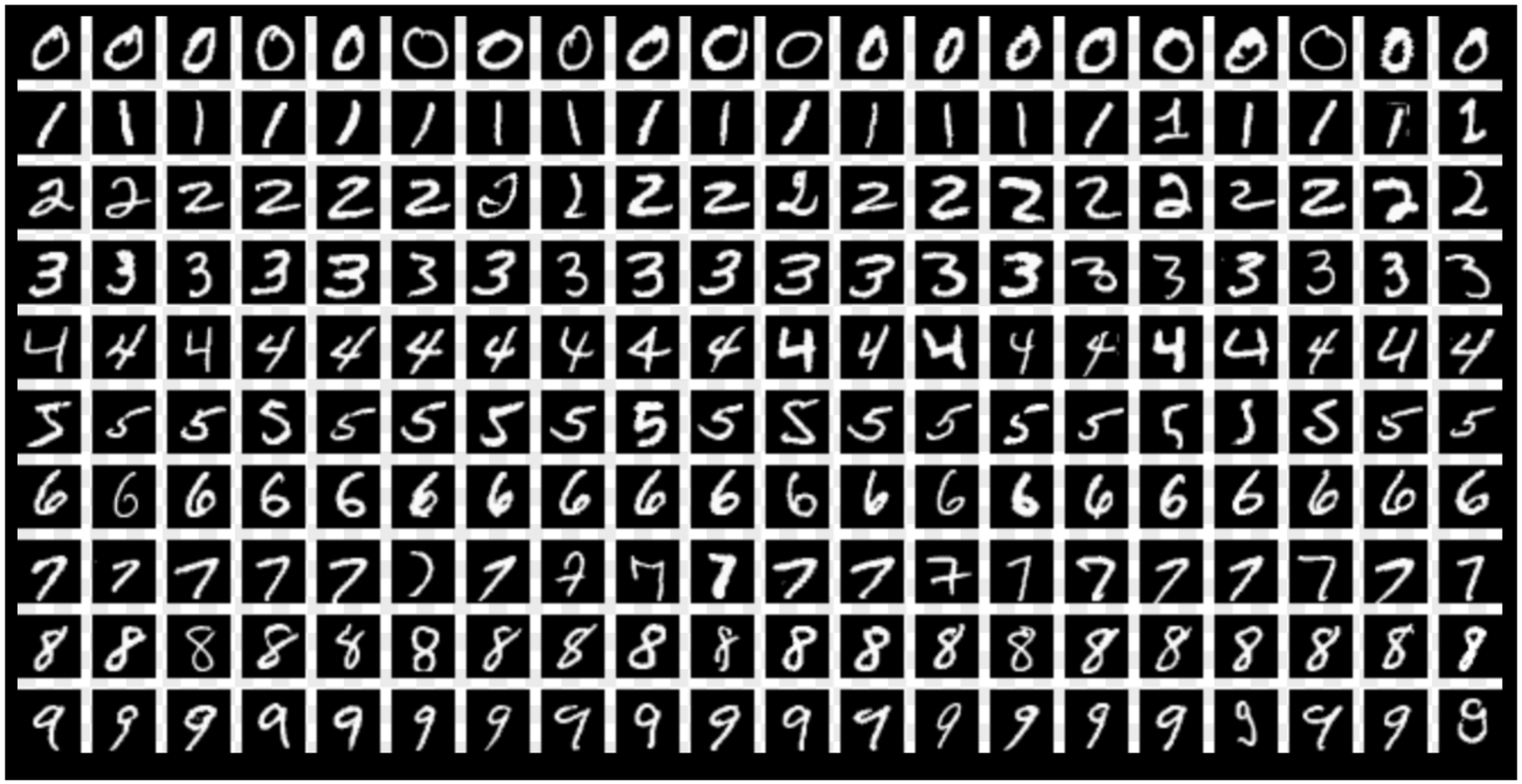
- So, instead of 2 features and proximity in a 2D space, we have distance in a 784D space!!

- What does that even *mean*?



MNIST Database Details

- 60,000 training images, 10,000 testing images



Euclidean Distance

- Let $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ be two points in nD space.

- When n = 2, this is just the familiar "distance between two points" problem

- In general, it is $\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

1	0	...	1
0	0	...	1
...	...	...	...
0	1		0
0	0		1

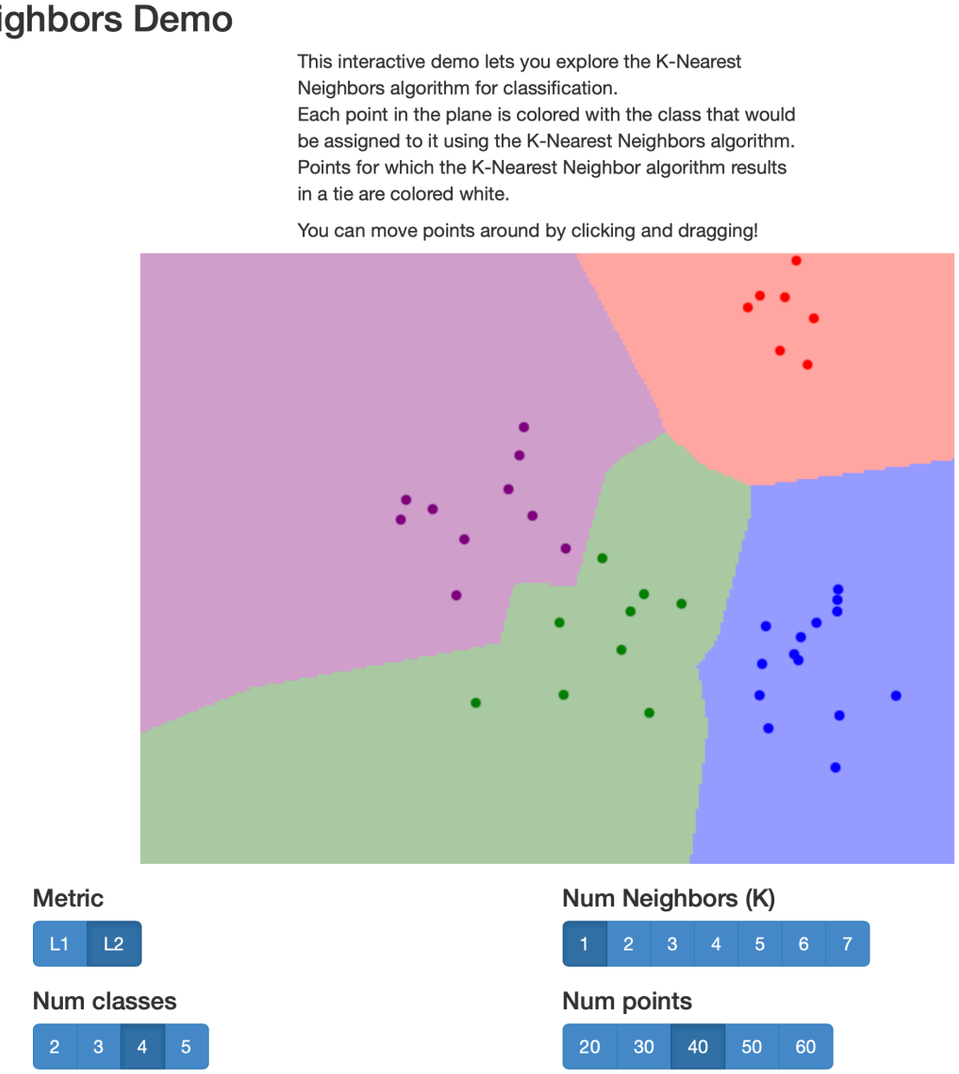
k-NN and the MINST Digits Dataset

- It turns out that with k=3 or 5, accuracy of around 95-97% is possible

- In contrast, a simple neural net can achieve an accuracy of 98%; a convoluted neural net, which we will study next, can achieve an accuracy of more than 99%

K-NN Demo

http://vision.stanford.edu/teaching/cs231n-demos/knn/



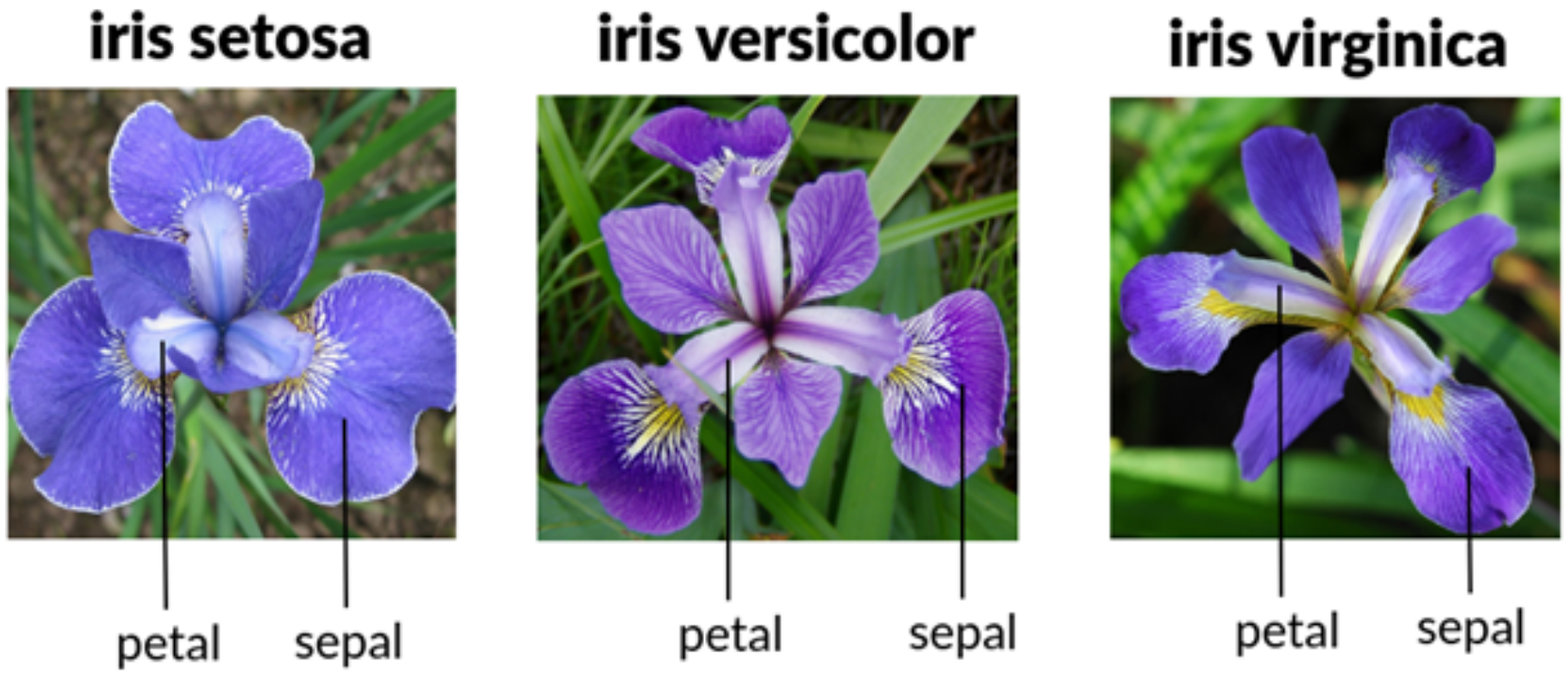
Decision Trees

- We've seen a *symbolic* decision tree, a set of rules written by people

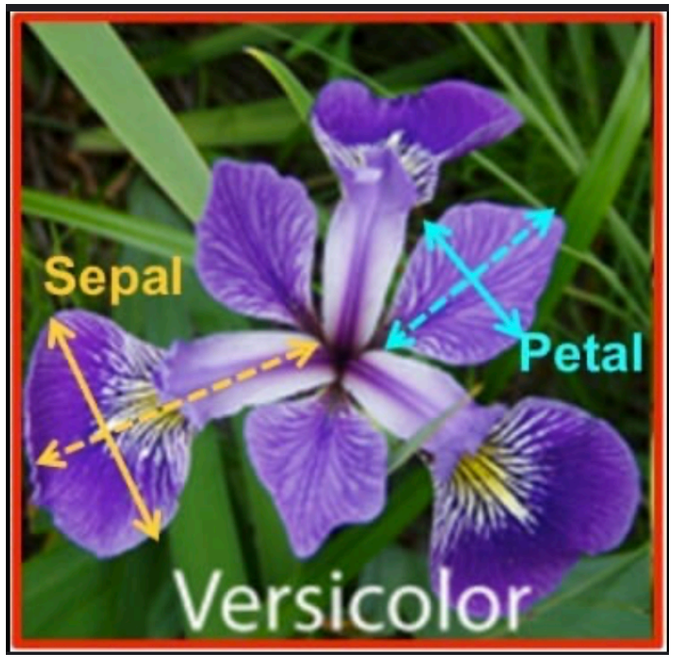
- With ML, though, the machine can *discover* the rules by processing data

- A classic example in ML involves measurements of 150 iris flowers, with 4 features and 3 labels

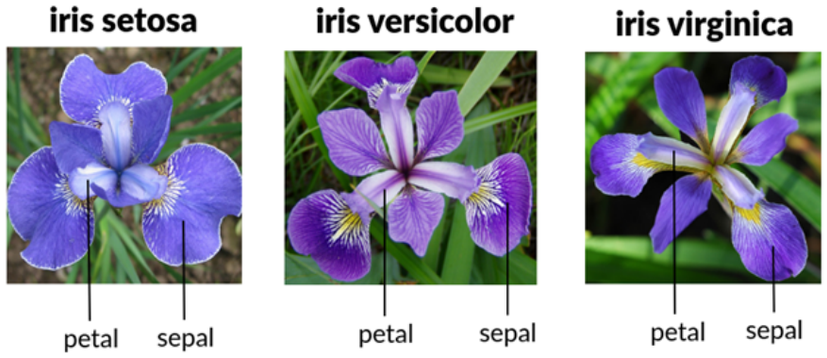
Decision Trees



Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.2	3.6	6.1	2.5	virginica

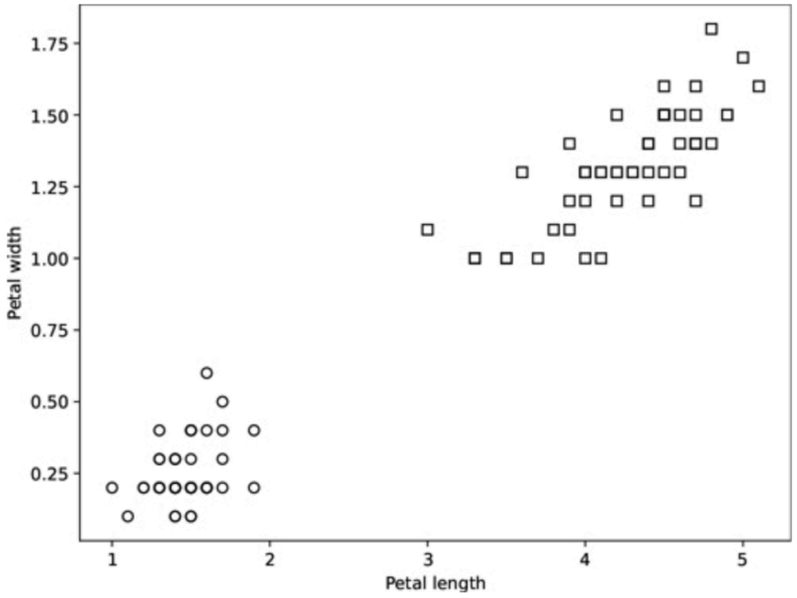


Decision Trees

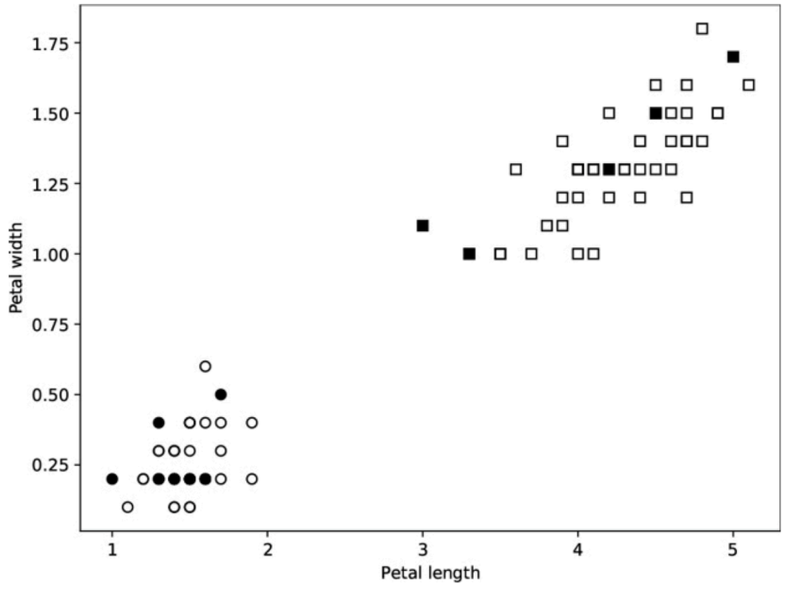


- If we just used setosa and versicolor, what would our decision tree look like?

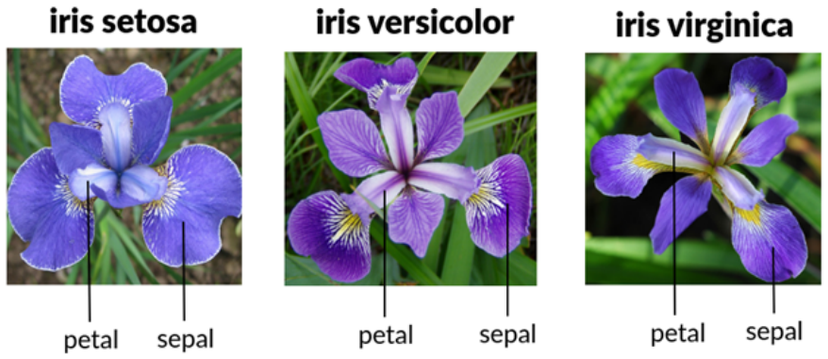
Training Dataset
setosa ○
versicolor □



Test Dataset
setosa ●
versicolor ■

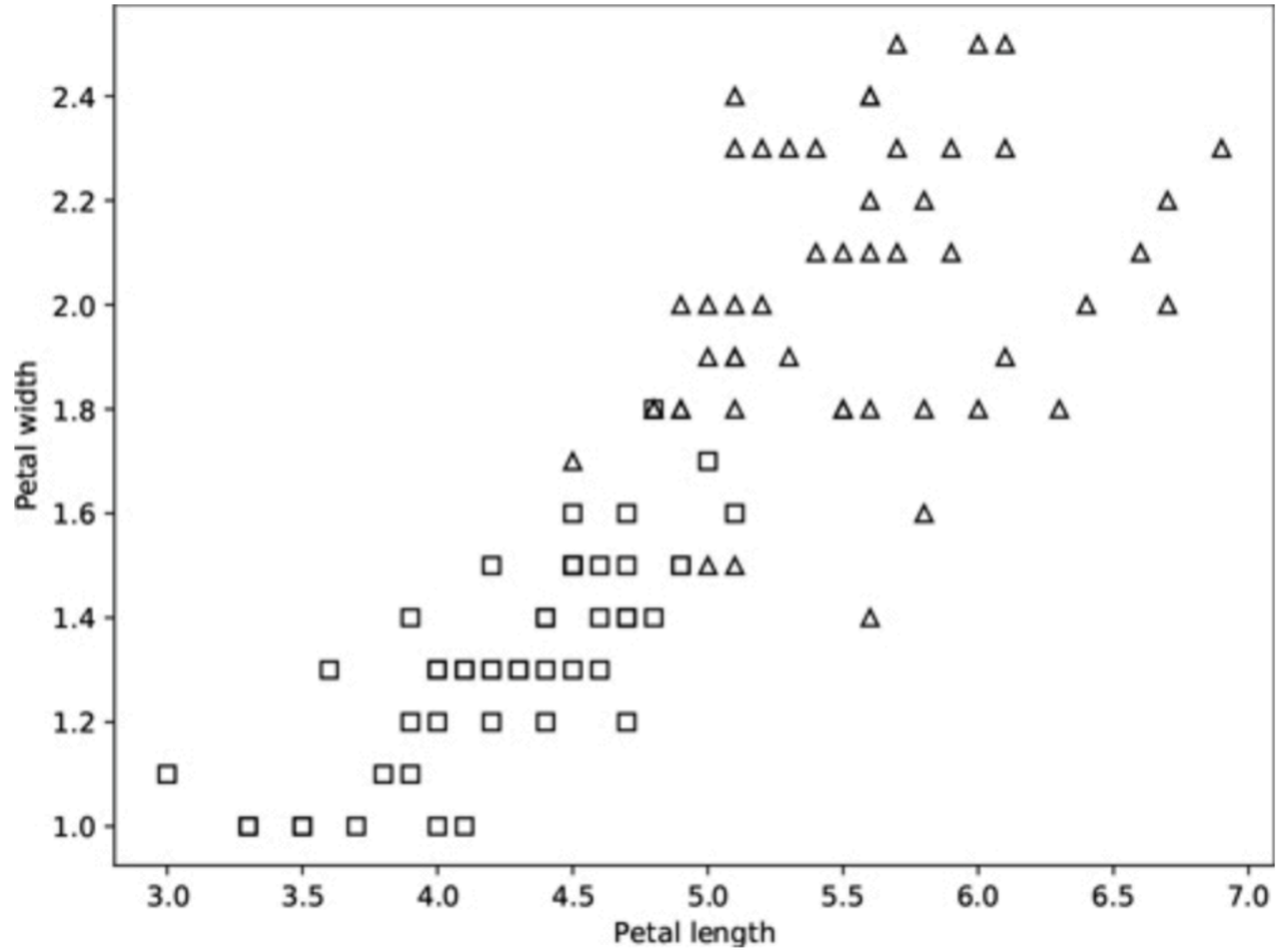


Decision Tree



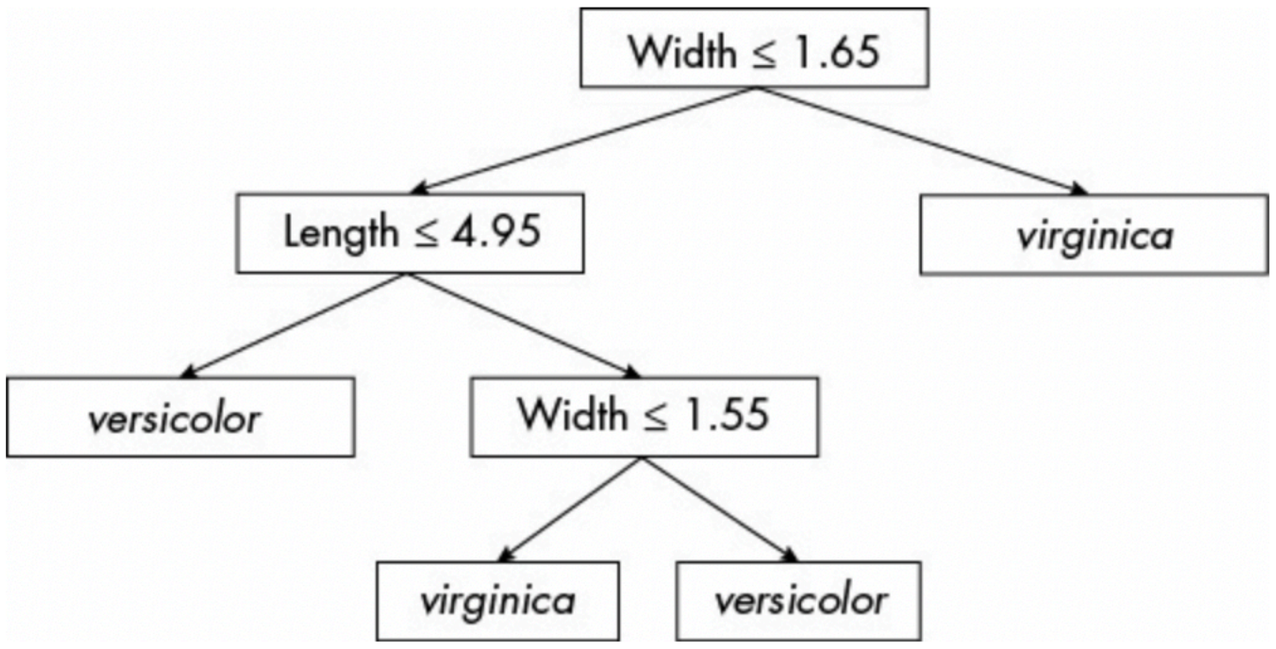
- But what if we used virginica instead of setosa?

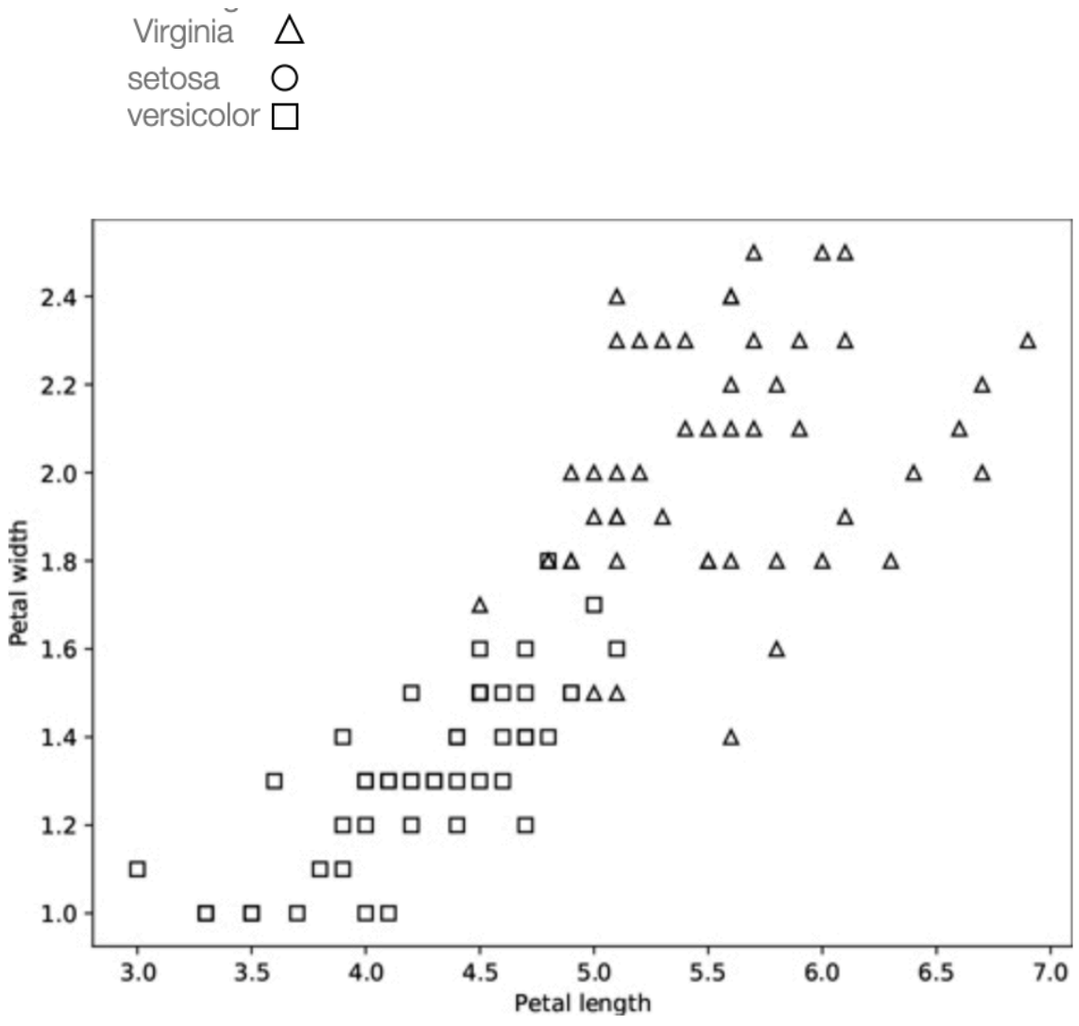
Training Dataset
Virginia △
versicolor □



Decision Tree, Decided!

- The tree here has an accuracy of 90%





Random Forests

- A random forest consists of a set of decision trees
- Each decision tree is created independently using a randomly chosen subset of:
  - all the samples in the training dataset and
  - all the features in the training set
- Test data is then supplied to all the decision trees, and, as with k-NN, majority rules
- This turns out to do surprisingly well