Machine Learning I

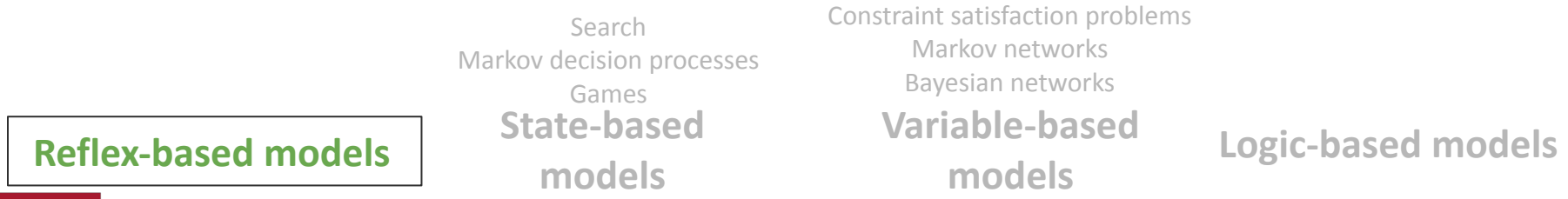
Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada
Sept. 17, 2024

Course Announcements

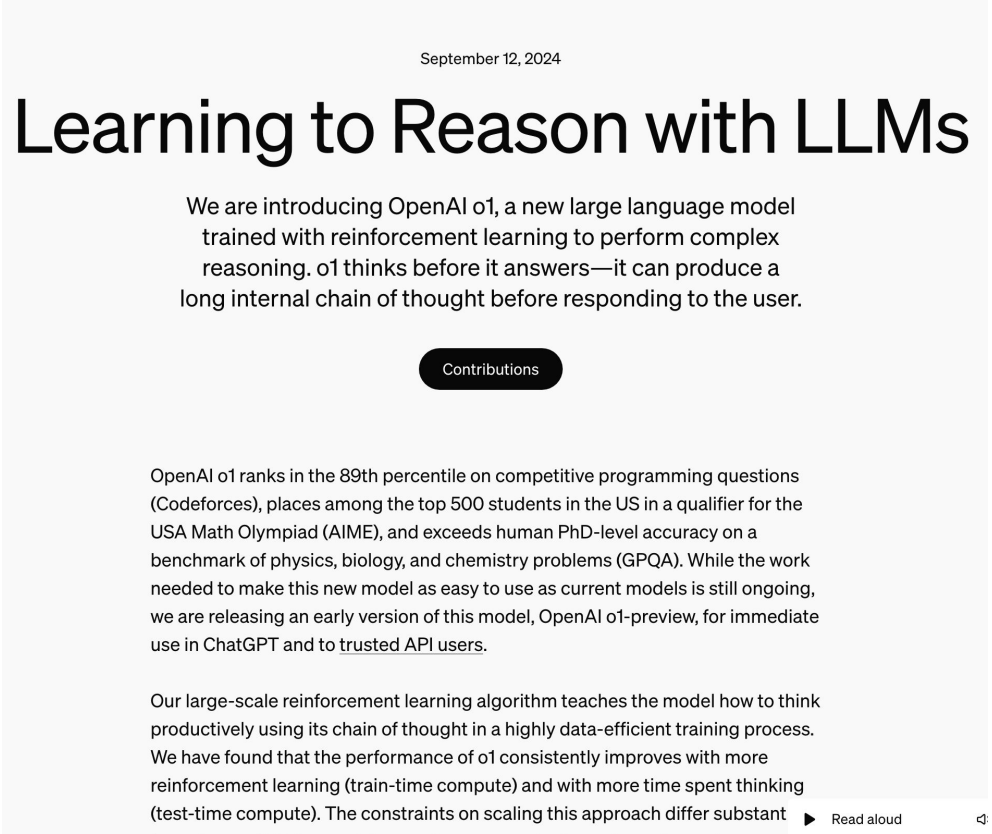
Midterm is set for Tues, October 29th in class.

Course Overview

Week 1 : Getting to know you
Week 2 : Introduction to Artificial Intelligence
Week 3 : Machine Learning I: Basic Unsupervised and Supervised Models (Classification)
Week 4 : Machine Learning II: Supervised (Regression) and Neural networks
Week 5 : Search
Week 6 : Markov Decision Processes
Week 7 : Reinforcement Learning
Week 8 : Midterm
Week 9 : Games
Week 10 : Constraint Satisfaction Problems
Week 11 : Hidden Markov Models and Bayesian Networks
Week 12 : Bayesian Networks and Logic
Week 13 : Ethics and Explainability



In the news…



September 12, 2024

Learning to Reason with LLMs

We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. o1 thinks before it answers—it can produce a long internal chain of thought before responding to the user.

Contributions

OpenAI o1 ranks in the 89th percentile on competitive programming questions (Codeforces), places among the top 500 students in the US in a qualifier for the USA Math Olympiad (AIME), and exceeds human PhD-level accuracy on a benchmark of physics, biology, and chemistry problems (GPQA). While the work needed to make this new model as easy to use as current models is still ongoing, we are releasing an early version of this model, OpenAI o1-preview, for immediate use in ChatGPT and to trusted API users.

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substant

Read aloud

Today's Plan

19.1-2 - Forms of Learning

Supervised Learning

Classification

19.3 - Learning Decision Trees

19.7 - K-Nearest Neighbours

19.4 - Evaluation methods

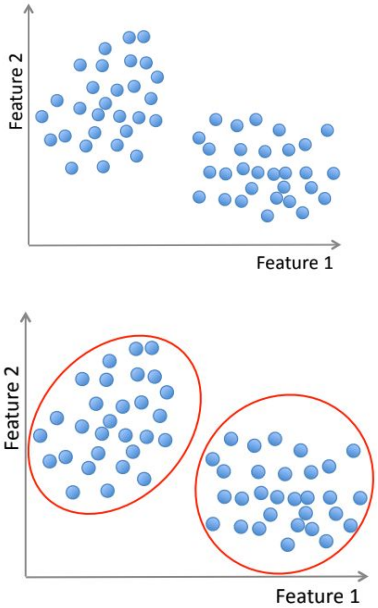
Regression

19.6.4 - Linear regression

Definitions and Intuitions

Types of Learning

- **Unsupervised Learning**: a type of machine learning where we attempt to make inferences about *unlabelled* data



These inferences could be:

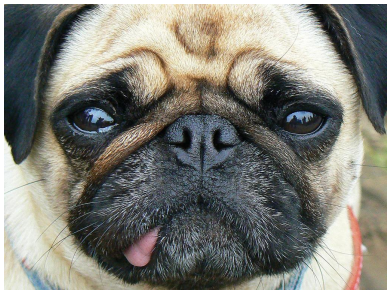
- Different ways to group that data
  - Clustering
- Anomalies that exist in the data
- Finding new patterns in the data

Types of Learning

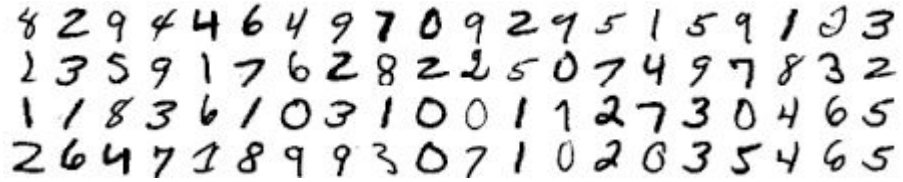
- **Supervised Learning**: a type of machine learning where we are given data as an input and attempt to output the correct **label**

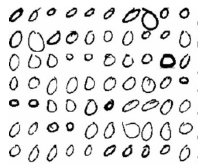
- For example, we can take an image as **input** and assign it with the **label** *cheetah*, *dog*, *wolf*, etc.

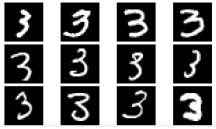
- A specific instance of our data (one particular email or image) is called an **example**



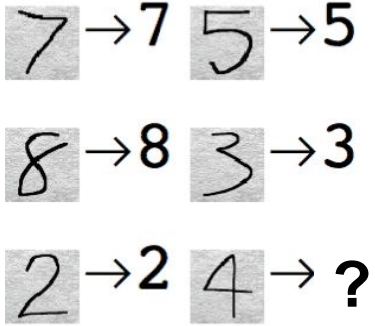
Unsupervised and Supervised Learning



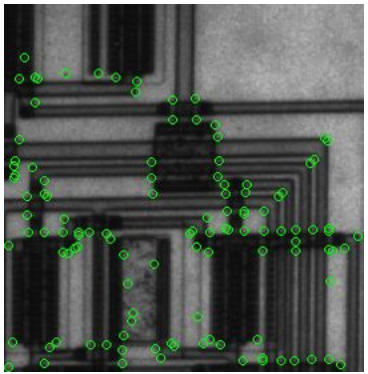




- **Unsupervised**: data set contains unlabeled examples
  - **Example**: unannotated images of handwritten digits
- **Supervised**: data set contains both examples and labels
  - **Example**: images of handwritten digits with human annotations

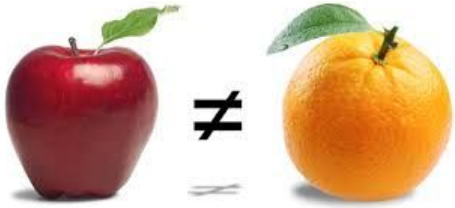


Data for Supervised Learning



Features (Input)

- A feature is a property of an object we can use to describe
- Quantify attributes of an object in a way that a computer can understand
- Want to choose the most relevant ones (feature selection)



Labels (Output)

- What you want to learn

Features and **labels** can be **binary**, **real-valued**, or take on a small number of **discrete values**

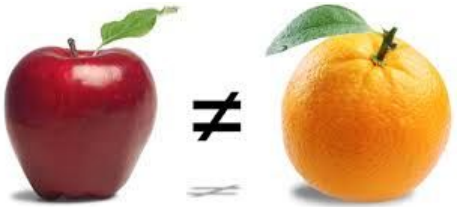
Data for Supervised Learning

- An **example**, **x**, is a particular instance of data (could be a picture, an email, a sentence, a song) which is represented by a set of features, e.g. weight, color, has seeds

  $\mathbf{x} = (x_1, x_2, x_3, \ldots)$ like $\mathbf{x} = (1.5, \text{green}, \text{yes})$

- A **labelled example** has both a feature and a label: (**x**, y)

  ((1.5, green, yes), apple)



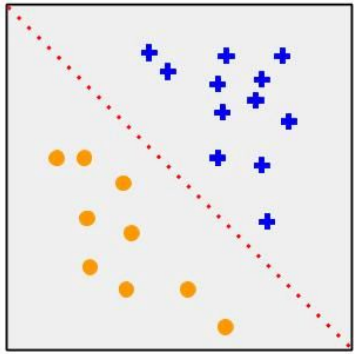
- A **dataset** is a collection of labelled or unlabelled examples

  $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots\}$
  like D = {((1.5, green, yes), apple), ((1.5, orange, yes), orange)}

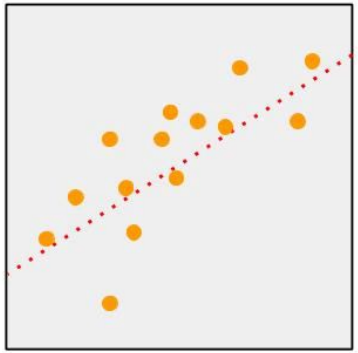
Supervised Learning

Two common types of tasks:

- **Classification**: When the label is a specific *class*
  - Determining if mail is spam or not spam
  - Determining if a picture has a cat or not
- **Regression:** When the label is a *real number*
  - Predicting tomorrow's temperature
  - Predicting the cost of coffee in 2020
- Classification labels are **discrete**
  - Small number of possible values
- Regression labels are **continuous**



Classification



Regression

Supervised Learning

Classification

Classification algorithms

- **Decision Trees**
- **Nearest Neighbors**
- Neural Networks
- ….

⇒ The features are typically computed or "hand-crafted" using a feature extractor

- Deep neural networks

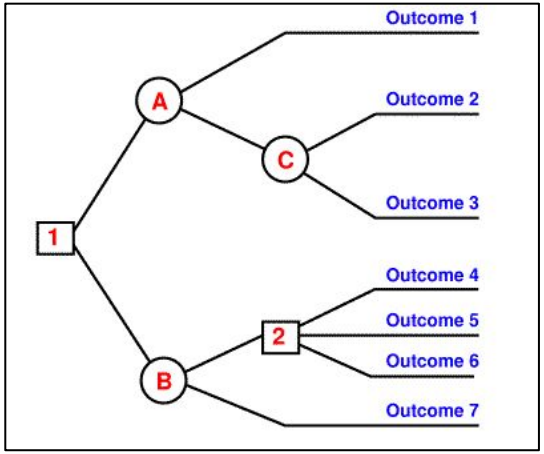
⇒ features are extracted as part of the network

Supervised Learning: Classification

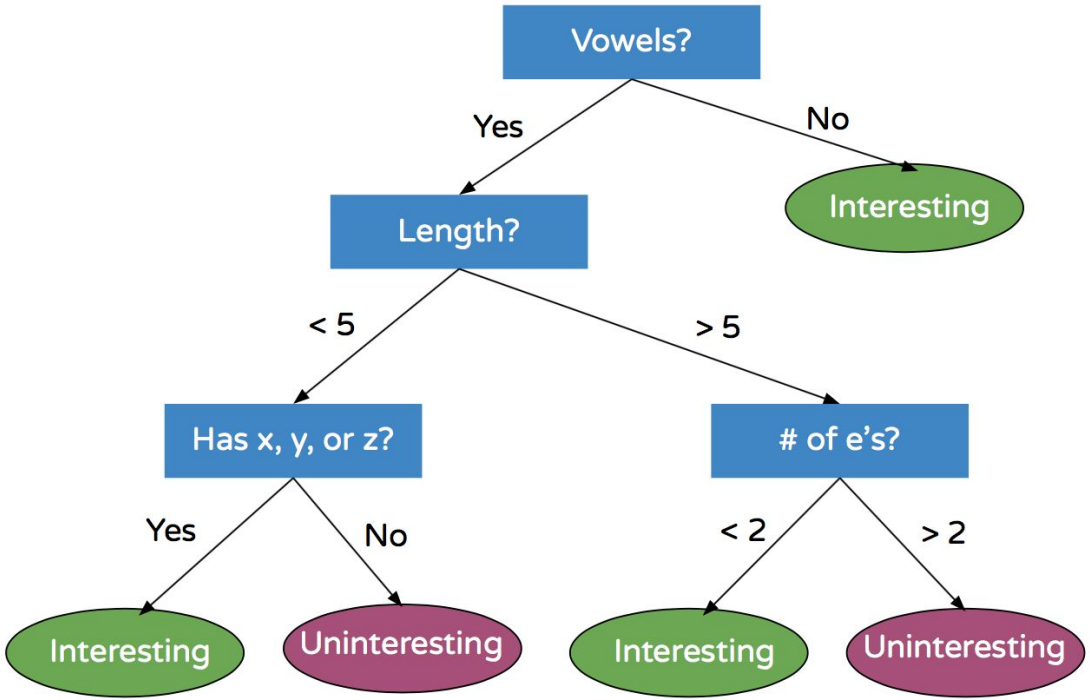
Decision Trees

Supervised: Decision Trees

- An algorithm for classification that makes decisions by splitting up the training set based on specific features
- When trying to decide how to classify an example, we will ask a series of questions (often yes/no questions)
- Which questions we ask will depend on what the training set looks like



Example

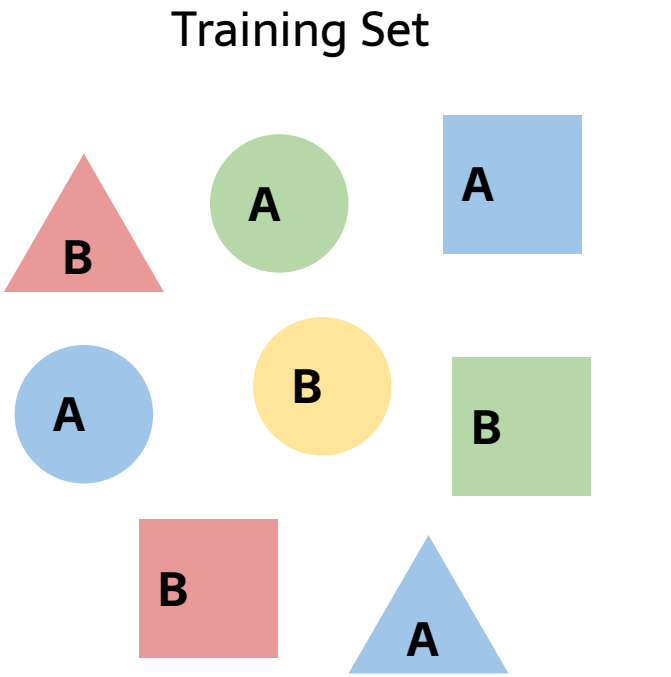


How would we classify the following words according to this decision tree?

- Rhythm: _____
- Abibliophobia: _____
- Emergence: _____
- Zen: _____

Learning a decision tree

- We use our training data to build a decision tree
- We aim to build a decision tree that:
  - Classifies the examples in our training set correctly
  - Generalizes for unseen examples



Learn a tree that classifies the colored shapes best as A or B

Learning a decision tree

Step 1:

- Select the features that give the "best split"

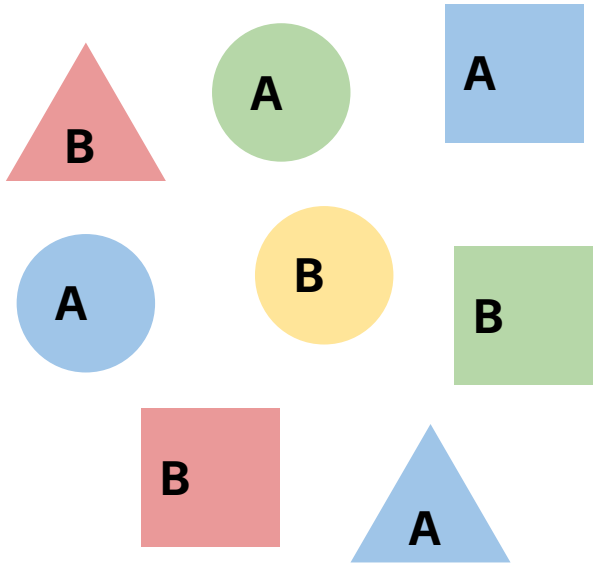
Feature(s):

- Green
- Square

Only one item :(

Label as group B

Training Set



Learning a decision tree

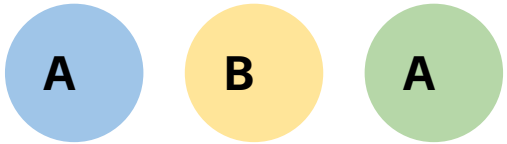
Step 1:

- Select the features that give the "best split"

Feature(s):

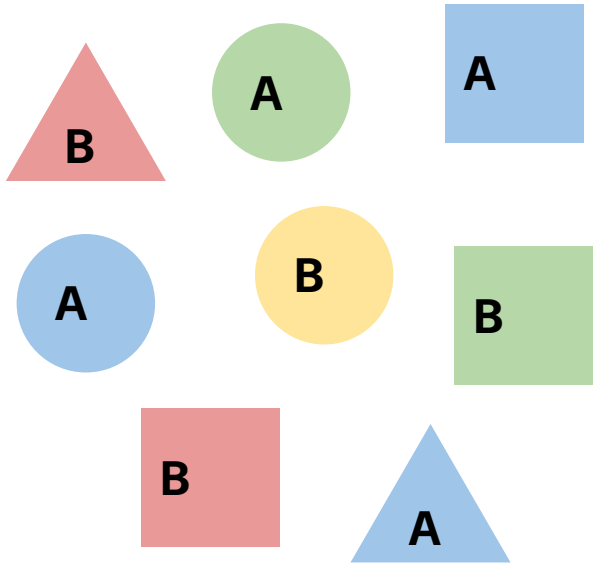
- Circle

One item mislabeled :(



Label as group A

Training Set



Learning a decision tree

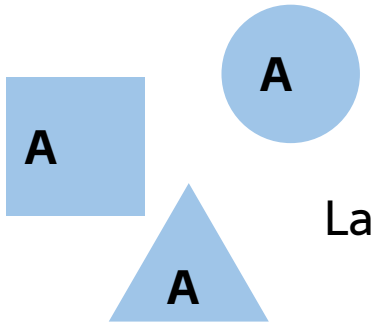
Step 1:

- Select the features that give the "best split"

Feature(s):

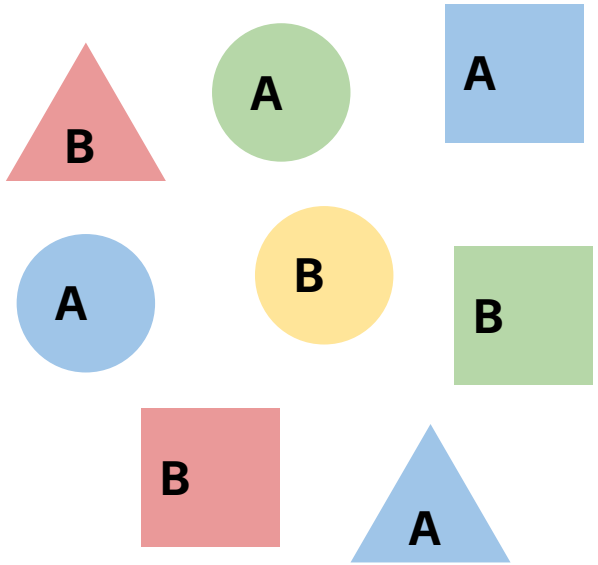
- Blue

Three items correctly labeled! :)



Label as group A

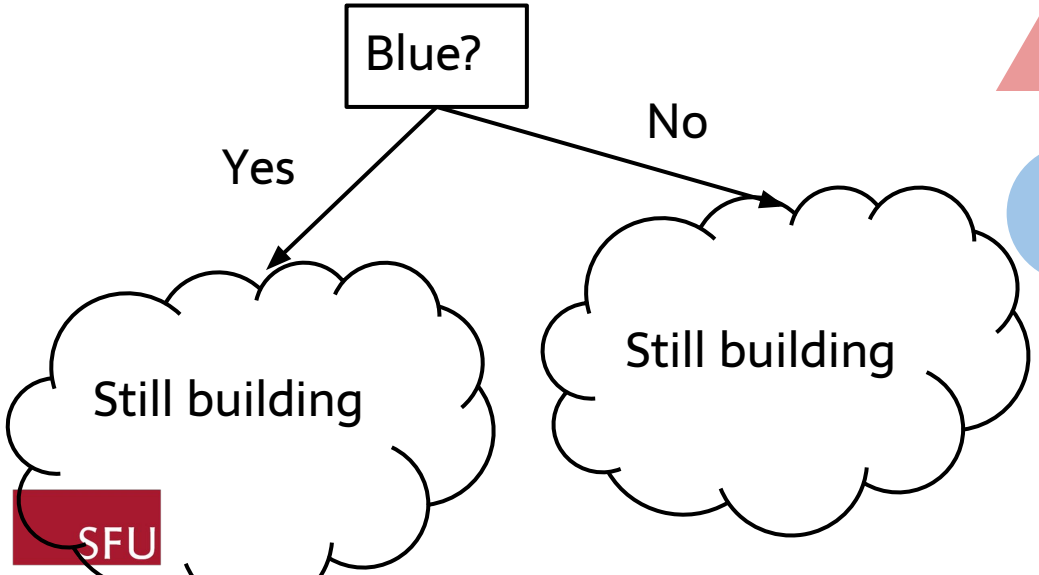
Training Set



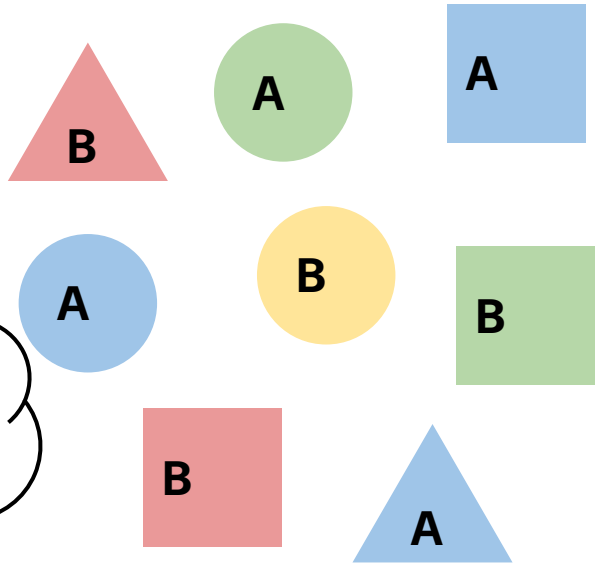
Learning a decision tree

Step 2:

- Create a new node in the tree based on this split



Training Set



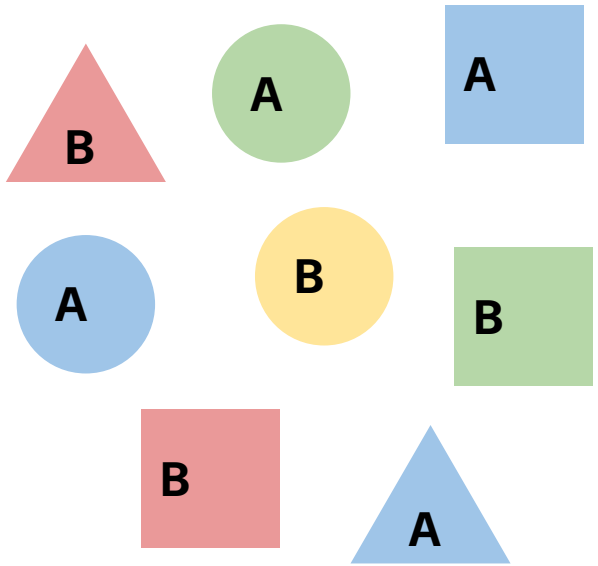
Learning a decision tree

All combinations of features are checked.

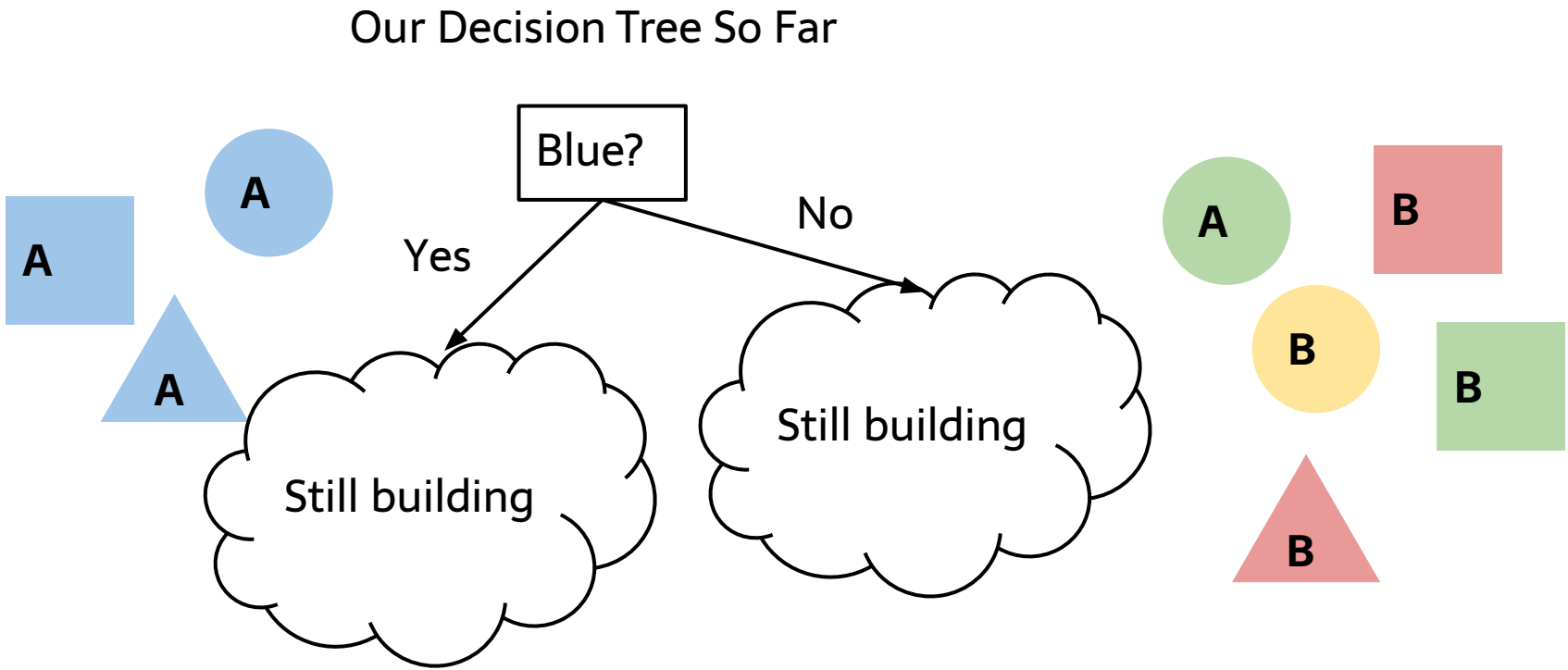
Step 3:

- Separate your data based on this split

Training Set



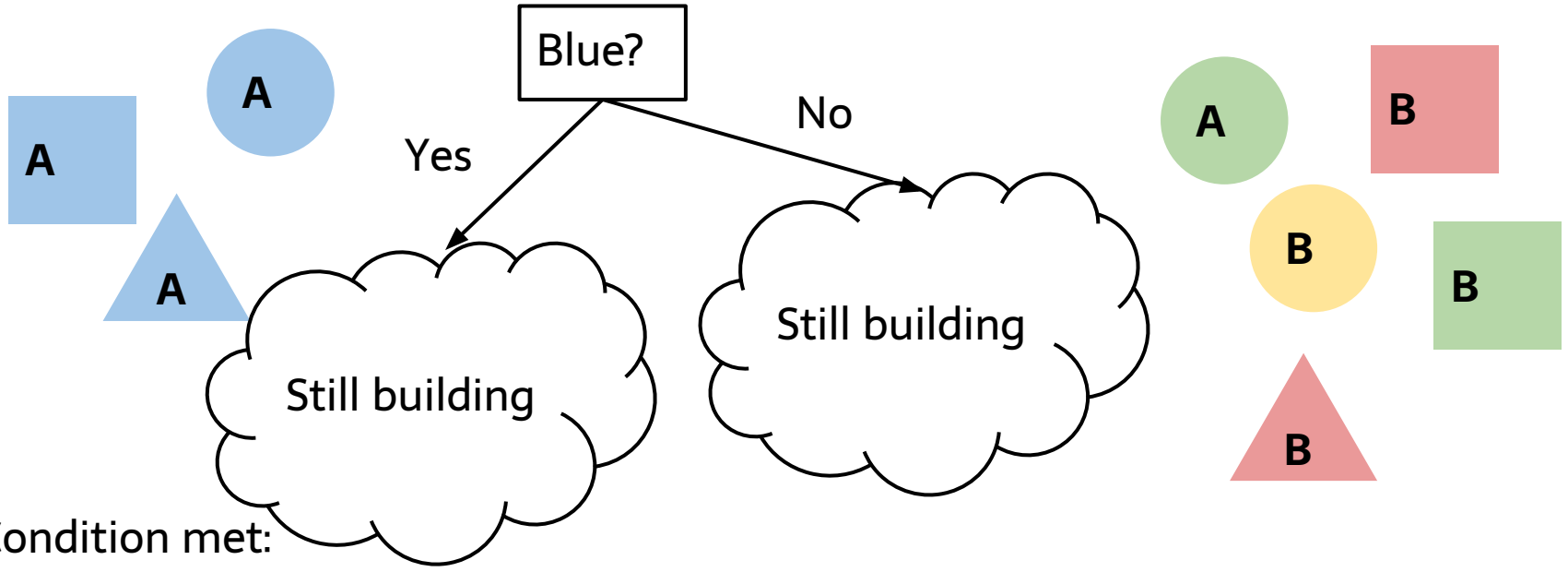
Learning a decision tree



Learning a decision tree

1. Select the feature that gives the "**best split**"
2. Create a new node in the tree based on this split
3. Separate your training data based on this split
4. For each group, repeat steps 1 to 3 until one of the following is met:
   a. All examples in the group have the same class
   b. The group has less than some number of examples
   c. You've already split the tree a certain number of times
5. If met, create a child node that represents the class

Our Decision Tree So Far

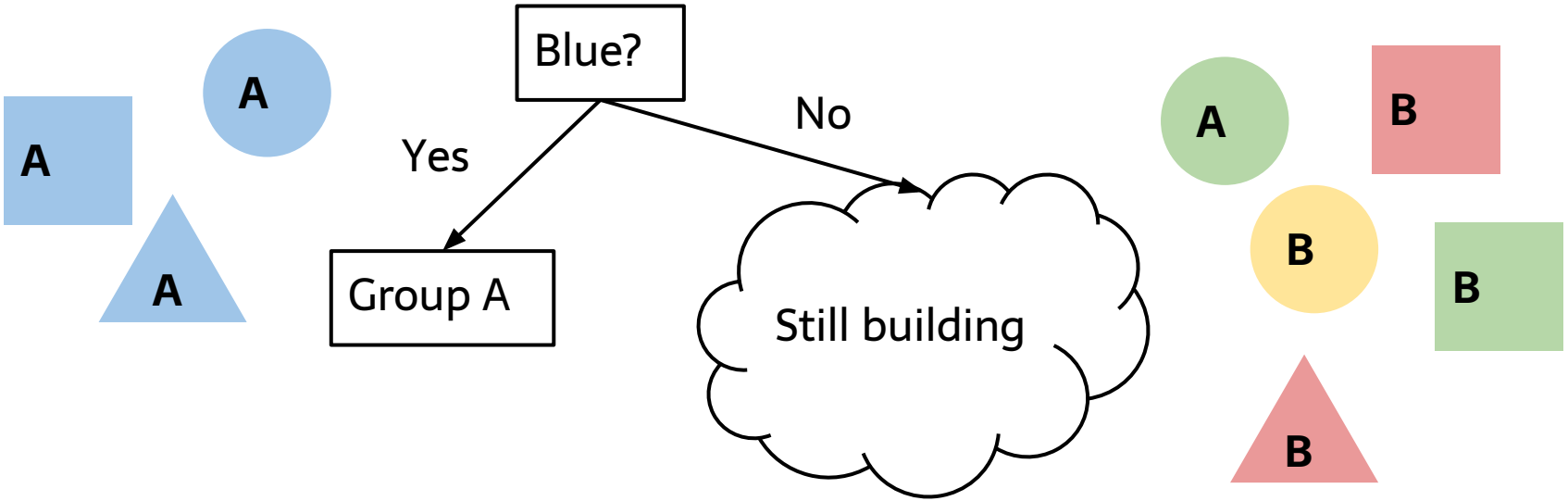


Condition met:
All same class
(group A)

Conditions:

- All same class
- < 3 examples
- Tree is deeper than 4 questions

Our Decision Tree So Far



No conditions met; repeat step 2

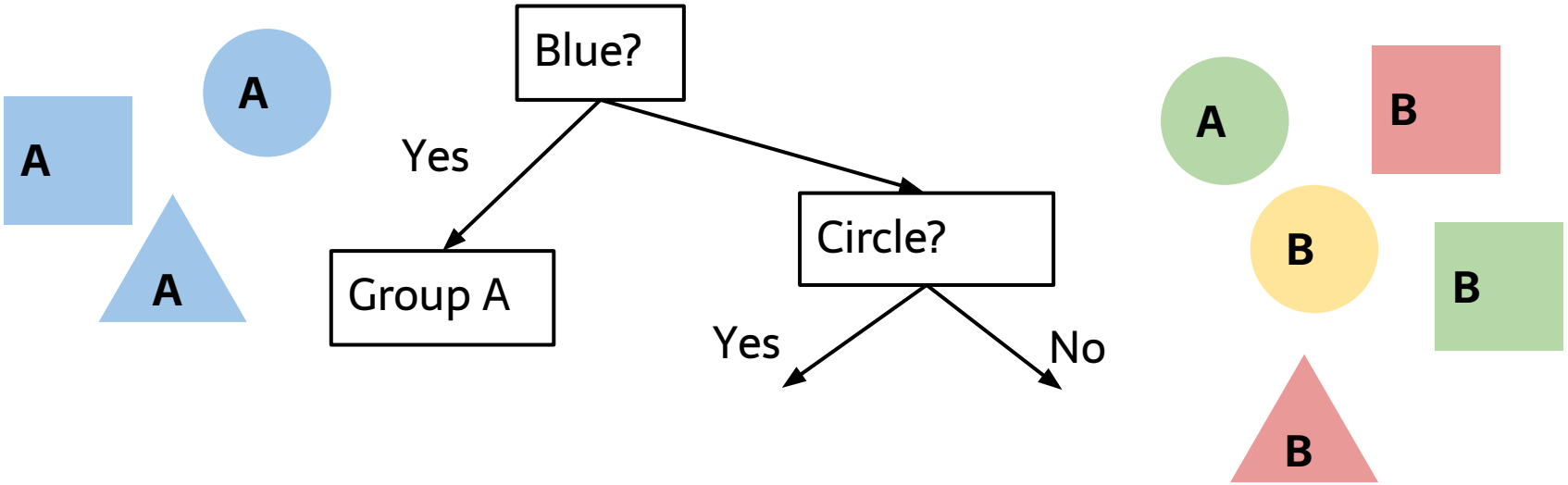
Conditions:

- All same class
- < 3 examples
- Tree is deeper than 4 questions

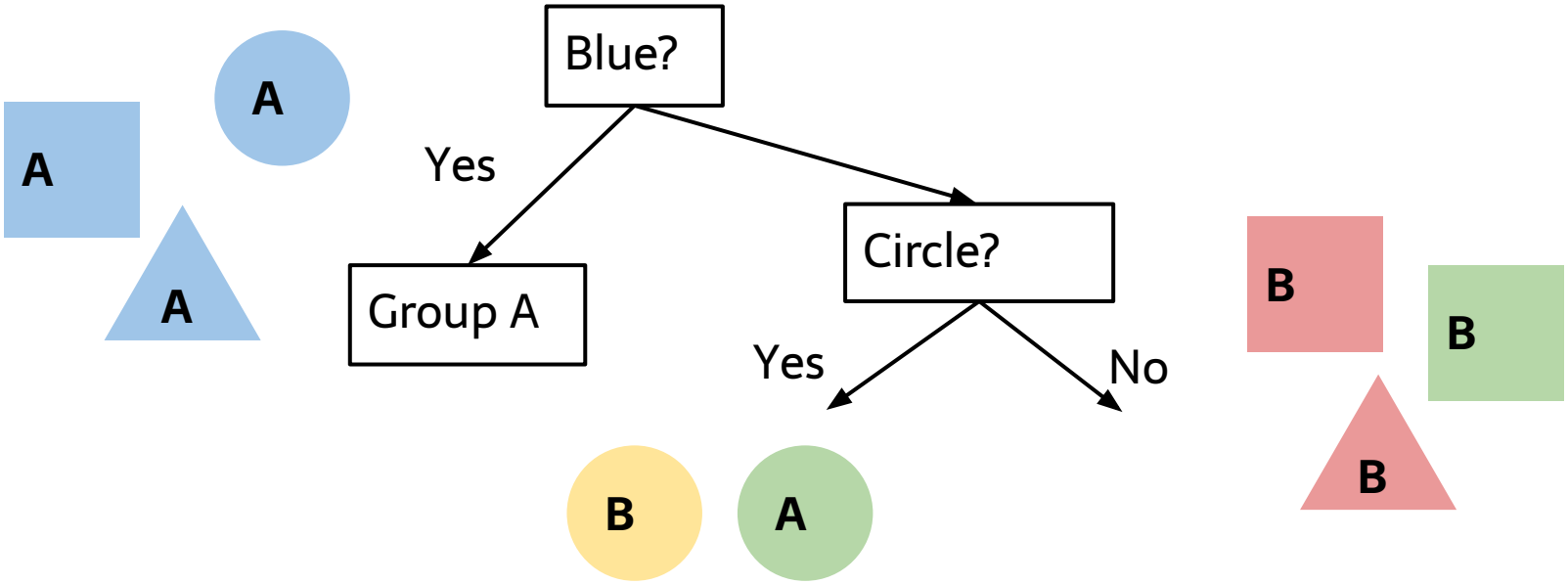
Learning a decision tree

1. Select the feature that gives the "**best split**"
2. Create a new node in the tree based on this split
3. Separate your training data based on this split
4. For each group, repeat steps 1 to 3 until one of the following is met:
   a. All examples in the group have the same class
   b. The group has less than some number of examples
   c. You've already split the tree a certain number of times
5. If met, create a child node that represents the class

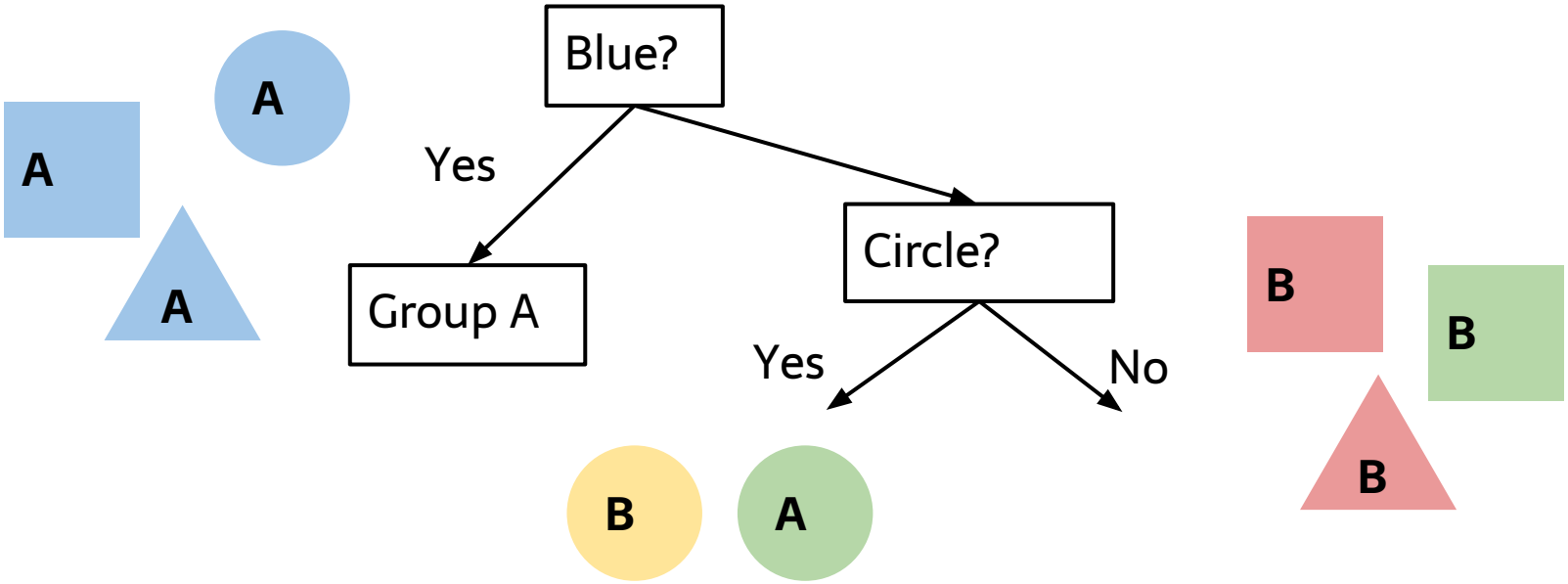
Our Decision Tree So Far



Our Decision Tree So Far



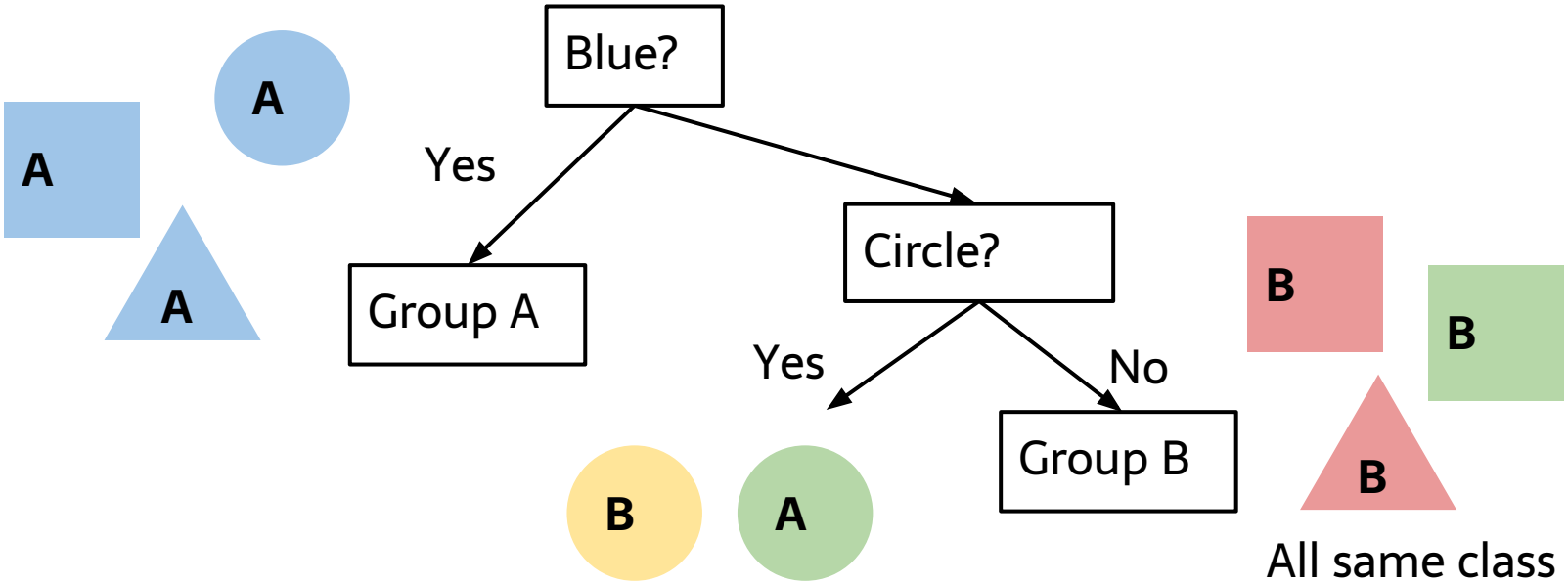
Our Decision Tree So Far



Conditions:

- All same class
- < 3 examples
- Tree is deeper than 4 questions

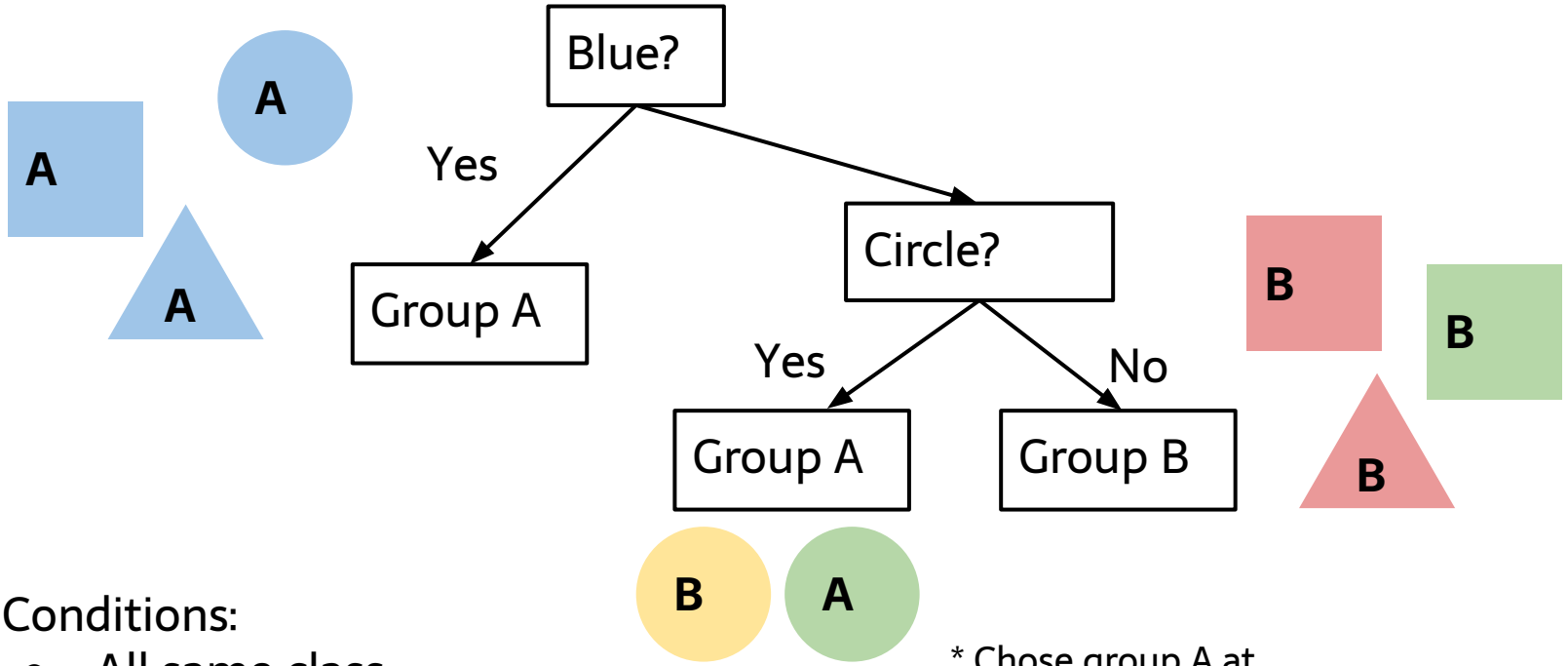
Our Decision Tree So Far



Conditions:

- All same class
- < 3 examples
- Tree is deeper than 4 questions

Our Decision Tree So Far

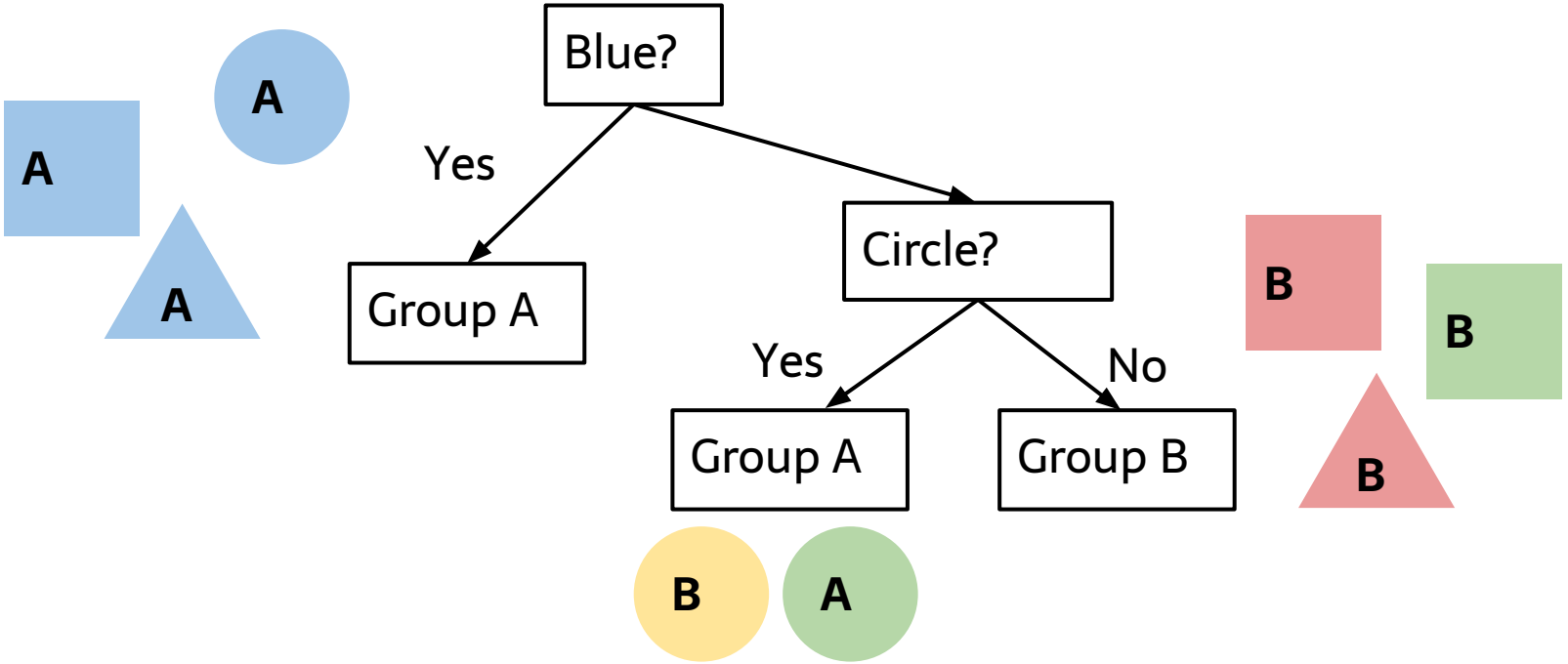


Conditions:

- All same class
- < 3 examples
- Tree is deeper than 4 questions

* Chose group A at random, since we get 1 wrong either way

Our Final Decision Tree



When considering all the possible subsets of features to use and how to classify them, there may be no optimal solution for determining when to prune or when to consider multiple features. You must do empirical research to determine this.

Learning a decision tree

So how do we find the feature that gives us the best split?

- One approach: find the split with the **least classification error** after the split
- That is, if you were forced to make a classification for each of your groups after the split, how would you split your data so you get the most examples classified correctly?
- We would do this for all of our features and compare to see which one gives the best split

Impurity and Entropy

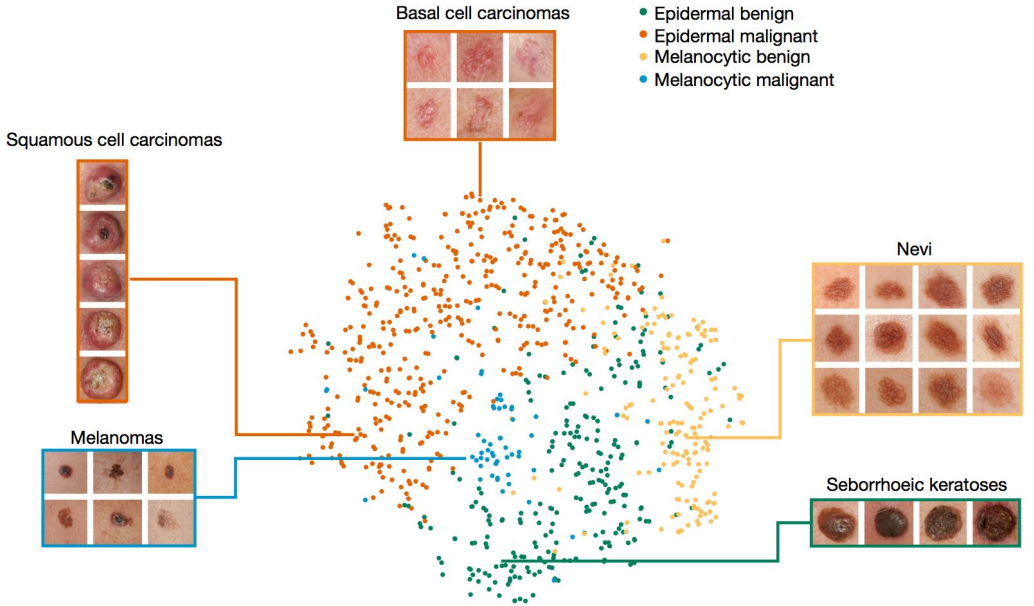
- Another way to think of this is from the standpoint of **impurity**
- You can think of a split as being *pure* if after the split, everything in each group belongs to one class
- We can use a measure called **entropy** to quantify the amount of impurity after a given split
- The idea is to build our tree so that after each split, we reduce impurity and entropy as much as possible

Supervised Learning: Classification

(K-)Nearest Neighbours

Nearest Neighbours Classification

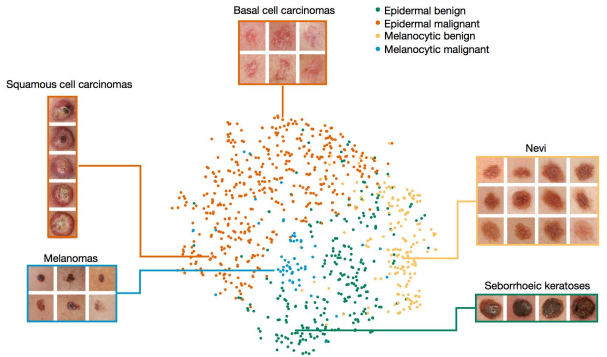
Let's take the task of classifying a skin image as cancerous or not



Feature extraction: Images to points

- For example, say we define each picture by two features: *darkness* and *size*
- For any given picture, we can choose to express it mathematically as

  (darkness, size)
- So an example could be (0.5, 0.6), meaning darkness is 0.5 and size is 0.6

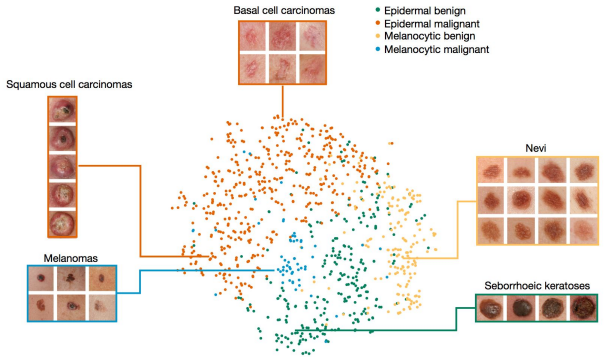


These values can be normalized.

Features may be extracted by experts or by the system which determines it.
The issue with having parameters determined by the system is that the method used for determing these values may not be clear.

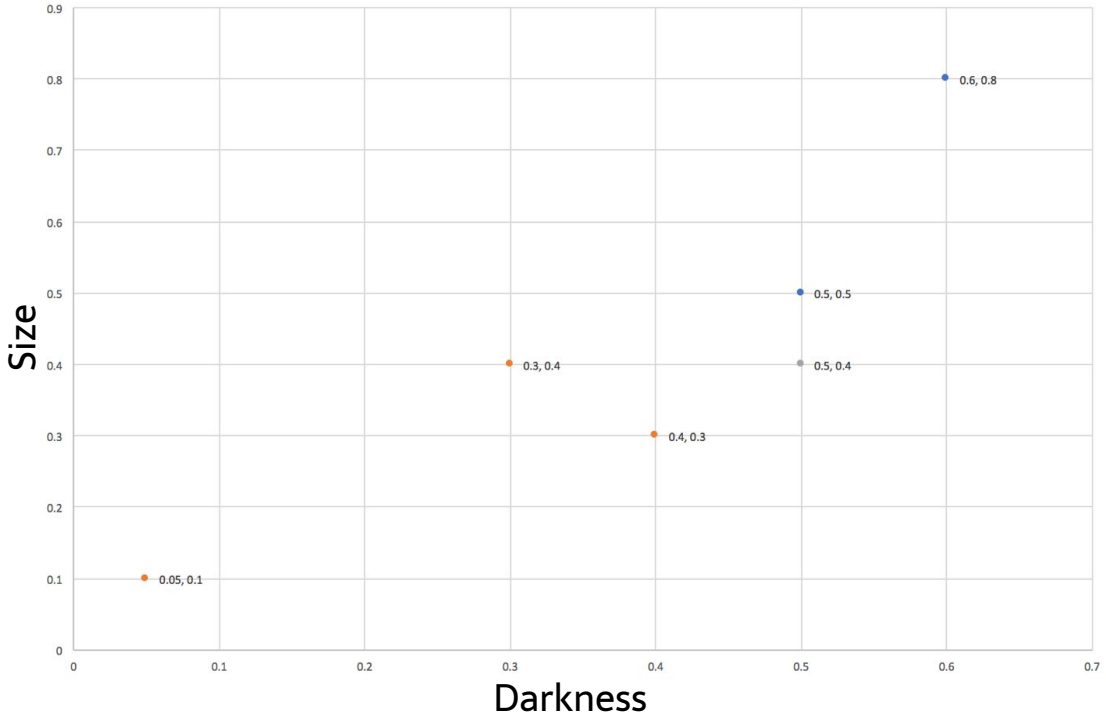
Feature selection

- Why pick the features *darkness* and *size*? And who picks these features?
- It's somewhat arbitrary - there are different reasons to pick one feature over another
- Often features are chosen by specialists (like the doctors themselves) because they know those features are meaningful
- Recent AI systems attempt to automatically pick the most meaningful features from the data



Nearest Neighbours Example

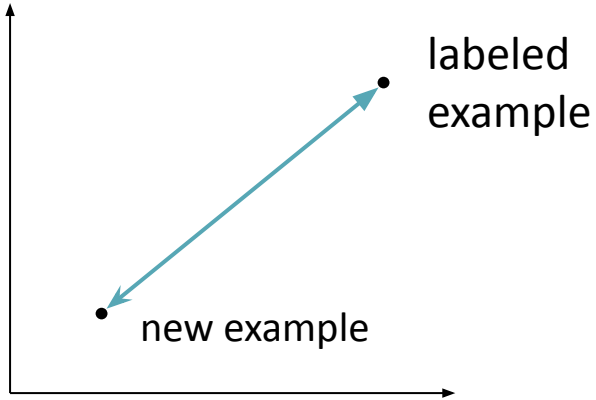
- New example
- Non-cancerous
- Cancerous



Nearest Neighbours Classification

- Relies on the assumption that similar examples should be classified the same
- Examples represented as points in an n-dimensional space
- Similarity is dependent on the distance to other examples on the graph

Simplest version: Give your new example the same classification of the “closest” training example

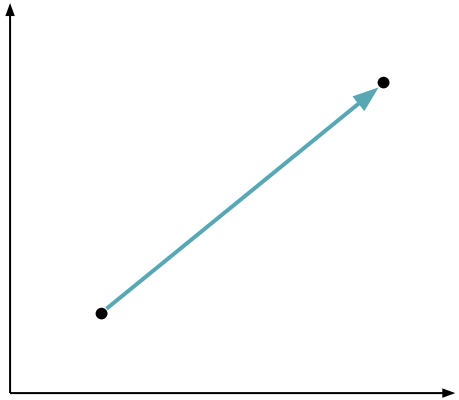


Nearest Neighbours Algorithm

1. Plot labelled training set examples using selected features

2. Given a new, unseen example, compute the distance from the new example to all our **n** training examples O(n)

3. Assign the example a classification based on the classification of the closest example

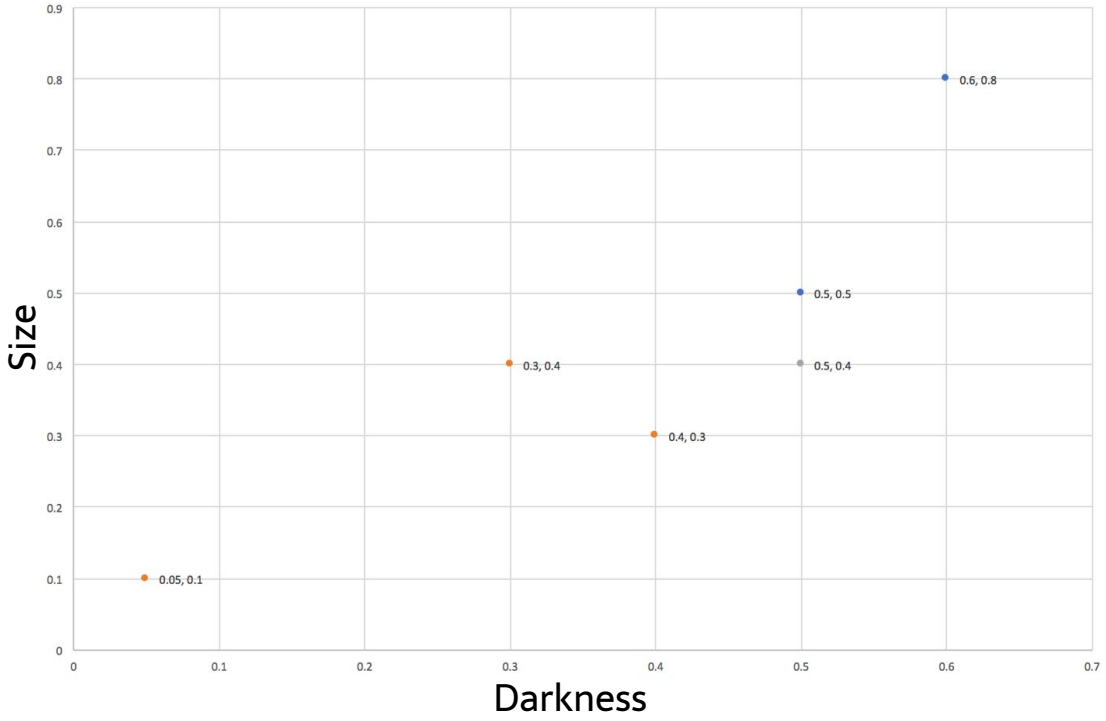


Nearest Neighbours Example

New example

Non-cancerous

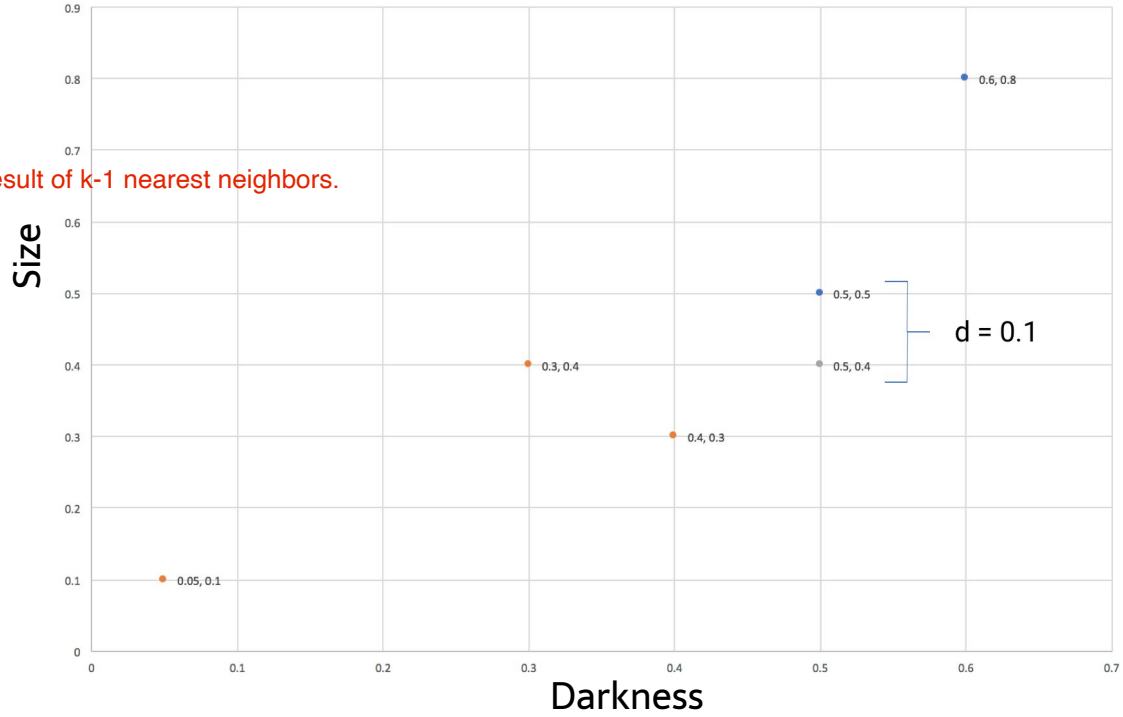
Cancerous



Nearest Neighbours Example

here is a tie, then you use the result of k-1 nearest neighbors.

- New example
- Non-cancerous
- Cancerous



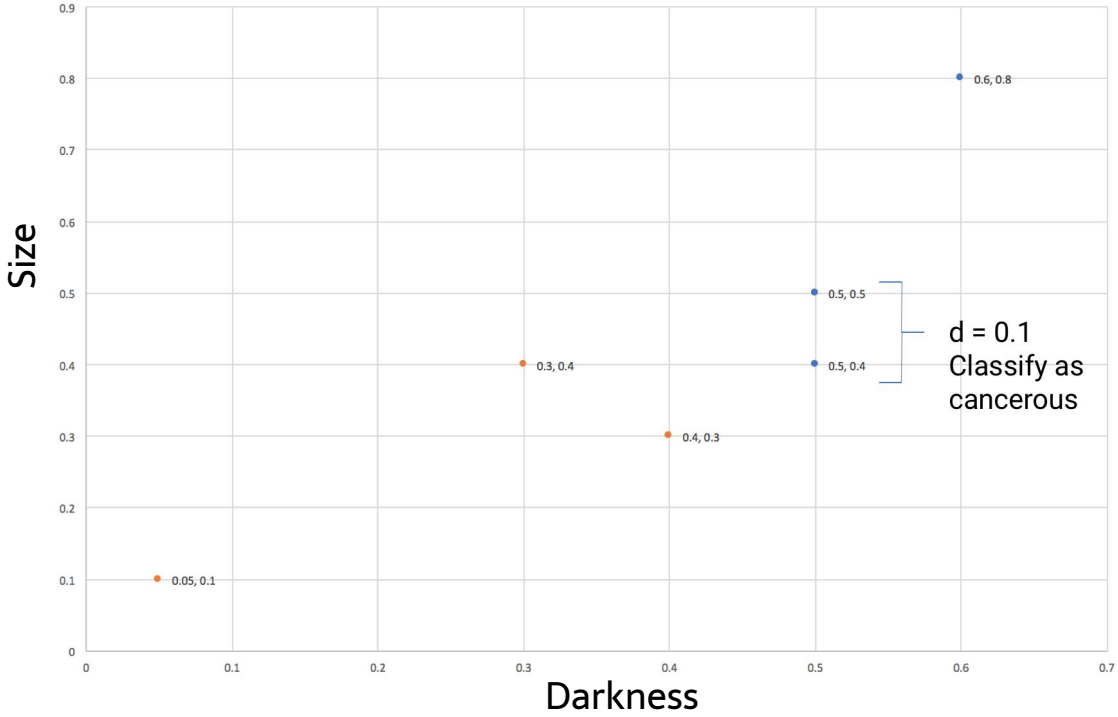
Distance metric (e.g. Euclidean distance) $\mathrm{d} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

Nearest Neighbours Example

New example

Non-cancerous

Cancerous



Distance metric (e.g. Euclidean distance) $\mathrm{d} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

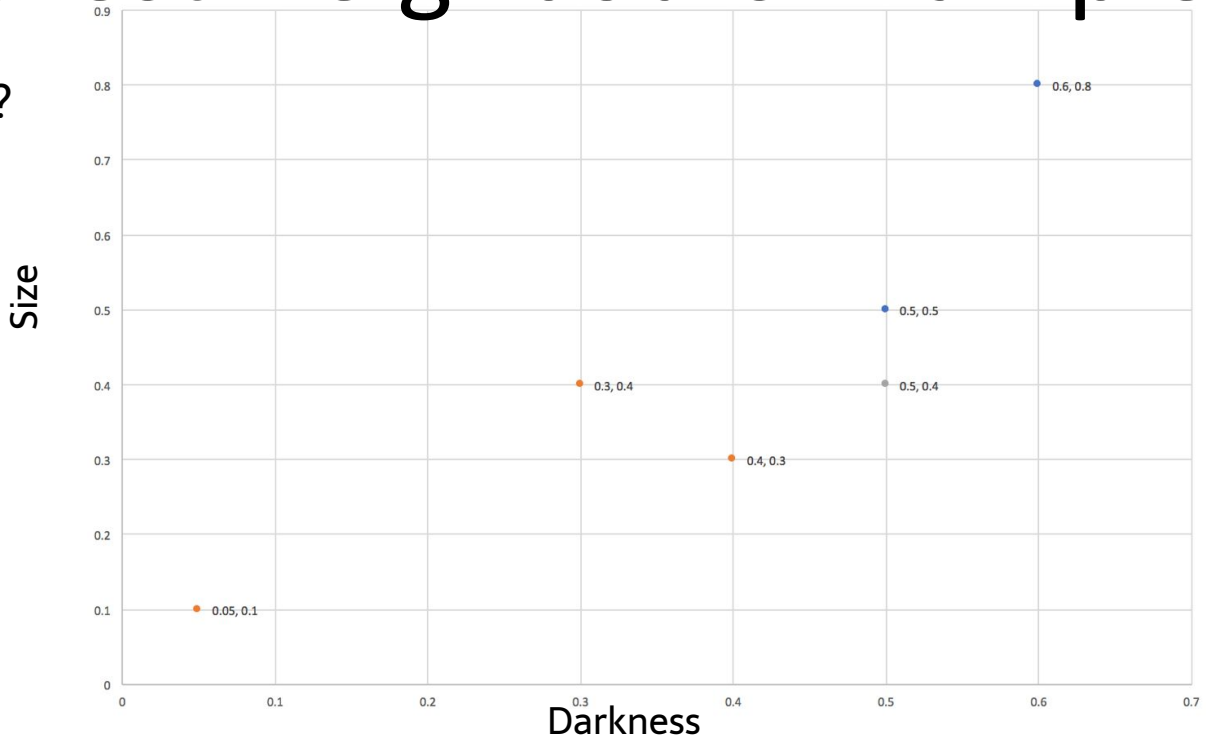
K-Nearest Neighbours Example

What if k = 2?

New example

Non-cancerous

Cancerous

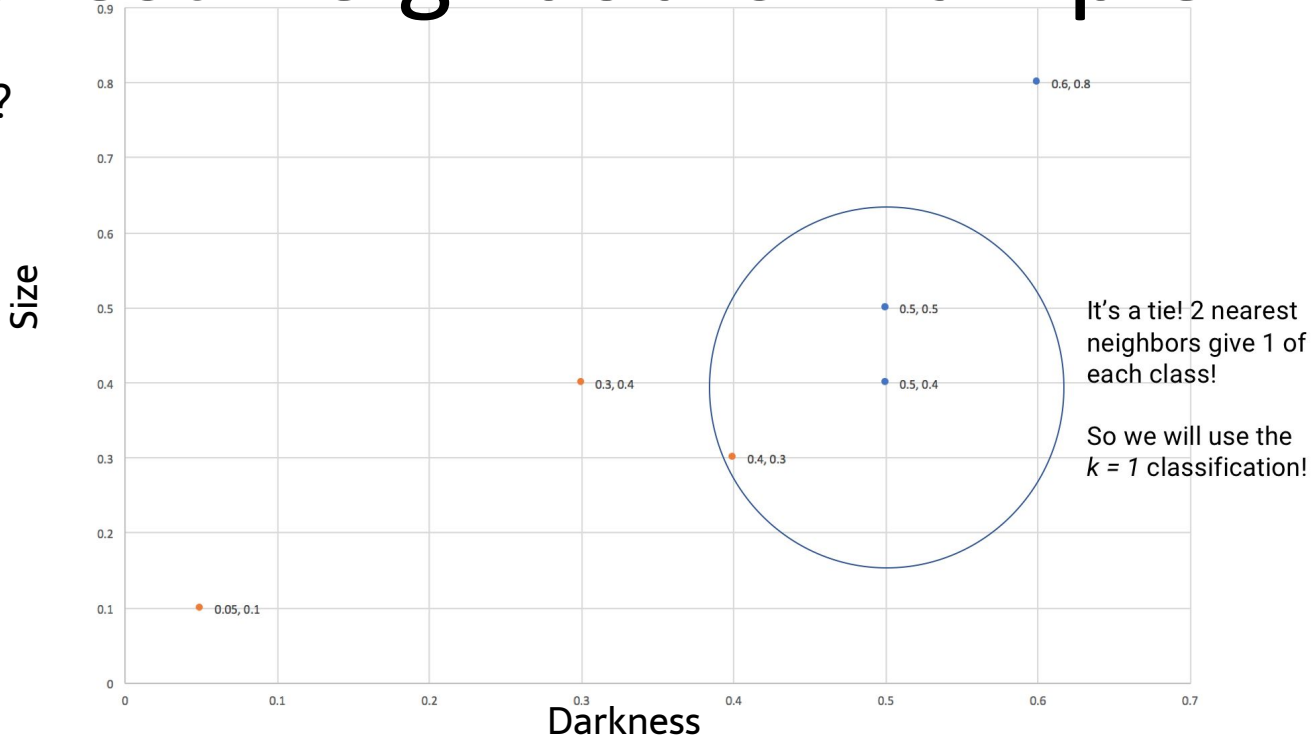


Distance formula: $\mathrm{d} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

K-Nearest Neighbours Example

What if k = 2?

- New example
- Non-cancerous
- Cancerous

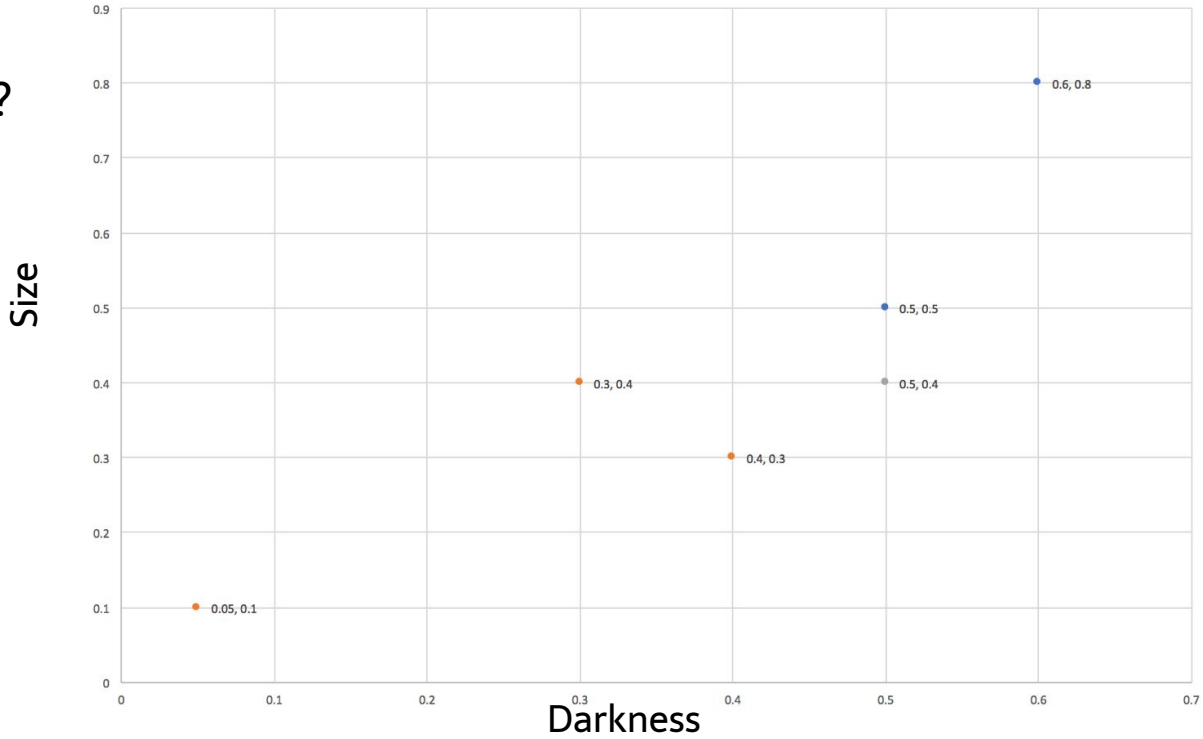


Distance formula: $\mathrm{d} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

K-Nearest Neighbours Example

What if k = 3?

- New example
- Non-cancerous
- Cancerous

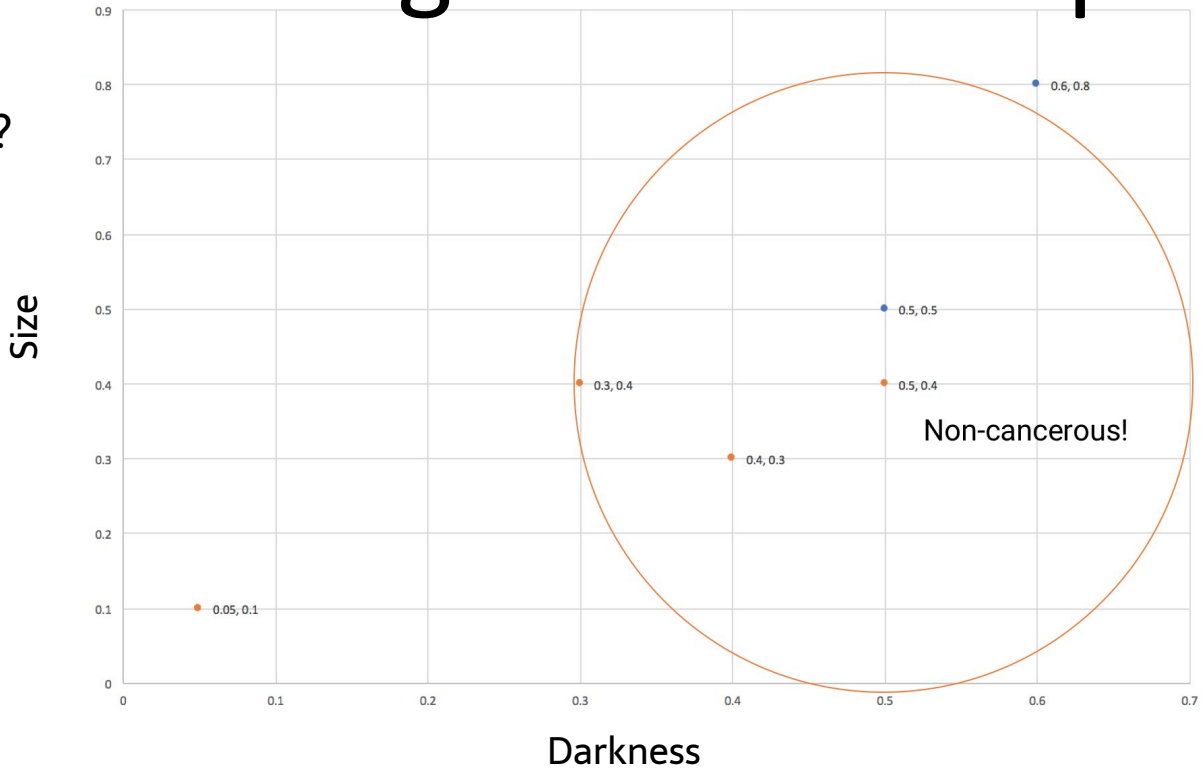


Distance formula: $d = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

K-Nearest Neighbours Example

What if k = 3?

- New example
- Non-cancerous
- Cancerous



Distance formula: $d = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

K-Nearest Neighbours (K-NN)

→ An extension of Nearest Neighbors

- Instead of only looking at the one closest example, base your classification off the *k* nearest examples
- More accurate for classification, because it looks for the majority classification among more examples

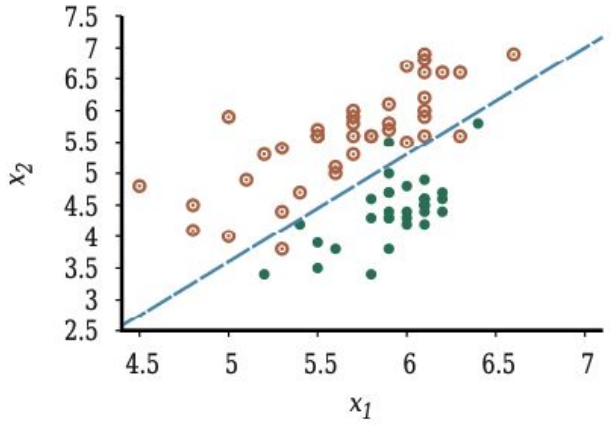
Classification: Use most common label of the *k* nearest examples

Regression: Use the mean of the *k* nearest examples, e.g.

If there is a tie, base the classification off the *k - 1* nearest examples

- 1-NN may suffice, and completes in O(1) Text
- Sensitive to local distribution and noise
- k is a parameter we pick

Another KNN example



Plot of two seismic data parameters, **body wave magnitude x_1** and **surface wave magnitude x_2**, for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998).

Classification problem: Is a new seismic data sample indicating an earthquake or an explosion?

The purple area is where the nuclear explosions belong to.
Is it possible to overfit and underfit?

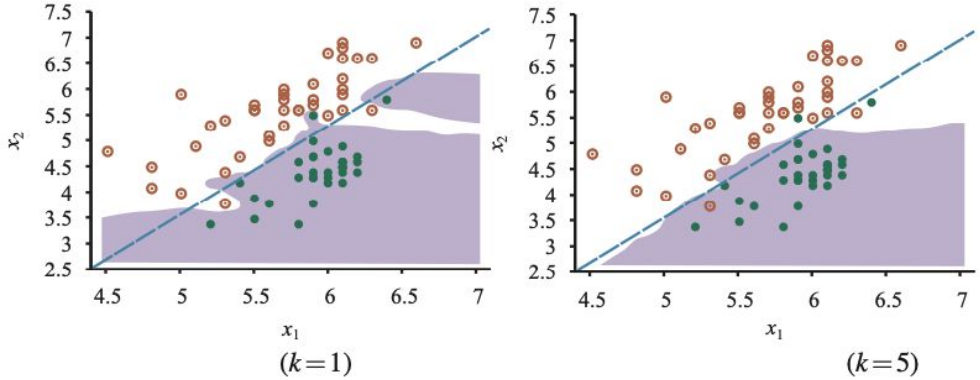


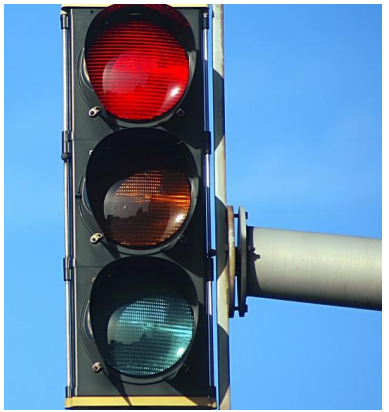
Figure 19.19 (a) A k-nearest-neighbors model showing the extent of the explosion class for the data in Figure 19.15, with $k=1$. Overfitting is apparent. (b) With $k=5$, the overfitting problem goes away for this data set.

Evaluation

How can we tell how good our model is?

- One way is to look at the **accuracy** of a model

- For **supervised learning**:
  - Say we have 100 images of street lights that we want to label as being green, yellow, or red
  - If our model correctly classifies 95 of these 100 images, then we say our model has a 95% accuracy



Other Metrics for Evaluation

- However, **accuracy** doesn't tell us the whole picture
- There are different types of mistakes that we can make

Take the example of detecting whether there is a fire in your home or not

- **False Positive (FP)**: When you think there is a fire, but there isn't
- **False Negative (FN)**: When you think there is not a fire, but there is

Are these errors equally bad?

Other Metrics for Evaluation

	Actual: Yes	**Actual: No**
Predicted: Yes	True Positive (TP)	False Positive (FP)
Predicted: No	False Negative (FN)	True Negative (TN)

This table, when filled in with actual numbers, is called a **confusion matrix**

Other Metrics for Evaluation

- **Precision**: Probability that a positive prediction is correct, TP / (TP + FP)
- **Recall** (or Sensitivity): Probability that an actual positive outcome is predicted correctly, TP / (TP + FN)
- **Specificity**: Probability that an actual negative outcome is predicted correctly, TN / (TN + FP)
- **F1 Score**: Combination of precision and recall,

  (2 * Precision * Recall) / (Precision + Recall)

	Actual: Yes	Actual: No
Predicted: Yes	True Positive (TP)	False Positive (FP)
Predicted: No	False Negative (FN)	True Negative (TN)

Evaluation Example

F1 score helps you avoid making the same guess.

Actual	Predicted	TP/FP/TN/FN?
Fire	Fire	TP
No Fire	No Fire	TN
Fire	Fire	TP
No Fire	Fire	FP
Fire	Fire	TP
No Fire	Fire	FP
Fire	No Fire	FN
No Fire	Fire	FP
No Fire	No Fire	TN
No Fire	No Fire	TN

Accuracy? 6/10 = 3/5 = 0.6

TP: 3	FP: 3
FN: 1	TN: 3

Precision: TP / (TP + FP) = ? 3/6 = 50%

Recall: TP / (TP + FN) = ? 3/4 = .75

Specificity: TN / (TN + FP) = ? 3/6 = .5

F1 Score: (2 * P * R) / (P + R) = ? 2 *.5 * .75 / (1.25) = 0.75 / 1.25 = 3/5 = 0.6

Memorization vs. Generalization

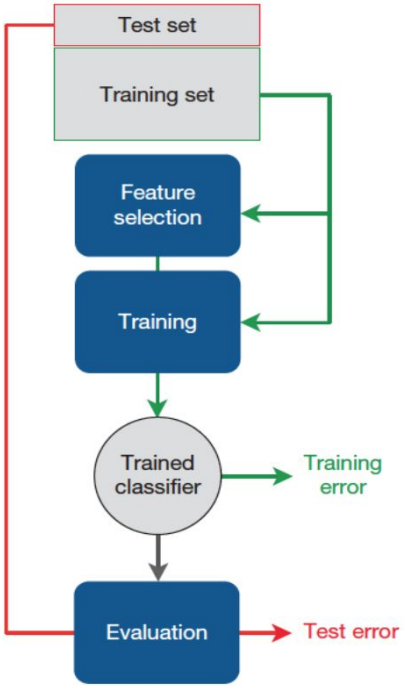
You can underfit and overfit at the same time.

- **Memorization:** Ability to do well on data we have already seen
- **Generalization:** Ability to do well on data we have not seen
- We do not want to just memorize → overfitting
- Math test example:
  - Memorizing would be memorizing the answers to the specific questions asked on a practice test
  - Generalizing would be understanding the rules used in the practice test and applying them to the actual test

Maximizing accuracy or minimizing error?

How do we learn to generalize well?
We have to evaluate our algorithms on data it hasn't seen before.

- To do this, we typically split the data we have into two sets:
  - **Training Set**: data used to train a model
  - **Test Set**: data used to test a model
- We then measure performance in two ways:
  - **Training Error**: how well the model performs on data we have seen
  - **Test Error**: how well the model performs on unseen data

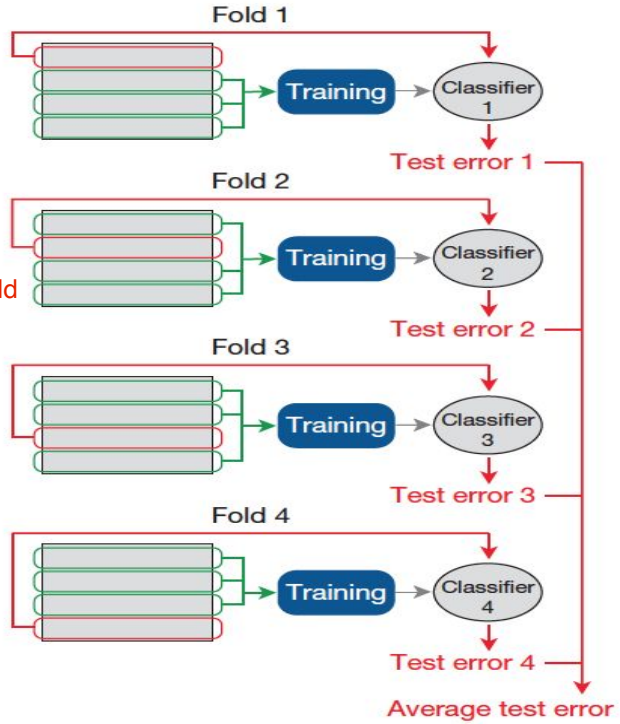


Cross-Validation

- Especially useful if dataset is small
- Break your data down into a training and testing set, e.g. 75-25 split
- For each “fold” we ensure that training and test data is disjoint
- To balance the size of test and training set, we can repeat this many times

We want to forget the parameters in order to make sure the old parameter values are not passed on.

E.g. 4-fold cross-validation



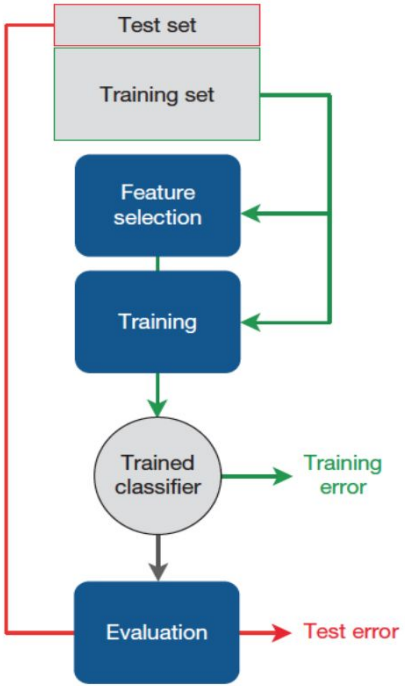
If your dataset is small, the issue is that the data used for training the model may contain enough information such that the model can overfit. By leaving out data pertaining to a certain group and using it for the testing data, you can make sure the model generalizes.

No "cheating" - using a dev / validation set

- **Training Set**: data used to train a model
- **Test Set**: data used to test a model

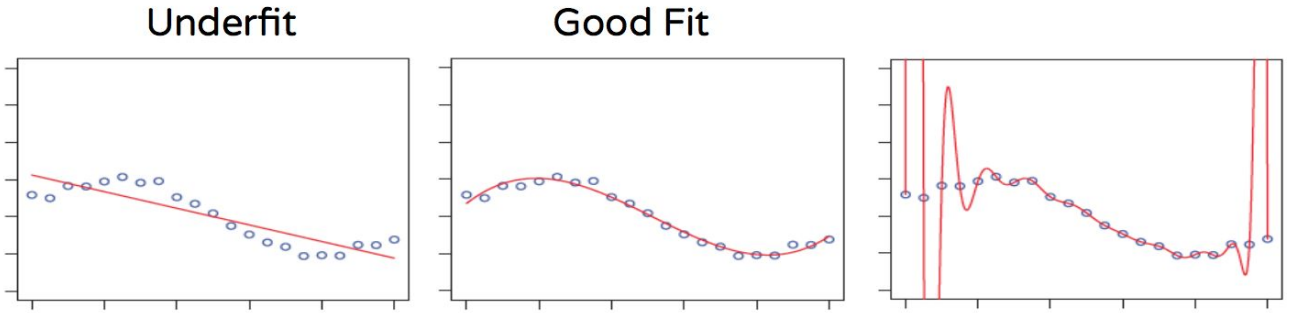
More recently, we introduce a **validation** set that we can use to help us train the model. The true **test set** is held out so it is never "seen" during training.

Want to hide test set to make sure the model cannot memorize the information in the test set.
Want to only use the test set once.



Fitting to your data

- When we don't even fit well to the data we've seen, this is called **underfitting**
- When we fit too well to the data we've seen, but do not generalize well, this is called **overfitting**



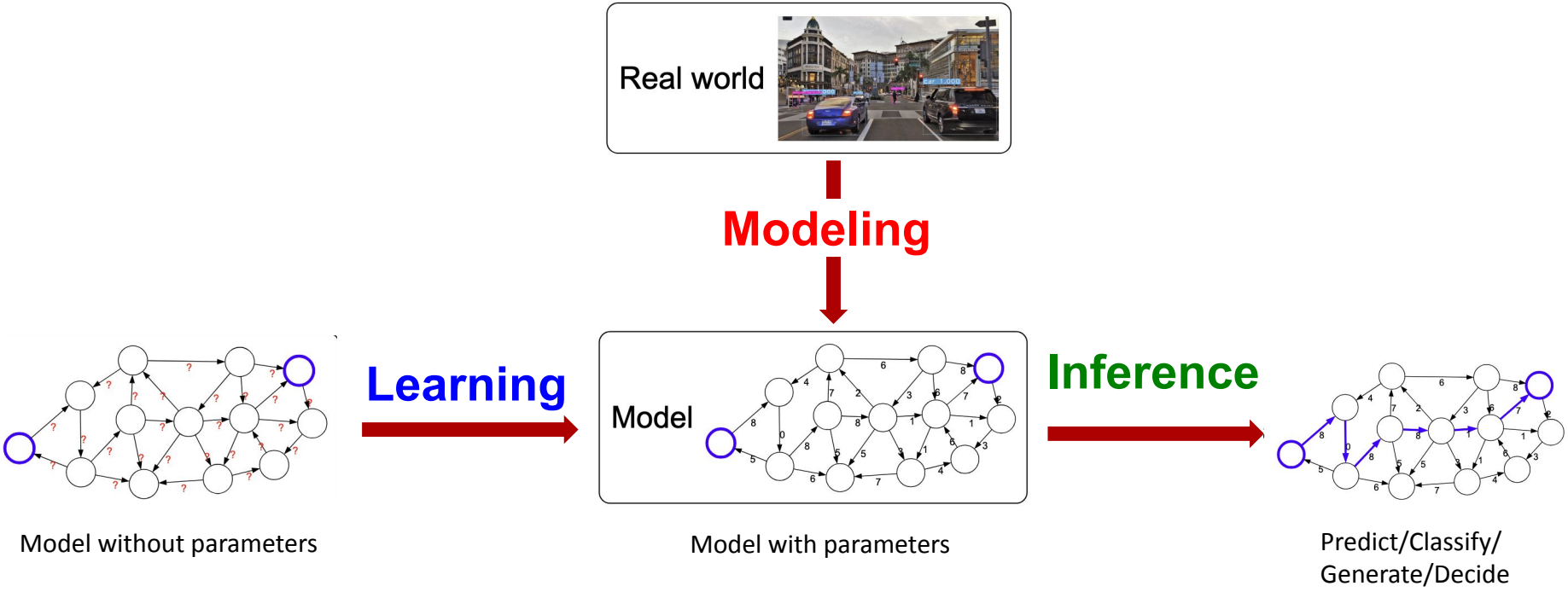
Evaluation

For the same problem type (e.g. classification), you can try multiple models to solve it.

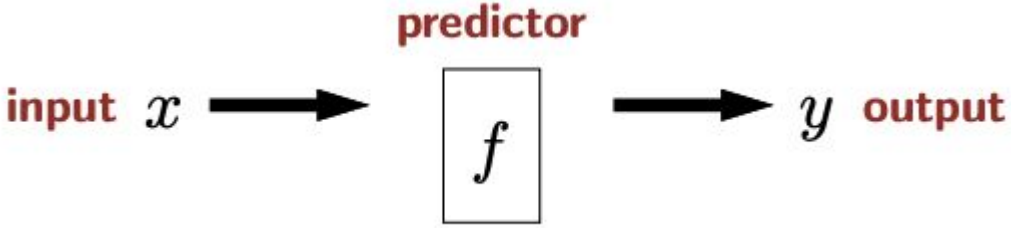
Model	Accuracy	F1 Score
Decision Tree		
KNN		

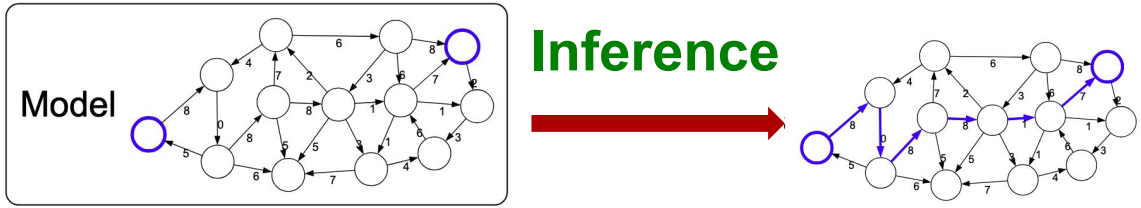
Formalizations

Recall: modeling-learning-inference



Recall: modeling-learning-inference

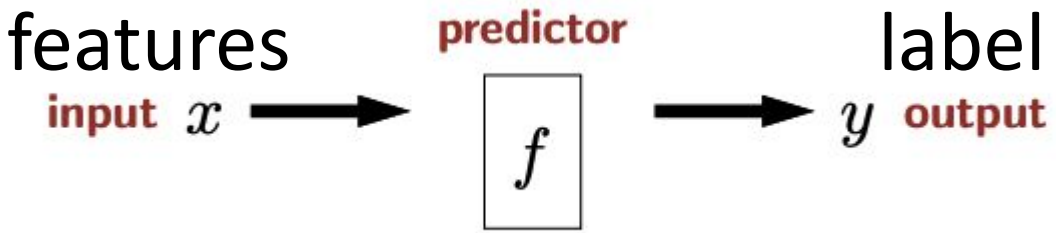




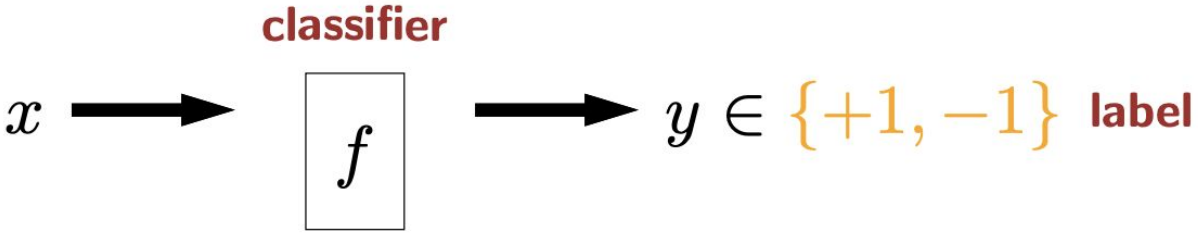
Model with parameters

Predict/Classify/
Generate/Decide

Machine Learning



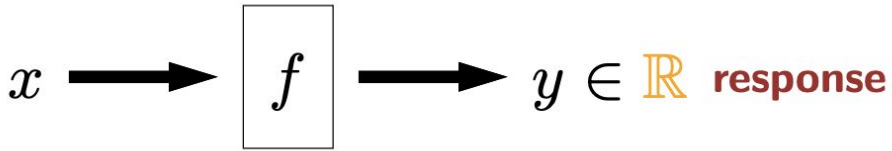
Binary Classification



Spam detection: online comment → toxic or not toxic
Earthquake: measurements of event → earthquake or explosion
Melanoma: mole size and color → cancerous or non-cancerous

Extension: **multiclass** classification $y \in \{1, \ldots, K\}$

Regression



Housing: information about house ⟶ price
Arrival times: destination, weather, time ⟶ time of arrival
Robotic vision: image, xy-coord ⟶ distance

Structured prediction

$x \longrightarrow \boxed{f} \longrightarrow y$ is a complex object

Machine translation: English sentence → Japanese sentence

Dialogue: conversation history → next utterance

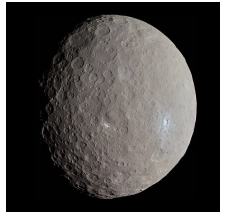
Image captioning: image → sentence describing image

Image segmentation: image → segmentation

Supervised Learning

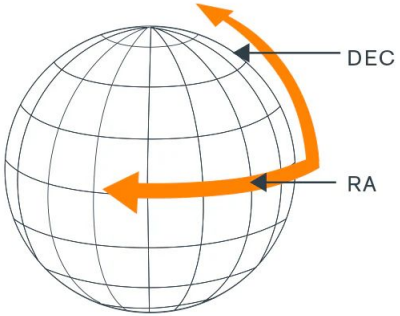
Regression

History of regression

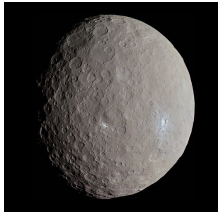


In 1801, astronomer Piazzi discovered Ceres, the first known asteroid, and made 19 observations of its location before it was obscured by the sun.
Astronomers wondered: when and where would Ceres be observed again?

Time	Right ascension	Declination
Jan 01, 20:43:17.8	50.91	15.24
Jan 02, 20:39:04.6	50.84	15.30
...	...	...
Feb 11, 18:11:58.2	53.51	18.43

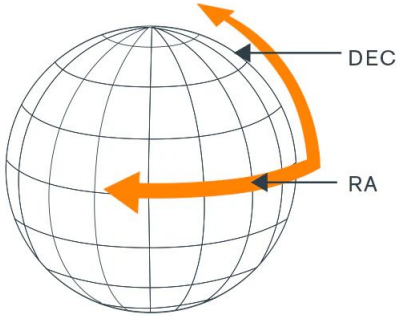


History of regression



Sept 1801: Gauss took Piazzi's data, created a model of Ceres' orbit, and made a prediction.

Time	Right ascension	Declination
Jan 01, 20:43:17.8	50.91	15.24
Jan 02, 20:39:04.6	50.84	15.30
...	...	...
Feb 11, 18:11:58.2	53.51	18.43

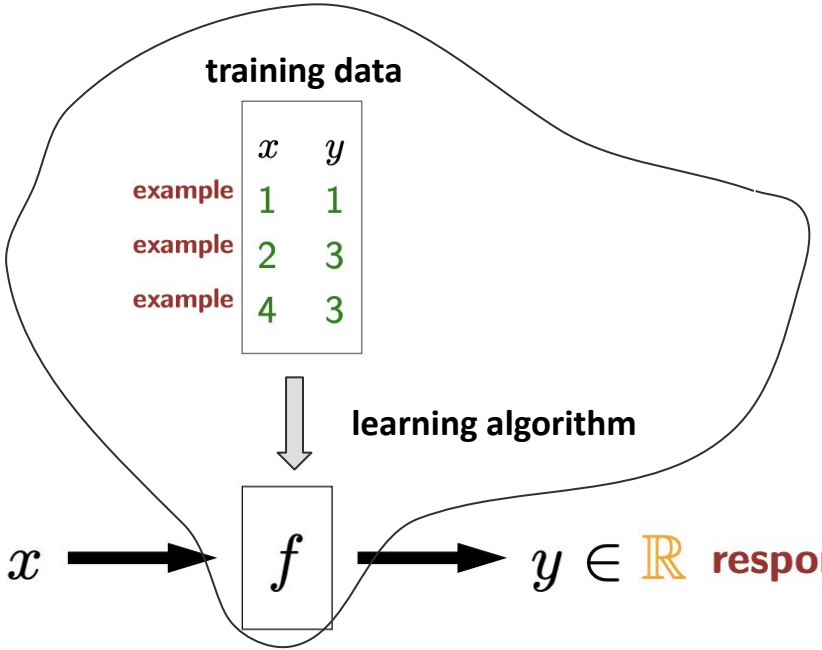


December 7, 1801: Ceres was located within 1/2 degree of Gauss's prediction, much more accurate than other astronomers. His method: **least squares linear regression**

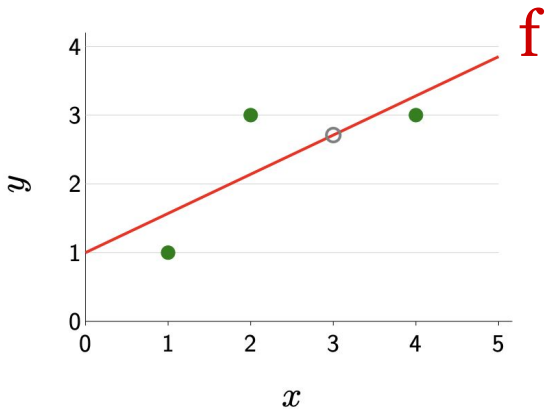
Linear Regression

Next week!

Linear regression framework



Training → finding our predictor f



3 input → f predictor → 2.71 output

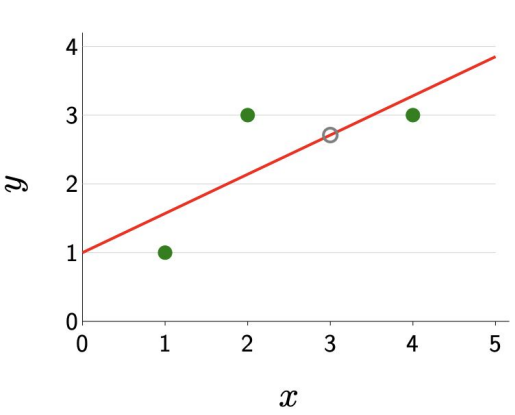
Which predictors **f** is the learning algorithm allowed to produce? Lines? Curves? Decide the **hypothesis class**.

How good is a predictor? Decide the **loss function**.

How do we find the best predictor? Decide the **optimization algorithm**.

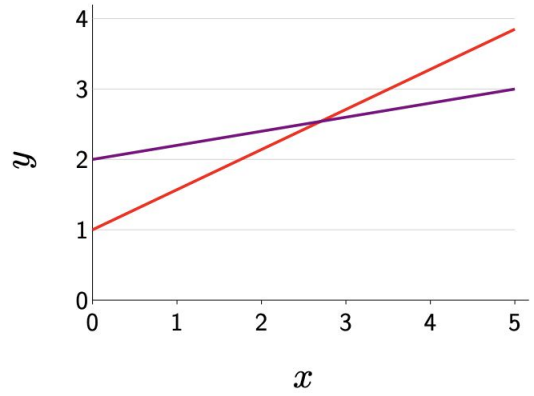
Hypothesis class: Which predictors to use?

Training → finding our predictor f



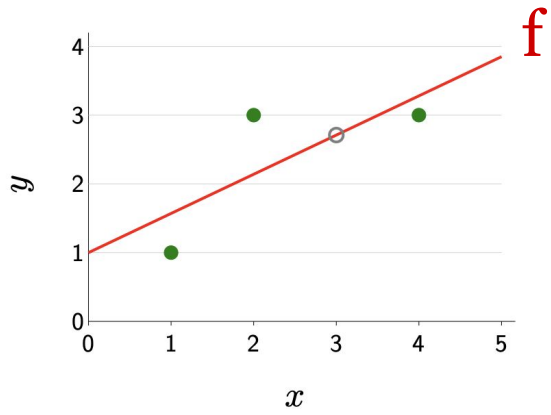
$$f(x) = 1 + 0.57x$$

$$f(x) = w_1 + w_2 x$$



$$f(x) = 2 + 0.2x$$

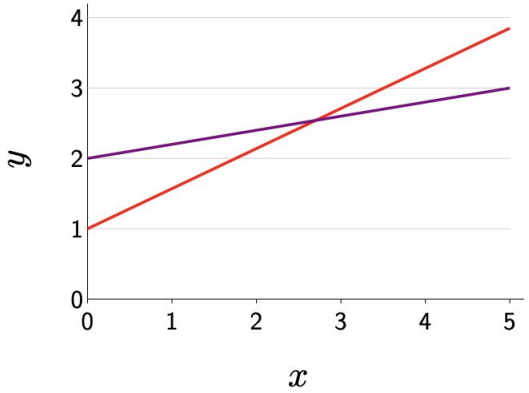
Hypothesis class: Which predictors to use?



$$f(x) = 1 + 0.57x$$

$[1, x]$ feature vector

$$f(x) = w_1 + w_2 x$$



$$f(x) = 2 + 0.2x$$

weight vector $\mathbf{w} = [w_1, w_2]$

Value of weight does not necessarily correspond to a higher/lower importance given to a feature

$$f_{\mathbf{w}}(3) = [1, 0.57] \cdot [1, 3] = 2.71$$

This week

- Do the online weekly activity by Monday at noon (published ~tomorrow) as a primer for gradients
- Read Chapter 19
- Check Appendix A - Mathematical Background
- Next week, we'll continue Machine Learning and look at gradients