

A blurred background image of a person's hands typing on a laptop keyboard, with the laptop screen showing some code or data. The image has a dark gray overlay.

Intro to AI & Machine Learning

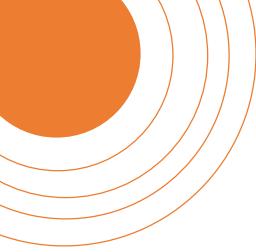


Table of Contents

1. What is machine learning? : 3
 2. Types of machine learning: 28
 3. Drawing a line close to our points: Linear regression: 50
 4. Optimizing the training process: Underfitting, overfitting, testing, and regularization: 85
 5. Using lines to split our points: The perceptron algorithm: 111
 6. A continuous approach to splitting points: Logistic regression: 174
 7. How do you measure classification models? Accuracy and its friends: 220
- 

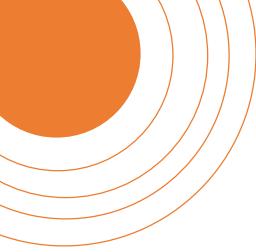


Table of Contents

8. Using probability to its maximum: The naive Bayes algorithm: 249
 9. Splitting data by asking questions: Decision trees: 294
 10. Combining building blocks to gain more power: Neural Networks: 368
 11. Combining models to maximize Results Ensemble learning: 432
 12. Finding boundaries with style: Support vector machines and the kernel method: 468
 13. Putting it in practice: A real-life example of data engineering and machine learning: 524
- 



1: What is machine learning?

What is machine learning?

This Lesson covers:

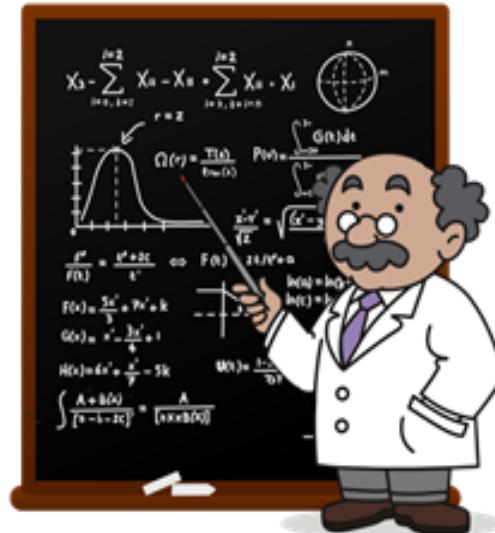
- What is machine learning?
- Is machine learning hard? (Spoiler: No)
- Why you should take this course?
- What will we learn in this course?
- How do humans think, how do machines think, and what does this have to do with machine learning?

Why this Course?

- This course is a step on that journey, that I'm very happy you're taking with me!



Music



Machine learning

- There is also a melody, and in this course we sing it.
- In the same way, machine learning is not about formulas and code.
- Music is not only about scales and notes. There is a melody behind all the technicalities.

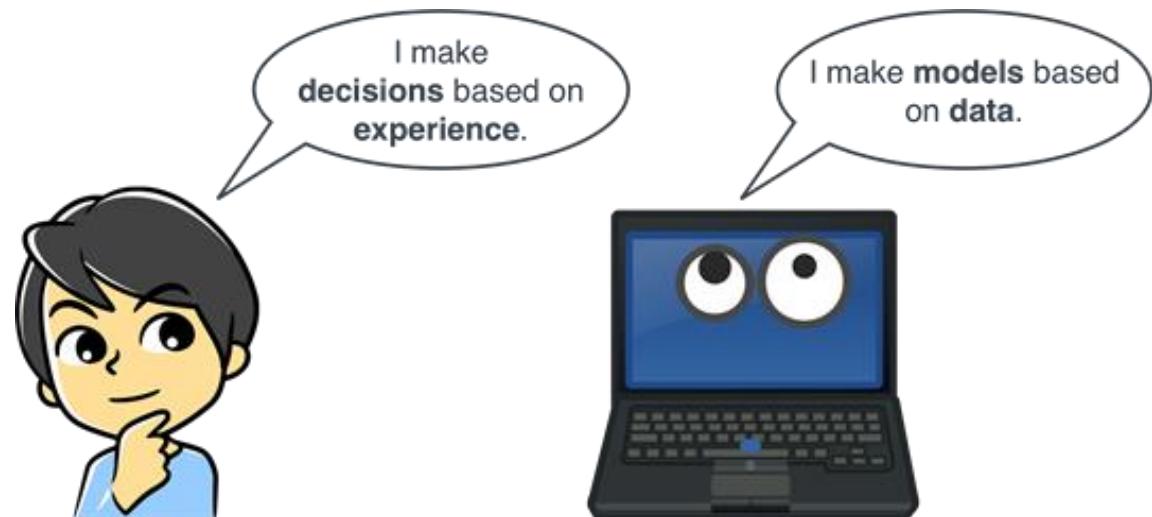
But what exactly is machine learning?

- Once upon a time, if we wanted to make a computer perform a task, we had to write a program, namely, a whole set of instructions for the computer to follow.
- This is good for simple tasks, but how do we get a computer to, for example, identify what is on an image?
- For example, is there a car on it, is there a person on it.

But what exactly is machine learning?

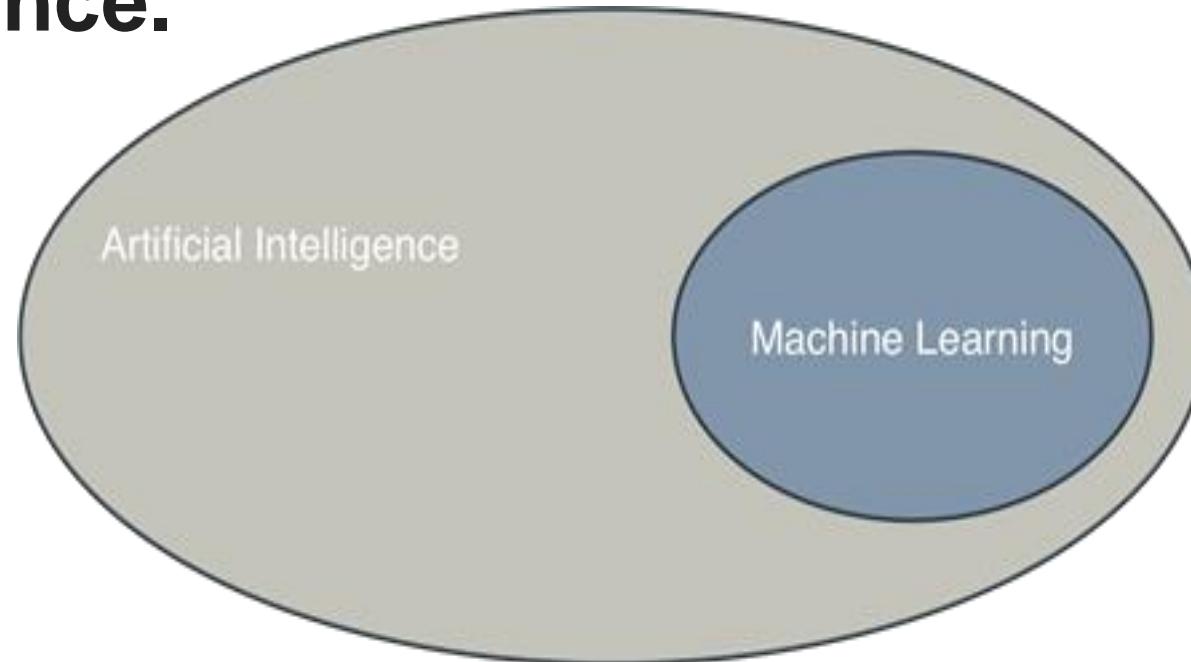
- Machine learning is common sense, except done by a computer.

In the same way that humans make decisions based on previous experiences, computers can make decisions based on previous data. The rules computers use to make decisions are called models. Machine learning is about computers making decisions based on experience.



What is the difference between artificial intelligence and machine learning?

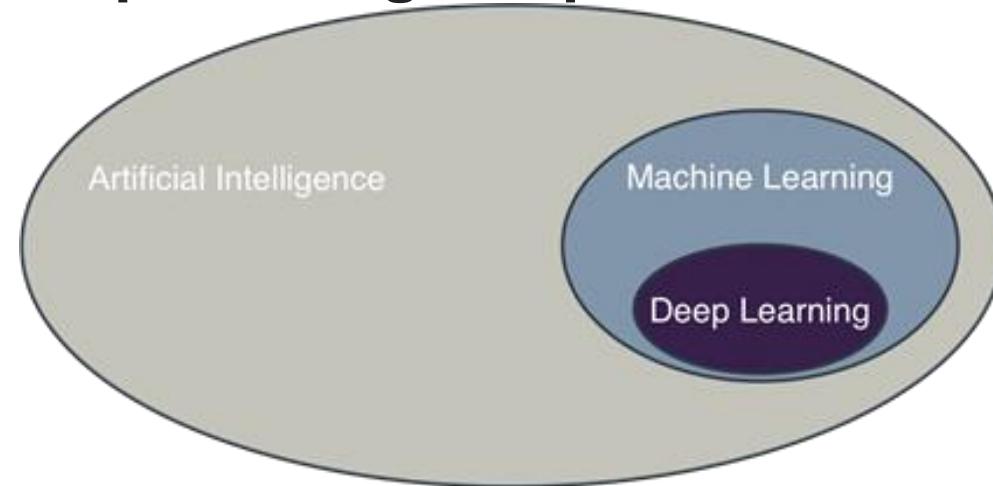
Machine learning is a part of artificial intelligence.



What about deep learning?

- If this course was about vehicles, then AI would be motion, ML would be cars, and deep learning (DL) would be Ferraris.

Deep learning is a part of machine learning.



Humans use the remember-formulate-predict framework to make decisions (and so can machines!)

- How does the computer make decisions based on previous data?
- For this, let's first see the process of how humans make decisions based on experience.
- And this is what I call the remember-formulate-predict framework.

How do humans think?

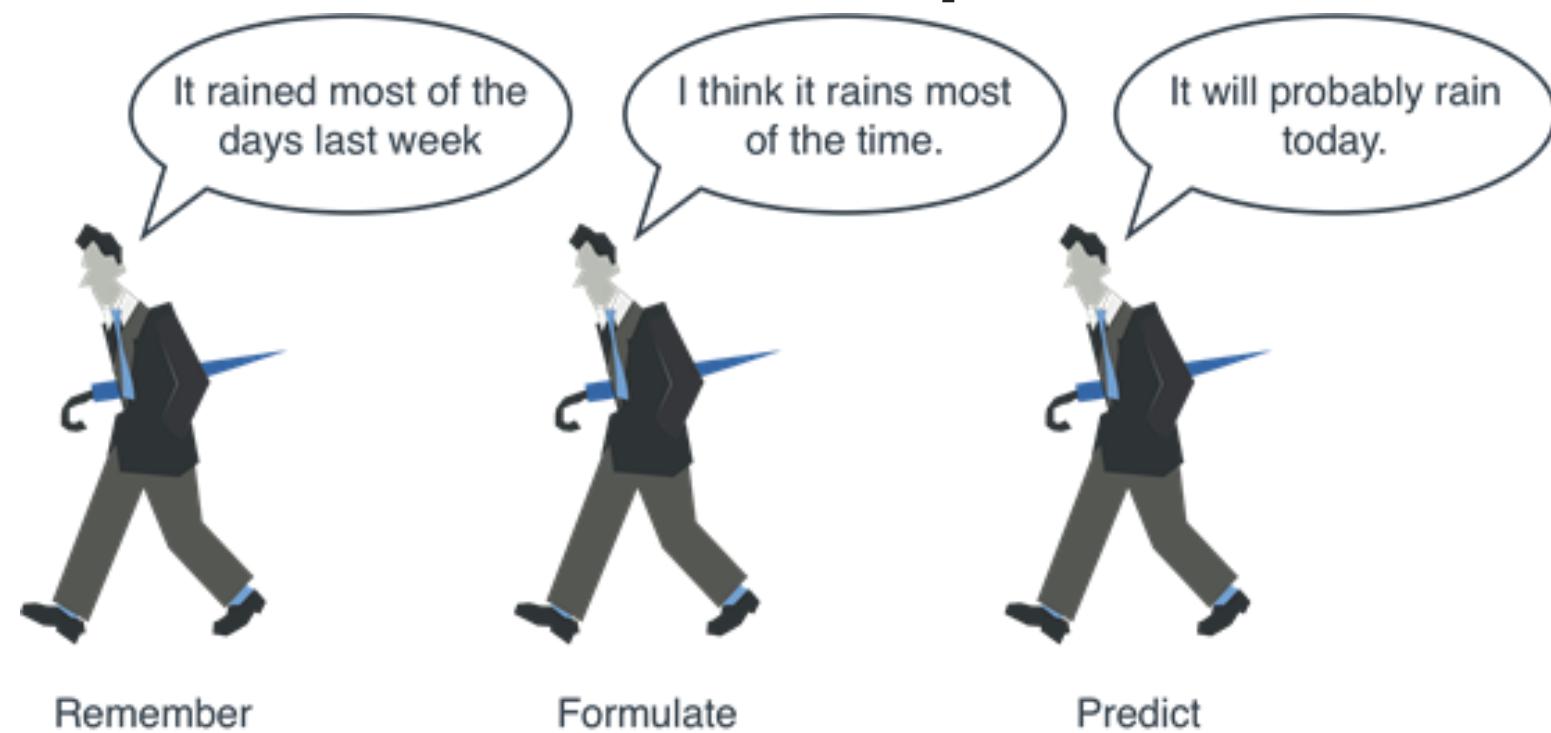
When we humans need to make a decision based on our experience, we normally use the following framework:

1. We remember past situations that were similar.
2. We formulate a general rule.
3. We use rules to predict what will happen if we take a certain decision.

How do humans think?

- We may be right or wrong, but at least, we are trying to make an accurate prediction.

The remember-formulate-predict framework



Remember

Formulate

Predict

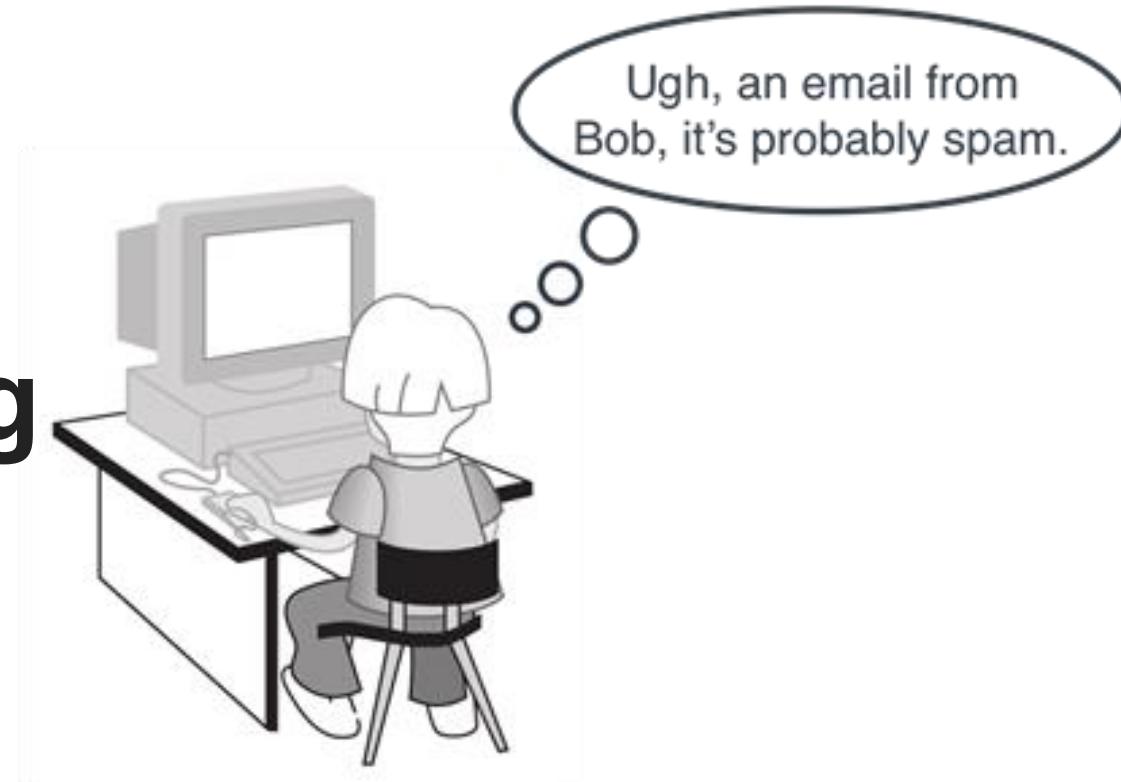
How do humans think?

Rule 1: 4 out of every 10 emails that Bob sends us are spam.

How do humans think?

- Let's try to analyze the emails a little more.
- Let's see when Bob sent the emails to see if we find a pattern.

**A very simple
machine learning
model.**



How do humans think?

Monday: Ham

Tuesday: Ham

Saturday: Spam

Sunday: Spam

Sunday: Spam

Wednesday: Ham

Friday: Ham

Saturday: Spam

Tuesday: Ham

Thursday: Ham

How do humans think?

- In the next we'll see a few more features.

A slightly more complex machine learning model, done by a human.



How do humans think?

Example 3: Things are getting complicated!

- Now, let's say we continue with this rule, and one day we see Bob in the street, and he says “Why didn't you come to my birthday party?”
- We have no idea what he is talking about.

How do humans think?

1KB: Ham

12KB: Ham

16KB: Spam

20KB: Spam

18KB: Spam

5KB: Ham

25KB: Spam

1KB: Ham

3KB: Ham

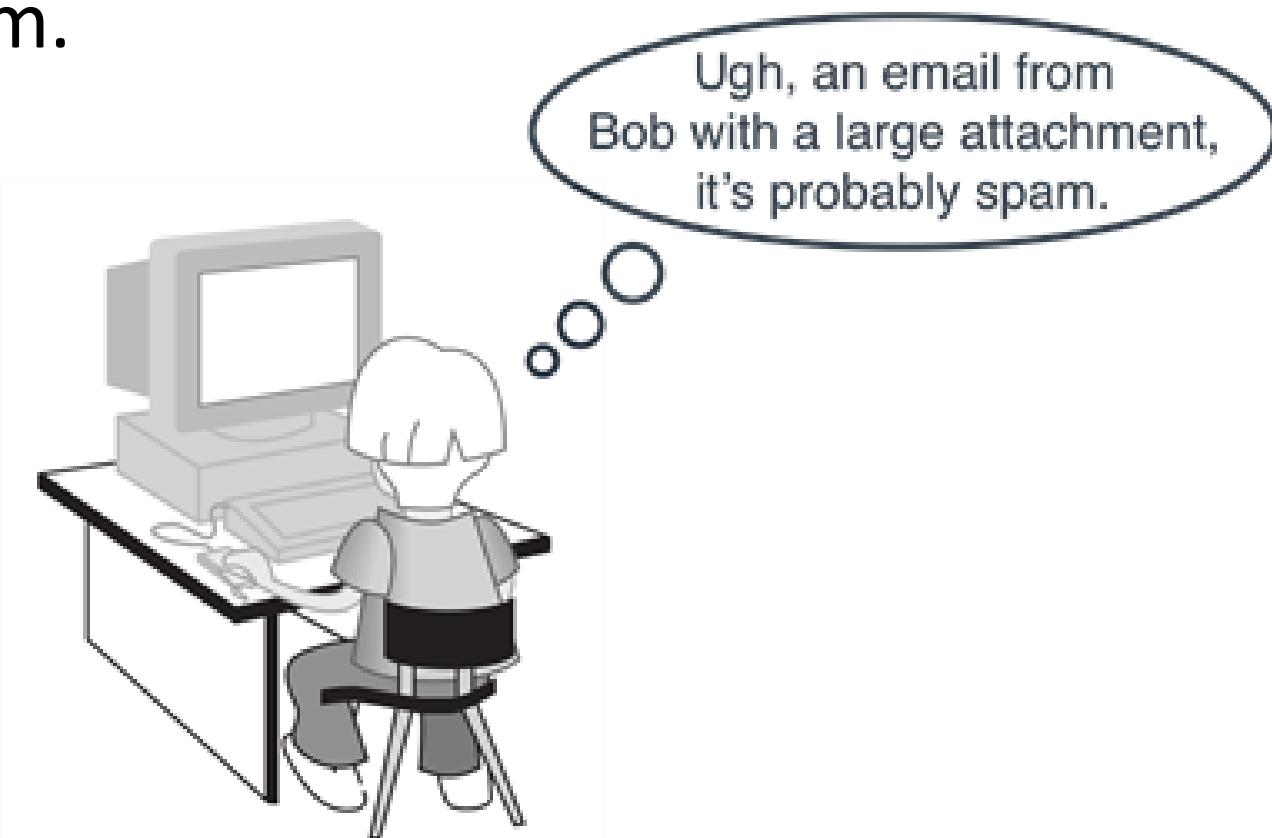
How do humans think?

**Rule 3: Any email larger of size 10KB or more is spam,
and any email of size less than 10KB is ham.**

How do humans think?

- We look at the email we received today, and the size is 19KB.
- So we conclude that it is spam.

Another slightly more complex machine learning model, done by a human.



How do humans think?

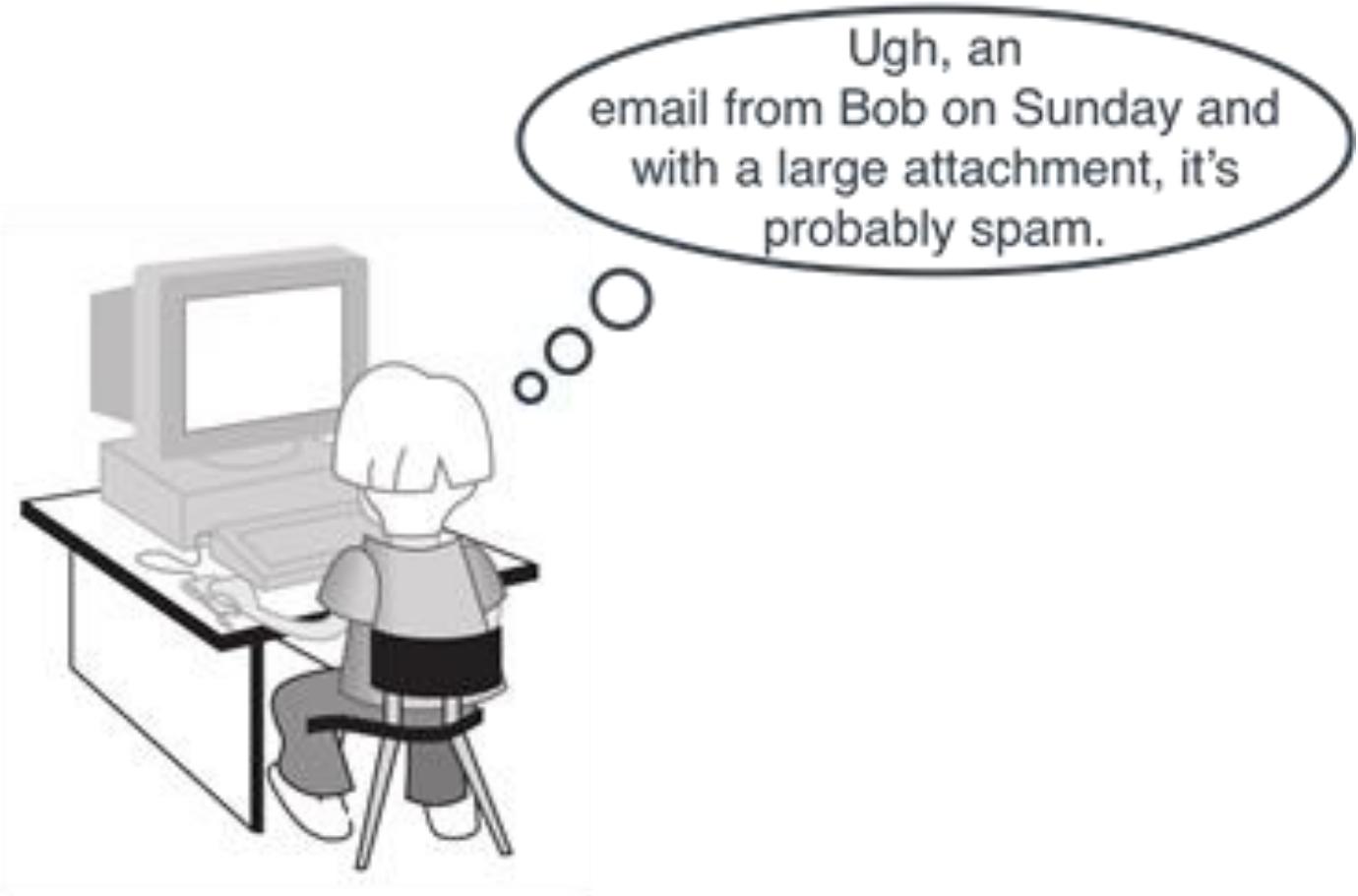
Rule 4: If an email is larger than 10KB or it is sent on the weekend, then it is classified as spam. Otherwise, it is classified as ham.

Rule 5: If the email is sent during the week, then it must be larger than 15KB to be classified as spam. If it is sent during the weekend, then it must be larger than 5KB to be classified as spam. Otherwise, it is classified as ham.

How do humans think?

- It is classified as ham.

An even more complex machine learning model, done by a human.



How do machines think?

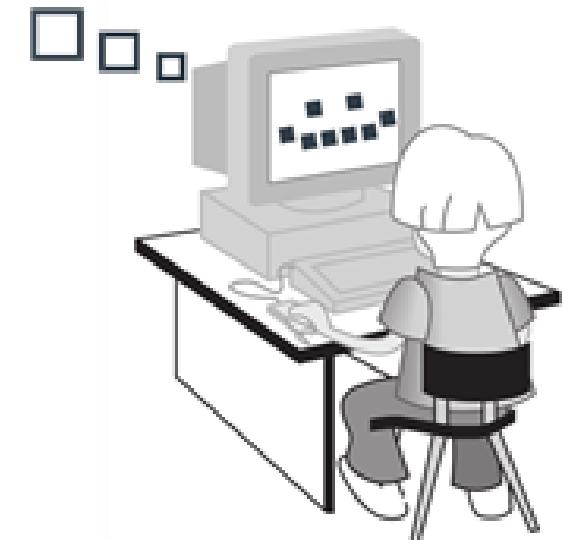
- **Remember:** Look at a huge table of data.
- **Formulate:** Go through many rules and formulas, and check which one fits the data best.
- **Predict:** Use the rule to make predictions about future data.

How do machines think?

- then we classify the message as spam.
- Otherwise we do not.

- Email from bob@email.com
- on Sunday after 3pm
- size > 10KB
- Contains the word "buy"
- It's probably spam.

A much more complex machine learning model, done by a computer.



What is this course about?

Good question. The 8 rules we discussed previously are examples of machine learning models, or classifiers. As you saw, these are of different types.

- Some use an equation on the features to make a prediction.
- Others use a combination of if statements.
- Others will return the answer as a probability.
- Others may even return the answer as a number!

Summary

As follows:

- **Remember:** Use previous data.
- **Formulate:** Build a model, or a rule, for this data.
- **Predict:** Use the model to make predictions about future data.

2: Types of machine learning

Types of machine learning

This Lesson covers:

- Three main different types of machine learning.
- The difference between labelled and unlabeled data.
- What supervised learning is and what it's useful for.
- The difference between regression and classification, and what are they useful for.
- What unsupervised learning is and what it's useful for.
- What reinforcement learning is and what it's useful for

What is the difference between labelled and unlabelled data?

Actually, what is data?

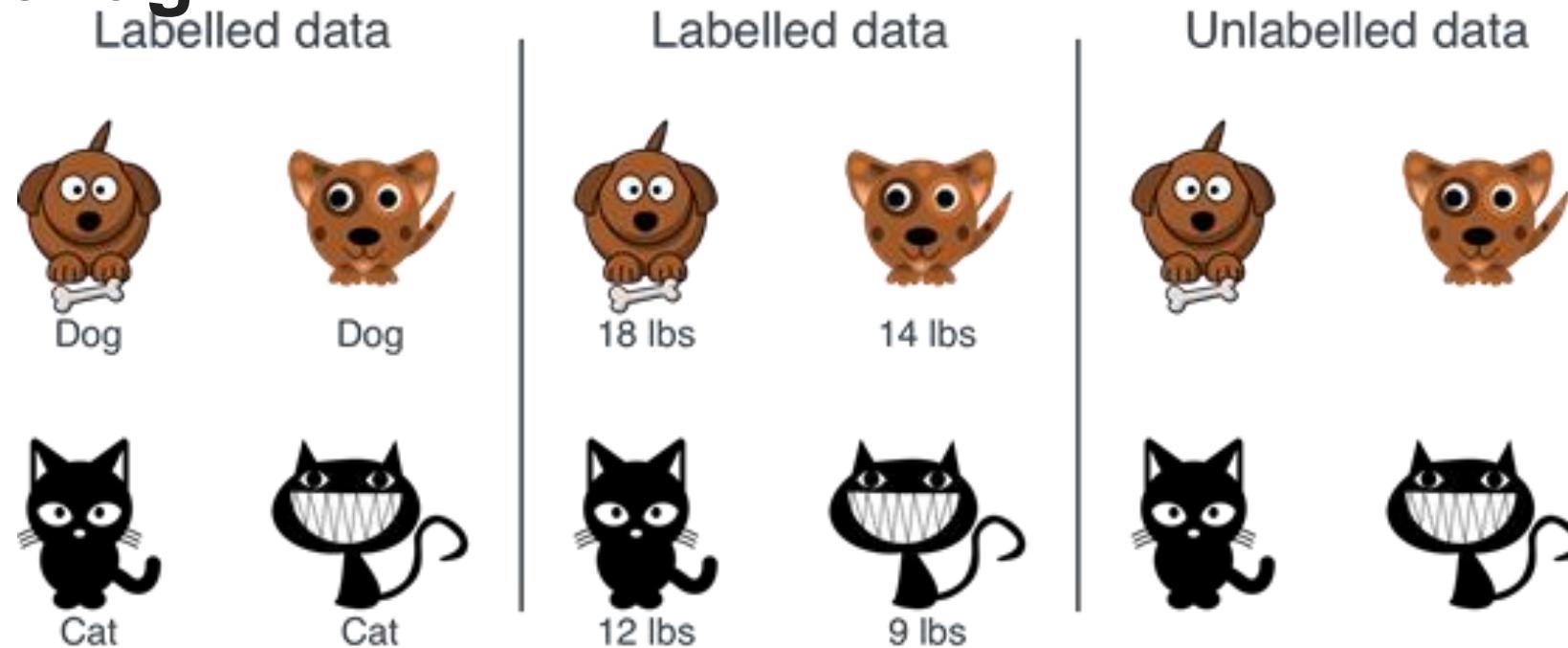
- Let's first establish a clear definition of what we mean by data.

Data is simply information.

- Any time we have a table with information, we have data.
Normally, each row is a data point.

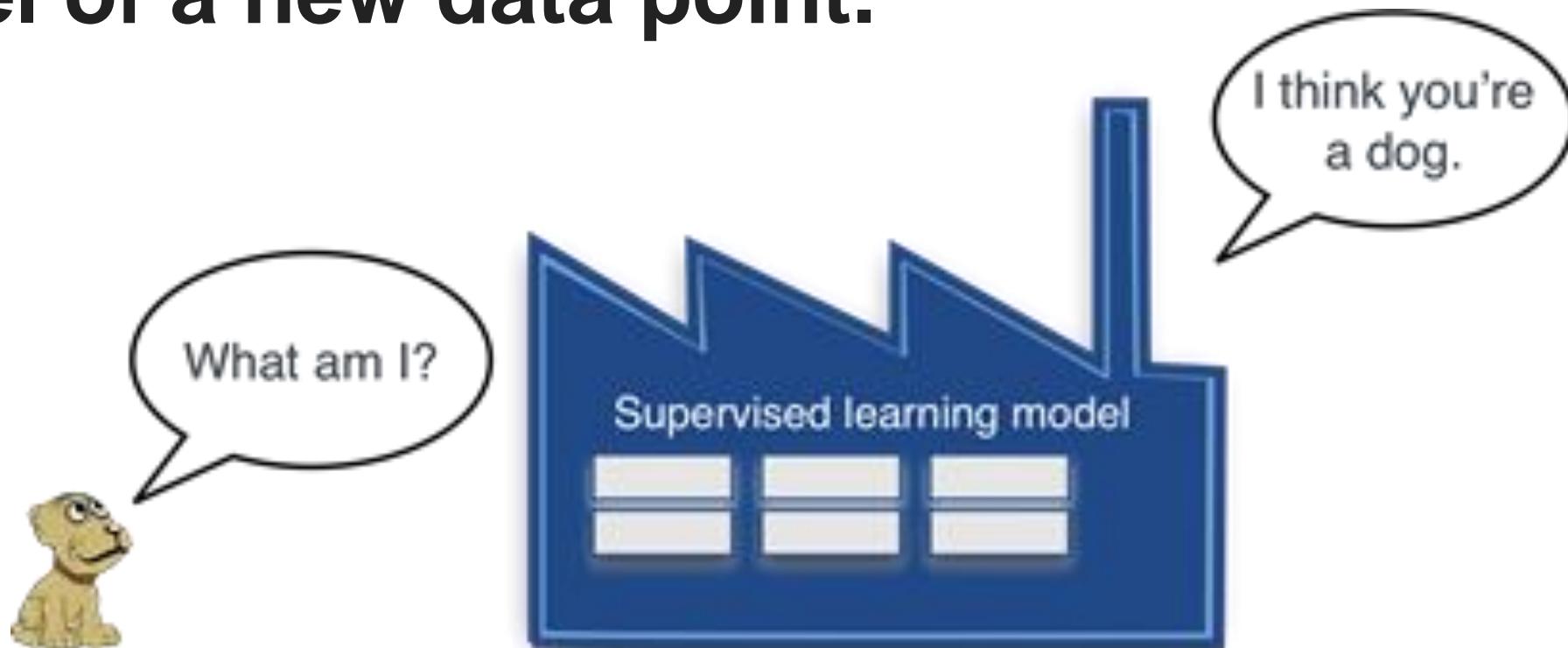
What is the difference between labelled and unlabelled data?

Labeled data is data that comes with a tag, like a name, a type, or a number. Unlabeled data is data that comes with no tag.



What is supervised learning?

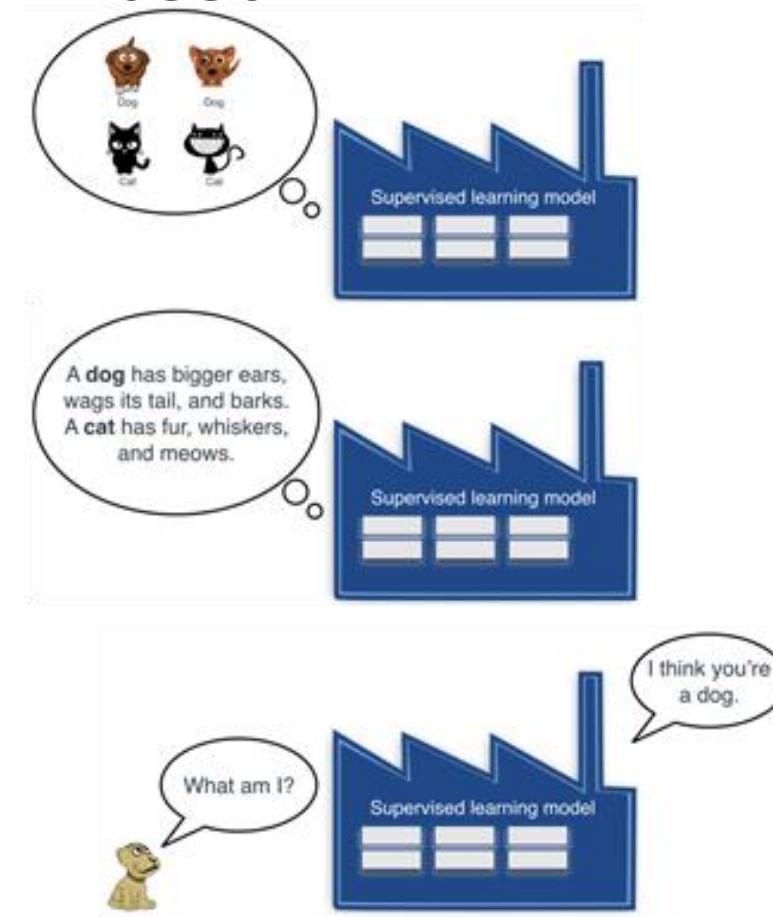
A supervised learning model predicts the label of a new data point.



What is supervised learning?

- The model makes a prediction about what the label of the image is, namely, is it a dog or a cat.

Supervised learning follows the Remember-Formulate-Predict framework from Lesson 1.



Remember

Formulate

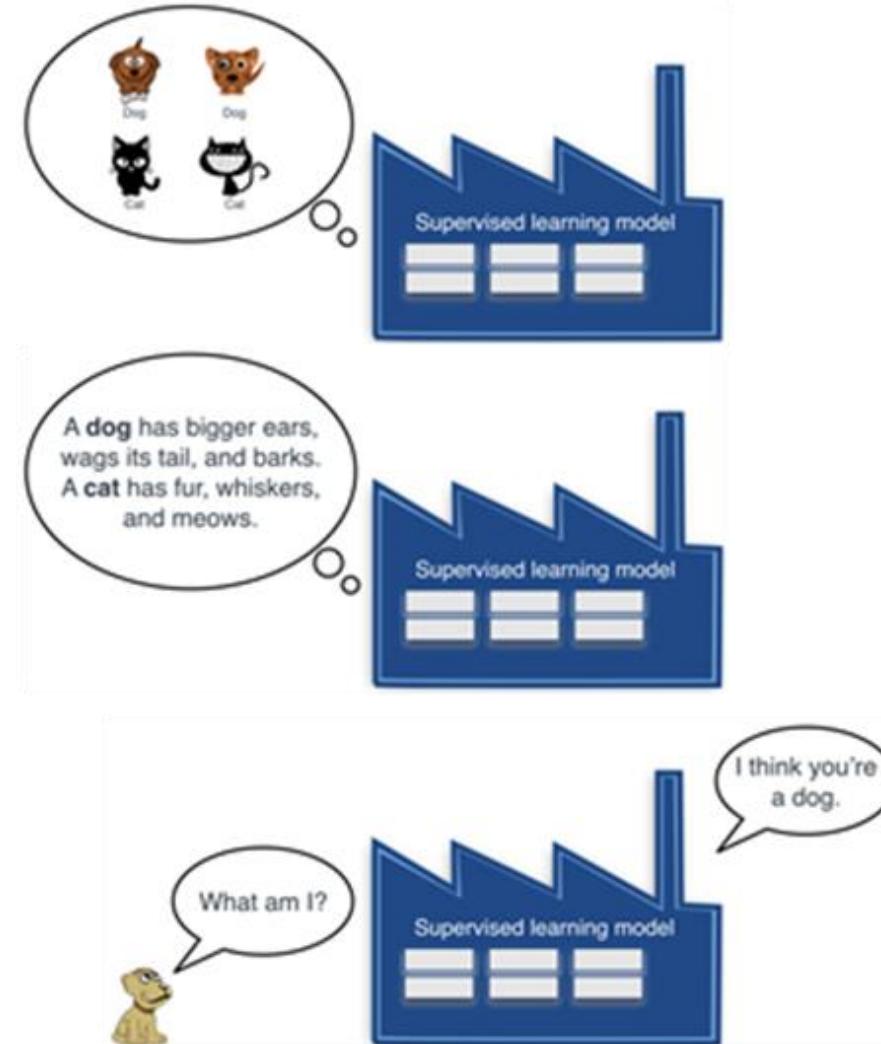
Predict

Regression models predict numbers

- The number is predicted from the features.
- In the housing example, the features can be the size of the house.
- The number of rooms, the distance to the closest school, the crime rate in the neighborhood, etc.

Classification models predict a state

- The example we saw in this figure is of classification, as it predicts the type of the pet, namely, 'cat' or 'dog'.



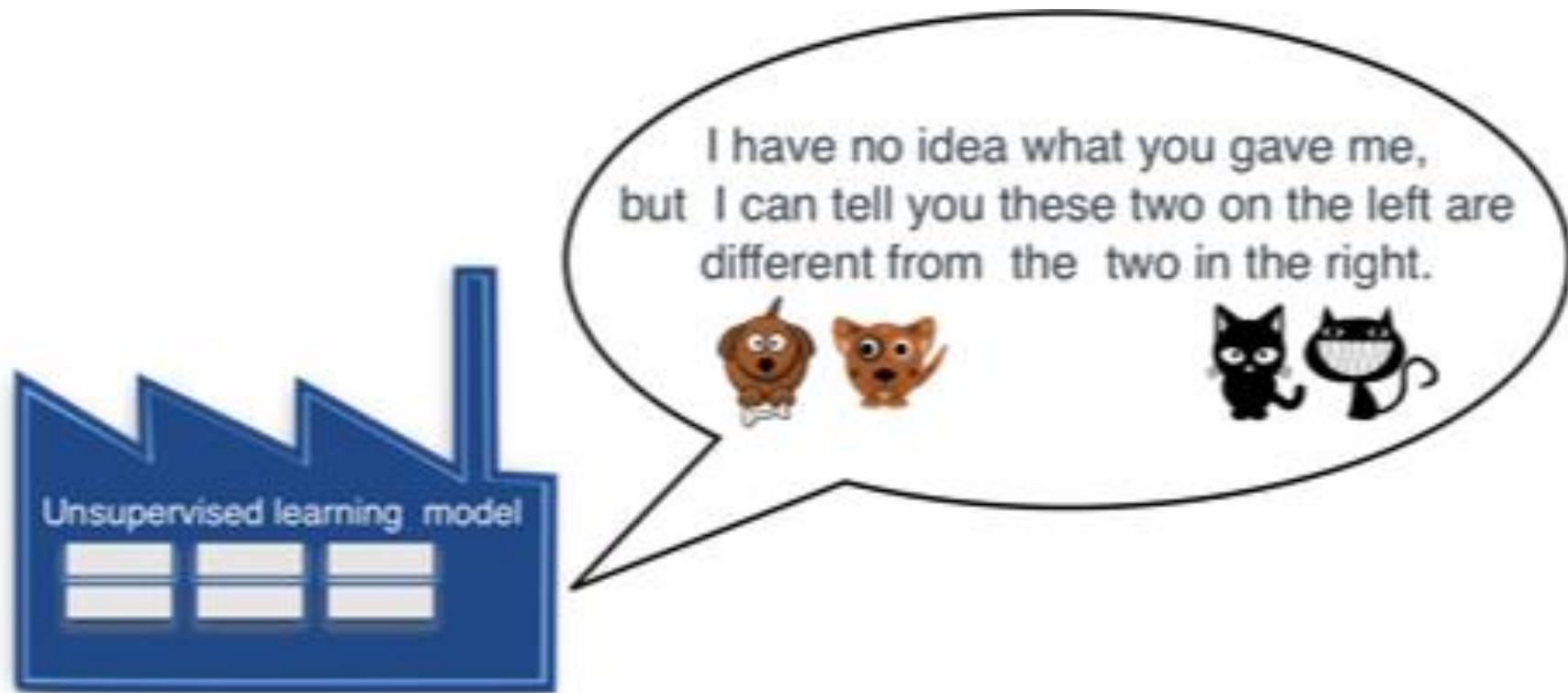
Remember

Formulate

Predict

What is unsupervised learning?

An unsupervised learning model can still extract information from data, for example, it can group similar elements together.



Clustering algorithms split a dataset into similar groups

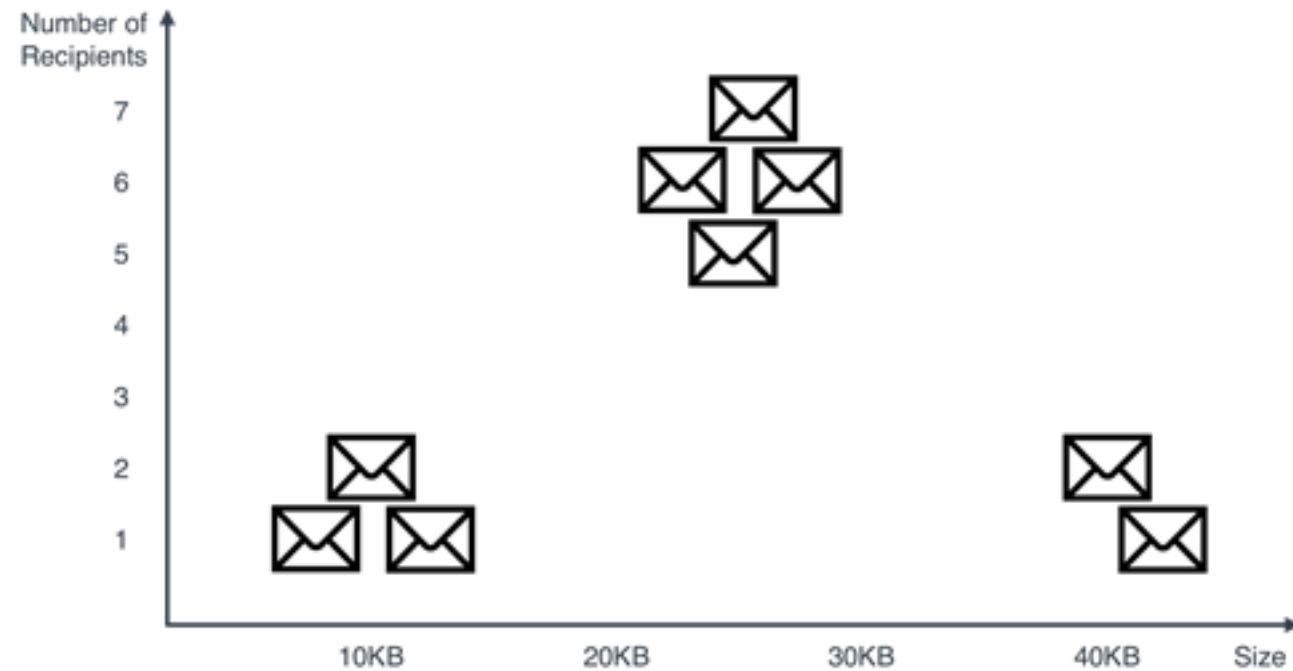
- let's look at examples we have a dataset with information about houses, but no prices.
- What could we do? Here is an idea: we could somehow group them into similar houses.
- We could group them by location, by price, by size, or by a combination of these factors, this is called clustering.

Clustering algorithms split a dataset into similar groups

Email	Size	Recipients
1	8	1
2	12	1
3	43	1
4	10	2
5	40	2
6	25	5
7	23	6
8	28	6
9	26	7

- To the naked eye, it looks like we could group them by size, where the emails in one group would have 1 or 2 recipients, and the emails in the other group would have 5 or more recipients.

Clustering algorithms split a dataset into similar groups

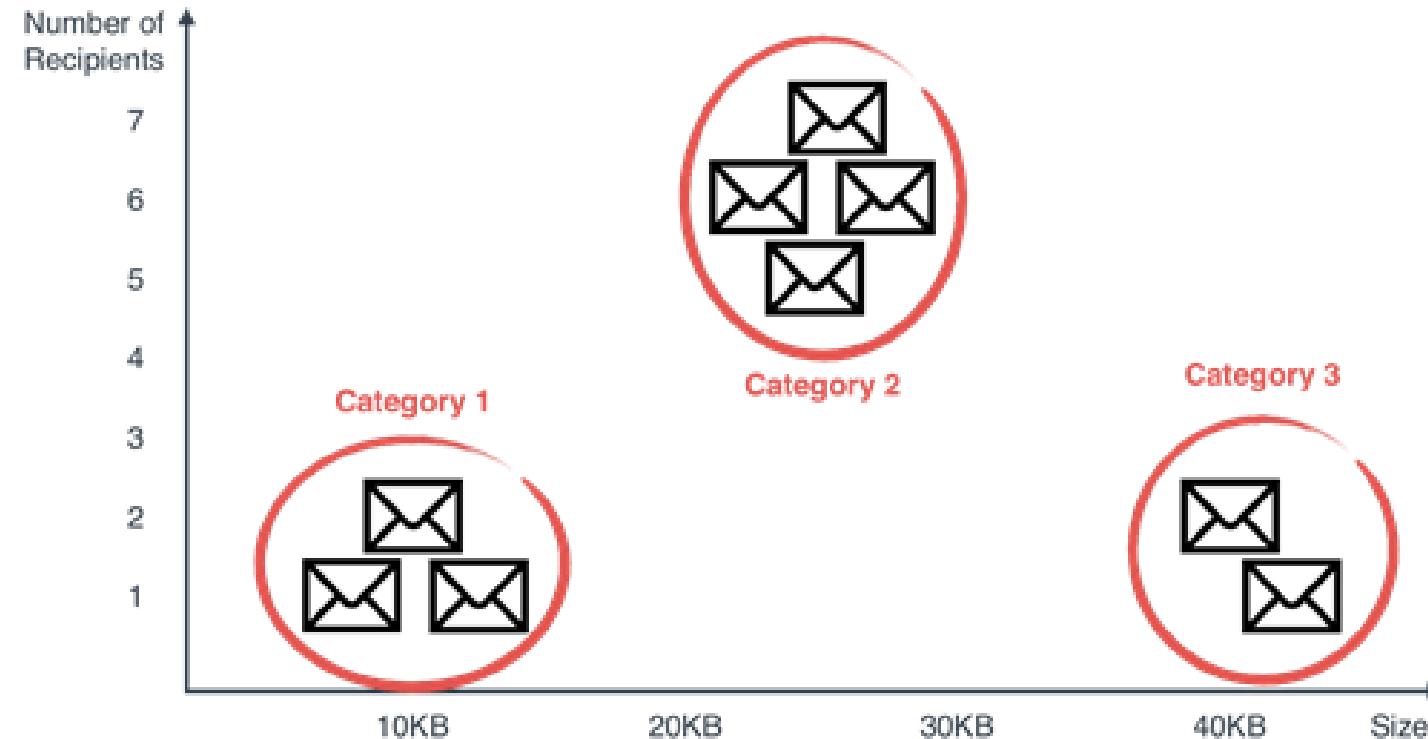


- We get the following plot.

A plot of the emails with size on the horizontal axis and number of recipients on the vertical axis. Eyeballing it, it is obvious that there are three distinct types of emails.

Clustering algorithms split a dataset into similar groups

- They are the ones we see in this Figure



Clustering the emails into three categories based on size and number of recipients.

Dimensionality reduction simplifies data without losing much information

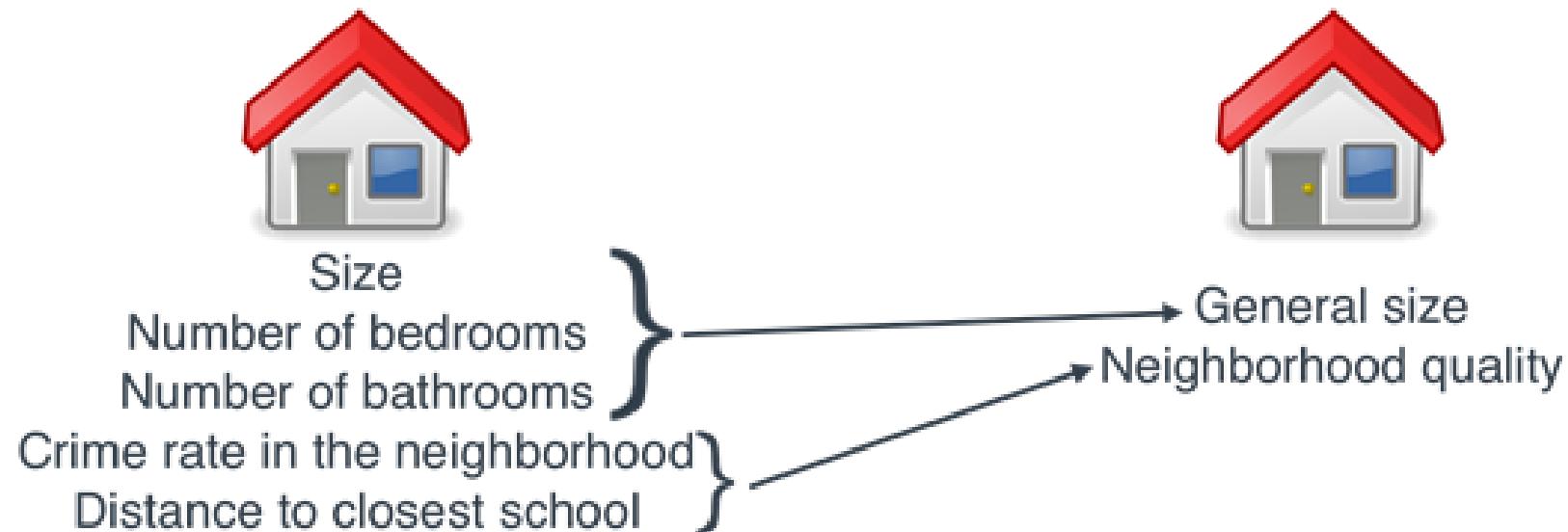
Let's say that we want to predict the price, and the features are the following:

- Size.
- Number of bedrooms.
- Number of bathrooms.
- Crime rate in the neighborhood.
- Distance to the nearest school.

Dimensionality reduction simplifies data without losing much information

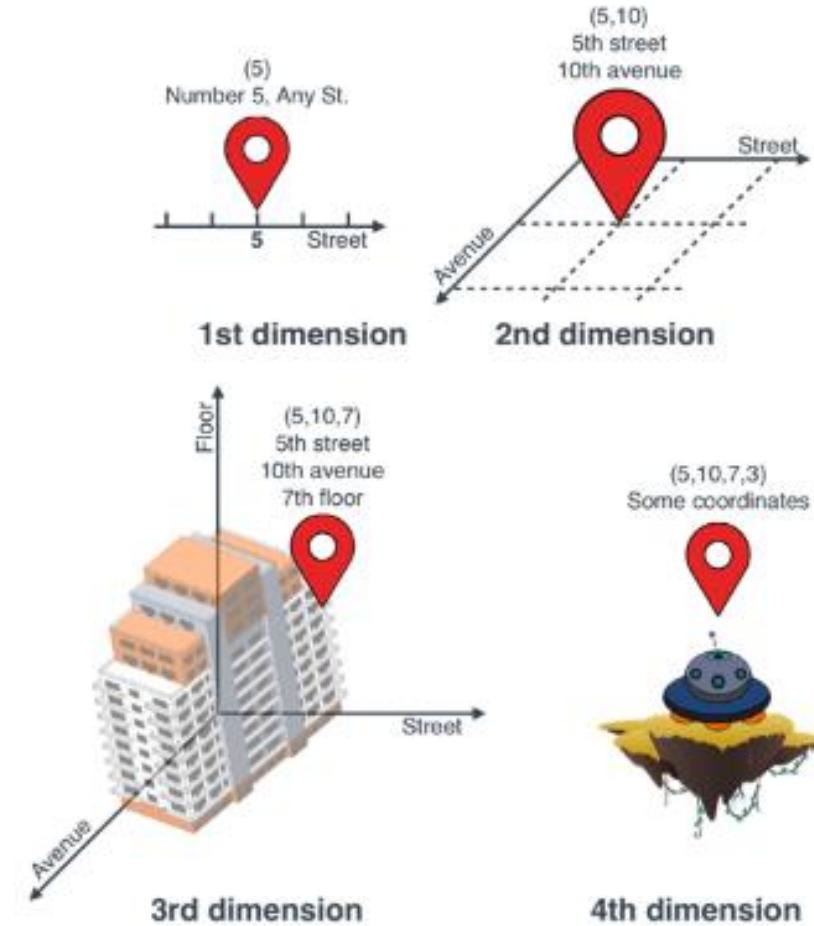
- Using dimensionality reduction to reduce the number of features in a housing dataset, without losing much information.

Dimensionality reduction

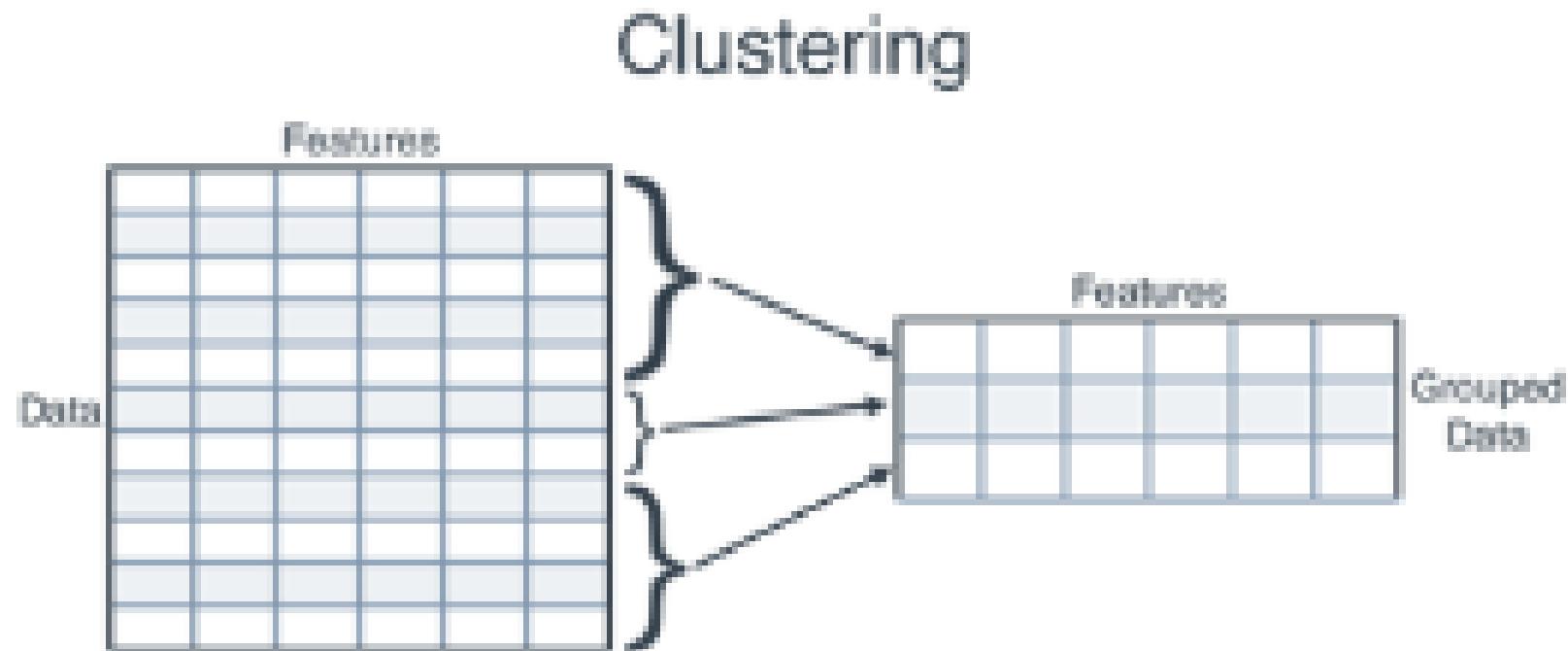


Dimensionality reduction simplifies data without losing much information

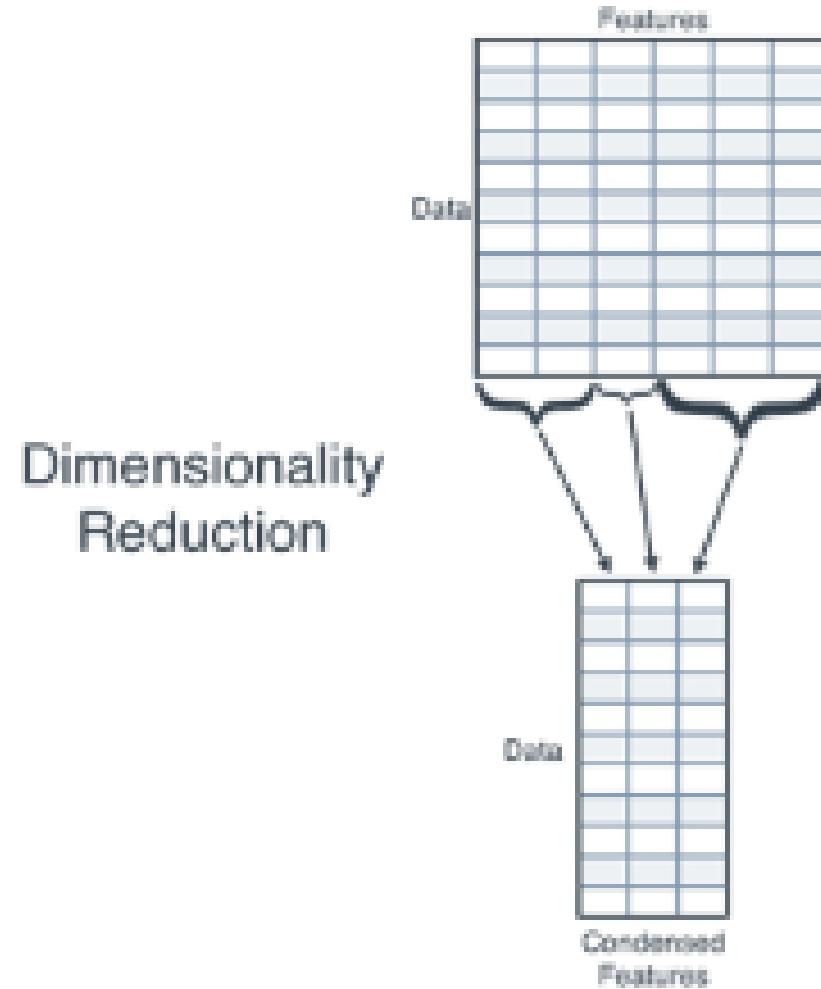
- When we went from five dimensions down to two, we reduced our 5-dimensional city into a 2-dimensional city, thus applying dimensionality reduction.



Matrix factorization and other types of unsupervised learning



Matrix factorization and other types of unsupervised learning

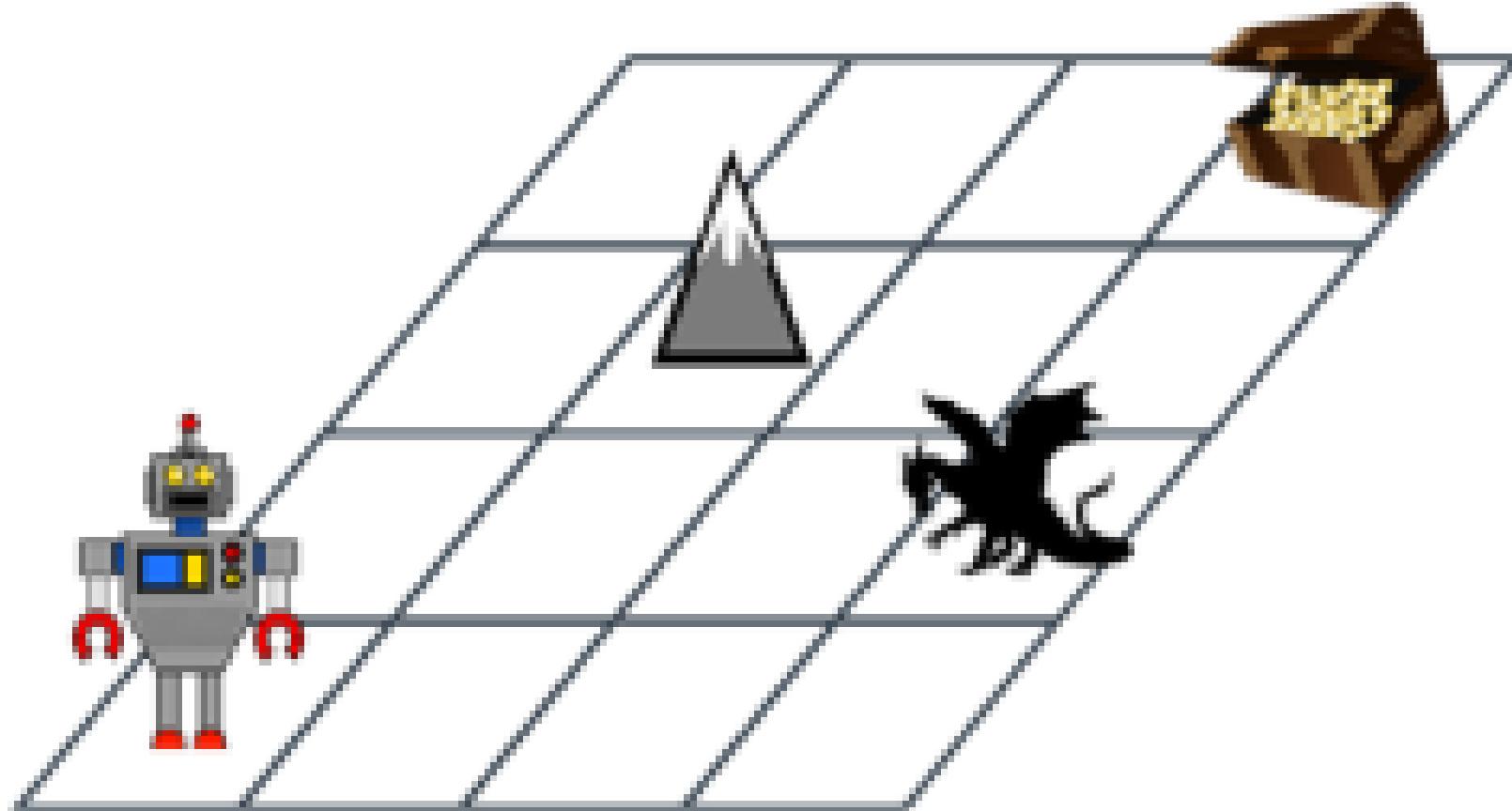


What is reinforcement learning?

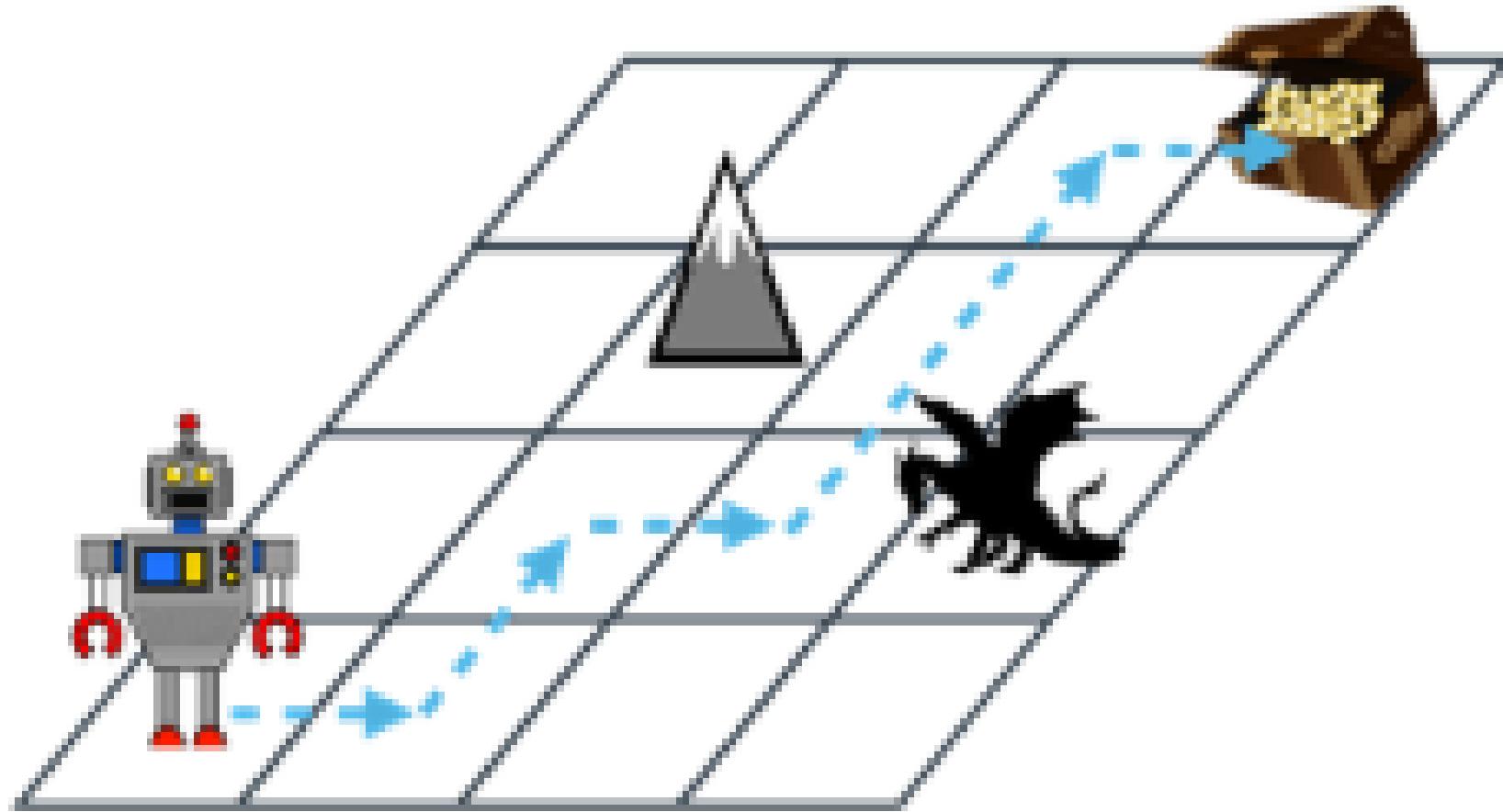
Example 1: Grid world

- We seen a grid world with a robot on the bottom left corner that is our agent.
- The goal is to get to the treasure chest in the top right of the grid.
- In the grid, we can also see a mountain, which means we cannot go through that square, since the robot cannot climb mountains.

What is reinforcement learning?

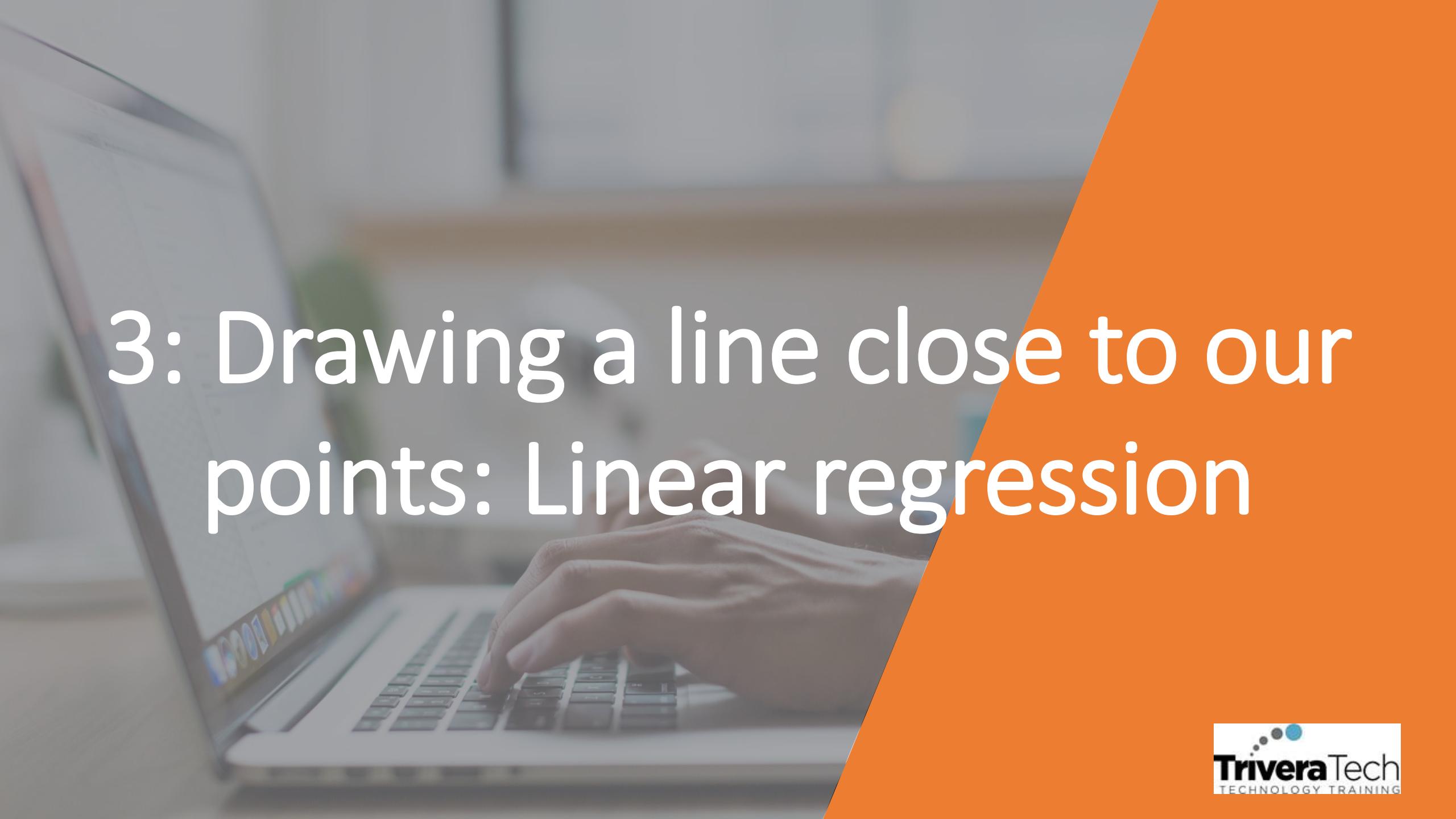


What is reinforcement learning?



Summary

- There are several types of machine learning, including supervised learning and unsupervised learning.
- Supervised learning is used on labelled data, and it is good for making predictions.
- Unsupervised learning is used on unlabelled data, and it is normally used as a preprocessing step.

A blurred background image of a person's hands typing on a laptop keyboard, suggesting a technical or data-related environment.

3: Drawing a line close to our points: Linear regression

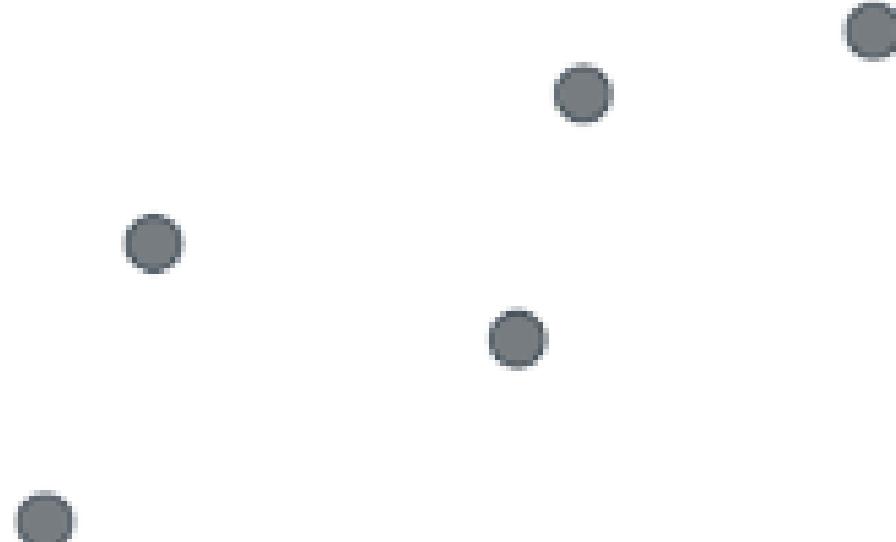
Drawing a line close to our points: Linear regression

This lesson covers

- What is linear regression?
- How to predict the price of a house based on known prices of other houses
- How to fit a line through a set of data points.
- How to code the linear regression algorithm in Python.

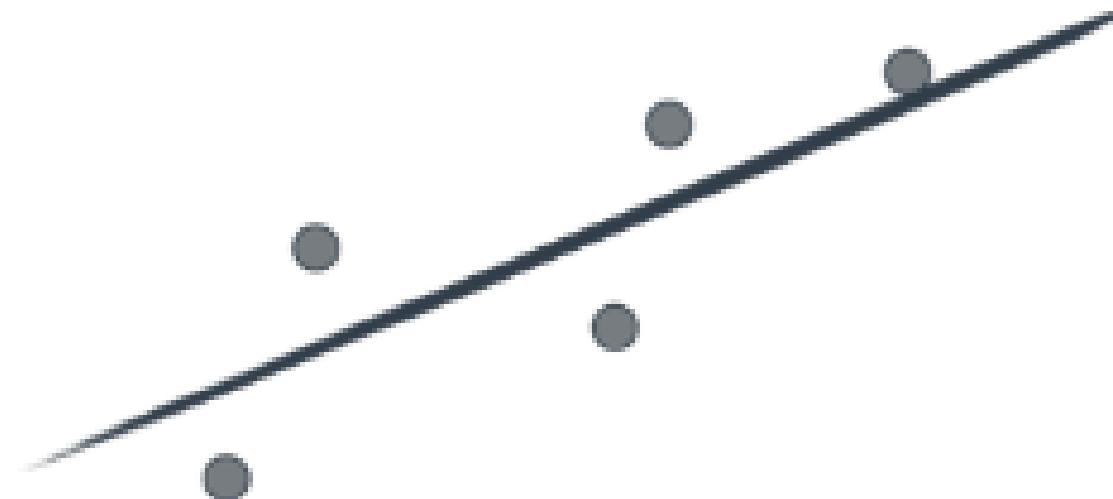
Drawing a line close to our points: Linear regression

- The mental picture of linear regression is simple.
- Let us say we have some points, that roughly look like they are forming a line, like in this figure



Drawing a line close to our points: Linear regression

- The goal of linear regression is to draw the line that passes as close as possible to these points.
- What line would you draw, that goes close to those points?
- The one I drew is in this figure.



The problem: We need to predict the price of a house

- Let's say we are a real estate agent, and we are in charge of selling a new house.
- We don't know the price, and we want to infer it by comparing it with other houses.

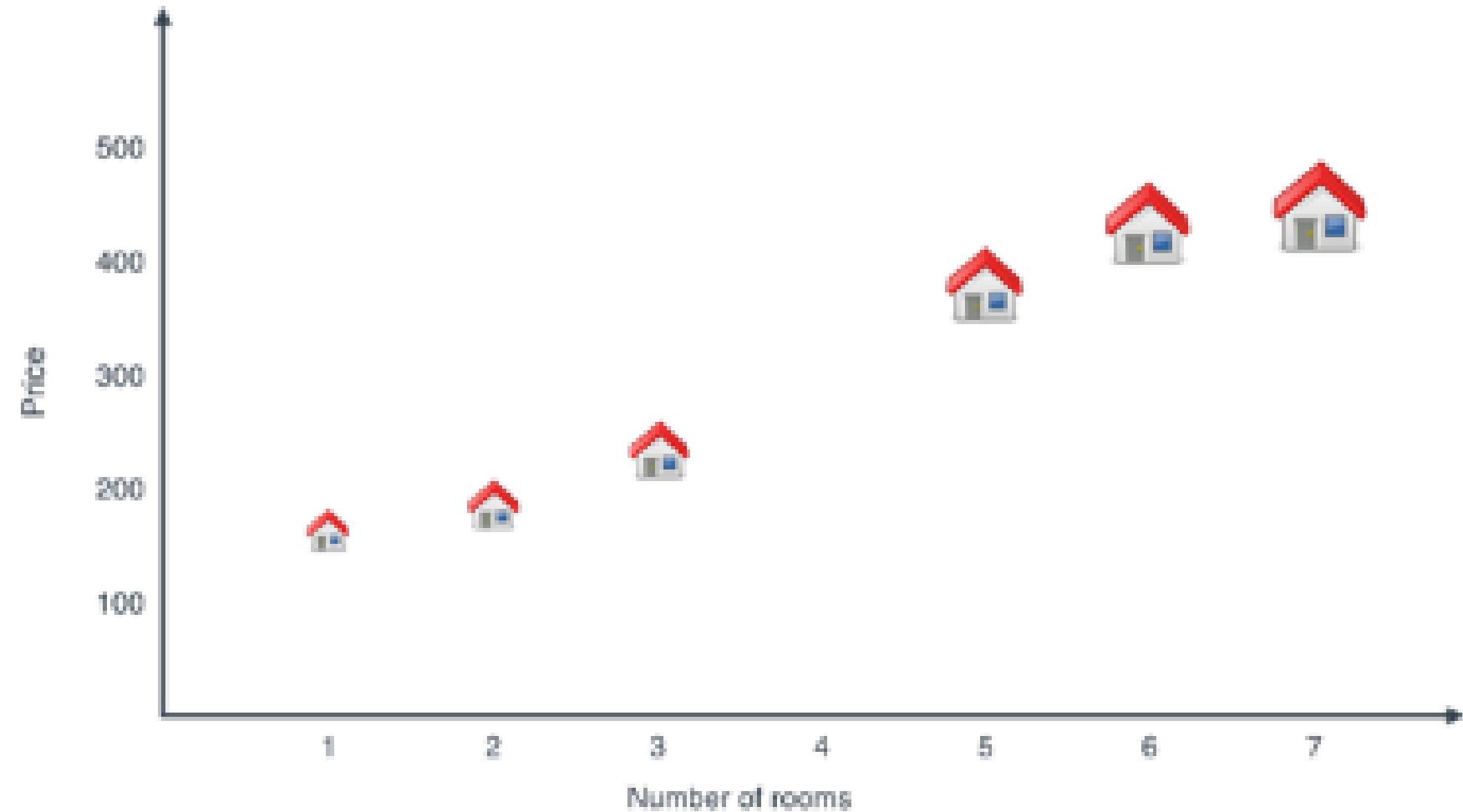
The solution: Building a regression model for housing prices

No of Rooms	Price
1	150
2	200
3	250
4	?
5	350
6	400
7	450

The remember step: looking at the prices of existing houses

No of Rooms	Price
1	155
2	197
3	244
4	?
5	356
6	407
7	448

The remember step: looking at the prices of existing houses

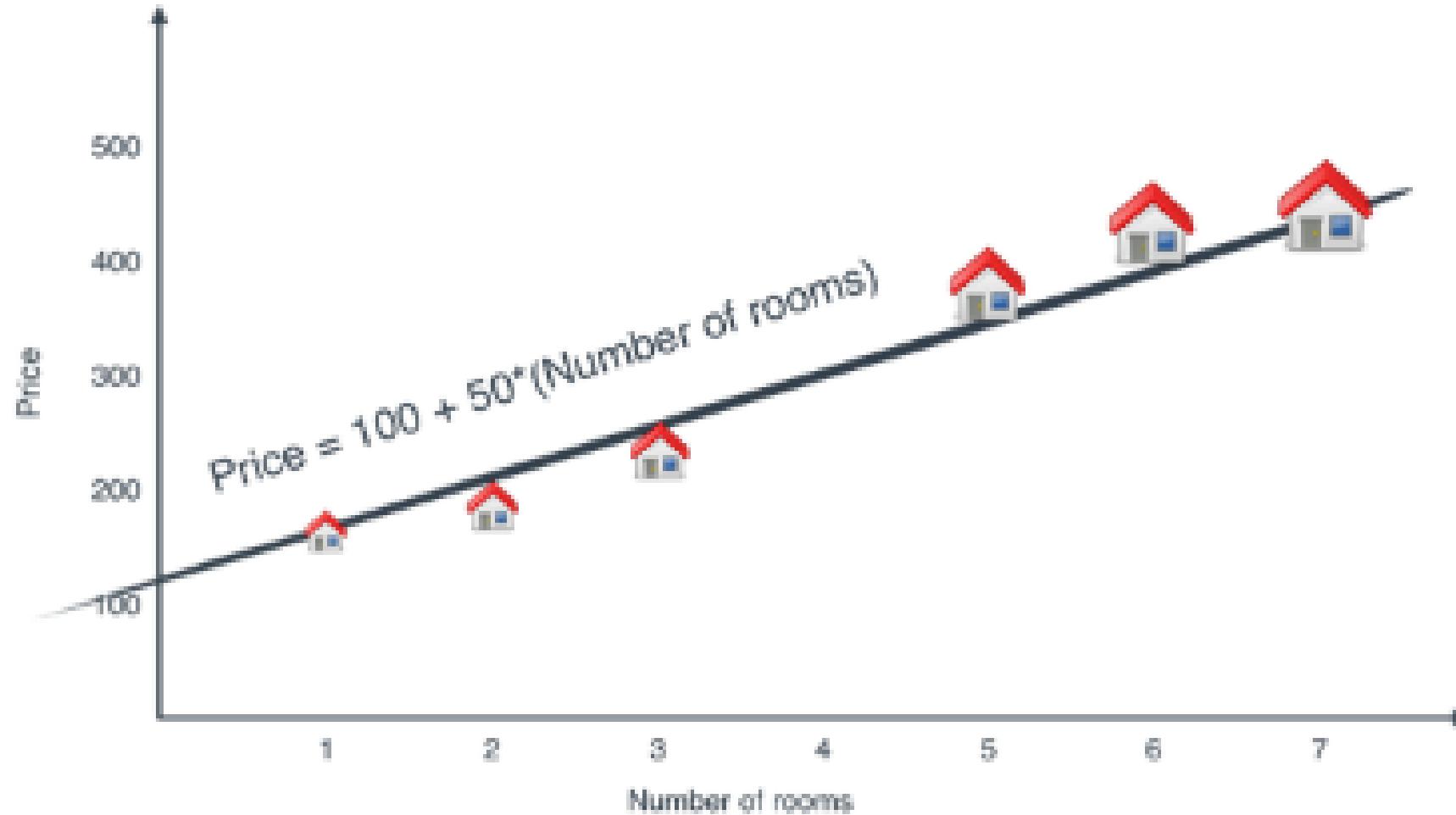


The formulate step: formulating a rule that estimates the price of the house

- The dataset in previous table is close enough to the one in Table 2, so we should feel safe to use the same formula for the price.
- The only difference is that now the prices are not exactly what the formula says, and we have a small error.
- The formula is as follows:

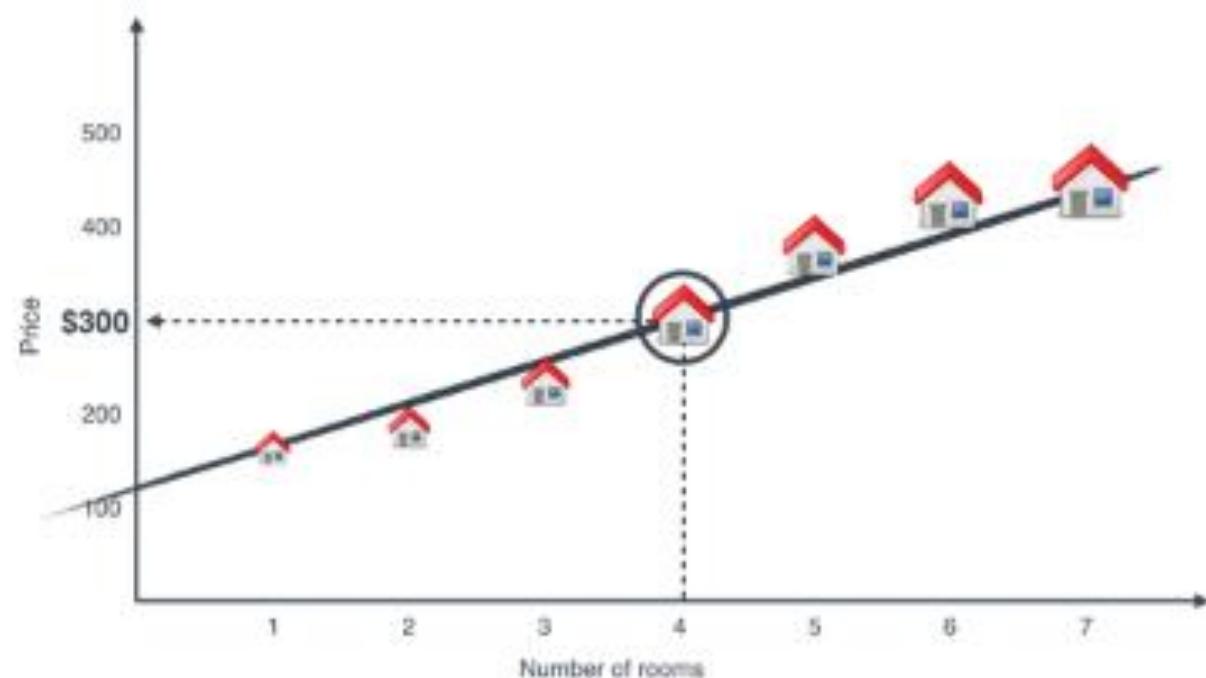
Price = 100 + 50*(Number of rooms) + (Small error)

The formulate step: formulating a rule that estimates the price of the house



The predict step: what do we do when a new house comes in the market?

- Therefore, our model predicted that the house costs \$300.
- This can also be seen graphically by using the line, as illustrated in this figure.



How to get the computer to draw this line: the linear regression algorithm

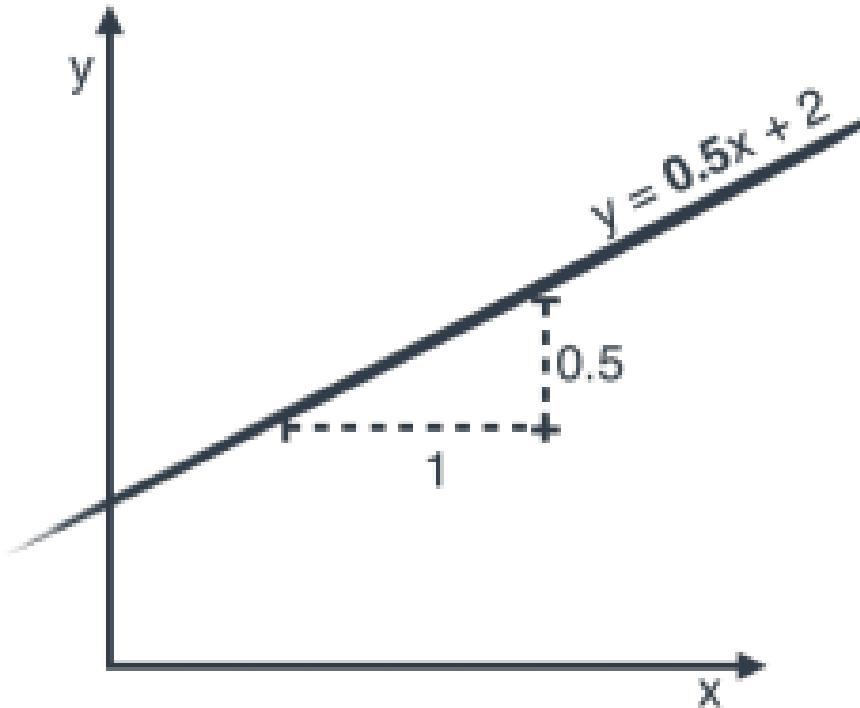
- Let's make a model that says that the price is **$\$51 + \$41 * (\text{Number of rooms})$** .
- This model assigns the house a price of **$\$51 + \$41 * 2 = \$133$** , which is closer to **$\$150$** .

How to get the computer to draw this line: the linear regression algorithm

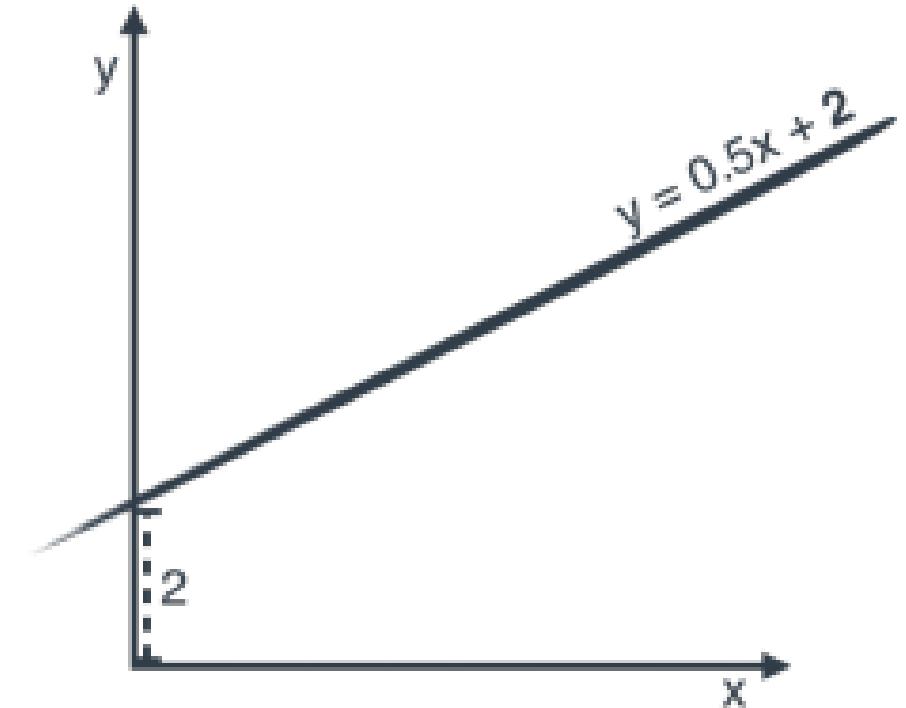
1. Start with random values for the base price and the price per room.
2. Repeat many times:
 - a) Pick a random house, and use it to increase or decrease the base price and price per room by a small amount, so the model will predict its price better.
3. Enjoy your model!

Crash course on slope and y-intercept

$$y = 0.5x + 2.$$



Slope = 0.5



y-intercept = 2

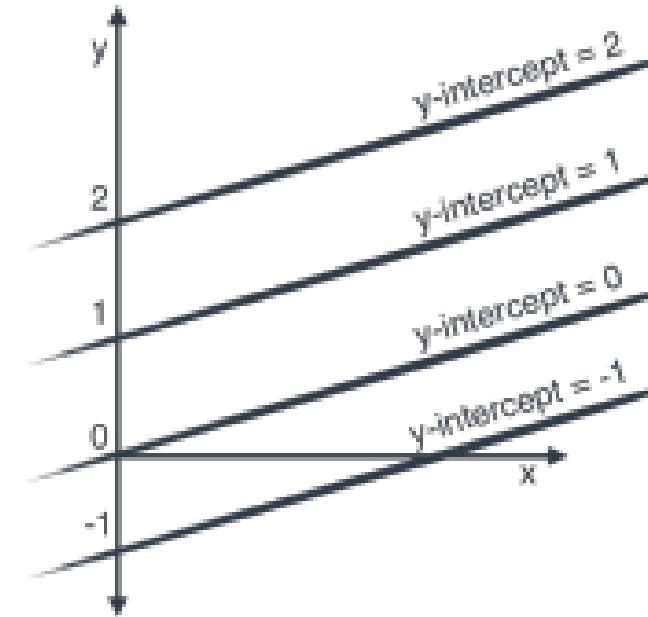
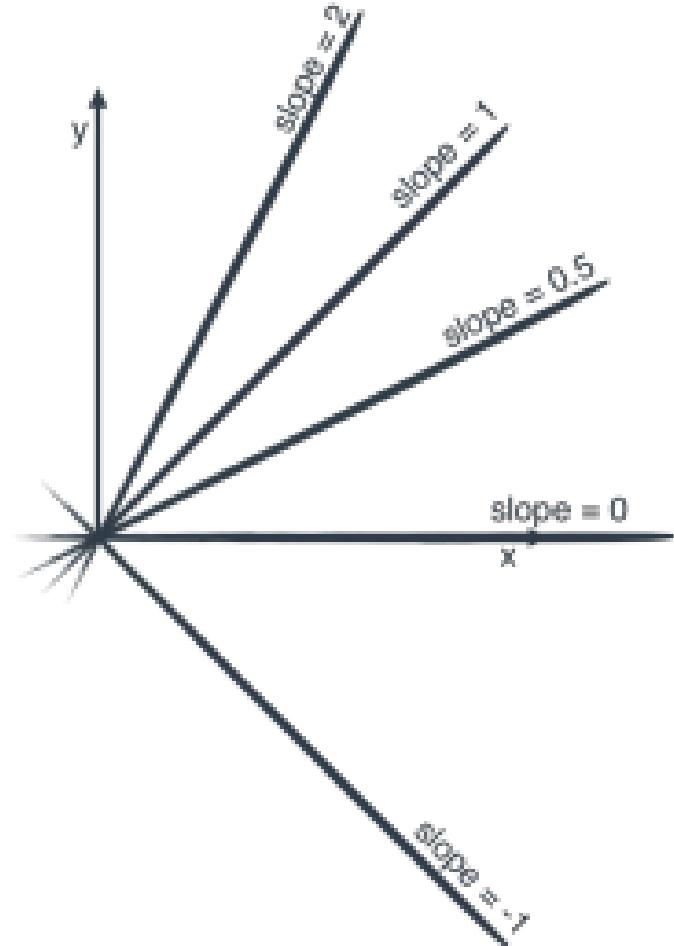
A simple trick to move a line closer to a set of points, one point at a time

Now we get to our problem, which is:

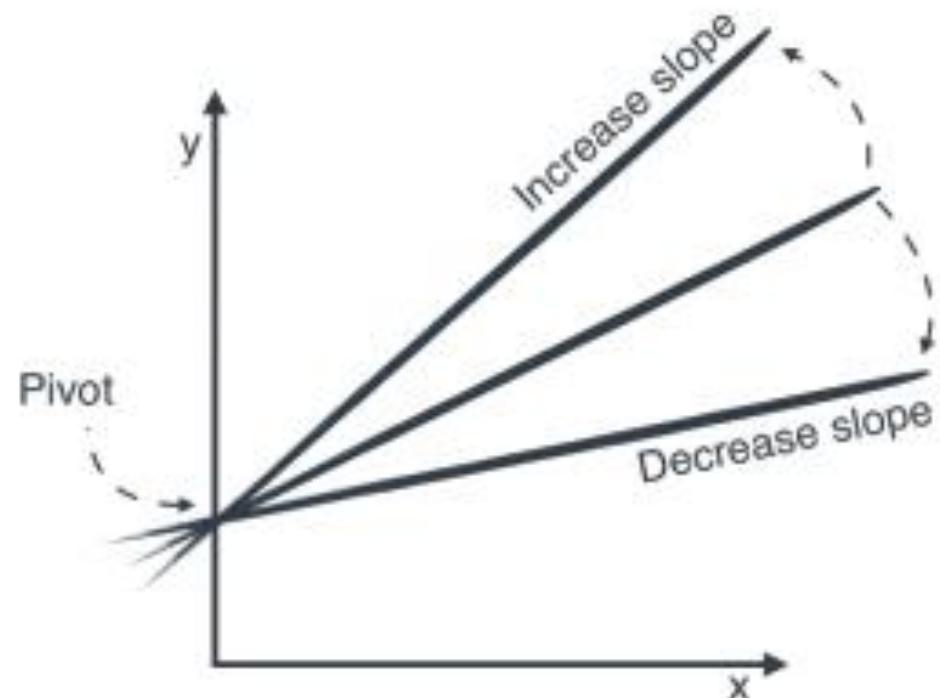
- we have a point, and a line, and we need to move the line closer to the point.



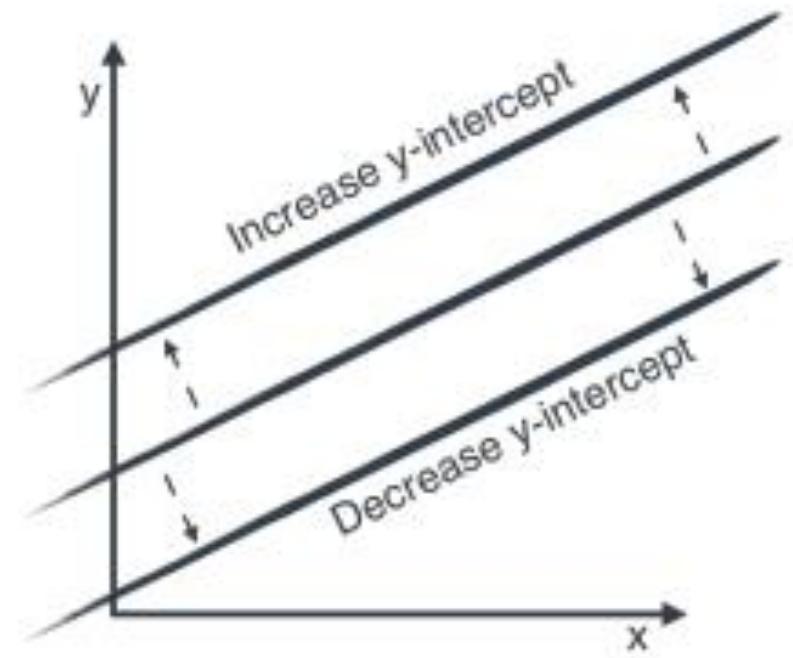
A simple trick to move a line closer to a set of points, one point at a time



A simple trick to move a line closer to a set of points, one point at a time

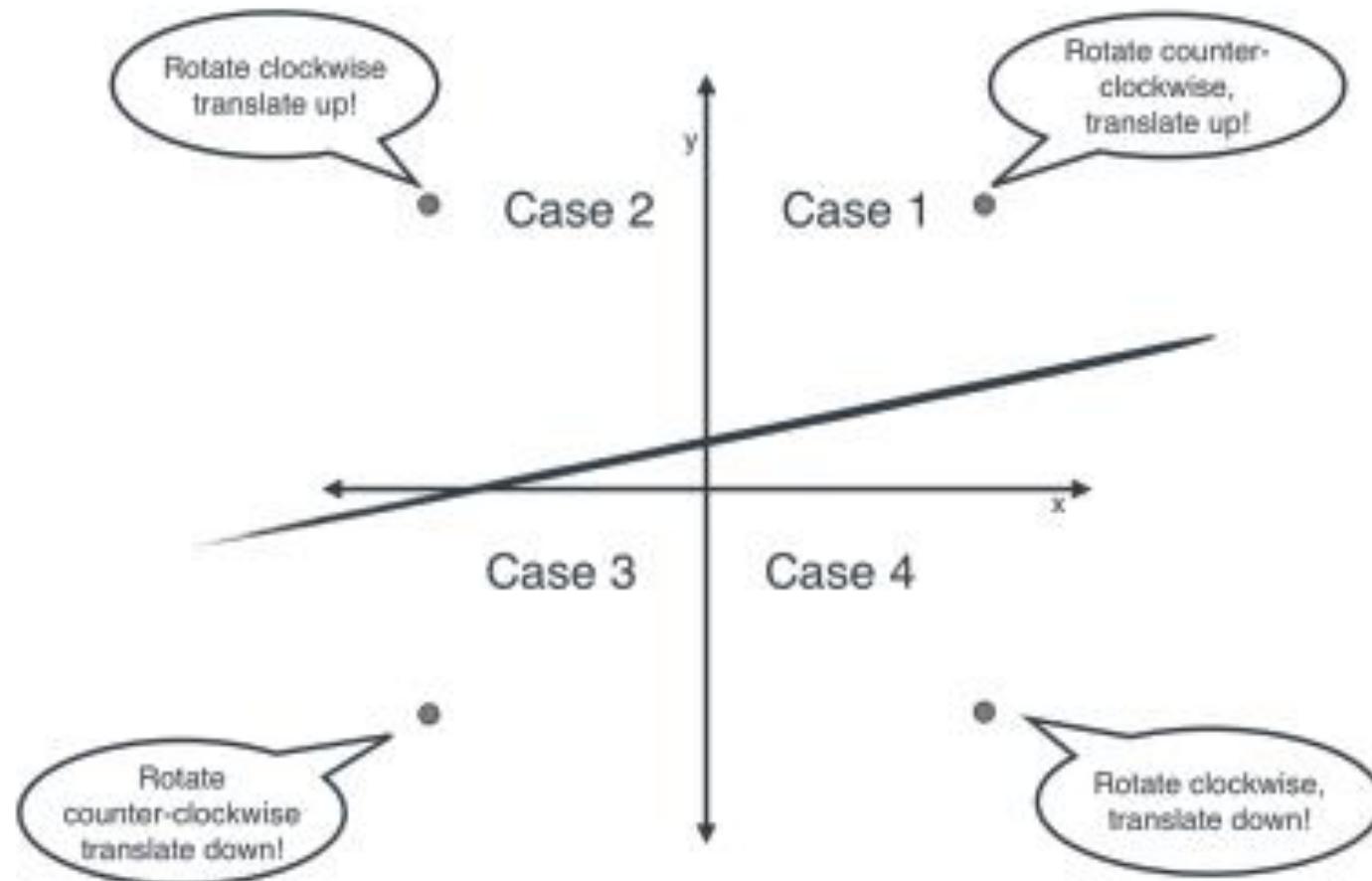


Rotate clockwise and
counterclockwise

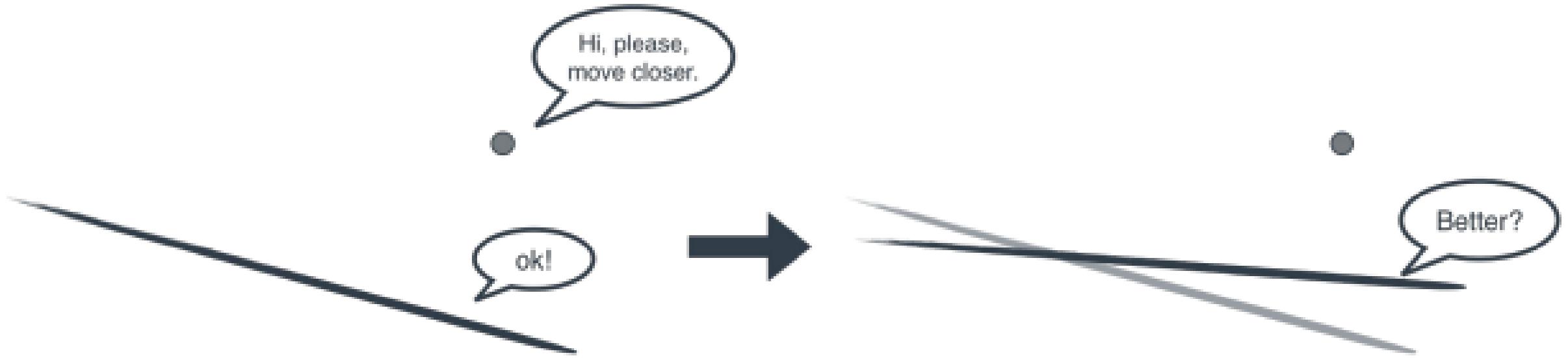


Translate up and down

A simple trick to move a line closer to a set of points, one point at a time



A simple trick to move a line closer to a set of points, one point at a time

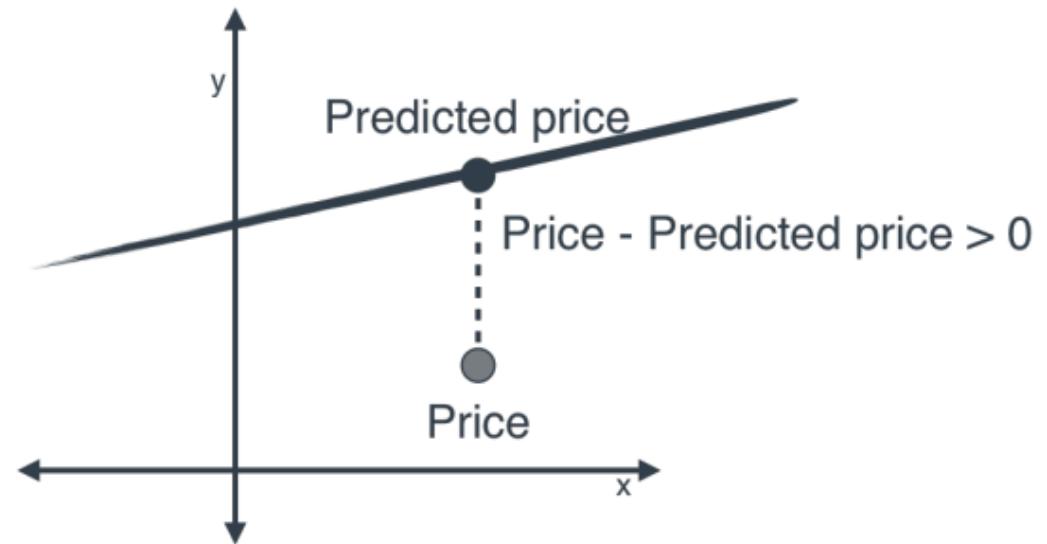
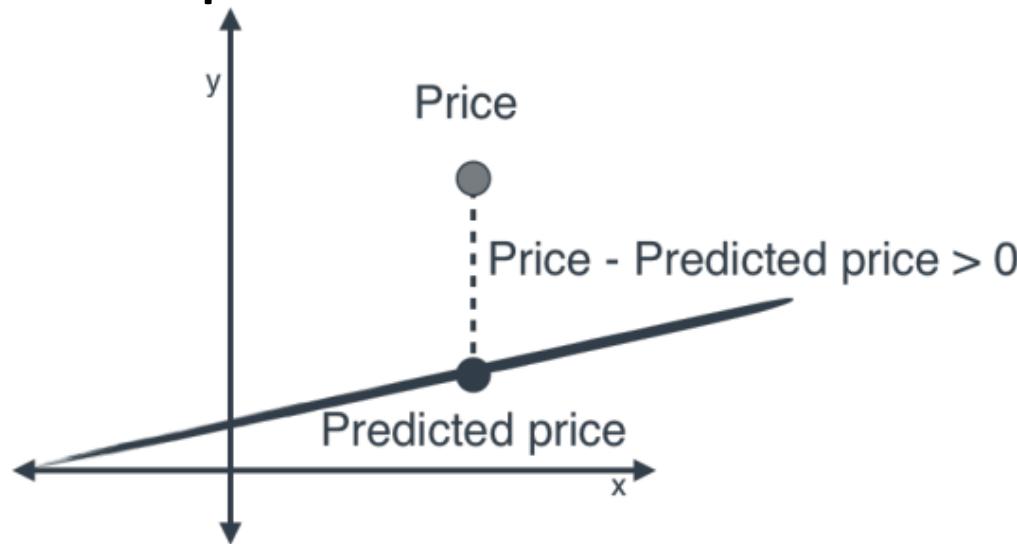


A simple trick to move a line closer to a set of points, one point at a time

```
def simple_trick(base_price, price_per_room, num_rooms, price, learning_rate): #A
    predicted_price = base_price + price_per_room*num_rooms #B
    if price > predicted_price and x > 0: #C
        price_per_room += learning_rate #D
        base_price += learning_rate #E
    if price > predicted_price and x < 0:
        price_per_room -= learning_rate
        base_price += learning_rate
    if price > predicted_price and x > 0:
        price_per_room += learning_rate
        base_price -= learning_rate
    if price > predicted_price and x < 0:
        price_per_room -= learning_rate
        base_price -= learning_rate
    return price_per_room, base_price
```

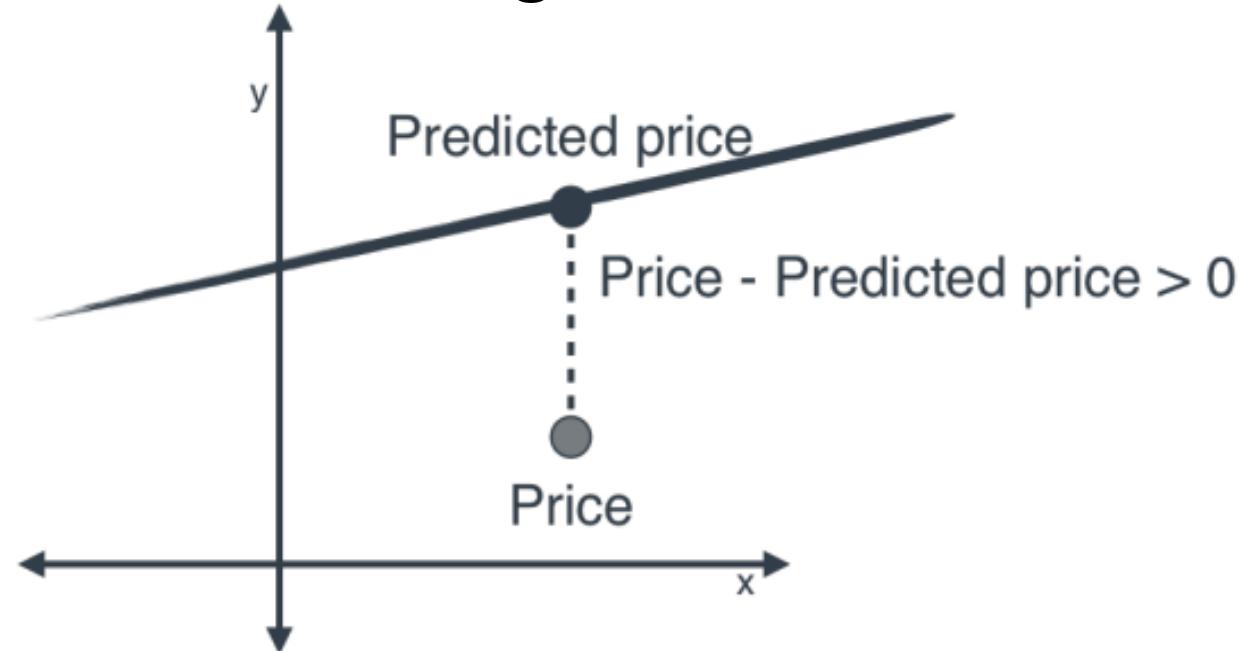
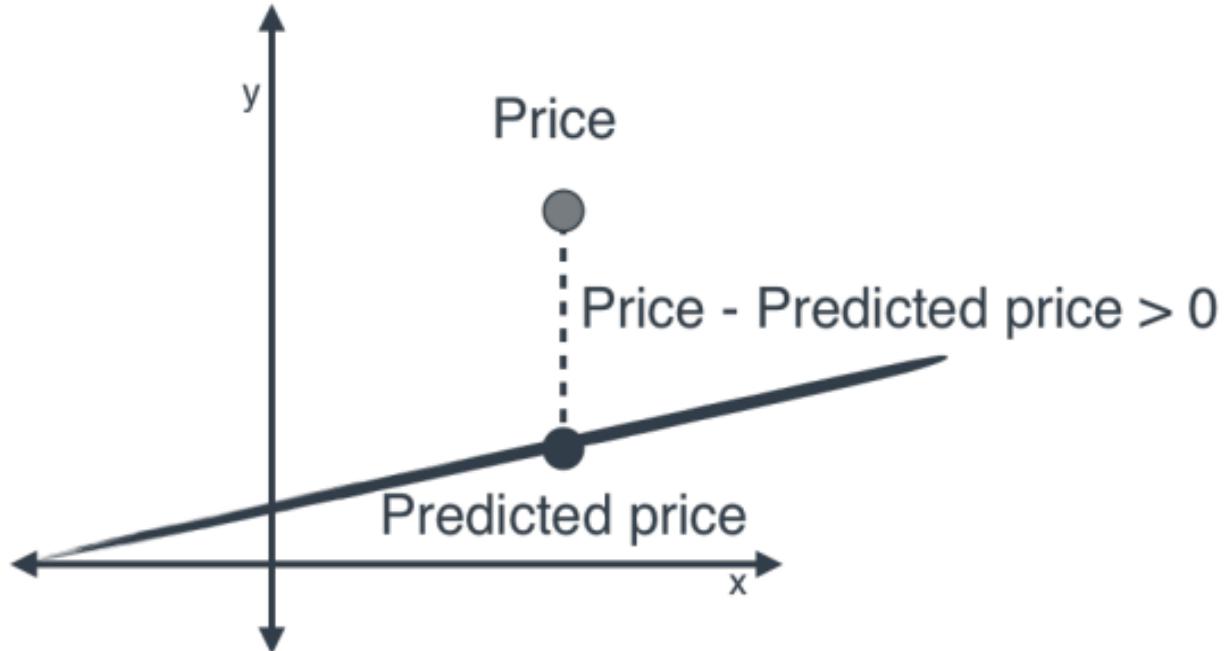
The square trick: A much more clever way of moving our line closer to one of the points

- **OBSERVATION 1** In the simple trick, when the point is above the line, we add a small amount to the y-intercept, and when the point is below the line, we subtract a small amount to the y-intercept.

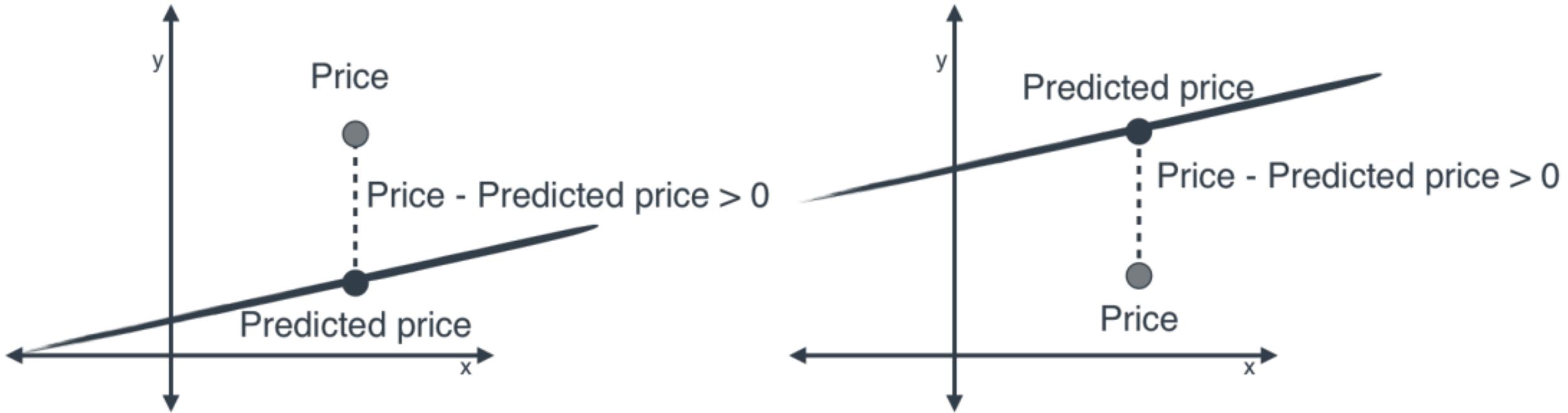


The square trick: A much more clever way of moving our line closer to one of the points

- **OBSERVATION 2** If a point is to the above the line, the difference of price minus predicted price is positive.
- If it is below the line, then this difference is negative.

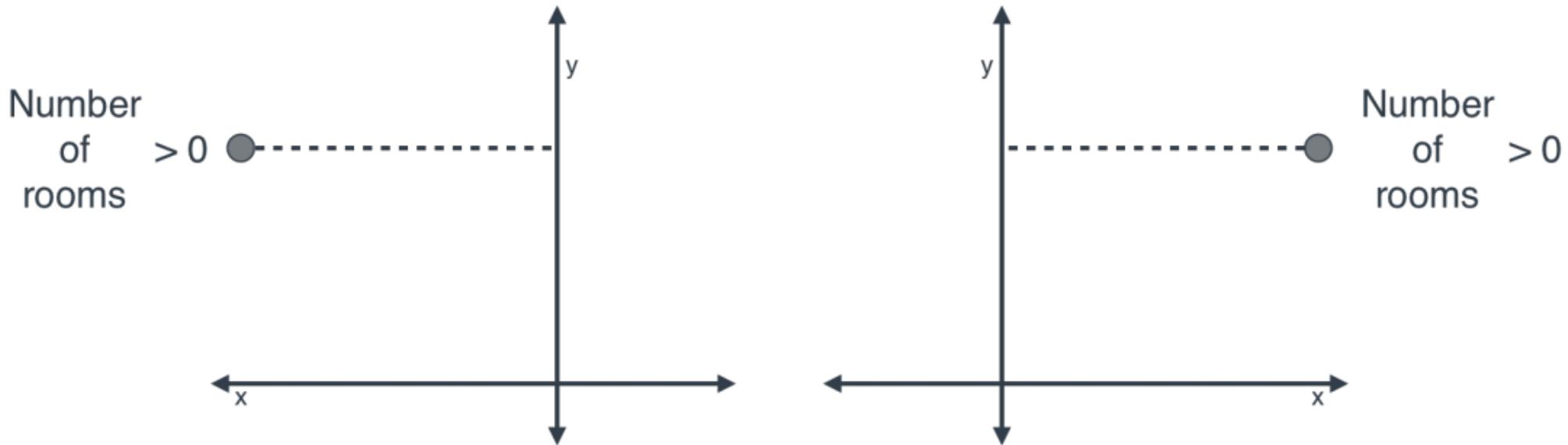


The square trick: A much more clever way of moving our line closer to one of the points



The square trick: A much more clever way of moving our line closer to one of the points

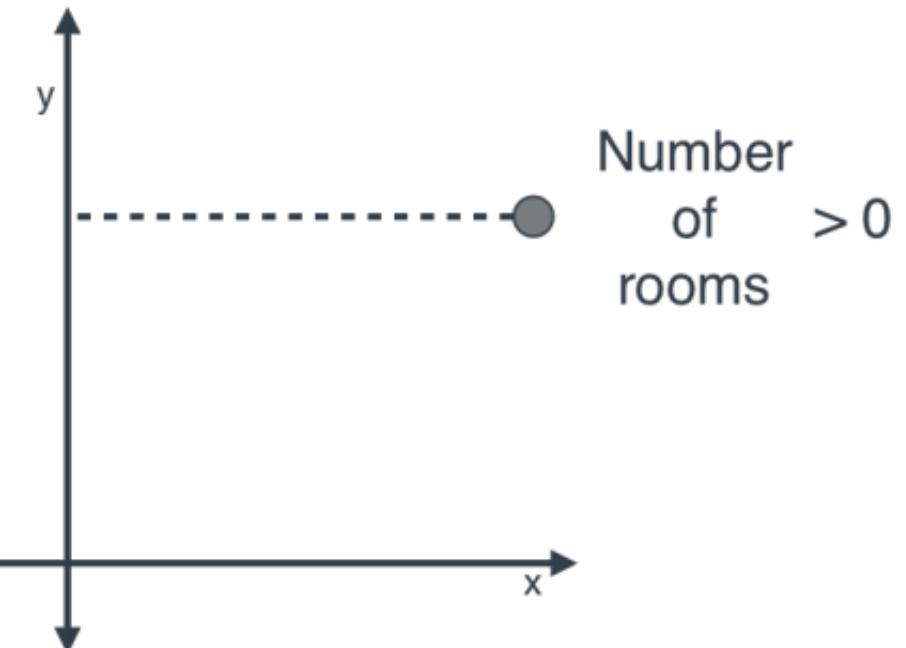
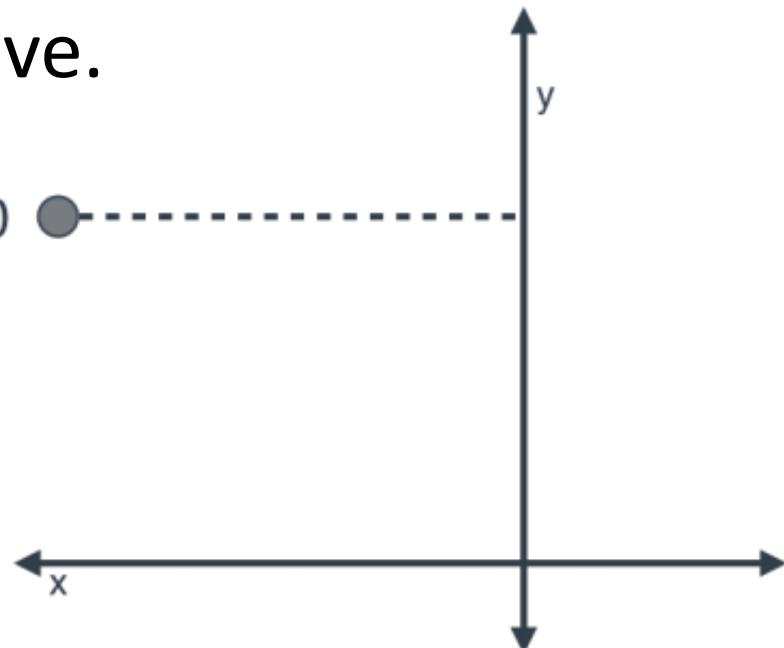
- **OBSERVATION 3** In the simple trick, when the point is either above the line and to the right of the y-axis, or below the line and to the left of the y-axis, we add a small amount to the slope.
- Otherwise, we subtract a small amount to the slope



The square trick: A much more clever way of moving our line closer to one of the points

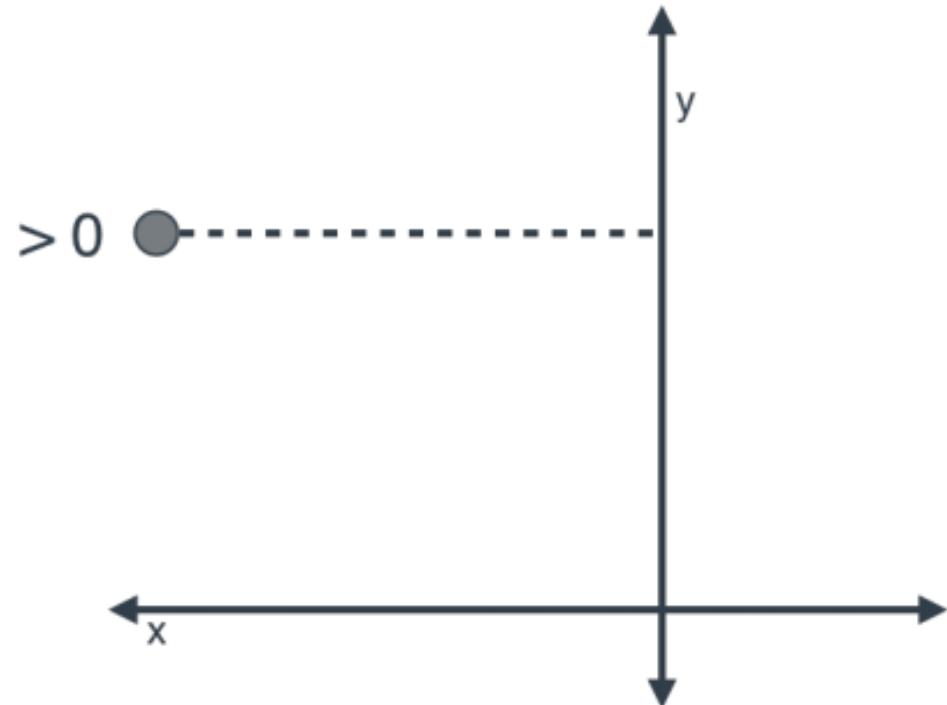
- **OBSERVATION 4** If the point is to the right of the y-axis, then the number of rooms is positive.
- If the point is to the left of the y-axis, then this quantity is negative.

Number
of
rooms

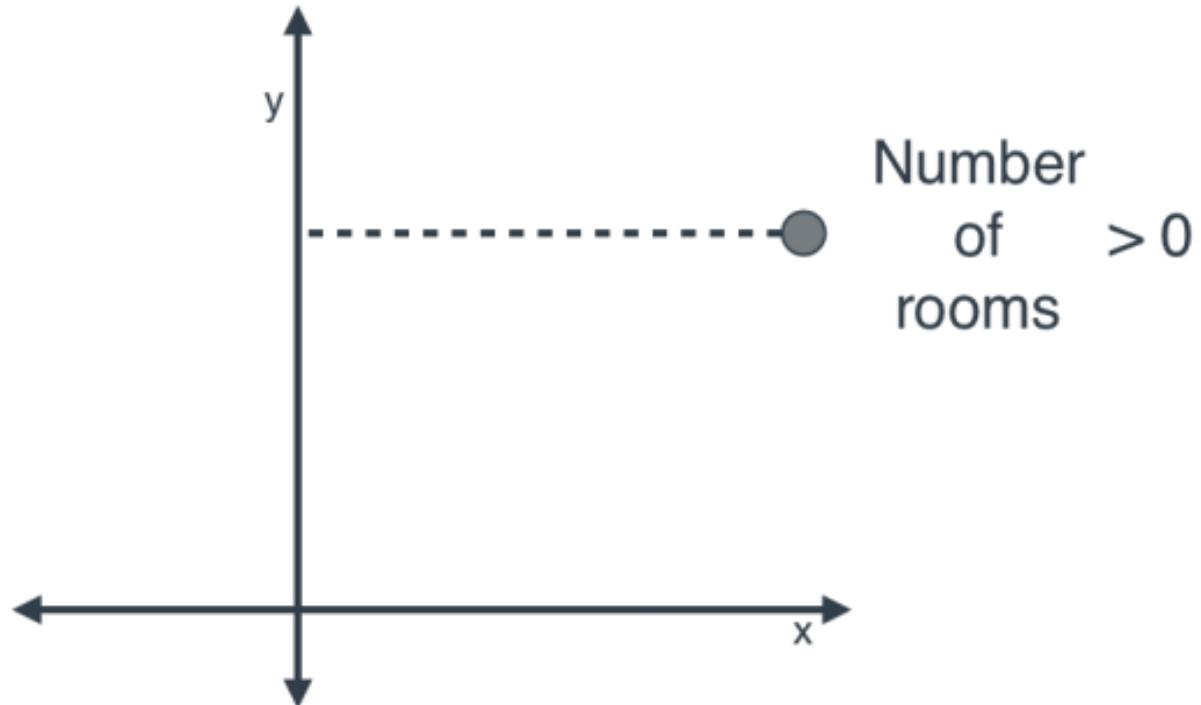


The square trick: A much more clever way of moving our line closer to one of the points

Number
of
rooms



Number
of
rooms



The square trick: A much more clever way of moving our line closer to one of the points

- here is the code:

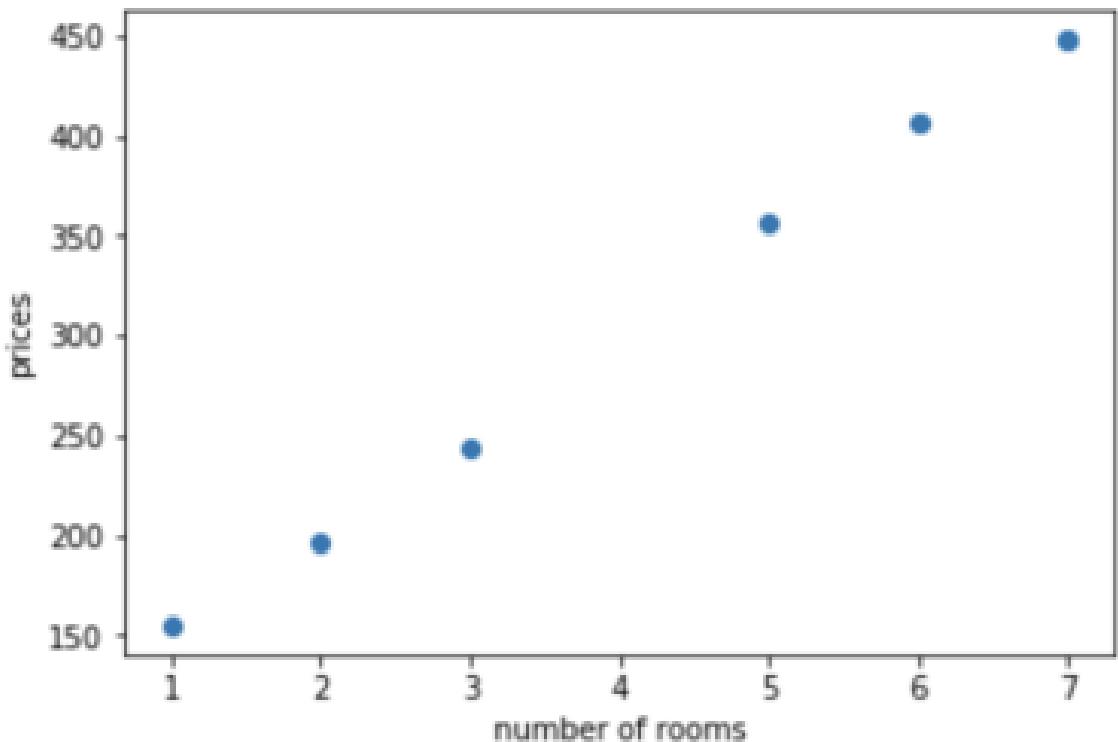
```
def square_trick(price_per_room, base_price, num_rooms, price, learning_rate):  
    predicted_price = base_price + price_per_room*num_rooms #A  
    base_price += learning_rate*(price-predicted_price) #B  
    price_per_room += learning_rate*num_rooms*(price-predicted_price) #C  
    return price_per_room, base_price
```

The linear regression algorithm: Repeating the square trick many times

```
import random #A
def linear_regression(features, labels, learning_rate=0.01, epochs = 1000):
    price_per_room = random.random()
    base_price = random.random() #B
    for i in range(epochs): #C
        i = random.randint(0, len(features)-1) #D
        num_rooms = features[i]
        price = labels[i]
        price_per_room, base_price = square_trick(base_price, #E
                                                    price_per_room,
                                                    num_rooms,
                                                    price,
                                                    learning_rate=learning_rate)
    return price_per_room, base_price
```

Plotting dots and lines

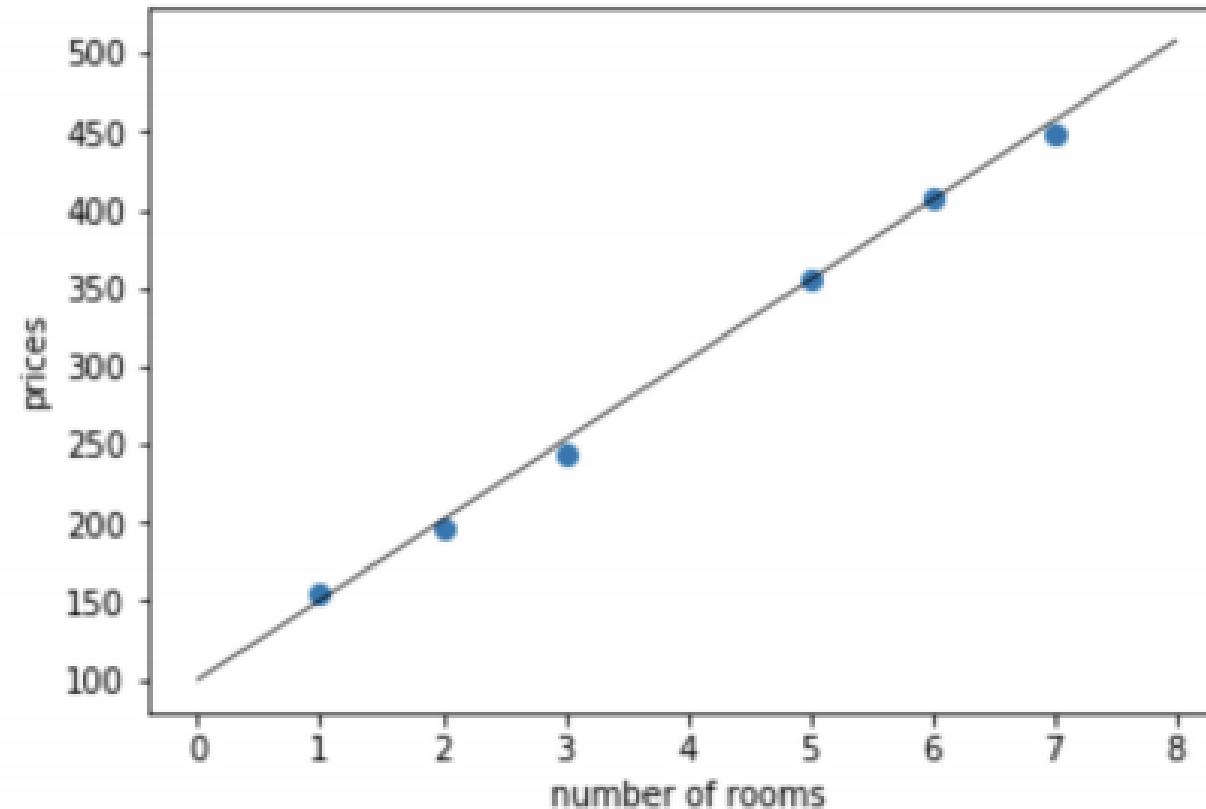
- The first plot we'll show here, which is the plot of the points in our small housing dataset.



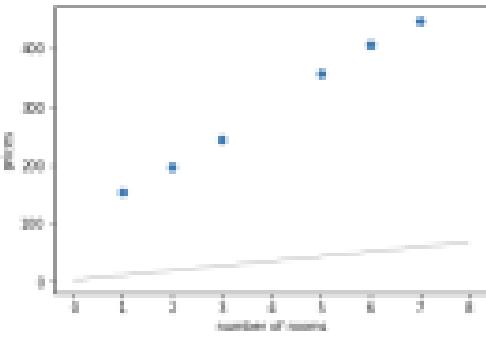
Using the linear regression algorithm in our dataset

Price per room: \$51.07296115119787

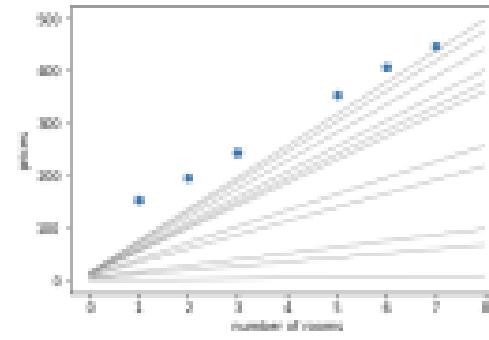
Base price: \$99.47510567502614



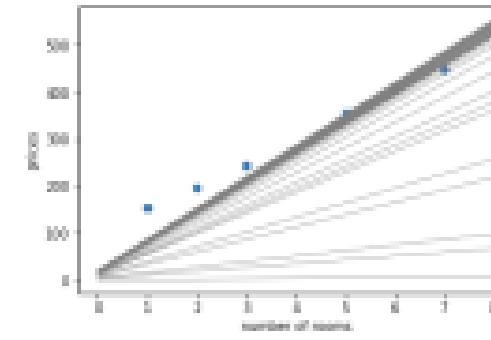
Using the linear regression algorithm in our dataset



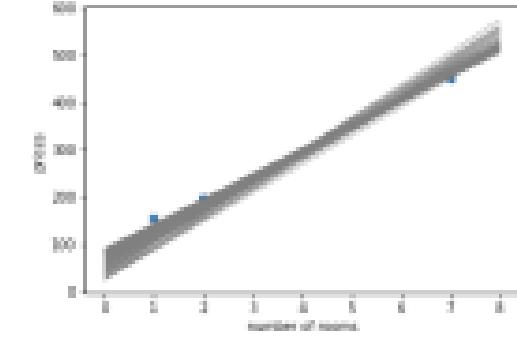
Starting point



Epochs 1-10



Epochs 1-50



Epochs 51-10,000

Video and music recommendations

- One of the ways used to generate video and music recommendations is to predict the amount of time a user will watch a video or listen to a song.

Product recommendations

- Stores and e-commerce websites also use linear regression to predict their sales.
- One way to do this is to predict how much will a customer spend in the store.

Healthcare

Regression has numerous applications in healthcare. Depending on what problem we want to solve, predicting the right label is the key. Here are a couple of examples:

- Predicting the lifespan of a patient, based on their current health conditions.
- Predicting the length of a hospital stay, based on current symptoms.

Summary

- Regression is a very important part of machine learning.
- It consists of training an algorithm with labelled data, and using it to make predictions on future (unlabeled) data.
- Labelled data is data that comes with labels, which in the regression case, are numbers.
- For example, the numbers could be prices of houses.

4: Optimizing the training process: Underfitting, overfitting, testing, and regularization

Optimizing the training process

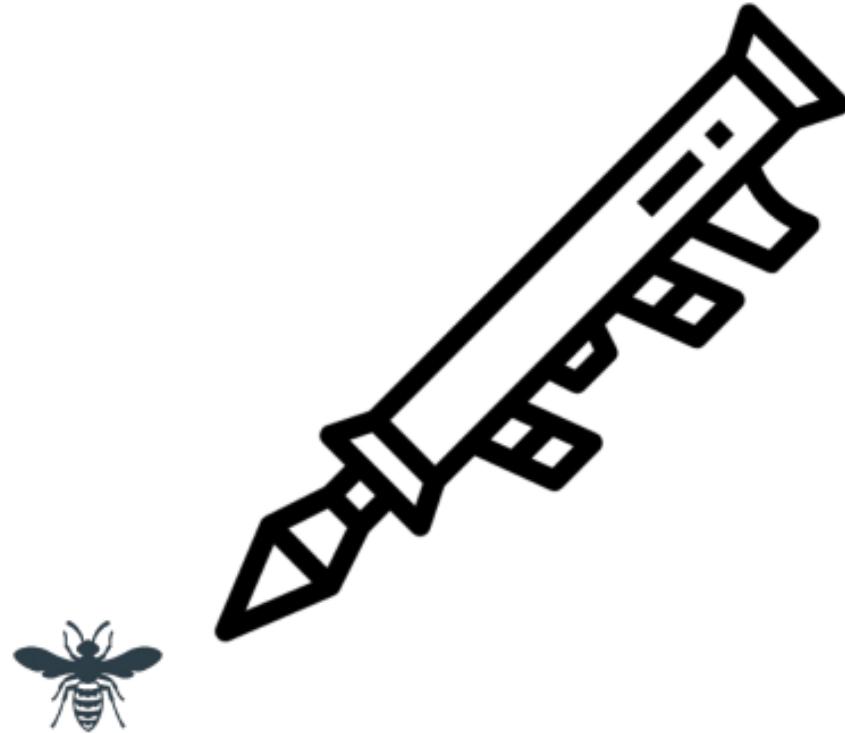
This mini-lesson covers

- What is underfitting and overfitting?
- Underfitting and overfitting in regression models.
- A solution for avoiding overfitting: Testing the model.
- Using a model complexity graph to take decisions on our model.
- Another solution to avoid overfitting: Regularization.
- Calculating the complexity of the model using the L1 and L2 norms.
- Picking the best model in terms of performance and complexity.

Optimizing the training process

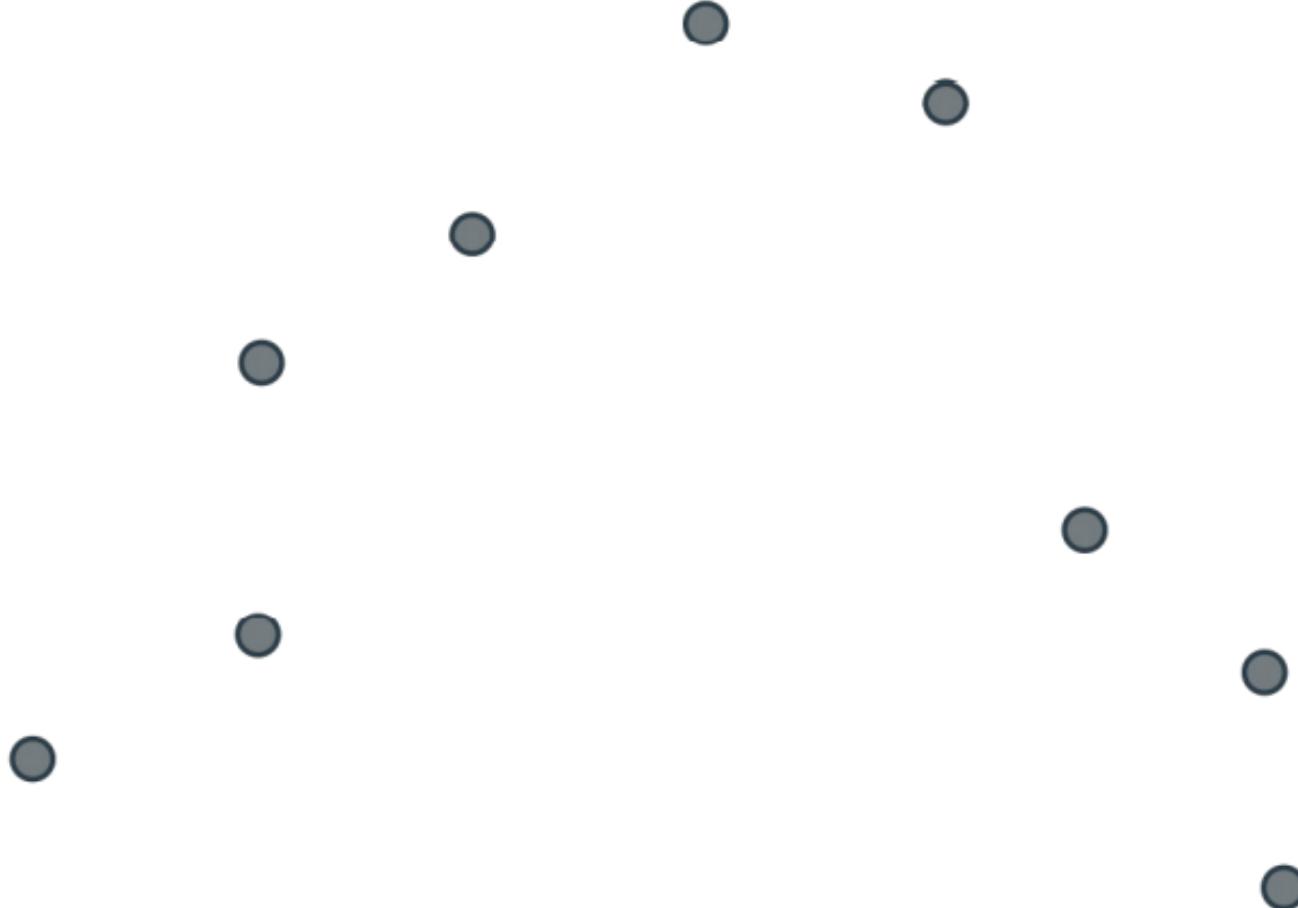


Underfitting

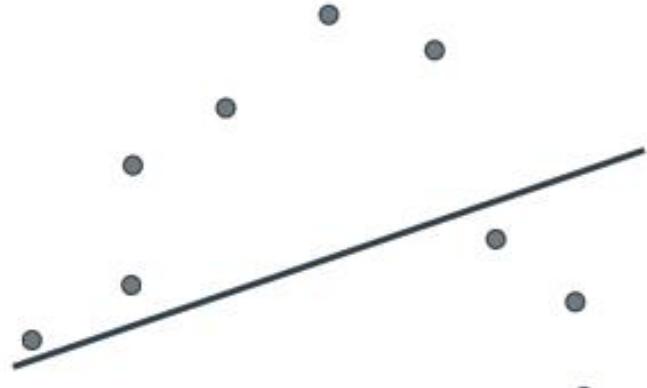


Overfitting

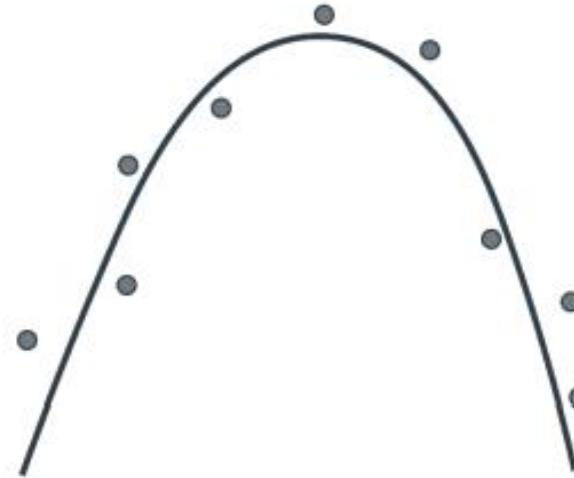
Optimizing the training process



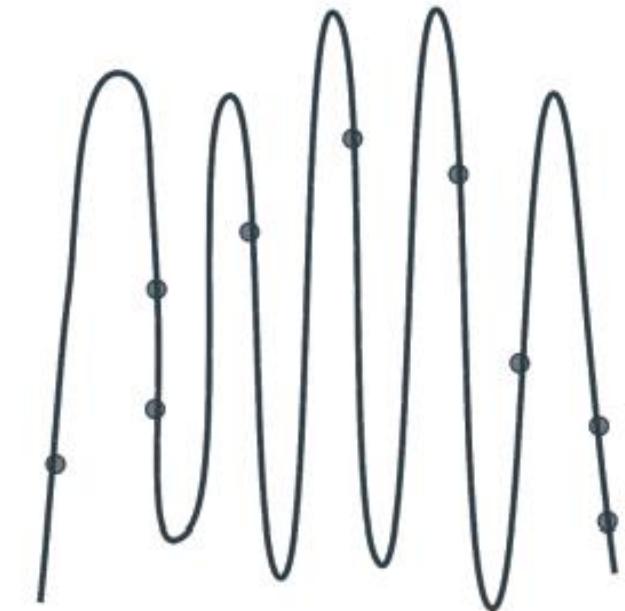
Optimizing the training process



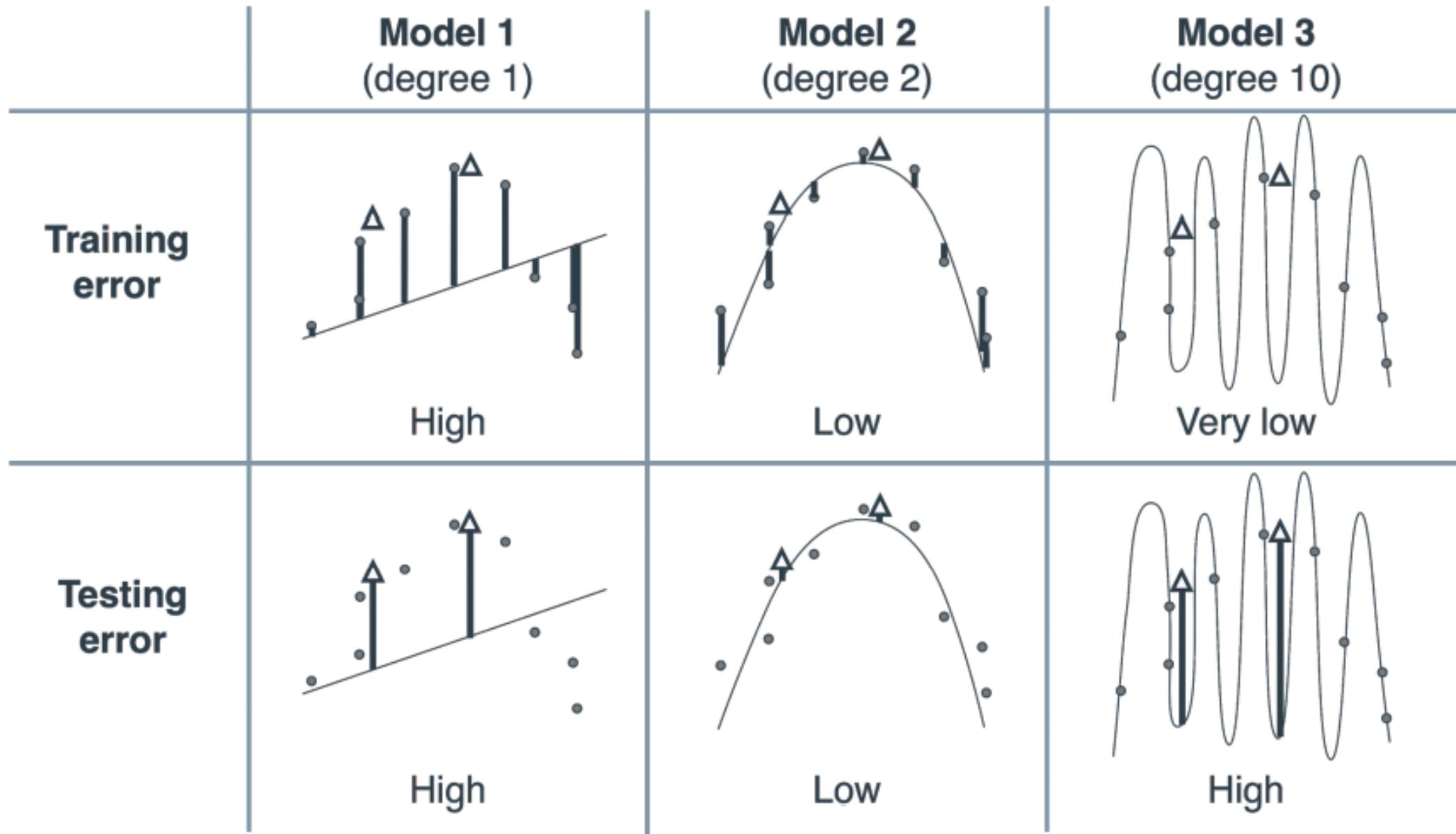
Model 1
Polynomial of degree 1
(a line)



Model 2
Polynomial of degree 2
(a parabola)



Model 3
Polynomial of degree 10



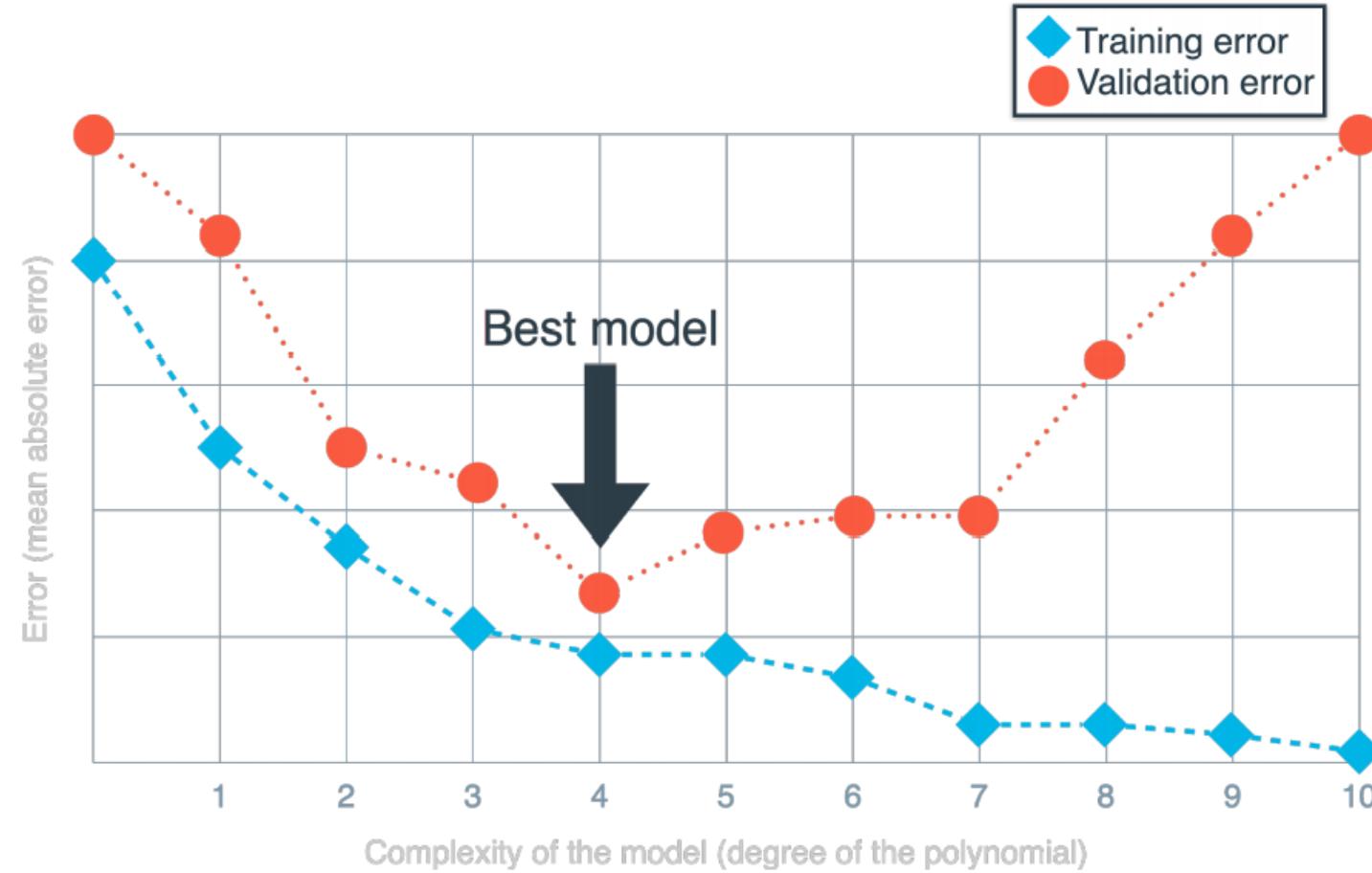
Optimizing the training process

Let's summarize what we have.

Models can

- Underfit: Use a model that is too simple to our dataset.
- Fit the data well: Use a model that has the right amount of complexity for our dataset.
- Overfit: Use a model that is too complex for our dataset.

A numerical way to decide how complex our model should be - The model evaluation graph



Regularization

Regularization

- We simply assign cost to function that also measures how complex is the model, so that at each stage, the model becomes simpler and simpler.
- Thus, our cost function consists of two parts, the part that measures performance and the part that measures complexity.
- Our new cost function looks like this:

New cost function = Performance cost function + Complexity cost function.

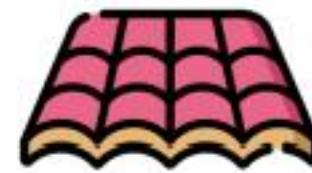


Problem: Broken roof



Roofer 1

Solution: Bandage
(Underfitting)



Roofer 2

Solution: Shingles
(Correct)



Roofer 3

Solution: Titanium
(Overfitting)

Movie recommendations

- Given that the model is a linear regression model, the equation for the predicted time the user will watch movie 11 is linear, and it will look like this:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + w_8x_8 + \\ w_9x_9 + w_{10}x_{10} + b$$

where:

- y is the amount of time the model predicts that the user will watch movie 11,
- x_i is the amount of time the user watched movie i ,
- w_i is the associated weight (given by the model),
- b is the bias.

Movie recommendations

- Now I would like to test your intuition.
- Out of the following two models (given by their equation), which ones look like they may be overfitting?

Model 1: $y = 2x_3 + 1.4x_7 - 0.5x_9 + 4$

Model 2:

$y = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8 + 378x_9 - 25x_{10} + 8$

Measuring how complex a model is - L1 and L2 norm

Model 1: $y = 2x_3 + 1.4x_7 - 0.5x_9 + 8$

Model 2:

$y = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8 + 378x_9 - 25x_{10} + 8$

L1 norm:

- **Model 1:** $|2| + |1.4| + |-0.5| = 3.9.$
- **Model 2:**
 $|22| + |-103| + |-14| + |109| + |-93| + |203| + |87| + |-55| + |378| + |-25| = 1,089$

Measuring how complex a model is - L1 and L2 norm

L2 norm:

- **Model 1:** $2^2 + 1.4^2 + (-0.5)^2 = 6.21.$
- **Model 2:**
 $22^2 + (-103)^2 + (-14)^2 + 109^2 + (-93)^2 + 203^2 + 87^2 + (-55)^2 + 378^2 + (-25)^2 = 227,131$

Lasso regression and ridge regression

- Lasso stands for “Least absolute shrinkage and selection operator”). The error function is the following.

Lasso regression cost = Regression cost + L1 Norm.

- The new cost function is the following.

Ridge regression cost = Regression cost + L2 Norm.

The Regularization parameter

- We multiply the regularization cost by lambda, and train the model
- using the sum of the regression and the regularization cost.

New cost = Regression cost + lambda Regularization cost

The Regularization parameter

- Imagine that we've trained the model, and the equation we get is the following.

Model:

$$y = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8 + 378x_9 - 25x_{10} + 8$$

The Regularization parameter

- If you use L1 regularization (Lasso regression), you end up with a model with less coefficients.
- In other words, L1 regularization will reduce the coefficients, and turn some of them into zero.
- Thus, you may end up with an equation such as the following:

$$y = 2x_3 + 1.4x_7 - 0.5x_9 + 8.$$

The Regularization parameter

- If you use L2 regularization (Ridge regression), you end up with a model with smaller coefficients.
- In other words, L2 regularization will reduce all the coefficients, but rarely turn them into zero.
- Thus, you may end up with an equation such as the following:

$$y = 0.2x_1 - 0.8x_2 - 1.1x_3 + 2.4x_4 - 0.03x_5 + 1.02x_6 + 3.1x_7 - 2x_8 + 2.9x_9 - 0.04x_{10} + 8$$

An intuitive way to see regularization

1. Start with random coefficients.
2. Repeat many times:
 - a) Pick a random point (or points)
 - b) Modify our coefficients slightly to move the model closer to the points.
3. Enjoy your model!

An intuitive way to see regularization

1. Start with random coefficients.
2. Repeat many times:
 - Pick a random point (or points)
 - Modify our coefficients slightly to move the model closer to the points.
 - Slightly reduce the coefficients using method 1 or method 2.
3. Enjoy your model!

An intuitive way to see regularization

- Notice what happens if we use method 1 to reduce our coefficient, and we do this during 200 epochs.
- We get the following sequence of values:

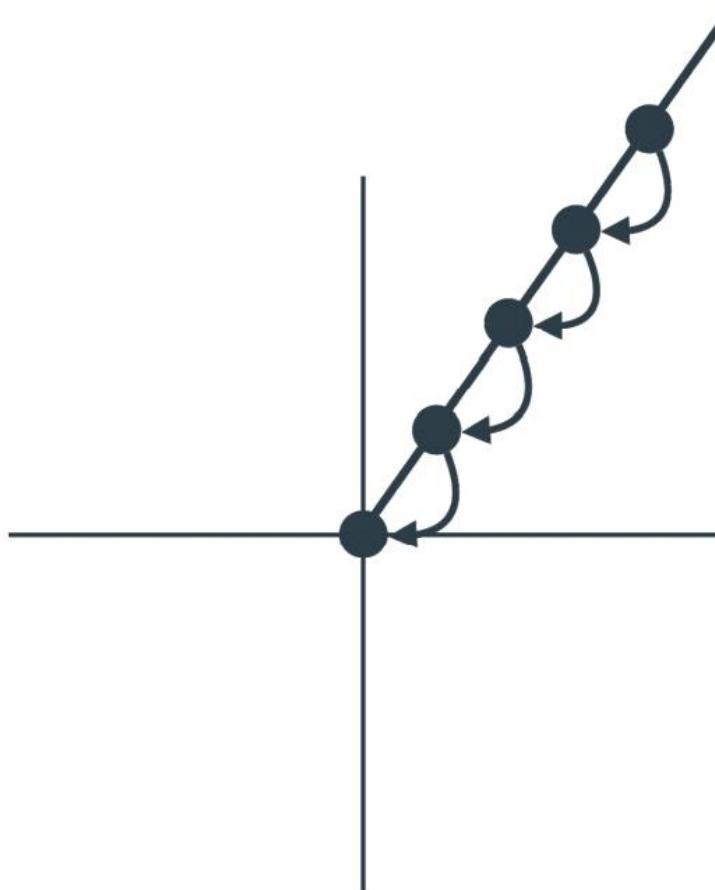
2 → 1.99 → 1.98 → ... → 0.02 → 0.01 → 0 .

An intuitive way to see regularization

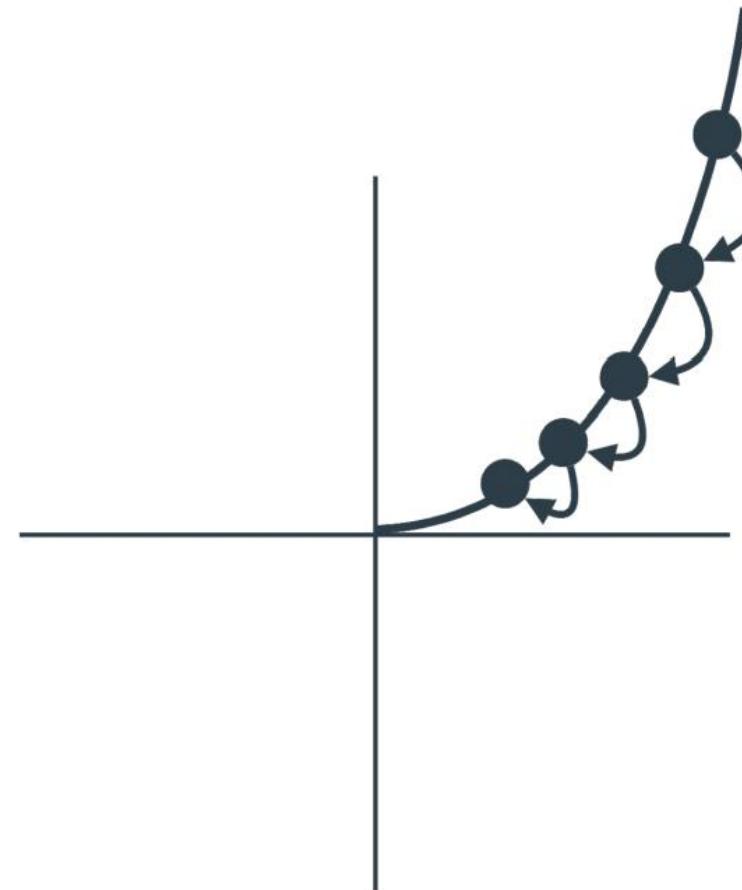
- Notice that after 200 epochs of our training, the coefficient becomes 0, and it never changes again.
- Now let's see what would happen if we apply method 2.
- We get the following sequence of values:

2 → 1.98 → 1.9602 → ... → 0.2734 → 0.2707 → 0.2680.

An intuitive way to see regularization



L1-regularization



L2-regularization

Summary

- When it comes to training models, many problems arise.
- Two problems that come up a lot are underfitting and overfitting.
- Underfitting is when we use a very simple model to fit our dataset.
- Overfitting is when we use an overly complex model to fit our dataset.
- An effective way to tell overfitting and underfitting apart is by using a testing dataset.

5: Using lines to split our points: The perceptron algorithm

Using lines to split our points: The perceptron algorithm

This lesson covers

- What is classification?
- Sentiment analysis: How to tell if a sentence is happy or sad using machine learning.
- What are perceptron's, or discrete logistic classifiers.
- What is the perceptron algorithm?
- How to draw a line that separates points of two different colors.
- How to measure the error of a line that separates points of two different colors.
- How to code the perceptron algorithm in Python.

Using lines to split our points: The perceptron algorithm

Sentence 1: I feel awful today

Score: +0 +0 -4 +0 = -4

Prediction: The sentence is sad.

Sentence 2: Everything is wonderful I am delighted

Score: +0 +0 +4 +0 +0 +3
= +7

Prediction: The sentence is happy.

Using lines to split our points: The perceptron algorithm

1. How do we know the score of every single word?
2. The scores we came up with are based on our perception. How do we know these are the good scores?
3. What if we have to build a sentiment analysis in a language that we don't speak?

The problem: We are in an alien planet, and we don't know their language!

Dataset:

- Alien 1
 - Mood: Happy
 - Sentence: "Aack, aack, aack!"
- Alien 2:
 - Mood: Sad
 - Sentence: "Beep beep!"
- Alien 3:
 - Mood: Happy
 - Sentence: "Aack beep aack!"
- Alien 4:
 - Mood: Sad
 - Sentence: "Aack beep beep beep!"

The problem: We are in an alien planet, and we don't know their language!

Data



Aack aack aack!



Beep beep!



Aack beep aack!



Aack beep beep beep!

Prediction

Is this alien happy or sad?



aack beep aack aack!

Simple sentiment analysis classifier

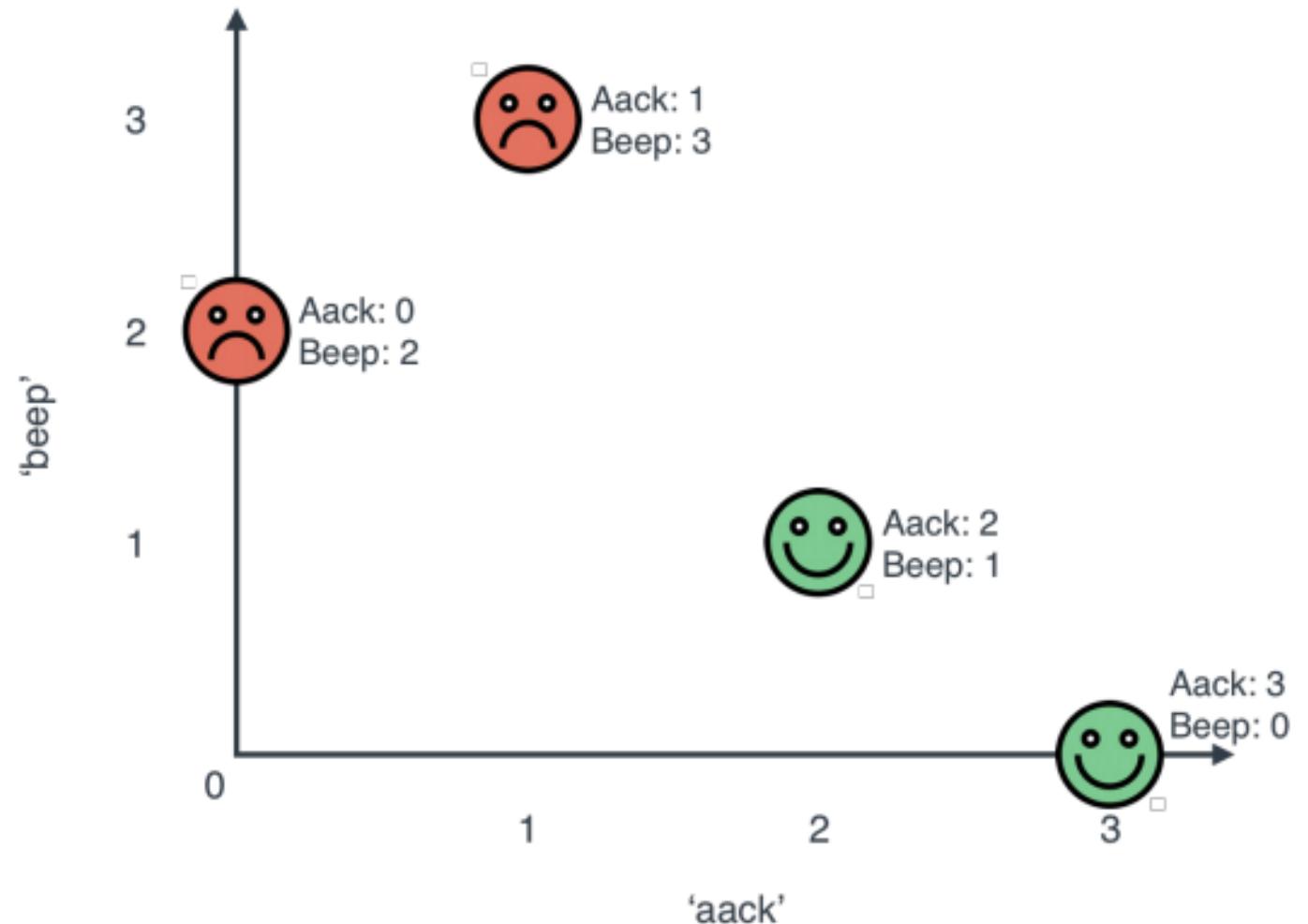
- Count the number of appearances of ‘aack’ and ‘beep’.
- If there are more ‘aack’s, then the alien is happy.
- If there are more ‘beep’s, then the alien is sad.

Notice that we didn’t specify what happens when the number of ‘aack’s is equal to the number of ‘beep’s.

Mathematical sentiment analysis classifier

Sentence	Aack	Beep	Mood
Aack aack aack!	3	0	 Happy
Beep beep!	0	2	 Sad
Aack beep aack!	2	1	 Happy
Aack beep beep beep!	1	3	

Mathematical sentiment analysis classifier



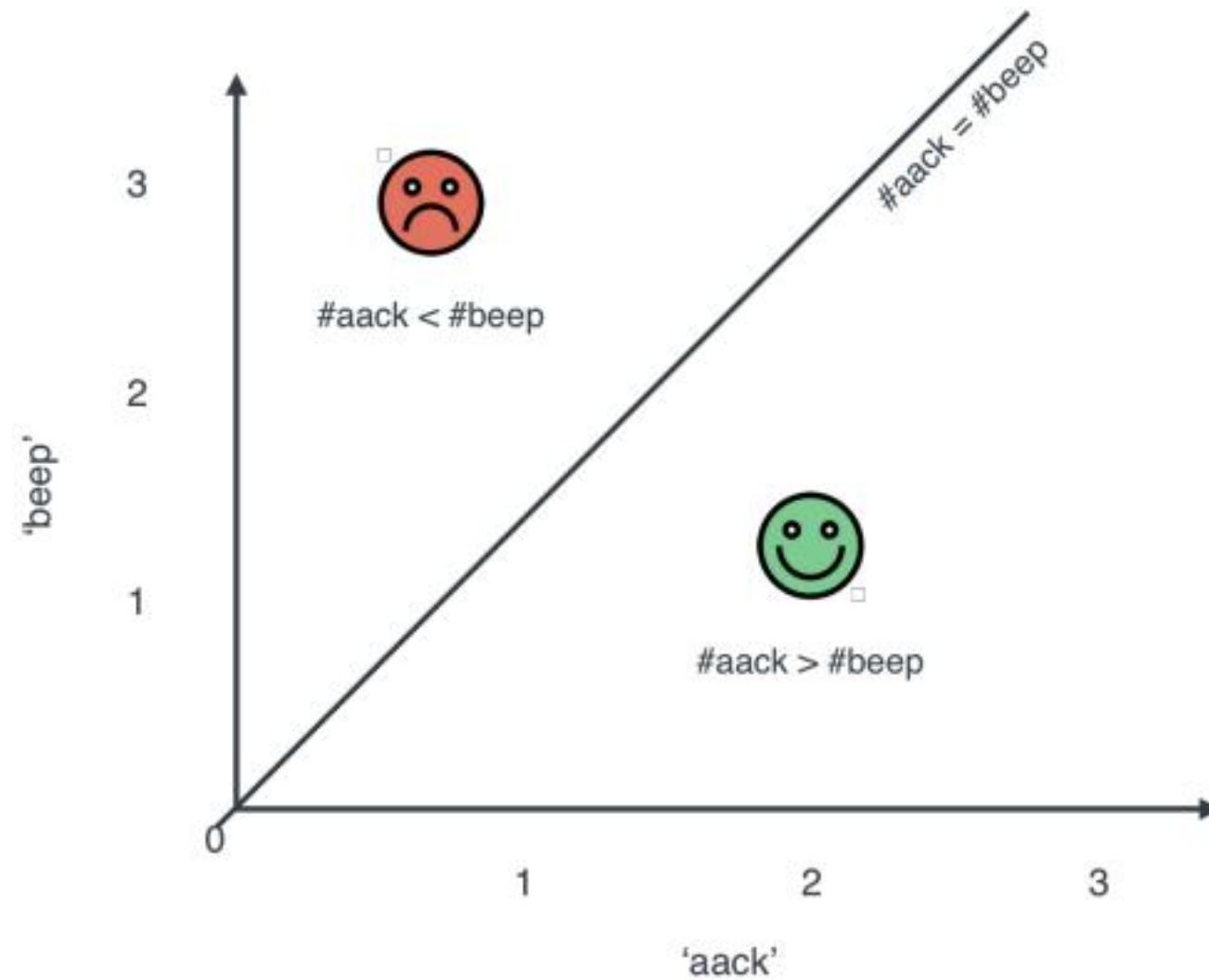
Mathematical sentiment analysis classifier

$$\#aack - \#beep = 0.$$

If you like equations with variables, this is simply the equation $y=x$, or equivalently, the equation $y-x=0$. We also have two areas, as follows:

- The positive (happy) area, where the sentences have more aack's than beep's, with equation $\#aack - \#beep > 0$.
- The negative (sad) area, where the sentences have more beep's than aack's, with equation $\#aack - \#beep < 0$.

Mathematical sentiment analysis classifier



Geometric sentiment analysis classifier

The classifier is defined by a line of equation
#aack - #beep = 0, and a rule.

Rule:

- If the point is in the positive region of this line, with equation
#aack - #beep > 0, or over the line, then the alien is classified as happy.
- If the point is in the negative region of this line, with equation
#aack - #beep < 0, then the alien is classified as sad.

A slightly more complicated planet

Dataset:

- Alien 1:
 - Mood: Sad
 - Sentence: "Crack!"
- Alien 2:
 - Mood: Sad
 - Sentence: "Dunk."
- Alien 3:
 - Mood: Sad
 - Sentence: "Crack dunk!"
- Alien 4:
 - Mood: Happy
 - Sentence: "Crack crack dunk dunk."
- Alien 5:
 - Mood: Happy
 - Sentence: "Crack dunk dunk crack crack!"
- Alien 6:
 - Mood: Happy
 - Sentence: "Dunk dunk crack dunk crack!"

A slightly more complicated planet

Sentence	Crack	Dunk	Mood
Crack!	1	0	 Sad
Dunk dunk!	0	2	 Sad
Crack dunk!	1	1	 Sad
Crack dunk dunk!	1	2	 Sad

A slightly more complicated planet

Crack crack dunk dunk!	1	3	 Happy
Crack dunk dunk crack crack!	2	2	 Happy
Dunk dunk crack dunk crack!	3	2	 Happy
Crack dunk dunk crack dunk!	2	3	 Happy

Mathematical sentiment analysis classifier

- The classifier is defined by the scores, and a rule, as follows:

Scores:

1. Aack: 1 point
2. Beep: 1 points

Rule: Add the scores of all the words.

1. If the score is larger than or equal to 3.5, predict that the alien is happy.
2. If the score is smaller than 3.5, predict that the alien is sad.

Modified mathematical sentiment analysis classifier



Geometric sentiment analysis classifier

The classifier is defined by a line of equation
#crack + #dunk - 3.5 = 0, and a rule.

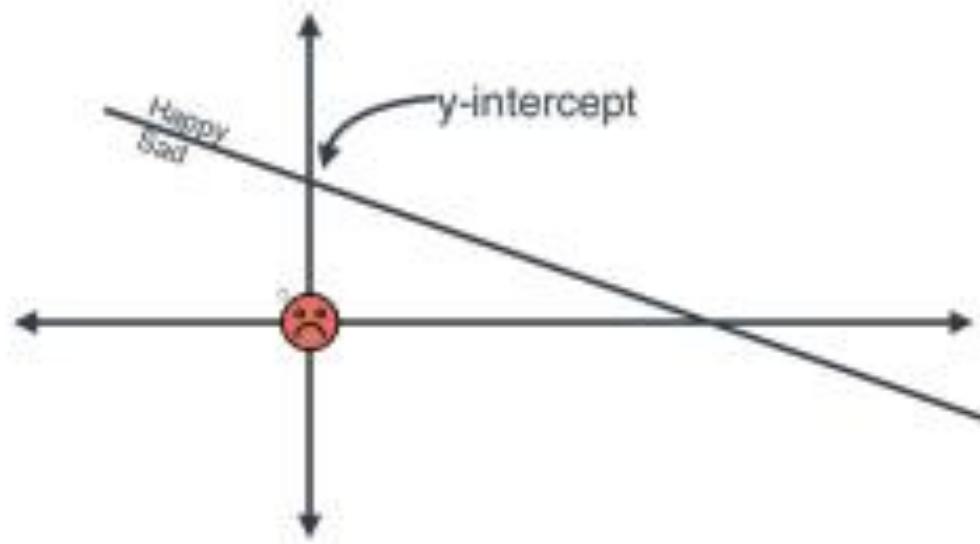
Rule:

- If the point is in the positive region of this line, with equation
#crack - #dunk - 3.5 >0, then the alien is classified as happy.
- If the point is in the negative region of this line, with equation
#crack - #dunk - 3.5 <0, then the alien is classified as sad.

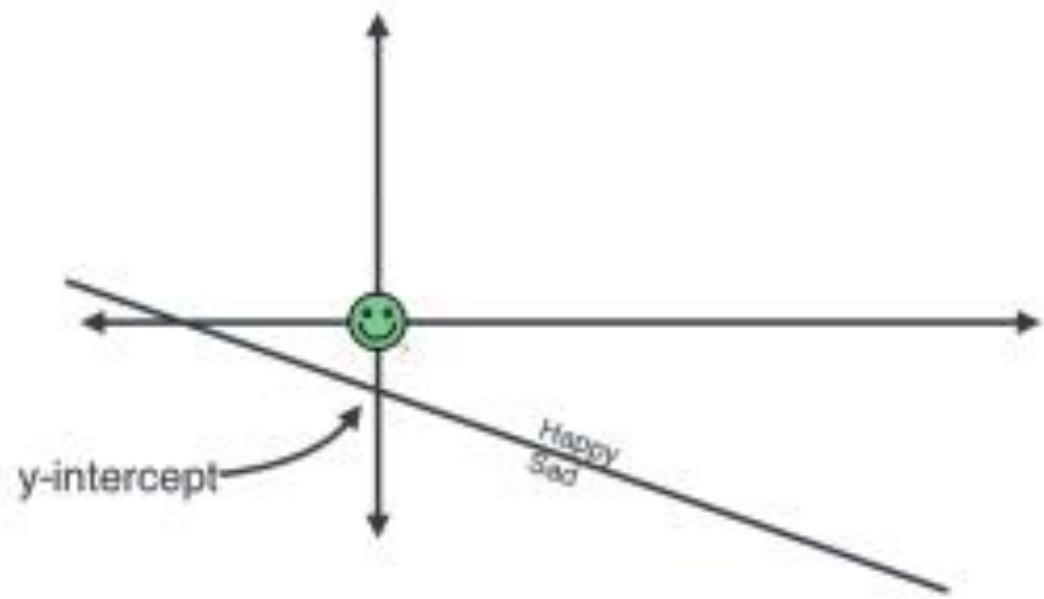
Geometric sentiment analysis classifier



The bias, the y-intercept, and the inherent mood of a quiet alien

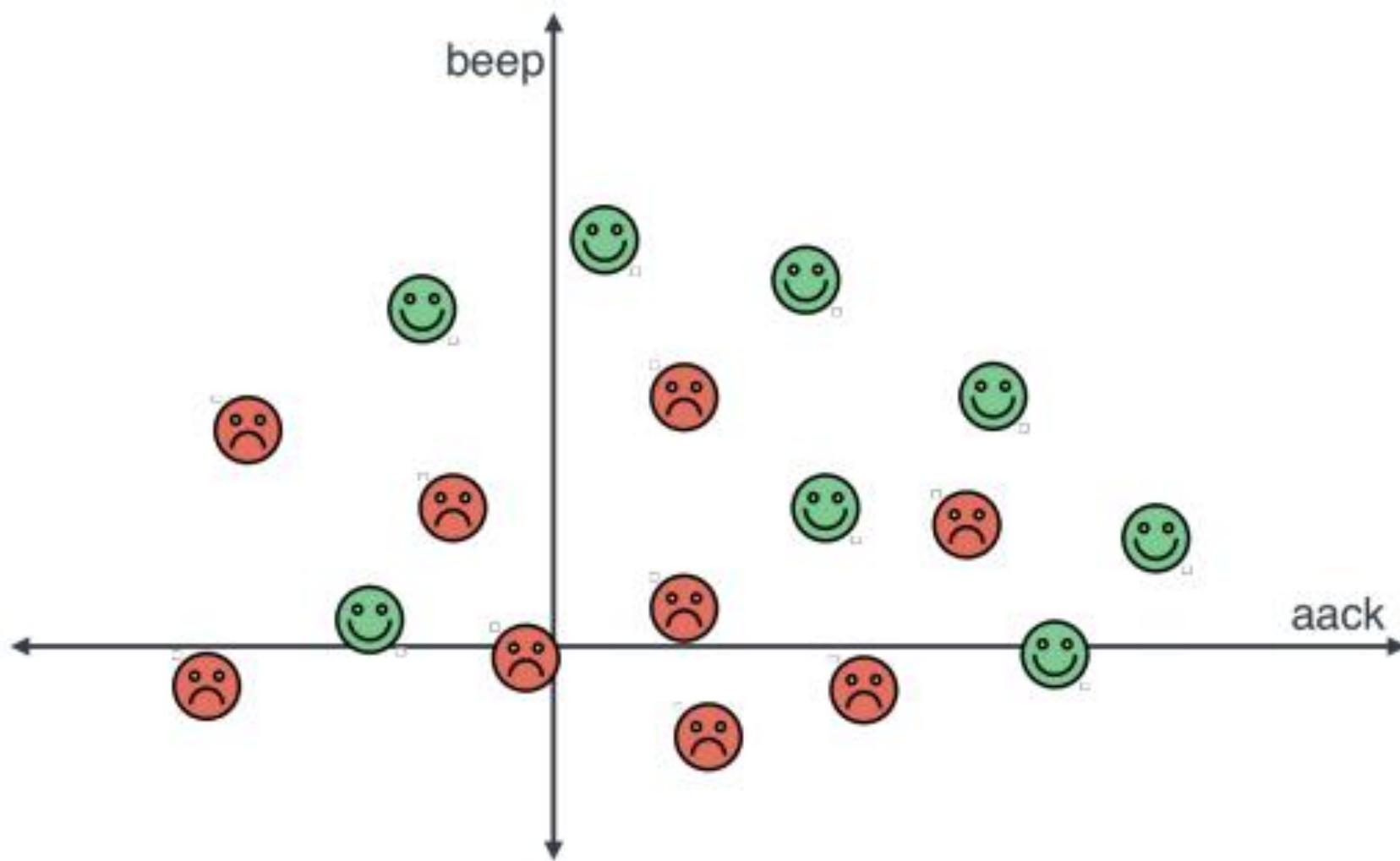


Negative bias
(positive y-intercept)
Quiet alien is sad

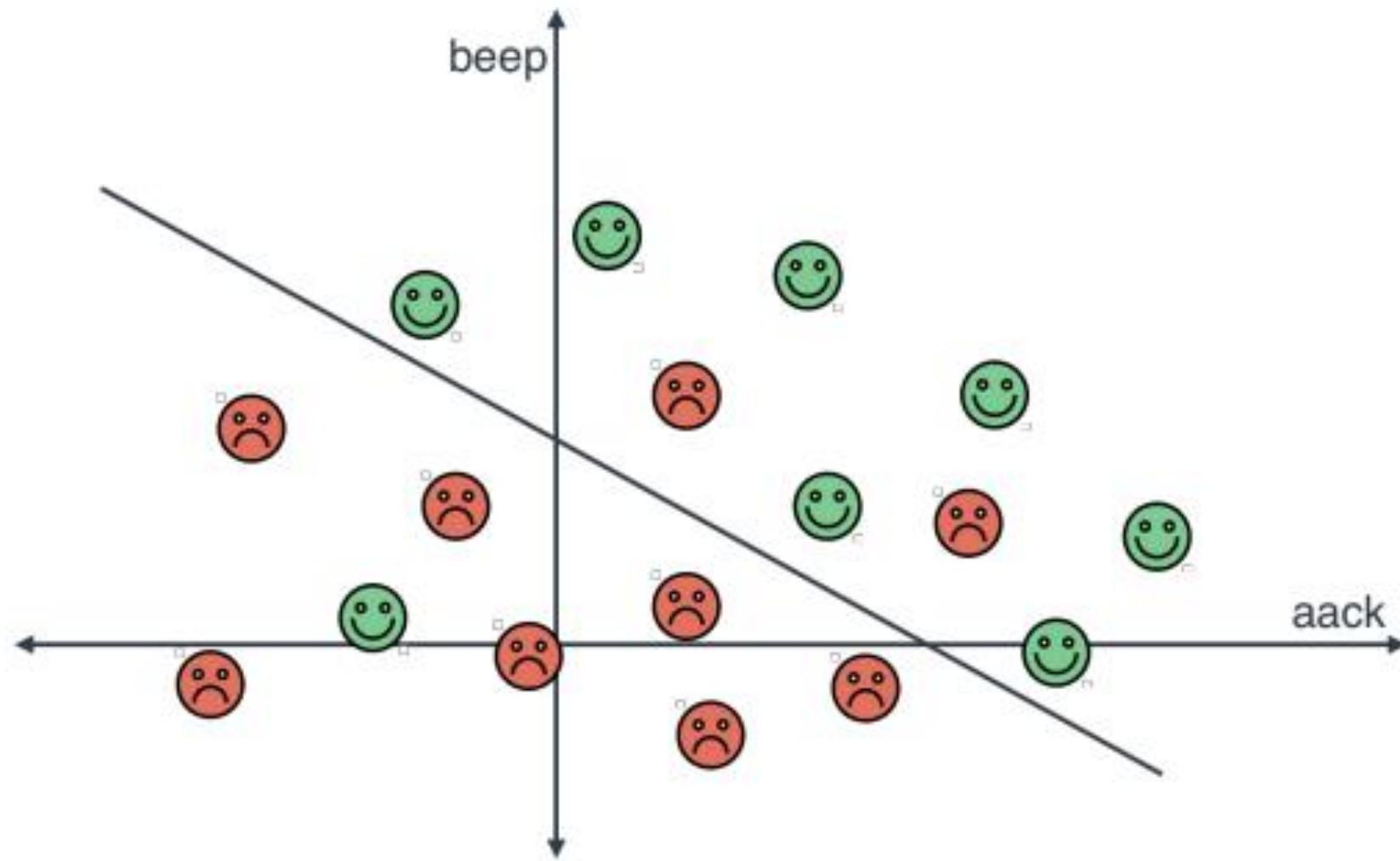


Positive bias
(negative y-intercept)
Quiet alien is happy

More general cases

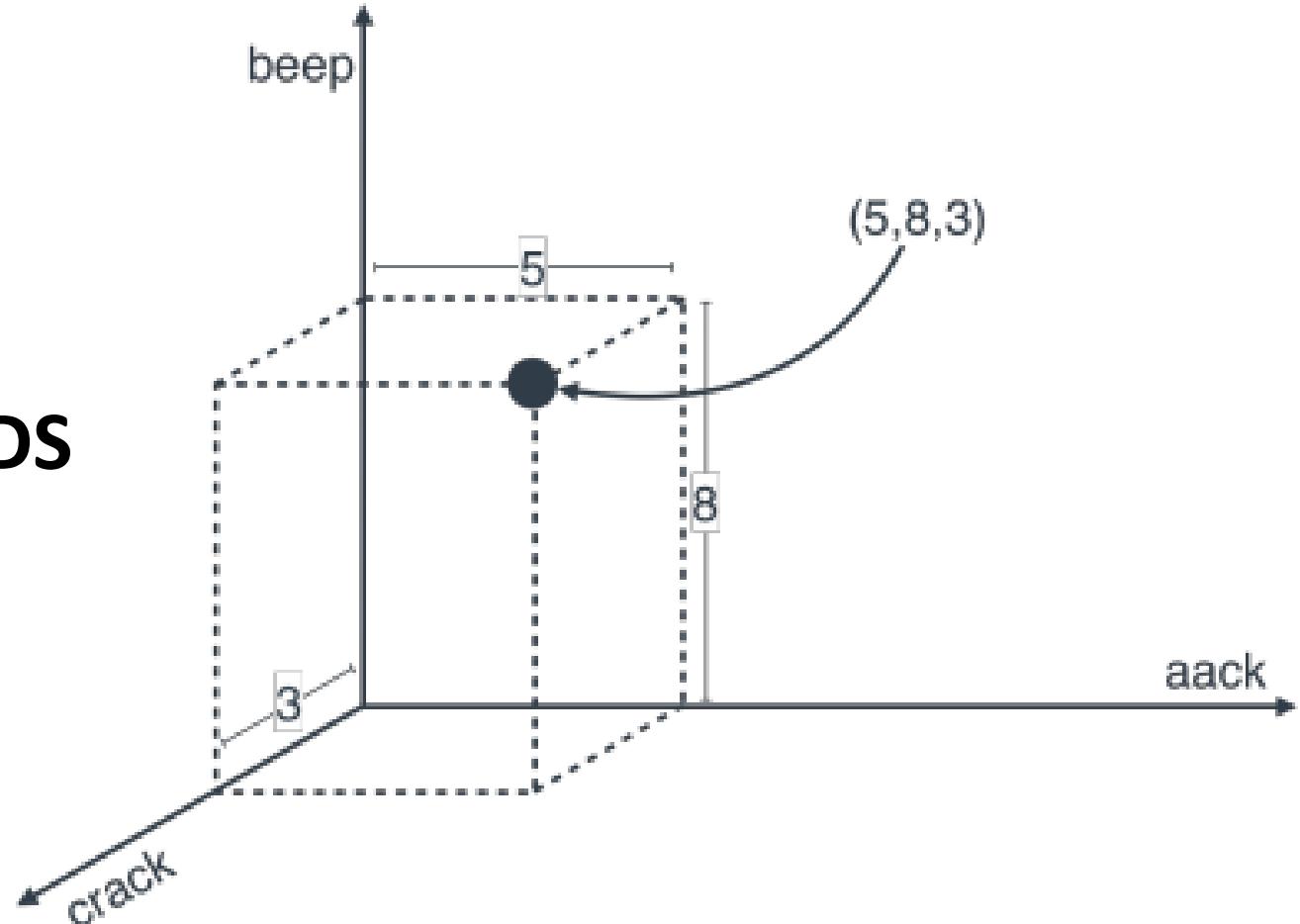


More general cases

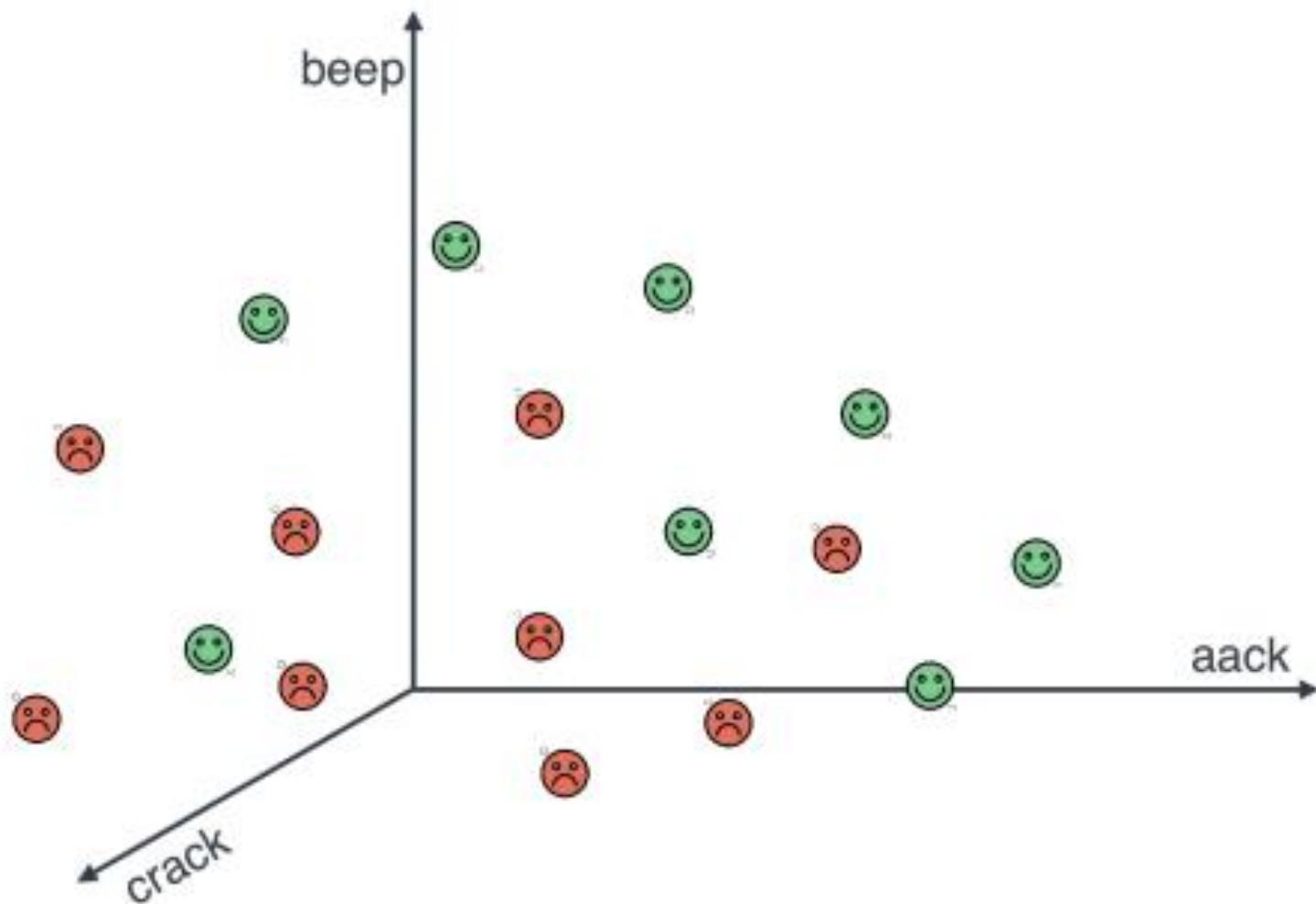


More general cases

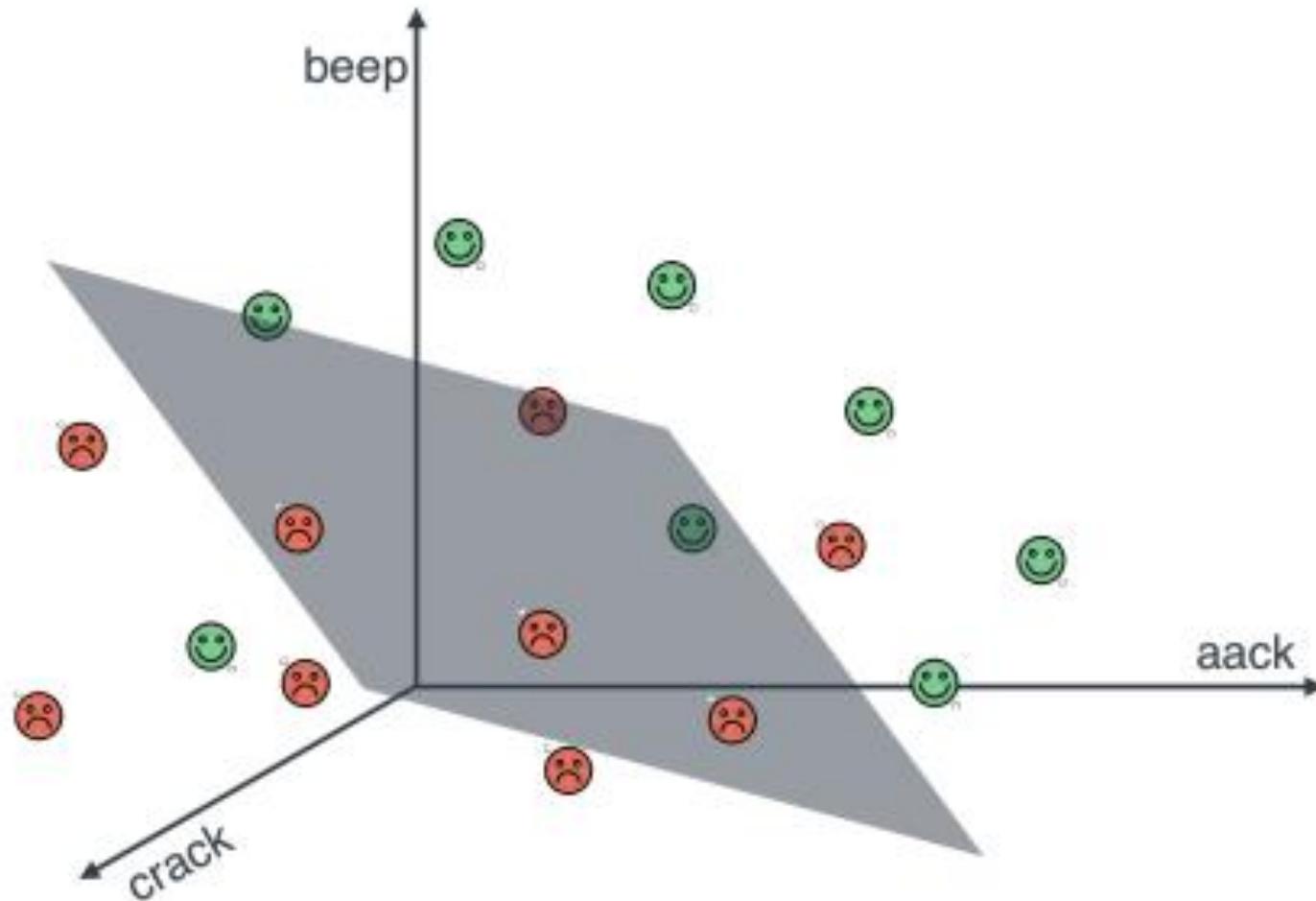
ALPHABETS WITH 3 WORDS



More general cases



More general cases



ALPHABETS WITH MANY WORDS

Scores:

- A: 0.1 points
- Aardvark: 1.2 points
- Aargh: -4 points
-
- Zygote: 0.4 points.

Threshold:

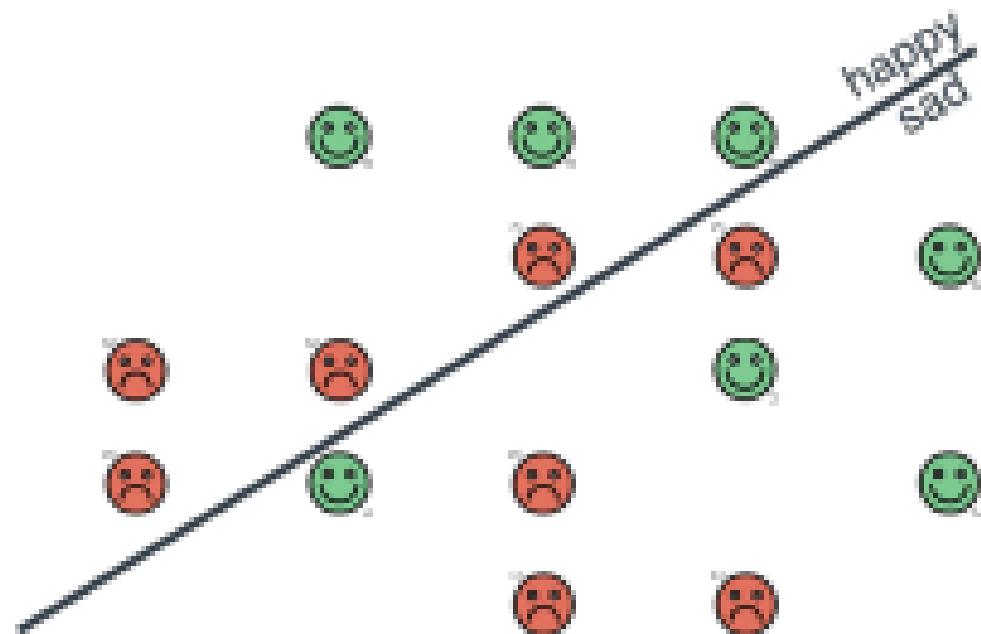
- 2.3 points.

How do we determine if a classifier is good or bad? The error function

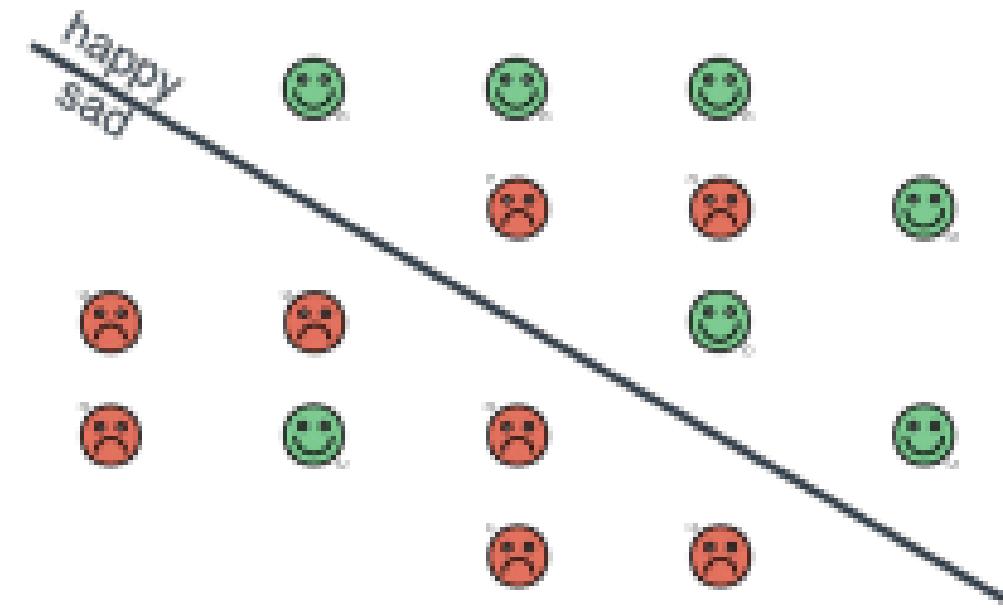
This will be done in a similar way as we did for regression in lesson three, namely:

1. Start with a set of random scores, and a random threshold.
2. Take a step to improve these scores just a little bit.
3. Repeat step 2 many times.
4. Enjoy your classifier!

How to compare classifiers? The error function

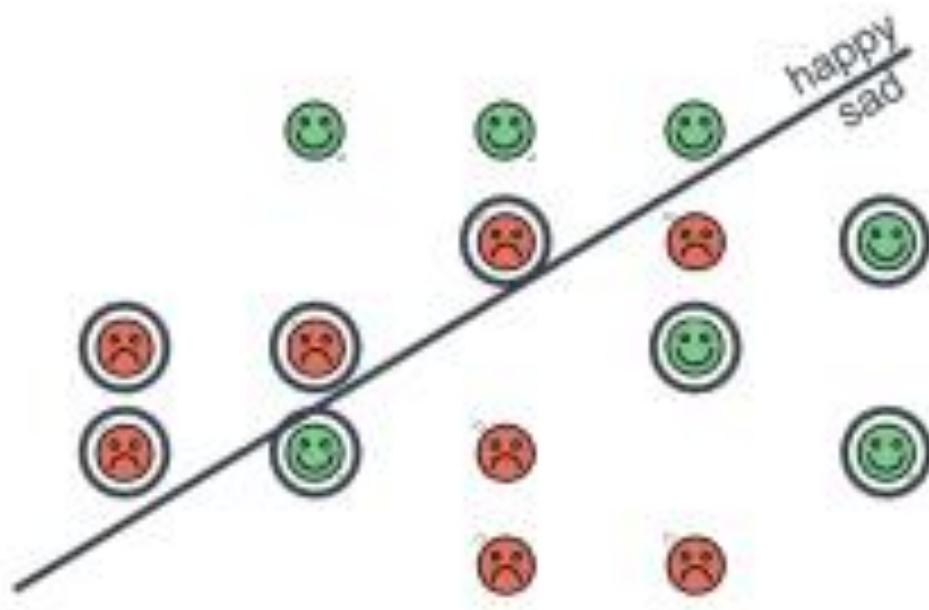


Bad classifier



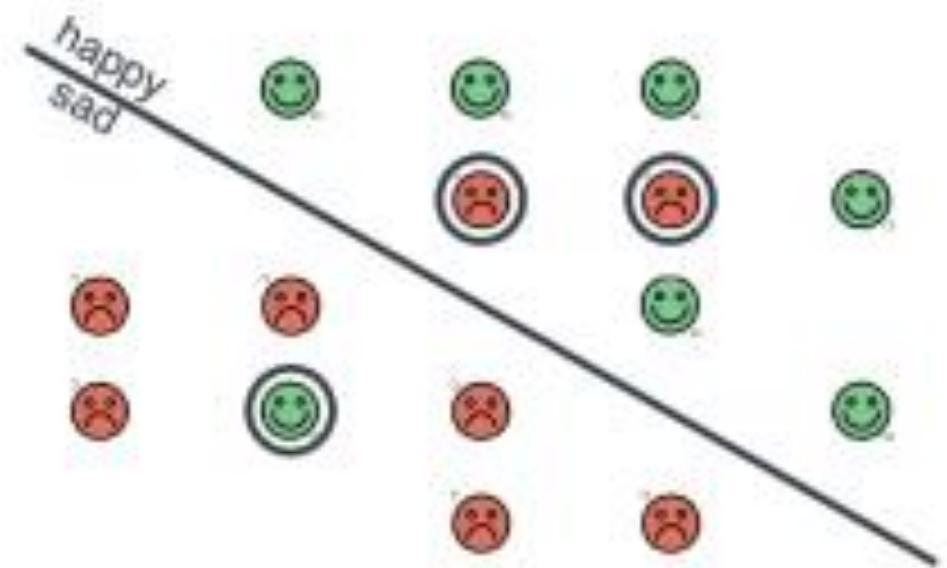
Good classifier

How to compare classifiers? The error function



Bad classifier

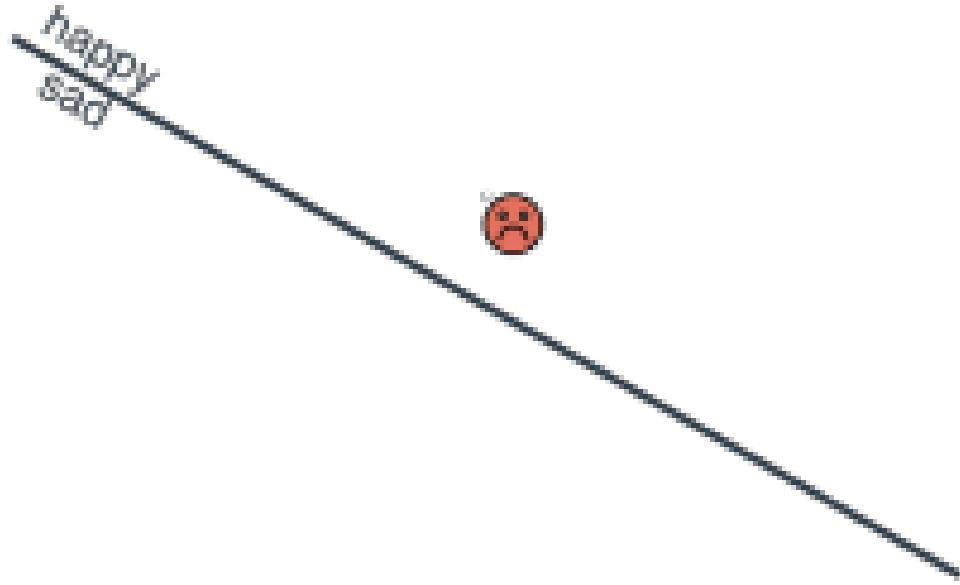
Error: 8



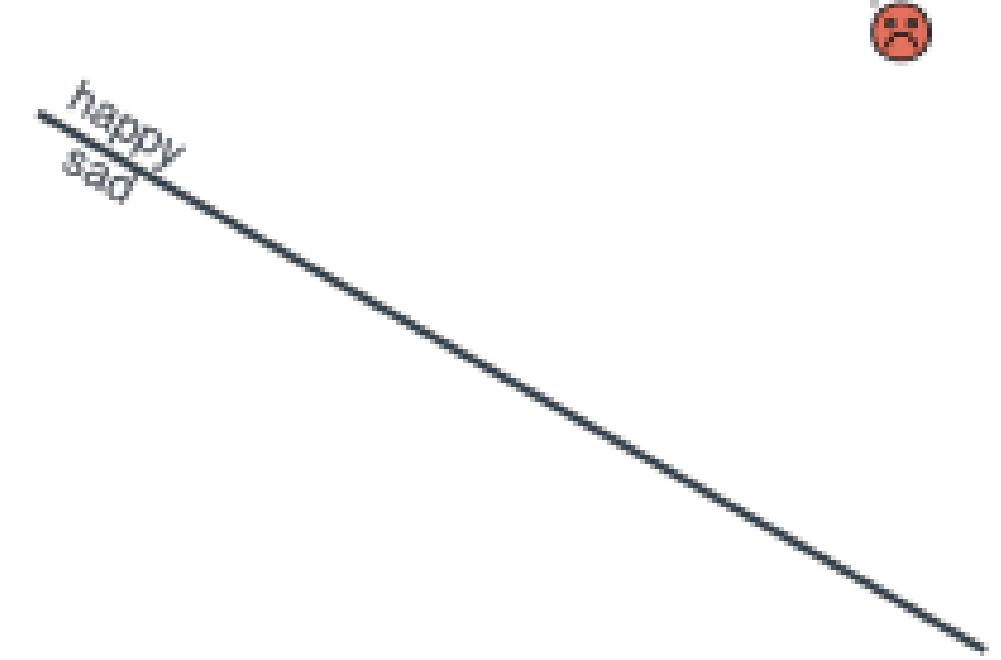
Good classifier

Error: 3

How to compare classifiers? The error function

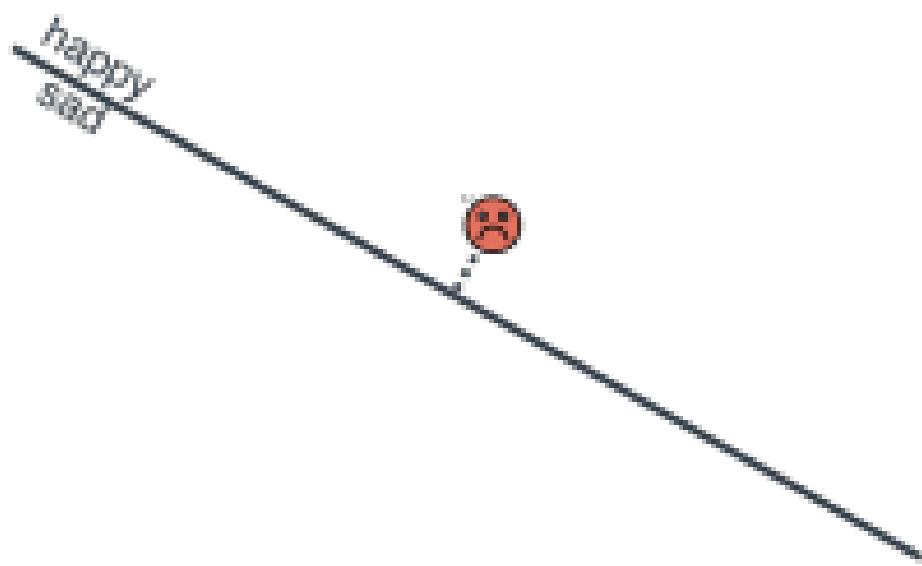


Not badly misclassified

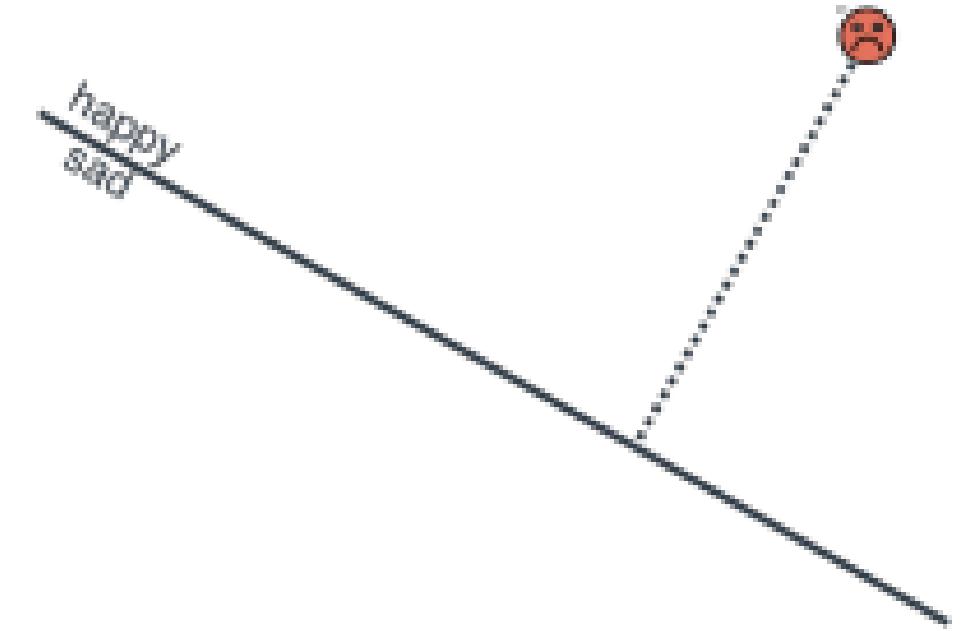


Badly misclassified

How to compare classifiers? The error function

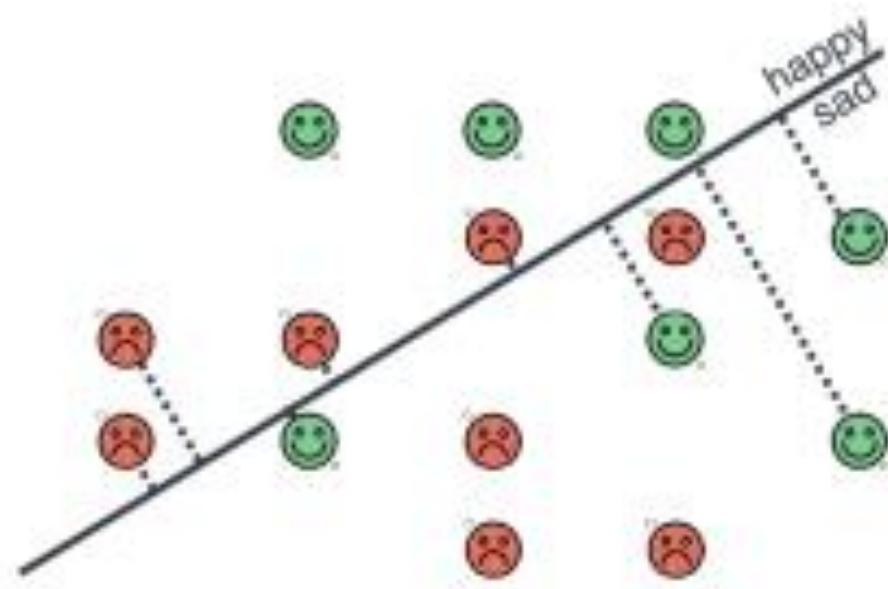


Small distance
Small error



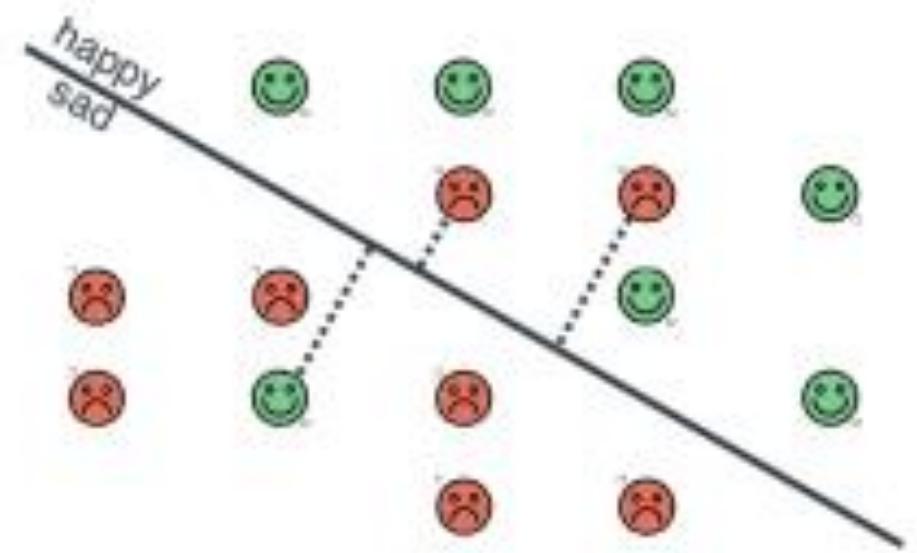
Large distance
Large error

How to compare classifiers? The error function



Bad classifier

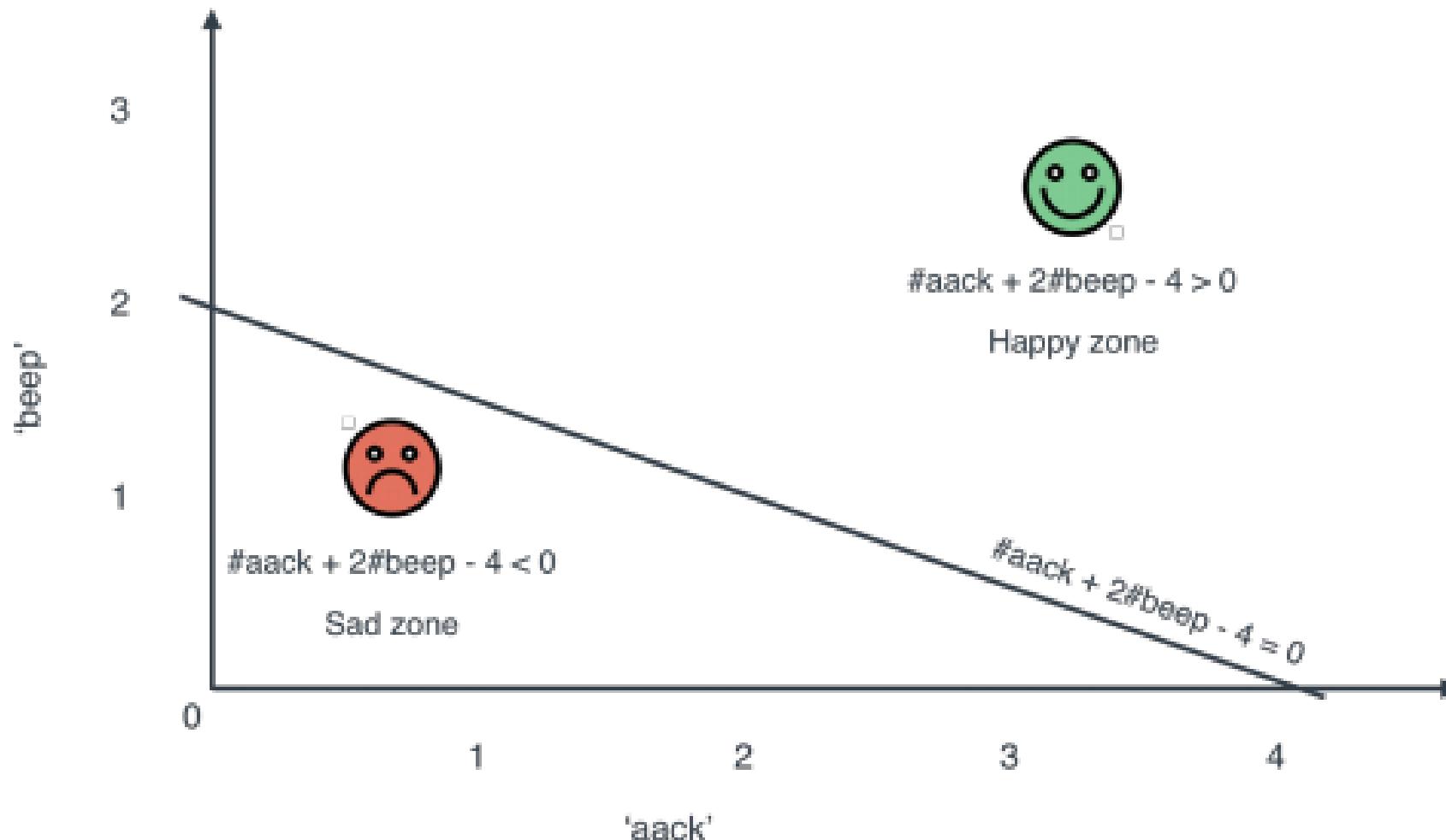
Error:



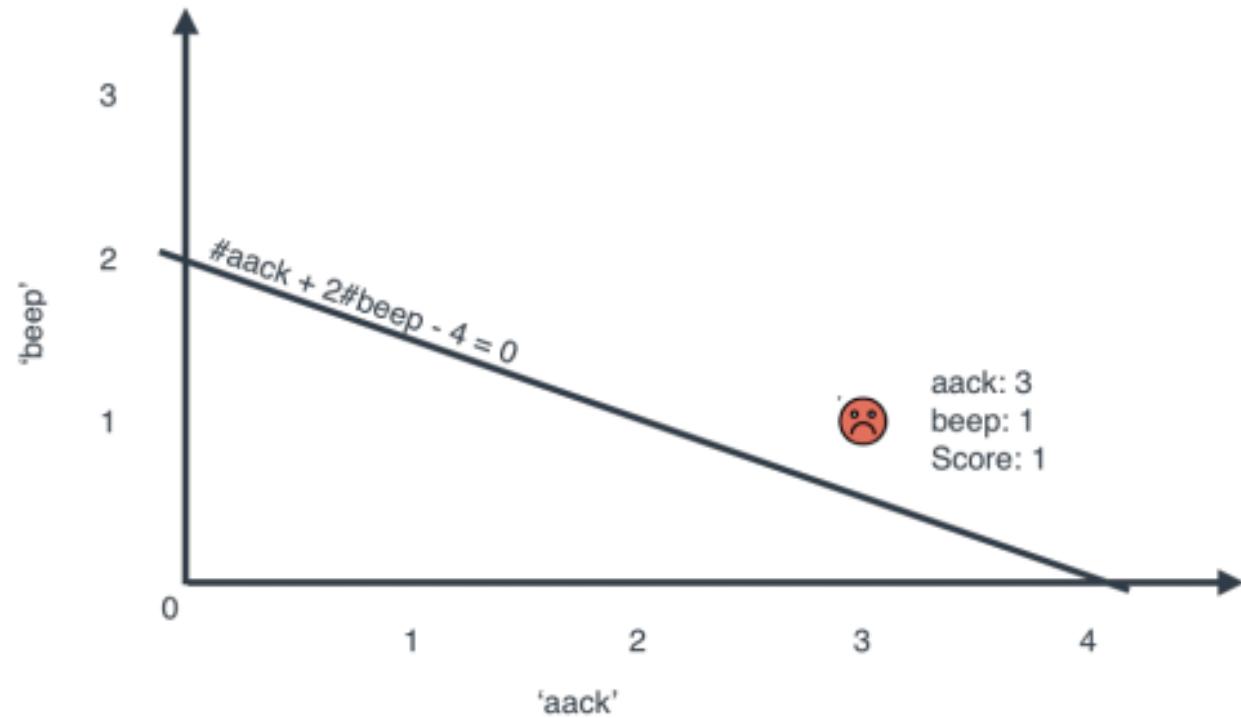
Good classifier

Error:

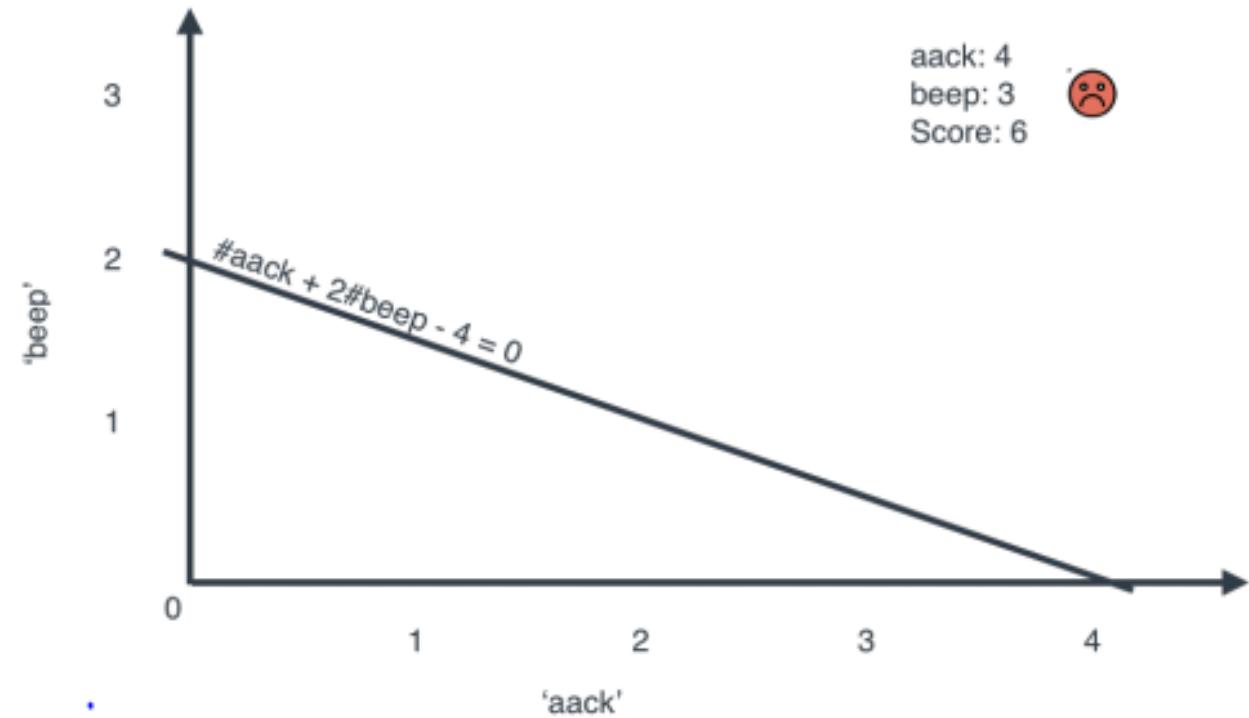
How to compare classifiers? The error function



How to compare classifiers? The error function



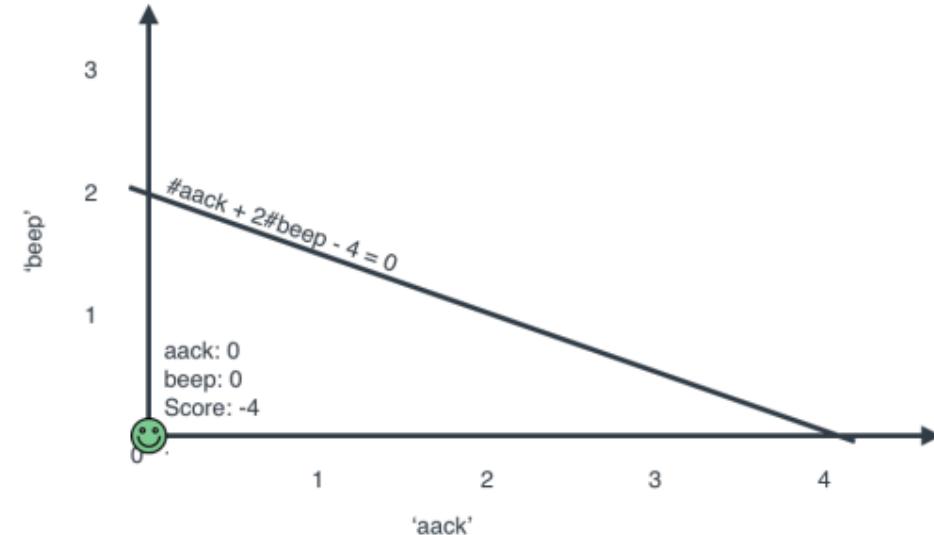
Not very badly classified point



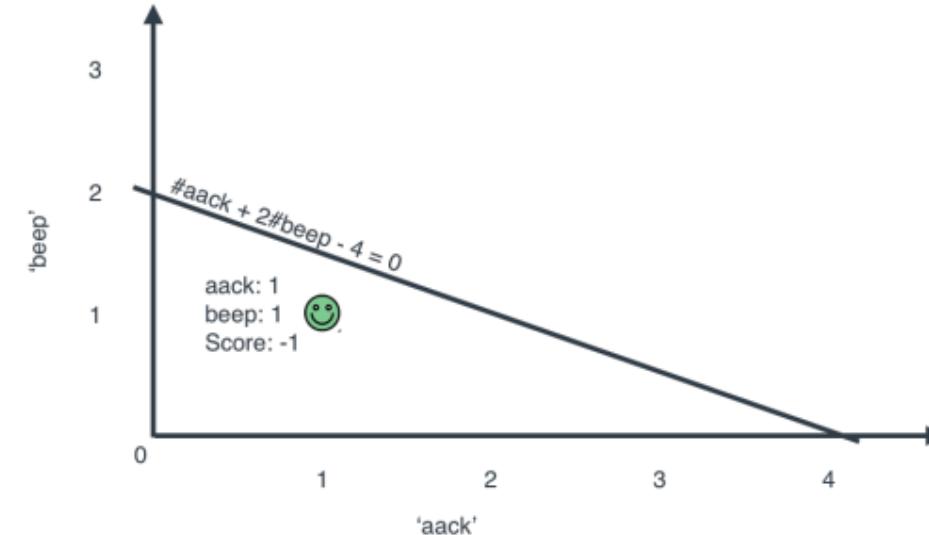
Very badly classified point

How to compare classifiers? The error function

- In this case, as you can see in this Figure, Sentence 3 is very badly misclassified, while Sentence 4 is misclassified, but not as badly, since it is much closer to the boundary line.



Very badly classified point



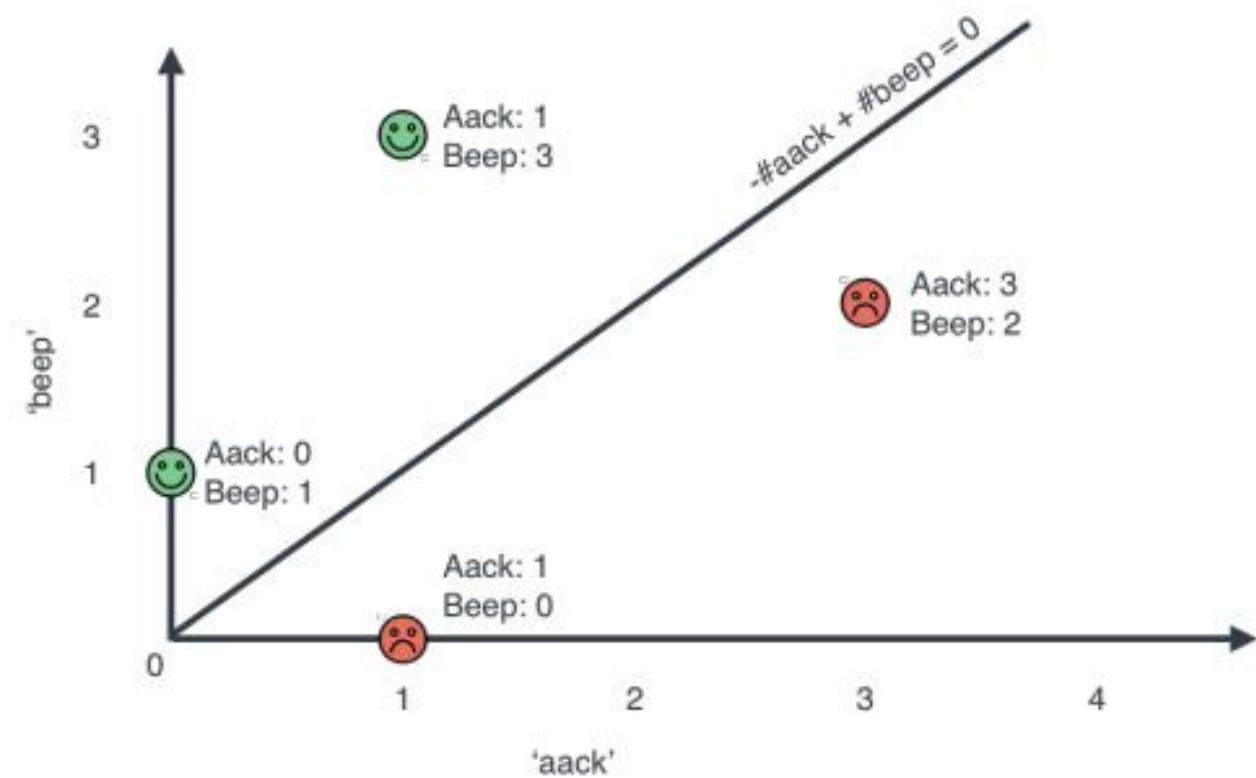
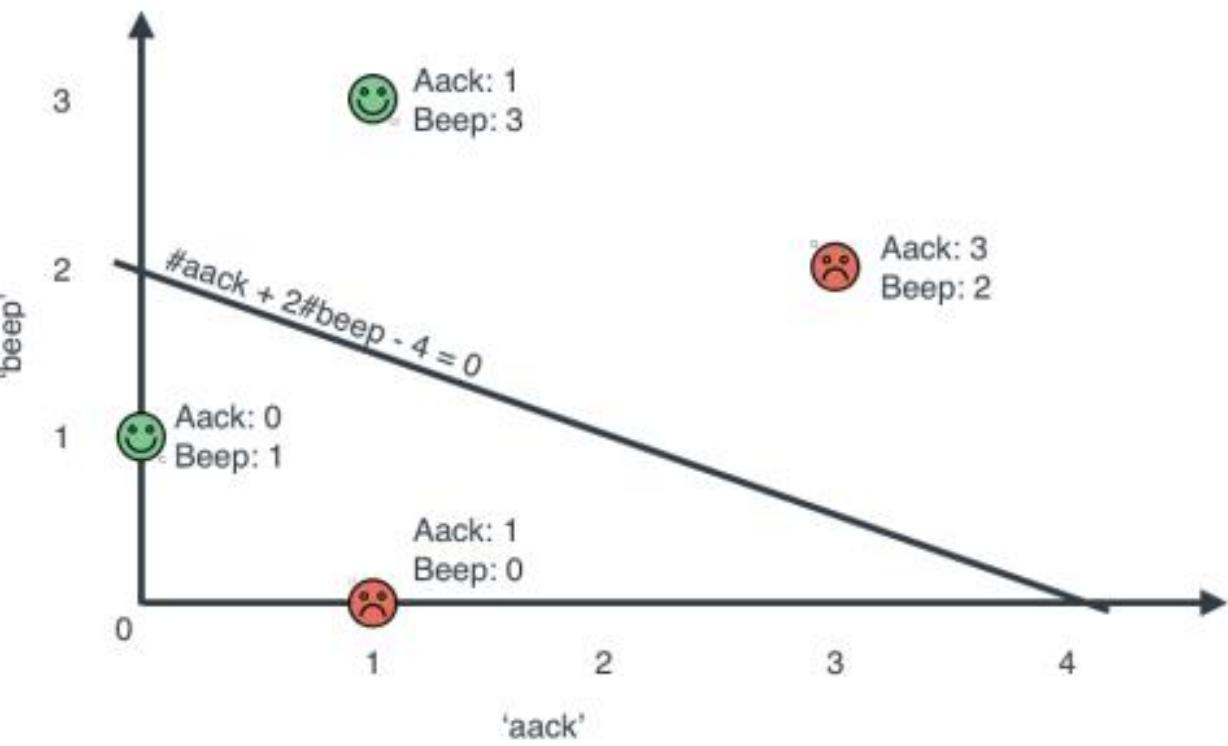
Not very badly classified point

How to compare classifiers? The error function

Perceptron error

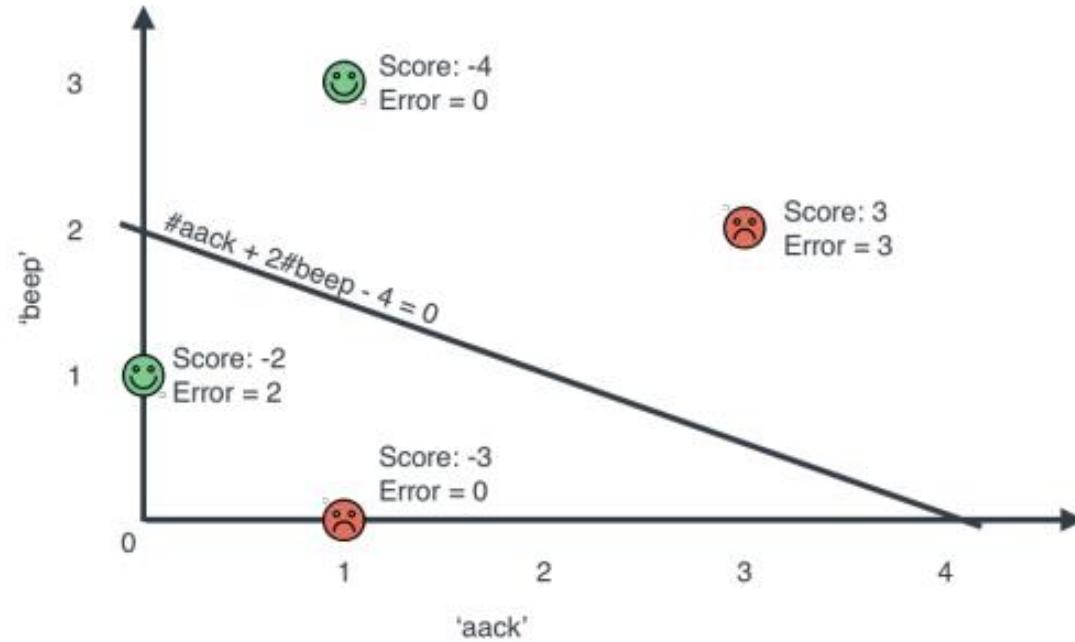
- If the point is correctly classified:
 - **Error = 0**
- Else, if the point is incorrectly classified:
 - **Error = |Score|.**

How to compare classifiers? The error function

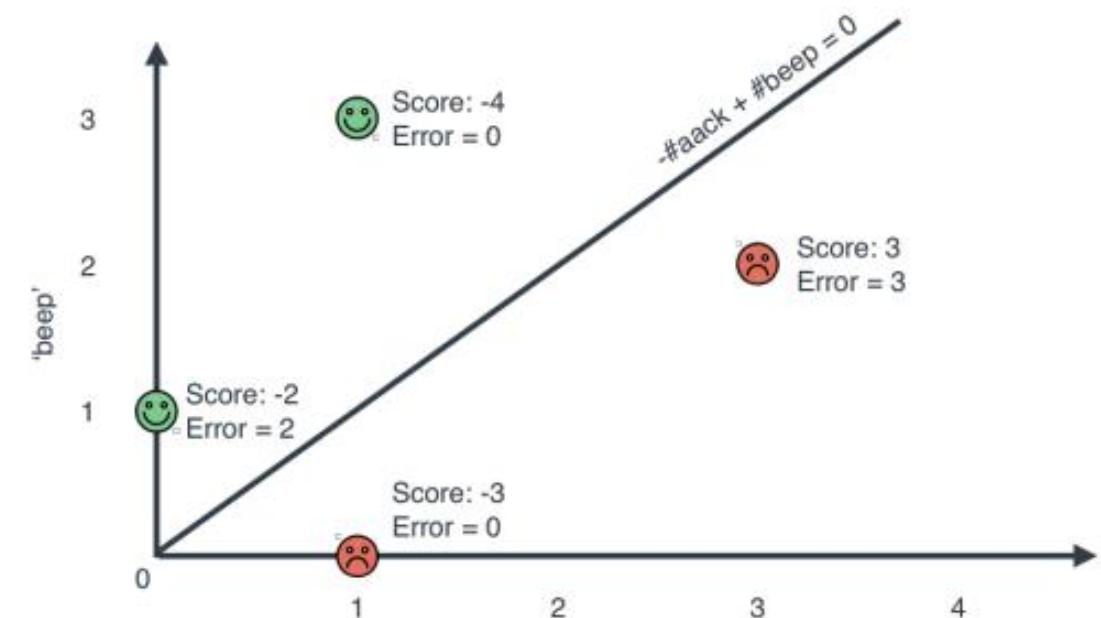


How to compare classifiers? The error function

- We then conclude that Classifier 2 is better than Classifier 1.
- The summary of these calculations is in this figure.



$$\text{Error} = 0 + 2 + 0 + 3 = 5$$



$$\text{Error} = 0 + 0 + 0 + 0 = 0$$

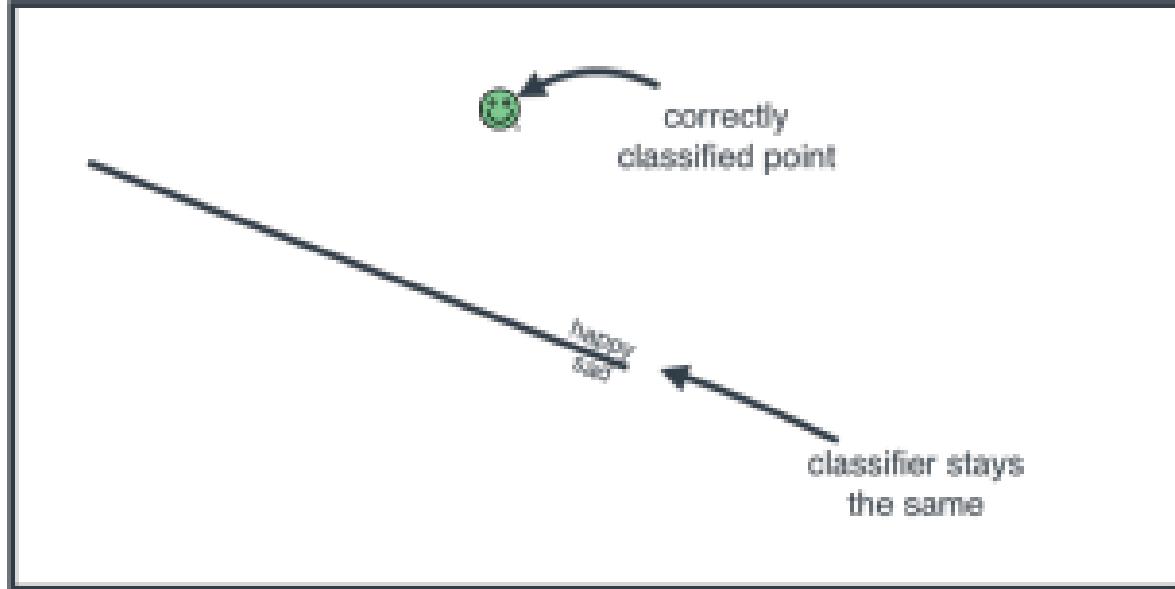
How to find a good classifier? The perceptron algorithm

Pseudocode for the perceptron algorithm:

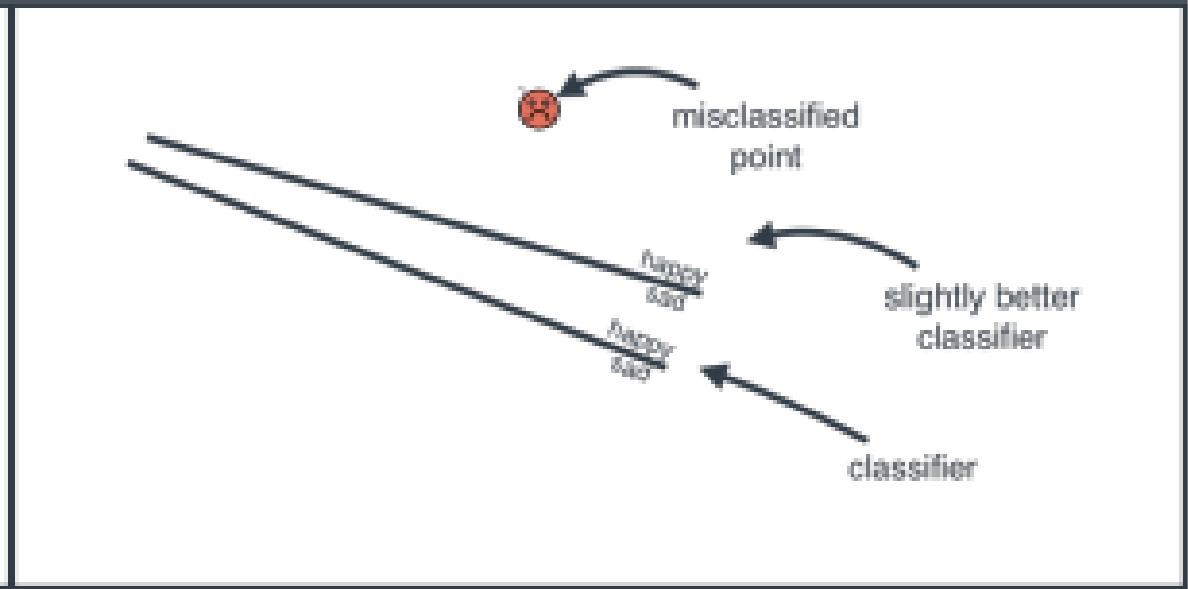
- Begin with a random classifier
- Loop many times:
 - Improve the algorithm a small amount
- Output a good classifier.

The perceptron trick

Case 1



Case 2



The perceptron trick

- Sentence 1 (sad): ‘Aack beep beep beep aack beep beep!’.
- We obtain the following classifier.

Better Classifier 1

- ‘Aack’: 0.98 points
- ‘Beep’: 0.95 points
- Bias: -4.01 points.

The perceptron trick

Sentence 2 (happy): ‘Aack.’

- This is again a misclassified point, and let's see why.
- The sentence has a happy label.
- However, its score is , $1.1+1.0 - 4 = -3$,as it contains the word ‘aack’ once and the word ‘beep’ zero times.
- Since the score is negative, the classifier predicts that the sentence is sad.

The perceptron trick

Perceptron trick (pseudocode):

Input:

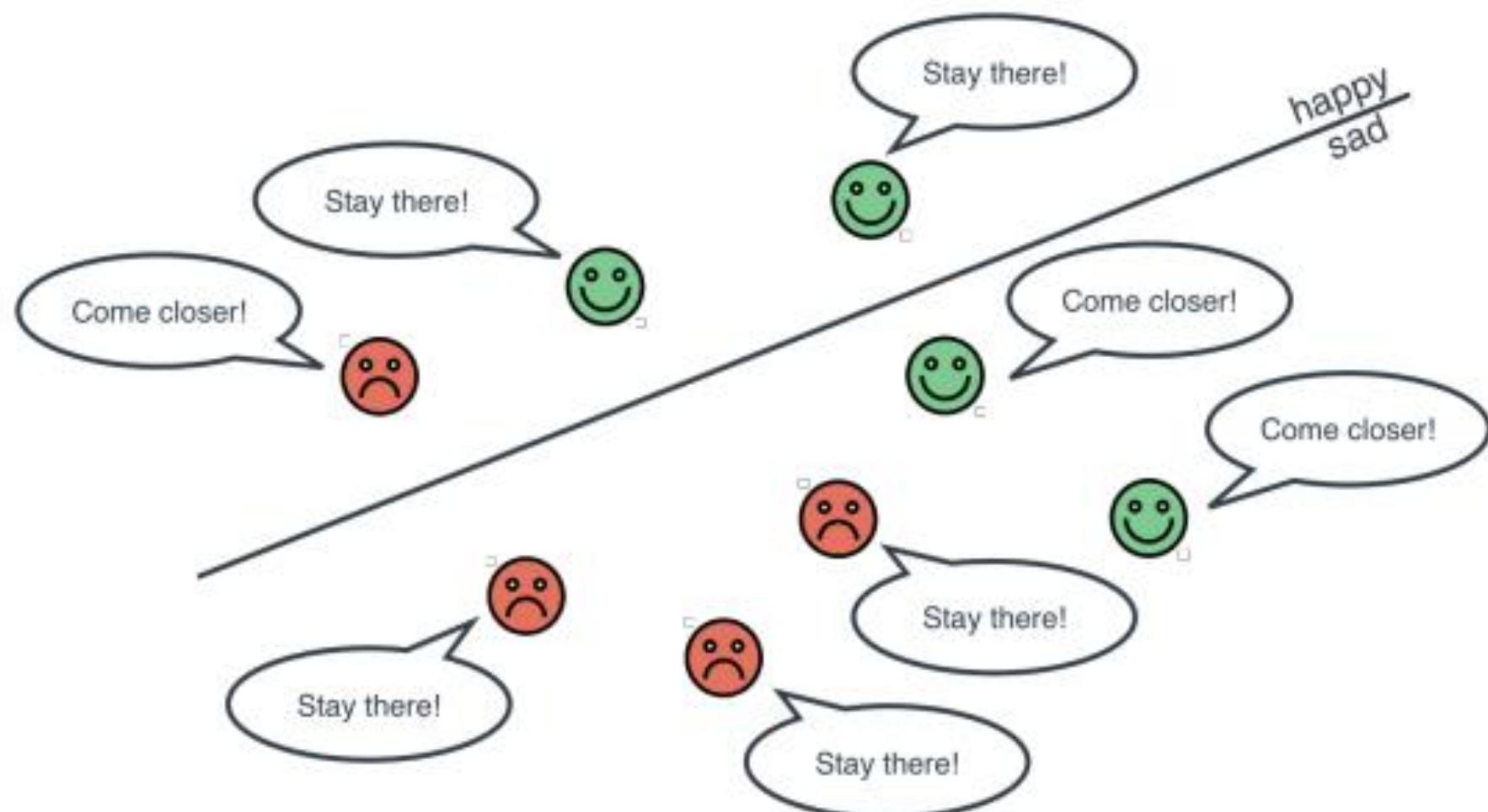
- A classifier with the following scores:
 - Score of 'aack': a.
 - Score of 'beep': b.
 - Bias: c.
- A point with coordinates (x, y) (where x is the number of appearances of the word 'aack', and y of the word 'beep').
- A learning rate η .

Procedure:

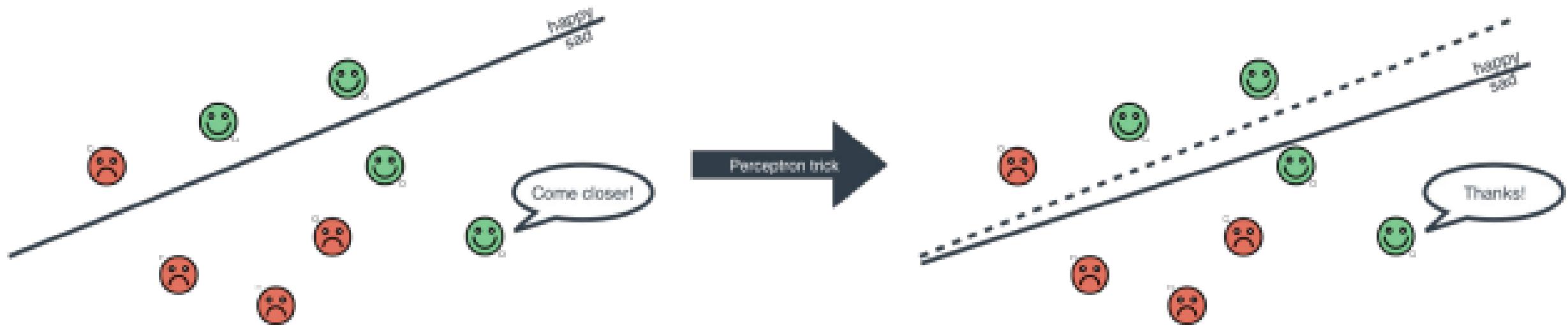
- If the point is correctly classified:
 - Output the exact same classifier.
- Else, if the point has a negative label but is misclassified as positive:
 - Output a classifier with the following scores:
 - Score of 'aack': $a - \eta x$.
 - Score of beep: $b - \eta y$.
 - Bias: $c - \eta$.
- Else, if the point has a negative label but is misclassified as positive:
 - Output a classifier with the following scores:
 - Score of 'aack': $a + \eta x$.
 - Score of beep: $b + \eta y$.
 - Bias: $c + \eta$.

Repeating the perceptron trick many times: The perceptron algorithm

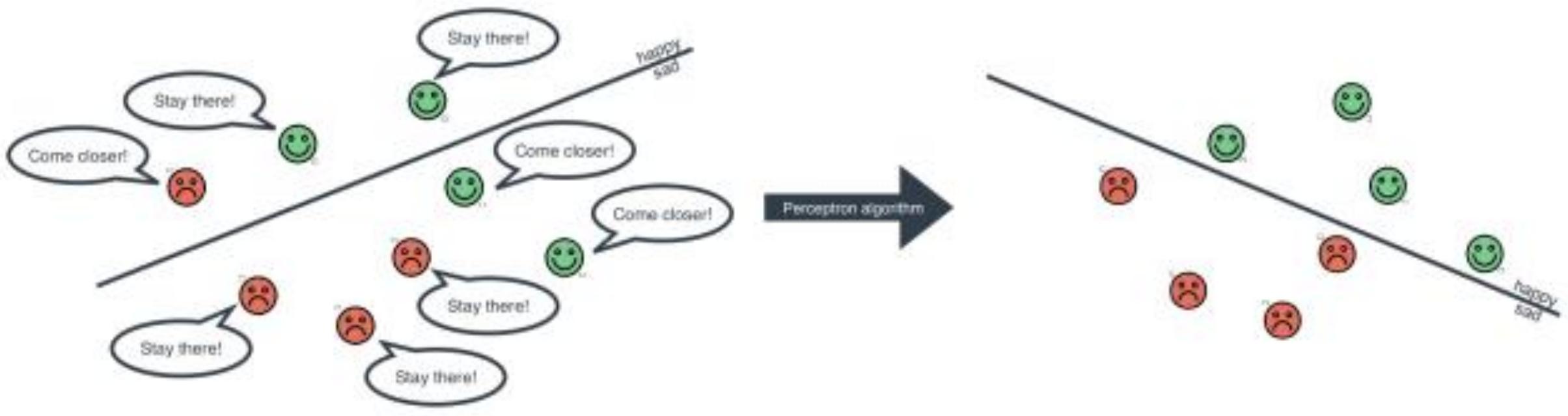
Repeating the perceptron trick many times: The perceptron algorithm



Repeating the perceptron trick many times: The perceptron algorithm



Repeating the perceptron trick many times: The perceptron algorithm



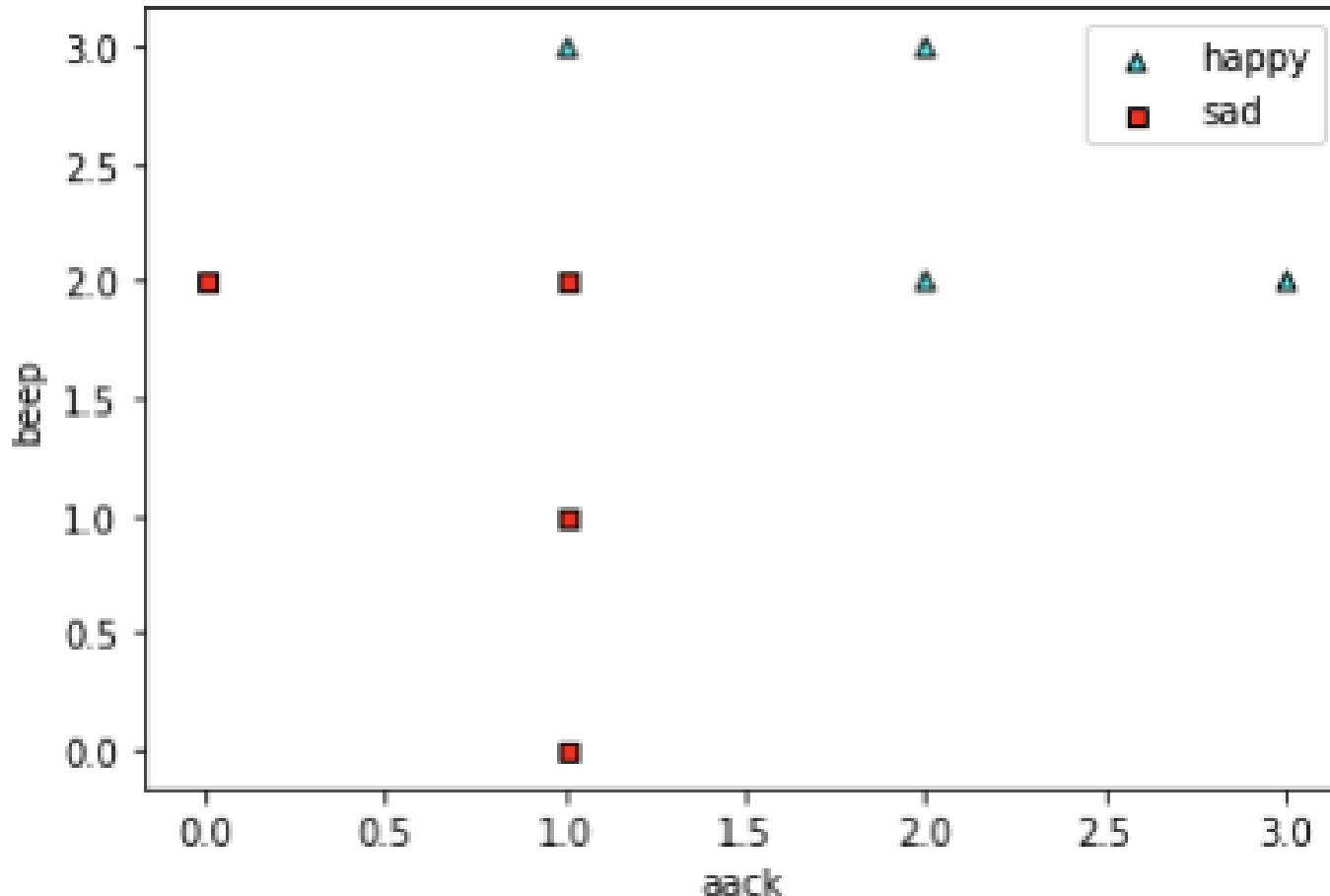
Coding the perceptron algorithm

Aack	Beep	Happy/Sad
1	0	0
0	2	0
1	1	0
1	2	0
1	3	1
2	2	1
3	2	1
2	3	1

Coding the perceptron algorithm

```
import pandas as pd  
X = pd.DataFrame([[1,0],[0,2],[1,1],[1,2],[1,3],[2,2],[3,2],[2,3]])  
y = pd.Series([0,0,0,0,1,1,1,1])
```

Coding the perceptron algorithm



Coding the perceptron trick

The score for a particular sentence is, just as before, the sum of the number of times each word appears (x_i) times the weight of that word (w_i), plus the bias (b).

- Score = $w_1x_1 + w_2x_2 + \dots + w_nx_n + b = \sum_{i=1}^n w_i x_i + b$.

This formula above is known as the *dot product*. We say that the dot product between the vectors (x_1, x_2, \dots, x_n) and (w_1, w_2, \dots, w_n) is the sum of the products of their entries, namely $w_1x_1 + w_2x_2 + \dots + w_nx_n$ (if the word vector is not familiar, just think of it as an ordered set of numbers).

Coding the perceptron trick

```
def score(weights, bias, features):  
    return features.dot(weights) + bias #A
```

```
def prediction(weights, bias, features):  
    if score(weights, bias, features) >= 0: #B  
        return 1  
    else:  
        return 0
```

Coding the perceptron trick

```
def error(weights, bias, features, label):  
    pred = prediction(weights, bias, features)  
    if pred == label:          #C  
        return 0  
    else:                      #D  
        return np.abs(score(weights, bias, features))
```

Coding the perceptron trick

- We now write a function that adds the errors of all the points in our dataset.

```
def total_error(weights, bias, X, y):  
    total_error = 0  
    for i in range(len(X)):          #E  
        total_error += error(weights, bias, X.loc[i], y[i])  
    return total_error
```

Coding the perceptron trick

```
def perceptron_trick(weights, bias, features, label, learning_rate = 0.01):
    pred = prediction(weights, bias, features)
    if pred == label:                      #F
        return weights, bias
    else:
        if label==1 and pred==0:      #G
            for i in range(len(weights)):
                weights[i] += features[i]*learning_rate
            bias += learning_rate
        elif label==0 and pred==1: #H
            for i in range(len(weights)):
                weights[i] -= features[i]*learning_rate
            bias -= learning_rate
    return weights, bias
```

Coding the perceptron trick

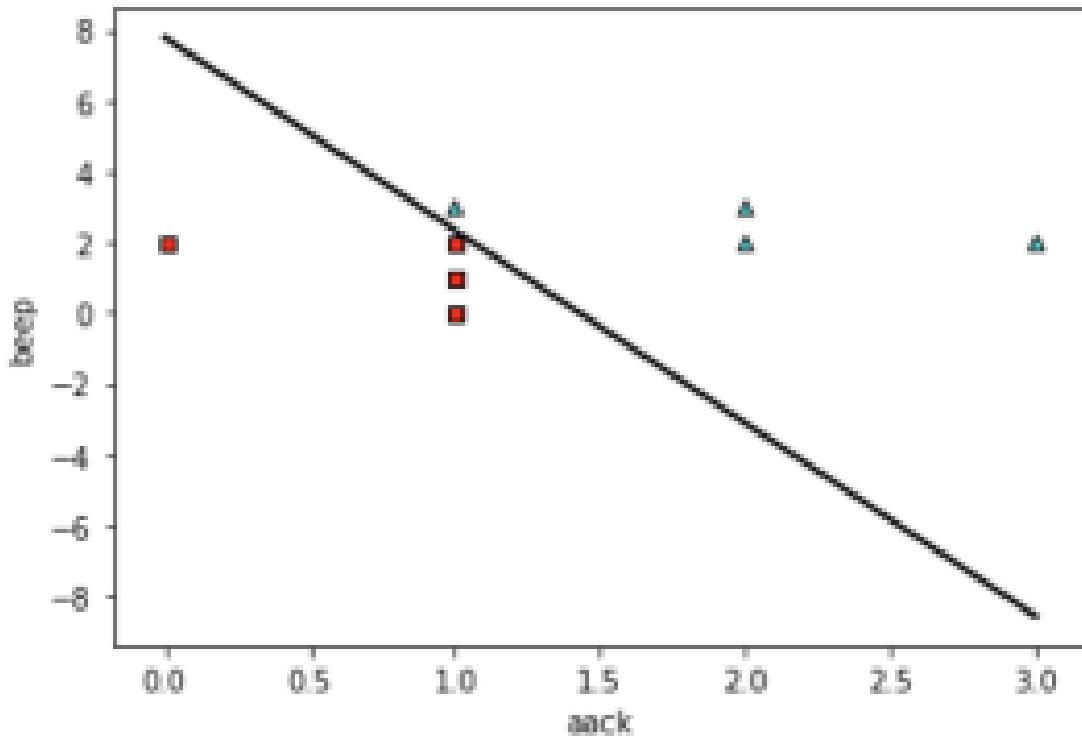
```
def perceptron_trick_clever(weights, bias, features, label, learning_rate = 0.01):
    pred = prediction(weights, bias, features)
    for i in range(len(weights)):
        weights[i] += (label-pred)*features[i]*learning_rate
        bias += (label-pred)*learning_rate
    return weights, bias
```

Coding the perceptron trick

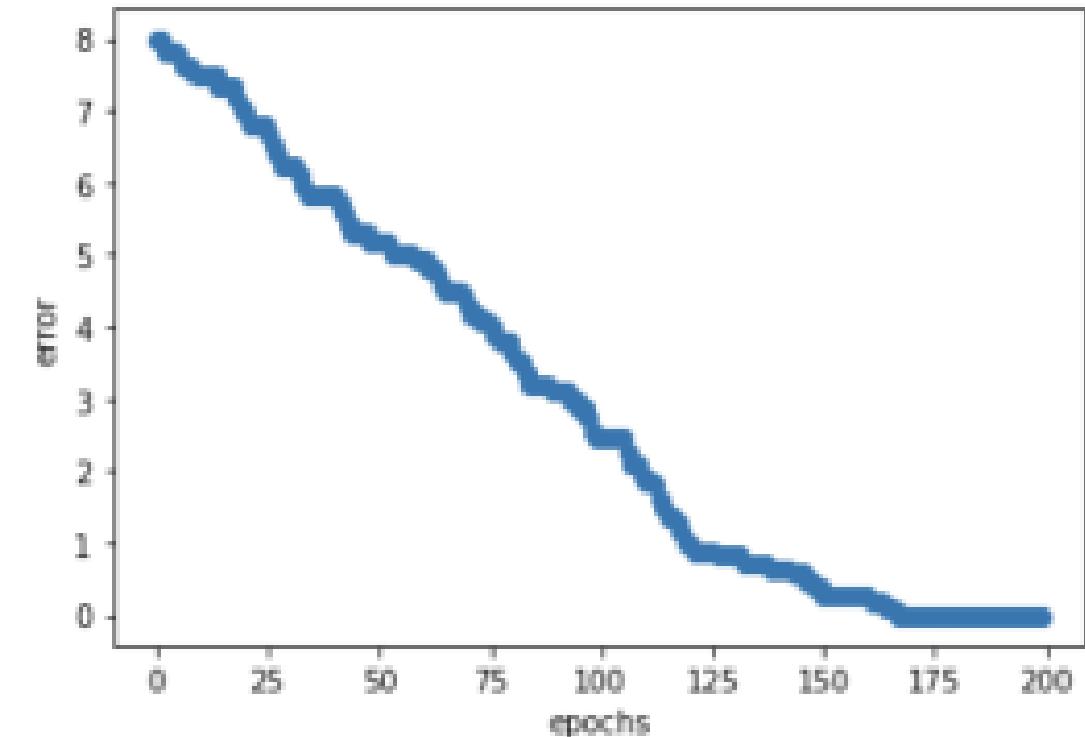
```
def perceptron_algorithm(X, y, learning_rate = 0.01, epochs = 200):
    weights = [1.0 for i in range(len(X.loc[0]))]      #I
    bias = 0.0
    errors = []                                         #J
    for i in range(epochs):                            #K
        errors.append(total_error(weights, bias, X, y)) #L
        j = random.randint(0, len(features)-1)         #M
        weights, bias = perceptron_trick(weights, bias, X.loc[j], y[j]) #N
    return weights, bias, errors
```

Coding the perceptron trick

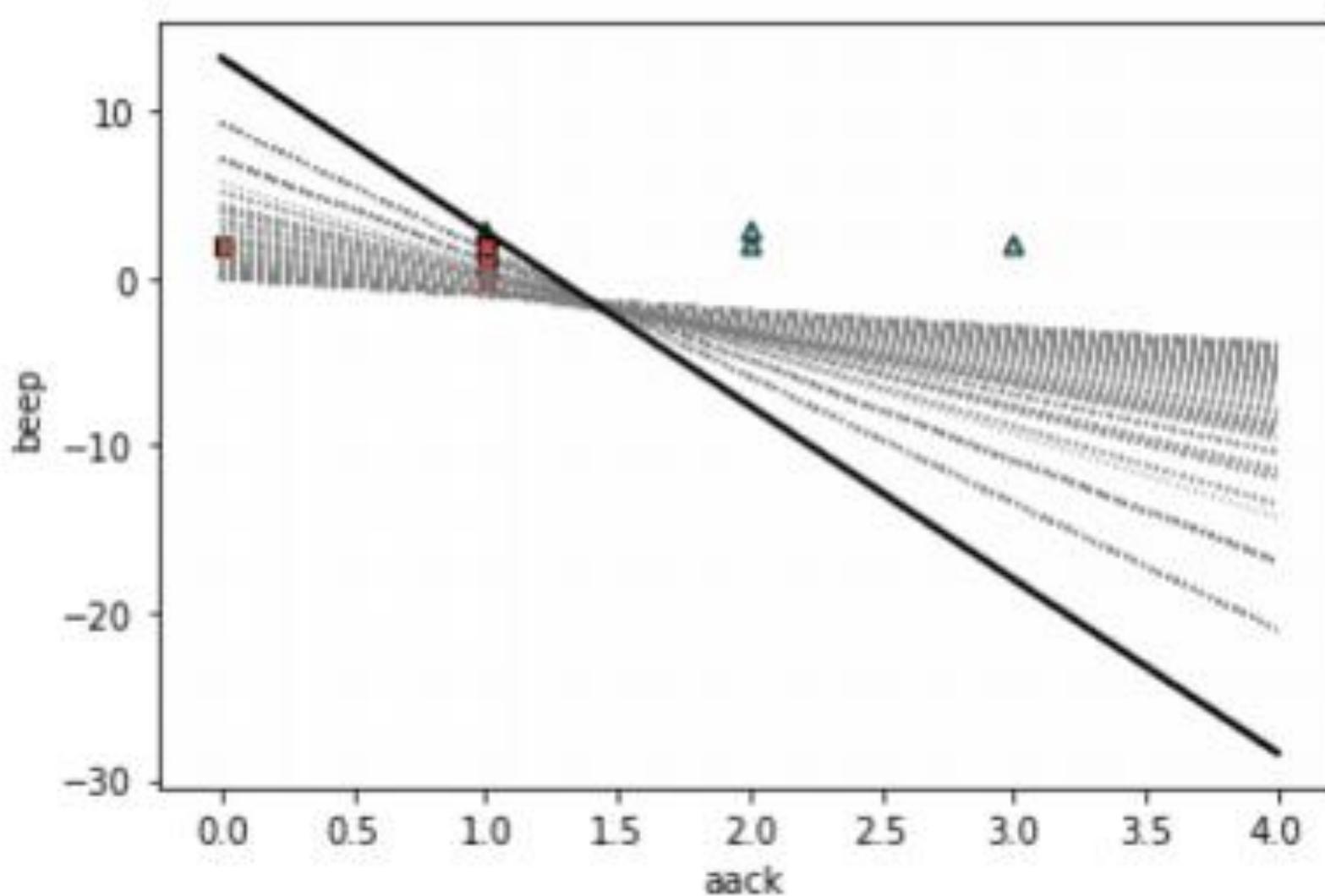
Classifier



Error



Coding the perceptron trick



Applications of the perceptron algorithm

We can also use other features, for example,

- length of the email,
- size of attachments,
- number of senders,
- if any of our contacts is a sender (categorical variable),
- and many others

Some drawbacks of the perceptron algorithm, which will be addressed very soon!

- Perceptrons are very useful for answering questions such as yes/no, which is good when our labels are one of two categories such as happy/sad, spam/ham, or dog/no-dog.
- However, what if we have more categories? Say we have an image dataset with dogs, cats, and birds.
- What would we do? In general, the approach is to use three perceptrons, one for deciding if the image is a dog or not a dog, one for cat, and one for bird.

Summary

- Classification is a very important part of machine learning.
- It is similar to regression in that it consists of training an algorithm with labelled data, and using it to make predictions on future (unlabeled) data, except this time the predictions are categories, such as yes/no, spam/ham, etc.

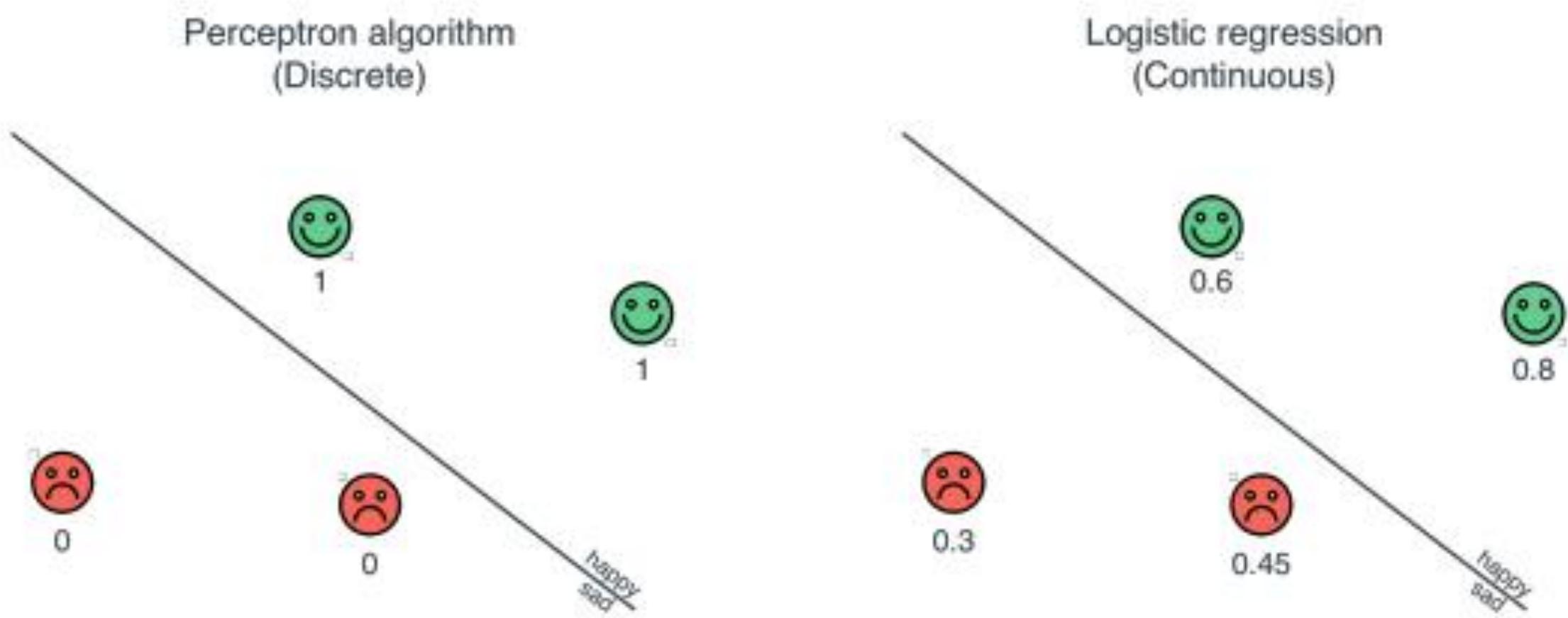
6: A continuous approach to splitting points: Logistic regression

A continuous approach to splitting points: Logistic regression

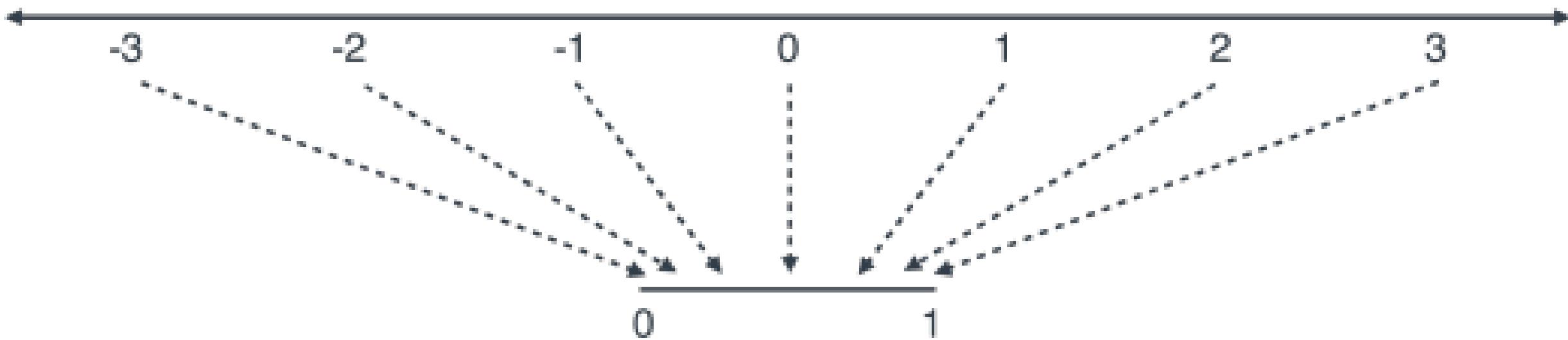
This lesson covers

- The difference between hard assignments and soft assignments.
- Activation functions such as the step function vs the sigmoid function.
- Discrete perceptron vs continuous perceptron.
- The logistic regression algorithm for classifying data.
- Coding the logistic regression algorithm in Python.
- Using the SoftMax function to build classifiers for more than two classes

Logistic Regression (or continuous perceptrons)



A probability approach to classification - The sigmoid function



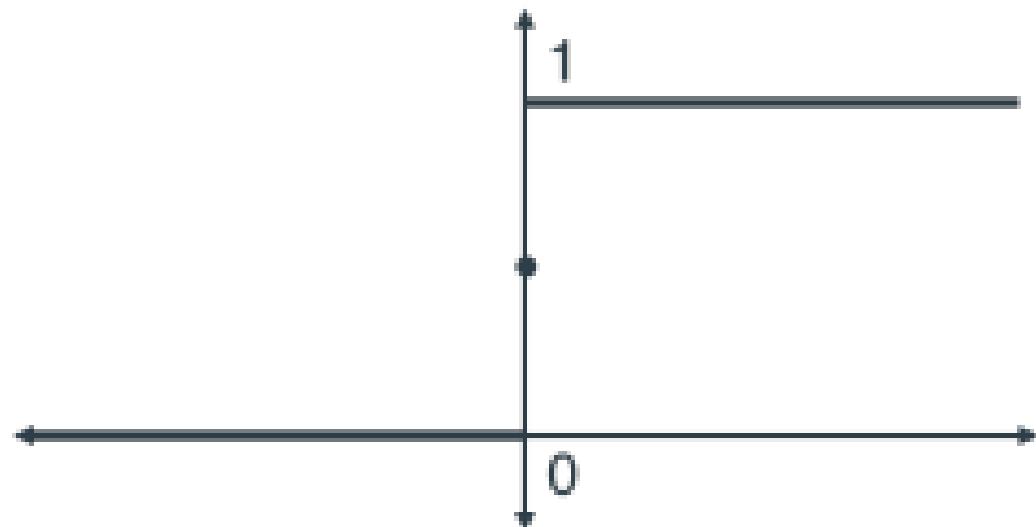
A probability approach to classification - The sigmoid function

- Many functions can help us here, and in this case, we'll use one called the sigmoid, denoted with the Greek letter σ . The formula for the sigmoid is the following:

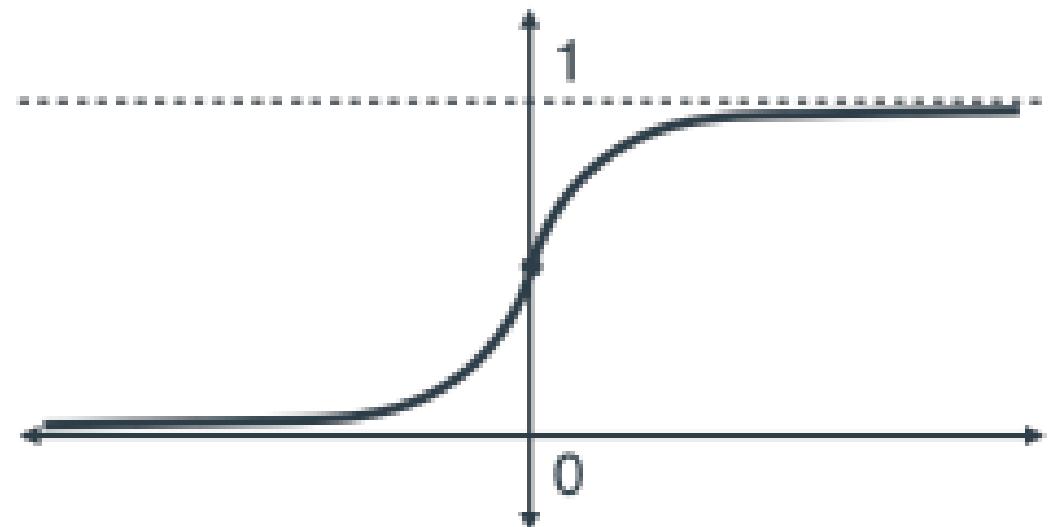
$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

A probability approach to classification - The sigmoid function

Step function
(discrete)



Sigmoid function
(continuous)



A probability approach to classification - The sigmoid function

- The code for the sigmoid function in Python is very simple; we make use of the numpy function exp, which takes any real number as an input, and returns e to the power of that number.

```
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

A probability approach to classification - The sigmoid function

x	$\sigma(x)$
-5	0.0067
-1	0.269
0	0.5
1	0.731
5	0.9933

A probability approach to classification - The sigmoid function

- Now we are ready to define a prediction.
- The prediction is obtained by applying the sigmoid function to the score, and it returns a number between 0 and 1 which, as I mentioned before, can be interpreted in our example as the probability that the sentence is happy.
- Here's the code for the prediction function

```
def lr_prediction(weights, bias, features):  
    return sigmoid(score(weights, bias, features))
```

The error functions - Absolute, square, and log loss

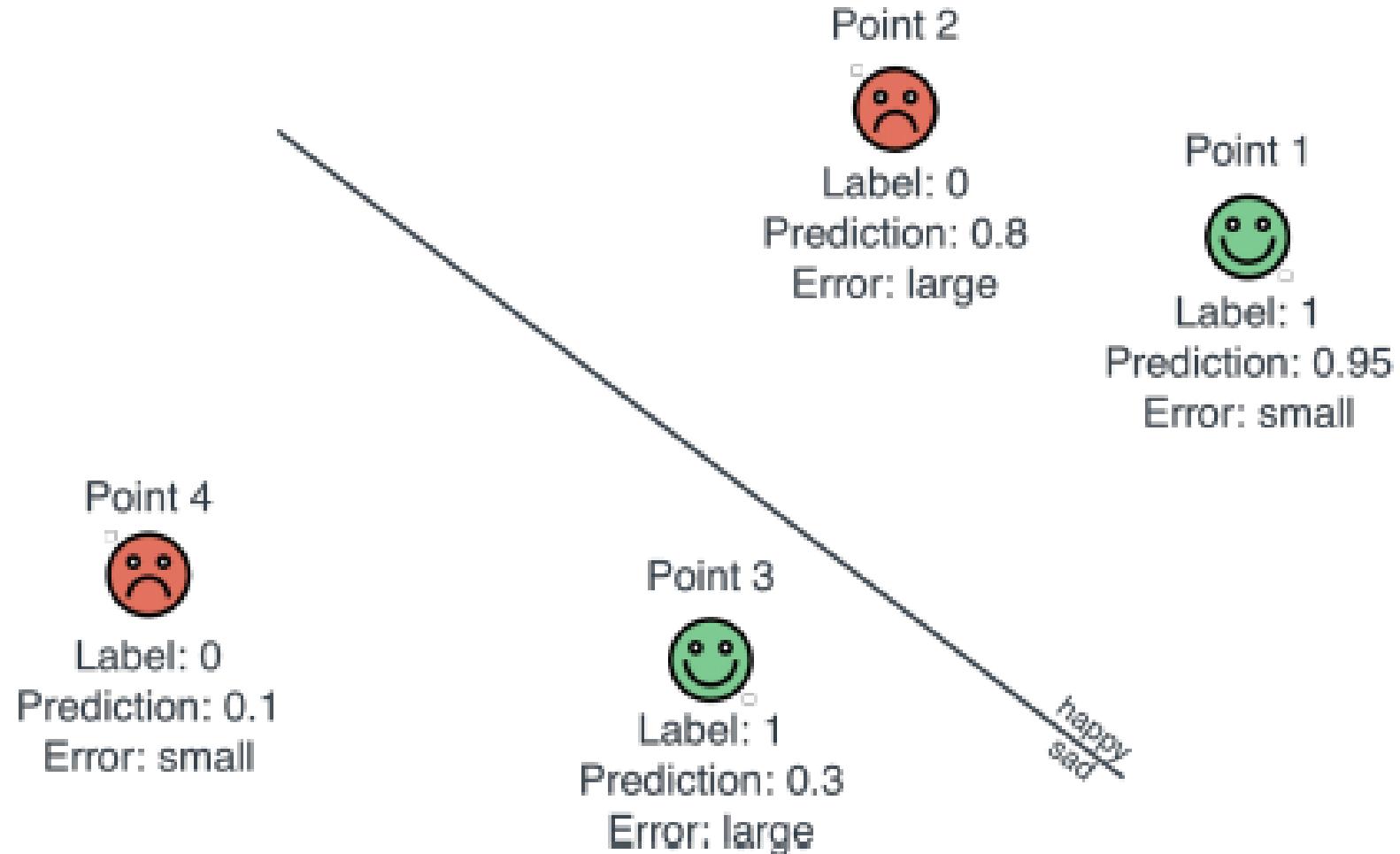
The classifier then makes a prediction between 0 and 1 for each point in the plane, as follows:

- The points on the line are given a prediction of 0.5.
- Points that are up and to the right of the line are given predictions higher than 0.5, and the farther a point is from the line in that direction, the closer its prediction is to 1.
- Points that are down and to the left of the line are given predictions lower than 0.5, and the farther a point is from the line in that direction, the closer its prediction is to 0.

The error functions - Absolute, square, and log loss

Point	True label	Predicate label	Error
1	1(Happy)	0.95	Should be small
2	0(Sad)	0.8	Should be large
3	1(Happy)	0.3	Should be large
4	0(Sad)	0.1	Should be small

The error functions - Absolute, square, and log loss



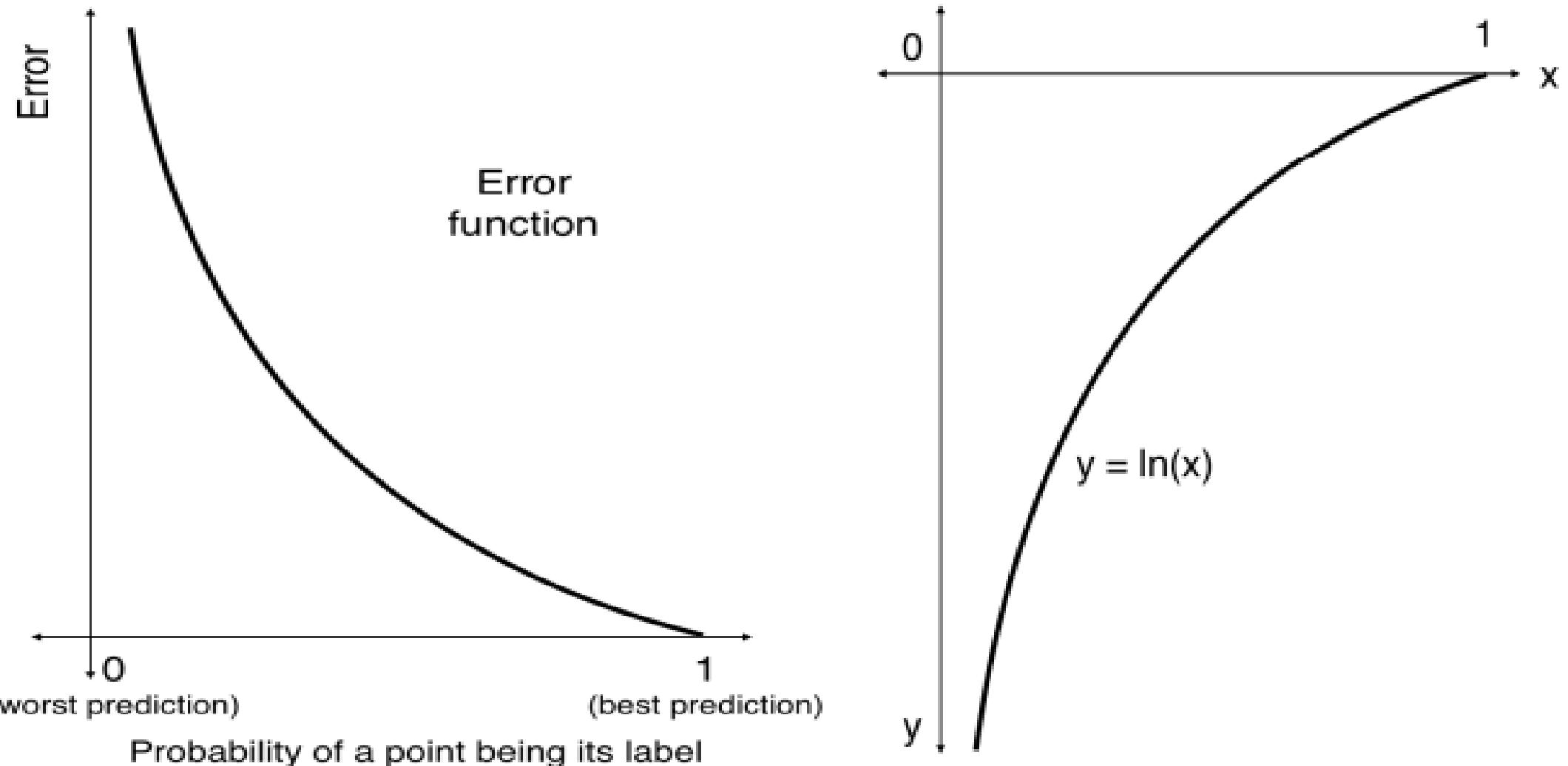
The error functions - Absolute, square, and log loss

Point	True label	Predicate label	Absolute Error	Square Error
1	1(Happy)	0.95	0.05	0.0025
2	0(Sad)	0.8	0.8	0.64
3	1(Happy)	0.3	0.7	0.49
4	0(Sad)	0.1	0.1	0.01

The error functions - Absolute, square, and log loss

- Point 1:
 - Label = 1 (happy)
 - Prediction = 0.95
 - Probability of being happy: 0.95
- Point 2:
 - Label = 1 (sad)
 - Prediction = 0.8
 - Probability of being sad: $1 - 0.8 = 0.2$
- Point 3:
 - Label = 1 (happy)
 - Prediction = 0.3
 - Probability of being happy: 0.3
- Point 4:
 - Label = 1 (sad)
 - Prediction = 0.1
 - Probability of being happy: $1 - 0.1 = 0.9$

The error functions - Absolute, square, and log loss



The error functions - Absolute, square, and log loss

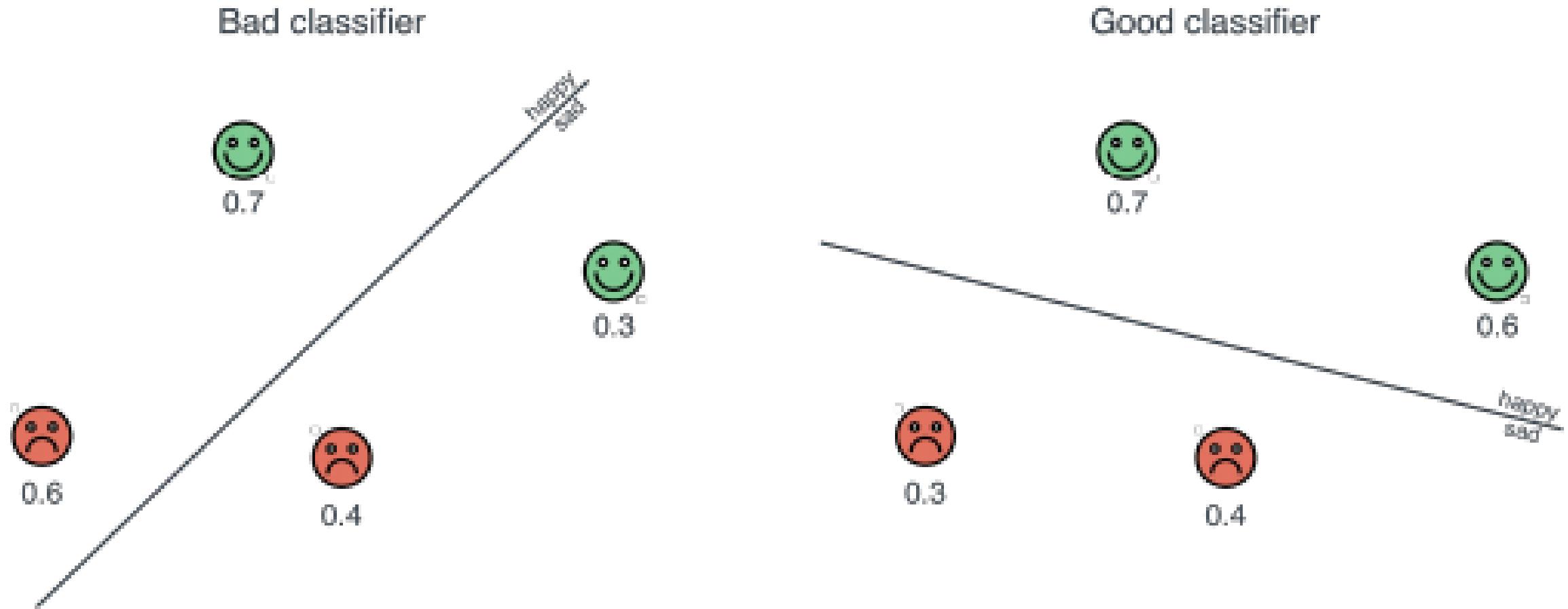
Point	True label	Predicate label	Probability of being its label	Log loss
1	1(Happy)	0.95	0.95	$-\ln(0.95)=0.051$
2	0(Sad)	0.8	0.2	$-\ln(0.2)=1.609$
3	1(Happy)	0.3	0.3	$-\ln(0.4)=0.916$
4	0(Sad)	0.1	0.9	$-\ln(0.9)=0.105$

The error functions - Absolute, square, and log loss

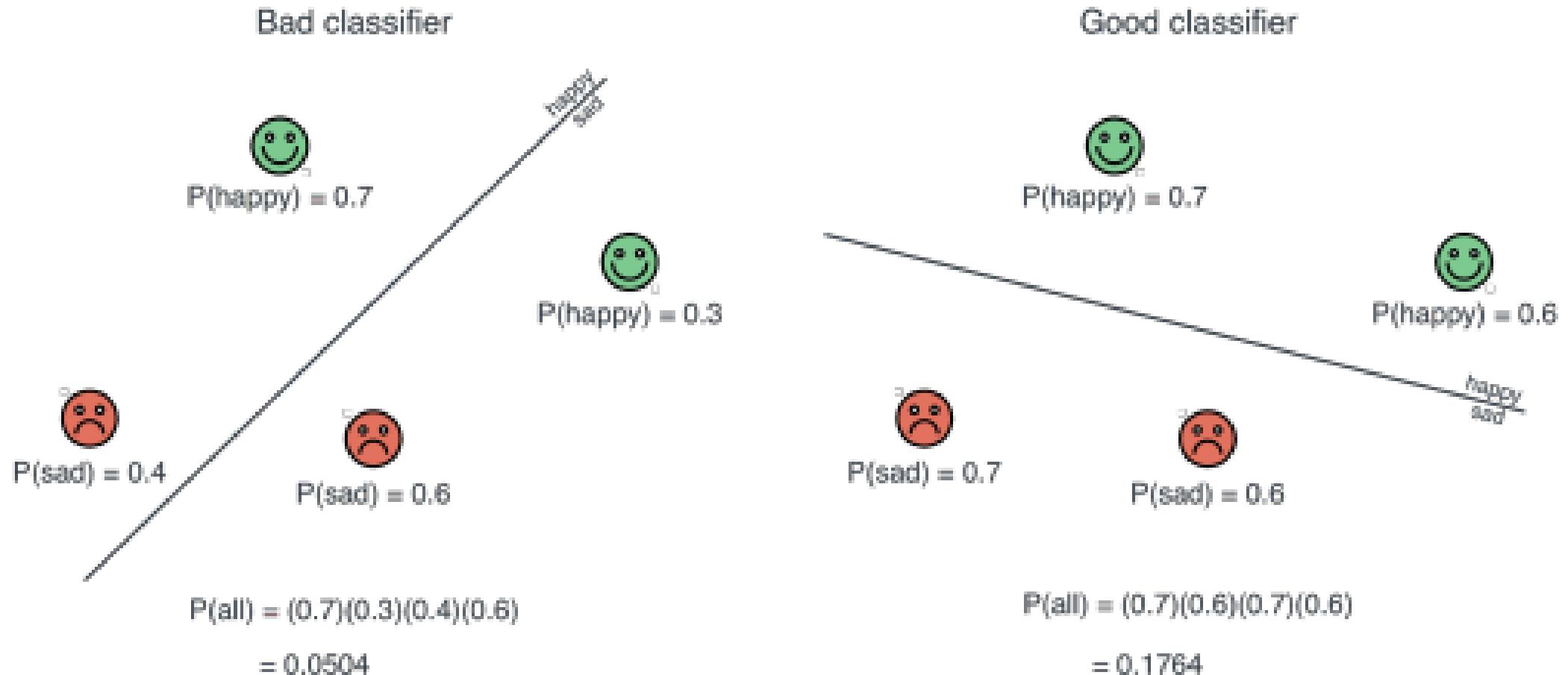
COMPARING THE ERROR FUNCTIONS

Point	True label	Predicate label	Absolute Error	Square Error	Log loss
1	1(Happy)	0.95	0.05	0.0025	0.051
2	0(Sad)	0.8	0.8	0.64	1.609
3	1(Happy)	0.3	0.7	0.49	0.916
4	0(Sad)	0.1	0.1	0.01	0.105

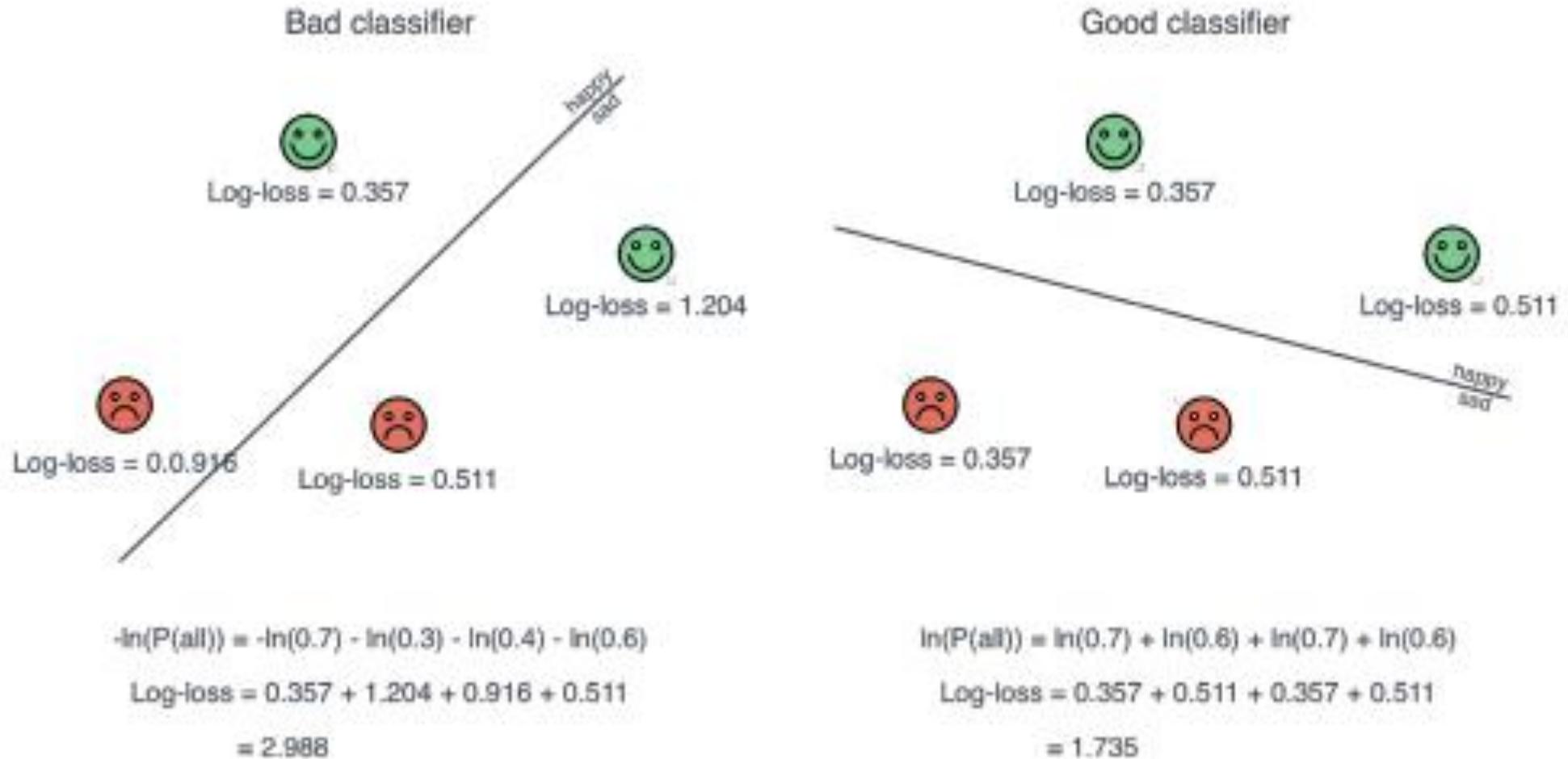
More on the log loss error function



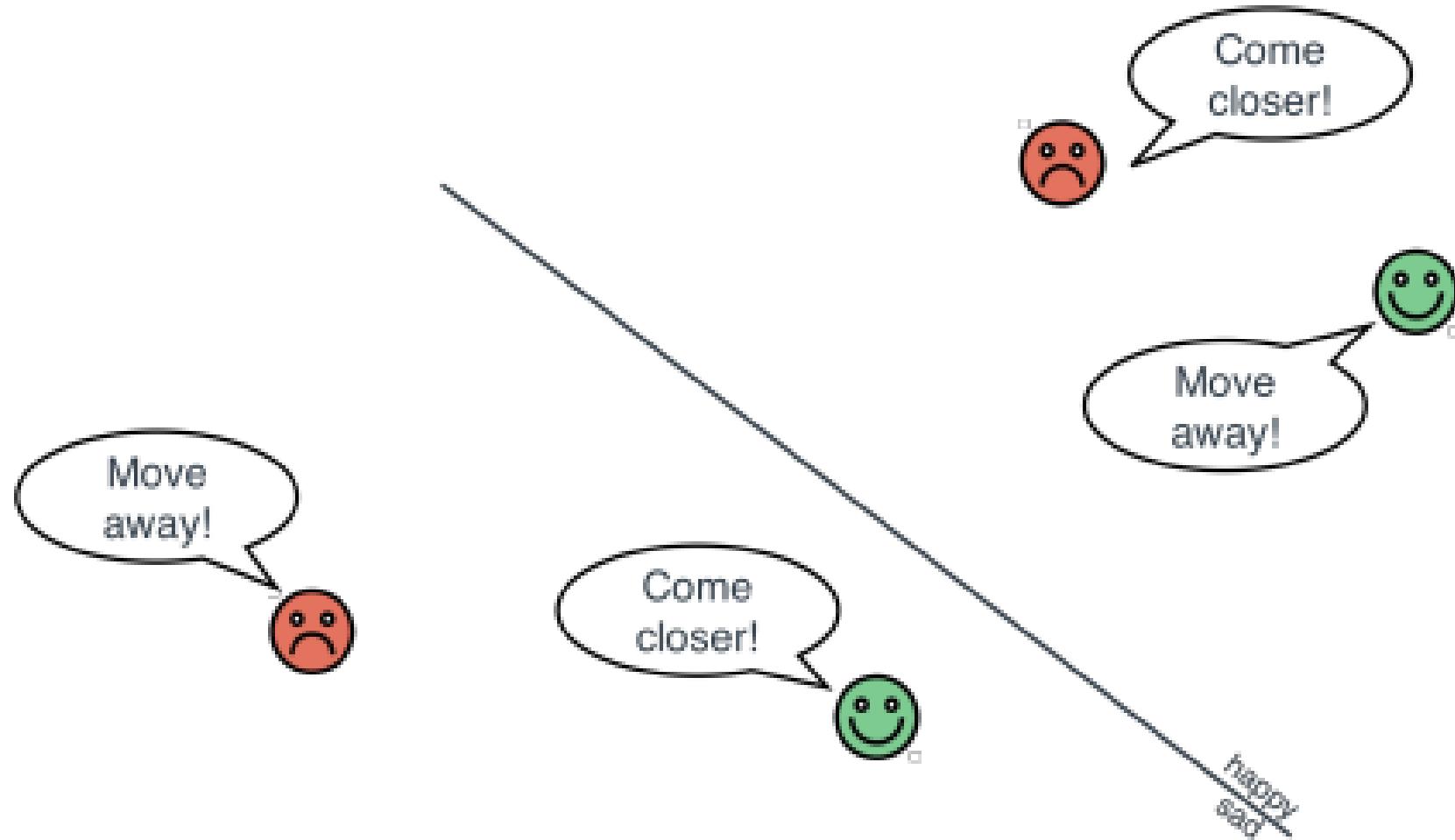
More on the log loss error function



More on the log loss error function



Reducing the log loss error: The logistic regression trick



An example with a discrete perceptron & continuous perceptron

- Classifier (scores):
 - ‘Aack’: 1 pt.
 - ‘Beep’: 1 pts.
 - Bias: -4 pts.
- Sentence 1:
 - Words: ‘Aack beep beep beep aack beep beep!’
 - Label: Sad

An example with a discrete perceptron & continuous perceptron

- The new classifier would assign a score of

$$0.98*2 + 0.96*4 - 4.01 = 1.79.$$

An example with a discrete perceptron & continuous perceptron

Using the logistic regression algorithm to improve our discrete perceptron classifier

- What would happen with a continuous perceptron classifier?
- This one would apply the sigmoid function to the score, and use that as a prediction.
- The sigmoid function gives us

$$\text{Prediction} = \sigma(2) = 0.881.$$

A second example with a discrete perceptron and a continuous perceptron

- One of the main benefits of the logistic regression algorithm over the perceptron algorithm is that if a point is correctly classified
- The perceptron algorithm leaves it alone and doesn't improve, while the logistic regression algorithm still uses the information of that point to improve the model.

A second example with a discrete perceptron and a continuous perceptron

- The score and prediction that this new classifier gives to our point is:

$$\text{Score} = 1.002*2 + 1.005*4 - 3.999 = 2.025$$

$$\text{Prediction} = \sigma(2.025) = 0.883$$

Moving the line to fit the points - The logistic regression algorithm

Logistic regression trick (pseudocode):

Input:

- A classifier with the following scores:
 - Score of ‘aack’: a.
 - Score of ‘beep’: b.
 - Bias: c.
- A point with coordinates (x_1, x_2) (where x_1 is the number of appearances of the word ‘aack’, and x_2 of the word ‘beep’).
- A learning rate η

Moving the line to fit the points - The logistic regression algorithm

Logistic regression algorithm (pseudocode):

Input:

- A dataset of points, where every point has a positive or negative label.
- A number of epochs, n .
- A learning rate η

Coding the logistic regression algorithm

The weights include the bias, which corresponds to no word, it simply gets added to the score of every sentence.

- Features: x_1, x_2, \dots, x_n
- Label: y
- Weights: w_1, w_2, \dots, w_n
- Bias: b

Coding the logistic regression algorithm

The score for a particular sentence is the sigmoid of the sum of the weight of each word (w_i) times the number of times that appears (x_i), plus the bias (b) (which we called the dot product).

- Prediction: $\hat{y} = \sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n + b) = \sigma(\sum_{i=1}^n w_i x_i + b)$.

Coding the logistic regression algorithm

```
def sigmoid(x):  
    return np.exp(x)/(1+np.exp(x))  
  
def lr_prediction(weights, bias, features):  
    return sigmoid(score(weights, bias, features))
```

- Now that we have the prediction, we can proceed to the log loss.
- We need to do a bit of math to figure out the formula for the log loss.

Coding the logistic regression algorithm

- Let's code that formula.

```
def log_loss(weights, bias, features, label):  
    pred = prediction(weights, bias, features)  
    return label*np.log(prediction) + (1-label)*np.log(1-prediction)
```

- We need the log loss over the whole dataset, so we can add over all the data points.

```
def total_log_loss(weights, bias, X, y):  
    total_error = 0  
    for i in range(len(X)):  
        total_error += log_loss(weights, bias, X.loc[i], y[i])  
    return total_error
```

Coding the logistic regression algorithm

```
def lr_trick(weights, bias, features, label, learning_rate = 0.01):
    pred = lr_prediction(weights, bias, features)
    for i in range(len(weights)):
        weights[i] += (label-pred)*features[i]*learning_rate
        bias += (label-pred)*learning_rate
    return weights, bias

def lr_algorithm(features, labels, learning_rate = 0.01, epochs = 200):
    weights = [1.0 for i in range(len(features.loc[0]))]
    bias = 0.0
    errors = []
    for i in range(epochs):
        draw_line(weights[0], weights[1], bias, color='grey', linewidth=1.0,
                  linestyle='dotted')
        errors.append(total_error(weights, bias, features, labels))
        j = random.randint(0, len(features)-1)
        weights, bias = perceptron_trick(weights, bias, features.loc[j], labels[j])
    draw_line(weights[0], weights[1], bias)
```

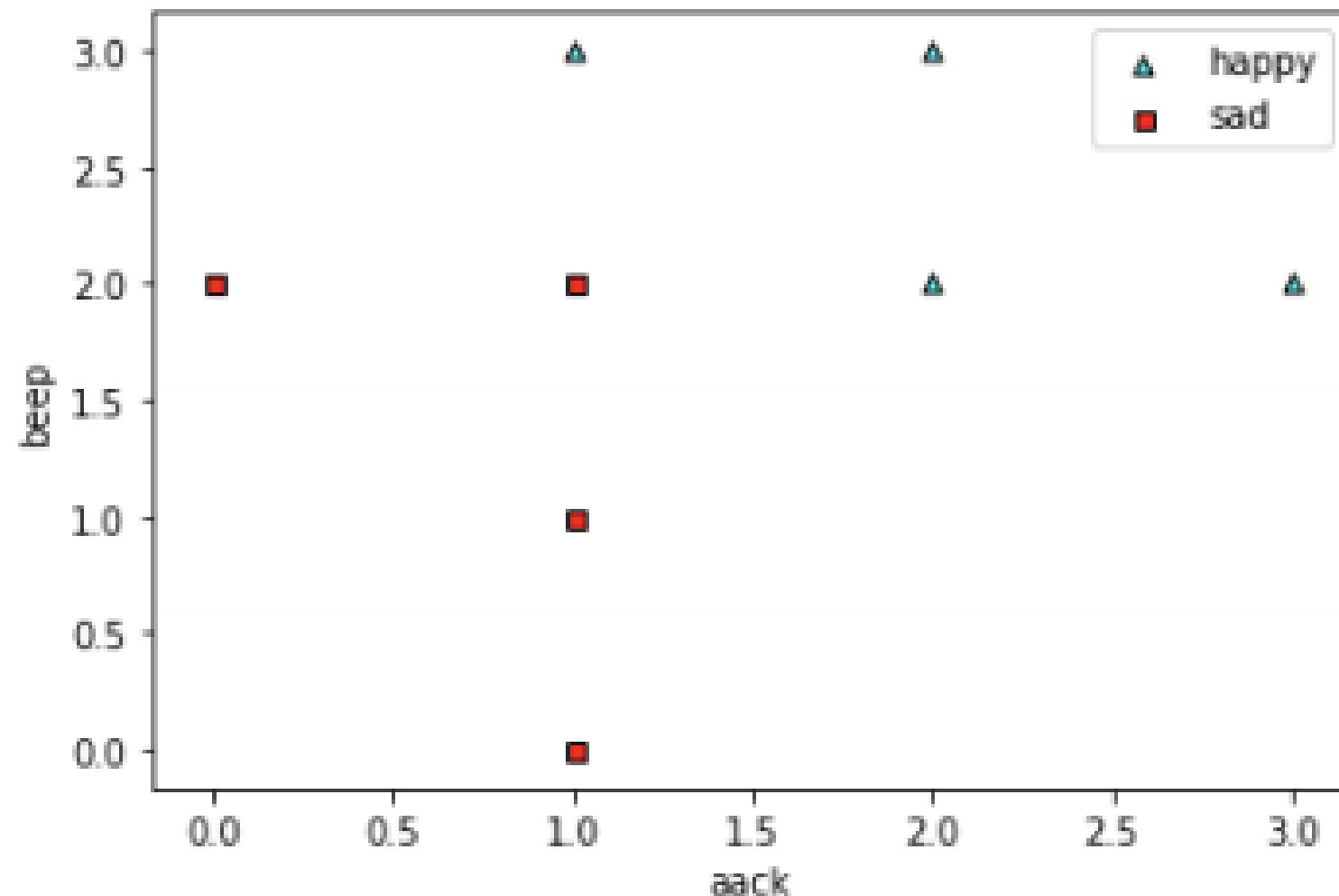
Coding the logistic regression algorithm

```
plot_points(features, labels)
plt.show()
plt.scatter(range(epochs), errors)
plt.xlabel('epochs')
plt.ylabel('error')
return weights, bias
```

- The code for loading our small dataset is below, and the plot of the dataset is in next figure.

```
import pandas as pd
X = pd.DataFrame([[1,0],[0,2],[1,1],[1,2],[1,3],[2,2],[3,2],[2,3]])
y = pd.Series([0,0,0,0,1,1,1,1])
```

Coding the logistic regression algorithm



Coding the logistic regression algorithm

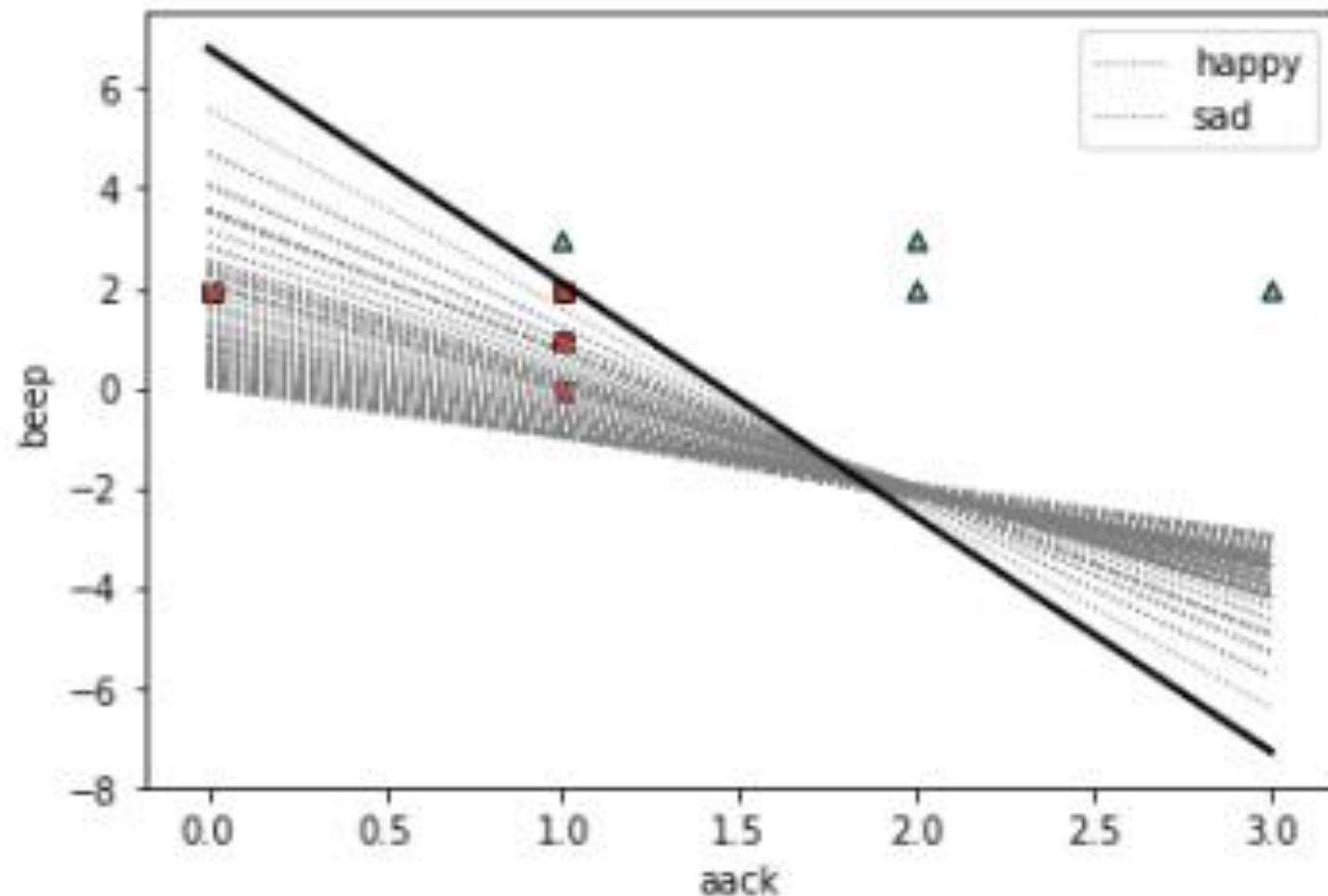
- We run the algorithm to get the classifier.

```
lr_algorithm(features, labels)
([0.4699999999999953, 0.0999999999999937], -0.680000000000004)
```

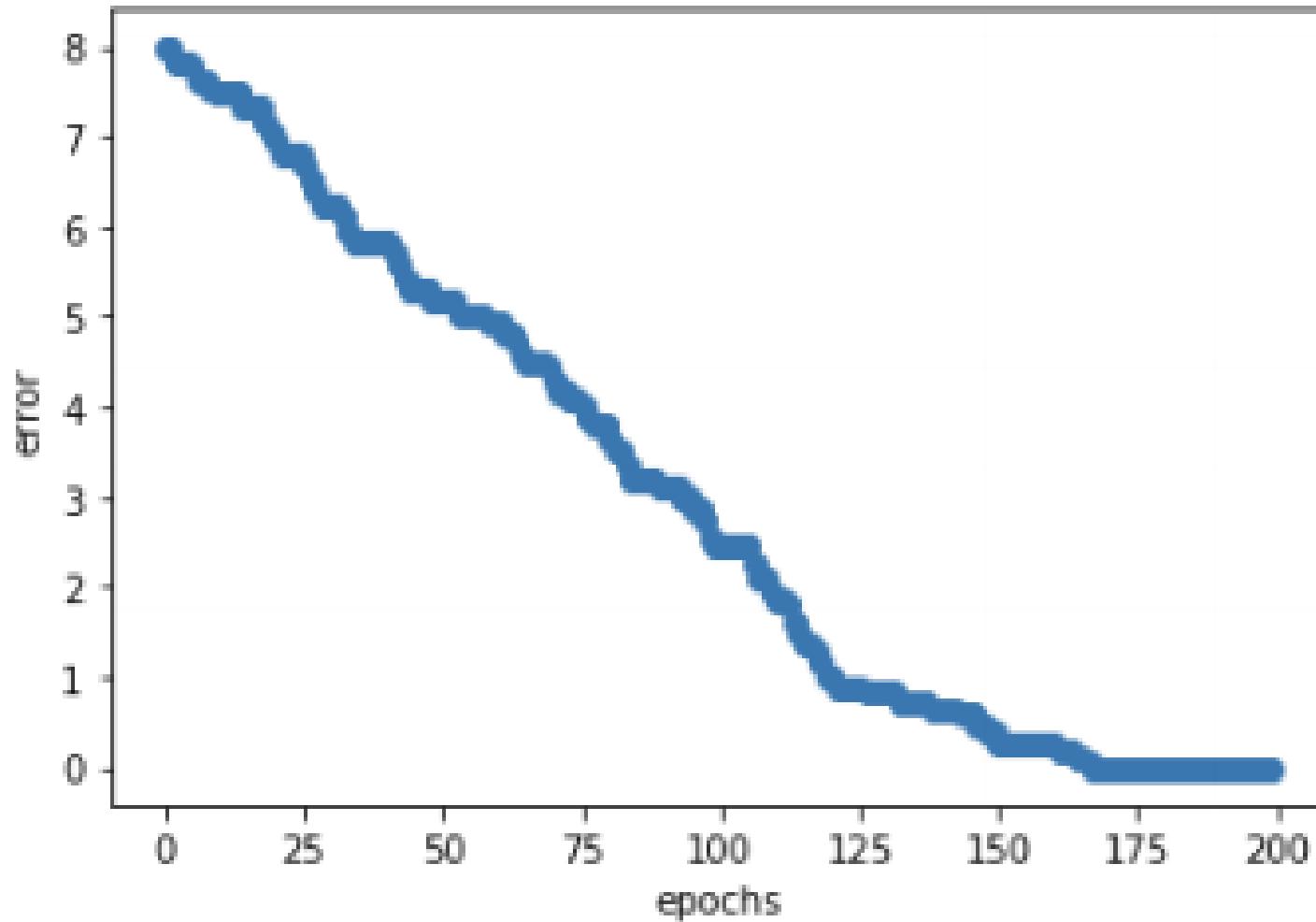
The classifier we obtain has the following weights and biases.

- $w_1 = 4.7$
- $w_2 = 0.1$
- $b = -0.6$

Coding the logistic regression algorithm



Coding the logistic regression algorithm



The logistic regression algorithm in Turi Create

```
import turicreate as tc  
data = tc.SFrame({'X1': X[0], 'X2': X[1], 'y': y})
```

X1	X2	y
1	0	0
0	2	0
1	1	0
1	2	0
1	3	1
2	2	1
3	2	1
2	3	1

[8 rows x 3 columns]

The logistic regression algorithm in Turi Create

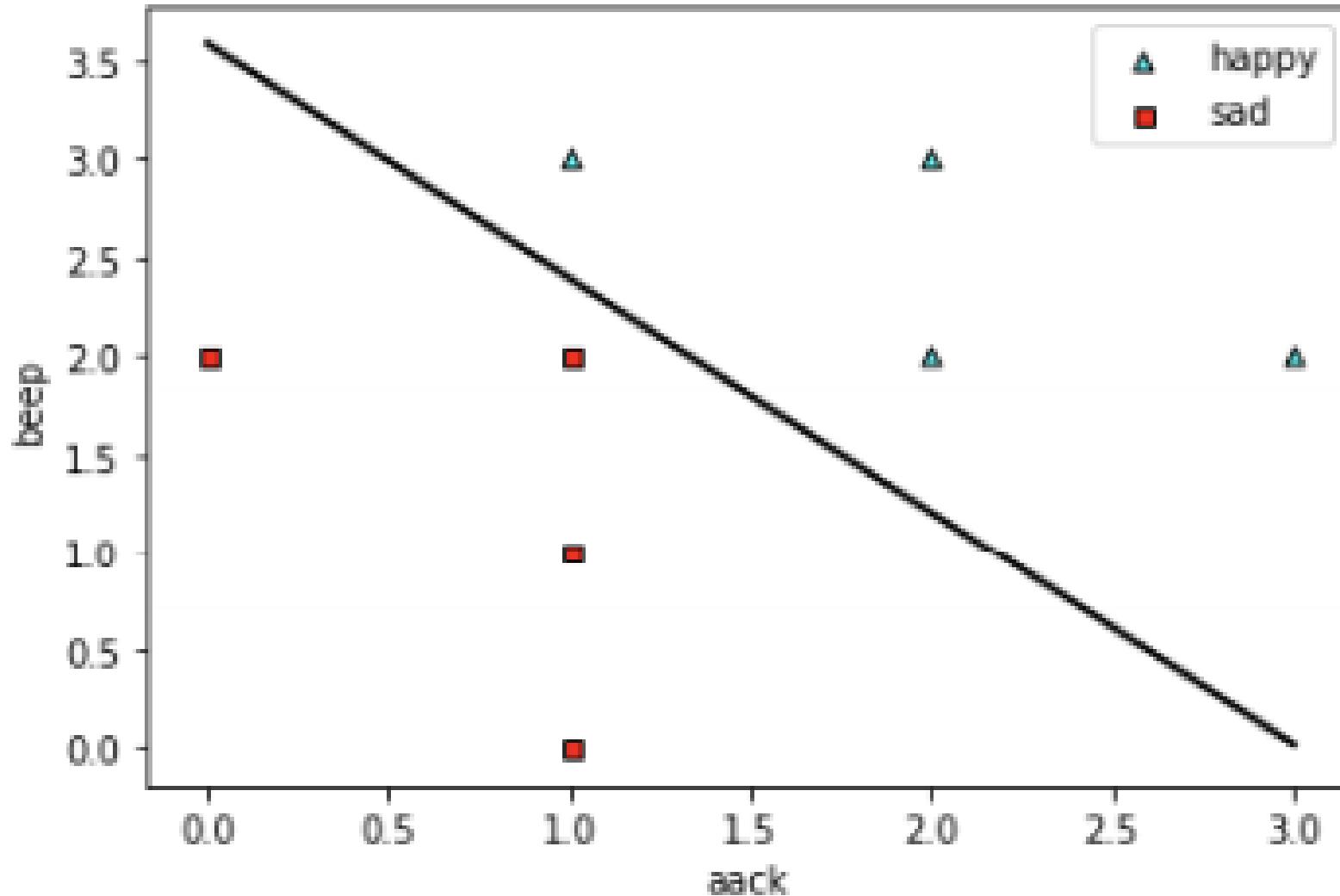
- Now, we train the classifier.

```
classifier = tc.logistic_classifier.create(data,
                                             features = ['X1', 'X2'],
                                             target = 'y',
                                             validation_set=None)
```

- First, let's evaluate the coefficients of this model, with the following command.

`classifier.coefficients`

The logistic regression algorithm in Turi Create



Classifying into multiple classes - The softmax function

As an example, say we have our three classifiers: a dog classifier, a cat classifier, and a bird classifier. For a particular data point, they all output scores in the following way:

- Dog classifier: 4
- Cat classifier: 2
- Bird classifier: 1

Classifying into multiple classes - The softmax function

This works, but what if we have the following numbers:

- Dog classifier: 2
- Cat classifier: 0
- Bird classifier: -2

Classifying into multiple classes - The softmax function

So we'll go with ex. We apply it to all the scores, to get the following.

- Dog classifier: $e^2 = 7.389$
- Cat classifier: $e^0 = 1$
- Bird classifier: $e^{-2} = 0.135$

Classifying into multiple classes - The softmax function

- If we have n classifiers which output the n scores a_1, a_2, \dots, a_n , the probabilities obtained are p_1, p_2, \dots, p_n , where

$$p_i = \frac{e^{a_i}}{e^{a_1} + e^{a_2} + \dots + e^{a_n}}.$$

- This formula is known as the SoftMax formula.

- The log loss is an error function for continuous perceptron's.
- It is calculated separately for every point as the natural logarithm of the probability that the point is classified correctly according to its label.
- The total log loss of a classifier on a dataset is the sum of the log loss at every point.

A blurred background image of a person's hands typing on a laptop keyboard, with the laptop screen showing some colorful icons.

7: How do you measure classification models? Accuracy and its friends

Accuracy and its friends

This lesson covers

- How accuracy can help us evaluate models.
- Types of errors a model can make: False positives and false negatives.
- Putting these errors in a table: The confusion matrix.
- What is recall, and what models require this metric?
- What is precision, and what models require this metric?
- Metrics that combine both recall and precision, such as the F-1 score and F -score

Accuracy - How often is my model correct?

- Accuracy is the most common way to evaluate a classification model, and we should always use it.
- However, sometimes accuracy doesn't paint the entire picture, as we'll see shortly.
- Let's begin by looking at two examples that we'll study throughout this lesson

Two examples of models - Coronavirus and spam email

- In this lesson we'll use our metrics to evaluate several models on two datasets.
- The first dataset is a medical dataset of patients, where some of them have been diagnosed with coronavirus.
- The second dataset is a dataset of emails that have been labeled as spam or not spam.
- This is a dataset that we'll study much more in detail in lesson 8, As we'll further mention in lesson 8, the term we'll use for emails that are not spam is ham.

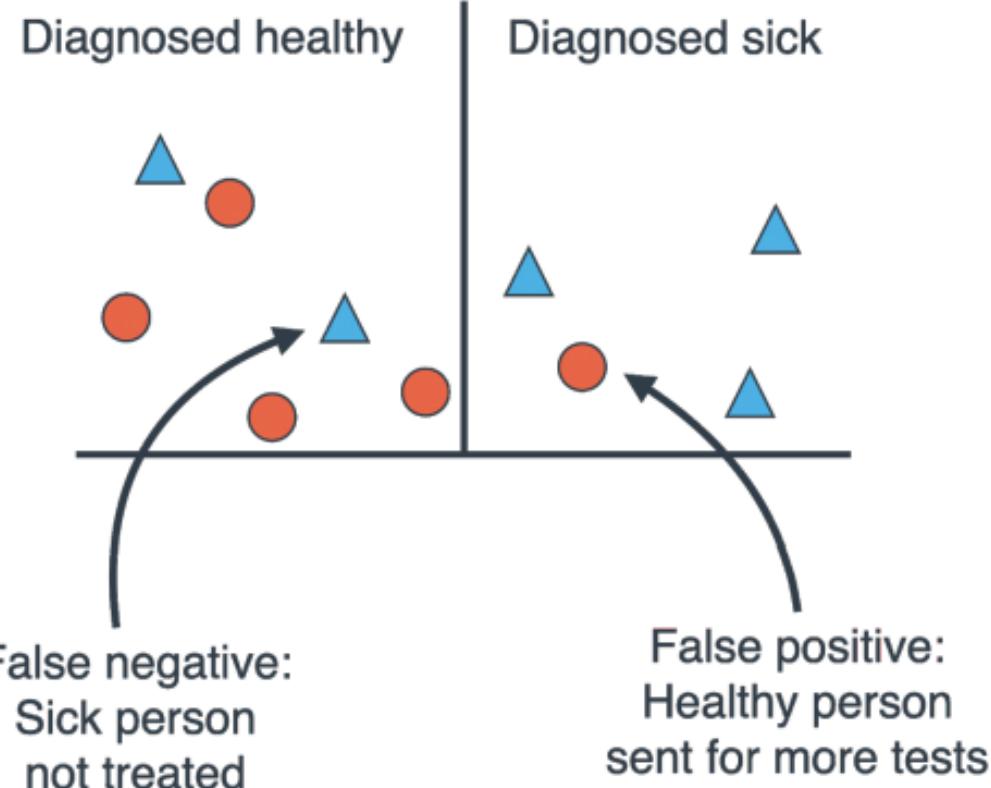
A super effective yet super useless model

- Accuracy is a very useful and widely used metric, but does it paint the whole picture of the model? It doesn't, and I'll illustrate it with an example.
- For now, let's focus on the coronavirus dataset. We'll come back to the email dataset
- Suppose I tell you the following: "I have developed a test for coronavirus that takes 10 seconds to run, doesn't require any examinations, and has an accuracy of 99%!". Would you get excited, or be skeptical?

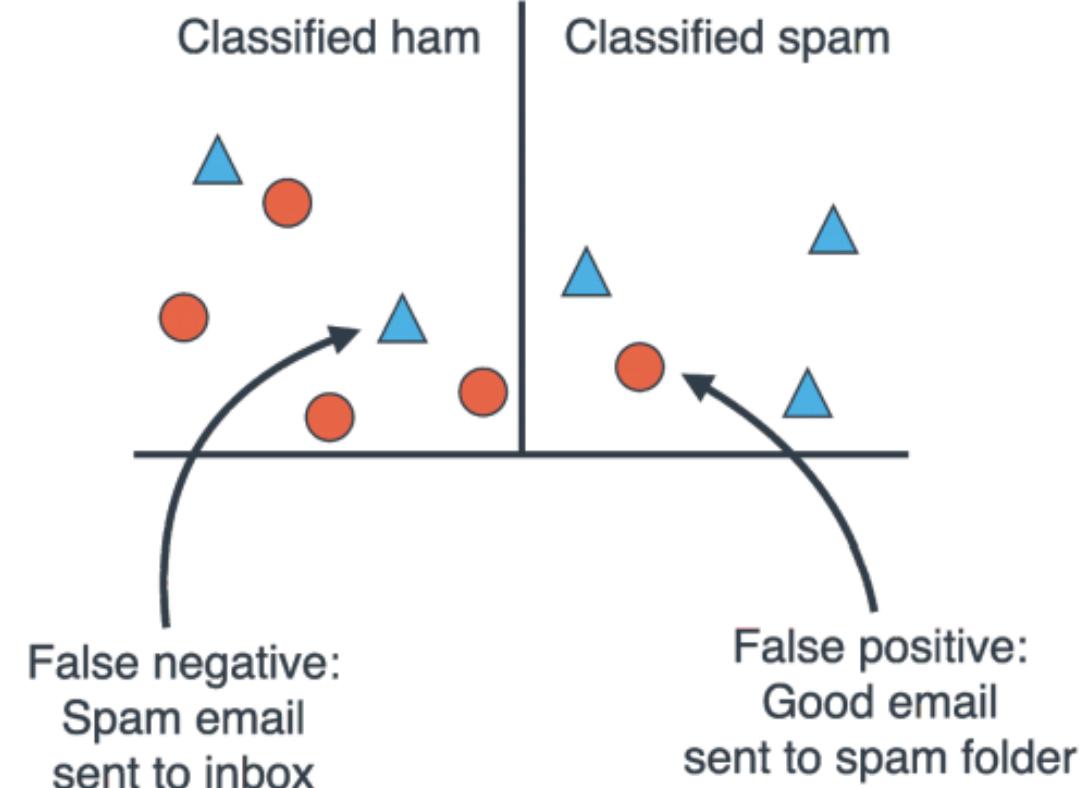
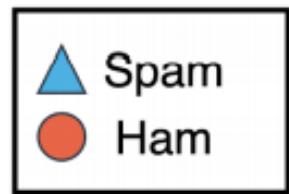
How to fix the accuracy problem?

- The first thing we need to study is types of errors.
- In section, we'll see that there are some errors which are more critical than others.
- Then in sections we'll learn different metrics which are more equipped to catch these critical errors than accuracy

Coronavirus model



Spam model



The confusion matrix

Person's condition	Predicted positive	Predicted negative
Positive (sick)	Number of true positives	Number of false negatives
Negative (healthy)	Number of false positives	Number of true negatives

The confusion matrix

Coronavirus Model 1	Diagnosed sick (predicted positive)	Diagnosed healthy (predicted negative)
Sick (positive)	0 (number of true positives)	10 (number of false negatives)
Healthy (negative)	0 (number of false positives)	990 (number of true negatives)

Among the positive examples, how many did we correctly classify?

- In more general lingo, what recall does is it finds the proportion of correct predictions among the data points with a positive label.
- This is the number of true positives, divided by the total number of positives.
- Coronavirus Model 1 has a total of 0 true positives among 10 positives, so its recall is $0/10 = 0$.
- Another way to put it is as the number of true positives divided by the sum of true positives and false negatives:

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}.$$

how many did we correctly classify?

Coronavirus Model 2	Diagnosed sick	Diagnosed healthy
Sick	8 (true positives)	2 (false negatives)
Healthy	48 (false positives)	942 (true negatives)

Precision - Among the examples we classified as positive, how many did we correctly classify?

- More formally, precision considers only the data points that have been labeled as positive, and among those, how many are true positives.
- Since the data points that are predicted positive are the union of the true positives and the false positives, the formula is the following:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}.$$

how many did we correctly classify?

Spam Model 1	Predicted spam	Predicted ham
Spam	30 (true positives)	10 (false negatives)
Ham	5 (false positives)	55 (true negatives)
Spam Model 2	Predicted spam	Predicted ham
Spam	35 (true positives)	5 (false negatives)
Ham	10 (false positives)	50 (true negatives)

how many did we correctly classify?

Spam Model 1:

- True positives (spam emails deleted) = 30
- False positives (ham emails deleted) = 5
- Precision = $30/35 = 85.7\%$.

Spam Model 2:

- True positives (spam emails deleted) = 35
- False positives (ham emails deleted) = 10
- Precision = $35/45 = 77.7\%$.

CALCULATING THE F-1 SCORE

- Our goal is to find a metric which gives us some number between the recall and the precision.
- The first thing that comes to my mind is the average between recall and precision.
- Would this work? It would, but it's not the one we pick, for one fundamental reason.
- A good model is one that has good recall and good precision.
- If a model has, say, recall of 50% and precision of 100%, the average is 75%.

CALCULATING THE F- SCORE

$$F_{\beta} = \frac{(1+\beta^2) P R}{\beta^2 P + R}$$

Let's analyze this formula carefully by looking at some values for β .

Case 1: $\beta = 1$.

When β is equal to 1, the $F\beta$ -score becomes the following:

$$F_1 = \frac{(1+1^2) P R}{1^2 P + R}$$

CALCULATING THE F- SCORE

Case 2: $\beta = 10$.

When β is equal to 10, the $F\beta$ -score becomes the following:

$$F_{10} = \frac{(1+10^2) PR}{10^2 P + R}.$$

This one is a bit more complex. Notice that we can write this as

$$\frac{101 PR}{100P + R}.$$

CALCULATING THE F- SCORE

Case 3: $\beta = 0.1$.

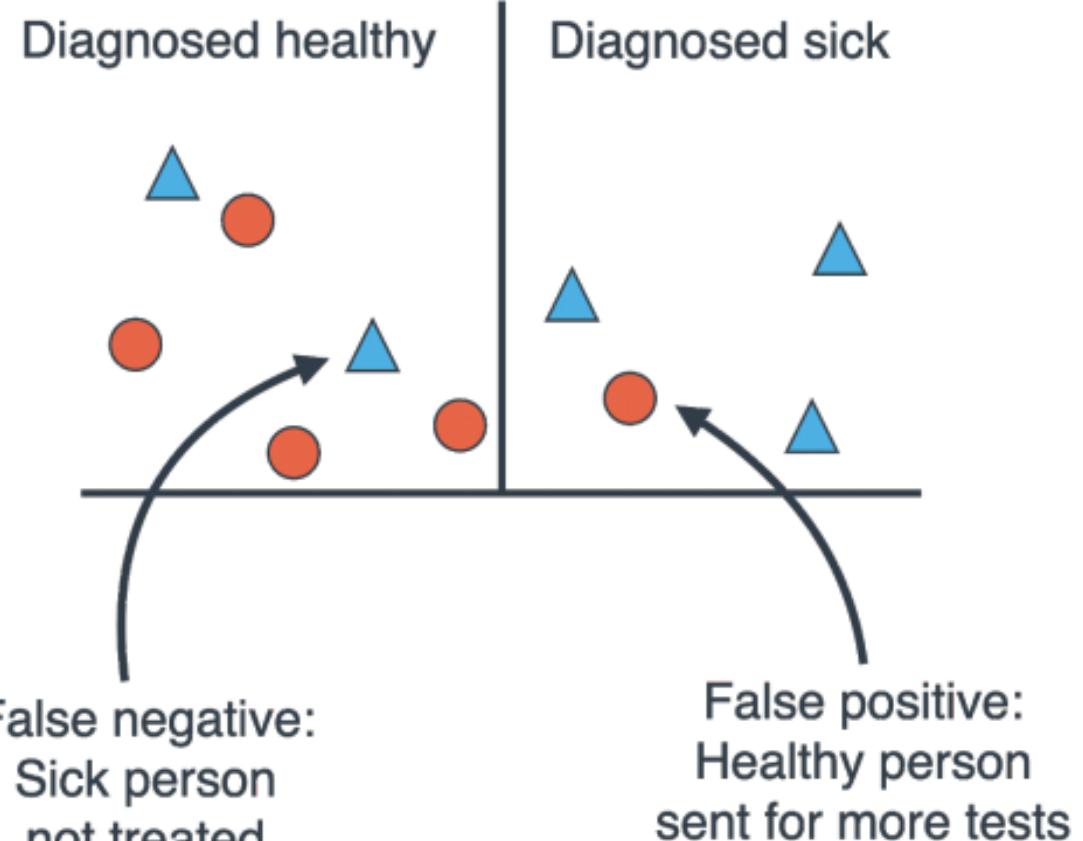
When β is equal to 0.01, the $F\beta$ -score becomes the following:

$$F_{100} = \frac{(1+0.1^2) PR}{0.1^2 P + R}.$$

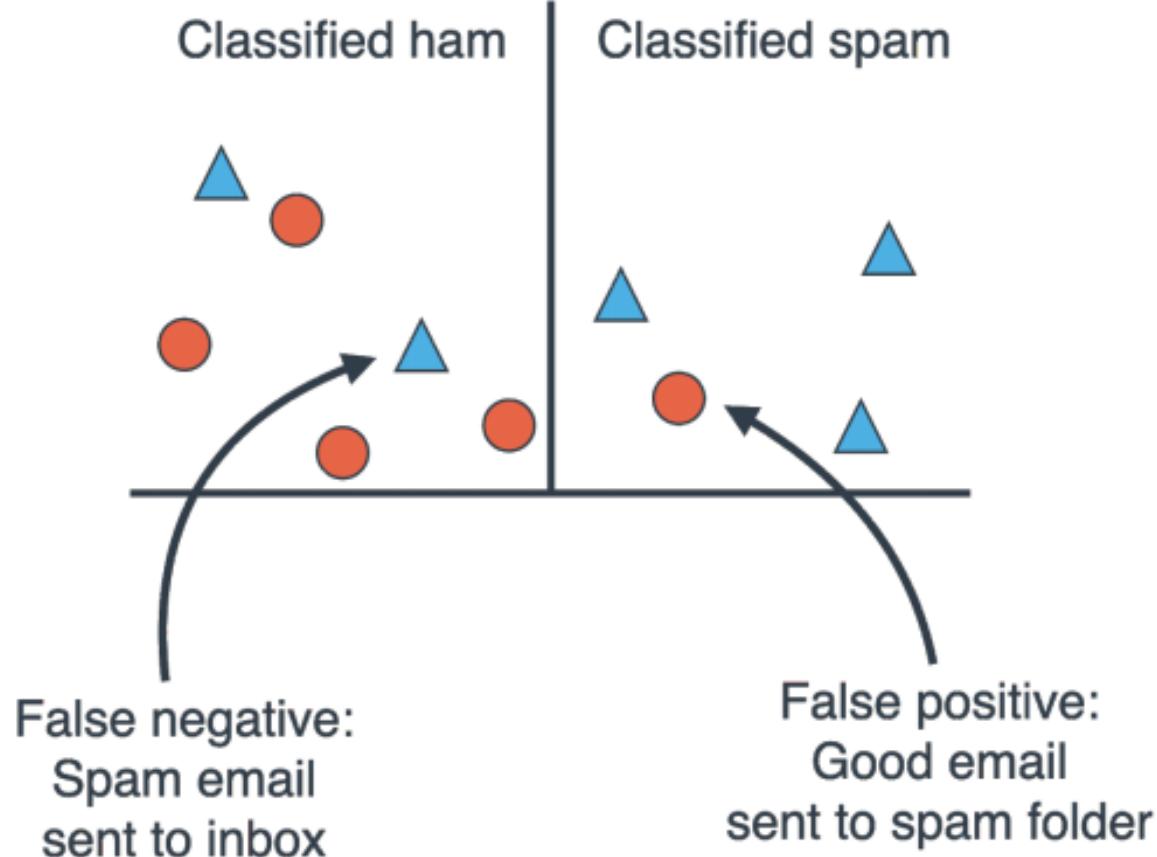
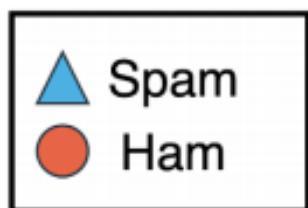
Just like before, we can write this as

$$\frac{1.01 PR}{0.01P + R}.$$

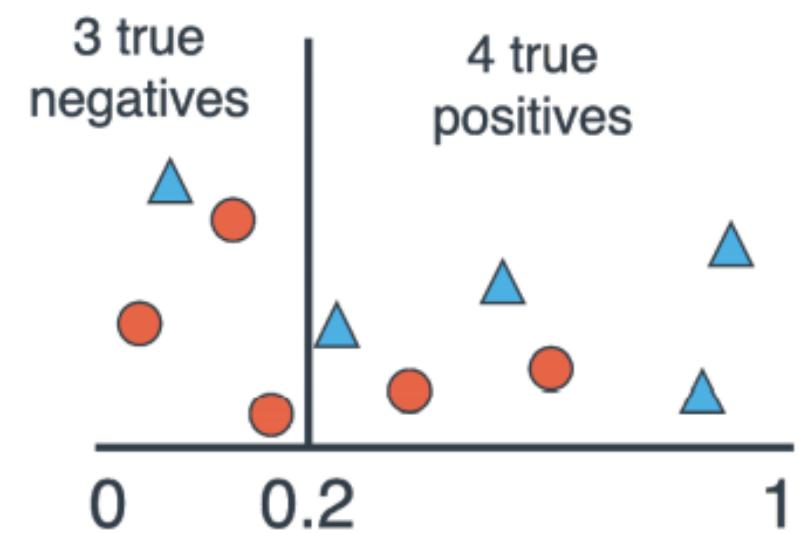
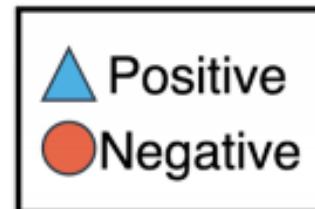
Coronavirus model



Spam model



General model



Threshold = 0.2
Sensitivity = 4/5
Specificity = 3/5



Threshold = 0.5
Sensitivity = 3/5
Specificity = 4/5



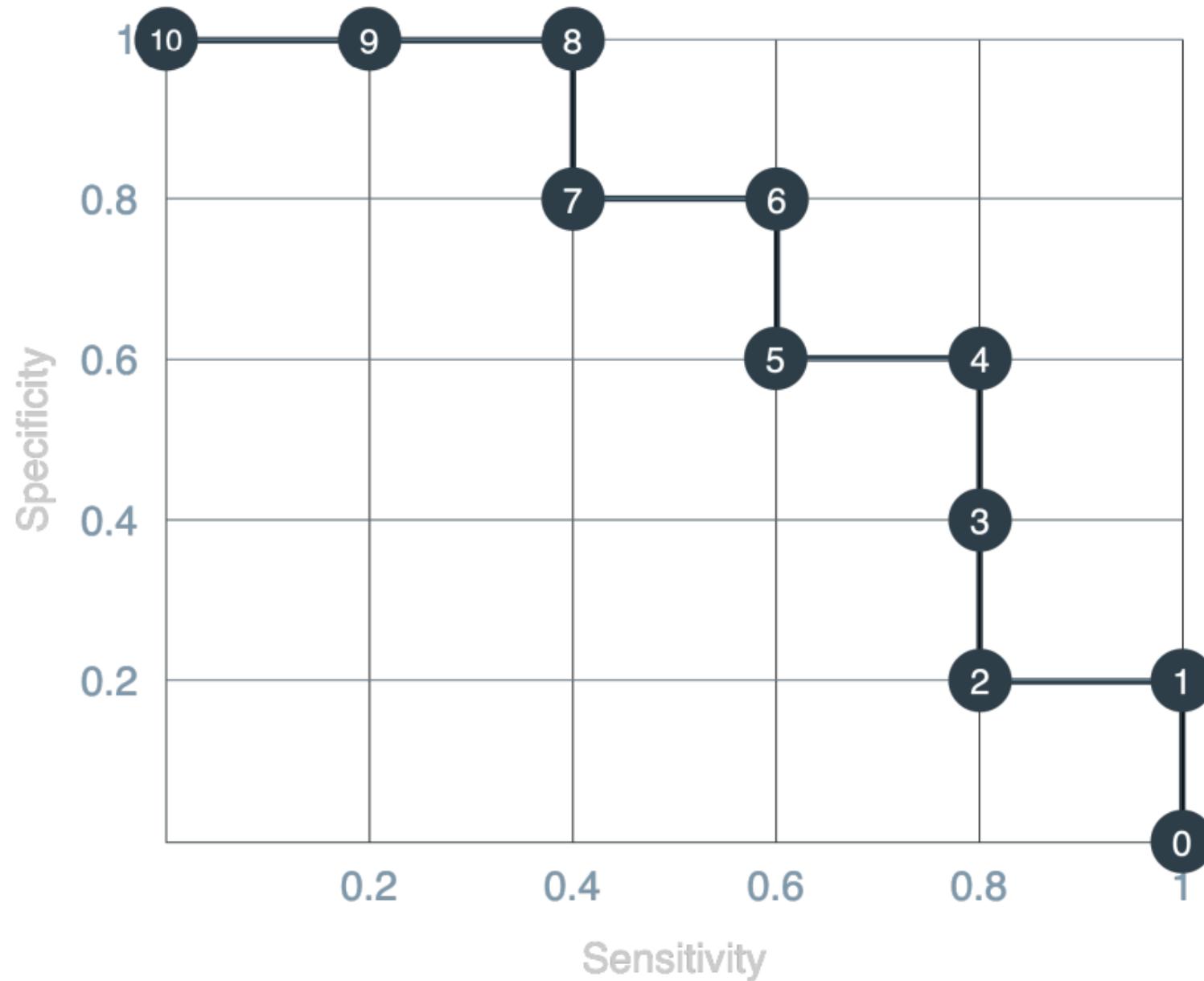
Threshold = 0.8
Sensitivity = 2/5
Specificity = 5/5

Timestep	True positives	Sensitivity	True negatives	Specificity
0	5	1	0	0
1	5	1	1	0.2
2	4	0.8	1	0.2
3	4	0.8	2	0.4
4	4	0.8	3	0.6
5	3	0.6	3	0.6
6	3	0.6	4	0.8
7	2	0.4	4	0.8

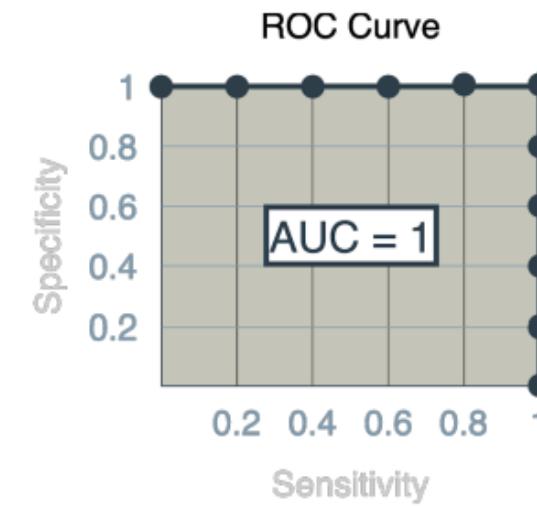
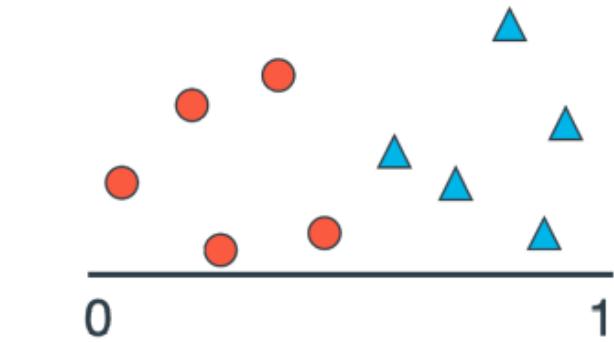
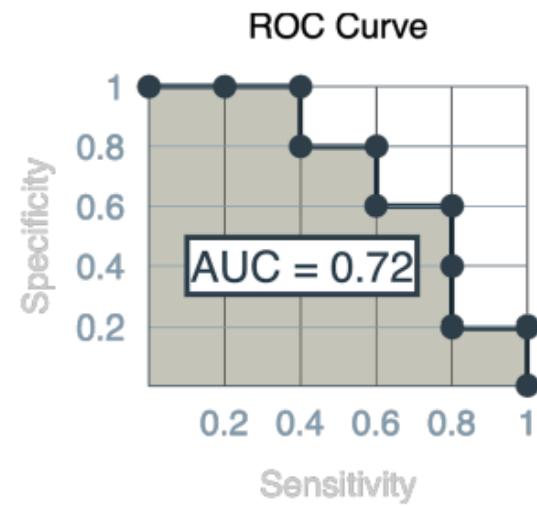
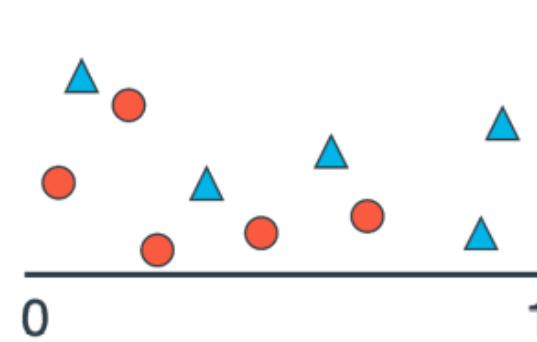
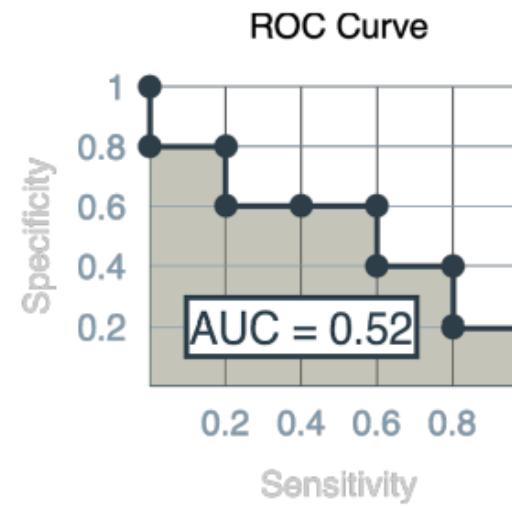
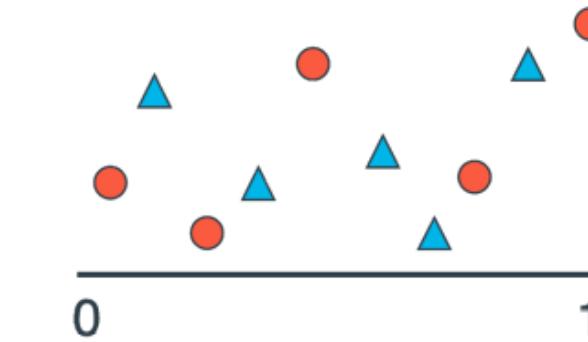
how many did we correctly classify?

8	2	0.4	5	1
9	1	0.2	5	1
10	0	0	5	1

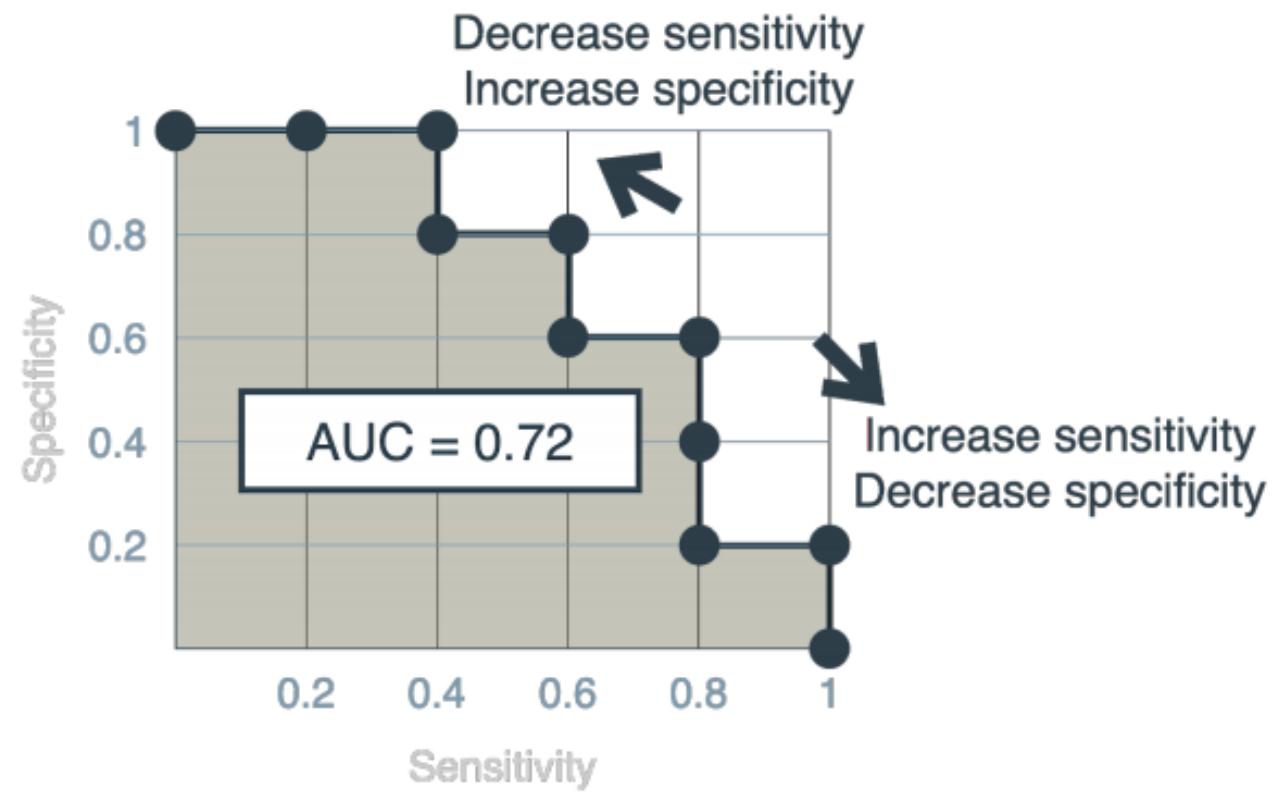
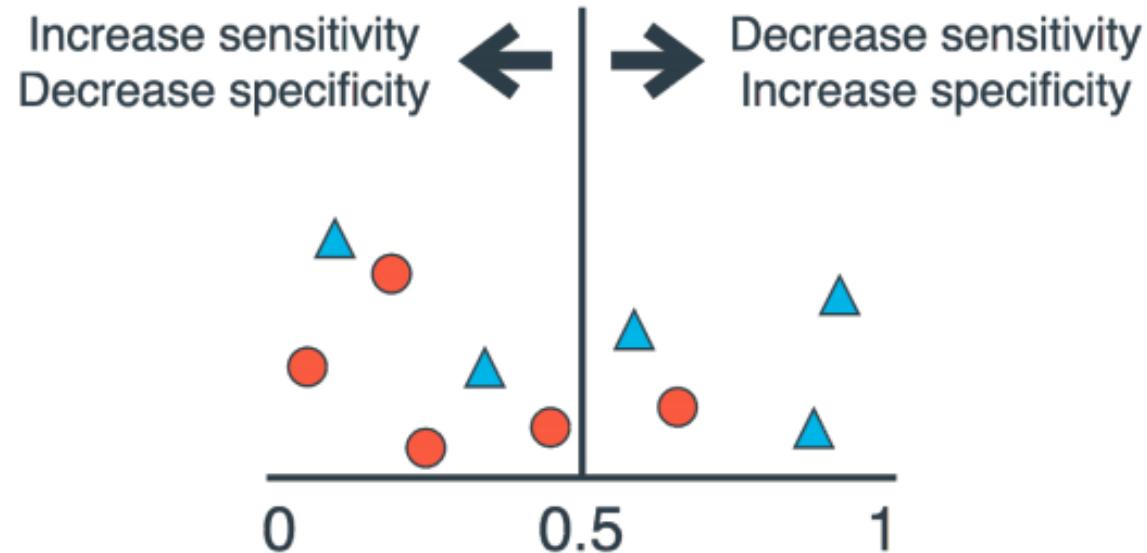
ROC Curve

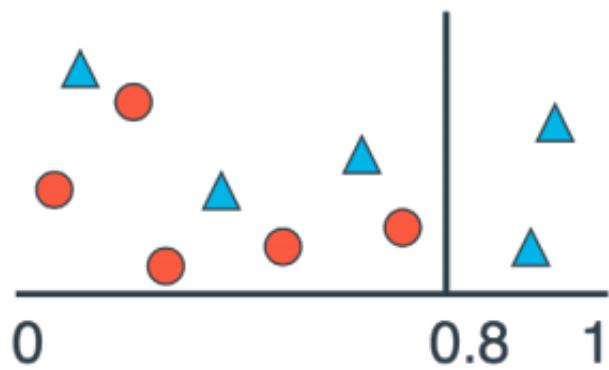


A metric that tells us how good our model is

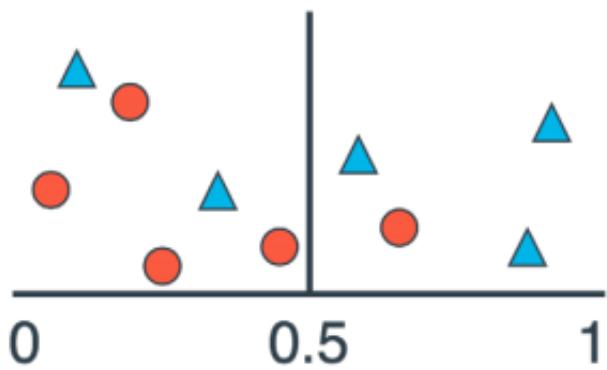


How to make decisions using the ROC curve

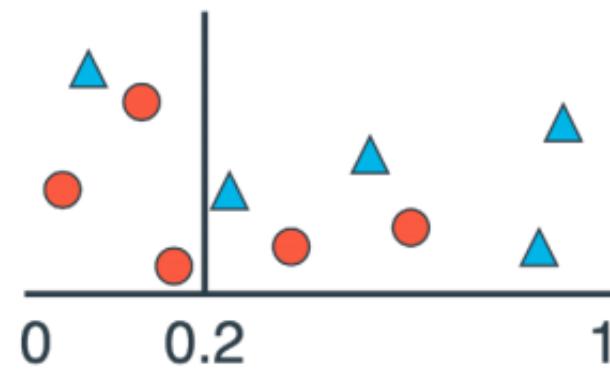




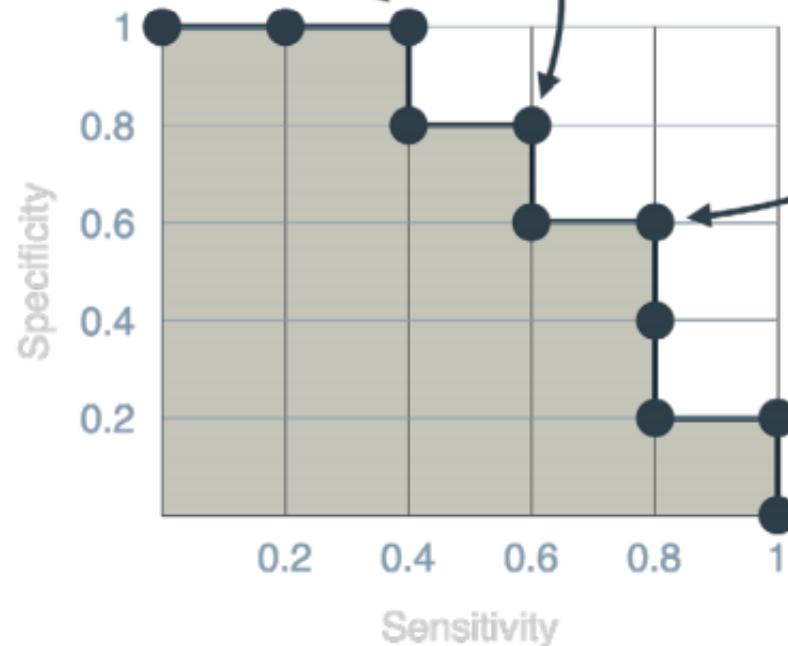
Threshold = 0.8
 Sensitivity = $2/5 = 0.4$
 Specificity = $5/5 = 1$

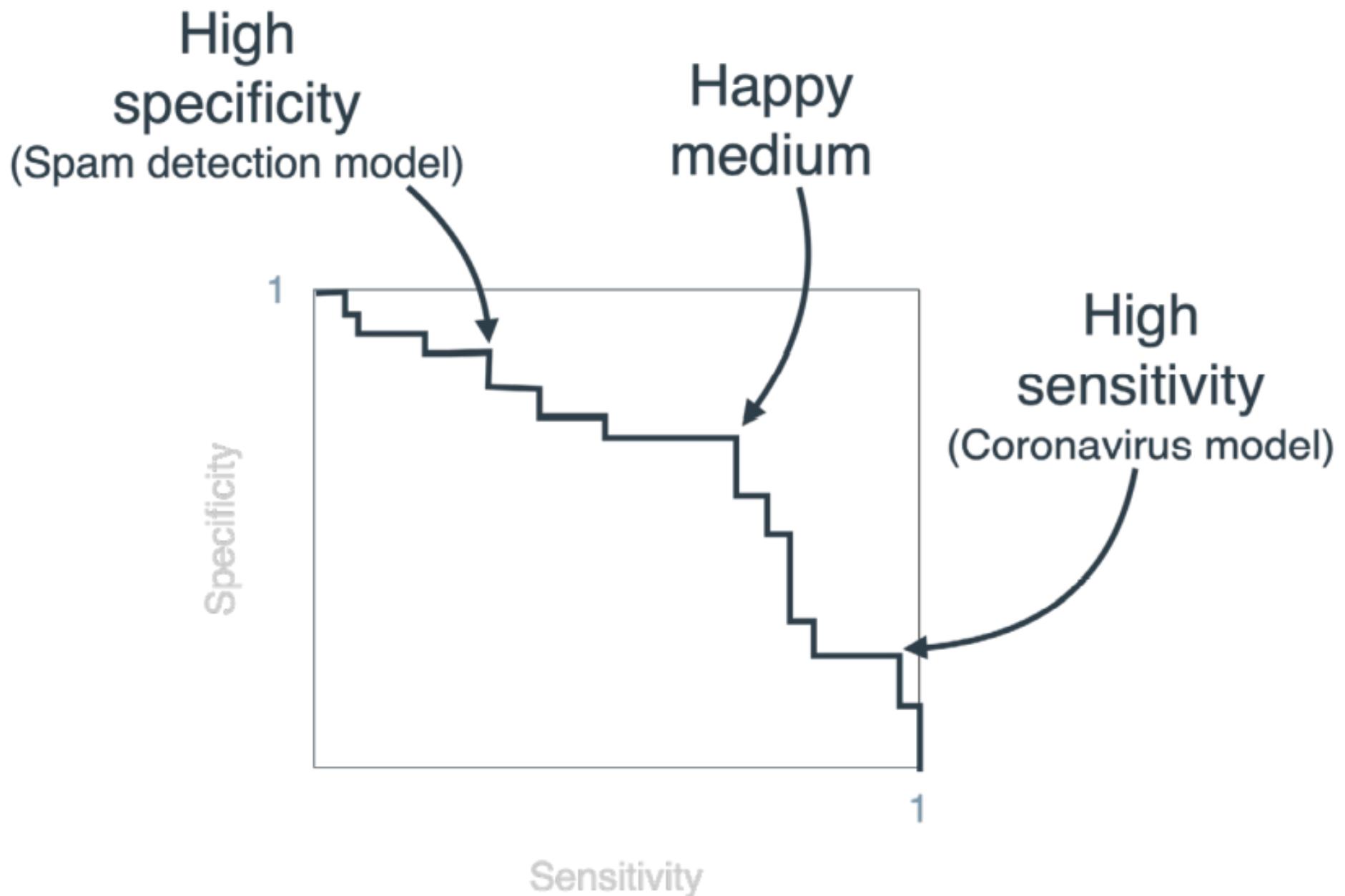


Threshold = 0.5
 Sensitivity = $3/5 = 0.6$
 Specificity = $4/5 = 0.8$



Threshold = 0.2
 Sensitivity = $4/5 = 0.8$
 Specificity = $3/5 = 0.6$





	Predicted positive	Predicted negative
Positive label	True positives	False negatives
Negative label	False positives	True negatives

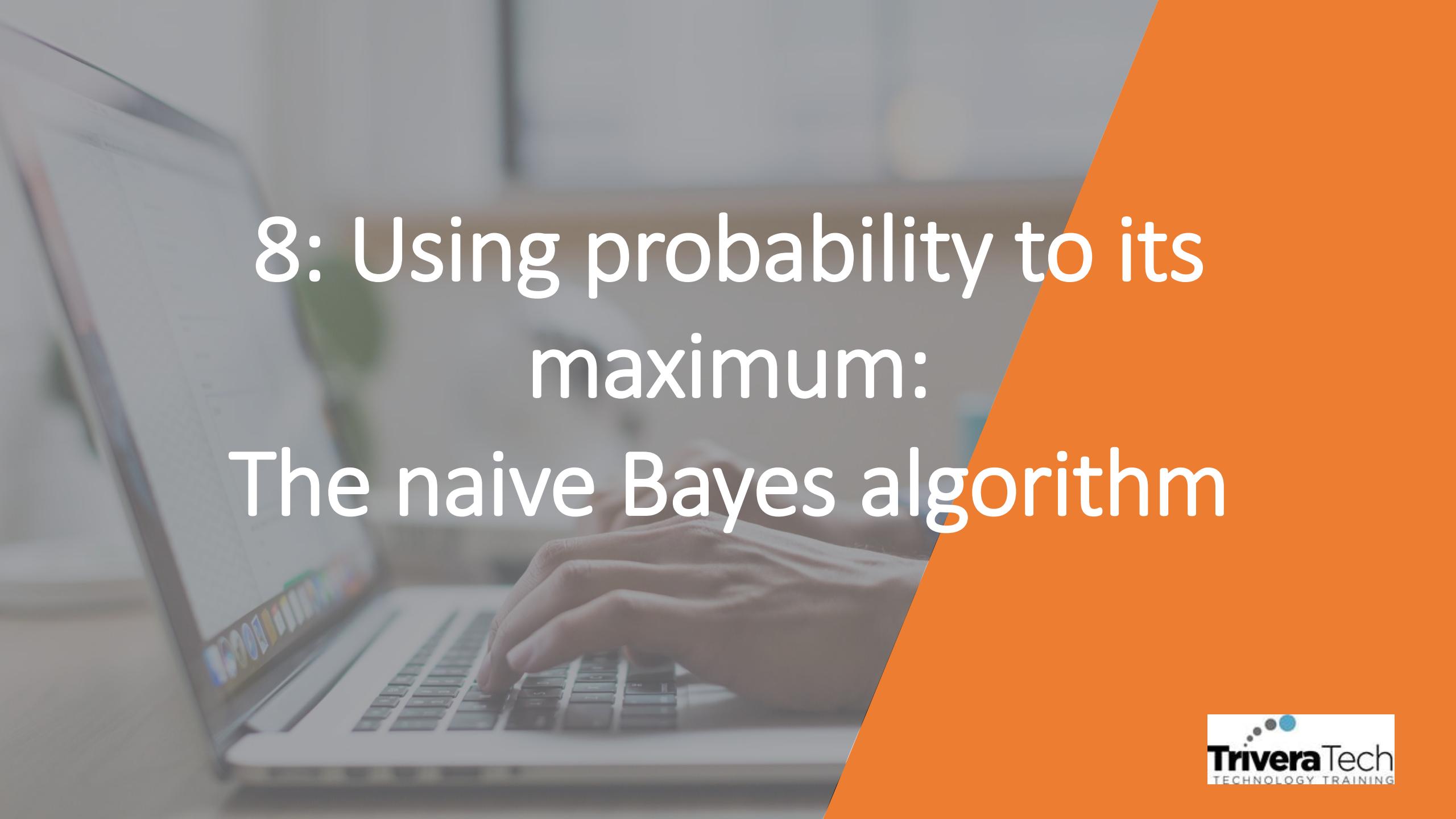
recall (sensitivity) = $\frac{TP}{TP + FN}$

specificity = $\frac{FP}{FP + TN}$

precision = $\frac{TP}{TP + FP}$

Summary

- Being able to evaluate a model is as important as being able to train one.
- There are several important metrics we can use to evaluate a model.
- The ones we learn in this lesson are accuracy, recall, precision, and F -score, specificity, and sensitivity.
- Accuracy calculates the ratio between correct predictions and total predictions.
- It is very useful, but can fail in certain cases, especially when the positive and negative labels are unbalanced.



8: Using probability to its maximum: The naive Bayes algorithm

The naive Bayes algorithm

This lesson covers

- What is Bayes theorem?
- When are events dependent or independent?
- The prior and the posterior probabilities.
- Calculating conditional probabilities based on events.
- What is the naive Bayes algorithm?
- Using the naive Bayes algorithm to predict if an email is spam or ham, based on the words in the email.
- Coding the naive Bayes algorithm in Python.

Sick or healthy? A story with Bayes Theorem

Consider the following scenario. Your (slightly hypochondriac) friend calls you, and the following conversation unfolds:

- **You:** Hello!
- **Friend:** Hi, I have some terrible news!
- **You:** Oh no, what is it?
- **Friend:** I heard about this terrible and rare disease, and I went to the doctor to be tested for it. The doctor said she would administer a very accurate test. Then today she called me and told me that I tested positive! I must have the disease!

Sick or healthy? A story with Bayes Theorem

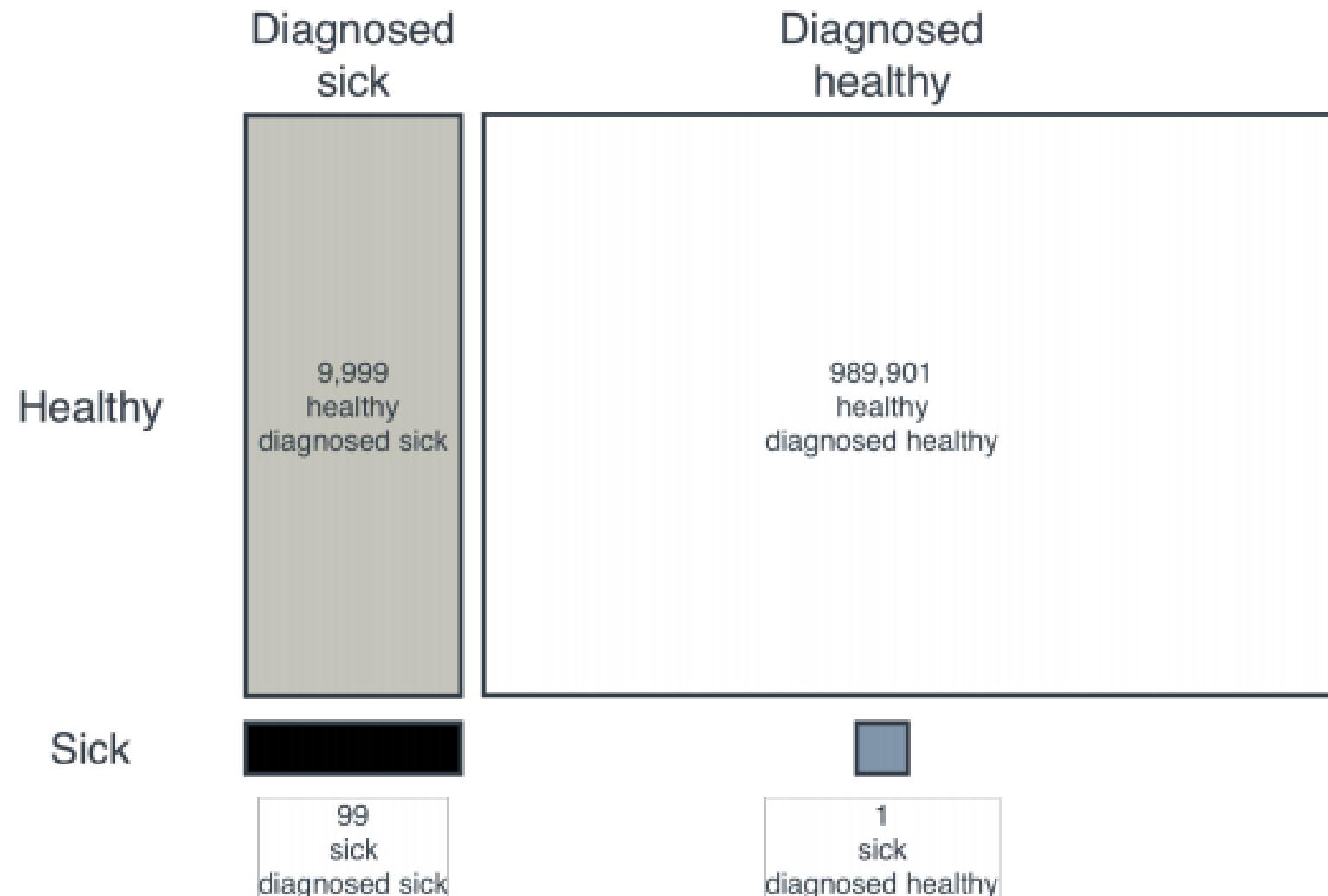
Friend: I was reading online, and it says that on average, 1 out of every 10,000 people have the disease.

You: Ok, let me get a piece of paper (*puts friend on hold*).

Sick or healthy? A story with Bayes Theorem

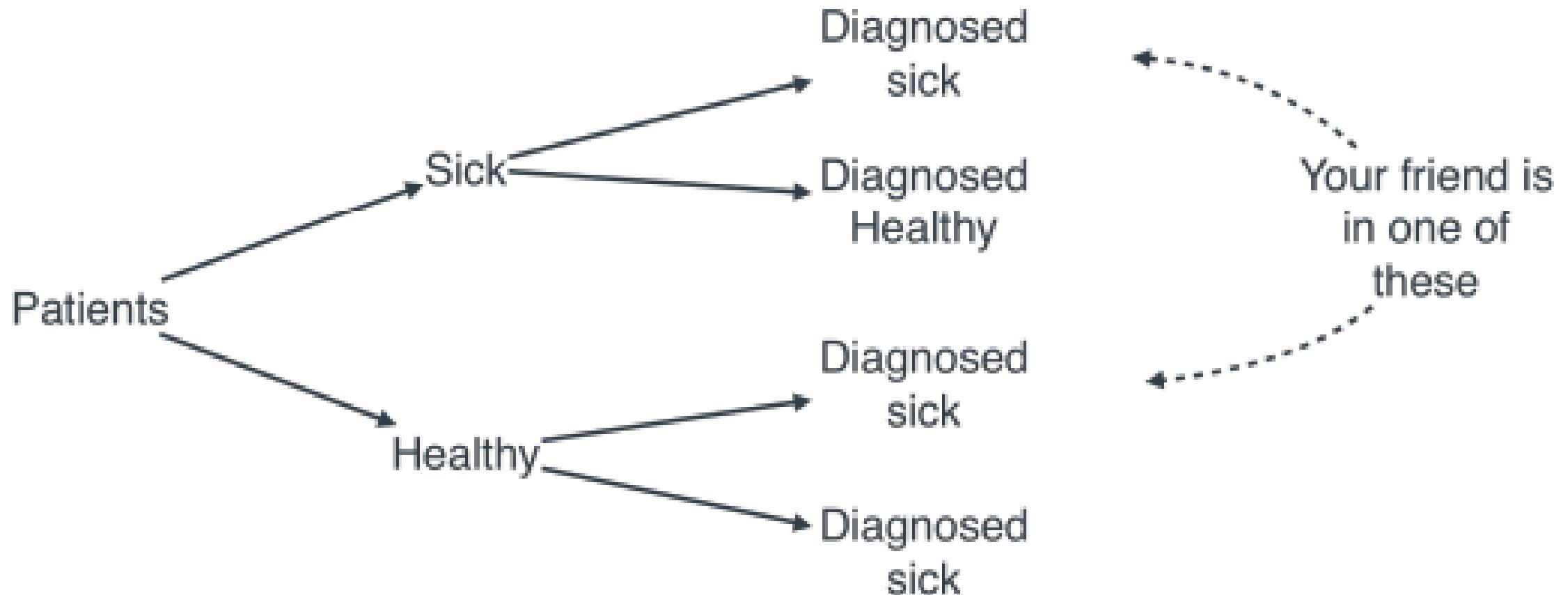
- **You:** Don't worry, based on the numbers you gave me, the probability that you have the disease given that you tested positive is actually only around 1%!
- **Friend:** Oh my God, really? That's such a relief, thank you!
- **You:** Don't thank me, thank math (winks eye).

Sick or healthy? A story with Bayes Theorem

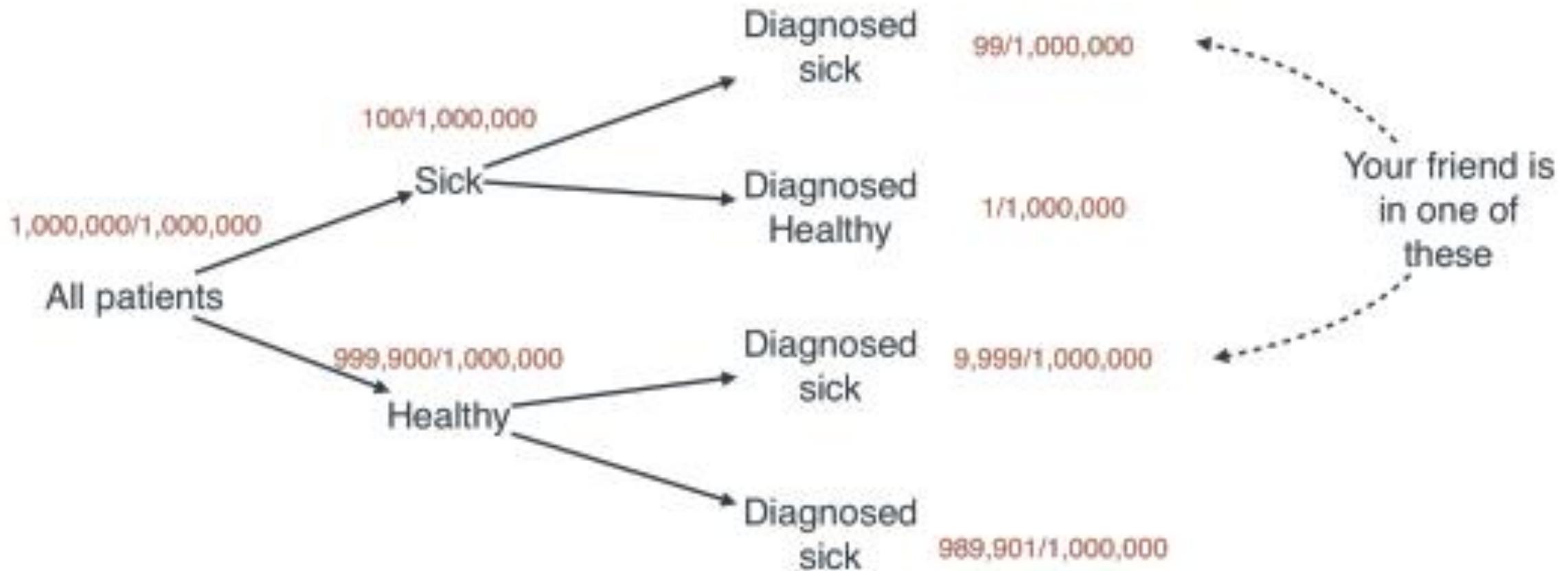


Sick or healthy? A story with Bayes Theorem

Sick or healthy? A story with Bayes Theorem



Sick or healthy? A story with Bayes Theorem

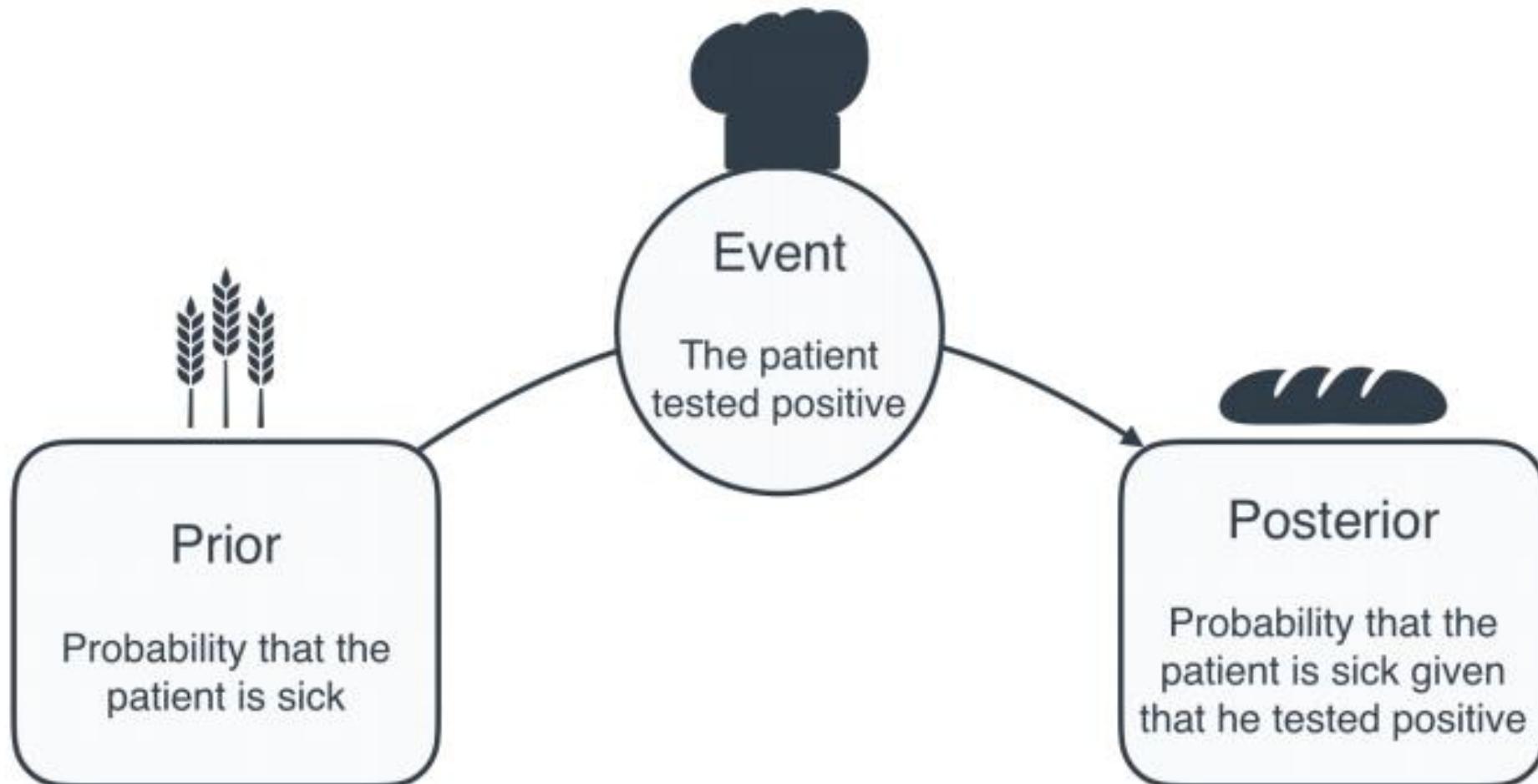


Sick or healthy? A story with Bayes Theorem

- From previous tree, we can again see that the probability that since among the patients that are diagnosed sick, 99 are actually sick, and 9,999 are healthy, from which we again can deduce that the probability that our friend is sick is:
- Probability of being sick when being diagnosed sick

$$\frac{99}{99 + 9,999} = 0.0098$$

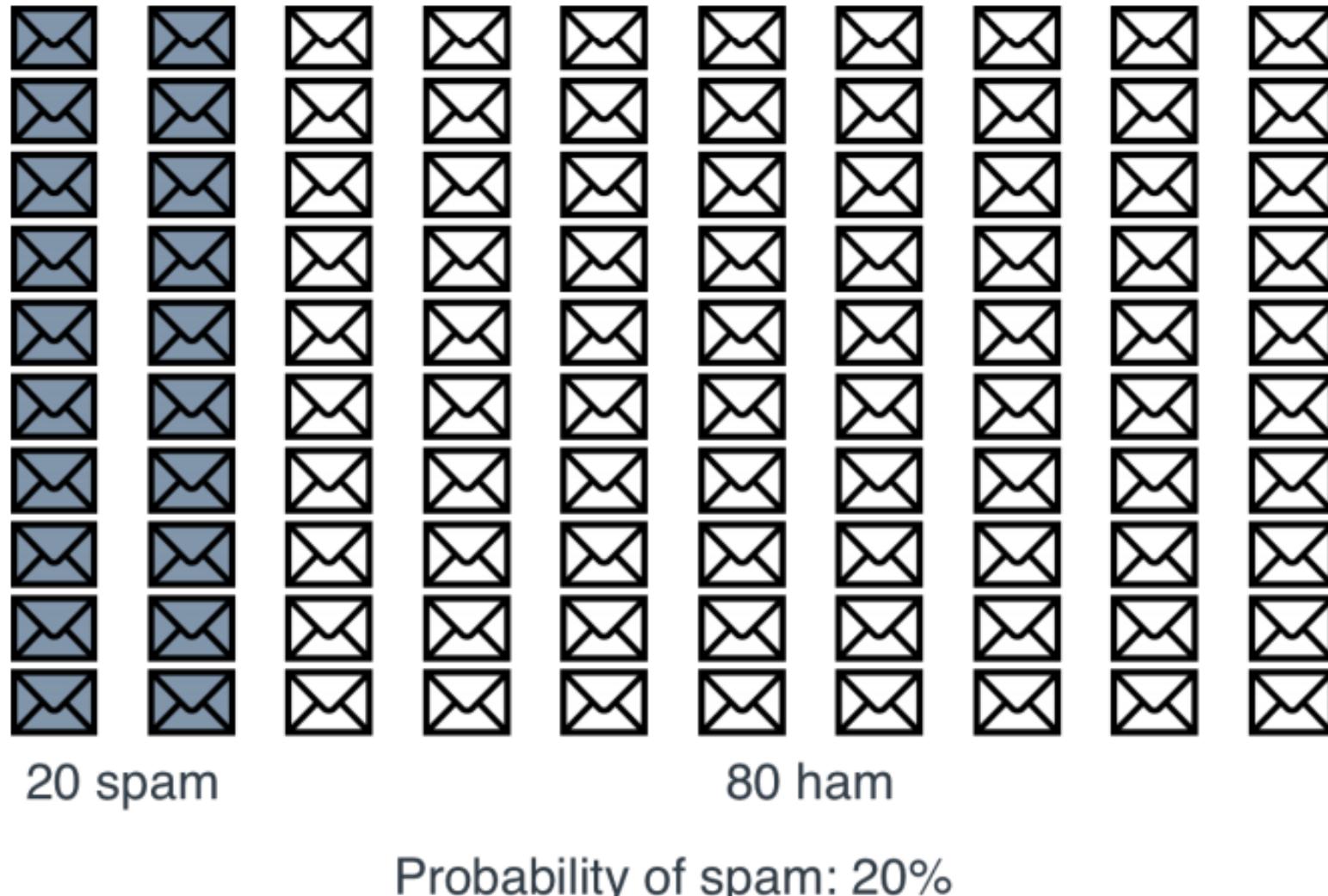
Prelude to Bayes Theorem: The prior, the event, and the posterior



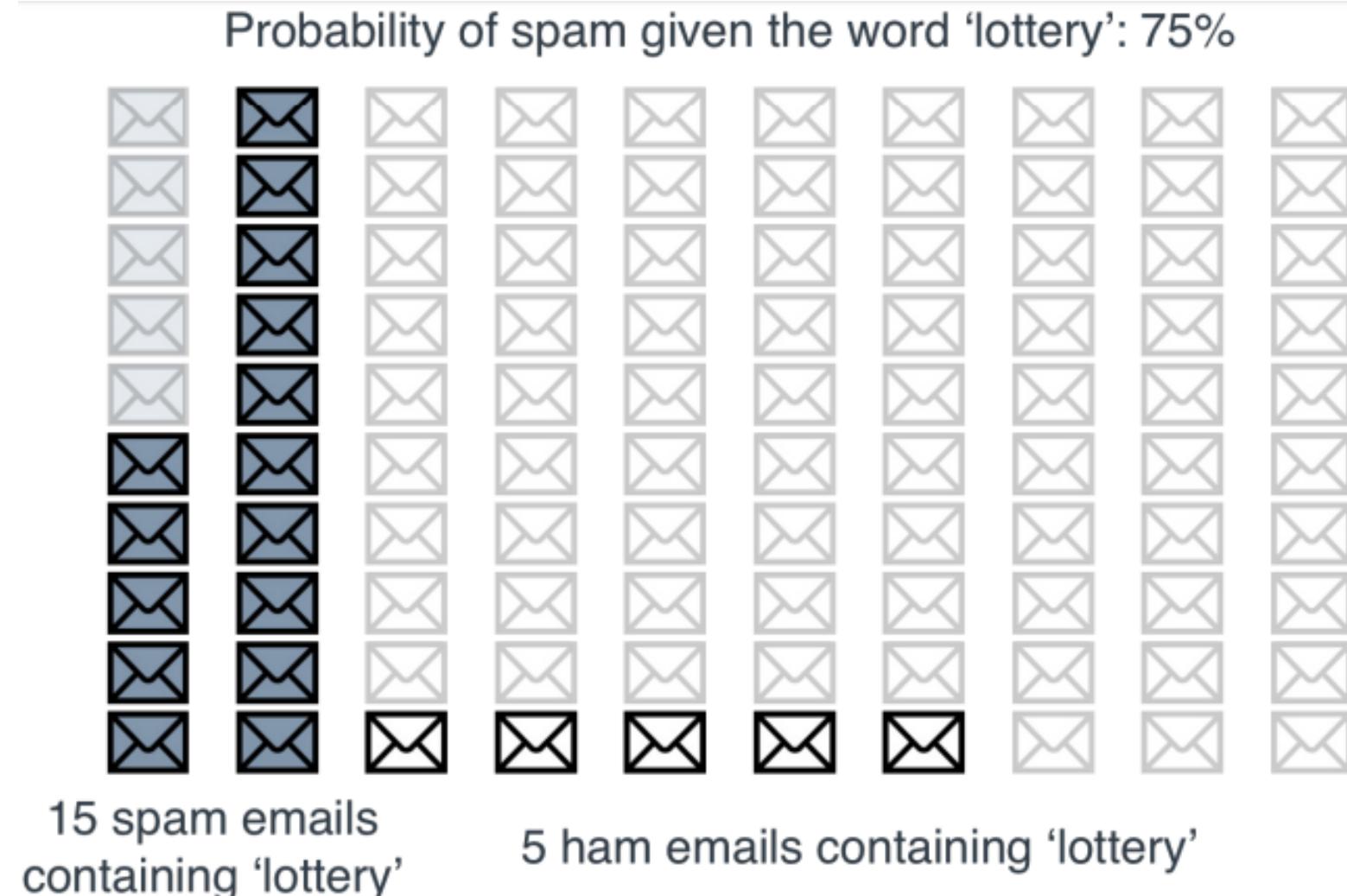
Use-case: Spam detection model

- Now consider the following situation, you are reading your email, and you are sick and tired of receiving spam (garbage) email.
- You would like to build a tool that helps you separate spam from non-spam emails

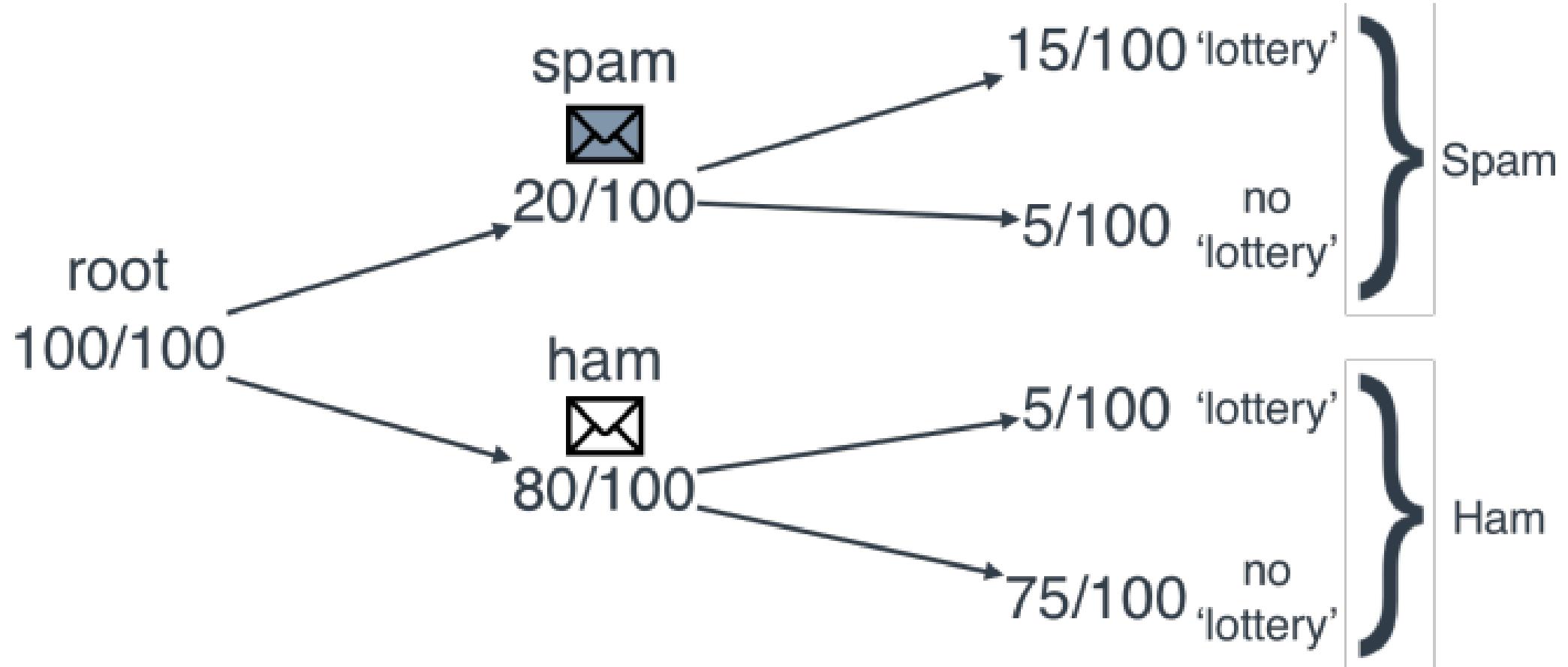
Finding the prior: The probability that any email is spam



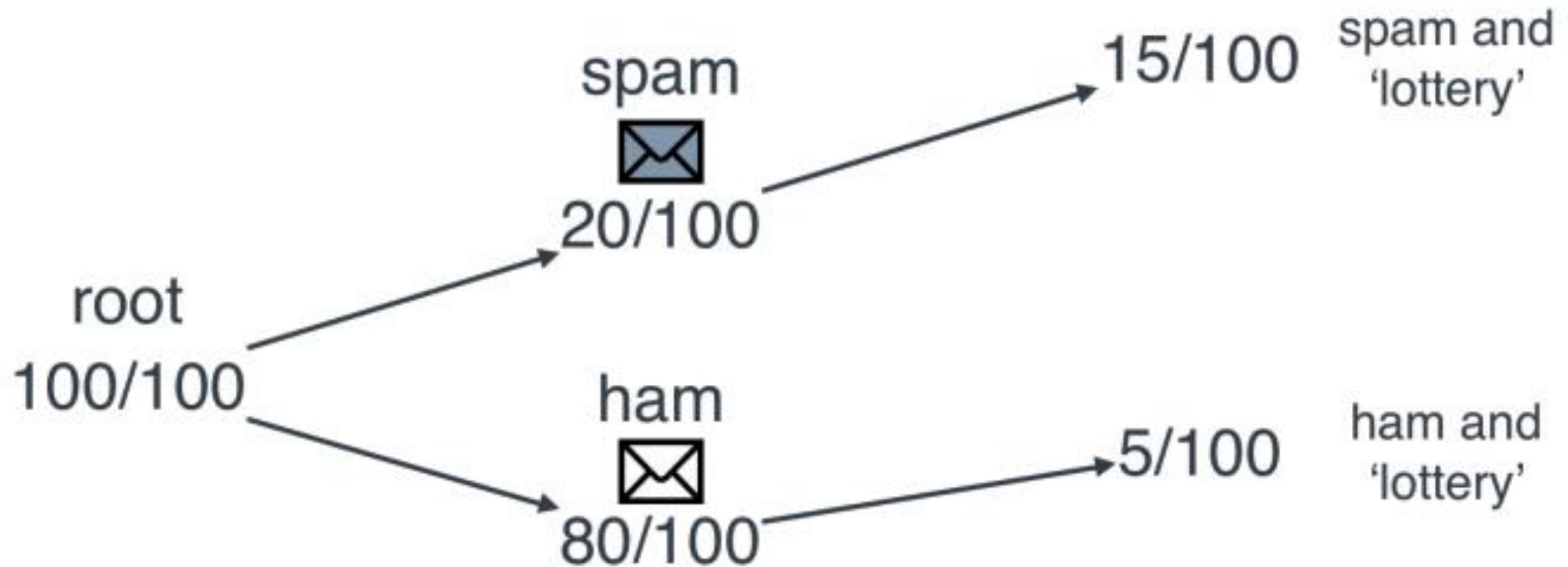
Finding the prior: The probability that any email is spam



What the math just happened? Turning ratios into probabilities



What the math just happened? Turning ratios into probabilities



What the math just happened? Turning ratios into probabilities

RULE OF COMPLEMENTARY PROBABILITIES For an event E , the complement of the event E , denoted E^c , is the event opposite to E . The probability of E^c is 1 minus the probability of E , namely,

$$P(E^c) = 1 - P(E).$$

So we have the following:

- $P(\text{spam}) = \frac{1}{5}$. The probability of an email being spam.
- $P(\text{ham}) = \frac{4}{5}$. The probability of an email being ham.

What the math just happened? Turning ratios into probabilities

- **P ('lottery'|spam) =3/4** The probability that a spam email contains the word 'lottery'.
- **P ('no lottery'|spam) =1/4** The probability that a spam email does not contain the word 'lottery'.
- **P ('lottery'|ham) =1/6** The probability that a ham email contains the word 'lottery'.
- **P (no 'lottery'|ham) =15/6** The probability that a ham email does not contain the word 'lottery'.

What the math just happened? Turning ratios into probabilities

These events are called intersections of events, and denoted with the symbol, thus, we need to find the following probabilities:

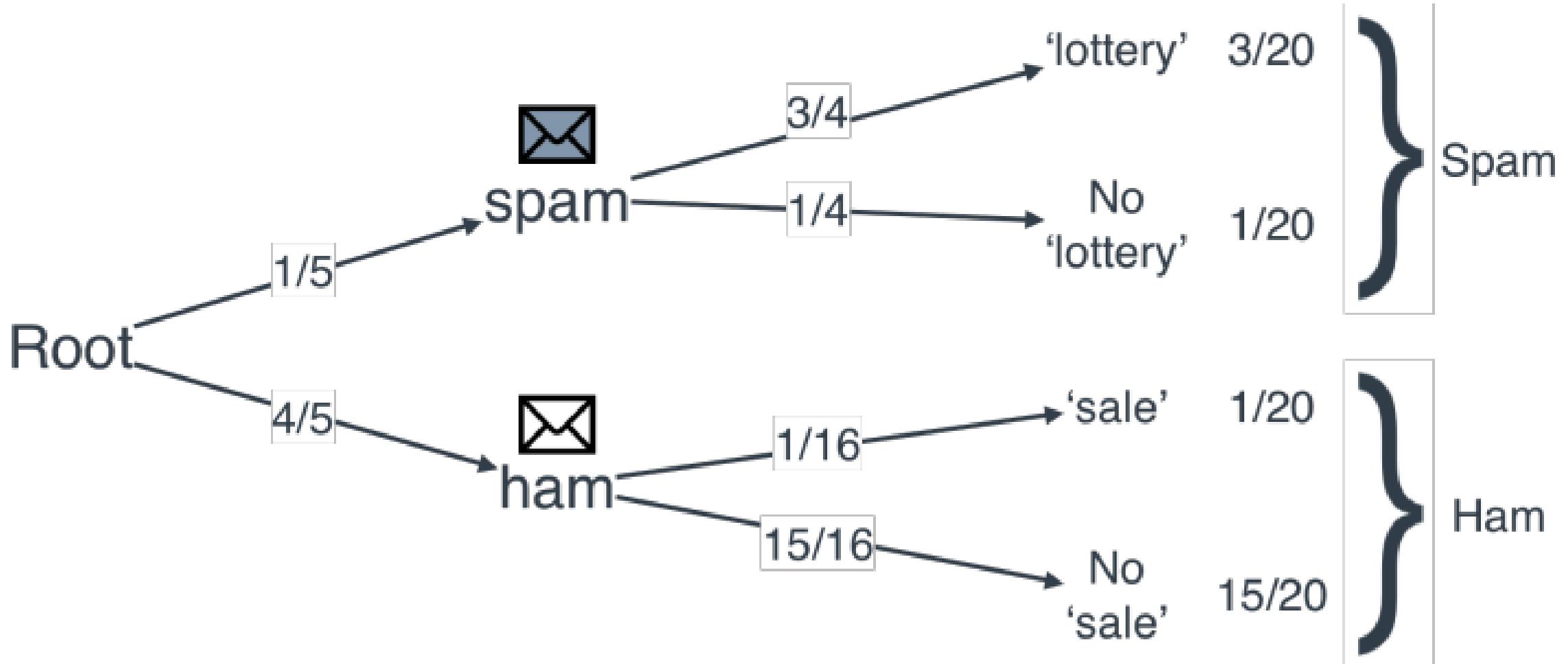
- $P(\text{'lottery} \cap \text{spam})$
- $P(\text{no 'lottery} \cap \text{spam})$
- $P(\text{'lottery} \cap \text{ham})$
- $P(\text{no 'lottery} \cap \text{ham})$

What the math just happened? Turning ratios into probabilities

- So now we can calculate these probabilities:

- $P('lottery' \cap \text{spam}) = P('lottery' | \text{spam}) \cdot P(\text{spam}) = \frac{1}{5} \cdot \frac{3}{4} = \frac{3}{20}$
- $P(\text{no } 'lottery' \cap \text{spam}) = P(\text{no } 'lottery' | \text{spam}) \cdot P(\text{spam}) = \frac{1}{5} \cdot \frac{1}{4} = \frac{1}{20}$
- $P('lottery' \cap \text{ham}) = P('lottery' | \text{ham}) \cdot P(\text{ham}) = \frac{4}{5} \cdot \frac{1}{16} = \frac{1}{20}$
- $P(\text{no } 'lottery' \cap \text{ham}) = P(\text{no } 'lottery' | \text{ham}) \cdot P(\text{ham}) = \frac{4}{5} \cdot \frac{15}{16} = \frac{15}{20}$.

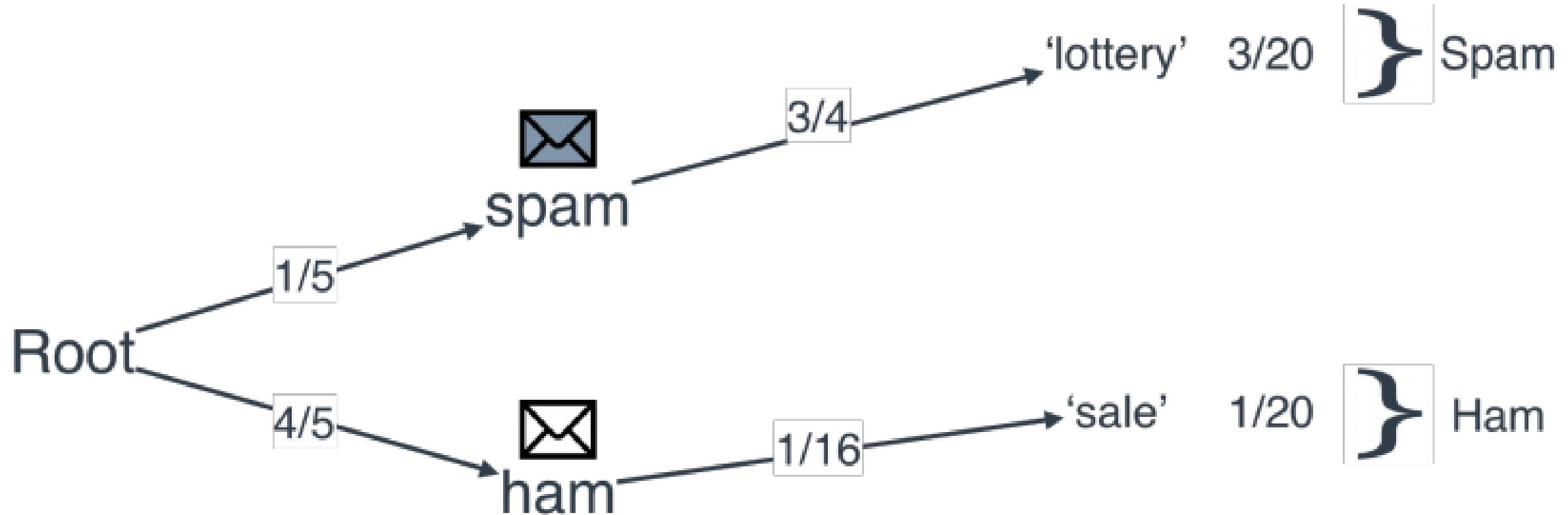
What the math just happened? Turning ratios into probabilities



What the math just happened? Turning ratios into probabilities

- Among the four events we just studied, in only two of them does the word ‘lottery’ appear. Thus, we only need to consider those, namely:
 - $P('lottery' \cap \text{spam}) = \frac{3}{20}$
 - $P('lottery' \cap \text{ham}) = \frac{1}{20}$.

What the math just happened? Turning ratios into probabilities



What the math just happened? Turning ratios into probabilities

In this case, the numbers become $\frac{3/20}{3/20 + 1/20}$ and $\frac{1/20}{3/20 + 1/20}$. These simplify to $\frac{3}{4}$ and $\frac{1}{4}$, which are the desired probabilities. Thus, we conclude that

- $P(\text{spam} \mid \text{'lottery'}) = \frac{3}{4} = 75\%$
- $P(\text{ham} \mid \text{'lottery'}) = \frac{1}{4} = 25\%.$

What the math just happened? Turning ratios into probabilities

- This is the same thing as dividing each one of them by their sum,
In math terms, we did the following:

$$P(\text{spam} \mid \text{'lottery'}) = \frac{P(\text{'lottery'} \cap \text{spam})}{P(\text{'lottery'} \cap \text{spam}) + P(\text{'lottery'} \cap \text{ham})}.$$

- If we remember what these two probabilities were, using the multiplication rule, we get the following:

$$P(\text{spam} \mid \text{'lottery'}) = \frac{P(\text{'lottery'} \mid \text{spam}) \cdot P(\text{spam})}{P(\text{'lottery'} \mid \text{spam}) \cdot P(\text{spam}) + P(\text{'lottery'} \mid \text{ham}) \cdot P(\text{ham})}.$$

What the math just happened? Turning ratios into probabilities

To verify, we plug in the numbers to get:

$$P(\text{spam} \mid \text{'lottery'}) = \frac{\frac{1}{5} \cdot \frac{3}{4}}{\frac{1}{5} \cdot \frac{3}{4} + \frac{4}{5} \cdot \frac{1}{16}} = \frac{\frac{3}{20}}{\frac{3}{20} + \frac{1}{20}} = \frac{\frac{3}{20}}{\frac{4}{20}} = \frac{3}{4}.$$

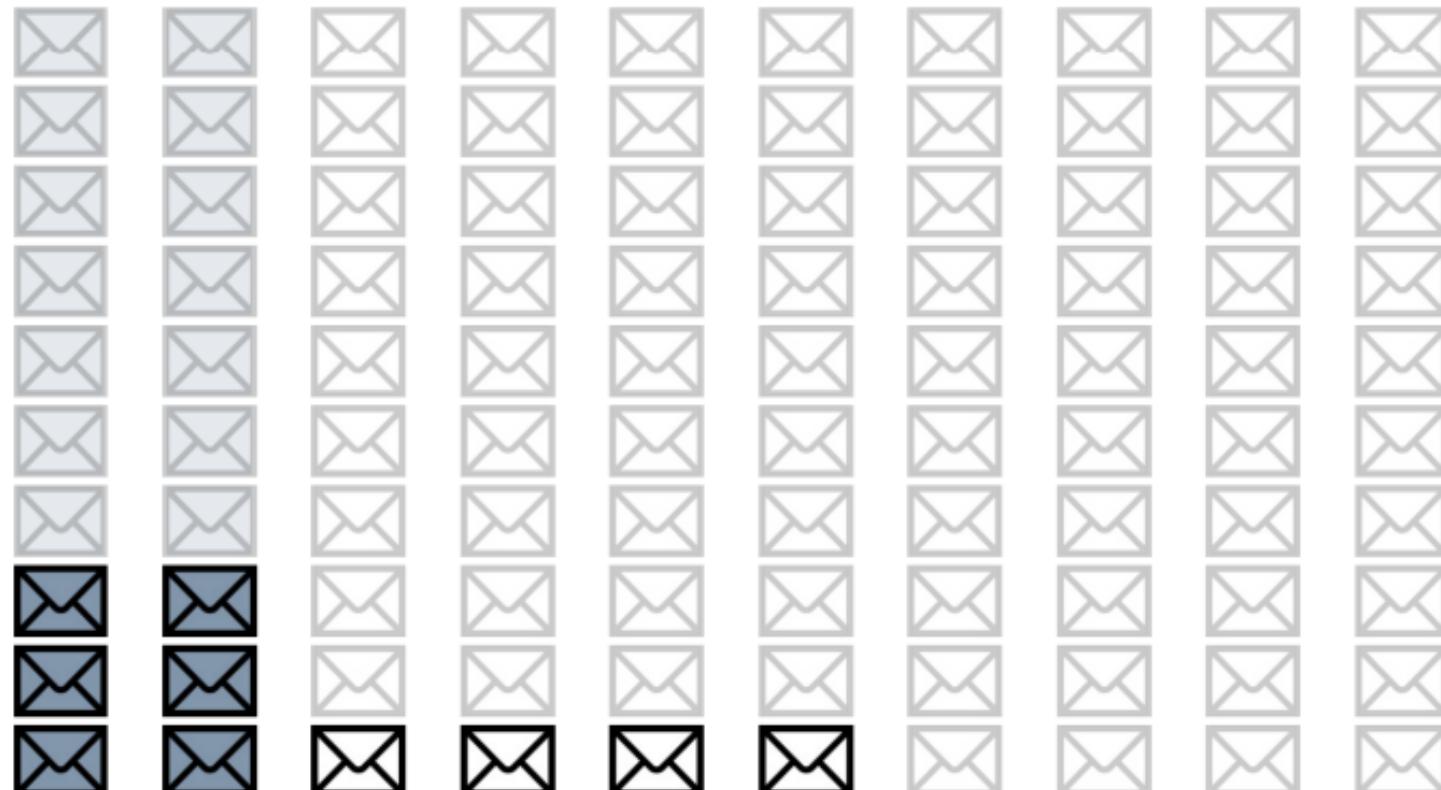
And this is the formula for Bayes Theorem! More formally:

BAYES THEOREM For events E and F,

$$P(E \mid F) = \frac{P(F \mid E) \cdot P(E)}{P(F \mid E) \cdot P(E) + P(F \mid E^c) \cdot P(E^c)}.$$

What about two words? The naive Bayes algorithm

Probability of spam given the word ‘sale’: 60%



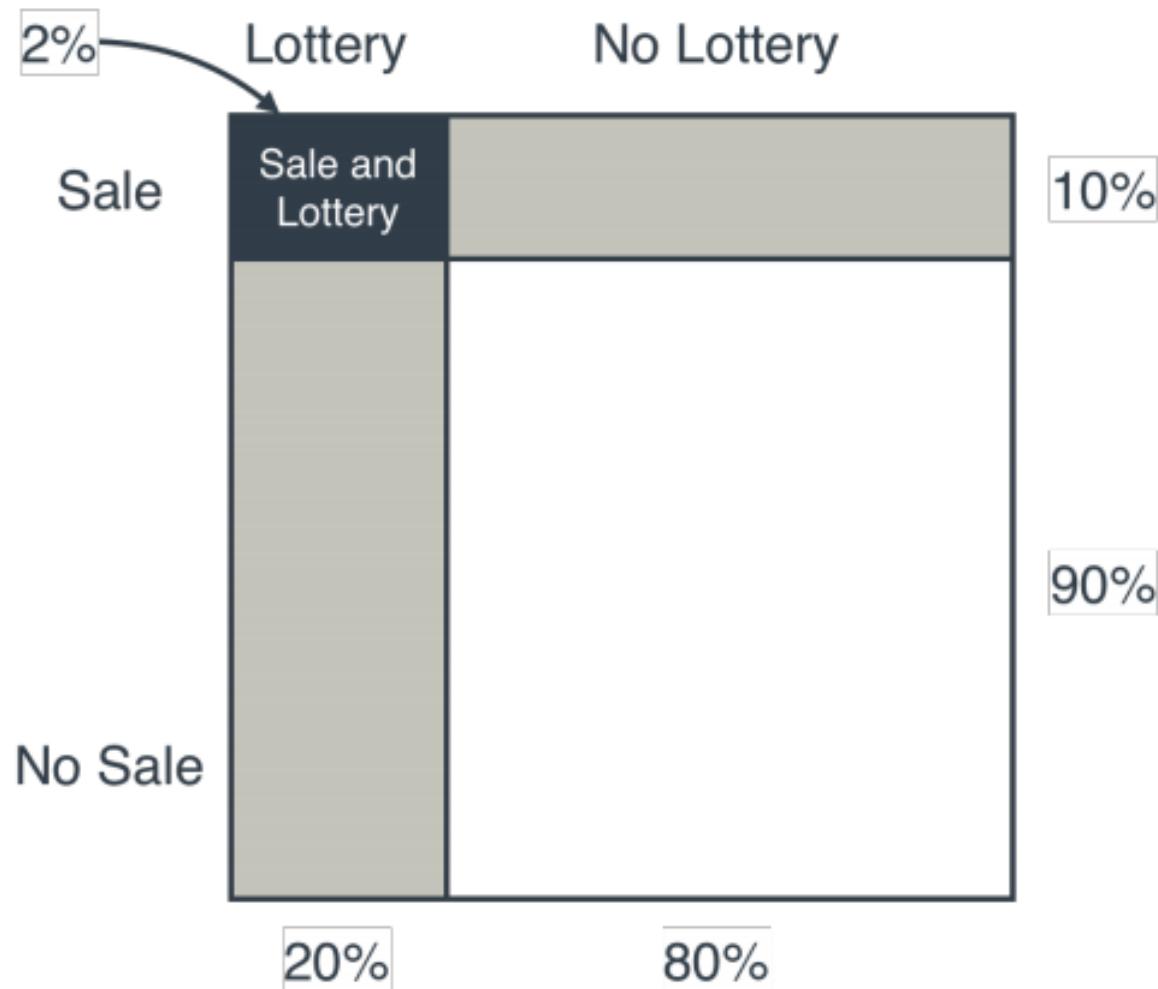
6 spam emails
containing ‘sale’

4 ham emails containing ‘sale’

What about two words? The naive Bayes algorithm

- What's the solution here, should we gather more data? That is always a good idea, but many times we can't, so we have to deal with the data we have.
- In some way, we can't rely on the emails that contain both words 'lottery' and 'sale'.

What about two words? The naive Bayes algorithm



What about two words? The naive Bayes algorithm

- In probability terms, our assumption boils down to the following:

- $P(\text{lottery} \mid \text{spam}) = \frac{3}{4}$
- $P(\text{sale} \mid \text{spam}) = \frac{3}{10}$
- $P(\text{lottery} \cap \text{sale} \mid \text{spam}) = \frac{3}{4} \cdot \frac{3}{10} = \frac{9}{40} = 0.225.$

What about two words? The naive Bayes algorithm

- In probability terms, our assumption is the following:

- $P(\text{lottery} \mid \text{ham}) = \frac{5}{80} = \frac{1}{16}$
- $P(\text{sale} \mid \text{ham}) = \frac{4}{80} = \frac{1}{20}$
- $P(\text{lottery} \cap \text{sale} \mid \text{ham}) = \frac{1}{16} \cdot \frac{1}{20} = \frac{1}{320} = 0.0625.$

What about two words? The naive Bayes algorithm



4.5 spam emails
containing 'lottery and sale'

0.0625 ham emails
containing lottery and 'sale'

What we did here, using probability, was using Bayes Theorem, except with the events

- $E = \text{lottery} \cap \text{sale}$
- $F = \text{spam},$

to get the formula

$$P(\text{spam} | \text{lottery} \cap \text{sale}) = \frac{P(\text{lottery} \cap \text{sale} | \text{spam}) \cdot P(\text{spam})}{P(\text{lottery} \cap \text{sale} | \text{spam}) \cdot P(\text{spam}) + P(\text{lottery} \cap \text{sale} | \text{ham}) \cdot P(\text{ham})}.$$

What about two words? The naive Bayes algorithm

$$P(\text{lottery} \cap \text{sale} \mid \text{spam}) = P(\text{lottery} \mid \text{spam}) \cdot P(\text{sale} \mid \text{spam}),$$

$$P(\text{lottery} \cap \text{sale} \mid \text{ham}) = P(\text{lottery} \mid \text{ham}) \cdot P(\text{sale} \mid \text{ham}).$$

Plugging them into the previous formula, we get

$$\begin{aligned} & P(\text{spam} \mid \text{lottery} \cap \text{sale}) \\ &= \frac{P(\text{lottery} \mid \text{spam}) \cdot P(\text{sale} \mid \text{spam}) \cdot P(\text{spam})}{P(\text{lottery} \mid \text{spam}) \cdot P(\text{sale} \mid \text{spam}) \cdot P(\text{spam}) + P(\text{lottery} \mid \text{ham}) \cdot P(\text{sale} \mid \text{ham}) \cdot P(\text{ham})}. \end{aligned}$$

What about two words? The naive Bayes algorithm

- $P(\text{lottery} \mid \text{spam}) = \frac{3}{4}$
- $P(\text{sale} \mid \text{spam}) = \frac{3}{10}$
- $P(\text{spam}) = \frac{1}{5}$
- $P(\text{lottery} \mid \text{ham}) = \frac{1}{16}$
- $P(\text{sale} \mid \text{ham}) = \frac{1}{20}$
- $P(\text{ham}) = \frac{4}{5}$

we get

$$P(\text{spam} \mid \text{lottery} \cap \text{sale}) = \frac{\frac{3}{4} \cdot \frac{3}{10} \cdot \frac{1}{5}}{\frac{3}{4} \cdot \frac{3}{10} \cdot \frac{1}{5} + \frac{1}{16} \cdot \frac{1}{20} \cdot \frac{4}{5}} = 0.9863.$$

Building a spam detection model with real data

- In the repo, we have processed the dataset as a Pandas Data Frame with the following command:

```
import pandas
emails = pandas.read_csv('emails.csv')
```
- If we look at the first 10 rows of this dataset, this is how it looks .

Building a spam detection model with real data

		text	spam
0	Subject: naturally irresistible your corporate...		1
1	Subject: the stock trading gunslinger fanny i...		1
2	Subject: unbelievable new homes made easy im ...		1
3	Subject: 4 color printing special request add...		1
4	Subject: do not have money , get software cds ...		1
5	Subject: great nnews hello , welcome to medzo...		1
6	Subject: here ' s a hot play in motion homela...		1
7	Subject: save your money buy getting this thin...		1
8	Subject: undeliverable : home based business f...		1
9	Subject: save your money buy getting this thin...		1

Data preprocessing

- Let's start by turning the text string into a list of words.
- This is done in the following function, which uses the `split()` function.
- Since we only check if each word appears in the email or not, regardless of how many times it appears, we turn it into a set, and then into a list again.

```
def process_email(text):  
    return list(set(text.split()))
```

Data preprocessing

- Now we use the `apply()` function to apply this change to the entire column.
- We call the new column `emails['words']`.

```
emails['words'] = emails['text'].apply(process_email)
```

Data preprocessing

	text	spam	words
0	Subject: naturally irresistible your corporate...	1	[all, through, portfolio, its, guaranteed, ,, ...
1	Subject: the stock trading gunslinger fanny i...	1	[and, merrill, is, nameable, clockwork, libret...
2	Subject: unbelievable new homes made easy im ...	1	[pre, and, all, show, being, visit, loan, 454,...
3	Subject: 4 color printing special request add...	1	[and, golden, 5110, 626, color, ca, an, canyon...
4	Subject: do not have money , get software cds ...	1	[comedies, all, old, tradgedies, be, money, is...
5	Subject: great nnews hello , welcome to medzo...	1	[va, groundsel, allusion, ag, tosher, confide,...
6	Subject: here ' s a hot play in motion homela...	1	[precise, all, chain, limited, indicating, ena...
7	Subject: save your money buy getting this thin...	1	[right, want, just, money, is, within, it, rea...
8	Subject: undeliverable : home based business f...	1	[unknown, grownups, co, telecom, is, mts, 000,...
9	Subject: save your money buy getting this thin...	1	[right, want, just, money, is, within, it, rea...

Finding the priors

- Let's first find out the probability that an email is spam (the prior).
- For this, we calculate the number of emails that are spam, and divide it by the total number of emails.
- Notice that the number of emails that are spam is simply the sum of entries in the 'spam' column.
- The following line will do the job.

```
sum(emails['spam'])/len(emails)  
0.2388268156424581
```

Finding the posteriors with Bayes theorem

```
model = {}

for email in emails:
    for word in email['words']:
        if word not in model:
            model[word] = {'spam': 1, 'ham': 1}
        if word in model:
            if email['spam']:
                model[word]['spam'] += 1
            else:
                model[word]['ham'] += 1
```

Let's examine some rows of the dictionary:

```
model
{'woods': {'ham': 4, 'spam': 2},
 'spiders': {'ham': 1, 'spam': 3},
 'hanging': {'ham': 9, 'spam': 2}}
```

Finding the posteriors with Bayes theorem

- This means that for example the word ‘woods’ appears 4 times in spam emails, and 2 times in ham emails.
- Let’s find out the appearances of our words ‘lottery’ and ‘sale’

```
model['lottery']
{'ham': 1, 'spam': 9}

model['sale']
{'ham': 42, 'spam': 39}
```

Implementing the naive Bayes algorithm

- The code to train the model is the following:

```
def predict(email):
    words = set(email.split())
    spams = []
    hams = []
    for word in words:          #A
        if word in model:       #B
            spams.append(model[word]['spam'])      #C
            hams.append(model[word]['ham'])
    prod_spams = long(np.prod(spams))           #D
    prod_hams = long(np.prod(hams))
    return prod_spams/(prod_spams + prod_hams)    #E
```

Implementing the naive Bayes algorithm

- And that's it! Let's test the algorithm on some emails:

```
predict_naive_bayes('hi mom how are you')  
0.0013894756610580057
```

```
predict_naive_bayes('meet me at the lobby of the hotel at nine am')  
0.02490194297492509
```

```
predict_naive_bayes('enter the lottery to win three million dollars')  
0.38569290647197135
```

```
predict_naive_bayes('buy cheap lottery easy money now')  
0.9913514898646872
```

Summary

- Bayes theorem is a technique widely used in probability, statistics, and machine learning.
- Bayes theorem consists in calculating a posterior probability, based on a prior probability and an event.
- The prior probability is a basic calculation of a probability, given very little information.

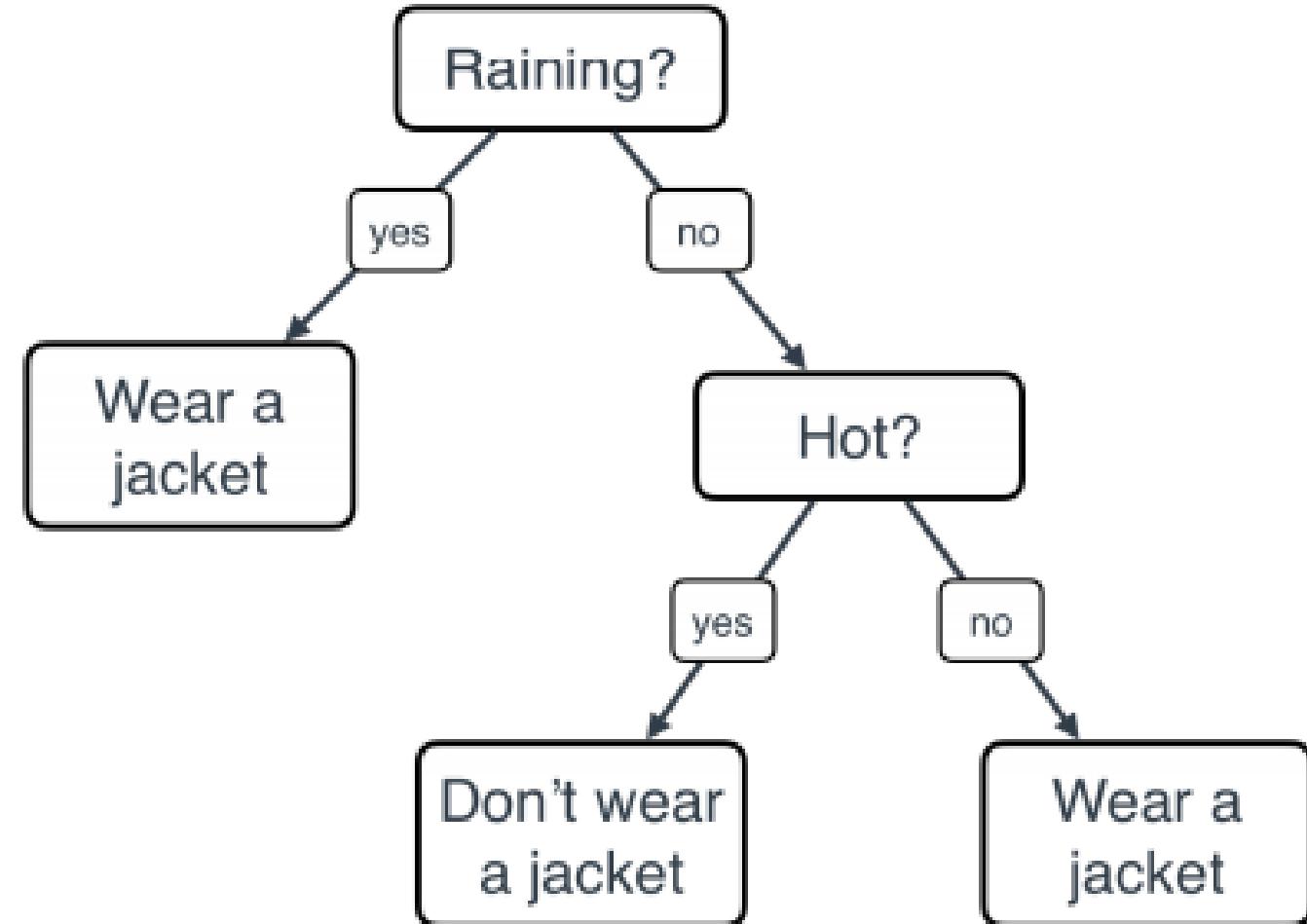
9: Splitting data by asking questions: Decision trees

Splitting data by asking questions: Decision trees

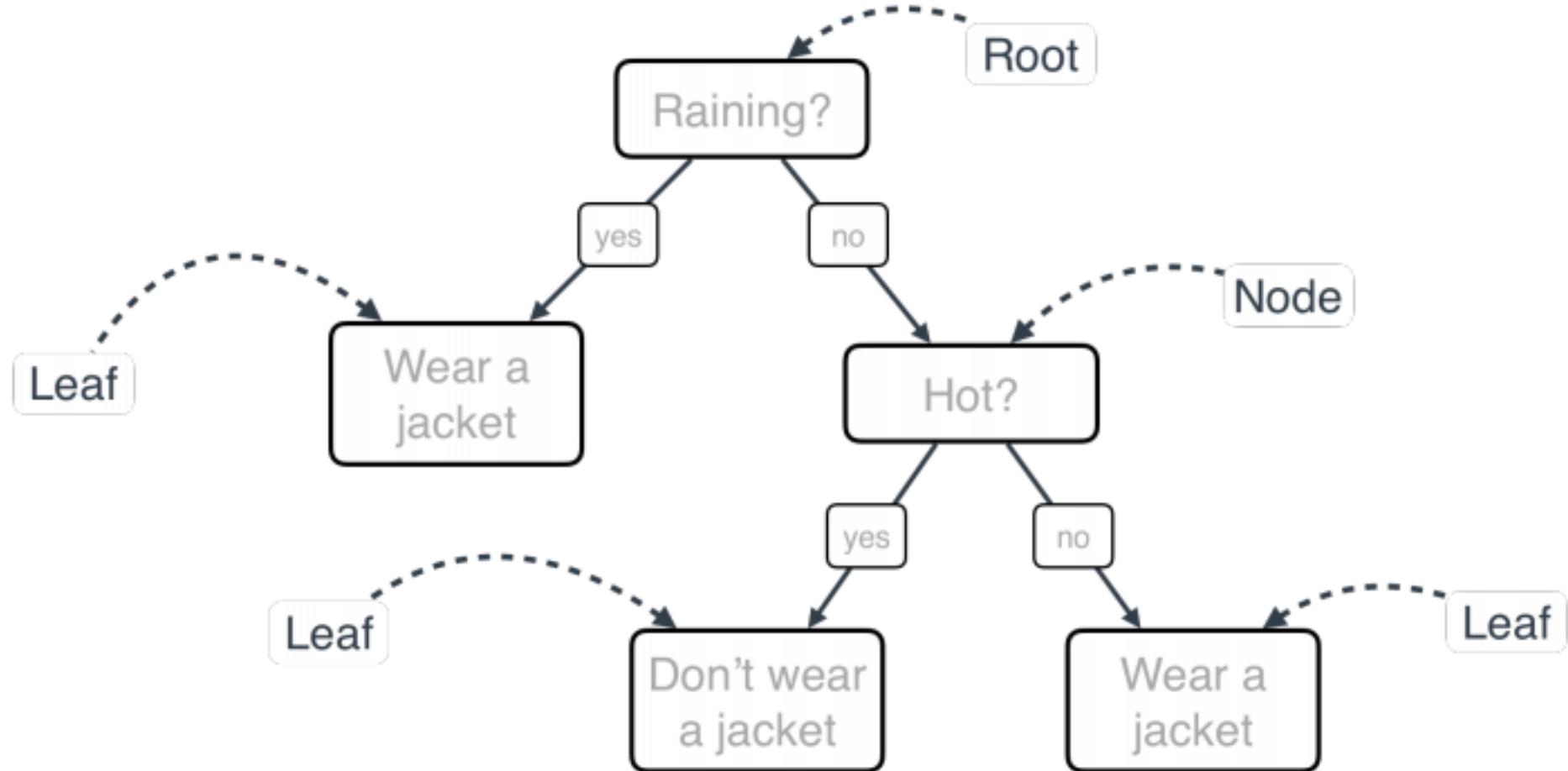
This lesson covers

- What is a decision tree?
- Recommending apps using the demographic information of the users.
- Asking a series of successive questions to build a good classifier.
- Accuracy, Gini index, and Entropy, and their role in building decision trees.

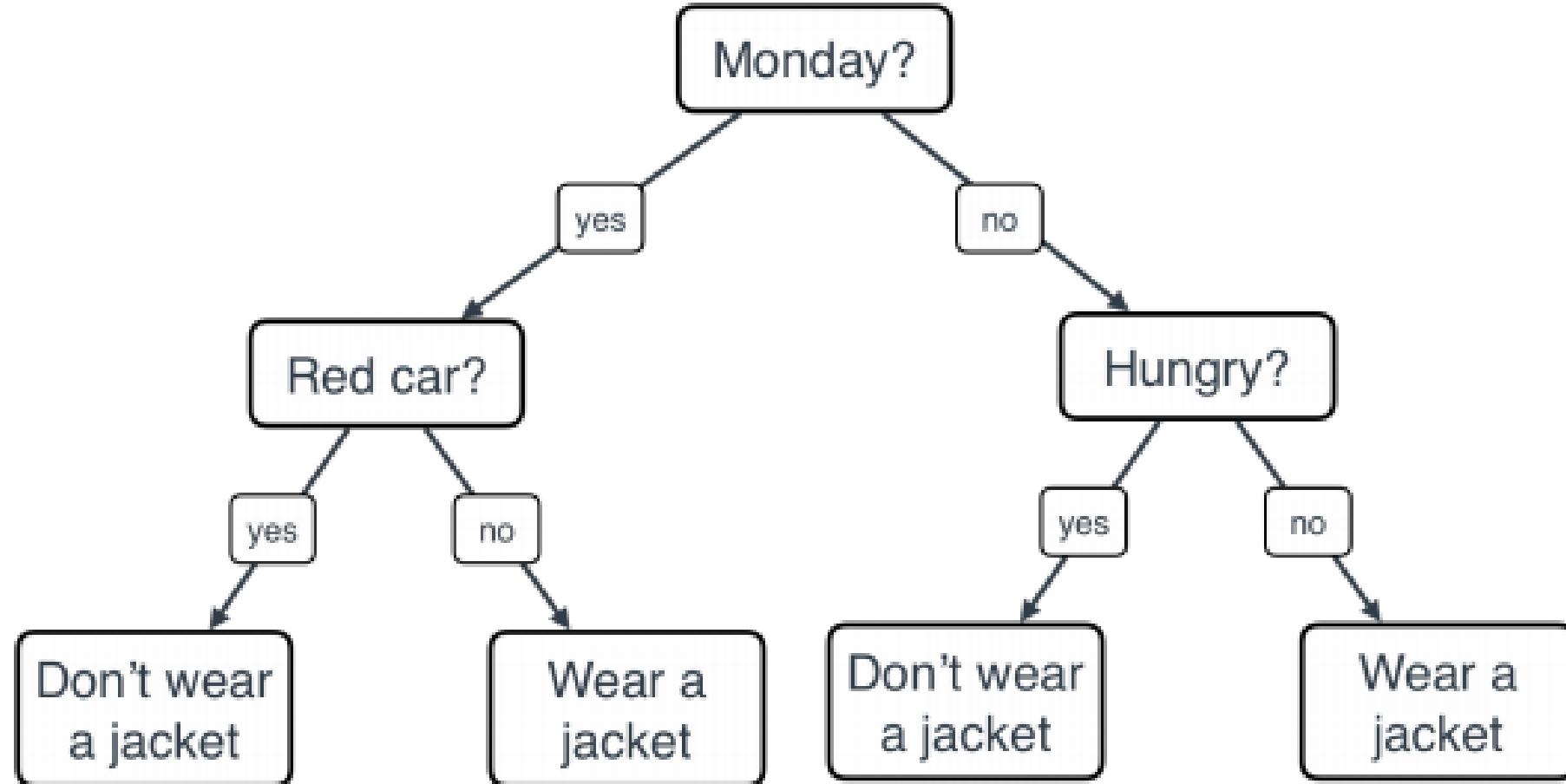
Splitting data by asking questions: Decision trees



Splitting data by asking questions: Decision trees



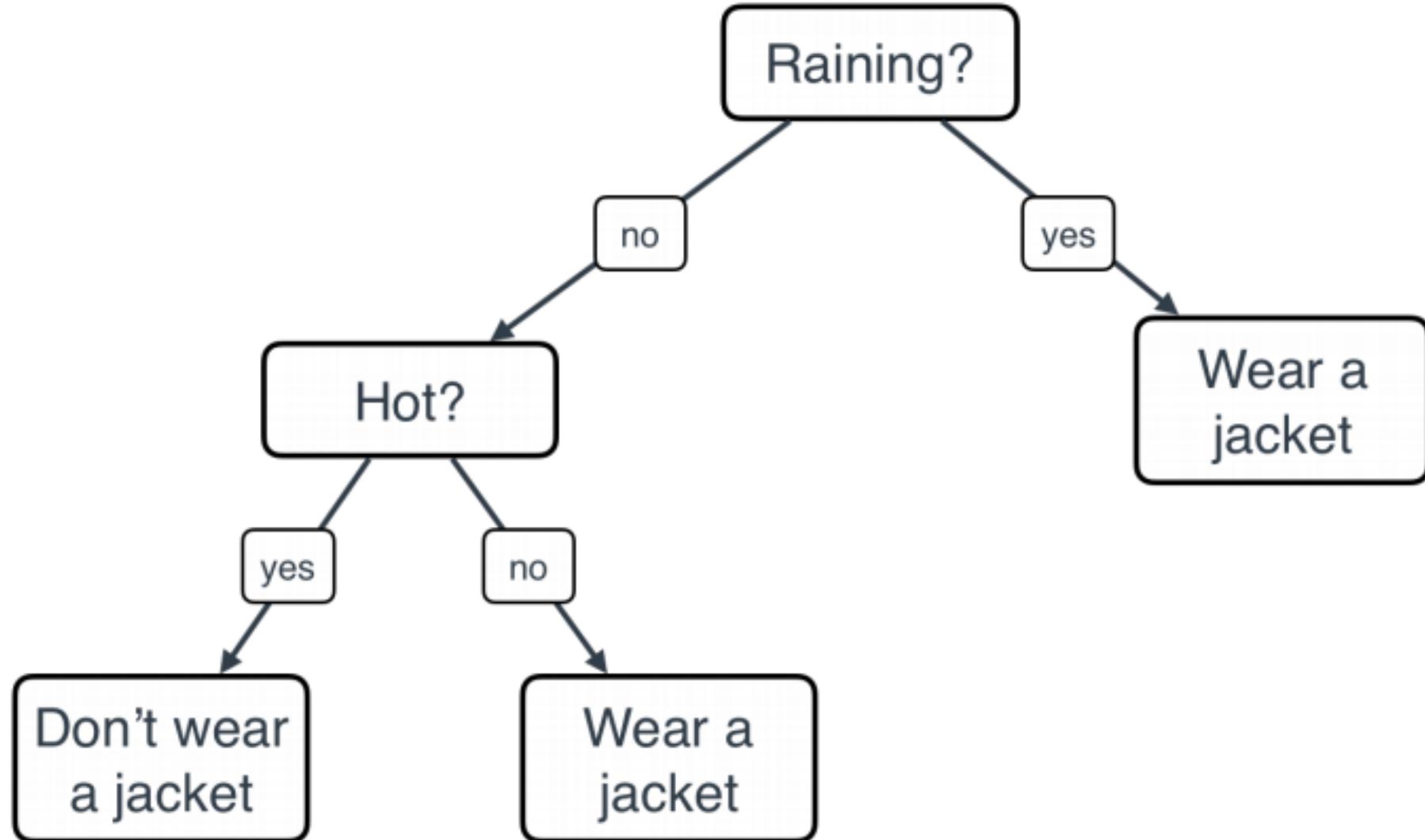
Splitting data by asking questions: Decision trees



Splitting data by asking questions: Decision trees



Splitting data by asking questions: Decision trees



The problem: We need to recommend apps to users according to what they are likely to download



Atom count

App for counting the number of atoms in your body



Beehive Finder

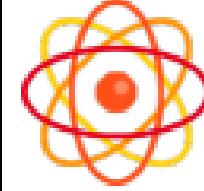
App for locating the nearest beehives to your location



Check Mate Mate

App for finding Australian chess players in your area

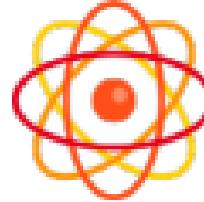
The problem: We need to recommend apps to users according to what they are likely to download

Gender	Age	App
F	15	 Atom Count
F	25	 Check Mate Mate

The problem: We need to recommend apps to users according to what they are likely to download

M	32	 Beehive Finder
F	35	 Check Mate Mate
M	12	 Atom Count
M	14	 Atom Count

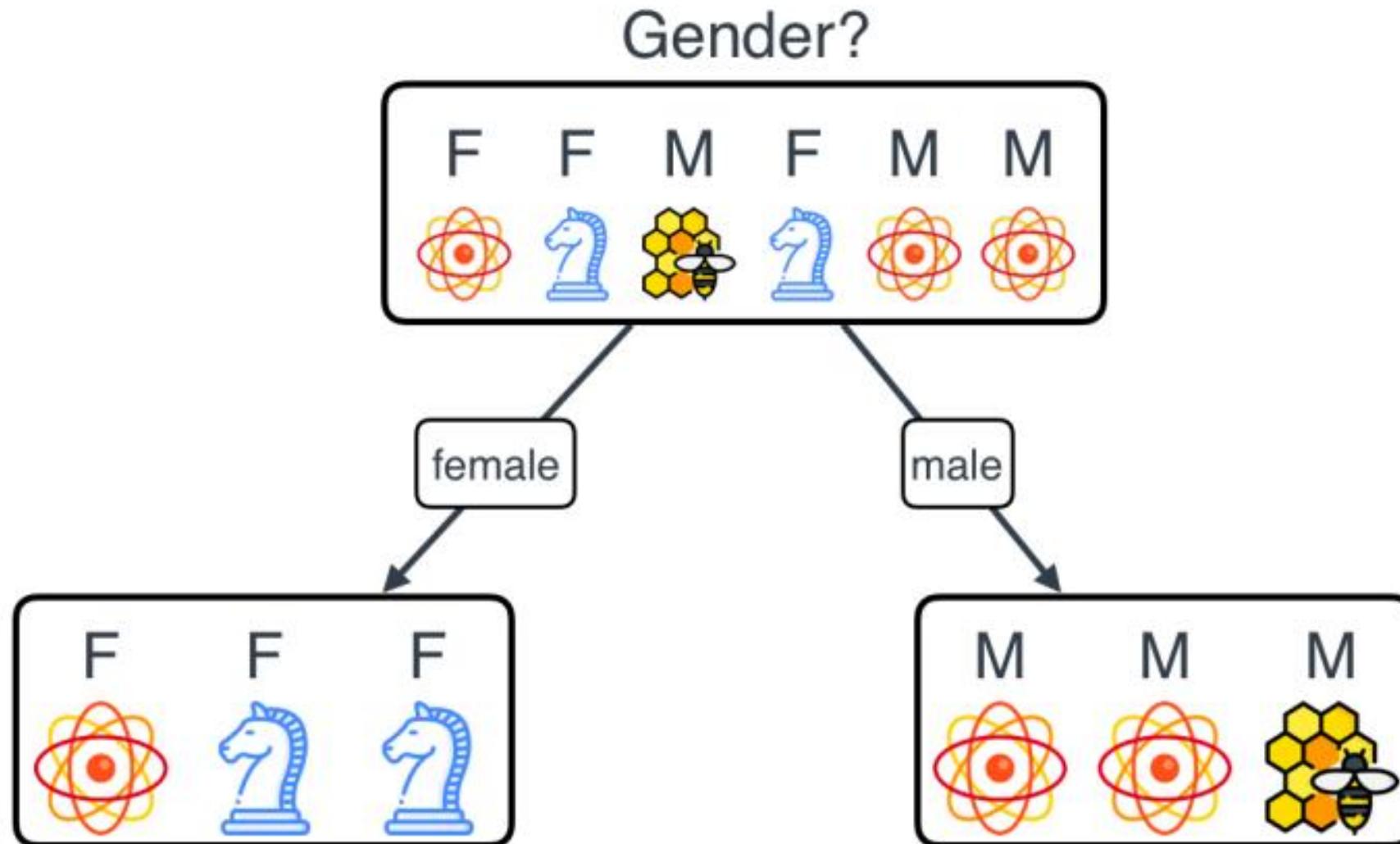
First step to build the model: Asking the best question

Gender	Age	App
F	young	 Atom Count
F	adult	 Check Mate Mate

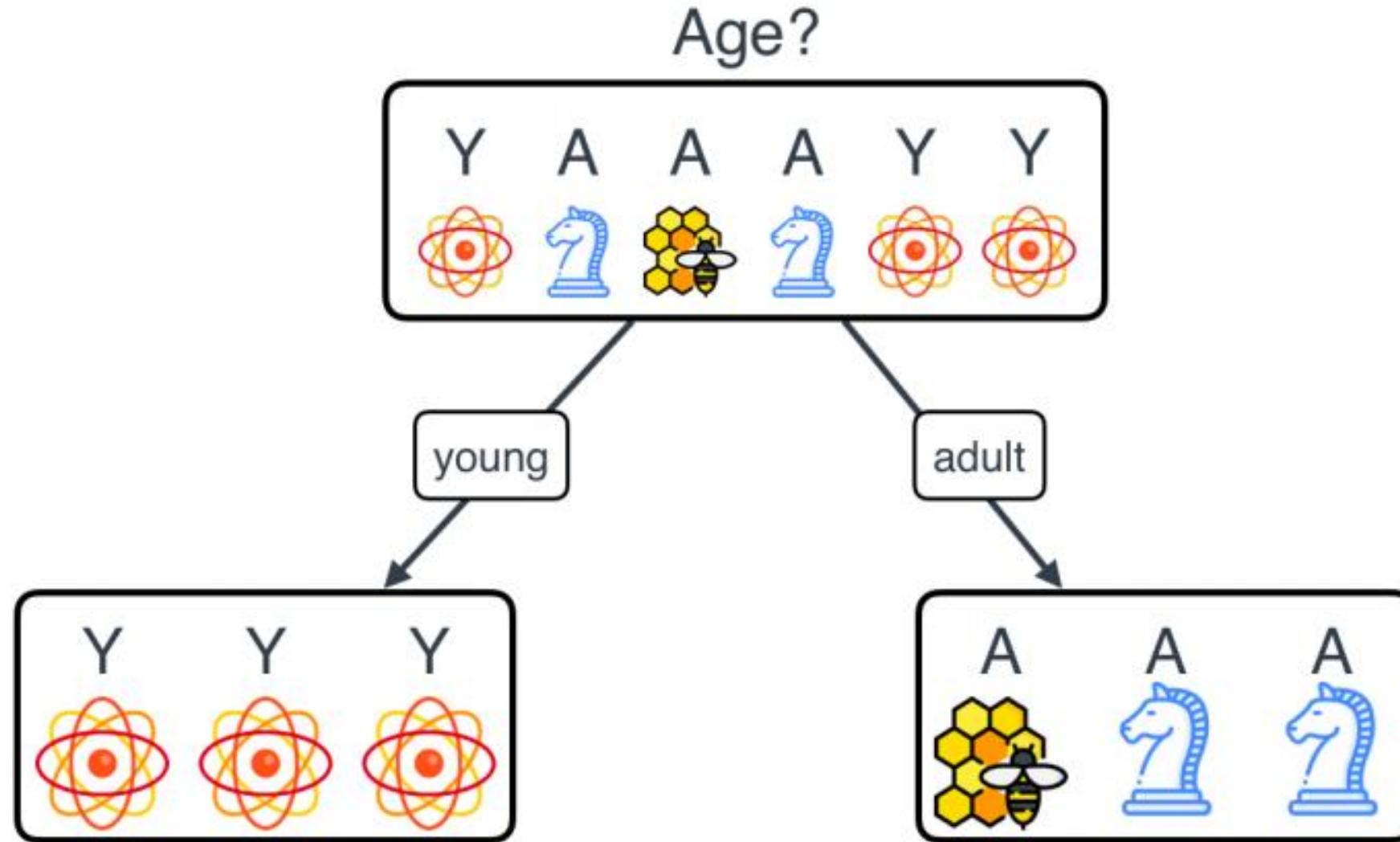
First step to build the model: Asking the best question

M	adult	 Beehive Finder
F	adult	 Check Mate Mate
M	young	 Atom Count
M	young	 Atom Count

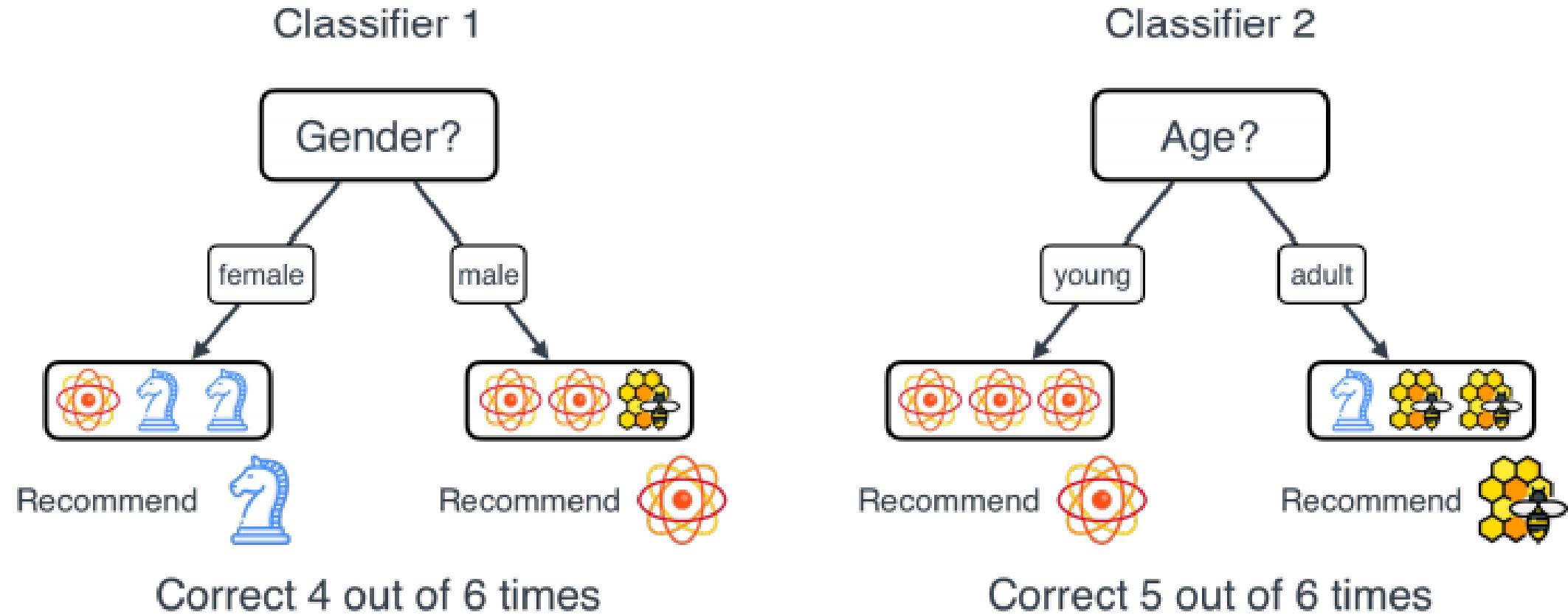
First step to build the model: Asking the best question



First step to build the model: Asking the best question



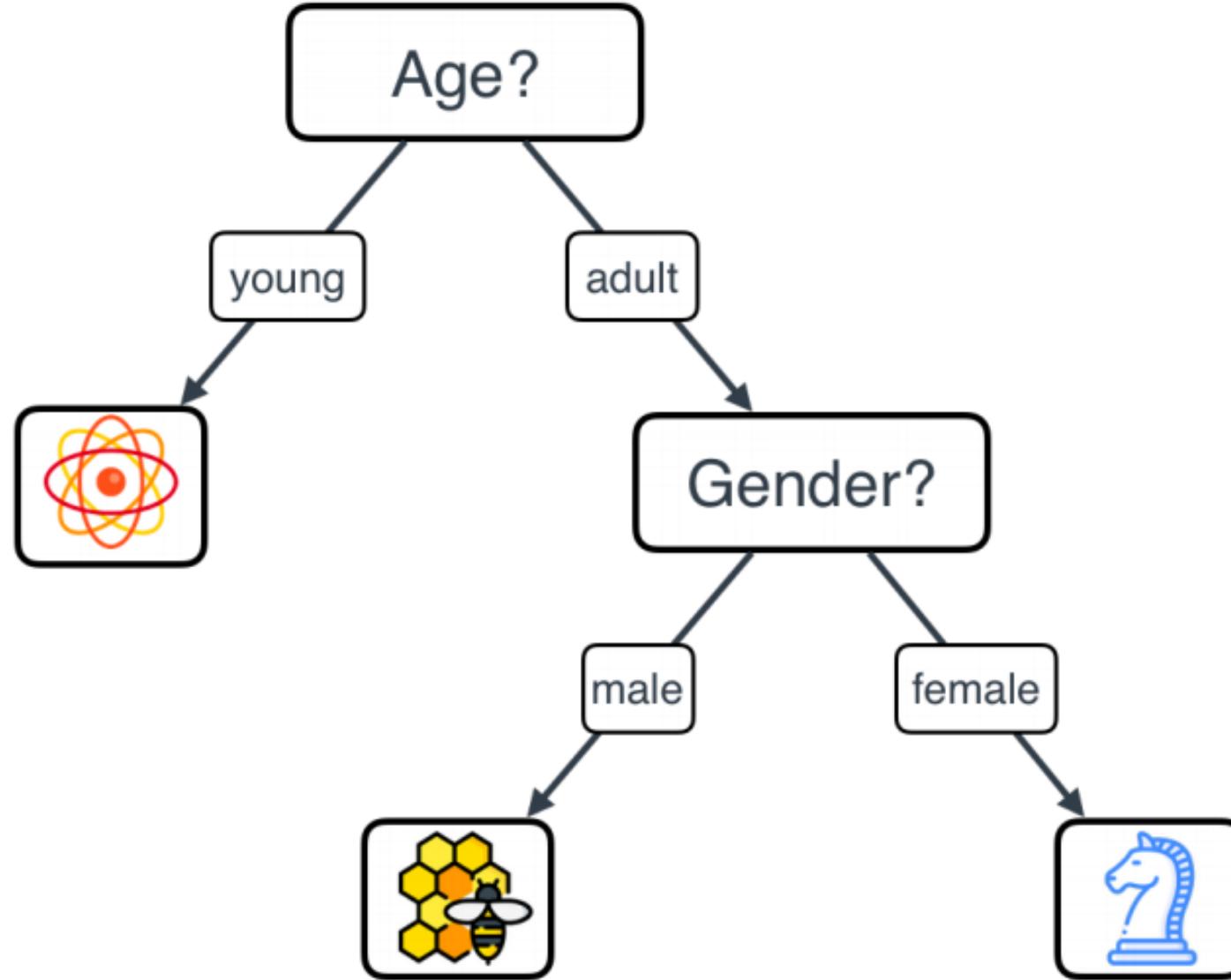
First step to build the model: Asking the best question



Next and final step: Iterate by asking the best question every time

Gender	Age	App
F	adult	 Beehive Finder
M	adult	 Check Mate Mate
F	adult	 Beehive Finder

Final tree



Next and final step: Iterate by asking the best question every time

- And we're done! This is our decision tree.
- Here is the pseudocode for the predictions that this model makes.

```
predict(user):
    if user is young:
        recommend Atom Count
    else:
        if user is female:
            recommend Check Mate Mate
        else:
            recommend Beehive Finder
```

Building the tree: How to pick the right feature to split

Gender	Age	Location	App
Female	Young	A	1
Male	Young	A	1
Female	Young	A	1
Male	Adult	A	1

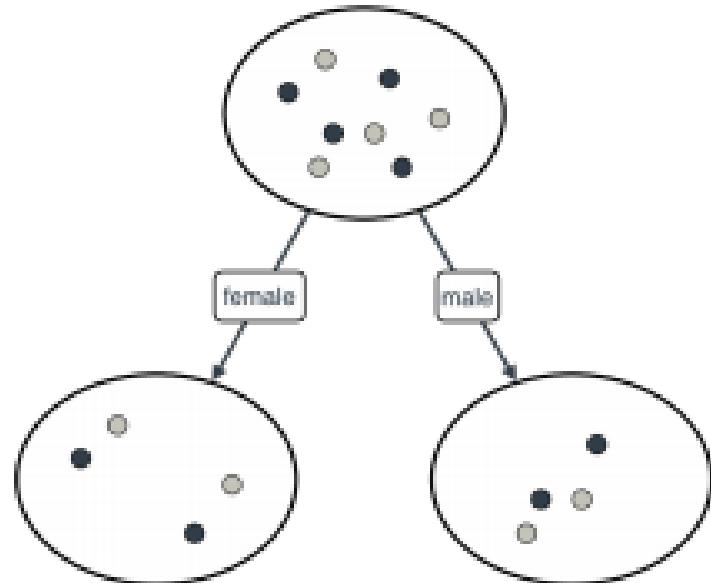
Building the tree: How to pick the right feature to split

Female	Young	B	2
Male	Adult	B	2
Female	Adult	B	2
Male	Adult	B	2

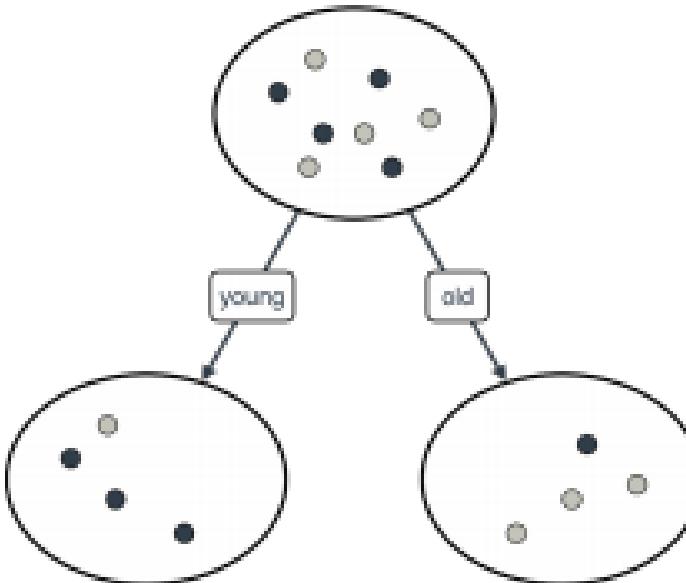
Building the tree: How to pick the right feature to split



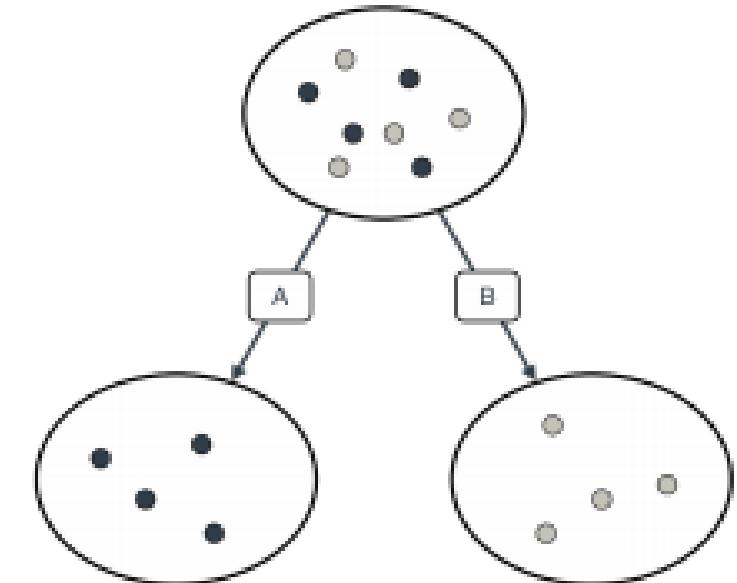
Split on gender



Split on age

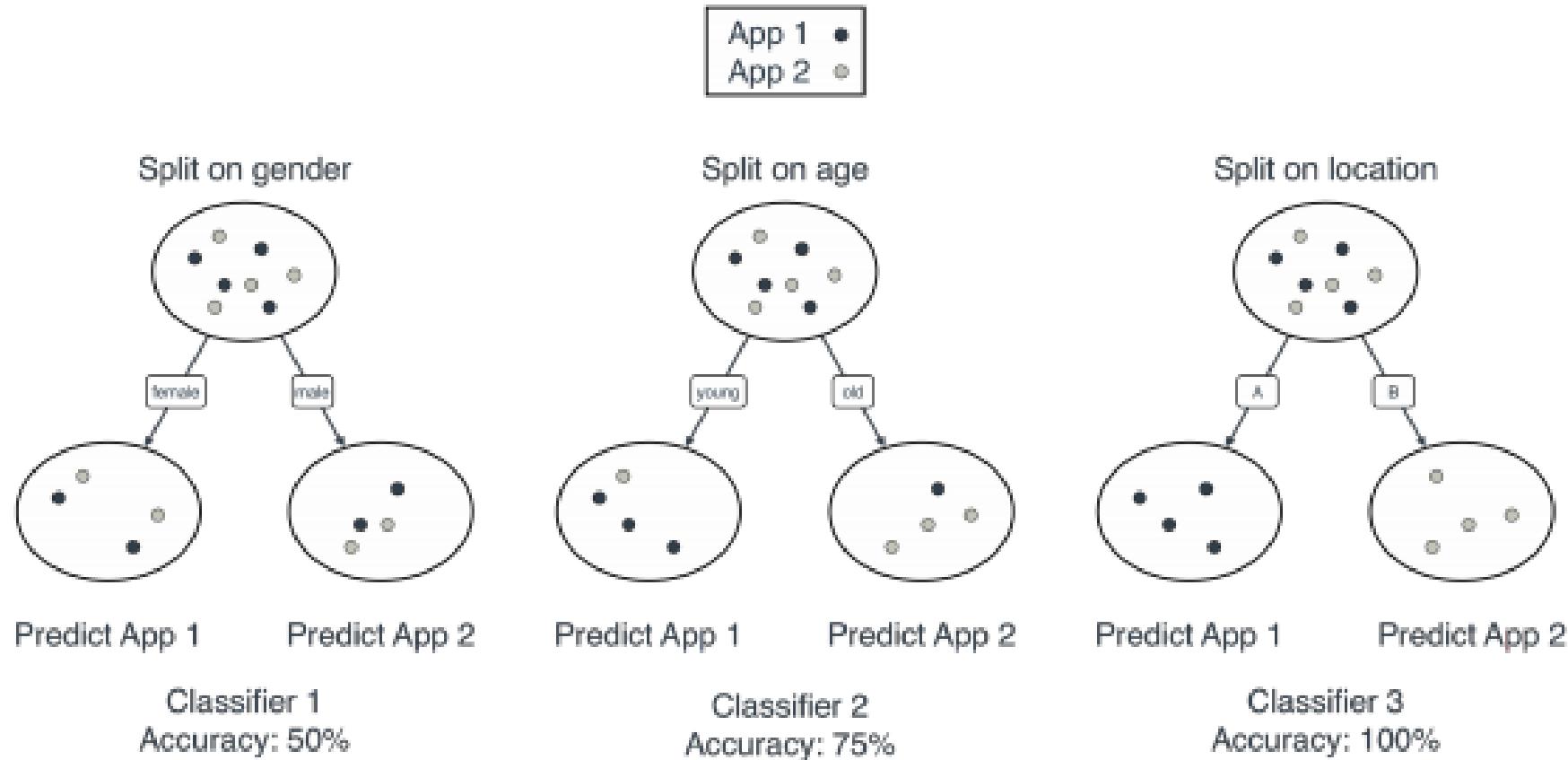


Split on location



How to pick the best feature to split our data: Accuracy

- The three classifiers, together with their accuracies, are shown here.



How to pick the best feature to split our data: Gini impurity



Low Gini
impurity index



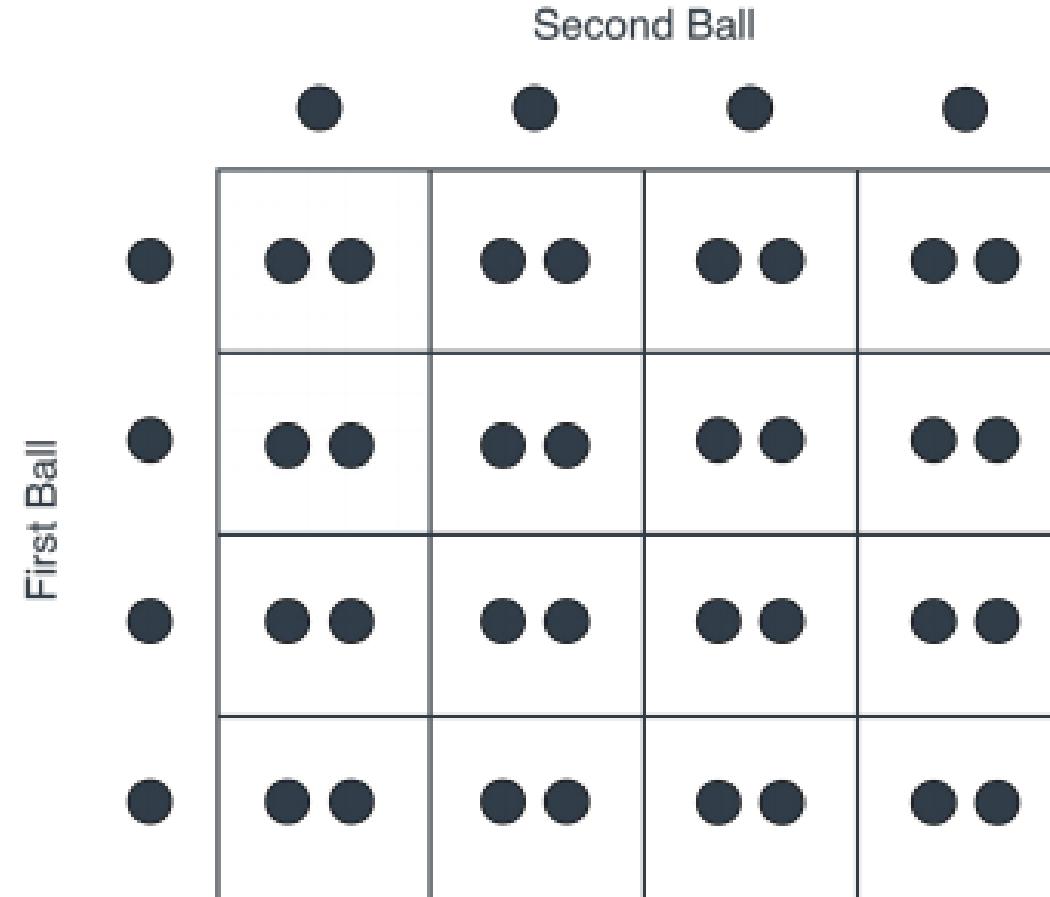
High Gini
impurity index

How to pick the best feature to split our data: Gini impurity

- Set 1: {black, black, black, black}
- Set 2: {black, black, black, white}
- Set 3: {black, black, white, white}
- Set 4: {black, white, white, white}
- Set 5: {white, white, white, white}

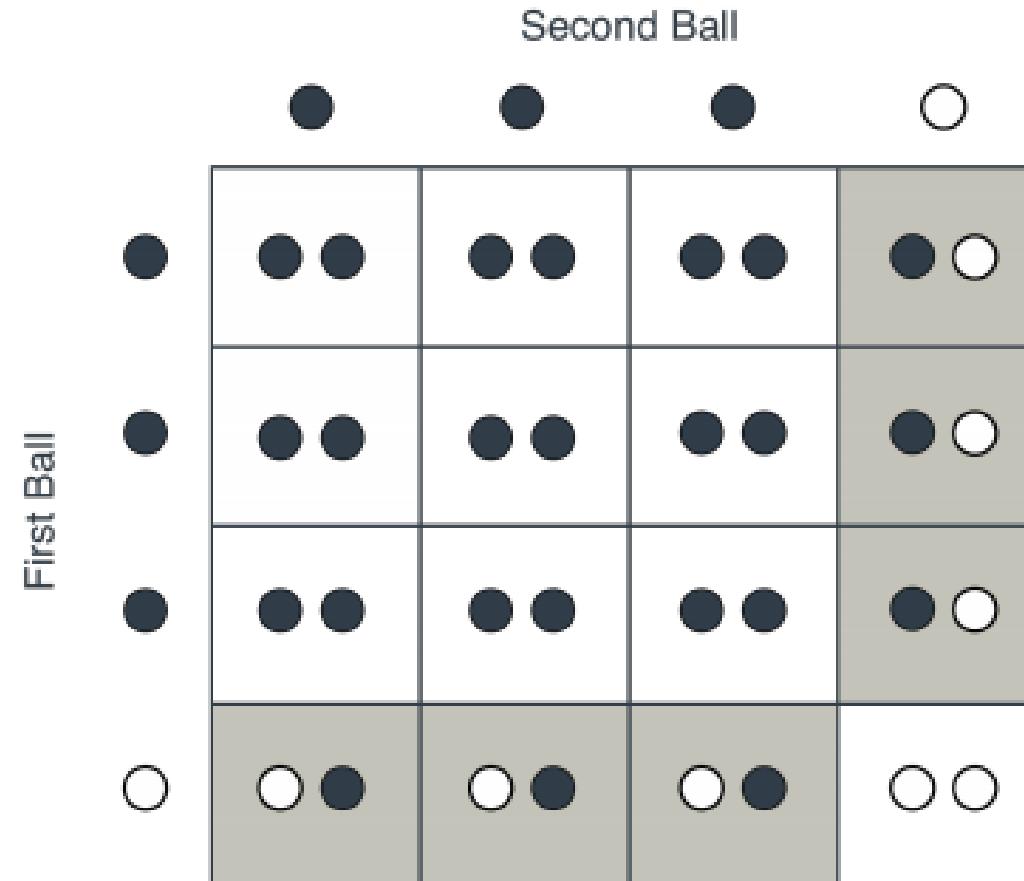
How to pick the best feature to split our data: Gini impurity

Set 1:



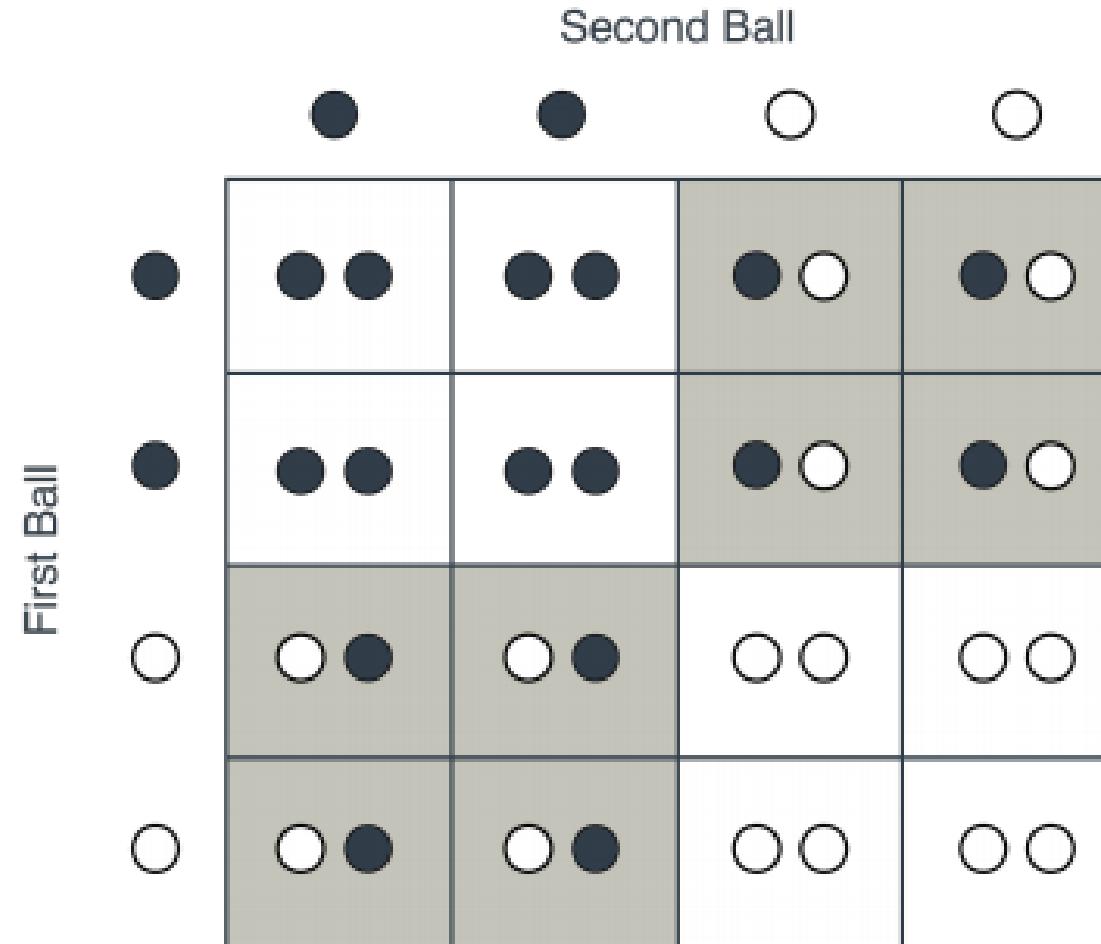
How to pick the best feature to split our data: Gini impurity

Set 2



How to pick the best feature to split our data: Gini impurity

Set 3



How to pick the best feature to split our data: Gini impurity

Set 1



Gini impurity index = 0

Set 2



Gini impurity index = 0.375

Set 3



Gini impurity index = 0.5

Set 4



Gini impurity index = 0.375

Set 5



Gini impurity index = 0

How to pick the best feature to split our data: Gini impurity

$$\begin{aligned}\text{Gini Impurity Index} &= P(\text{picking two balls of different color}) \\ &= 1 - P(\text{picking two balls of the same color}) \\ &= 1 - p_1^2 - p_2^2 - \dots - p_N^2\end{aligned}$$

The diagram illustrates the components of the Gini impurity formula. Three arrows point from the terms p_1^2 , p_2^2 , and p_N^2 in the equation to labels below: "P(Both balls are color 1)", "P(Both balls are color 2)", and "P(Both balls are color N)" respectively.

How to pick the best feature to split our data: Gini impurity

- Picking blue: $p_1 = \frac{3}{6} = \frac{1}{2}$.
- Picking red: $p_2 = \frac{2}{6} = \frac{1}{3}$.
- Picking yellow: $p_3 = \frac{1}{6}$.

Thus, the Gini impurity index is precisely $1 - p_1^2 - p_2^2 = 1 - (\frac{1}{2})^2 - (\frac{1}{3})^2 - (\frac{1}{6})^2 = \frac{22}{36}$.



How to pick the best feature to split our data: Gini impurity

Gini Impurity Index = P(picking two balls of different color)

$$= 1 - \left(\frac{3}{6}\right)^2 - \left(\frac{2}{6}\right)^2 - \left(\frac{1}{6}\right)^2$$

P(Both balls are black)

P(Both balls are grey)

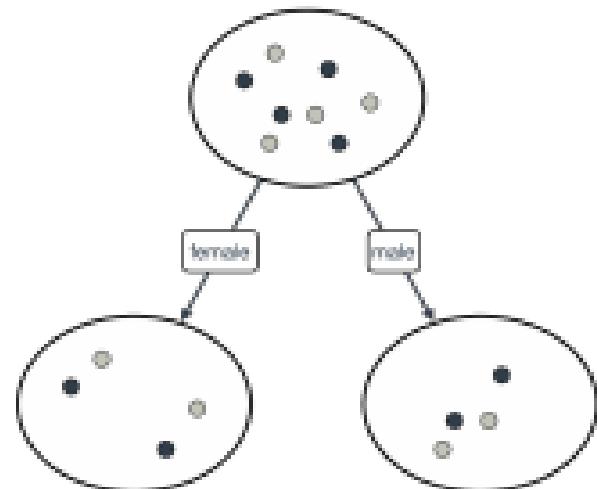
P(Both balls are white)

How to pick the best feature to split our data: Gini impurity

App 1 •
App 2 •

Split on gender

Gini = 0.5

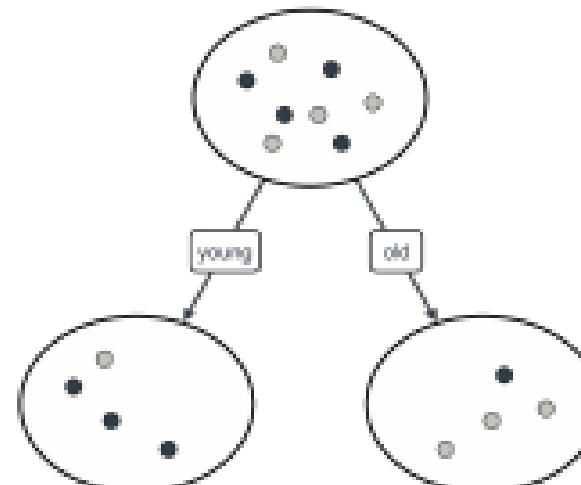


Gini = 0.5

Average Gini = 0.5
Gini gain = 0

Split on age

Gini = 0.5

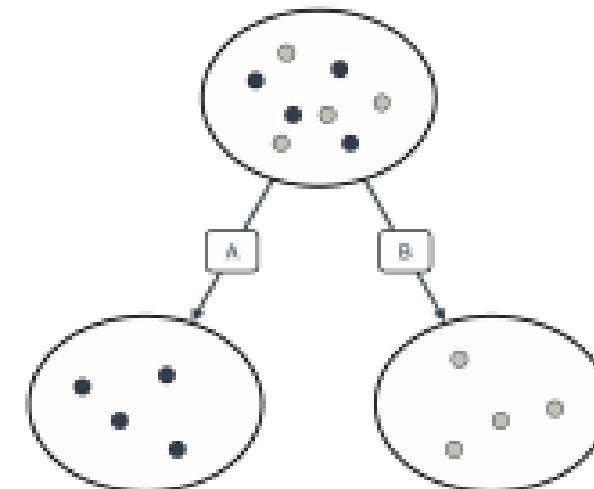


Gini = 0.375

Average Gini = 0.375
Gini gain = 125

Split on location

Gini = 0.5

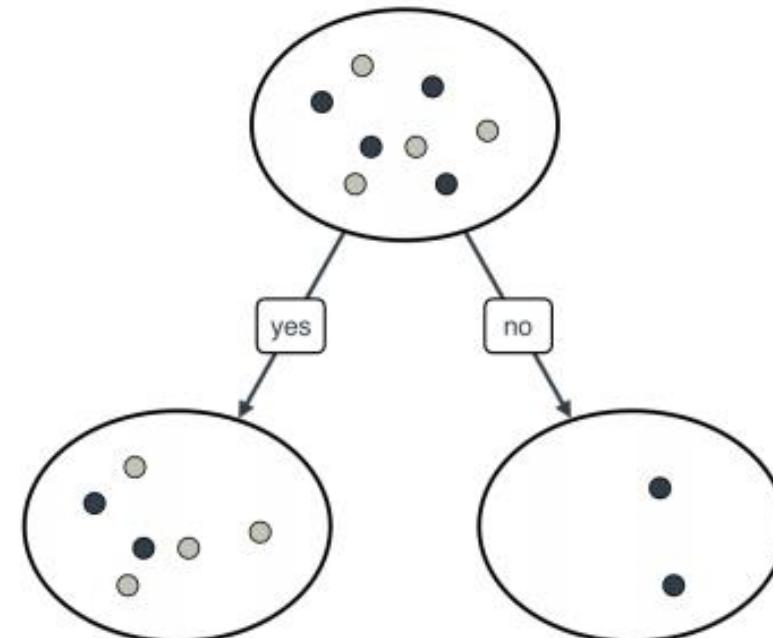


Gini = 0

Average Gini = 0
Gini gain = 0.5

How to pick the best feature to split our data: Gini impurity

WEIGHTED GINI IMPURITY

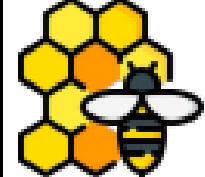


Gini impurity = 0.375

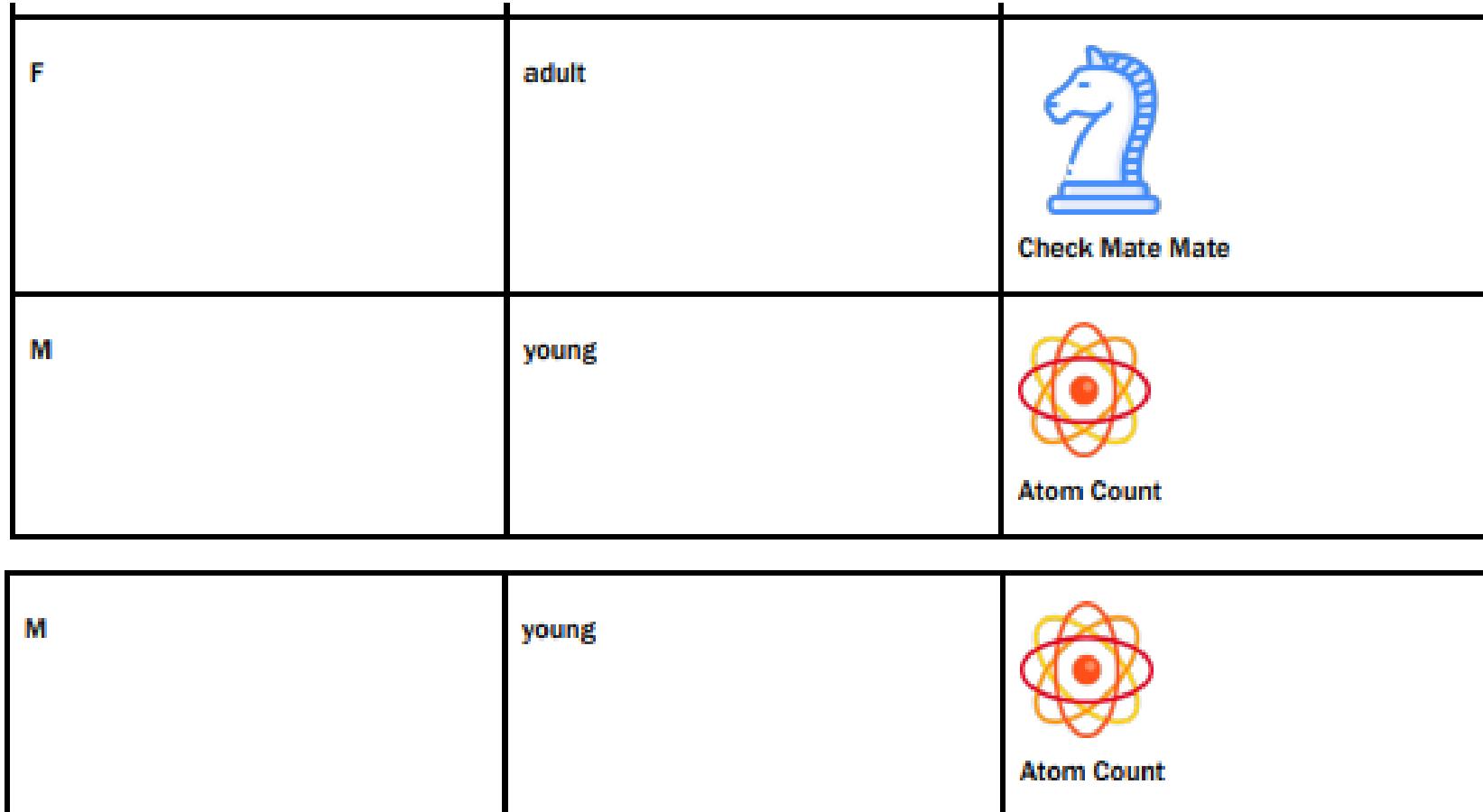
Gini impurity = 0

$$\text{Weighted average} = 0.375 \left(\frac{6}{8} \right) + 0 \left(\frac{2}{8} \right) = 0.25$$

Back to recommending apps: Building our decision tree using Gini index

Gender	Age	App
F	young	 Atom Count
F	adult	 Check Mate Mate
M	adult	 Beehive Finder

Back to recommending apps: Building our decision tree using Gini index



Back to recommending apps: Building our decision tree using Gini index

- Using the formula that we learned in the previous section, the Gini impurity index for this set is:

$$\text{Gini impurity index of } \{A, C, B, C, A, A\} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{6}\right)^2 - \left(\frac{1}{3}\right)^2 = 0.611$$

Back to recommending apps: Building our decision tree using Gini index

As we've seen before, splitting by gender gives us the two following sets:

- Females: {A, C, C}
- Males: {B, A, A}

The Gini impurity indices of these two are the following:

- Females: $1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.444$
- Males: $1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.444$

Therefore, the average Gini impurity index of this splitting is 0.611. Since the Gini impurity of the root was 0.444, we conclude that the Gini gain is:

$$\text{Gini gain} = 0.611 - 0.444 = 0.167.$$

Back to recommending apps: Building our decision tree using Gini index

SPLITTING BY AGE

The Gini impurity indices of these two are the following:

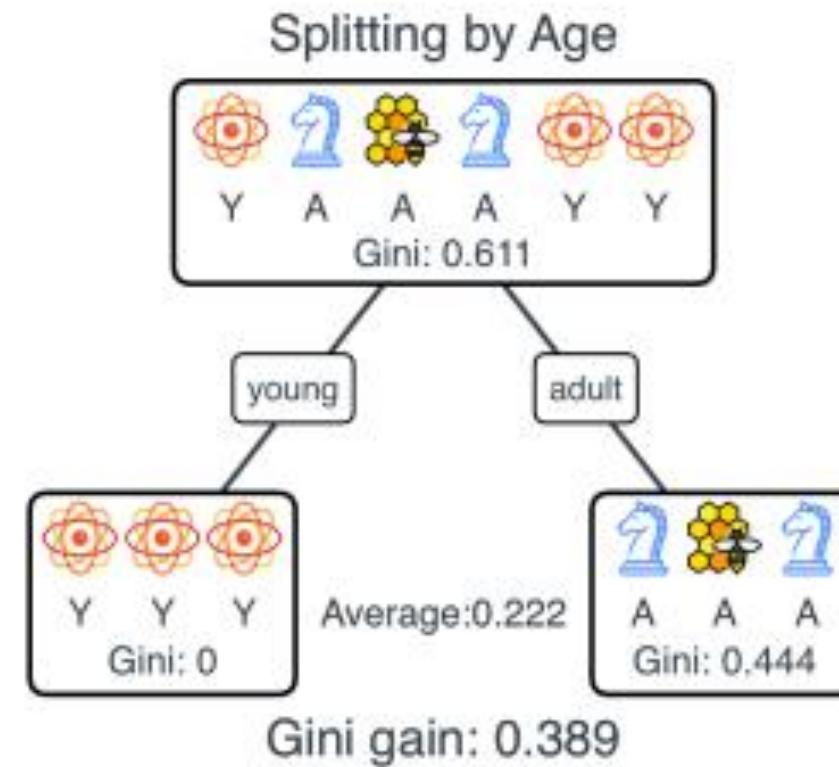
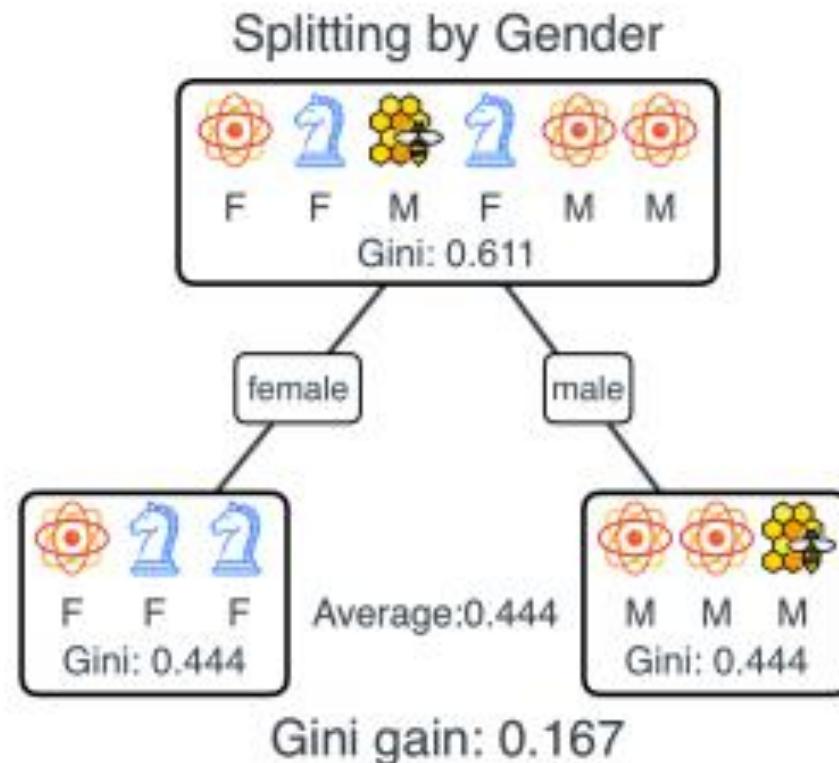
- Young: $1 - \left(\frac{3}{3}\right)^2 = 0$
- Adults: $1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = 0.444$

Therefore, the average Gini impurity index of this splitting is 0.222 (the average of 0 and 0.444). Since the Gini impurity of the root was 0.611, we conclude that the Gini gain is:

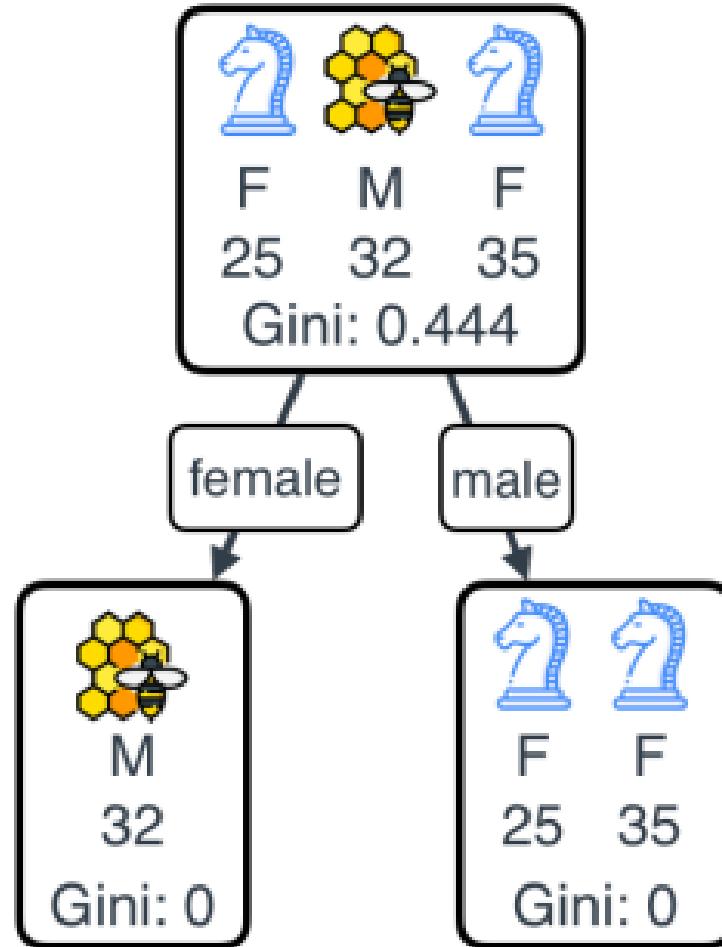
$$\text{Gini gain} = 0.611 - 0.222 = 0.389.$$

Back to recommending apps: Building our decision tree using Gini index

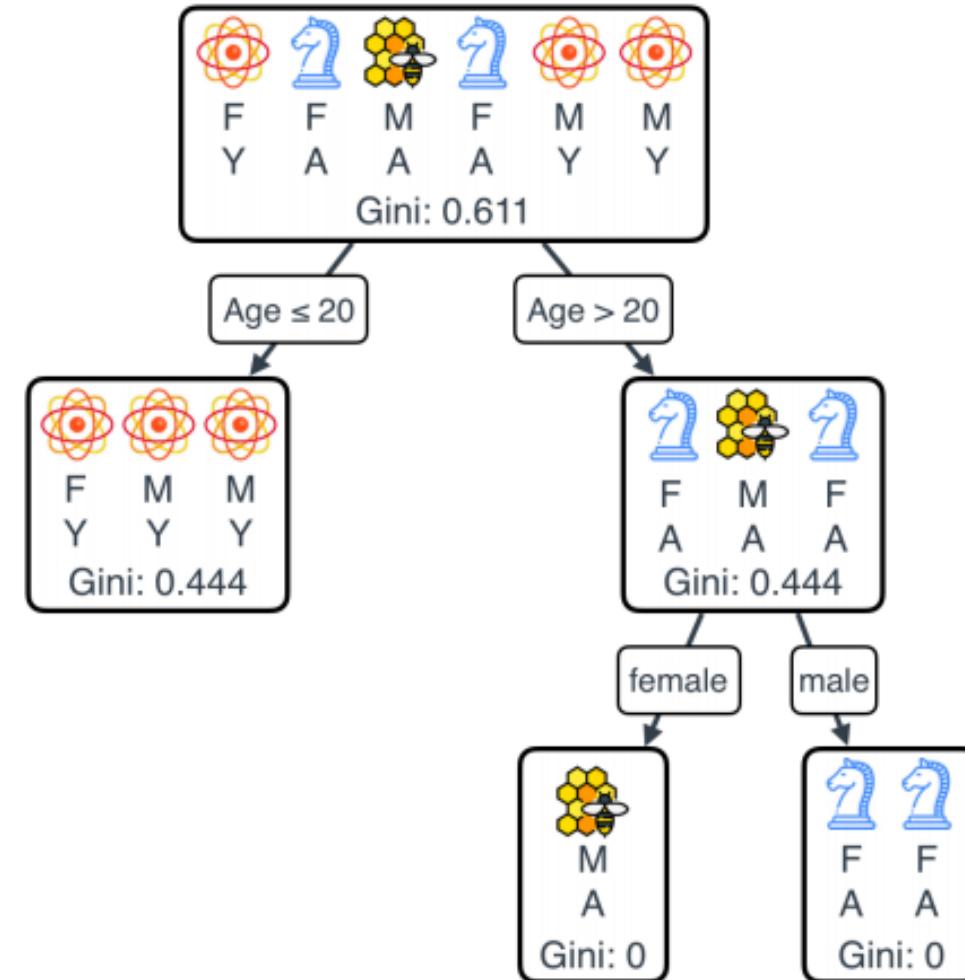
COMPARING GINI INDICES



Back to recommending apps: Building our decision tree using Gini index



Back to recommending apps: Building our decision tree using Gini index



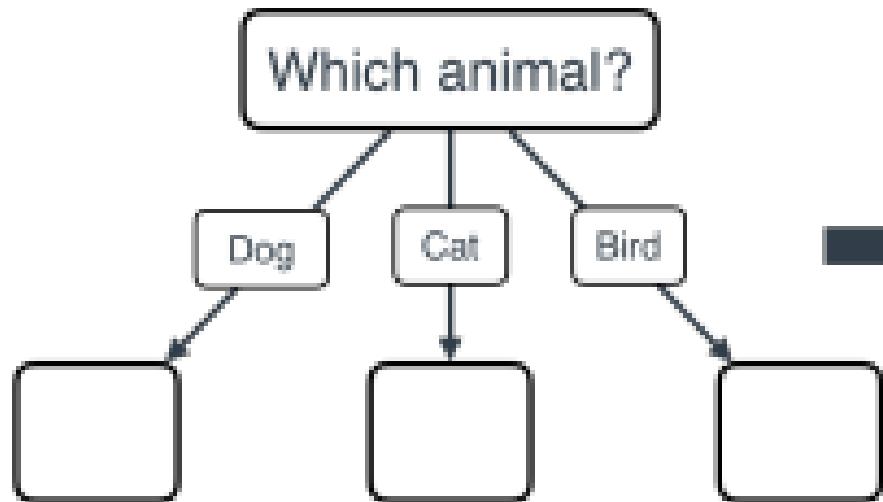
Features with more categories, such as Dog/Cat/Bird

When the feature has more classes, such as Dog/Cat/Bird, we simply use more binary (yes/no) questions. In this case, we would ask the following:

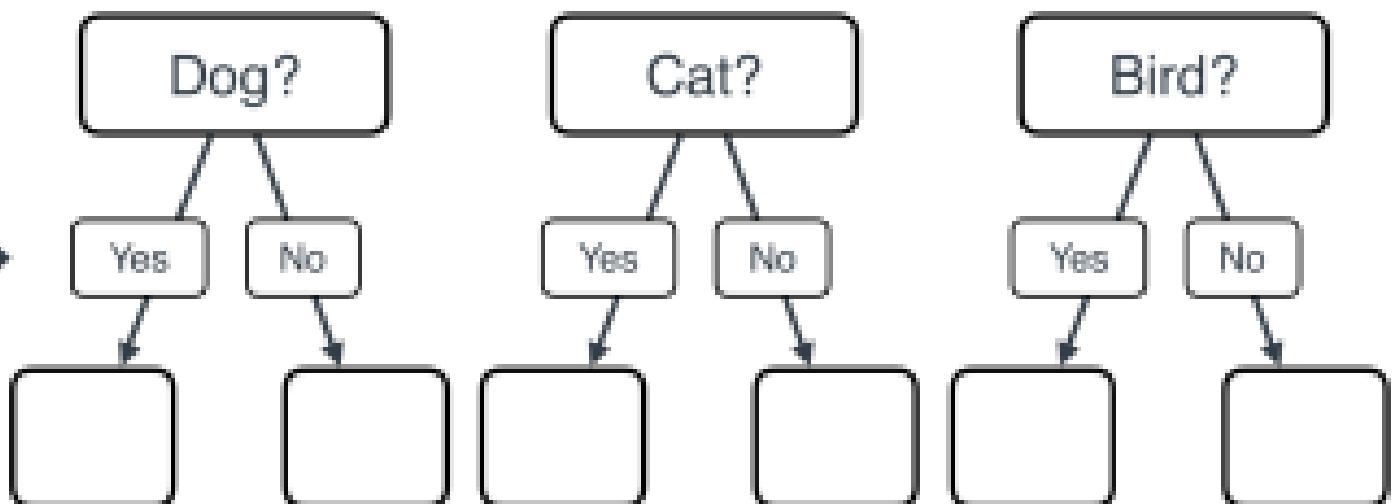
- Does the user own a dog?
- Does the user own a cat?
- Does the user own a bird?

Features with more categories, such as Dog/Cat/Bird

Non-binary feature



More binary features



Features with more categories, such as Dog/Cat/Bird

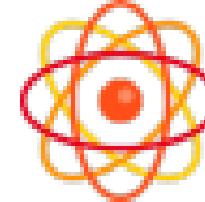
Animal
Dog
Cat
Bird
Dog
Bird

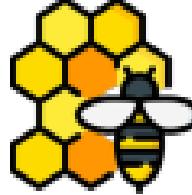
One-hot
encoding



Dog?	Cat?	Bird?
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1

Continuous features, such as a number

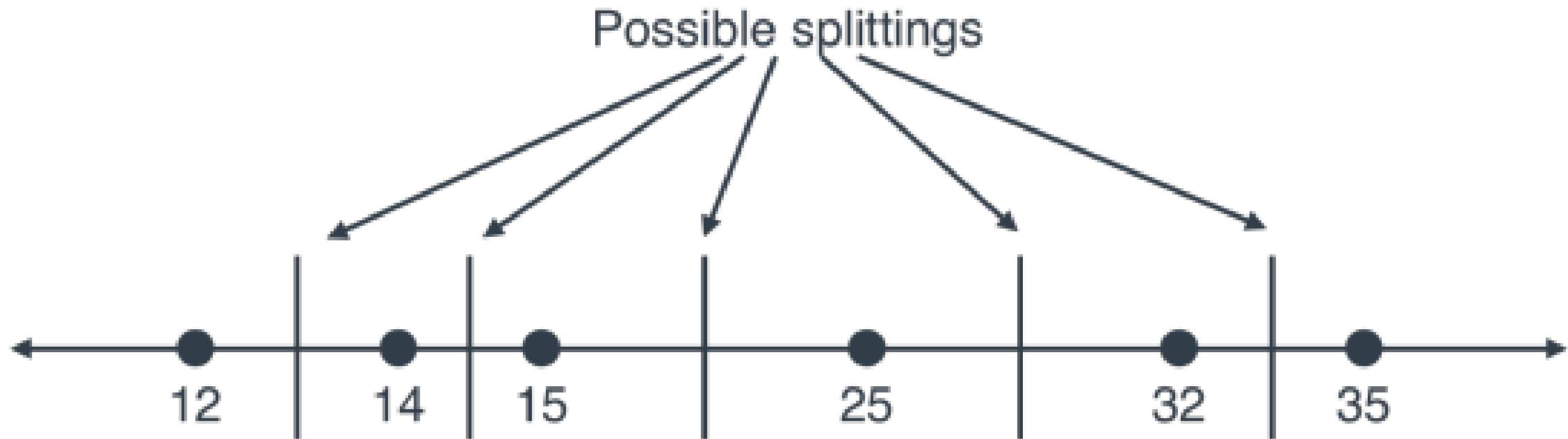
Gender	Age	App
F	15	 Atom Count
F	25	

		Check Mate Mate
M	32	 Beehive Finder
F	35	 Check Mate Mate
M	12	 Atom Count
M	14	 Atom Count

Continuous features, such as a number

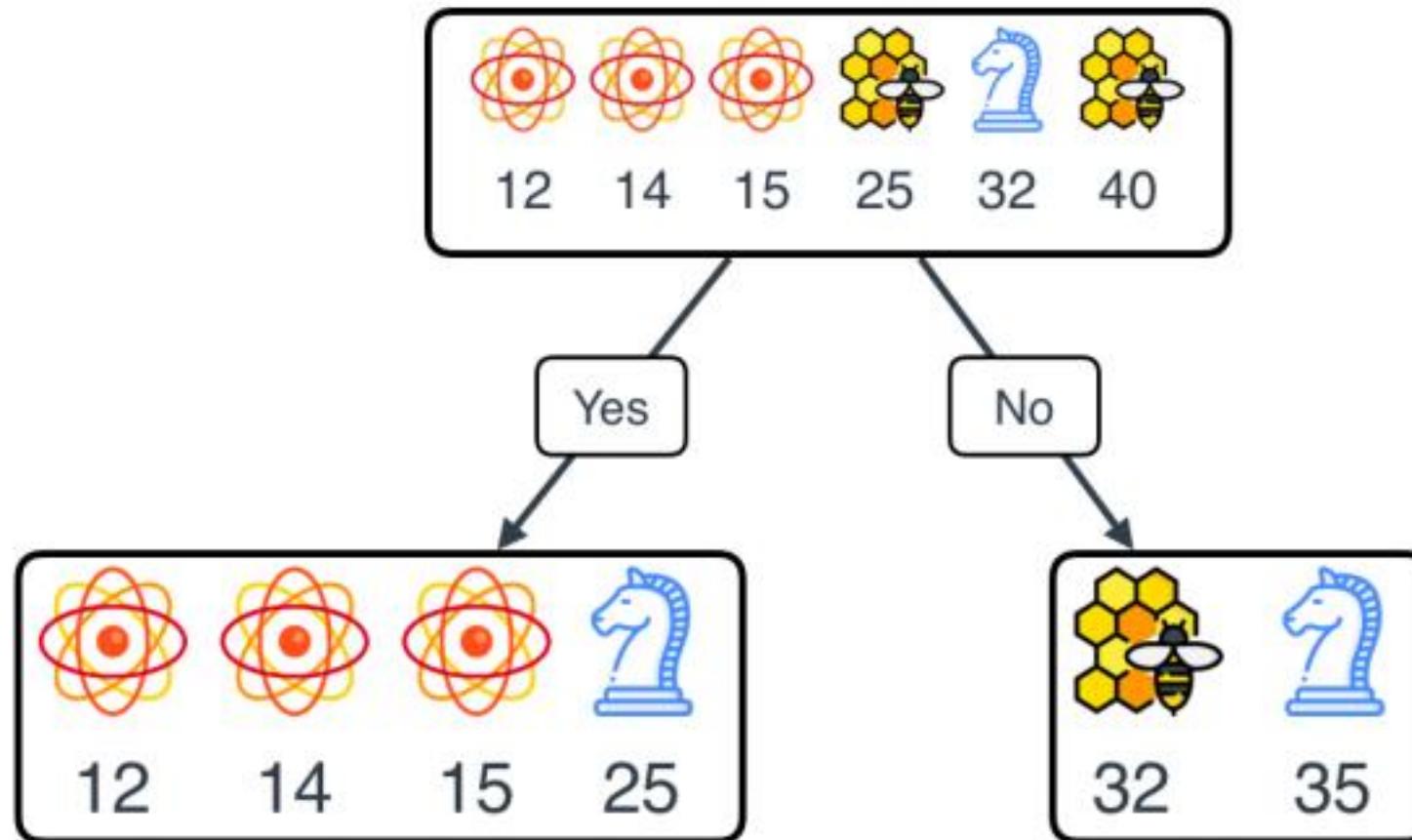
Question	First set	Second set
Is the user younger than 7?	empty	12, 14, 15, 25, 32, 35
Is the user younger than 13?	12	14, 15, 25, 32, 35
Is the user younger than 14.5?	12, 14	15, 25, 32, 35
Is the user younger than 20?	12, 14, 15	25, 32, 35
Is the user younger than 27?	12, 14, 15, 25	32, 35
Is the user younger than 33?	12, 14, 15, 25, 32	35
Is the user younger than 100?	12, 14, 15, 25, 32, 35	empty

Continuous features, such as a number



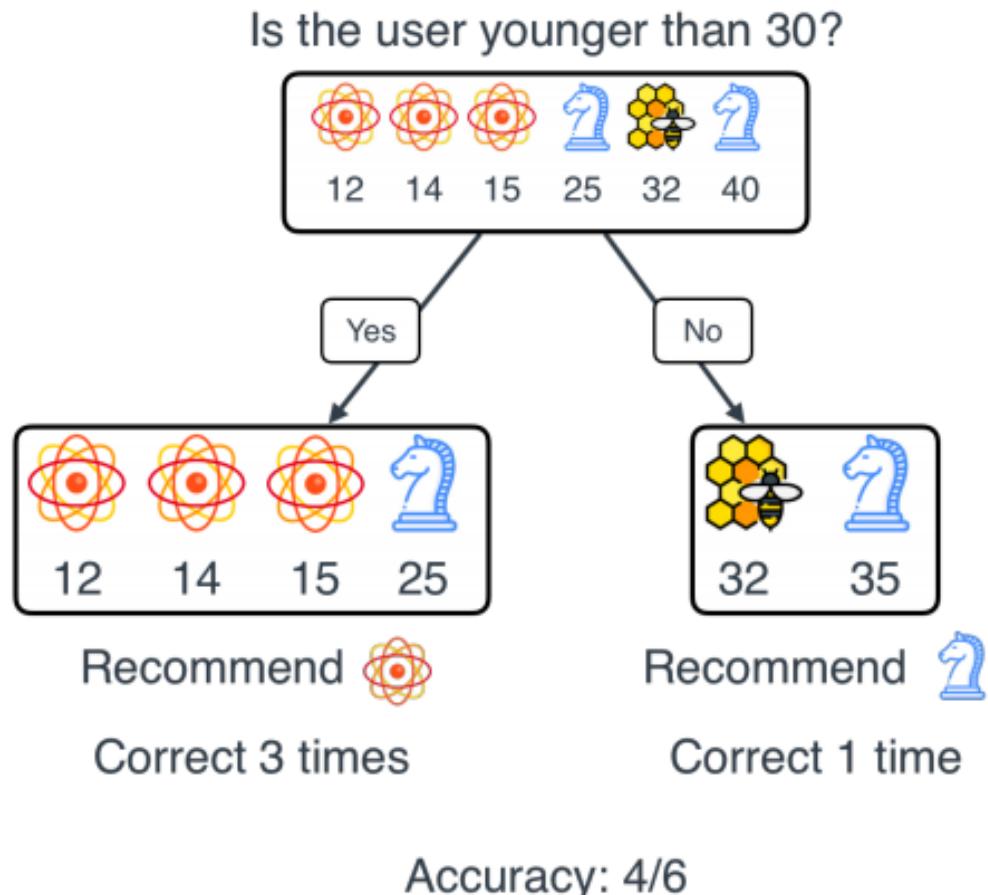
Continuous features, such as a number

Is the user younger than 30?



Continuous features, such as a number

COMPUTING THE ACCURACY OF THIS SPLIT



Continuous features, such as a number

- The left leaf has three users downloading Atom Count, and one downloading Check Mate Mate, so the Gini impurity is

$$1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

- The right leaf has one user downloading Instagram, and one downloading Check Mate Mate, so the Gini impurity is

$$1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

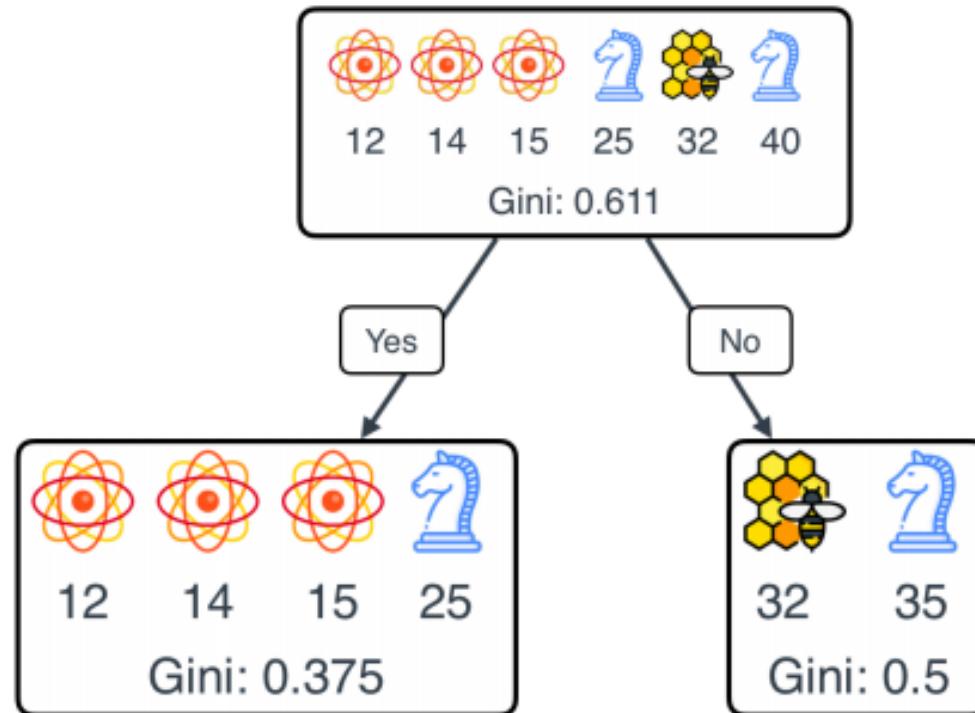
Continuous features, such as a number

- Also note that the left leaf has four users and the right leaf has two users.
- Since there are six users in total, then the weights for the entropies of the left and right leaves is $4/6$ and $2/6$, respectively (or $\frac{2}{3}$ and $\frac{1}{3}$).
- Thus, our weighted Gini impurity at the leaves is

$$\frac{4}{6} \cdot 0.375 + \frac{2}{6} \cdot 0.5 = 0.417$$

Continuous features, such as a number

Is the user younger than 30?



$$\text{Gini (weighted) average} = 0.375 \left(\frac{4}{6} \right) + 0.5 \left(\frac{2}{6} \right) = 0.417$$

$$\text{Gini gain} = 0.611 - 0.417 = 0.194$$

Continuous features, such as a number

Splitting	Labels	Accuracy	Gini gain
{12} {14, 15, 25, 32, 40}	A A, A, C, B, C	3/6	0.078
{12, 14} {15, 25, 32, 40}	A, A A, C, B, C	4/6	0.194
{12, 14, 15} {25, 32, 40}	A, A, A C, B, C	5/6	0.389
{12, 14, 15, 25} {32, 40}	A, A, A, C B, C	4/6	0.194
{12, 14, 15, 25, 32} {40}	A, A, A, C, B C	4/6	0.144

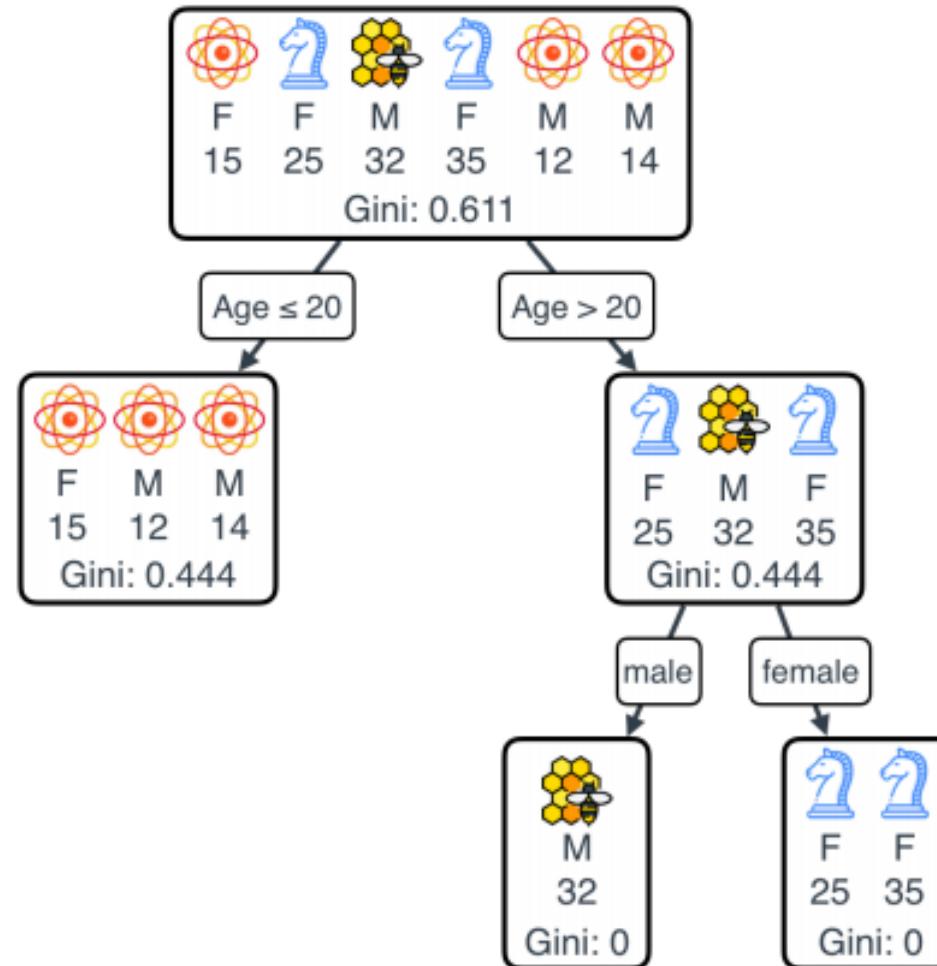
Continuous features, such as a number

Splitting	Labels	Accuracy	Gini impurity	Information gain
Male/Female	P, P, I P, W, W	0.666	0.444	

Continuous features, such as a number

```
predict(user):
    if user's age is less than or equal to 20:
        recommend Atom Count
    else:
        if user's gender is female:
            recommend Check Mate Mate
        else:
            recommend Beehive Finder
```

Continuous features, such as a number



Coding a decision tree with sklearn

```
import pandas as pd
app_dataset = pd.DataFrame({
    'Gender_Female':[1,1,0,1,0,0],
    'Gender_Male':[0,0,1,0,1,1],
    'Age': [15, 25, 32, 35, 12, 14],
    'App': ['Atom Count', 'Check Mate Mate', 'Beehive Finder', 'Check Mate Mate', 'Atom
        Count', 'Atom Count']})
print(app_dataset)
```

Coding a decision tree with sklearn

	Age	Gender_Female	Gender_Male	App
0	15	1	0	Atom count
1	25	1	0	Check Mate Mate
2	32	0	1	Beehive Finder
3	35	1	0	Check Mate Mate
4	12	0	1	Atom count
5	14	0	1	Atom count

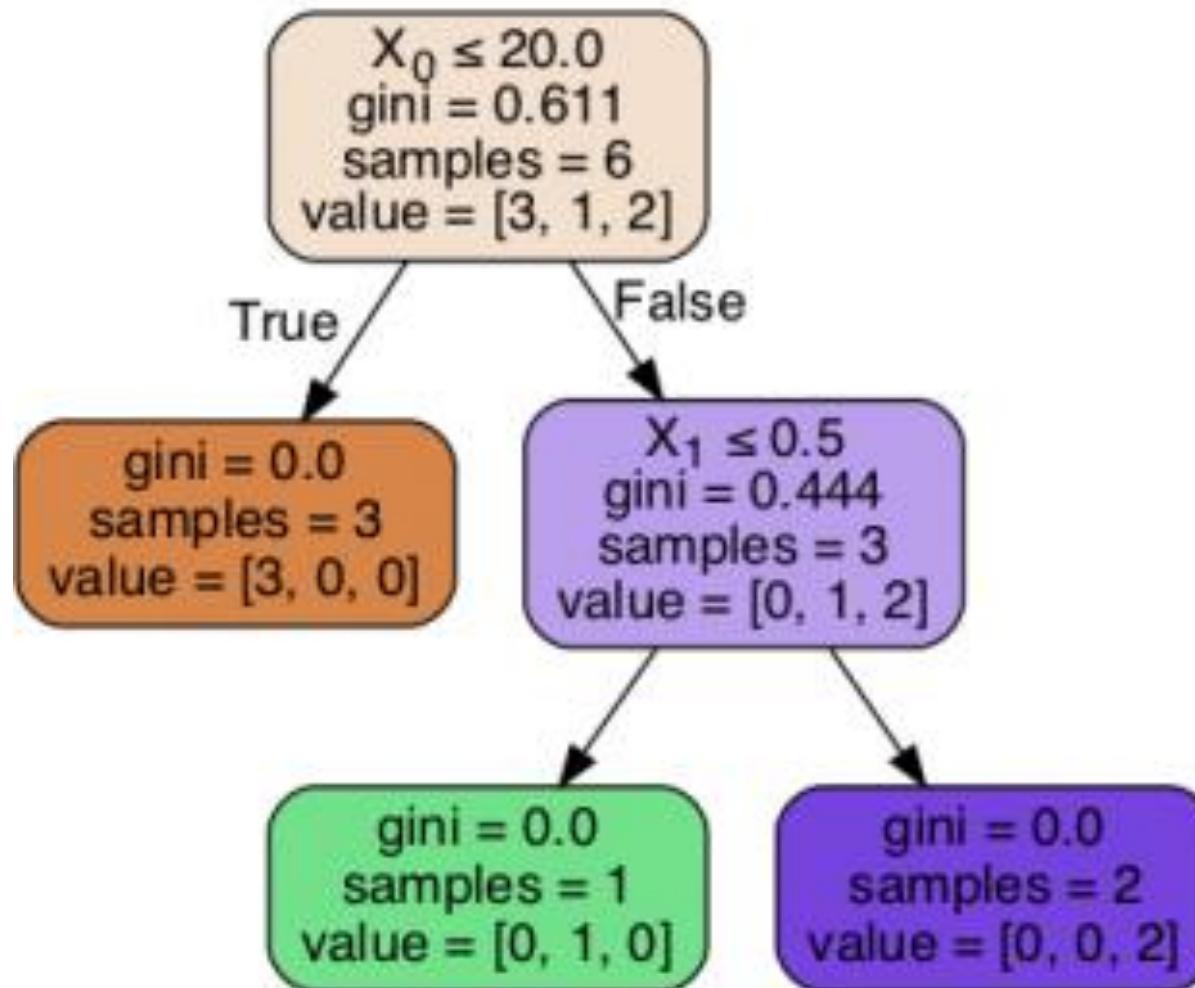
Coding a decision tree with sklearn

```
features = app_dataset_new[['Age', 'Gender_Female', 'Gender_Male']]  
labels = app_dataset_new[['App']]
```

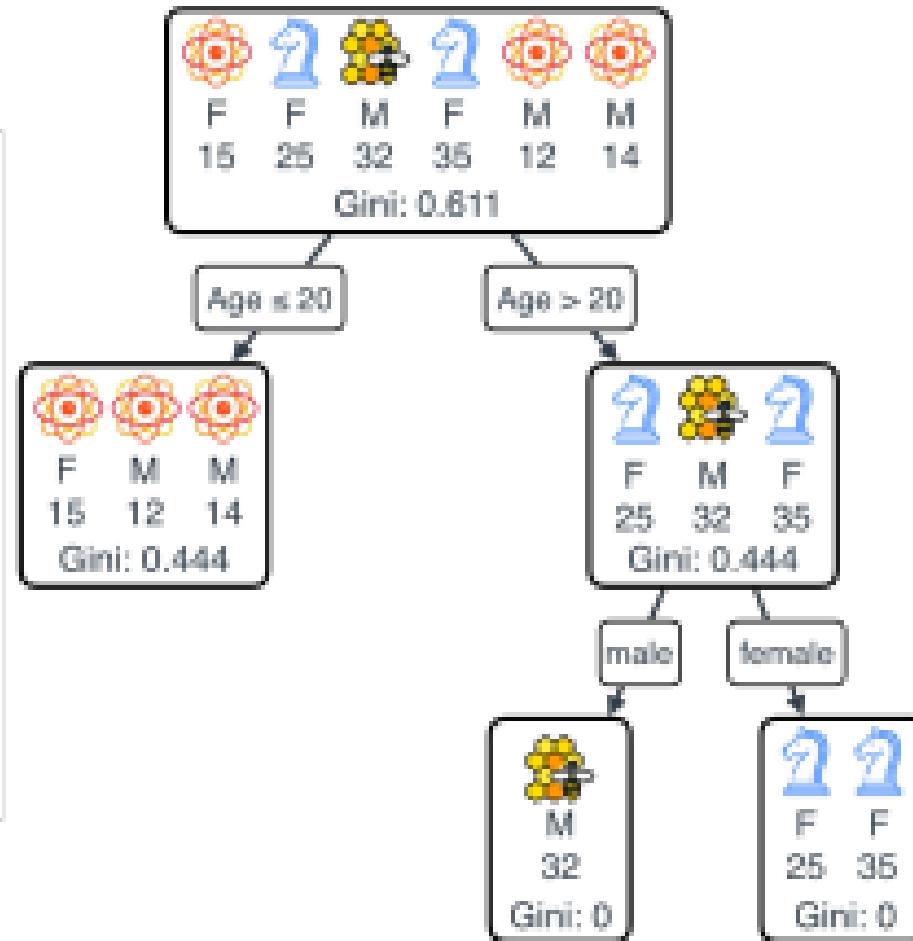
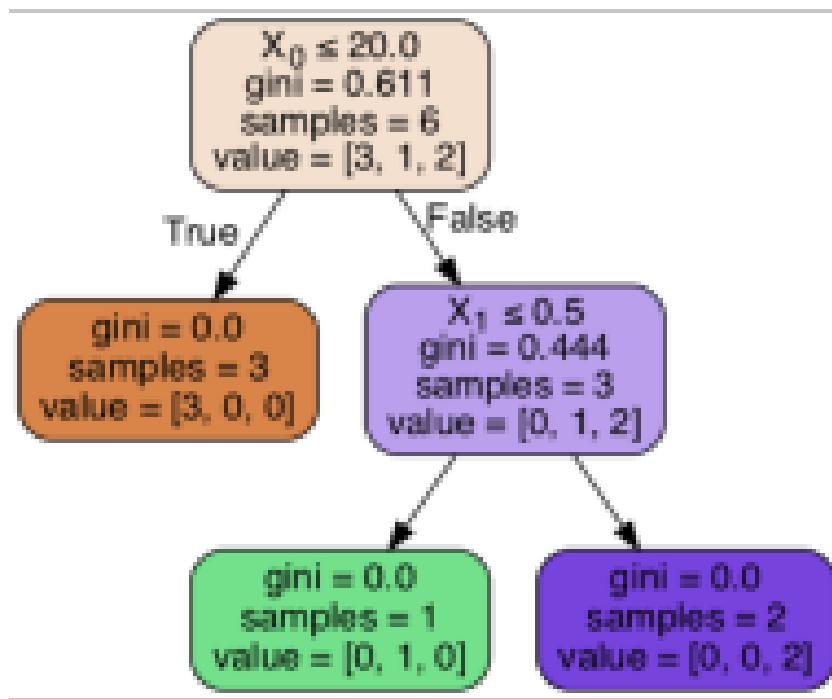
Finally, we train our model by using the `DecisionTreeClassifier()` class from `sklearn`. The model will be called '`app_decision_tree`', and the command for fitting the data is simply '`fit()`'.

```
app_decision_tree = DecisionTreeClassifier()  
app_decision_tree.fit(features, labels)
```

Coding a decision tree with sklearn



Coding a decision tree with sklearn



A slightly larger example: Spam detection again!

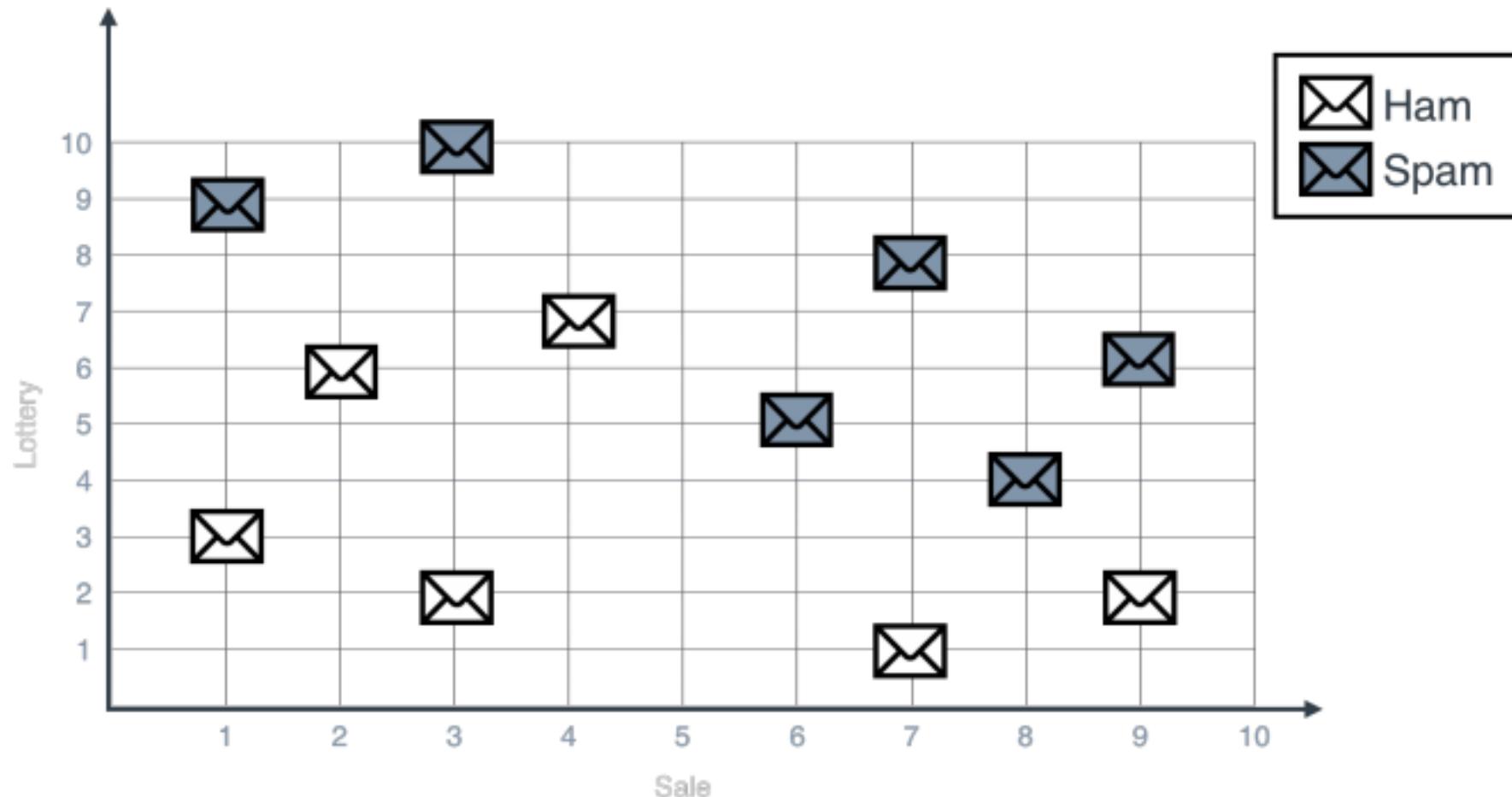
Lottery	Sale	Spam
7	1	No
3	2	No
3	9	No
1	3	No
2	6	No
4	7	No
1	9	Yes
3	10	Yes
6	5	Yes
7	8	Yes
8	4	Yes
9	6	Yes

A slightly larger example: Spam detection again!

- First, let's put our data into a Pandas DataFrame.

```
spam_dataset = pd.DataFrame({  
    'Lottery':[7,3,9,1,2,4,1,3,6,7,8,9],  
    'Sale':[1,2,3,3,6,7,9,10,5,8,4,6],  
    'Spam': [ 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham']})
```

A slightly larger example: Spam detection again!



A slightly larger example: Spam detection again!

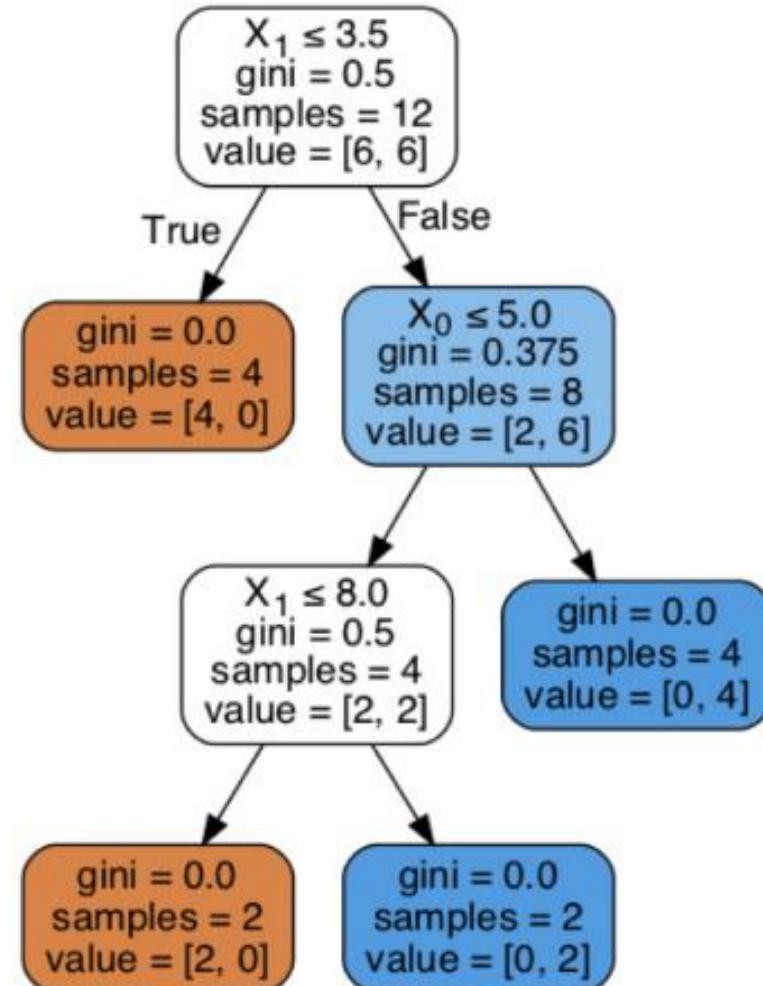
Next, we split into features and labels

```
features = spam_dataset[['Lottery', 'Sale']]  
labels = spam_dataset['Spam']
```

Now, let's define our decision tree model and train it.

```
spam_decision_tree = DecisionTreeClassifier()  
spam_decision_tree.fit(X,y)
```

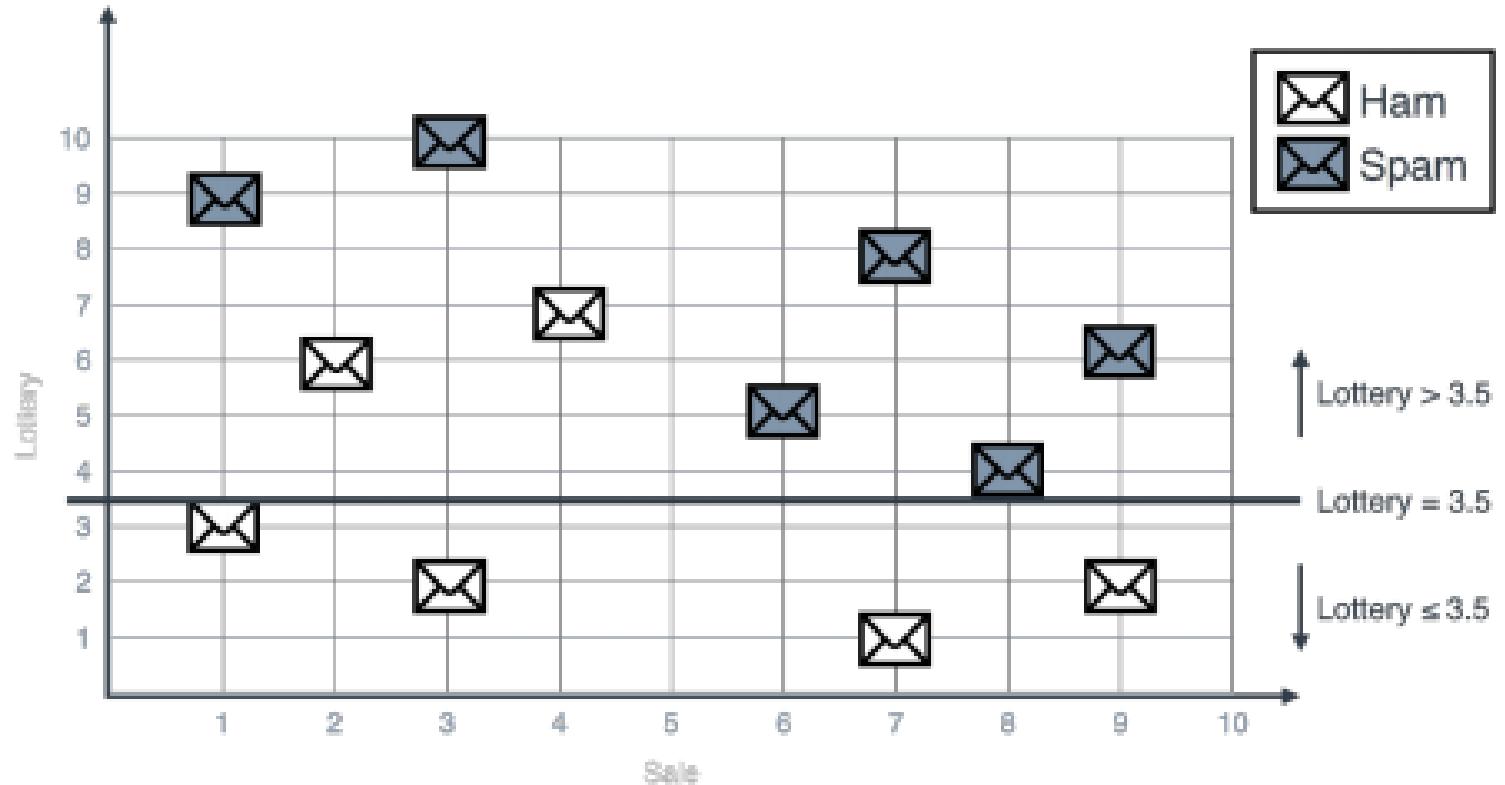
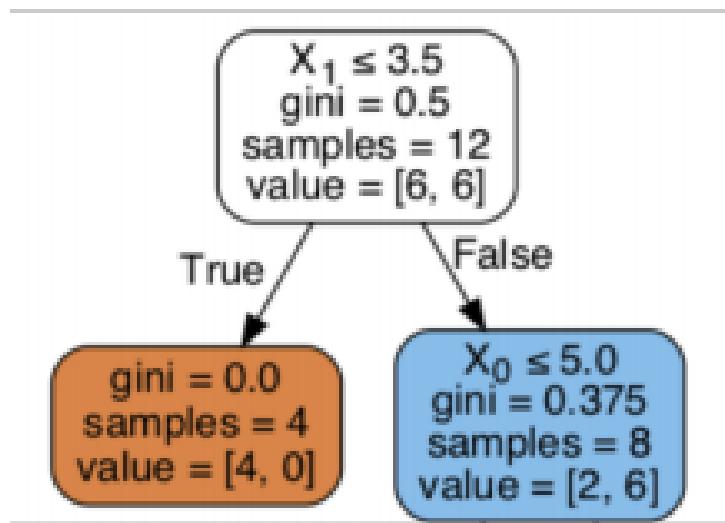
A slightly larger example: Spam detection again!



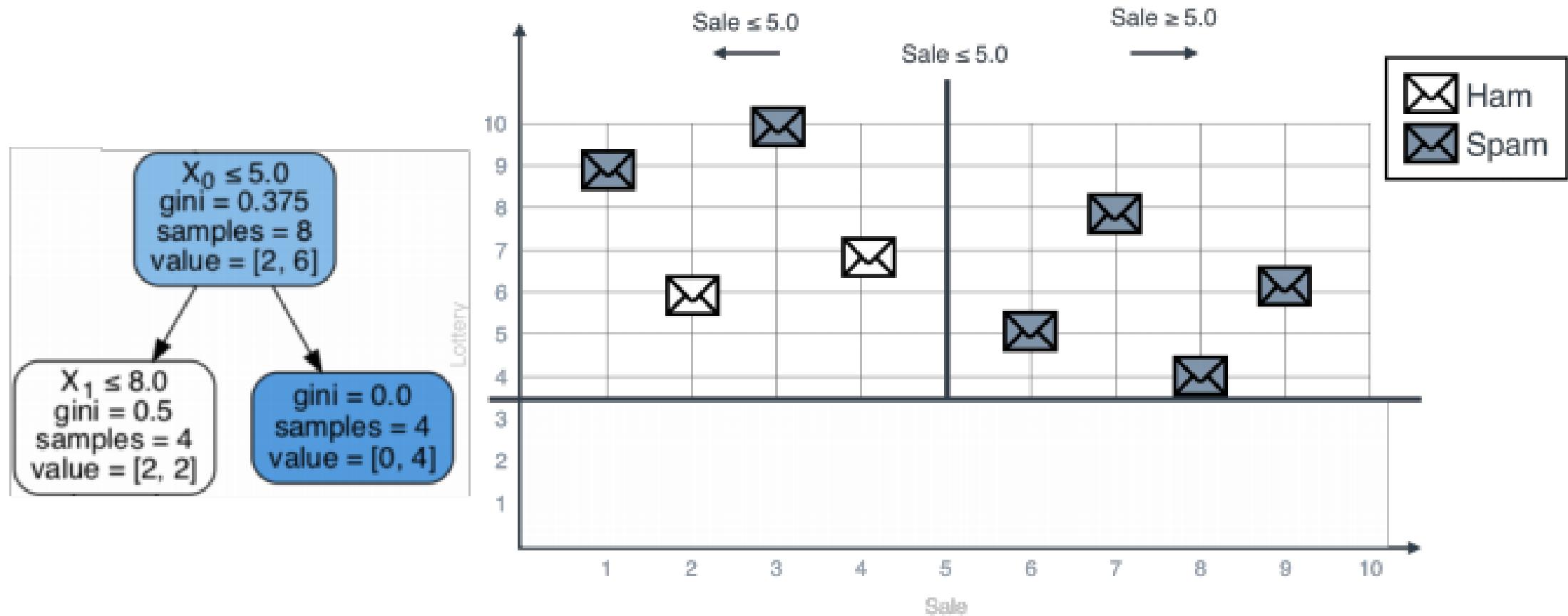
A slightly larger example: Spam detection again!

```
predict(email):
    if number of appearances of 'sale' is less than or equal to 3.5:
        classify as ham
    else:
        if number of appearances of 'lottery' is greater than 5.0:
            classify as spam
        else:
            if number of appearances of 'sale' is less than or equal to 8.0:
                classify as ham
            else:
                Classify as spam
```

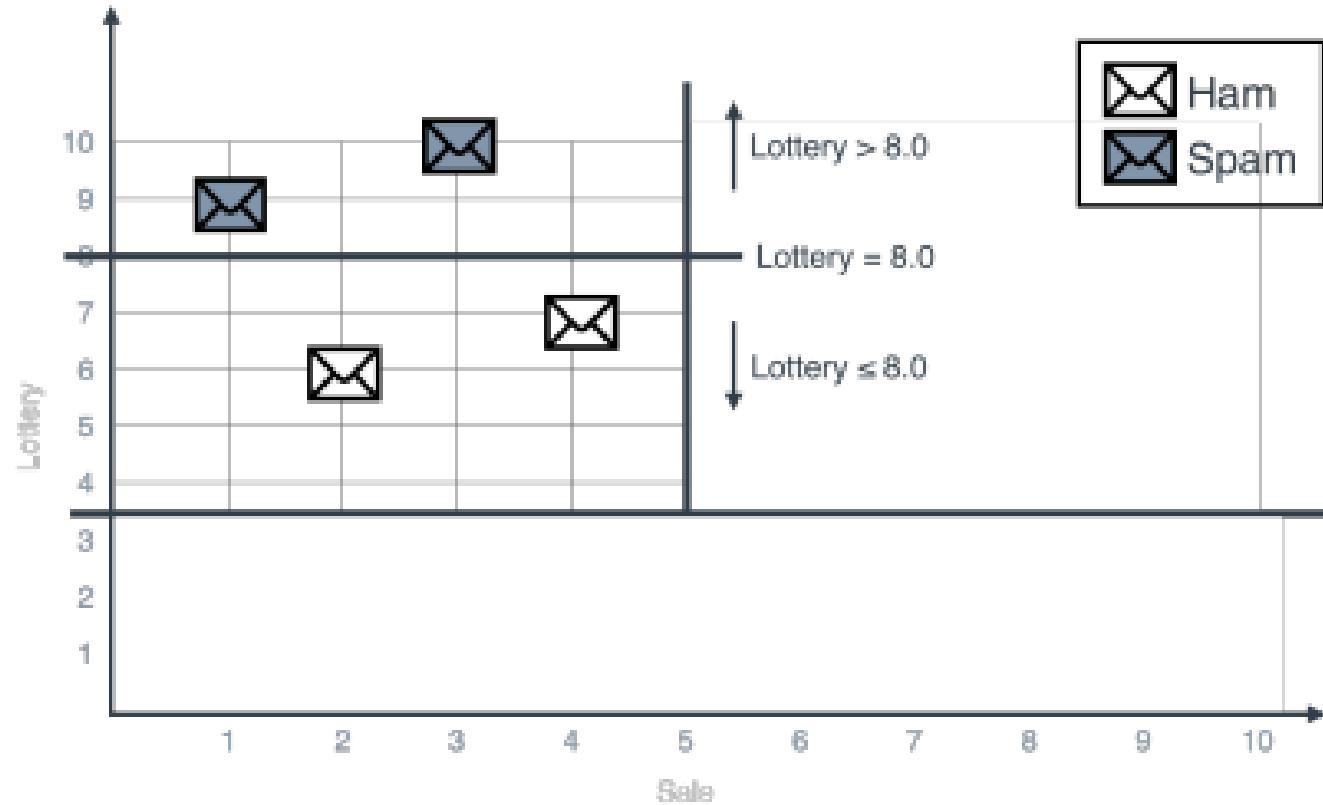
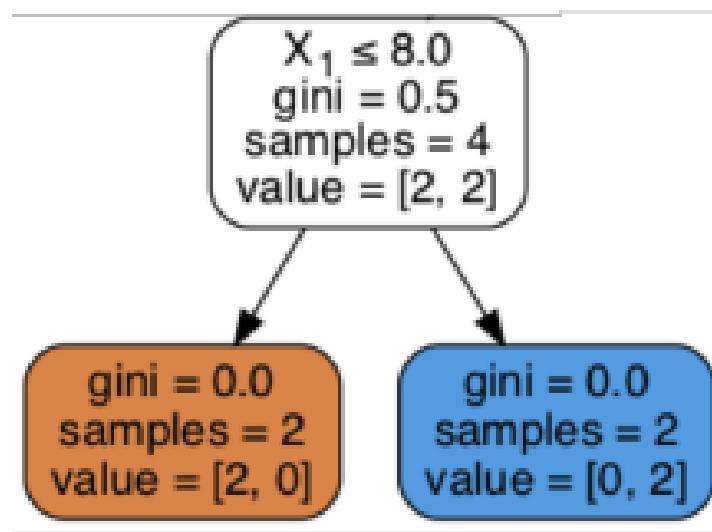
A slightly larger example: Spam detection again!



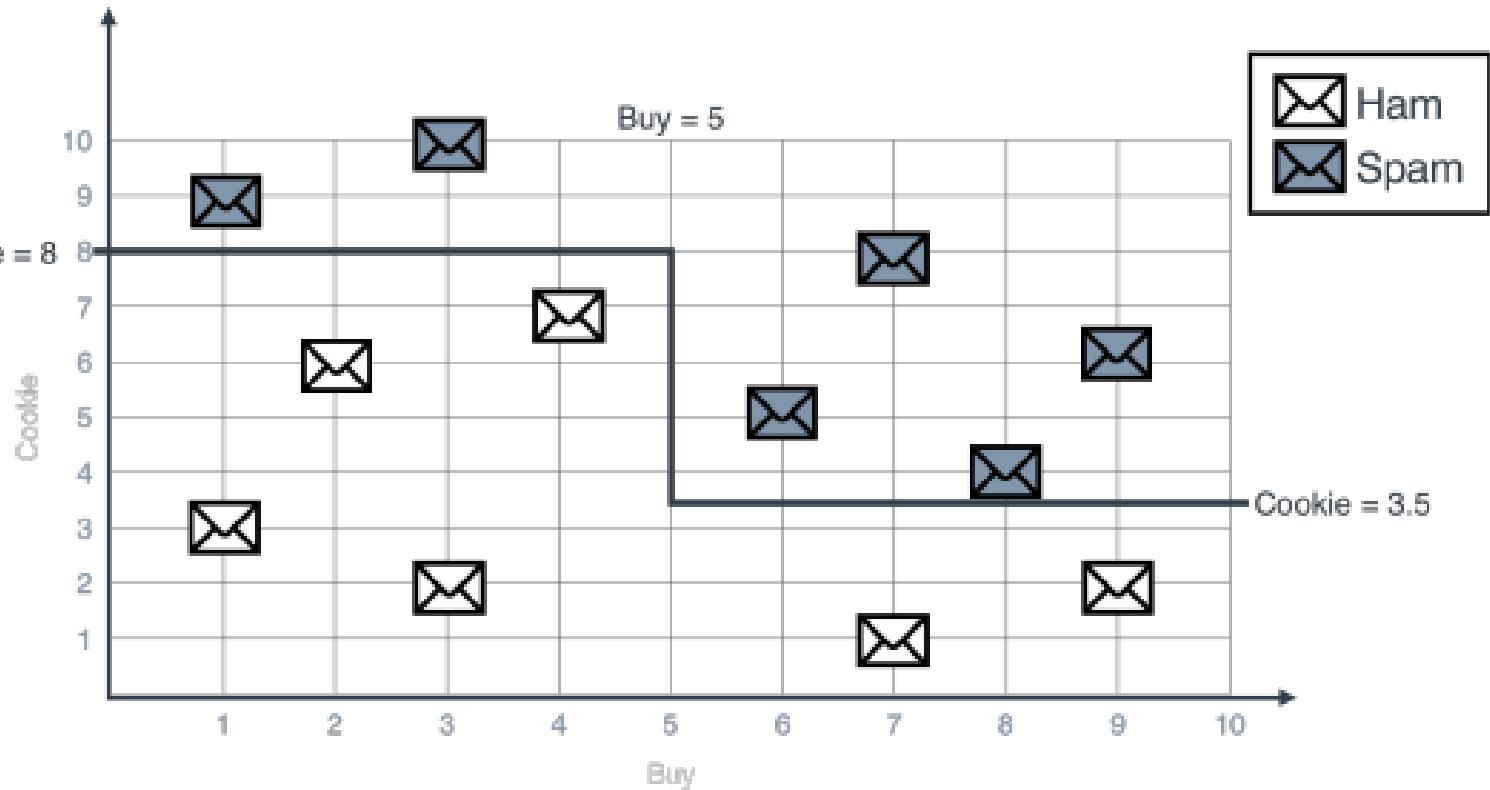
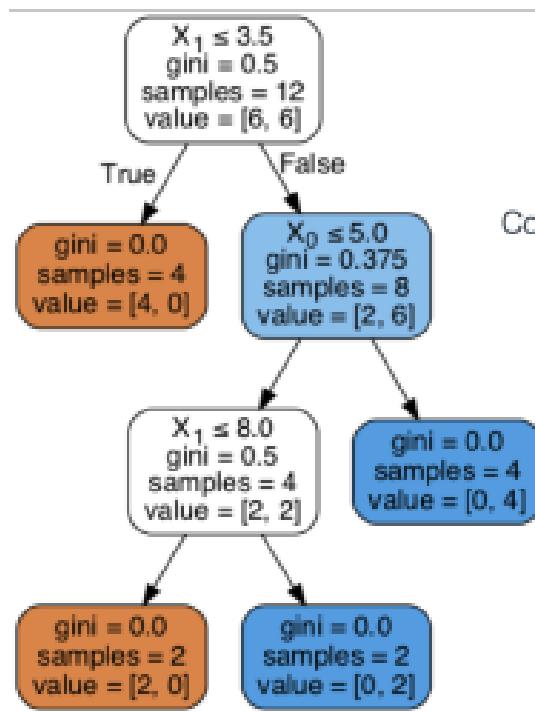
A slightly larger example: Spam detection again!



A slightly larger example: Spam detection again!



A slightly larger example: Spam detection again!



Decision trees are widely used in health care

- Decision trees are widely used in medicine, not only to make predictions, but also to identify features that are determinant in the prediction.
- You can imagine that in medicine, a black box saying “the patient is sick” or “the patient is healthy” is not good enough.

Summary

- Decision trees are a very important algorithm in machine learning, used for classification.
- The way decision trees work is by asking binary questions about our data, and making the prediction based on the answers to those questions.

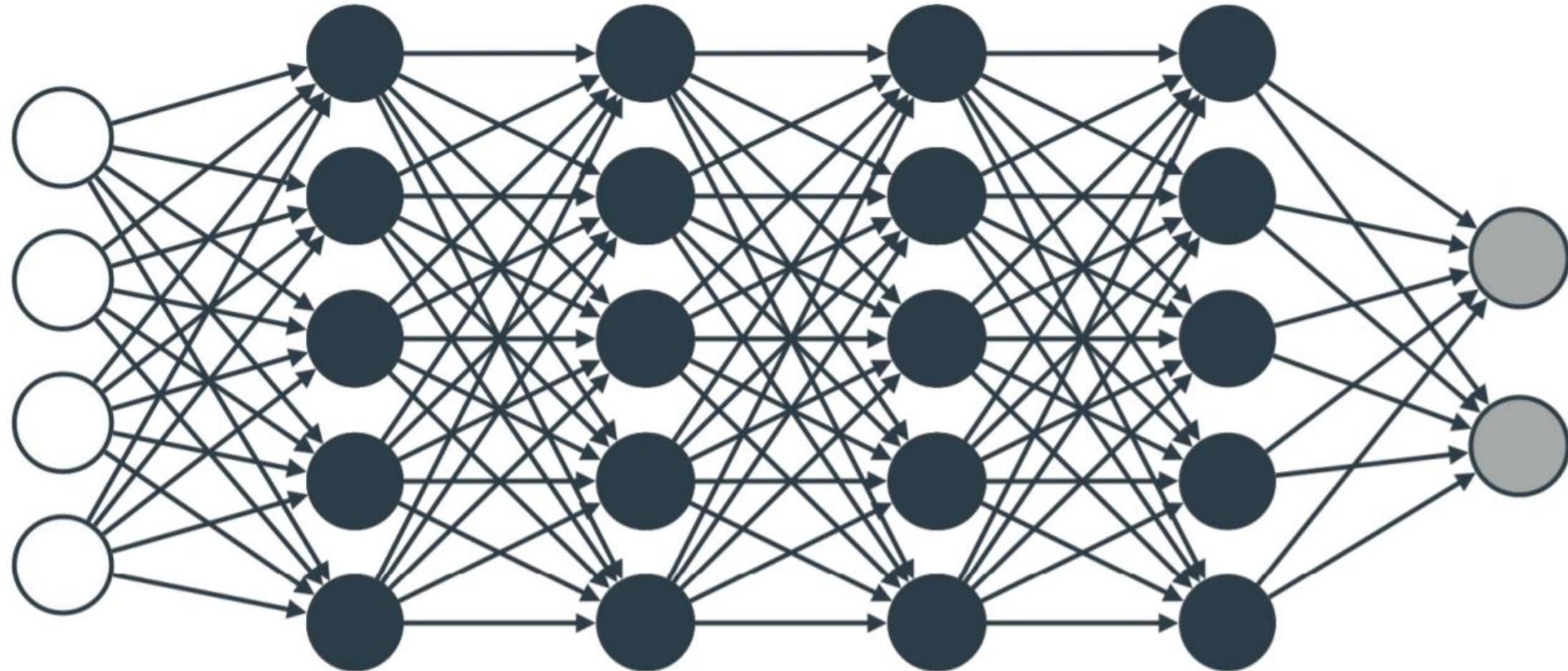
10: Combining building blocks to gain more power: Neural Networks

Neural Networks

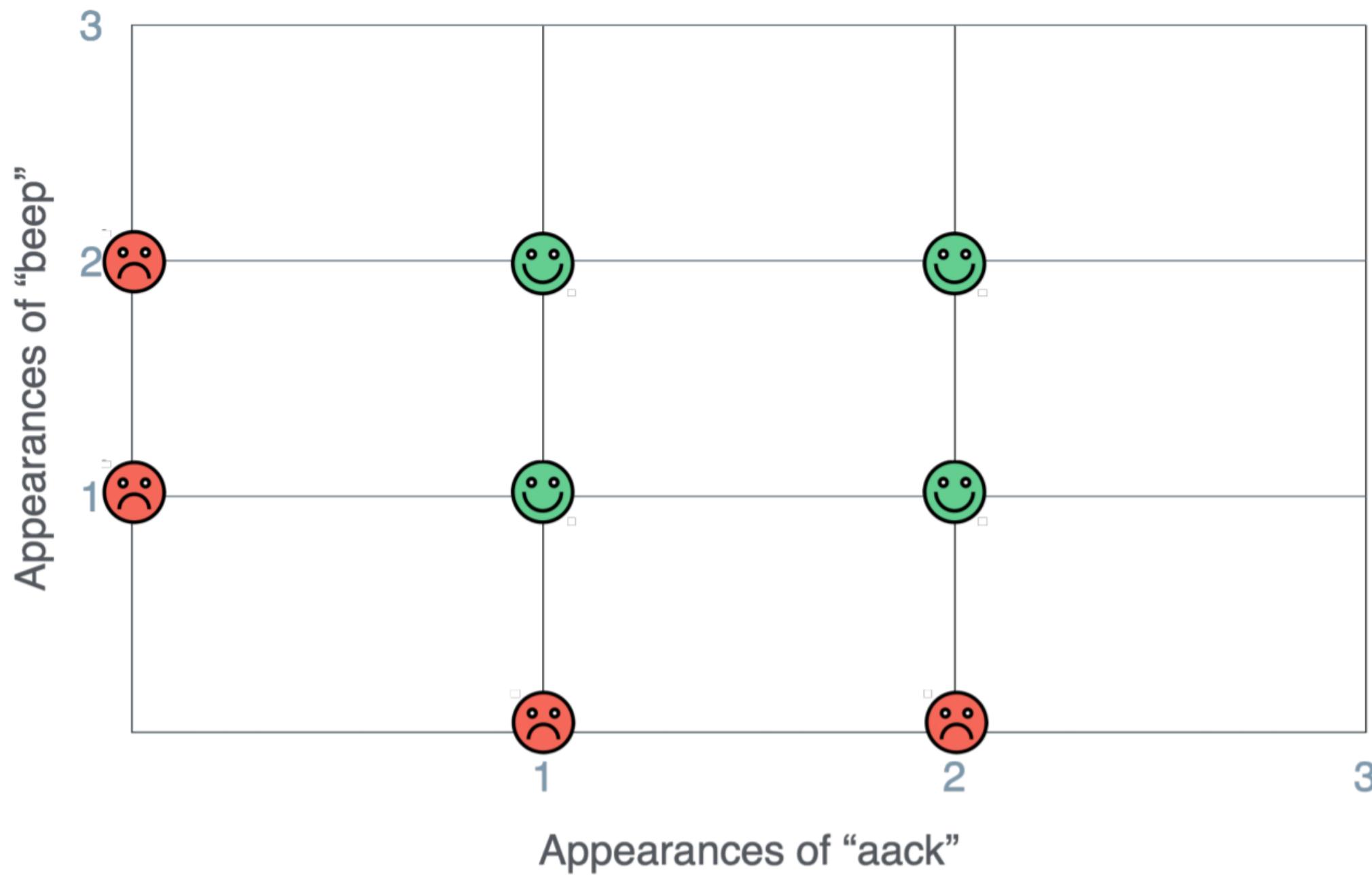
This lesson covers

- What is a neural network?
- What is a perceptron?
- Using neural networks in a simple application: sentiment analysis.
- Training neural networks using backpropagation.
- Potential problems in training neural networks, and techniques that can be used to avoid these problems.
- How to code the linear regression algorithm in Keras.

Neural Networks



Sentence	Aack	Beep	Mood
"Aack"	1	0	Sad
"Aack aack"	2	0	Sad
"Beep"	0	1	Sad
"Beep beep"	0	2	Sad
"Aack beep"	1	1	Happy
"Aack aack beep"	1	2	Happy
"Beep aack beep"	2	1	Happy
"Beep aack beep aack"	2	2	Happy

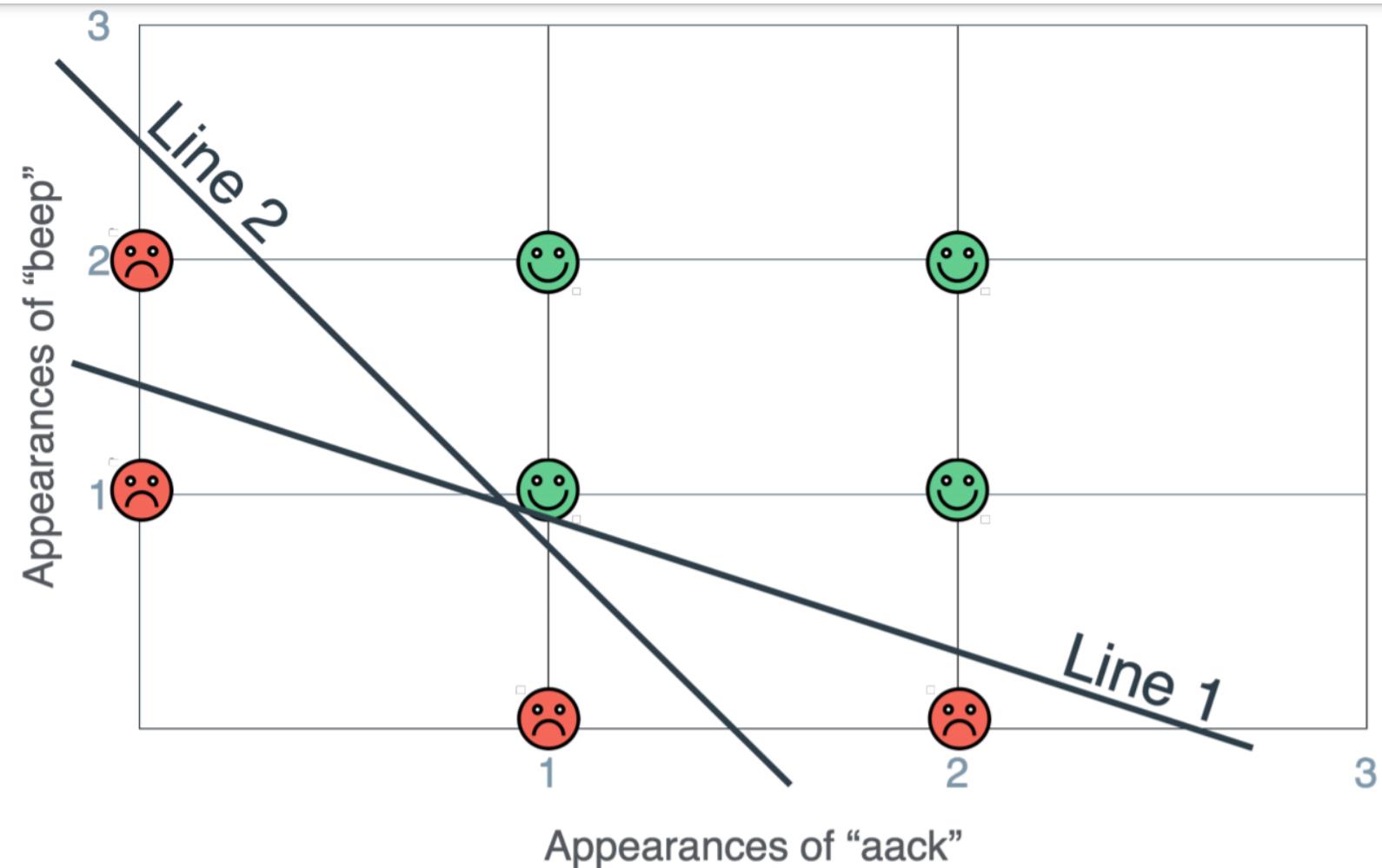


The problem - A more complicated alien planet!

If our goal is to separate the points in previous Figure, clearly one line won't do it. What is better than one line? I can think of two things:

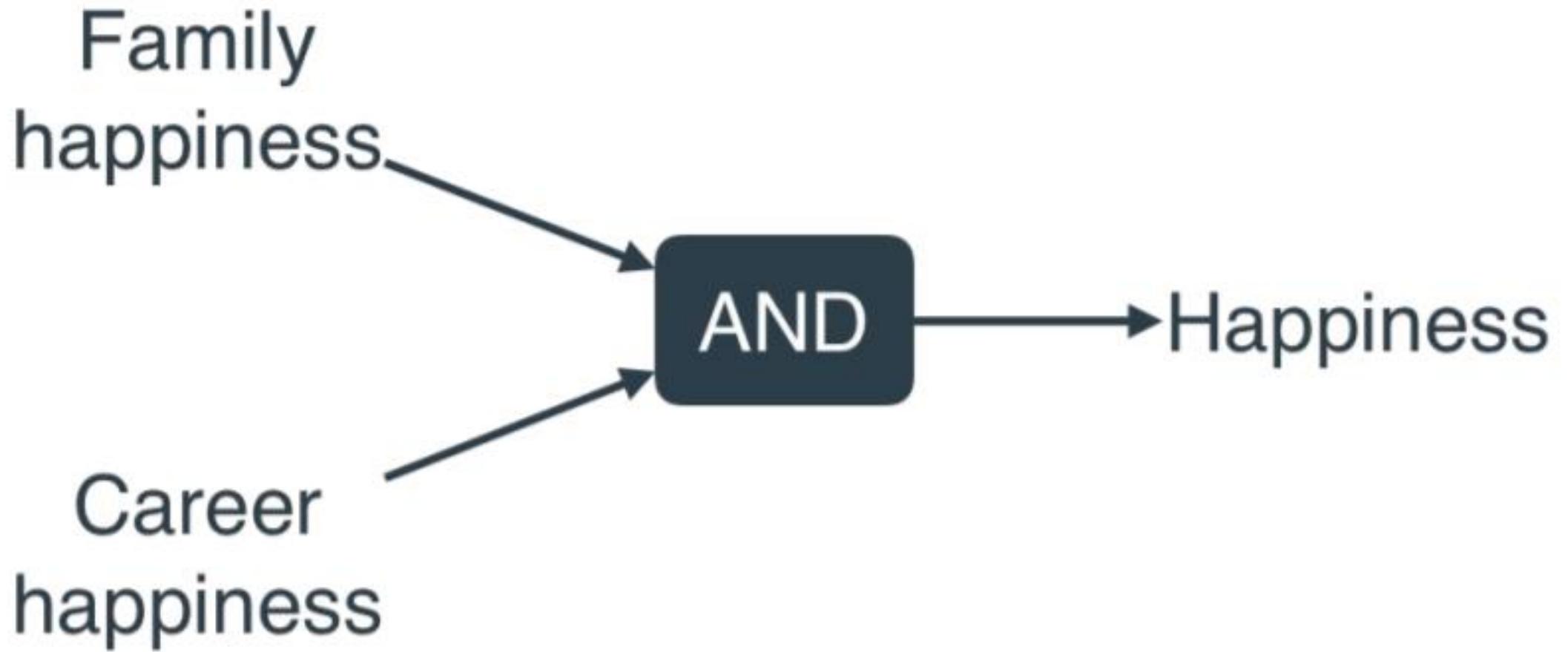
1. Two lines.
2. A curve.

Solution - If one line is not enough, use two lines to classify your dataset



Sentence	Aack	Beep	Equation 1	<i>Eq 1 ≥ 15?</i>	Equation 2	<i>Eq 2 ≥ 15?</i>	<i>Both eqs. ≥ 15</i>
"Aack"	1	0	6	no	10	no	no
"Aack aack"	2	0	12	no	20	yes	no
"Beep"	0	1	10	no	6	no	no
"Beep beep"	0	2	20	yes	12	no	no
"Aack beep"	1	1	16	yes	16	yes	yes
"Aack aack beep"	1	2	26	yes	22	yes	yes
"Beep aack beep"	2	1	22	yes	26	yes	yes
"Beep aack beep aack"	2	2	32	yes	32	yes	yes

Why two lines? Is happiness not linear?



Perceptrons and how to combine them

- In this lesson we have seen two classifiers that are very similar to those in lesson 4: The family happiness and the career happiness classifier.
- Let's study them more carefully. We'll start with the family happiness classifier.
- It was given by Line 1, which had the equation
6*(#aack) + 10*(#beep) ≥ 15

Perceptrons and how to combine them

Classifier 1 (family happiness)

Scores:

- Aack: 6 points
- Beep: 10 points
- Threshold: 15

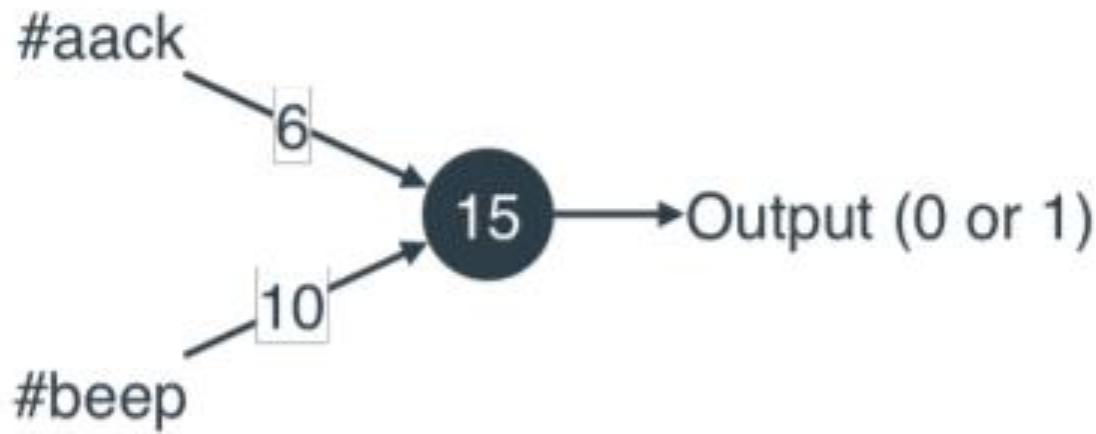
Rule:

Add the scores of all the words.

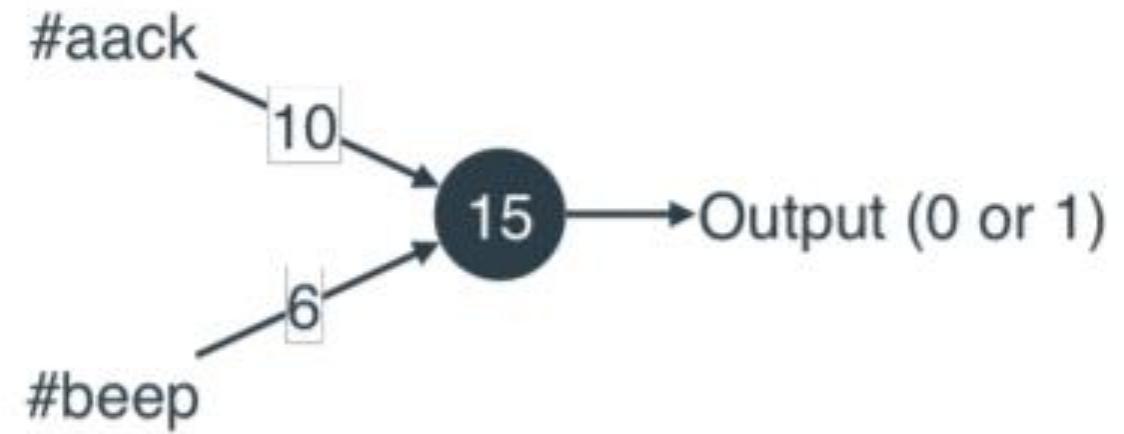
- If the score is larger than or equal to the threshold of 15, predict that the alien has a happy family life.
- If the score is smaller than the threshold of 15, predict that the alien doesn't have a happy family life.

Perceptrons and how to combine them

Classifier 1 (family)



Classifier 2 (career)



Perceptrons and how to combine them

Classifier 2 (career happiness)

Scores:

- Aack: 10 points
- Beep: 6 points
- Threshold: 15

Rule:

Add the scores of all the words.

- If the score is larger than or equal to the threshold of 15, predict that the alien has a happy career life.
- If the score is smaller than the threshold of 15, predict that the alien doesn't have a happy career life.

Sentence	Aack	Beep	Family Happiness	Career Happiness
“Aack”	1	0	0	0
“Aack aack”	2	0	0	1
“Beep”	0	1	0	0
“Beep beep”	0	2	1	0
“Aack beep”	1	1	1	1
“Aack aack beep”	1	2	1	1
“Beep aack beep”	2	1	1	1
“Beep aack beep aack”	2	2	1	

Family Happiness (feature)	Career Happiness (feature)	Happiness (label)
0	0	0
0	1	0
0	0	0
1	0	0
1	1	1
1	1	1
1	1	1
1	1	1

Classifier 3 (happiness)

Scores:

- Family happiness: 1 point
- Career happiness: 1 point
- Threshold: 1.5

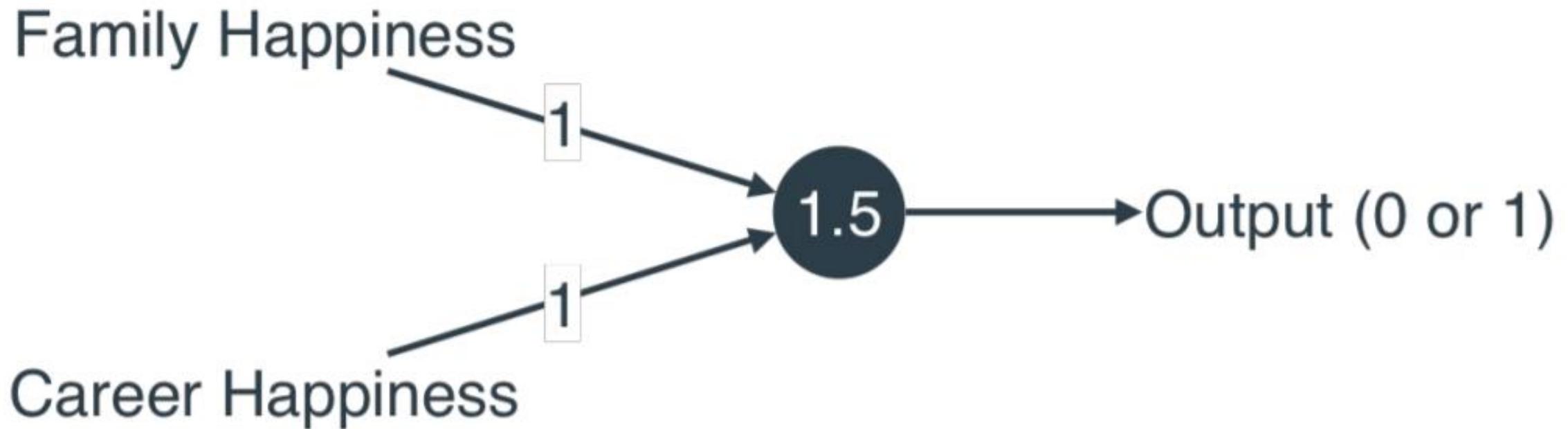
Rule:

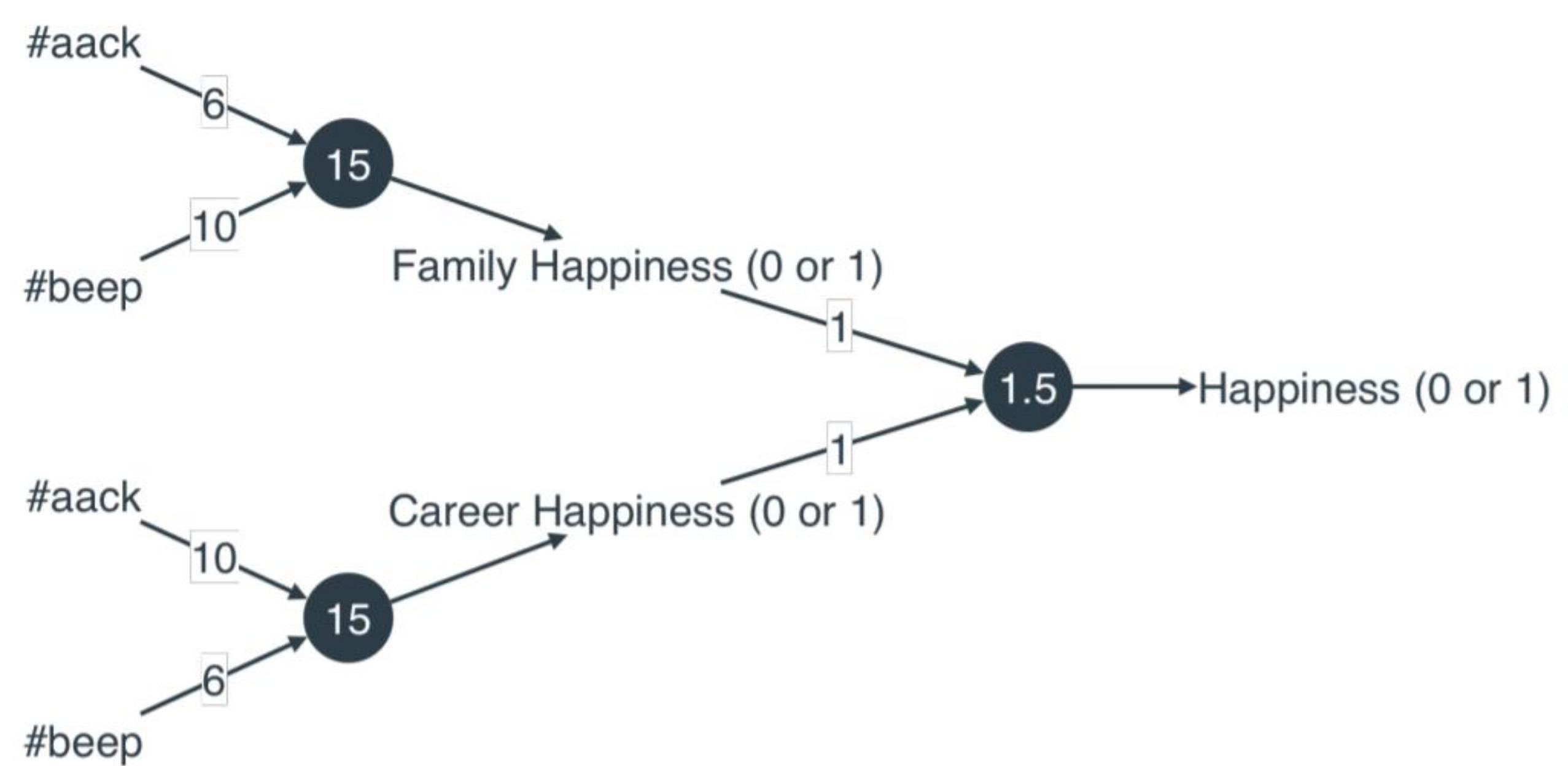
Add the scores of all the words.

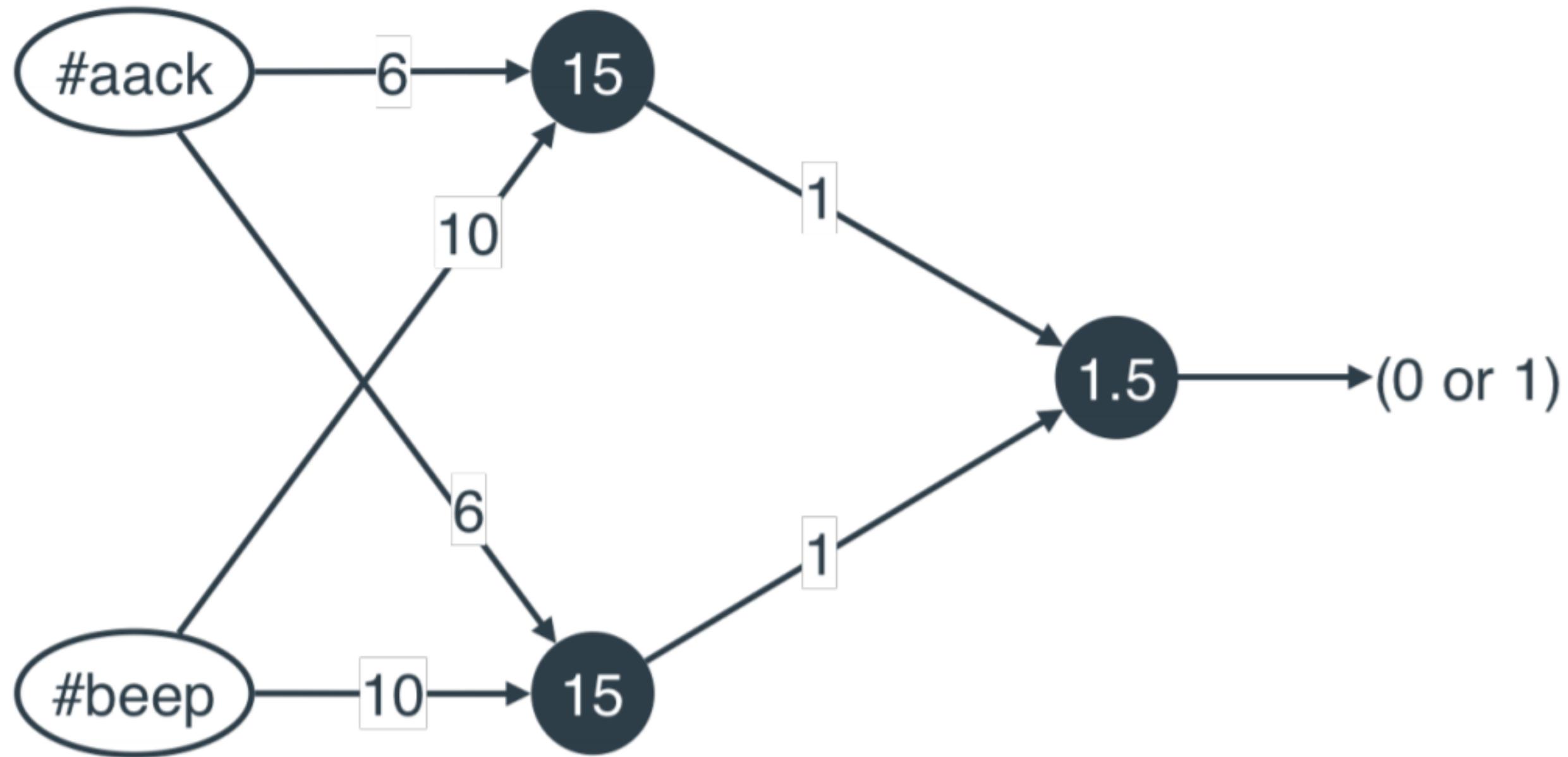
- If the score is larger than or equal to the threshold of 1.5, predict that the alien has a happy career life.
- If the score is smaller than the threshold of 1.5, predict that the alien doesn't have a happy career life.

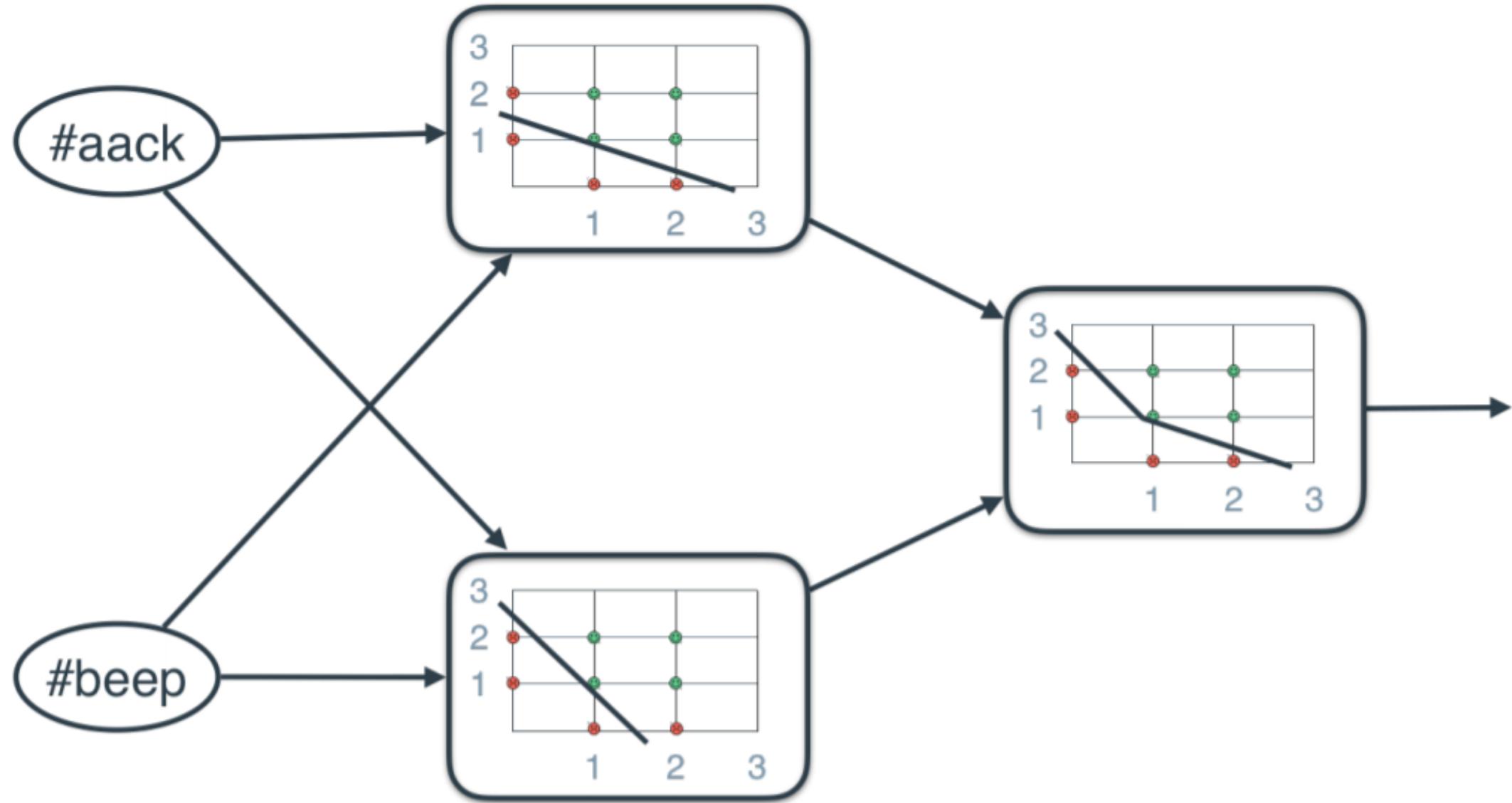
Perceptrons and how to combine them

Classifier 3 (happiness)









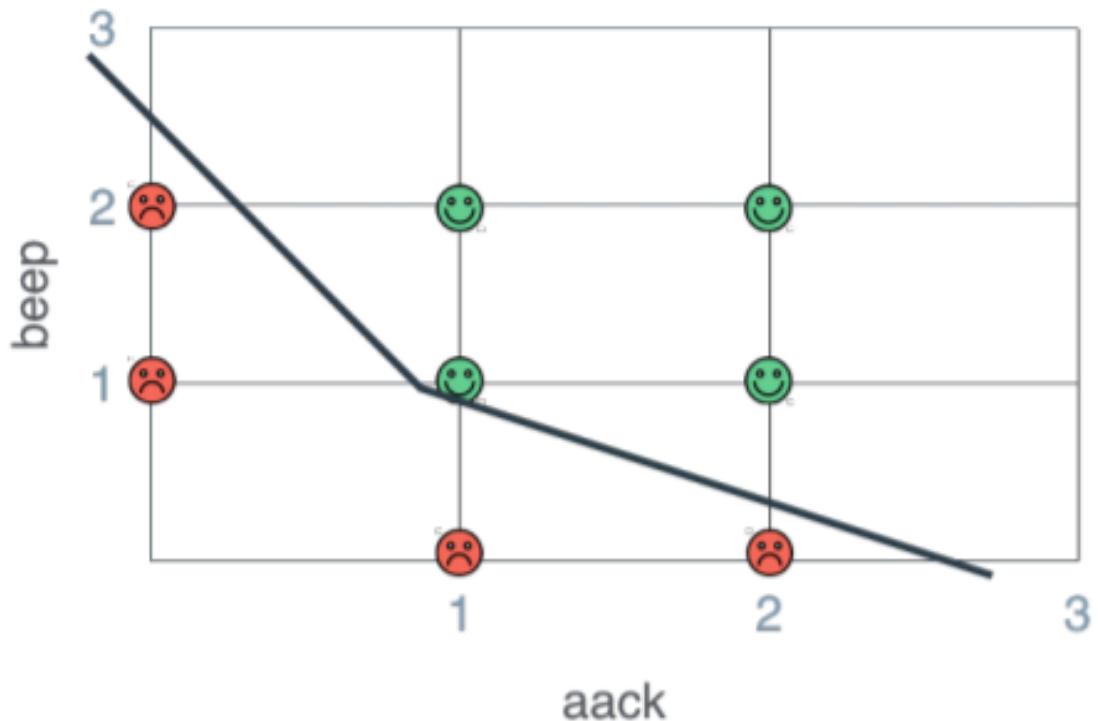
A trick to improve our training

- Similarly, the equation for classifier 2, the career happiness classifier, is
Output = $\sigma\sigma(10 \times (\#aaaaaaa) + 6 \times (\#bbbbbb) - 15)$,
- and that of classifier 3, the happiness classifier, is
Output = $\sigma\sigma(1 \times (CCCCCCCCCCCC) + 1 \times (\#bbbbbb) - 1.5)$,
- where Career and Family are the outputs of classifiers 1 and 2, respectively.

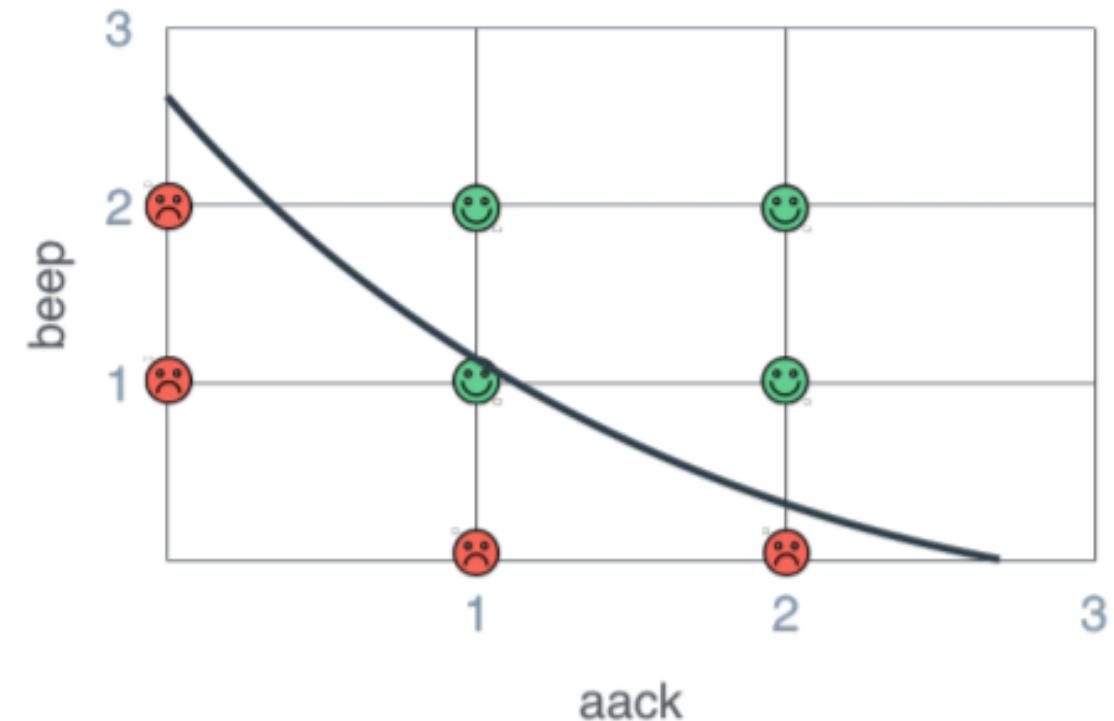
Sentence	Aack	Beep	Career Happiness	Family Happiness	Happiness
“Aack”	1	0	0.000	0.007	0.183
“Aack aack”	2	0	0.047	0.993	0.387
“Beep”	0	1	0.007	0.000	0.183
“Beep beep”	0	2	0.993	0.047	0.387
“Aack beep”	1	1	0.731	0.731	0.491
“Aack aack beep”	1	2	1.000	0.999	0.622
“Beep aack beep”	2	1	0.999	1.000	0.622
“Beep aack beep aack”	2	2	1.000	1.000	0.622

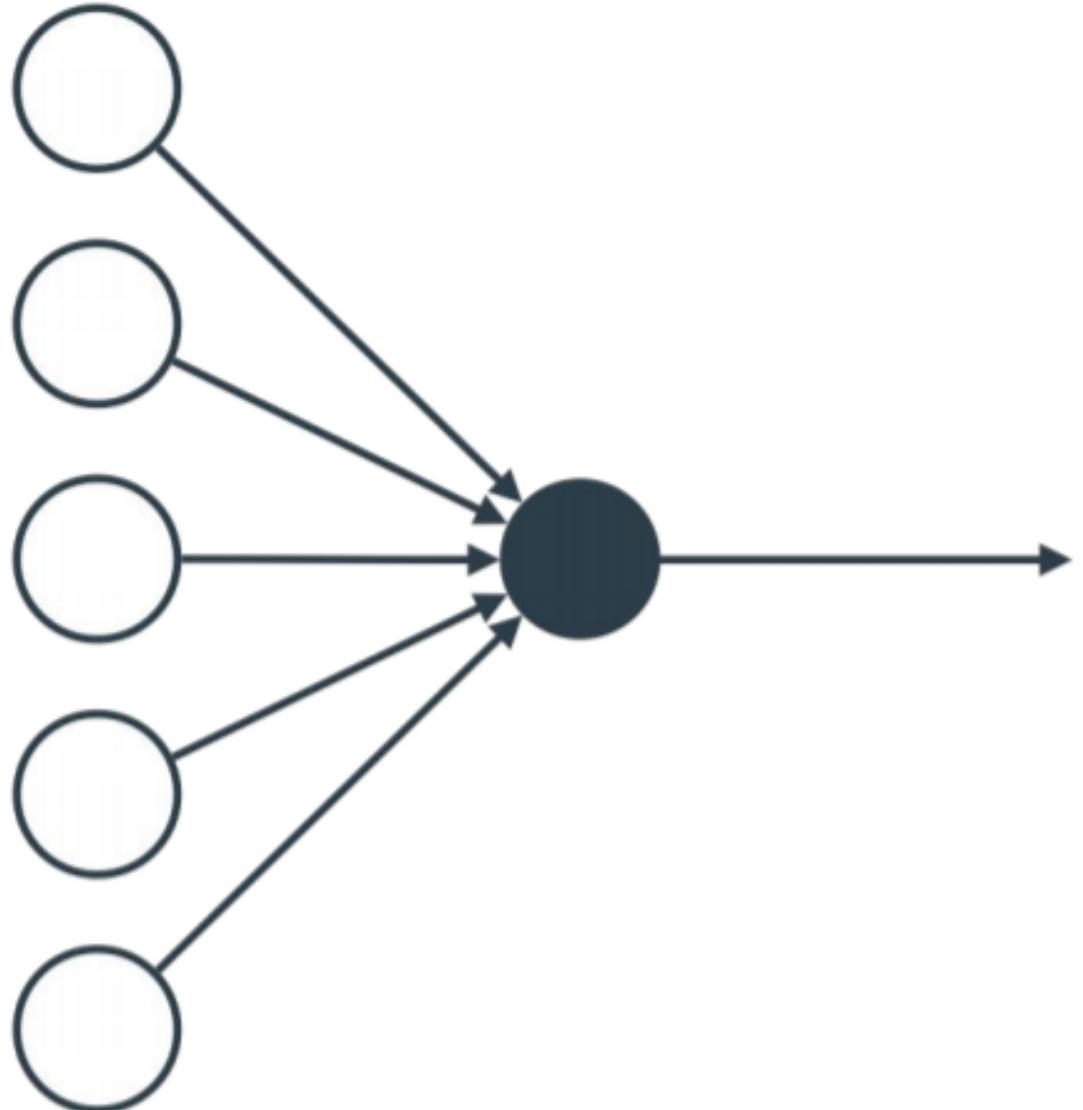
A trick to improve our training

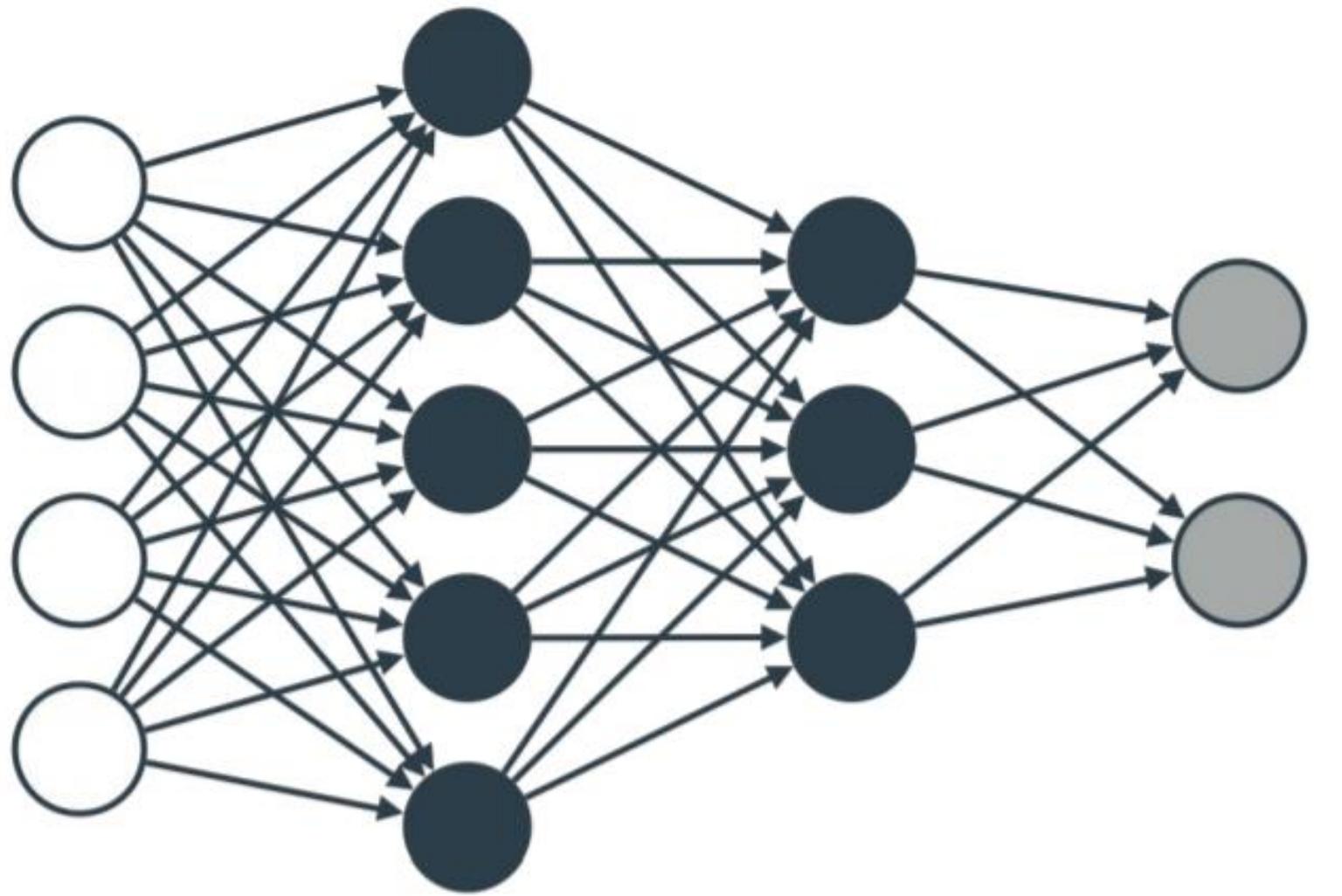
Discrete multilayer perceptron



Continuous multilayer perceptron







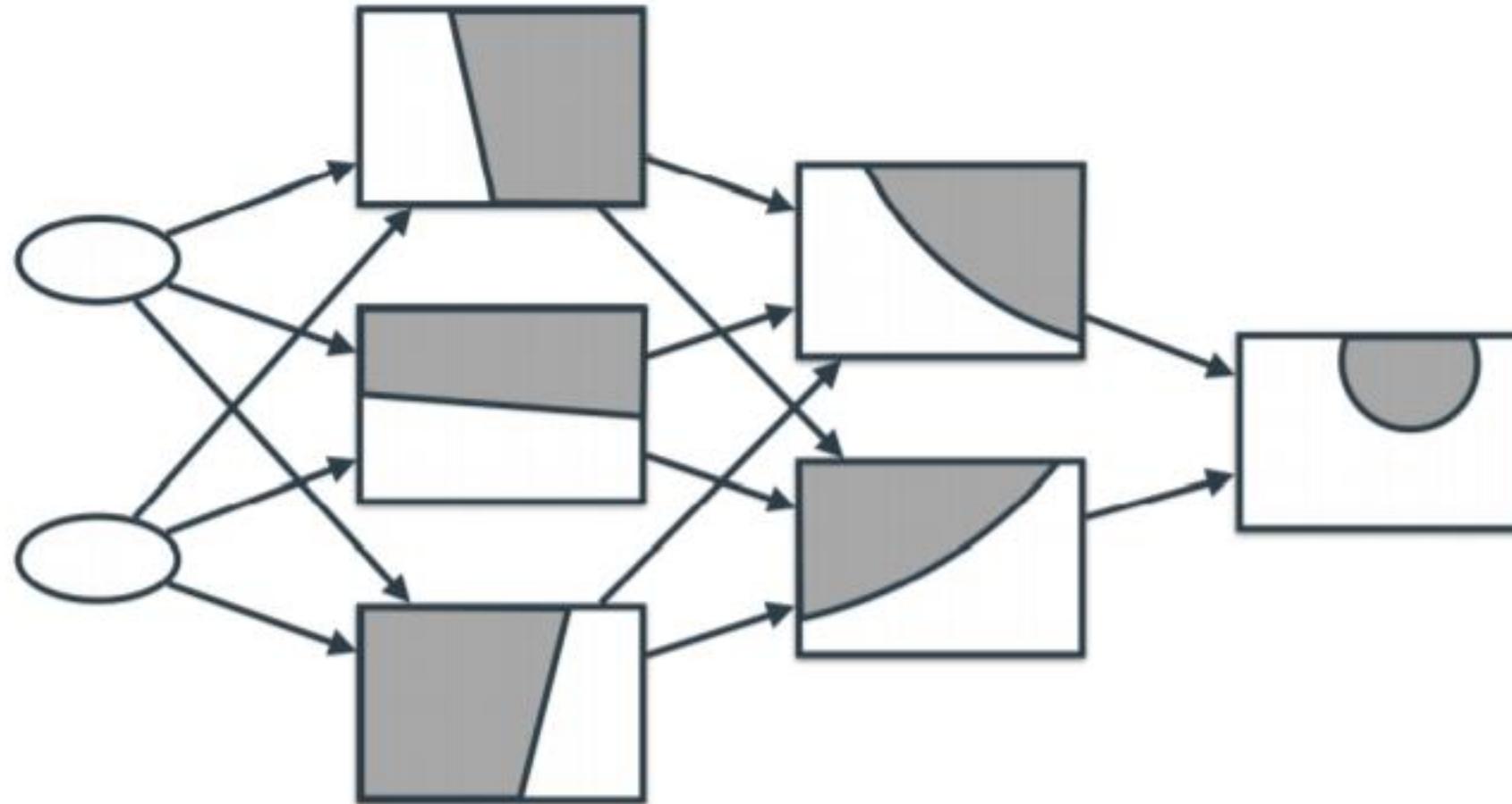
Input
layer

Hidden
layer

Hidden
layer

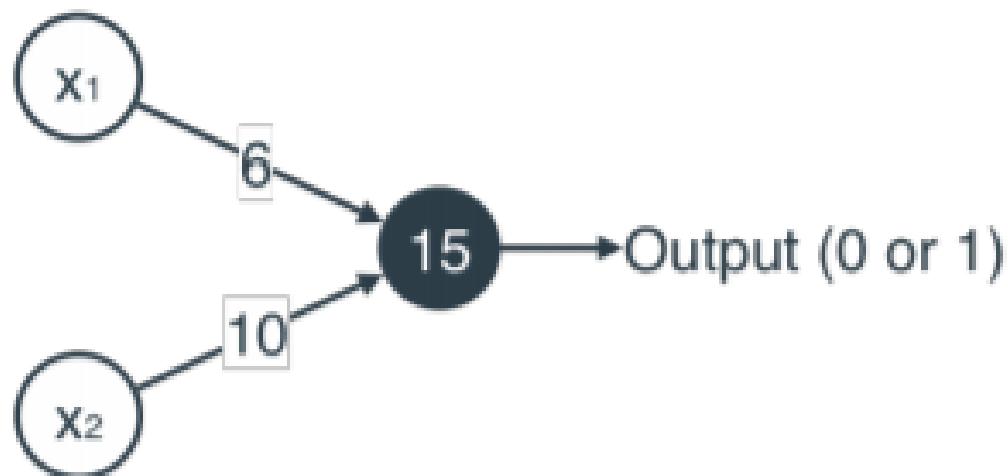
Output
layer

The architecture of a neural network

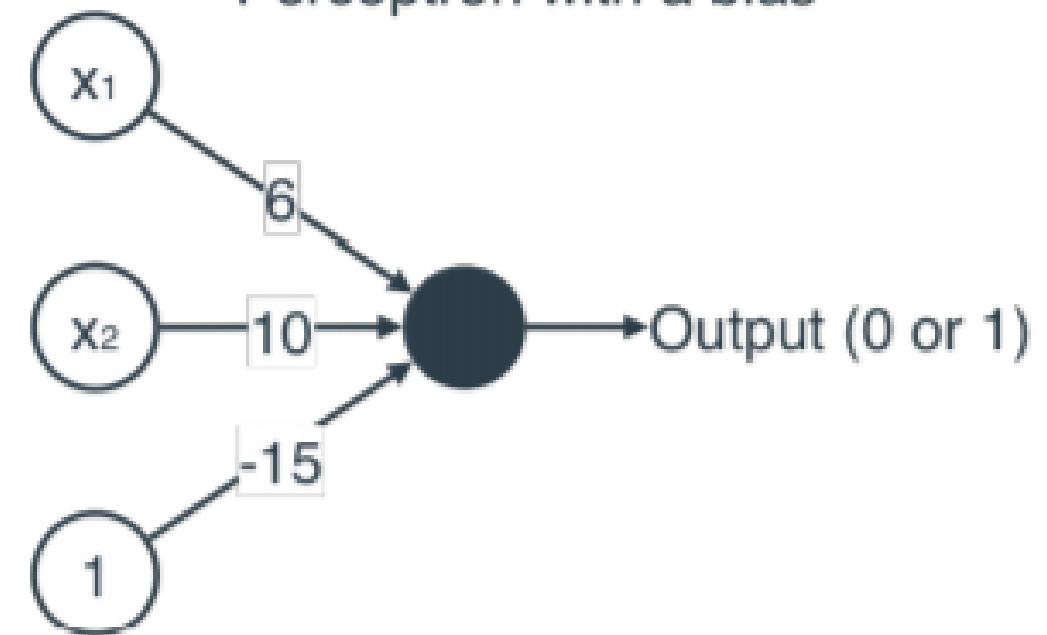


Bias vs Threshold

Perceptron with a threshold



Perceptron with a bias



Training neural networks

Error function - A way to measure how our neural network is performing

- The training process in most machine learning models makes use of an error function which constantly informs us how the model is doing at every step this is the case for neural networks as well.
- The error function we use here is actually very similar from the one we used for logistic regression in lesson 5; in fact, it is the same log loss function

Training neural networks

function is defined as follows:

- If the label is 0:
 - log loss = $-\ln(1 - \text{prediction})$
- If the label is 1:
 - log loss = $-\ln(\text{prediction})$.

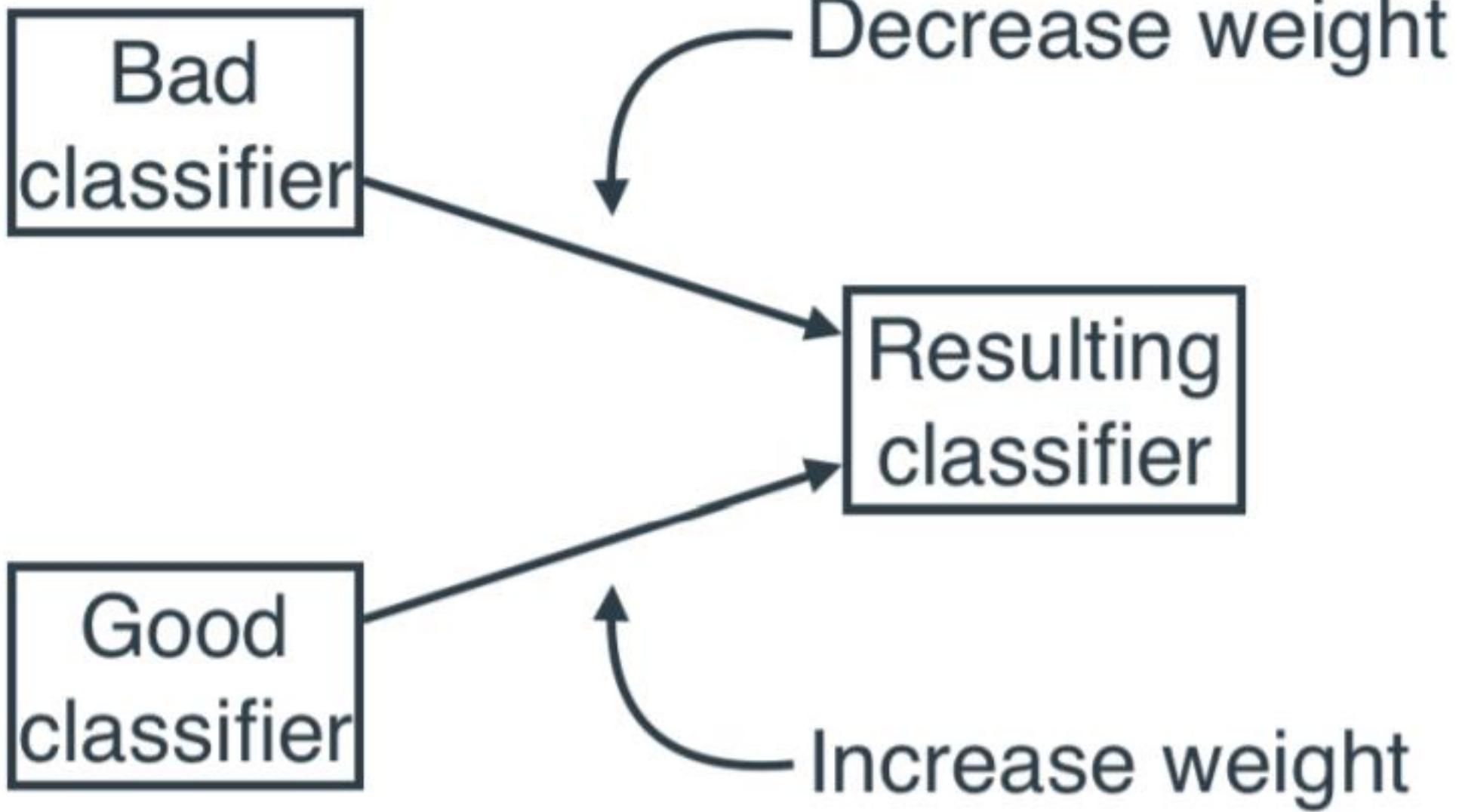
These can be summarized into one formula as:

$$\text{log loss} = (\text{-label}) \ln(\text{prediction}) - (1\text{-label}) \ln(1 - \text{prediction})$$

Training neural networks

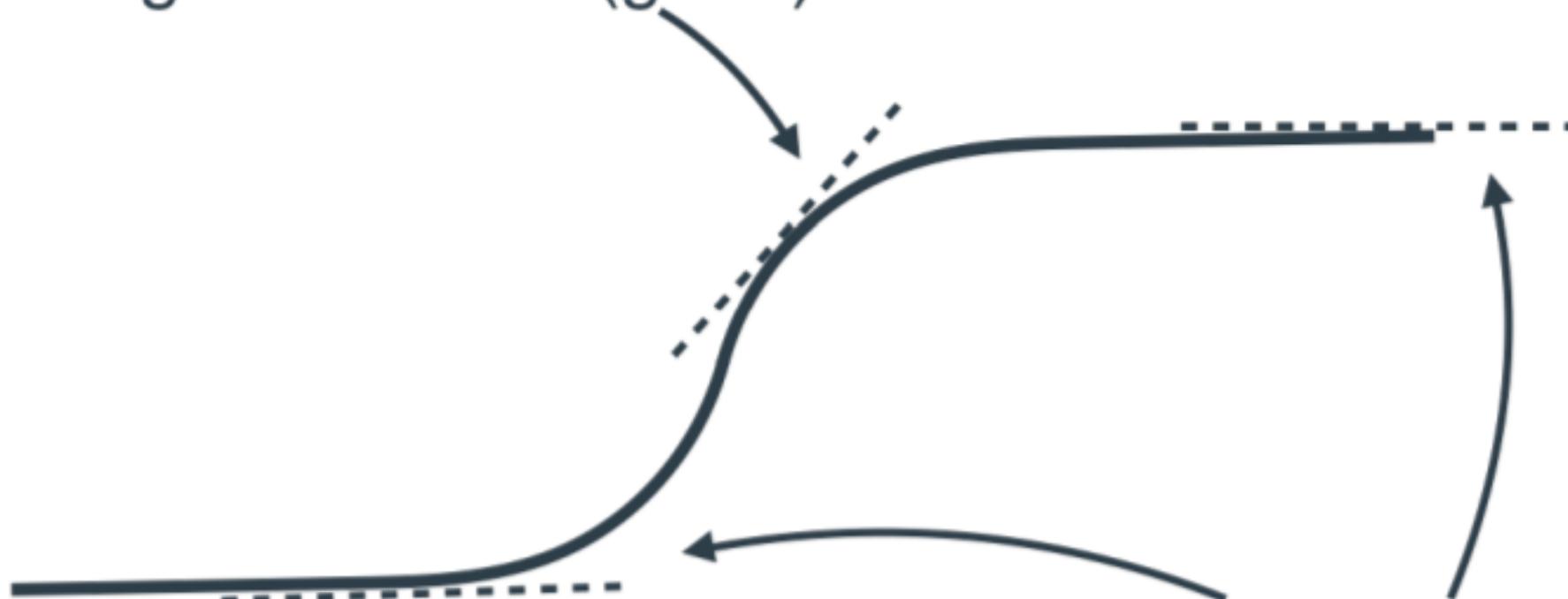
Gradient descent:

1. Start with random weights.
2. Repeat many times:
 - a) Calculate the loss function.
 - b) Take a small step in the direction of the gradient in order to decrease the loss function by a small amount.
3. The weights you obtain correspond to a neural network that (hopefully) fits the data well



Training neural networks

Large derivative (good)



Derivatives too
close to zero (bad)

REGULARIZATION - A WAY TO REDUCE OVERFITTING BY PUNISHING HIGH WEIGHTS

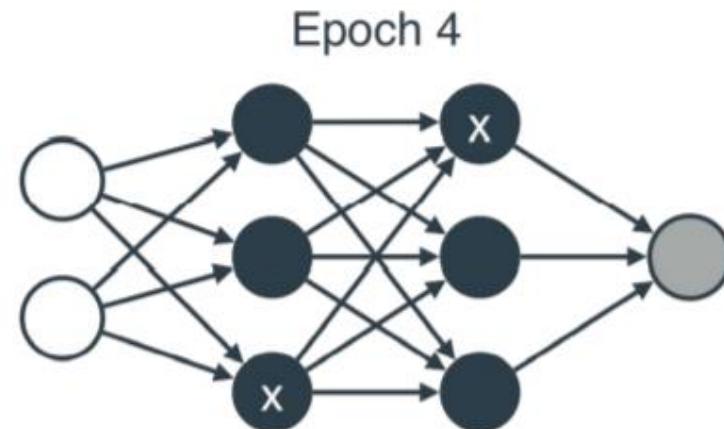
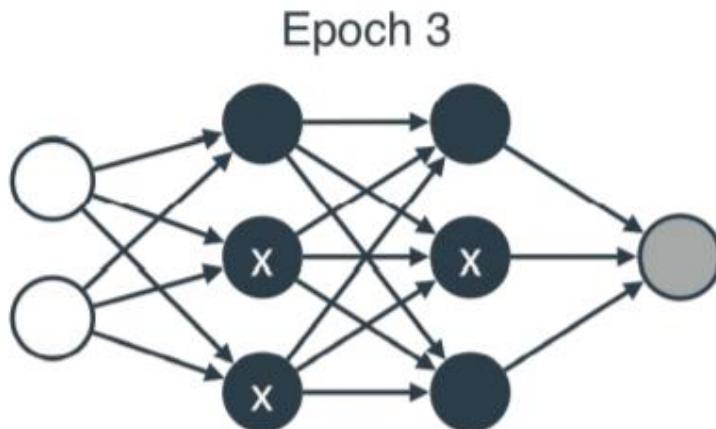
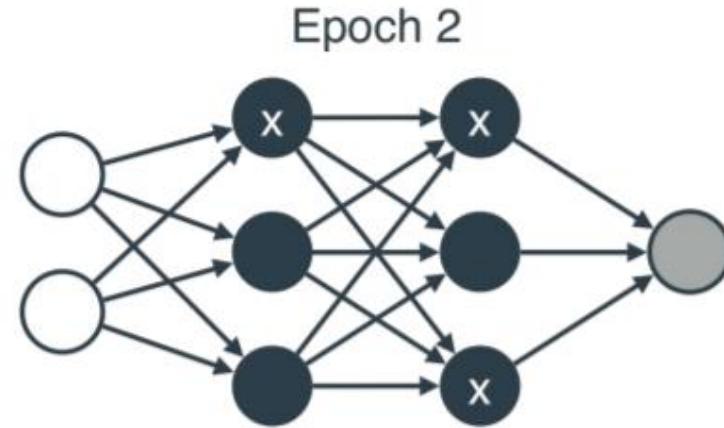
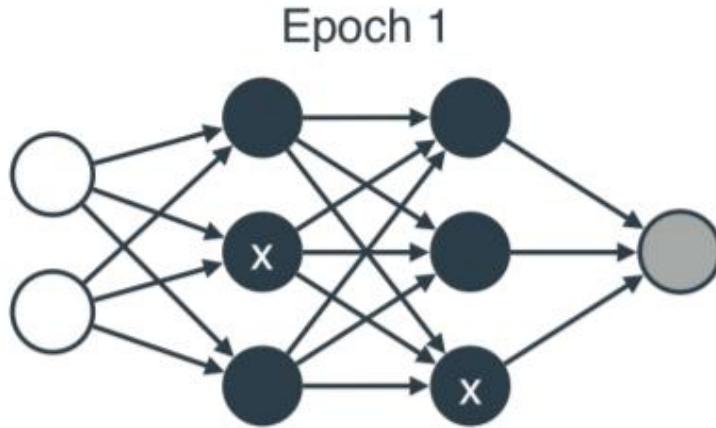
- As we've learned in this course, L1 and L2 regularization are techniques that we can use to decrease overfitting in many algorithms such as linear and logistic regression, and neural networks are not the exception.

Training neural networks

DROPOUT - A WAY TO MAKE SURE A FEW STRONG NODES ARE NOT DOMINATING THE TRAINING

- Dropout is a very interesting technique used to reduce overfitting in neural networks, and to understand it I like to think of the following analogy.
- Let's say that we are right-handed, and we like to go to the gym. After some time we start noticing that after a while in the gym, our right bicep is growing a lot, but our left one is not growing at all.

Training neural networks



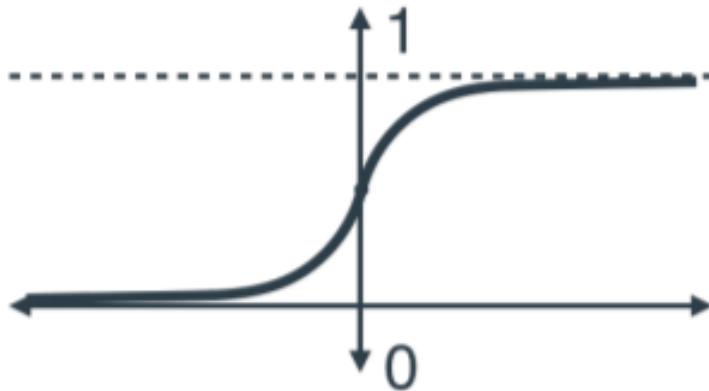
HYPERBOLIC TANGENT (TANH)

- The hyperbolic tangent function tends to work better than the sigmoid function in the practice,
- due to its shape.
- This one is given by the following formula:

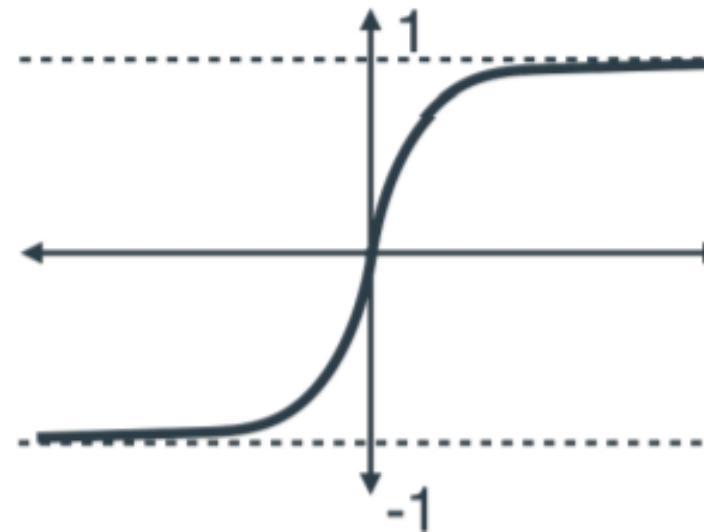
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Training neural networks

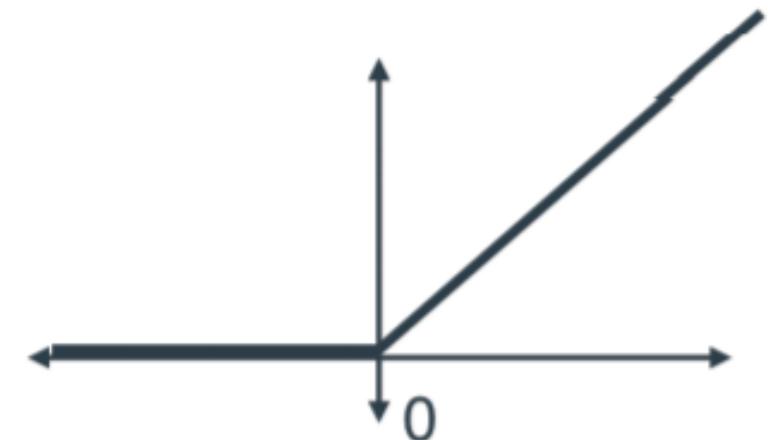
Sigmoid



Hyperbolic tangent
(tanh)



Rectified linear unit
(ReLU)



RECTIFIED LINEAR UNIT (RELU)

- A much more popular activation that is commonly used in neural networks is the Rectified Linear Unit, or ReLU.
- This one is very simple, if the input is negative, the output is zero, while if the input is positive, the output is the same input.
- In other words, if $x \geq 0$, the output is x , and if $x \leq 0$, the output is 0.

Training neural networks

For example, that it outputs the numbers 1, 2, and 3. This means that the neural network has given the dog a score of 1, the cat a score of 2, and the bird a score of 3. Let's record that.

- $\text{Score(dog)} = 1$
- $\text{Score(cat)} = 2$
- $\text{Score(bird)} = 3.$

Training neural networks

- I can think of one, the exponential function! What if we raise e to the power of the scores?

- $e^{Score(dog)} = e^1 = 2.718$
- $e^{Score(cat)} = e^2 = 7.389$
- $e^{Score(bird)} = e^3 = 20.086.$

Now if we divide the three of them by their sum, which is 30.193, we get the following probabilities:

- Probability(dog) = 0.090
- Probability(cat) = 0.245
- Probability(bird) = 0.665.

SOFTMAX FUNCTION

- Given n scores, a_1, a_2, \dots, a_n , the softmax function will output the n numbers (p_1, p_2, \dots, p_n) which add to 1, and correspond to probabilities, where

$$p_i = \frac{e^{a_i}}{e^{a_1} + e^{a_2} + \dots + e^{a_n}}.$$

Training neural networks

- We have the following:
 - Score(dog) = 2
 - Score(not dog) = 0

Now, we calculate the softmax.

- Probability(dog) = $\frac{e^2}{e^2+e^0}$
- Probability(not dog) = $\frac{e^0}{e^2+e^0}$

Training neural networks

Remembering that $e^0 = 1$, we get that the probability that the image is a dog is $\frac{e^{\epsilon}}{e^2+1}$. Multiplying the numerator and the denominator by e^2 , we get that

$$P(\text{dog}) = \frac{1}{1+e^{-2}} = \sigma(2).$$

Training neural networks

LEARNING RATE - THE LENGTH OF THE STEP THAT WE USE DURING OUR TRAINING

- Just like in linear or logistic regression, neural networks use the learning rate as the size of each step that it takes during the training process.
- This is a very important hyperparameter, because a learning rate that is too big will give very large steps to get to the solution, but it may accidentally miss it.

Training neural networks

NUMBER OF EPOCHS - THE NUMBER OF STEPS WE USE FOR OUR TRAINING

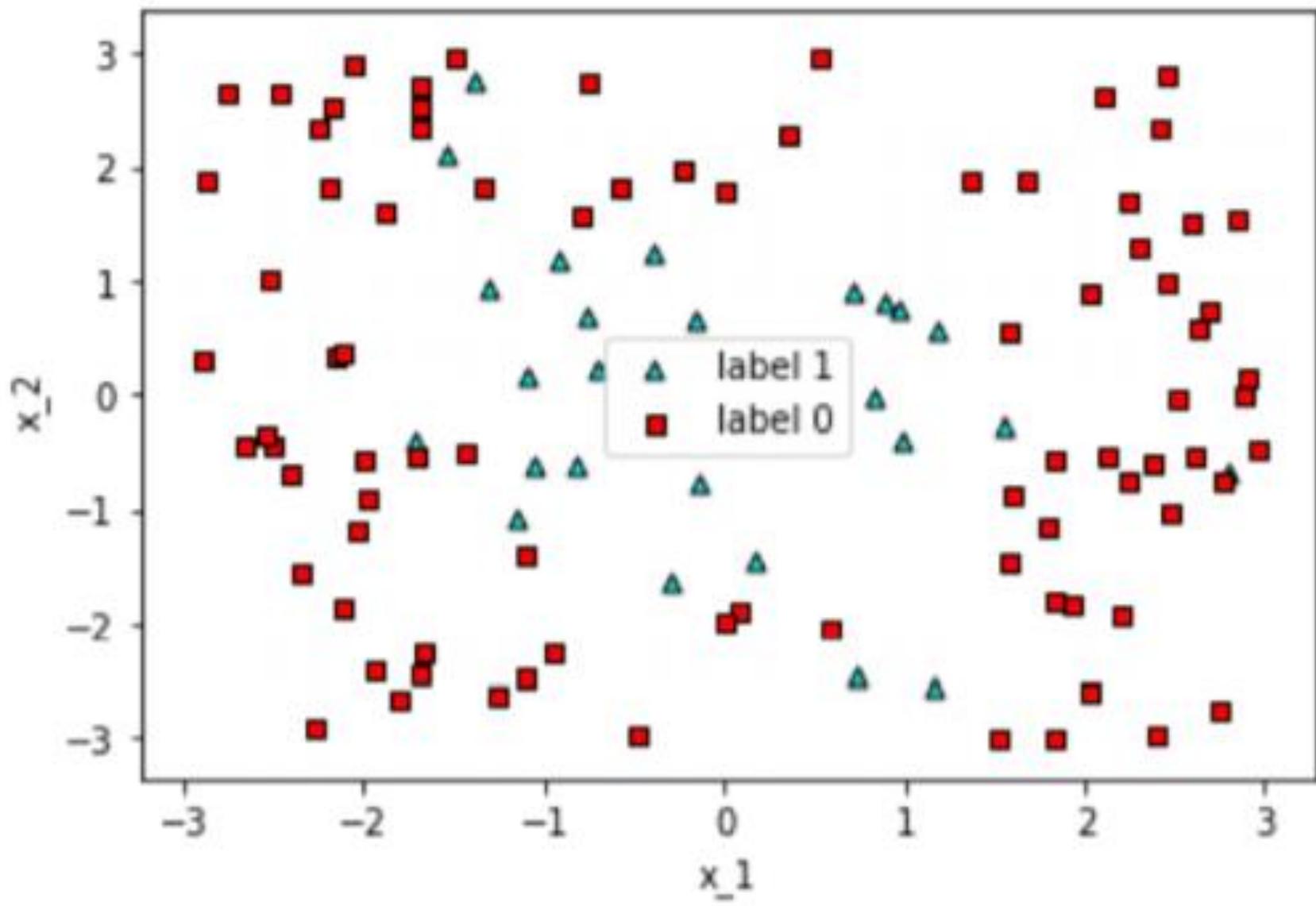
Normally, the more computational power we have, the more epochs we should use. However, there are other ways to tell when to stop the training, such as the following:

- When the loss function reaches a certain value that we have predetermined.
- When the loss function doesn't decrease during several epochs.

Training neural networks

BATCH VS MINI-BATCH VS STOCHASTIC GRADIENT DESCENT - HOW MANY POINTS AT A TIME ENTER THE TRAINING PROCESS

- The training process of a neural network can be very slow if our dataset is very large. Luckily, there are tricks to speed this process up.
- In the same way that we eat a large sandwich by taking small bites, we can train a model on a very large dataset by dividing it into many batches (chosen at random), and applying backpropagation in each one of these batches separately.



How to code a neural network in Keras

x_1	x_2	y
-0.759416	2.753240	0
-1.885278	1.629527	0
...
0.729767	-2.479655	1
-1.715920	-0.393404	1

How to code a neural network in Keras

Categorizing our data - a way to turn categorical features into numbers

- Preprocessing data is one of the most important parts of the work of a data scientist.
- There is a particular preprocessing step which is recommended when training neural networks, called categorizing the data, which I'll show you in this section.

How to code a neural network in Keras

- We'd like to one-hot encode this data, in such a way that the labels are two columns.
- The labels of these two columns are called `y_0` and `y_1`, and they are the following.

If $y = 0$, then $y_0 = 1$ and $y_1 = 0$.

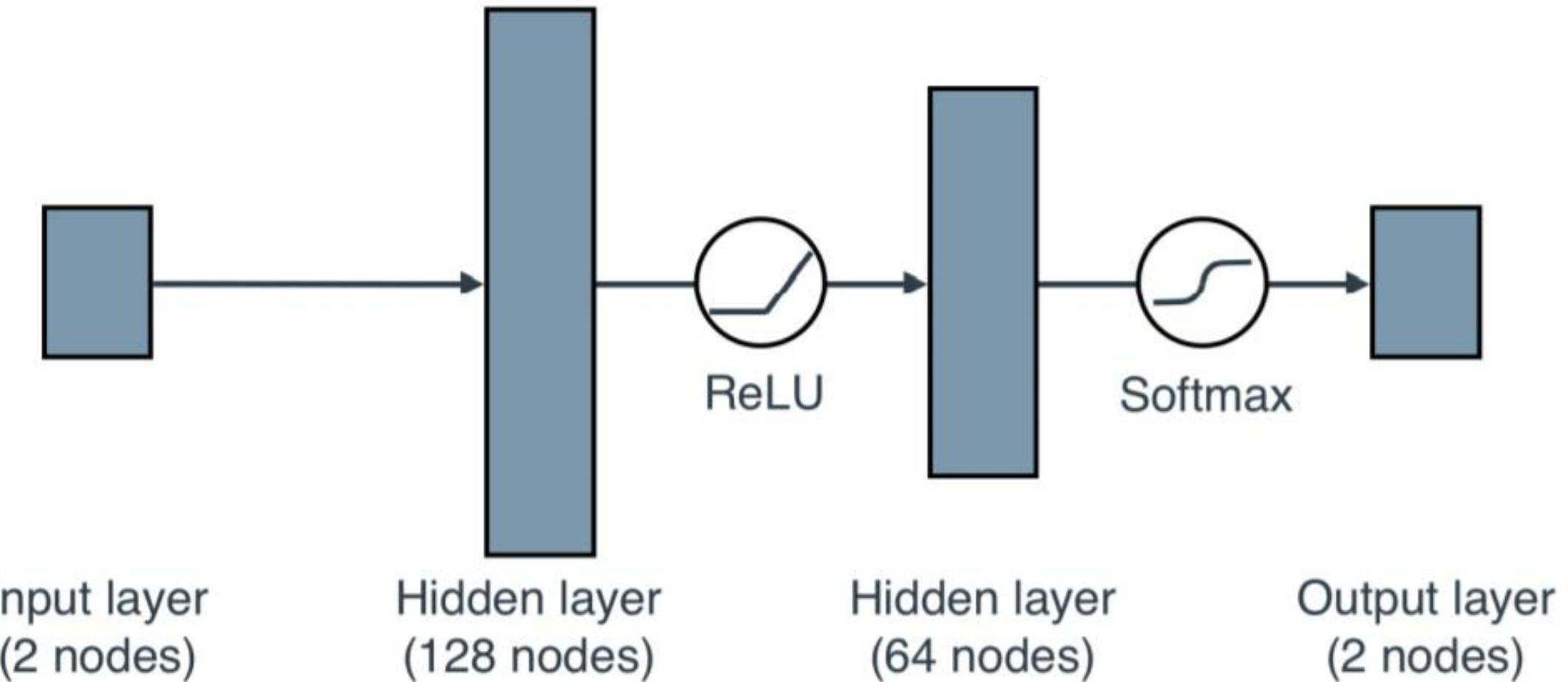
If $y = 1$, then $y_0 = 0$ and $y_1 = 1$.

- This is easily done with the following command
- ```
from tensorflow.keras.utils import to_categorical
y = np.array(to_categorical(y, 2))
```

# The architecture of a neural network that we'll use to train this dataset

- Input layer
  - Size: 2
- First hidden layer
  - Size: 128
  - Activation function: ReLU
- Second hidden layer
  - Size: 64
  - Activation function: ReLU
- Output layer (size = 2)
  - Size: 2
  - Activation function: Softmax

# How to code a neural network in Keras



# How to code a neural network in Keras

- This can all be done in Keras in only a few lines of code.
- The first thing we have to do is some useful imports.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
```

# How to code a neural network in Keras

```
model = Sequential()
```

- Now, we proceed to add the hidden layers and the output layer, as follows:

```
model.add(Dense(128, activation='relu', input_shape=(2,)))
#A
```

```
model.add(Dropout(.2)) #B
```

```
model.add(Dense(64, activation='relu')) #C
```

```
model.add(Dropout(.1))
```

```
model.add(Dense(2, activation='softmax')) #D
```

# How to code a neural network in Keras

- Once the model is defined, we need to compile it with the following line of code.

```
model.compile(loss = 'categorical_crossentropy',
 optimizer='adam', metrics=['accuracy'])
```

Model: "sequential\_5"

| Layer (type)            | Output Shape | Param # |
|-------------------------|--------------|---------|
| dense_10 (Dense)        | (None, 128)  | 384     |
| dropout_6 (Dropout)     | (None, 128)  | 0       |
| dense_11 (Dense)        | (None, 64)   | 8256    |
| dropout_7 (Dropout)     | (None, 64)   | 0       |
| dense_12 (Dense)        | (None, 2)    | 130     |
| <hr/>                   |              |         |
| Total params: 8,770     |              |         |
| Trainable params: 8,770 |              |         |
| Non-trainable params: 0 |              |         |

# How to code a neural network in Keras

## Training the model in Keras

In the previous section we defined the model, and now we get to train it. For training, only one simple line of code suffices:

```
model.fit(X, categorized_y, epochs=200, batch_size=10)
```

Let's examine each of the inputs to this fit function.

- X and categorized\_y: The features and labels, respectively.
- epochs: The number of times we run backpropagation on our whole dataset. Here we do it 200 times.
- batch\_size: The length of the batches that we use to train our model. Here we are introducing our data to the model in batches of 10. For a small case dataset like this one, we don't need to input it in batches, but in this example we are doing it for exposure.

# How to code a neural network in Keras

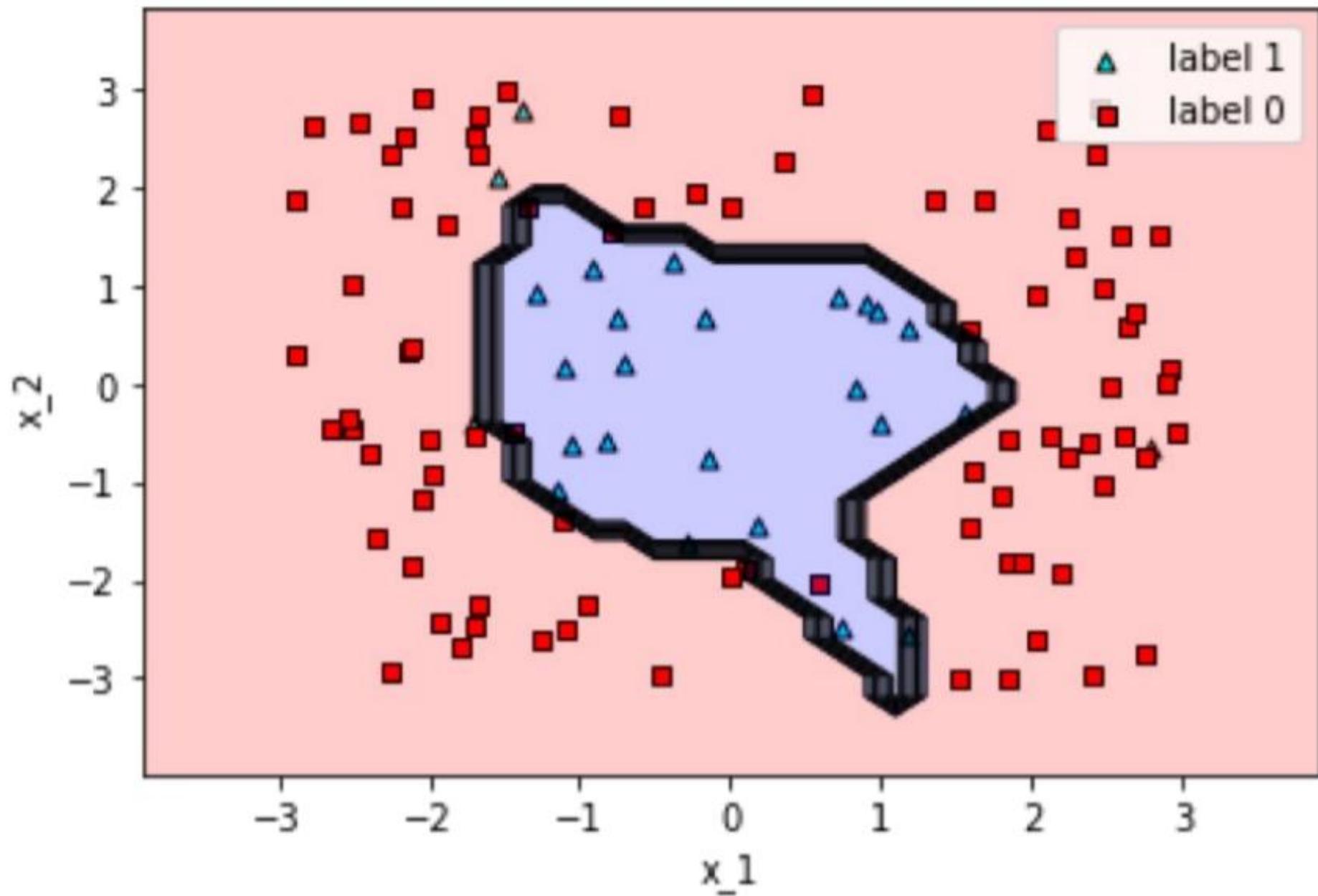
Epoch 1/200

11/11 [=====] - 0s 1ms/step - loss: 0.5906 - accuracy: 0.6273

Epoch 200/200

11/11 [=====] - 0s 2ms/step - loss: 0.1953 - accuracy: 0.9091

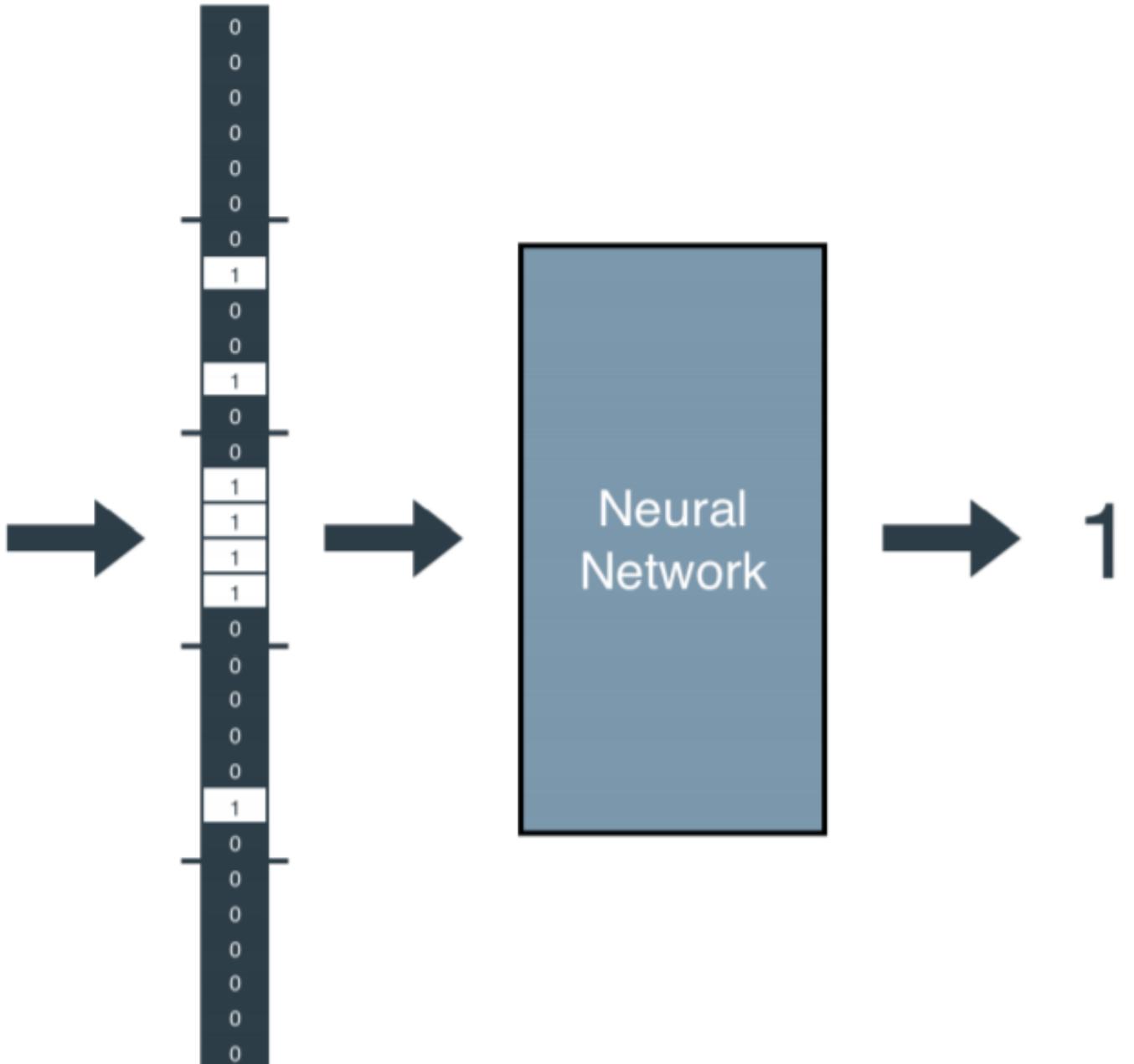
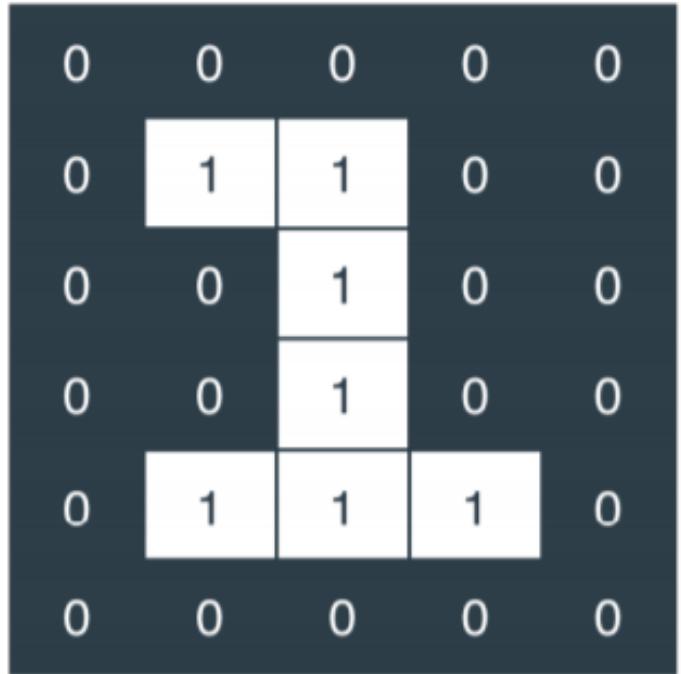
- The final accuracy of the model is 0.9091, which is pretty good.
- Now, let's plot the boundary with our `plot_model` function to have a visual of how the neural network performed



# Other more complicated architectures and some sci-fi applications

## **How neural networks see - Image recognition**

- Neural networks are great with images, and there are many applications where one can use this, such as the following:
- Image recognition: The input is an image, and the output is the label on the image.
- Some famous datasets used for image recognition are the following:
  - MNIST: Handwritten digits in 28 by 28 gray scale images.



# Other more complicated architectures and some sci-fi applications

## **How neural networks talk - Natural language processing**

- One of the most fascinating applications of neural networks is when we can get them to talk to us.
- This involves listening to what we say or reading what we write, analyzing it, and being able to respond or take action.
- The ability for computers to understand and process language is called natural language processing. Neural networks have had a lot of success in natural language processing

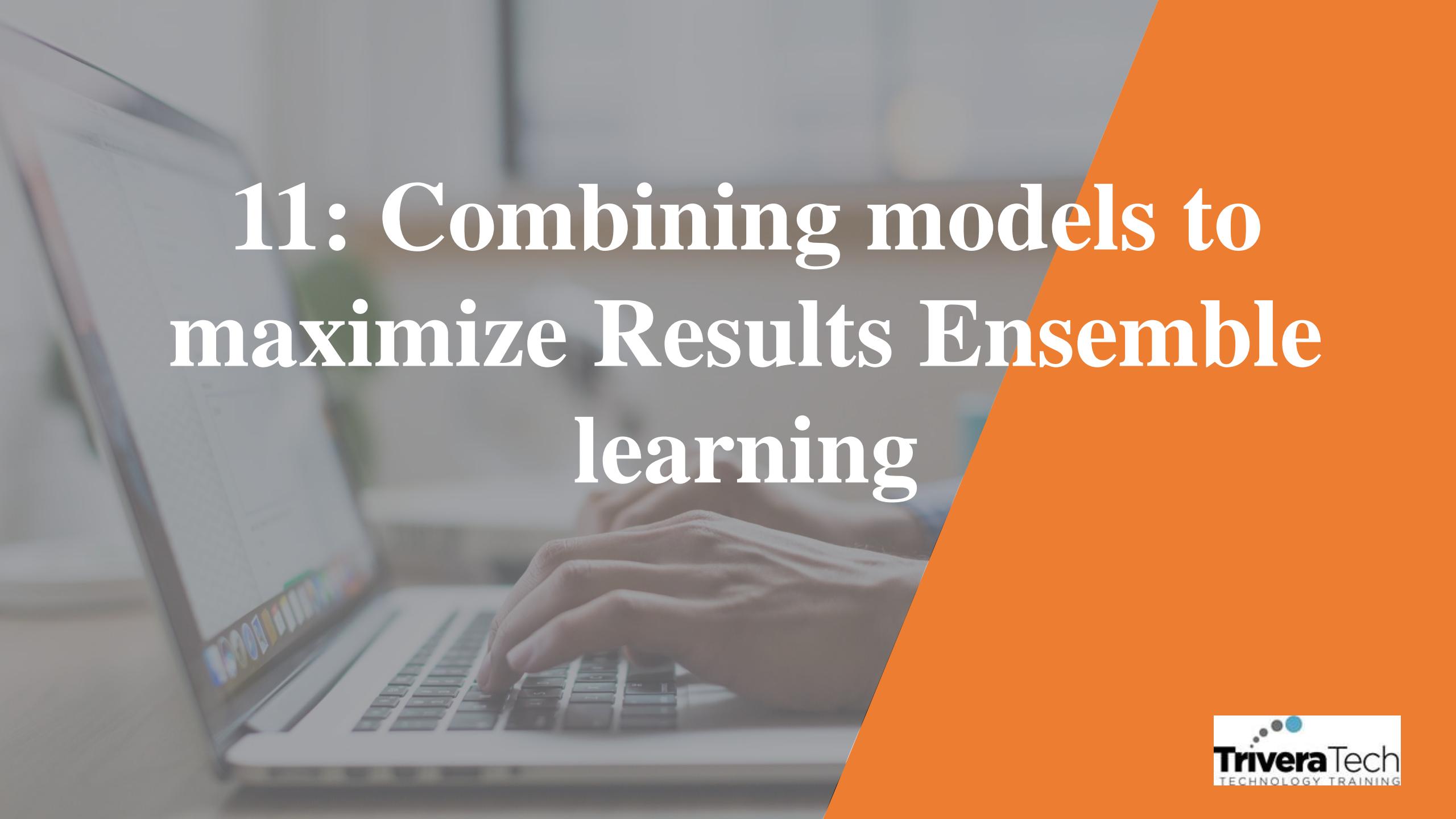
# Other more complicated architectures and some sci-fi applications

## **How neural networks generate faces that look real – Generative adversarial networks**

- In my opinion, the most fascinating among all the current applications of neural networks is in generation.
- So far, neural networks have worked well in predictive machine learning, namely, being able to answer questions successfully.
- For example, a machine learning model can answer questions such as “how much is that?”, or “is this A or B?”.

# Summary

- Neural networks are a very powerful algorithm used for classification and regression.
- A neural network consists of a set of perceptrons organized in layers, where the output of one layer serves as input to the next layer.
- Their complexity allows them to achieve great success in applications that are very difficult for other machine learning models.
- Neural networks have cutting edge applications in many areas, including image recognition and text processing.



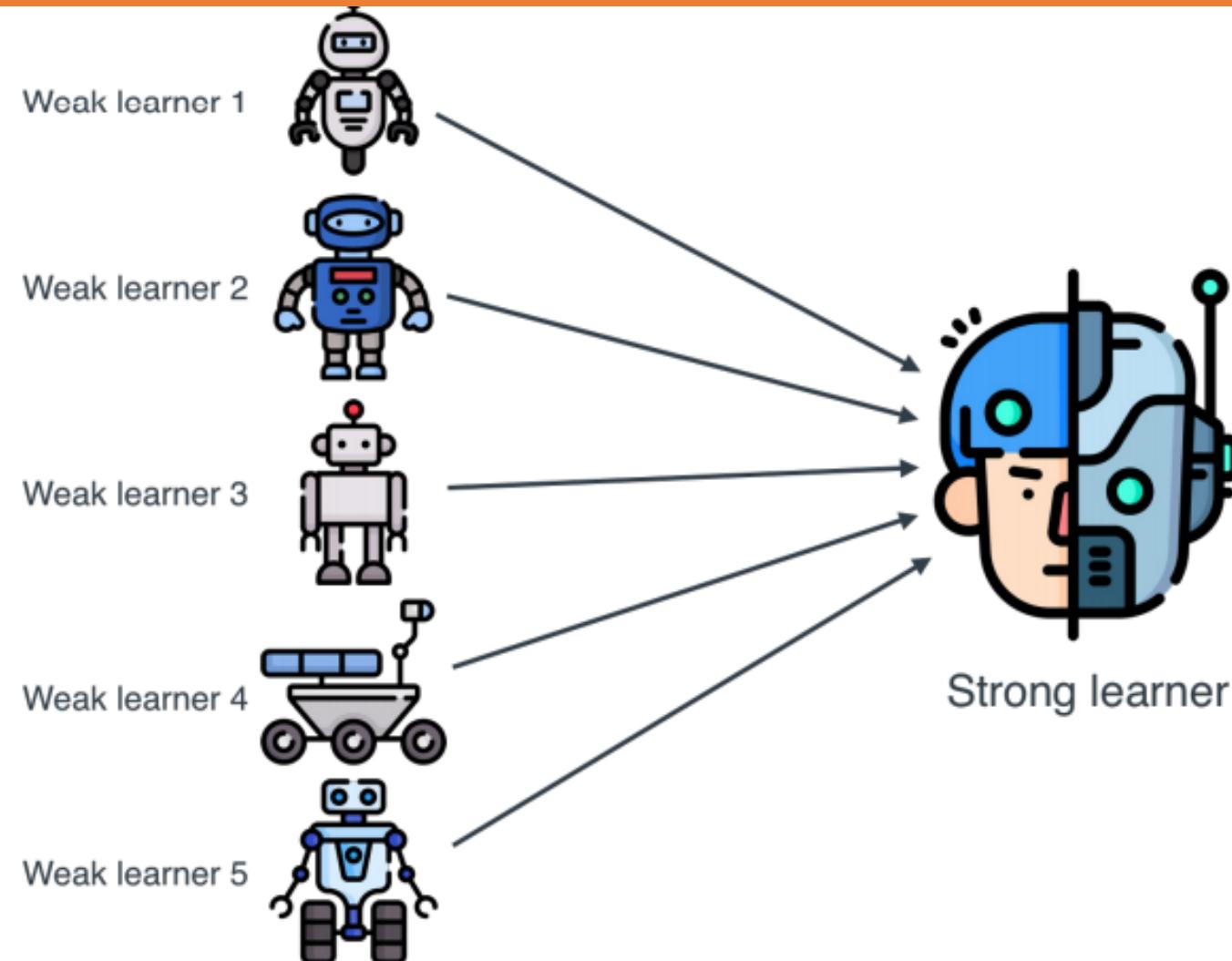
# 11: Combining models to maximize Results Ensemble learning

# Combining models to maximize results Ensemble learning

## This lesson covers

- What is ensemble learning.
- Joining several weak classifiers to form a strong classifier.
- Bagging: A method to randomly join several classifiers.
- Boosting: A method to join several classifiers in a smarter way.
- AdaBoost: A very successful example of boosting methods.

# With a little help from our friends



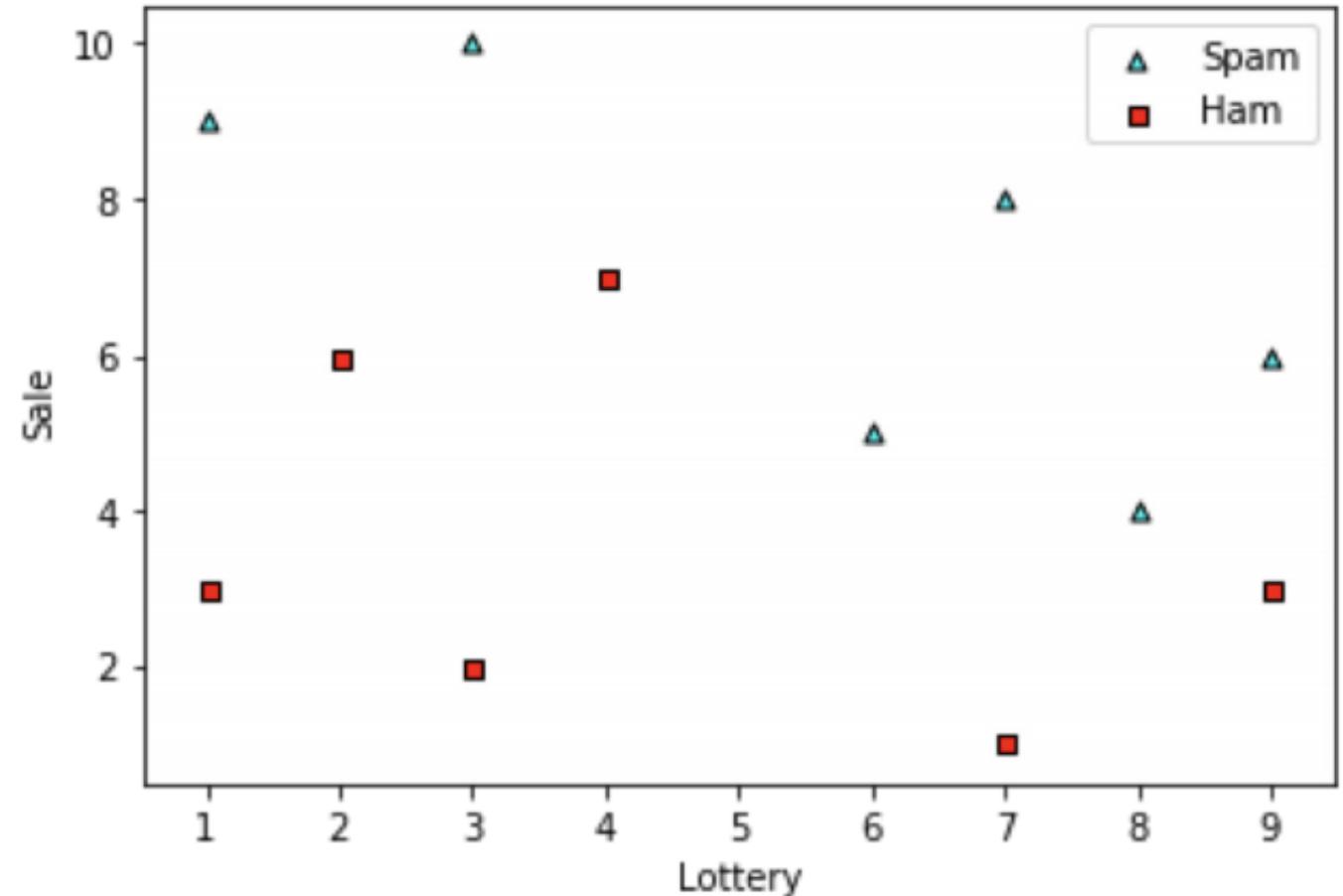
# Why an ensemble of learners? Why not just one really good learner?

| Lottery | Sale | Spam |
|---------|------|------|
| 7       | 1    | No   |
| 3       | 2    | No   |
| 3       | 9    | No   |
| 1       | 3    | No   |
| 2       | 6    | No   |
| 4       | 7    | No   |

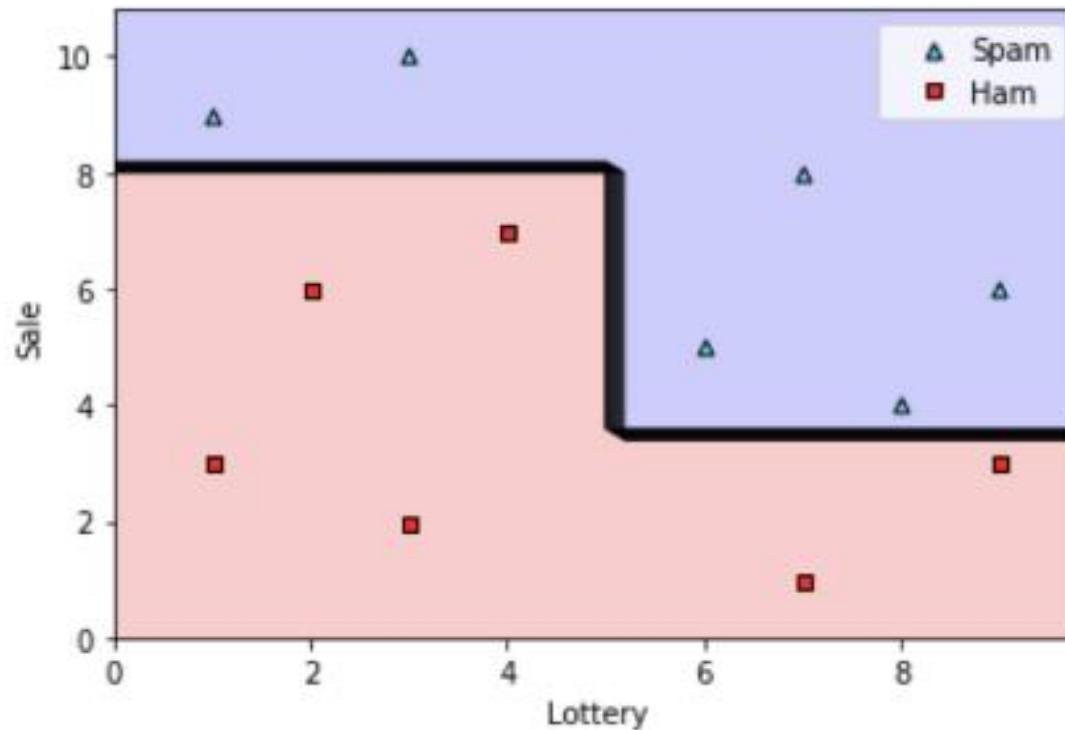
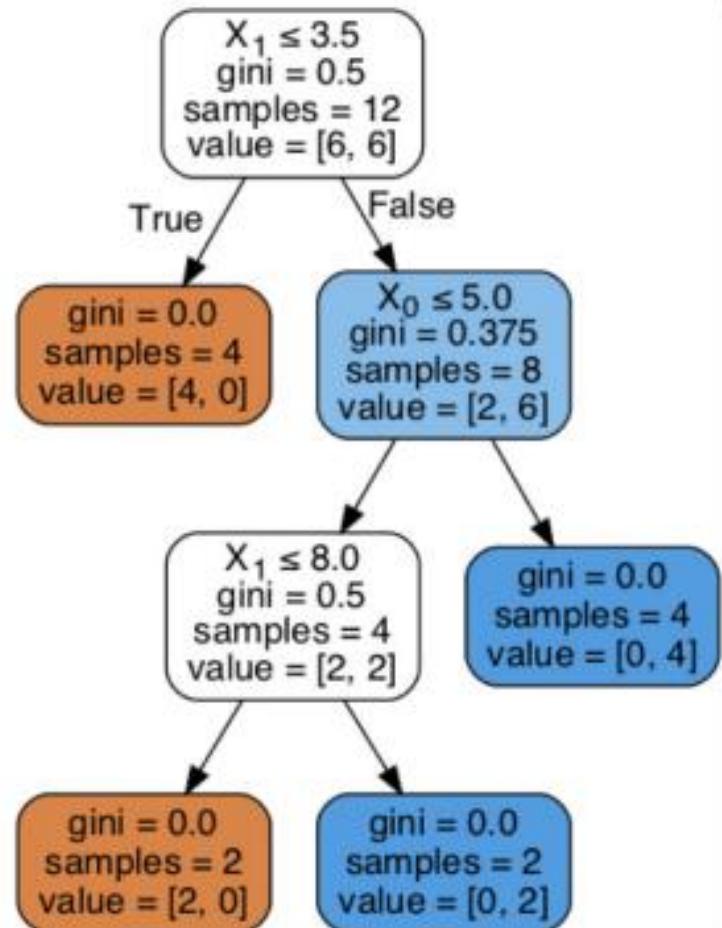
# Why an ensemble of learners? Why not just one really good learner?

|   |    |     |
|---|----|-----|
| 1 | 9  | Yes |
| 3 | 10 | Yes |
| 6 | 5  | Yes |
| 7 | 8  | Yes |
| 8 | 4  | Yes |
| 9 | 6  | Yes |

# Why an ensemble of learners? Why not just one really good learner?



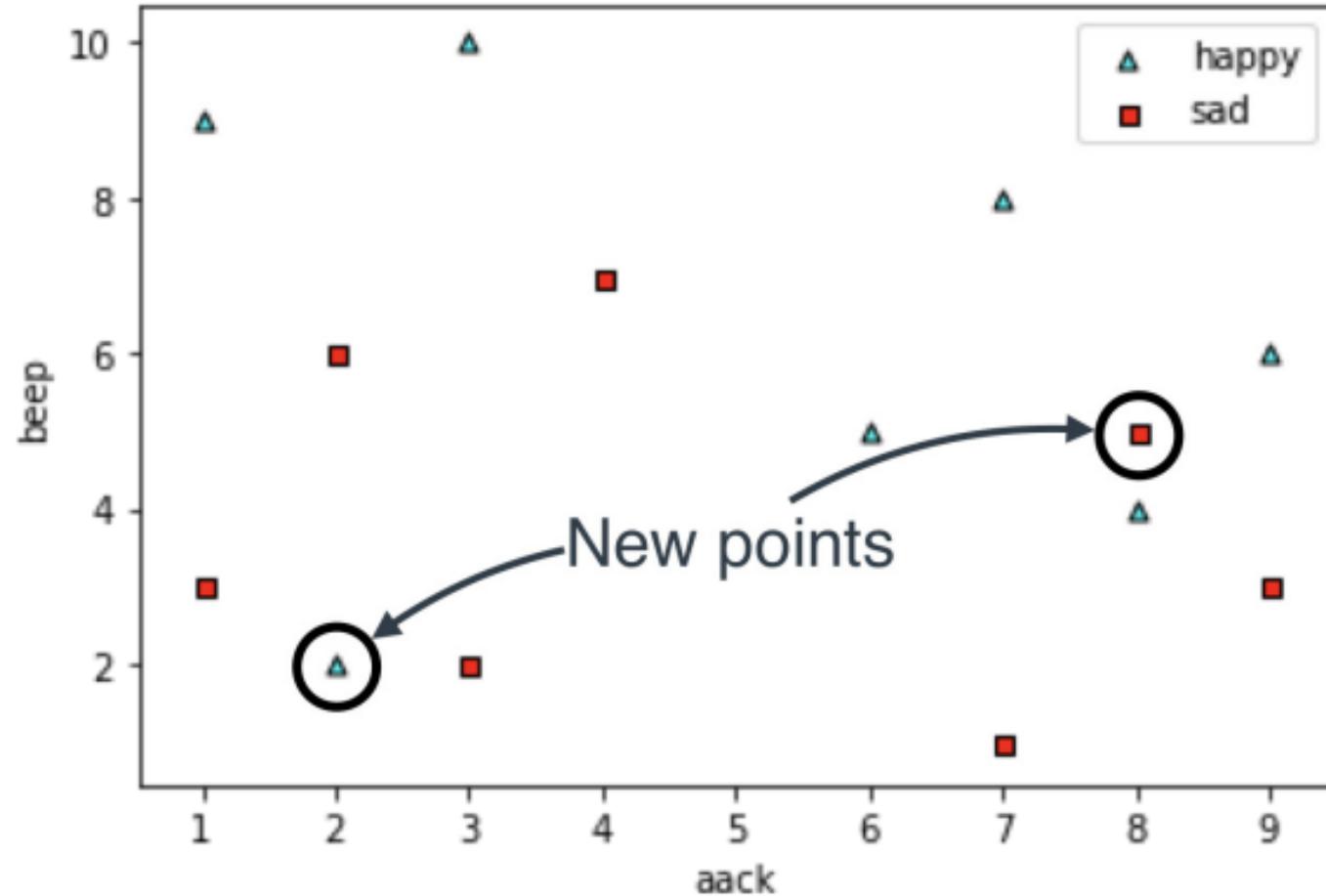
# Why an ensemble of learners? Why not just one really good learner?



# Why an ensemble of learners? Why not just one really good learner?

| Buy | Lottery | Spam |
|-----|---------|------|
| 8   | 6       | No   |
| 2   | 2       | Yes  |

# Why an ensemble of learners? Why not just one really good learner?

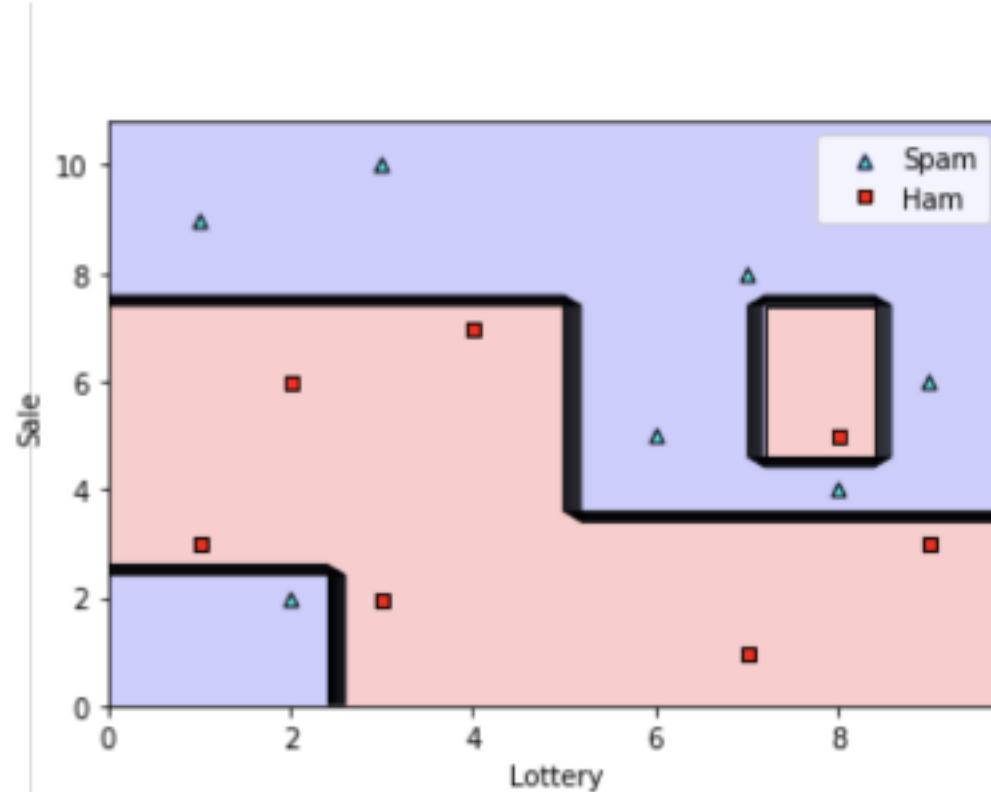


# Why an ensemble of learners? Why not just one really good learner?

- Now let's try to fit a decision tree to this data. We can use sklearn, like we did in lesson 7, with the following command in sklearn:

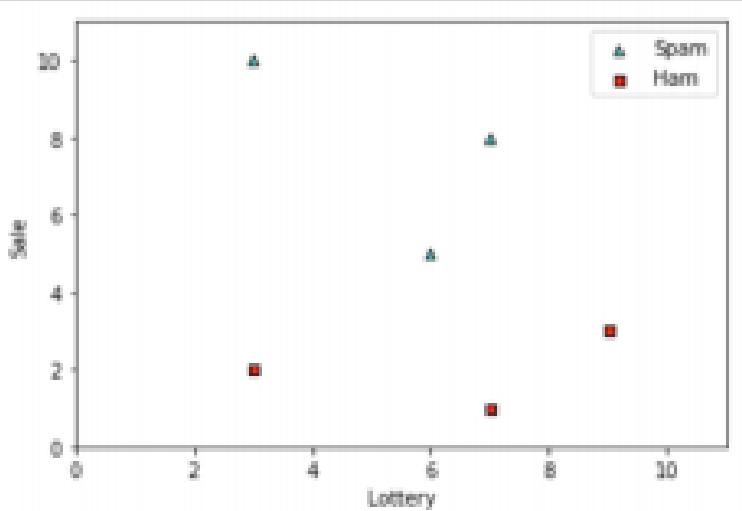
```
spam_decision_tree = DecisionTreeClassifier()
spam_decision_tree.fit(new_X, new_y)
```

# Why an ensemble of learners? Why not just one really good learner?

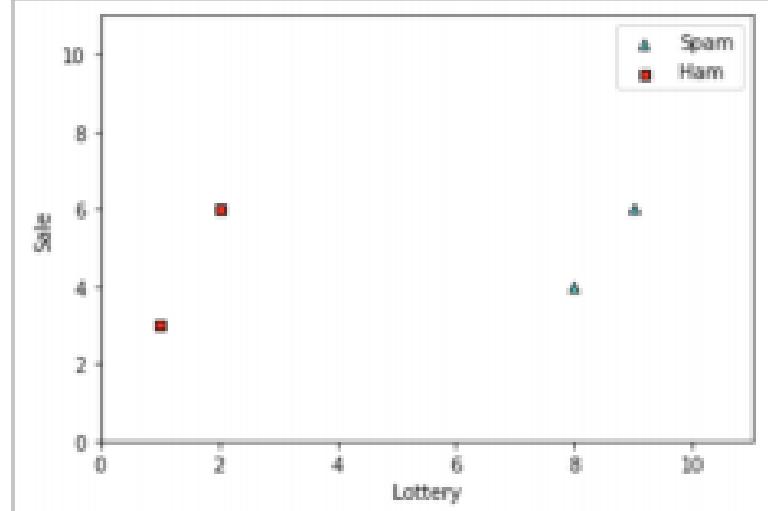


# Building random forests by joining several trees

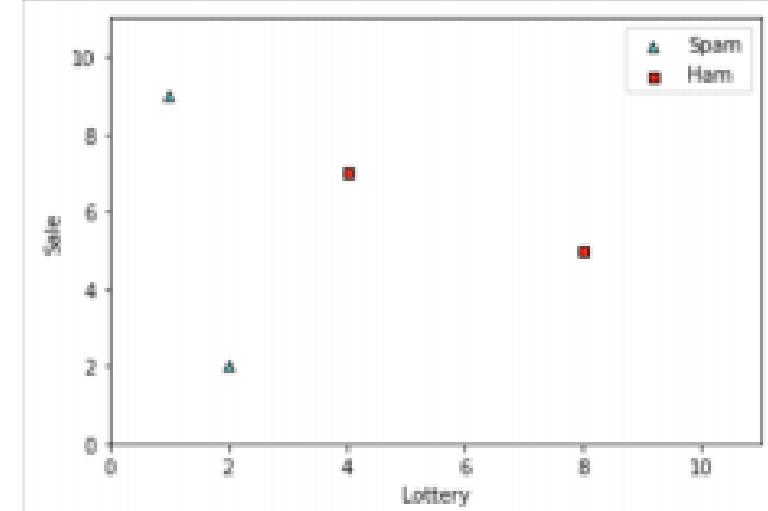
Subset 1



Subset 2

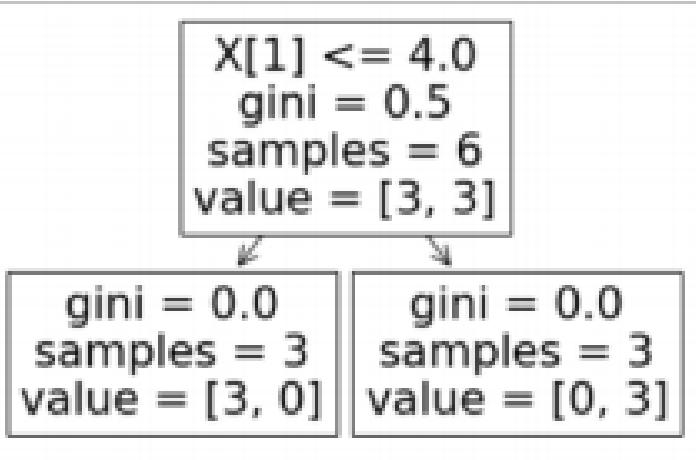


Subset 3

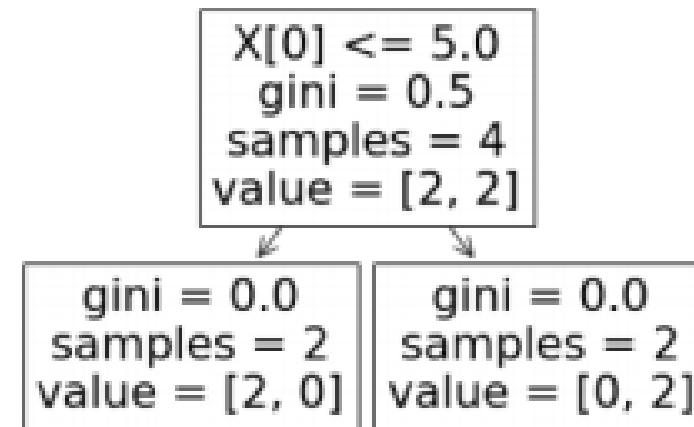


# Building random forests by joining several trees

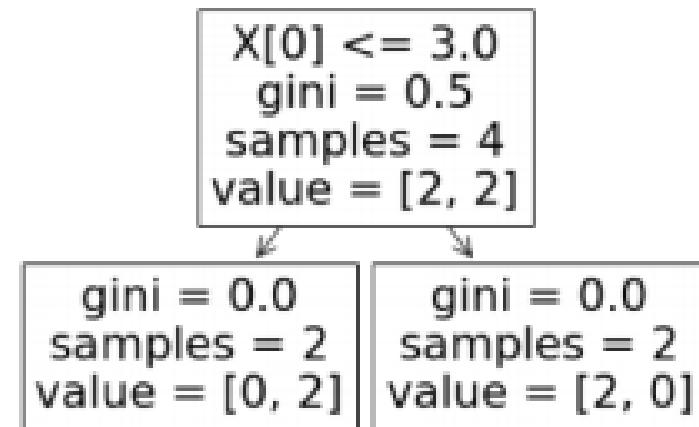
## Decision Tree 1



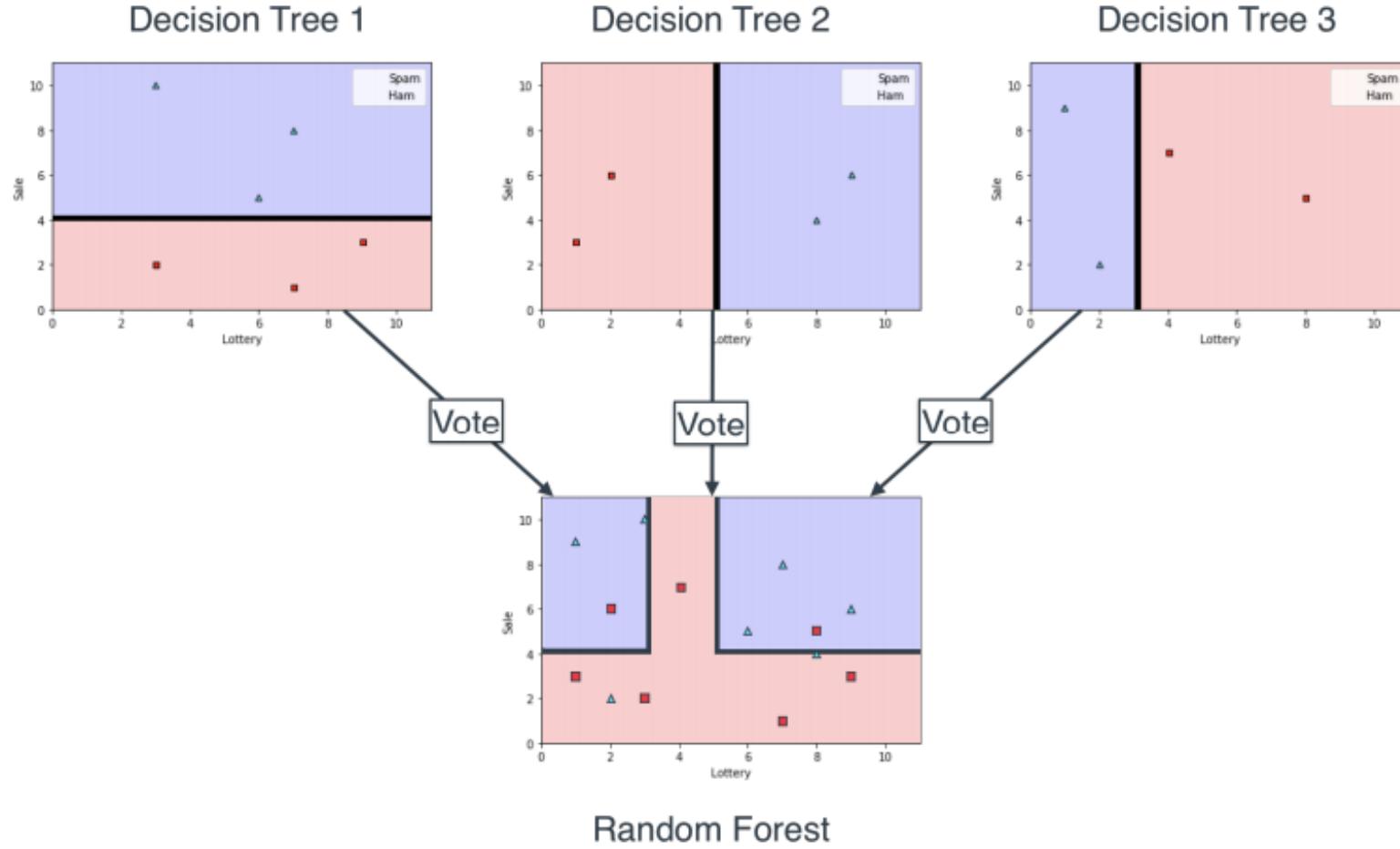
## Decision Tree 2



## Decision Tree 3



# Building random forests by joining several trees

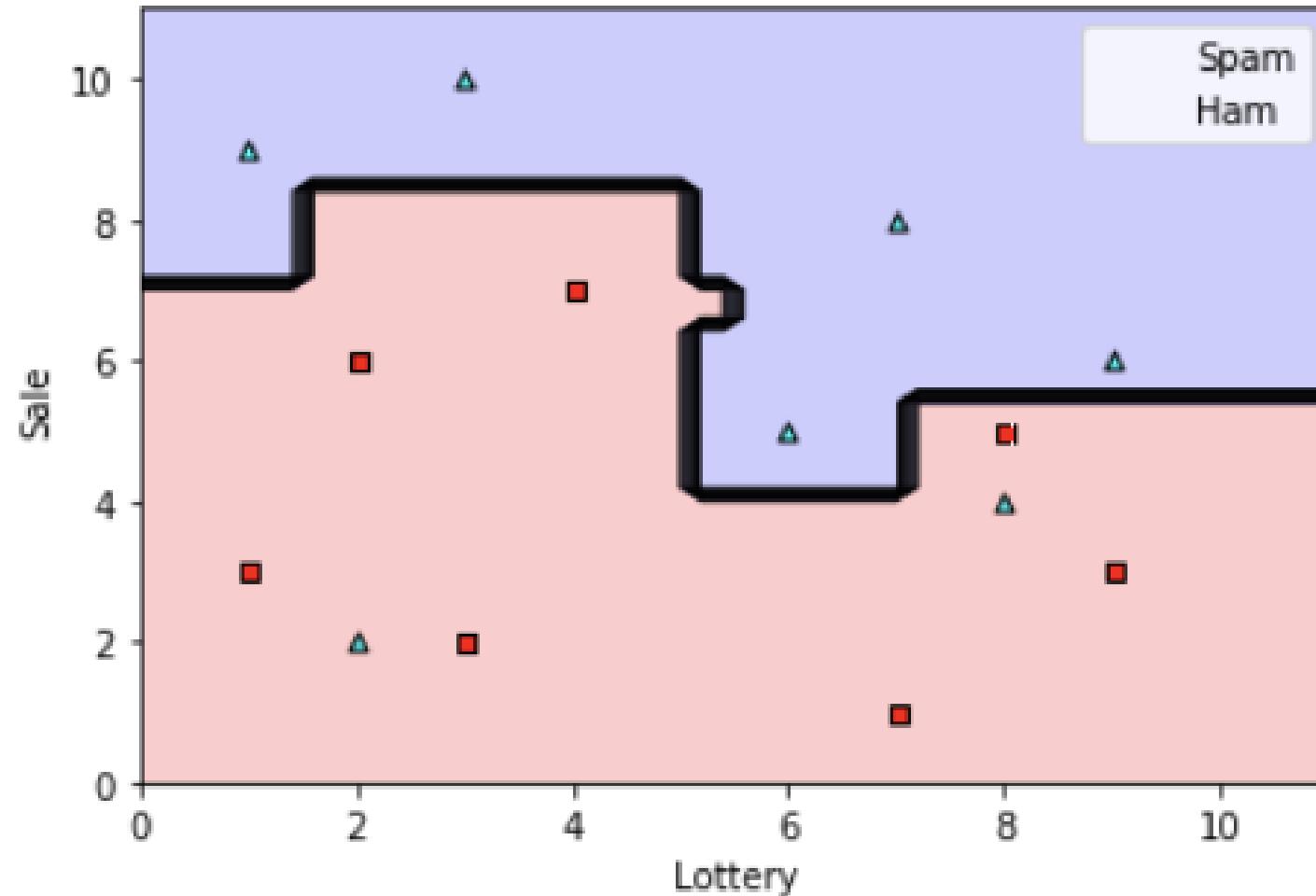


# Coding a random forest in sklearn

- We will build one with five decision trees, or ‘estimators’.
- The command is the following:

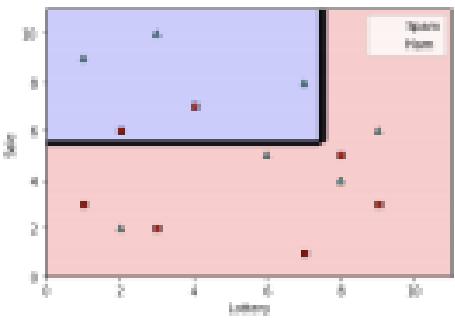
```
from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier(random_state=0, n_estimators=5)
random_forest_model.fit(new_X, new_y)
random_forest_model.score(new_X, new_y)
```

# Coding a random forest in sklearn

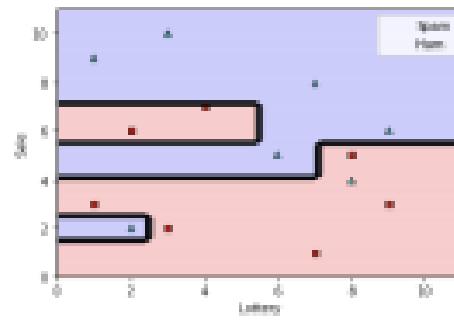


# Coding a random forest in sklearn

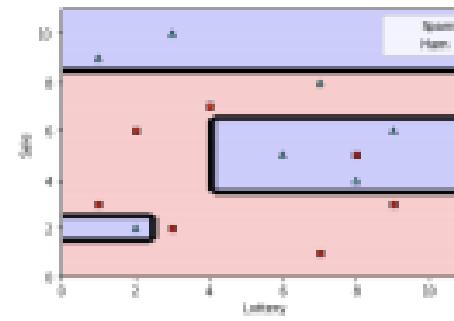
Estimator 1



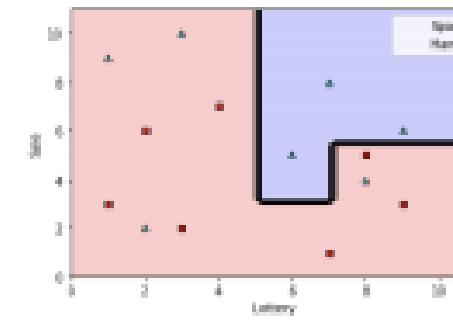
Estimator 2



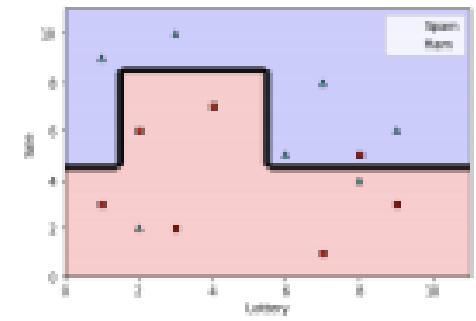
Estimator 3



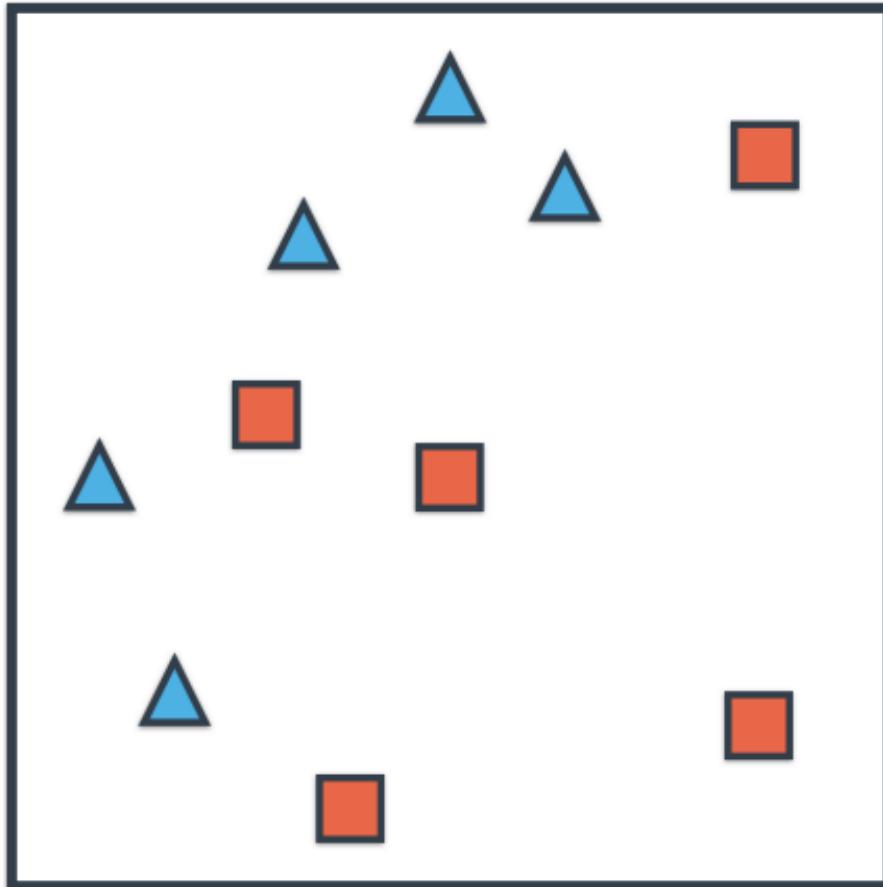
Estimator 4



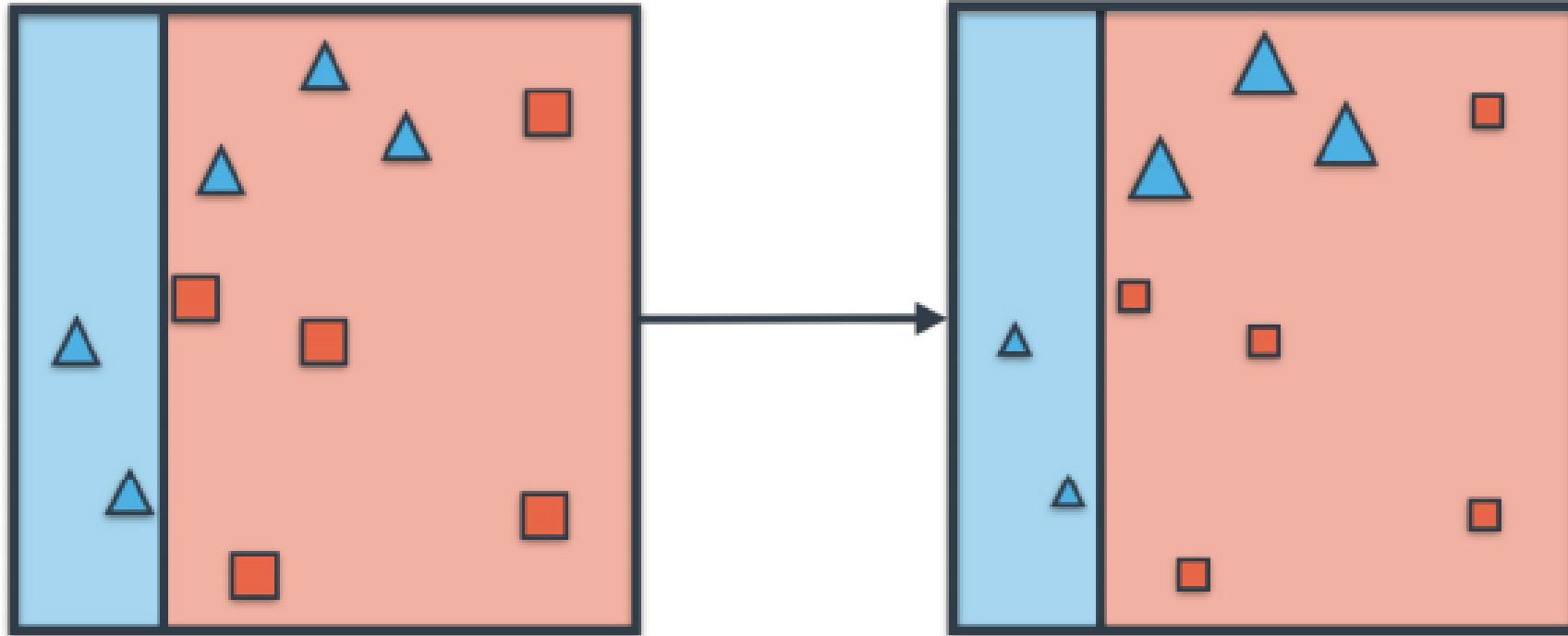
Estimator 5



# Boosting - Joining some classifiers together in a smarter way to get a stronger classifier



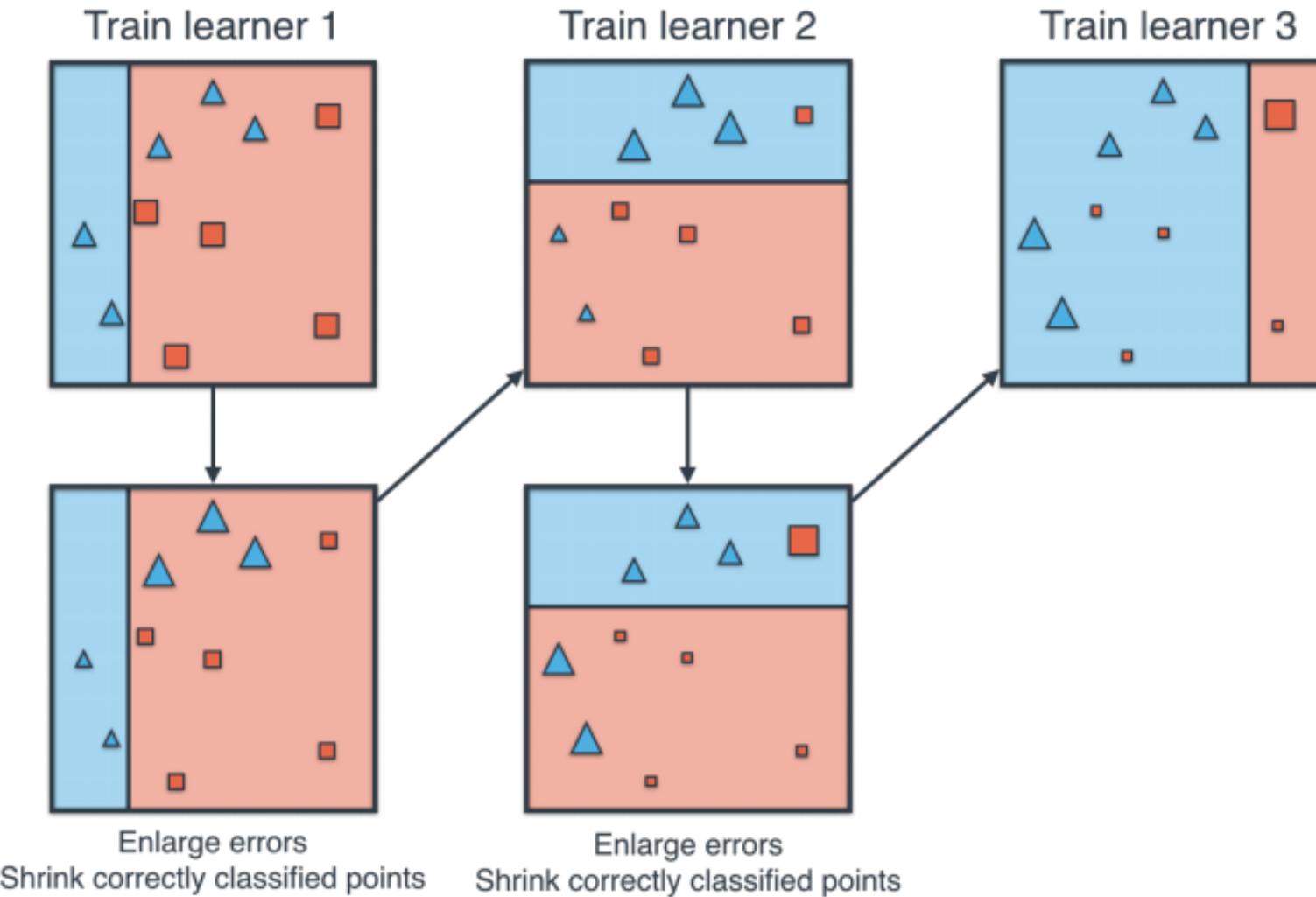
# A big picture of AdaBoost



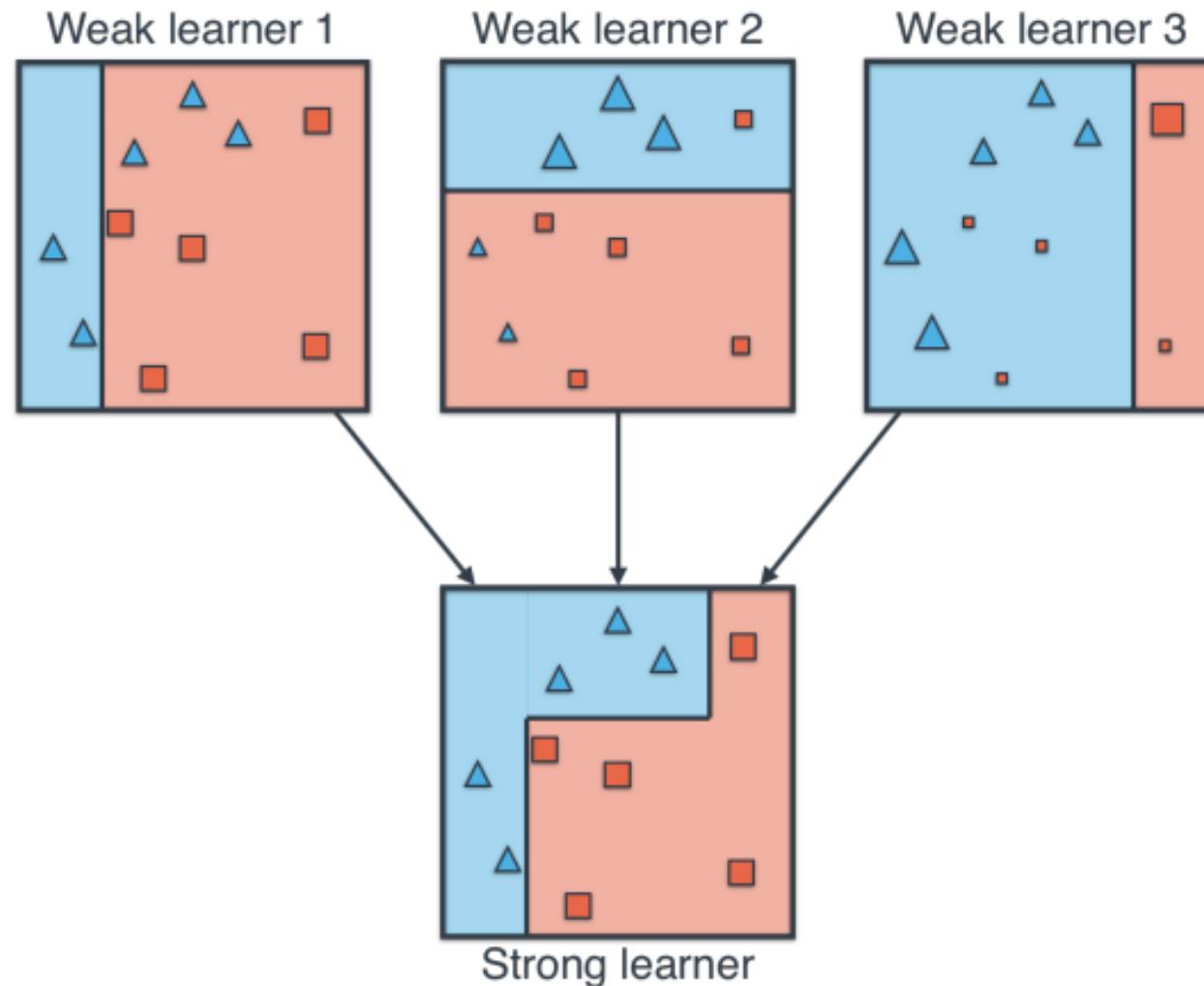
Train learner 1

Enlarge errors  
Shrink correctly classified points

# A big picture of AdaBoost



# A big picture of AdaBoost



# A detailed (mathematical) picture of AdaBoost

$$P(\text{red ball}) = \frac{\# \text{ red balls}}{\# \text{ balls}},$$

Then the formula for odds (the odds ratio--OR) is

$$OR(\text{red ball}) = \frac{\# \text{ red balls}}{\# \text{ blue balls}}.$$

# A detailed (mathematical) picture of AdaBoost

Notice that since the total number of balls is #balls = #red balls + #blue balls, then we can conclude that

$$OR(\text{red ball}) = \frac{\# \text{ red balls}}{\# \text{ balls} - \# \text{ red balls}}.$$

From here, we can see that in general, probability and odds are related via the following equation:

$$OR = \frac{P}{1 - P},$$

where OR is the odds ratio. In the previous example, if the probability of a red ball is  $P(\text{red ball}) = \frac{2}{3}$ , then the odds ratio is:

$$OR(\text{red ball}) = \frac{P(\text{red ball})}{1 - P(\text{red ball})} = \frac{\frac{2}{3}}{1 - \frac{2}{3}} = 2.$$

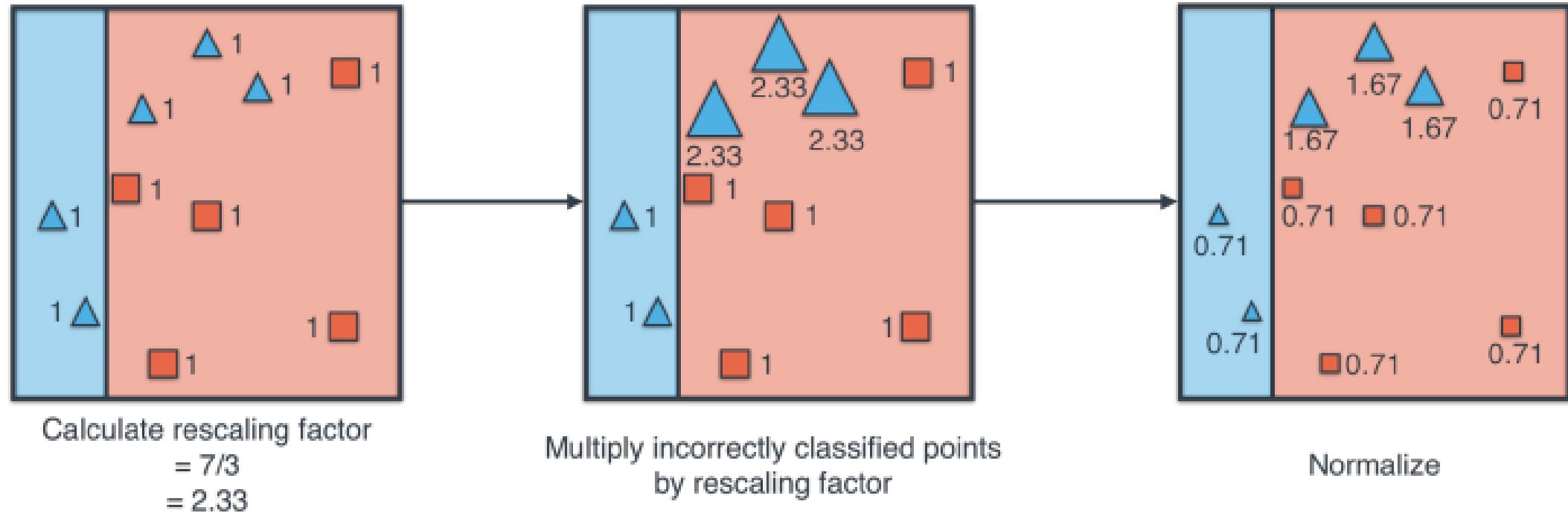
# A detailed (mathematical) picture of AdaBoost

$$\text{Rescaling factor} = \frac{\text{Number of correctly classified points}}{\text{Number of incorrectly classified points}}.$$

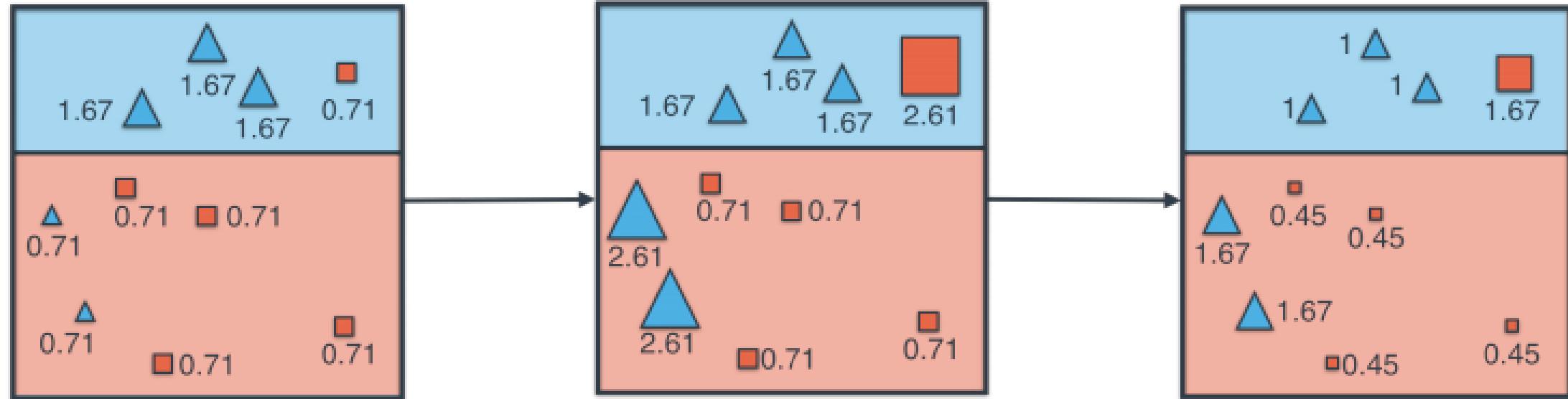
As a matter of fact, we can do better. Since we'll change the weights of the points during the process, the better way to formulate the rescaling factor is the following.

$$\text{Rescaling factor} = \frac{\text{Sum of weights of correctly classified points}}{\text{Sum of weights of incorrectly classified points}}.$$

# A detailed (mathematical) picture of AdaBoost



# A detailed (mathematical) picture of AdaBoost

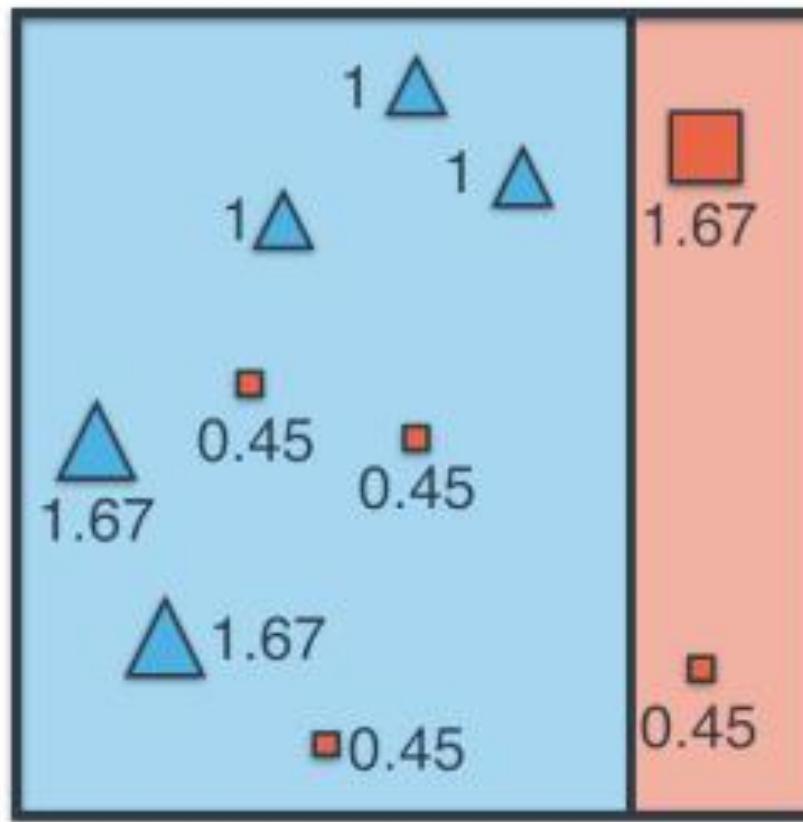


Calculate rescaling factor  
 $= 7.84/2.13$   
 $= 3.68$

Multiply incorrectly classified points  
by rescaling factor

Normalize

# A detailed (mathematical) picture of AdaBoost



Calculate rescaling factor  
 $= 8.46/1.35 = 6.27$

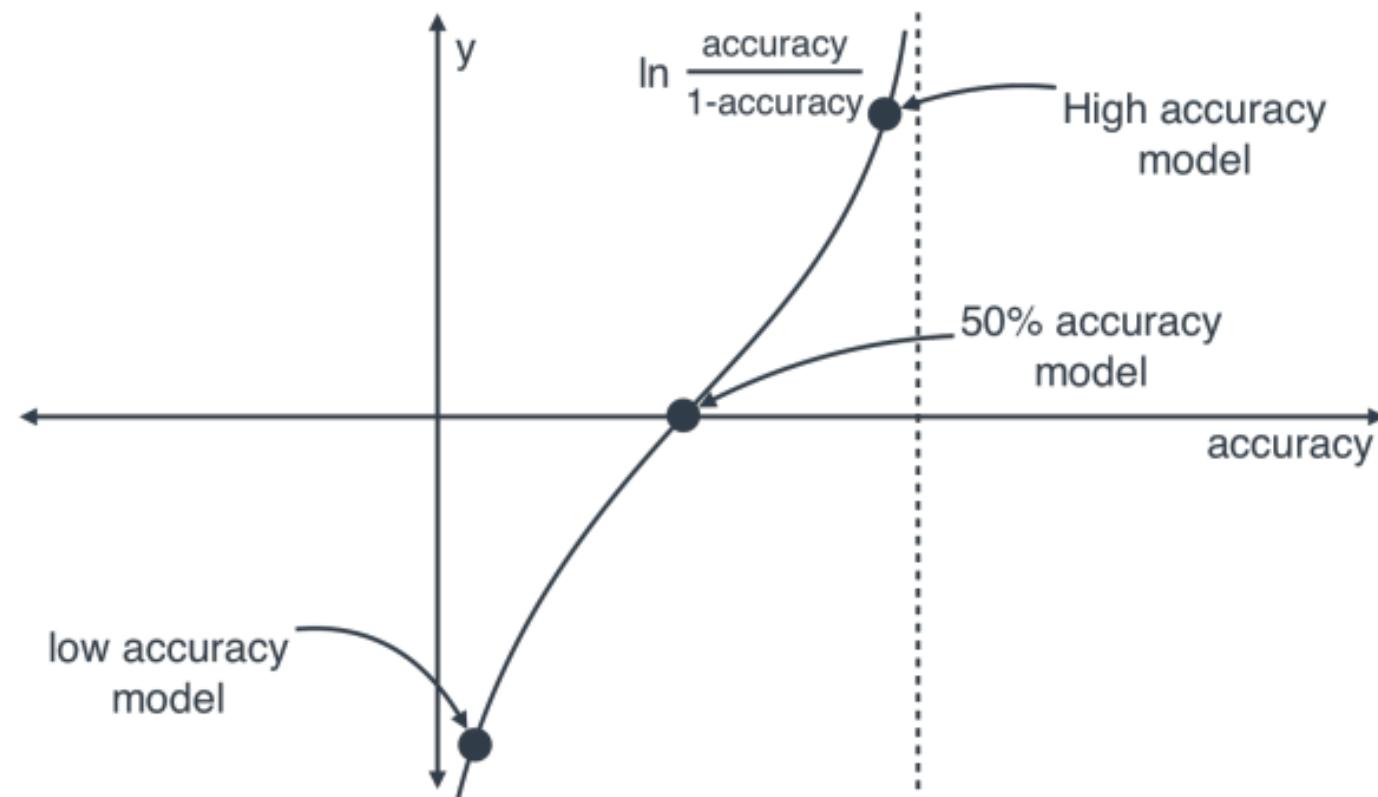
# A detailed (mathematical) picture of AdaBoost

| Accuracy | Odds = #Correct/#Errors | In(Odds) |
|----------|-------------------------|----------|
| 99%      | $99/1 = 99$             | 4.595    |
| 70%      | $70/30 = 2.333$         | 0.8473   |
| 50%      | $50/50 = 1$             | 0        |
| 30%      | $30/70 = 0.4286$        | -0.8473  |

# A detailed (mathematical) picture of AdaBoost

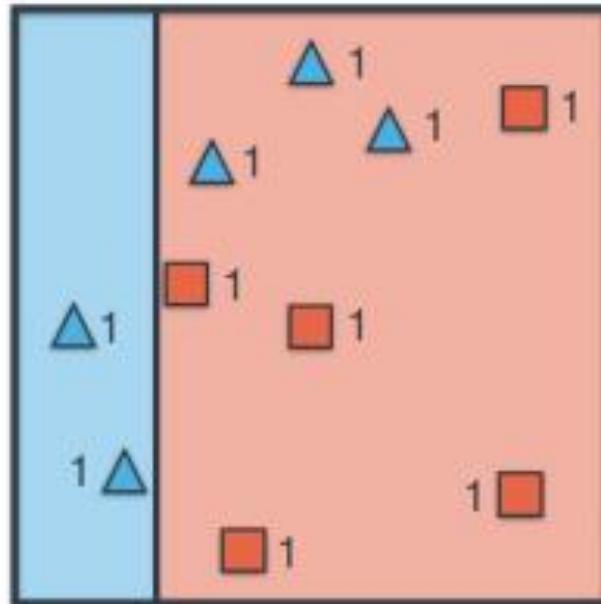
|    |                 |        |
|----|-----------------|--------|
| 1% | $1/99 = 0.0101$ | -4.595 |
|----|-----------------|--------|

The logarithm of the odds, which is commonly known as logit (short for "logistic unit"), is the weight that we assign to each model in the voting. In Figure 10.18 we can see a plot of this.



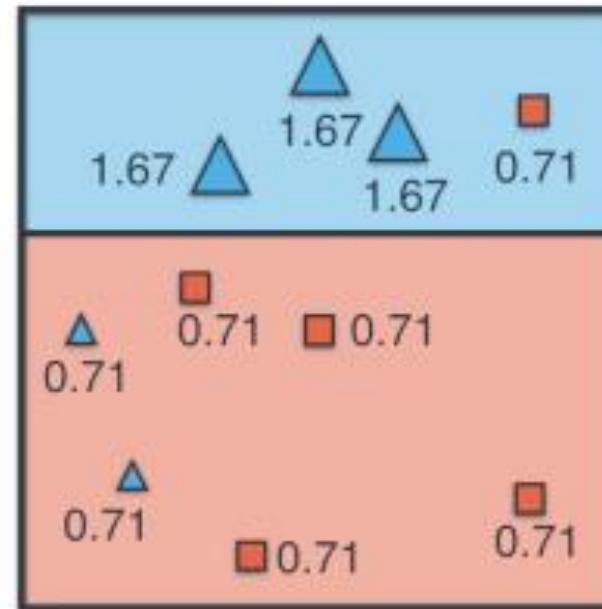
# A detailed (mathematical) picture of AdaBoost

Weak learner 1



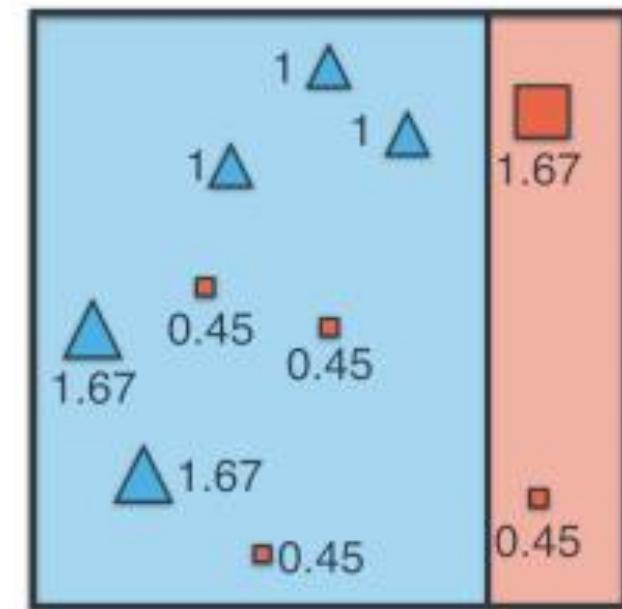
Weight = 0.846

Weak learner 2



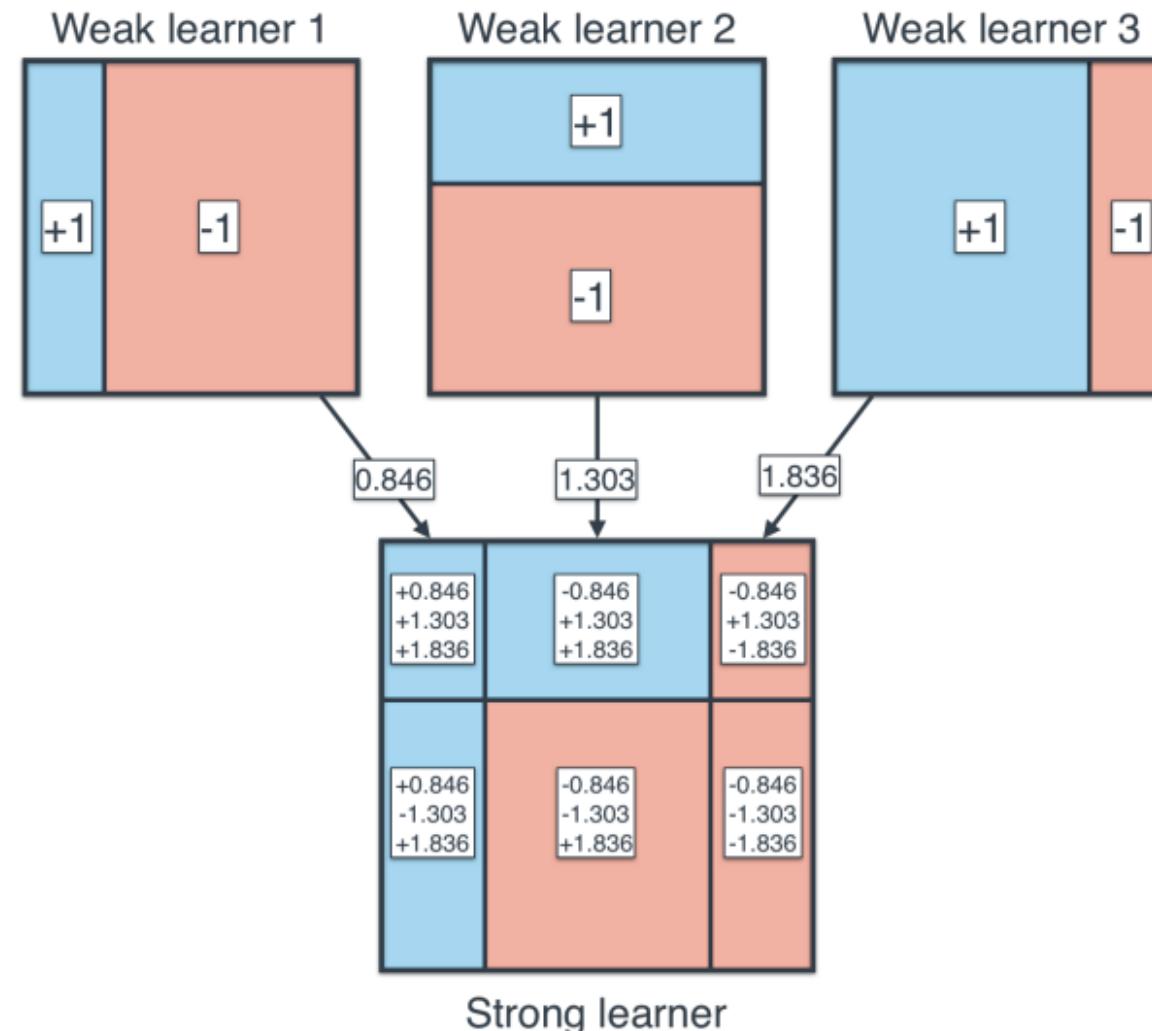
Weight = 1.303

Weak learner 3



Weight = 1.836

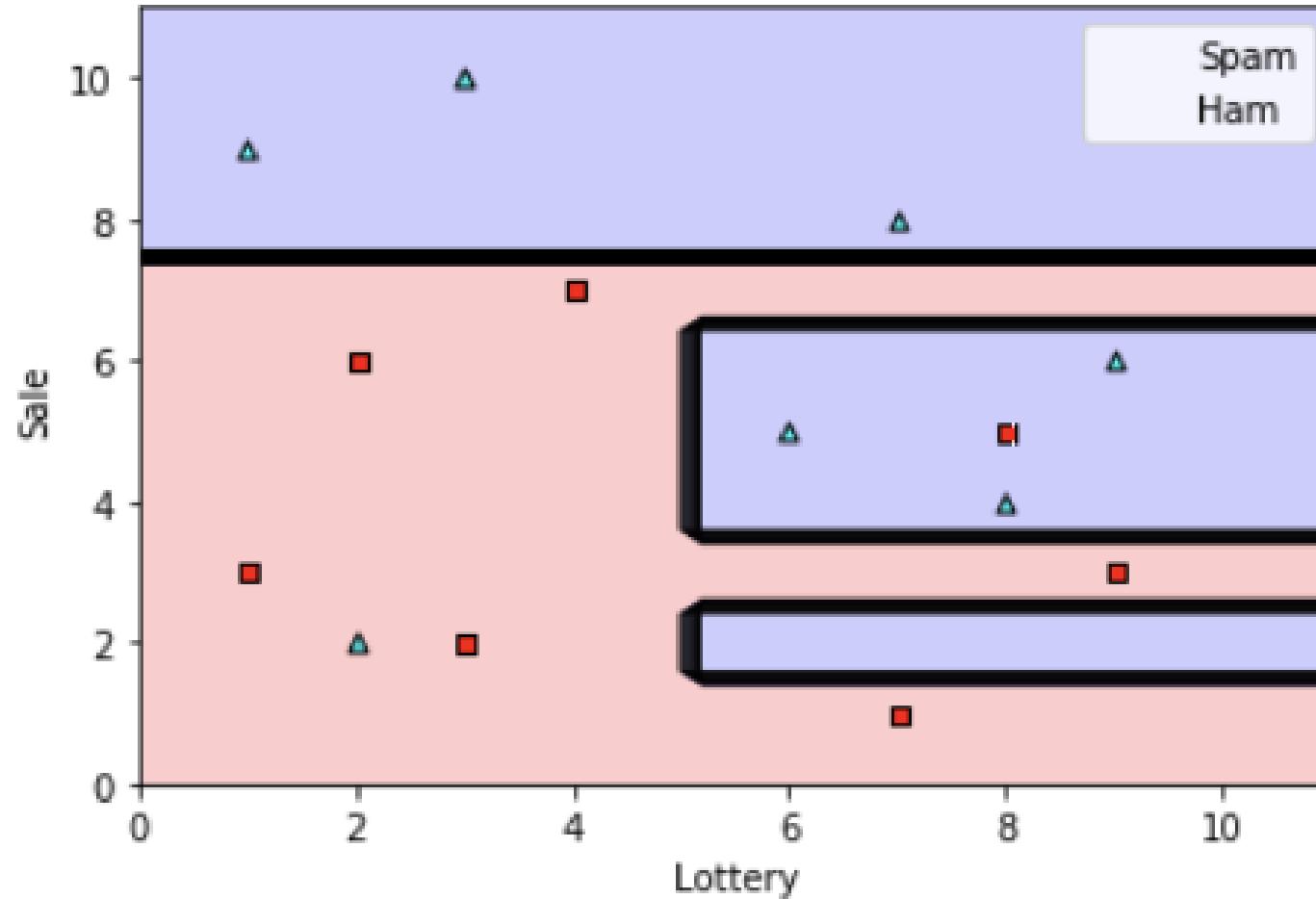
# A detailed (mathematical) picture of AdaBoost



# Coding AdaBoost in Sklearn

```
from sklearn.ensemble import AdaBoostClassifier
adaboost_model = AdaBoostClassifier(random_state=0, n_estimators=6)
adaboost_model.fit(new_X, new_y)
```

# Coding AdaBoost in Sklearn

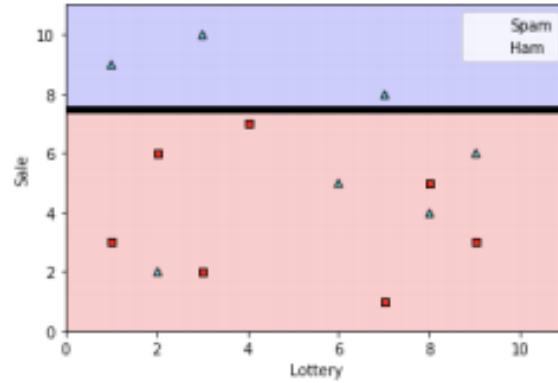


# Coding AdaBoost in Sklearn

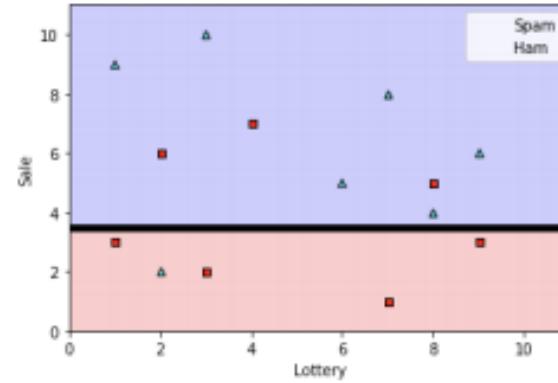
```
estimators = adaboost_model.estimators_
for estimator in estimators:
 plot_model(new_X, new_y, estimator)
plt.show()
```

# Coding AdaBoost in Sklearn

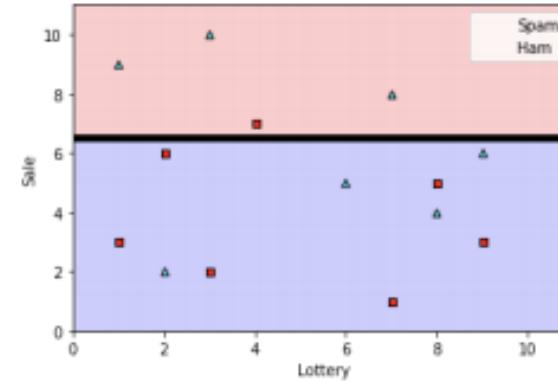
Learner 1



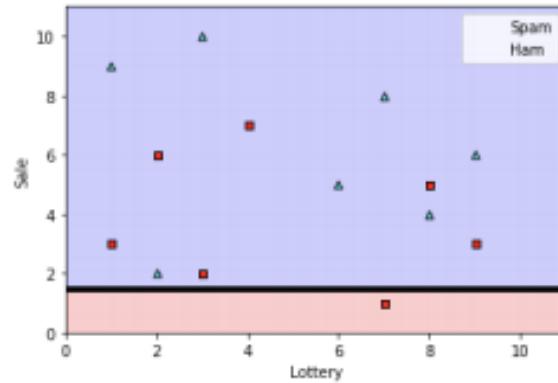
Learner 2



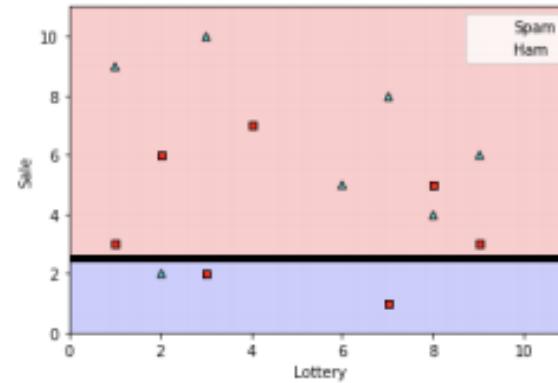
Learner 3



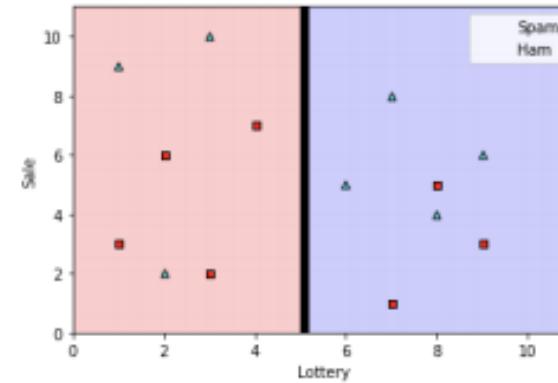
Learner 4



Learner 5



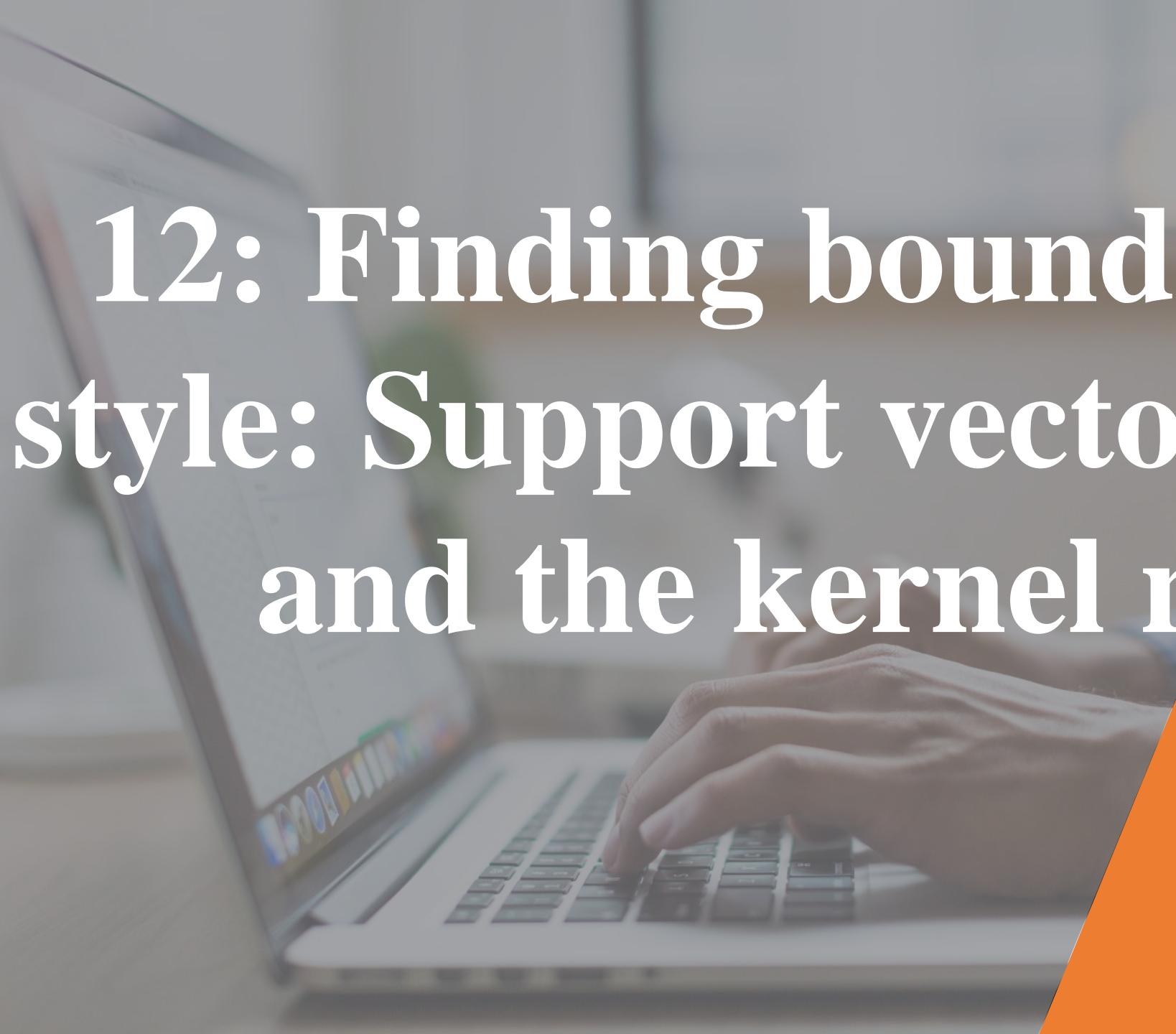
Learner 6



# Summary

- Ensemble methods are ways we use to combine weak learners into a strong one
- There are two major types of ensemble methods: Bagging and boosting.
- Bagging, or bootstrap aggregating, consists of building successive learners on random subsets of our data, and then building a strong classifier based on a majority vote.

# 12: Finding boundaries with style: Support vector machines and the kernel method



# Finding boundaries with style: Support vector machines and the kernel method

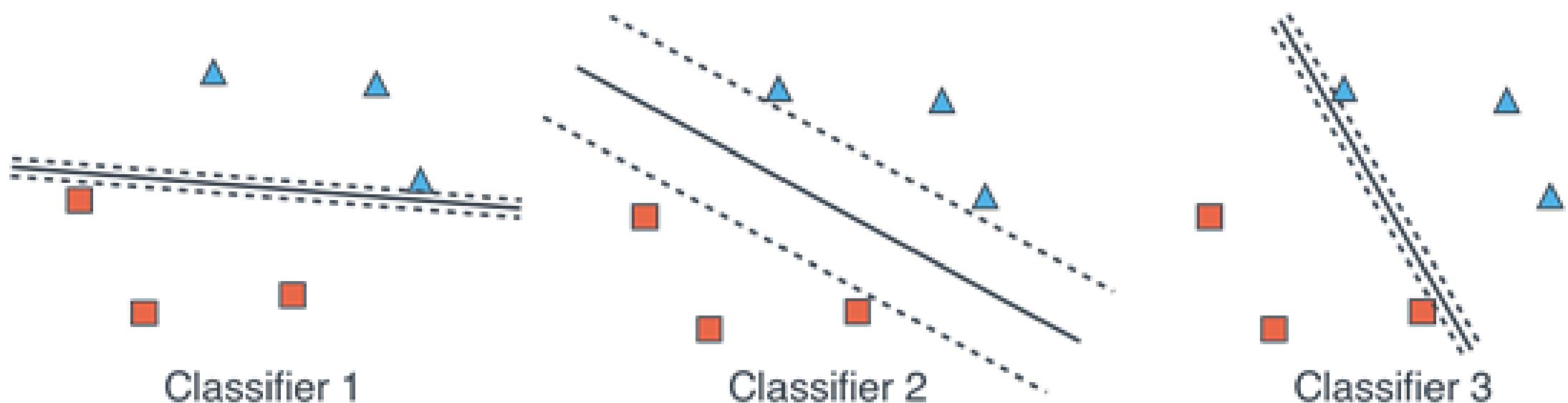
This lesson covers:

- What is a support vector machine?
- What does it mean for a linear classifier to fit well between the points?
- A new linear classifier which consists of two lines, and its new error function.
- Tradeoff between good classification and a good fit: the C parameter.
- Using the kernel method to build non-linear classifiers
- Types of kernels: polynomial and radial basis function (rbf) kernel.
- Coding SVMs and the kernel method in sklearn.



# Finding boundaries with style: Support vector machines and the kernel method

Figure . We draw our classifier as two parallel lines, as far apart from each other as possible. We can see that Classifier 2 is the one where the parallel lines are farther from each other. This means that the main line in Classifier 2 is the one best located between the points.



# A new error function

The error function should punish any model that doesn't achieve those things. Here, we want to achieve two things:

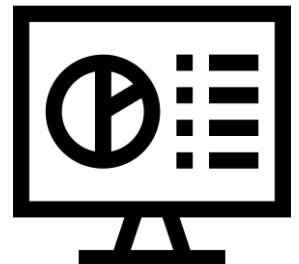
- We want the two lines to classify the points as best as possible.
- We want the two lines to be as far away from each other as possible.

# Classification error function

- We will use the central line as a frame of reference L with equation  $w_1x_1 + w_2x_2 + b = 0$ , and construct two lines, one above it and one below it, with the respective equations

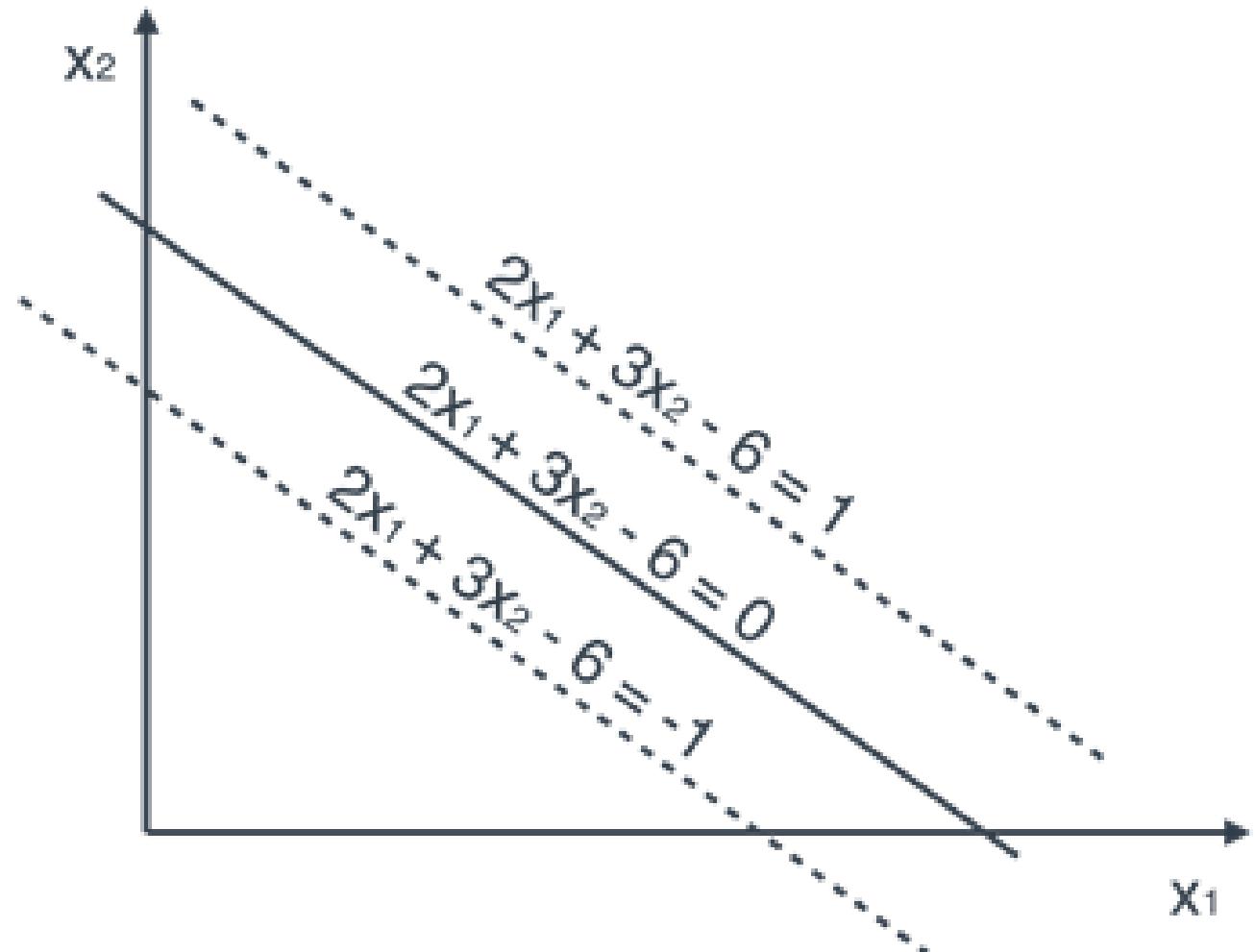
**L+:  $w_1x_1 + w_2x_2 + b = 1$ , and**

**L-:  $w_1x_1 + w_2x_2 + b = -1$ .**



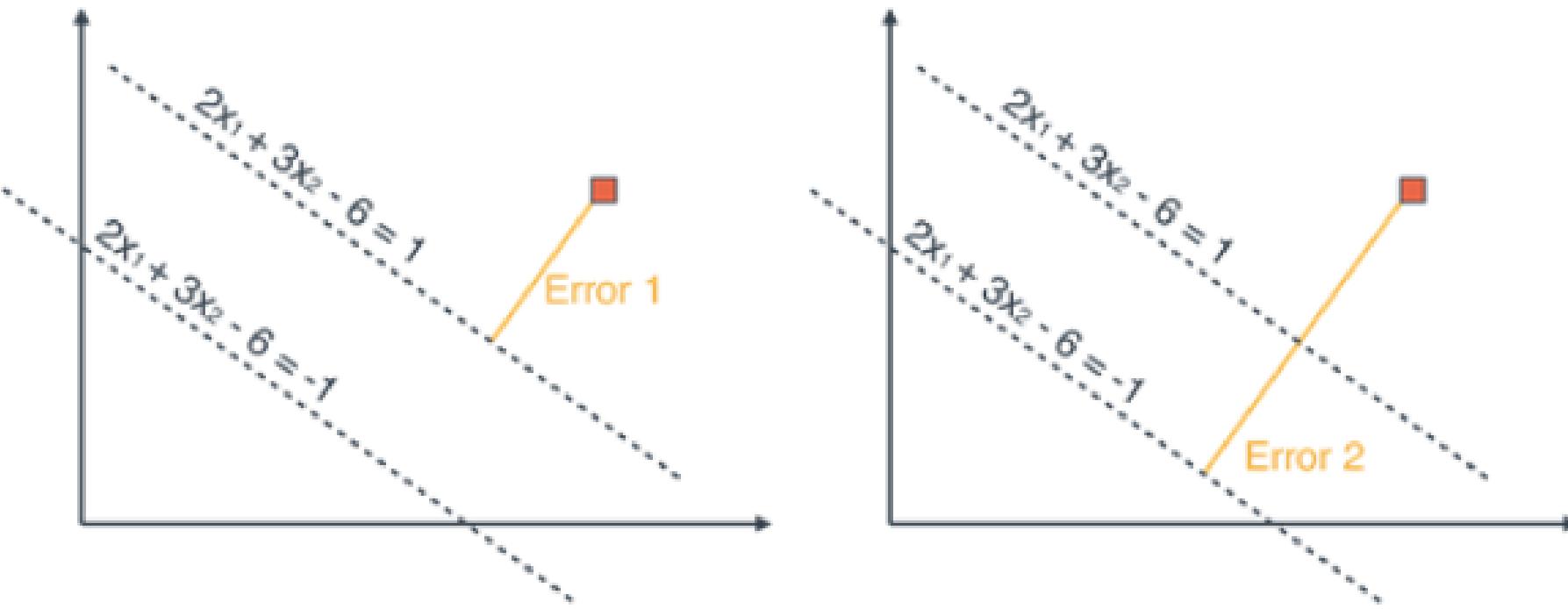
# Classification error function

- Our main line L is the one in the middle.
- We build the two parallel lines L+ and L- by slightly changing the equation of L.



# Classification error function

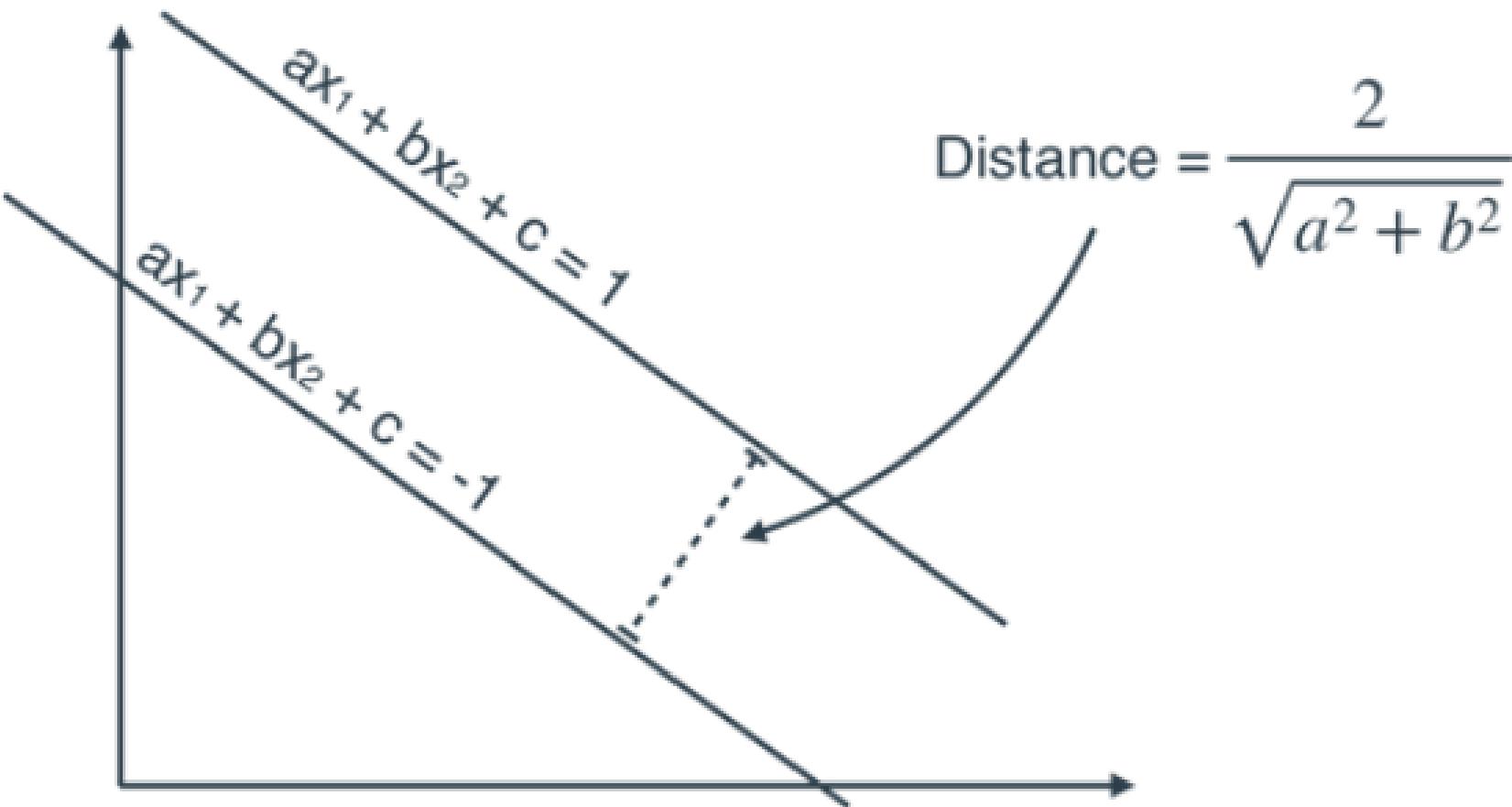
Figure . Now that our classifier consists of two lines, the error of a misclassified point is measured with respect to both lines. We then add the two errors to obtain the classification error.



$$\text{Error} = \text{Error 1} + \text{Error 2}$$

# Distance error function

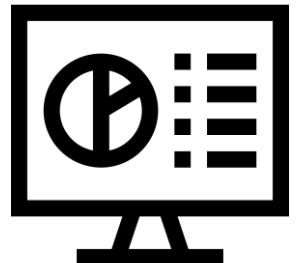
Figure The distance between the two parallel lines can be calculated based on the equations of the lines.



# Distance error function

Knowing previous figure, notice that

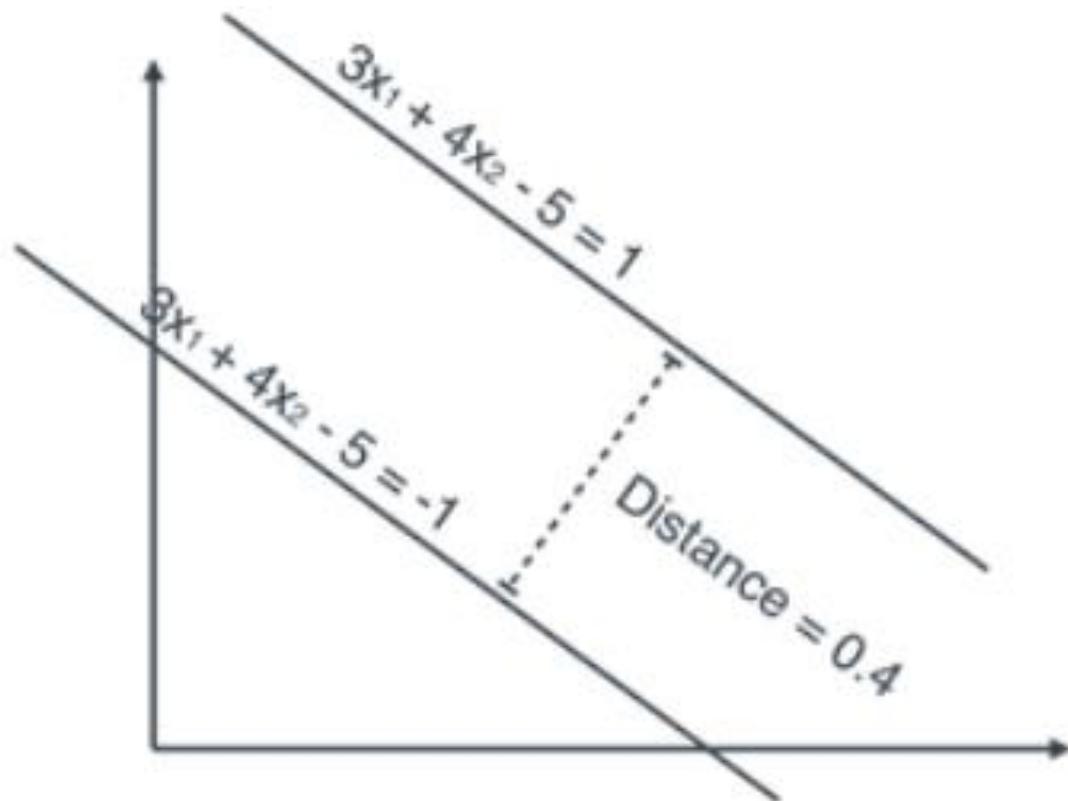
- When  $a^2 + b^2$  is large,  $\frac{2}{\sqrt{a^2 + b^2}}$  is small.
- When  $a^2 + b^2$  is small,  $\frac{2}{\sqrt{a^2 + b^2}}$  is large



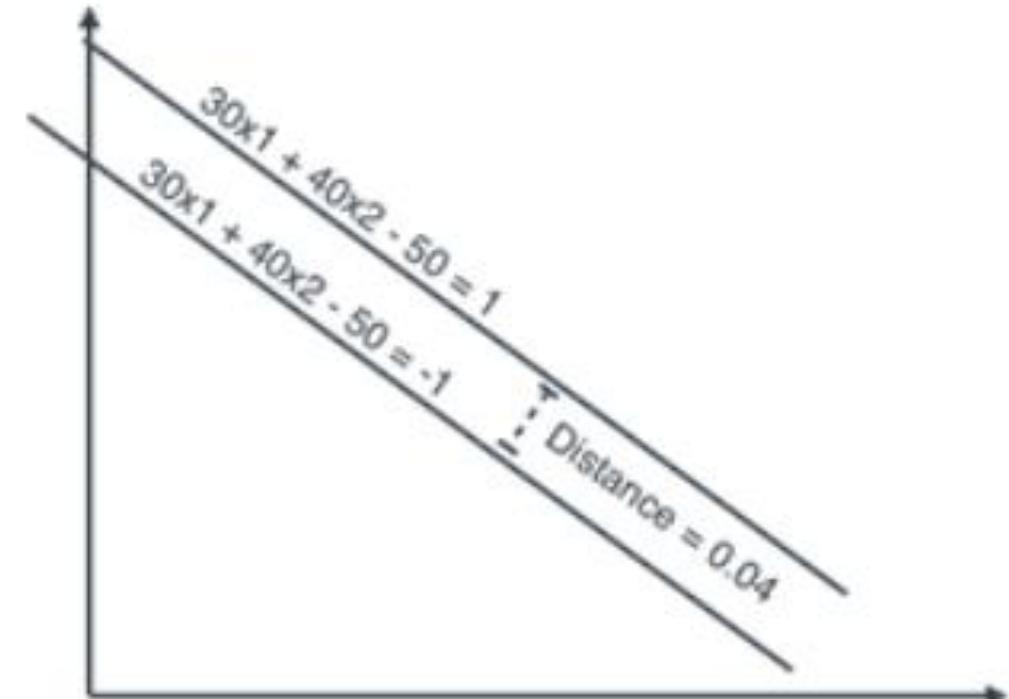
# Distance error function

- Notice also from next figure, that the lines are much closer in classifier 2 than in classifier 1, which makes classifier 1 a much better classifier (from the distance perspective).
- The distance between the lines in classifier 1 is  $\frac{2}{\sqrt{3^2 + 4^2}} = 0.4$   
whereas in classifier 2 it is  $\frac{2}{\sqrt{30^2 + 40^2}} = 0.04$

# Distance error function

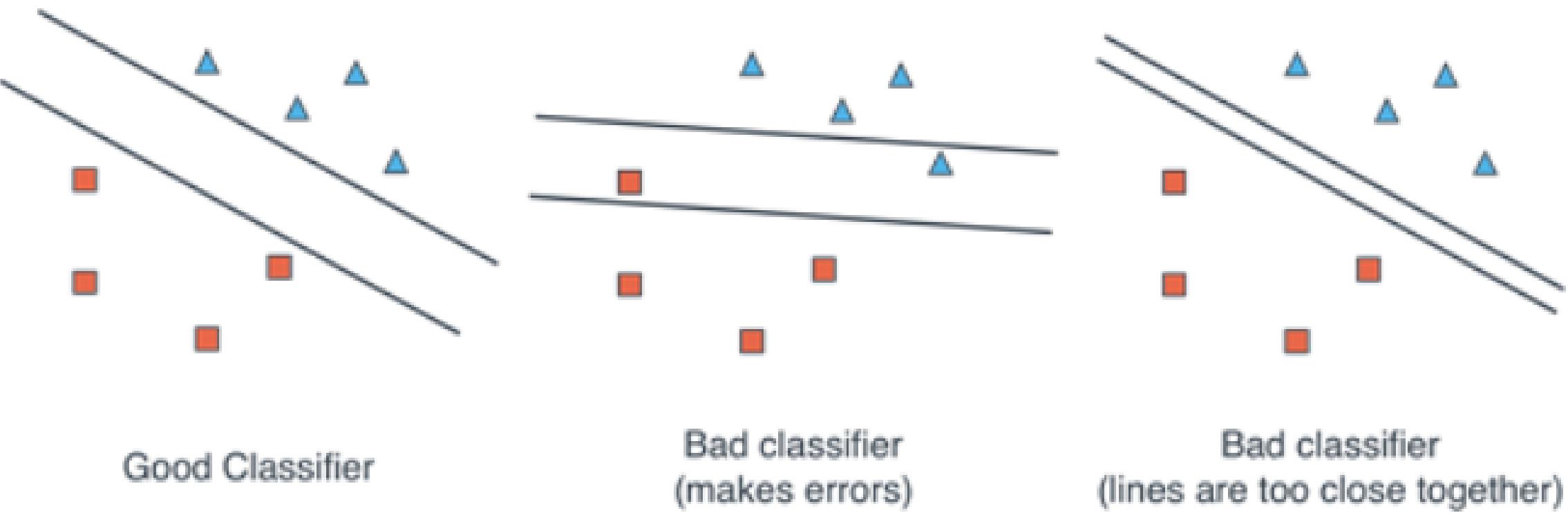


Error function = 25  
Good classifier



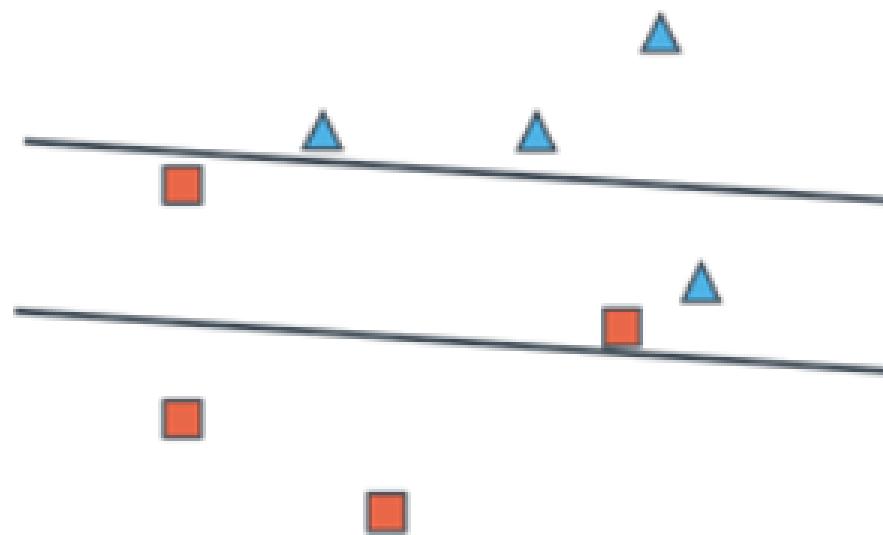
Error function = 2500  
Bad classifier

# Adding them to obtain the error function

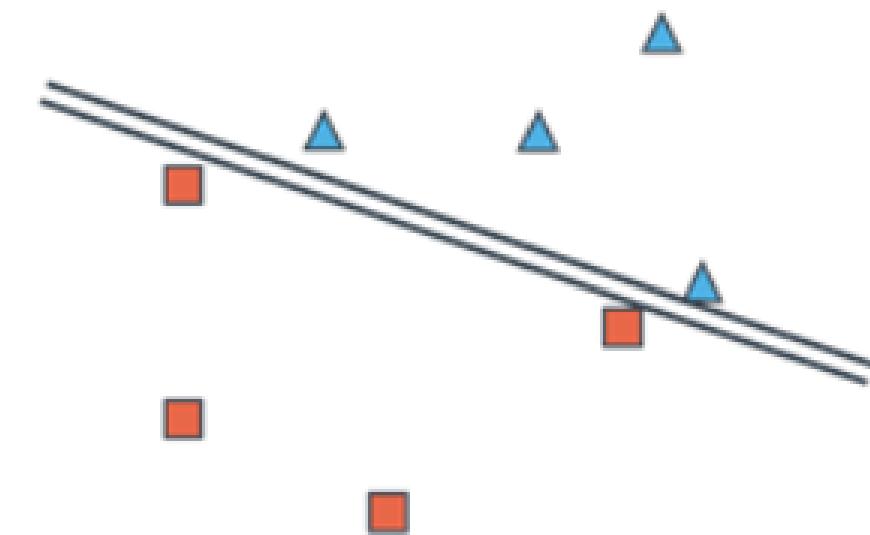


# Using a dial to decide how we want our model: The C parameter

Figure . Both of these classifiers have one pro and one con. The one in the left has the lines well spaced (pro), but it misclassifies some points (con). The one in the right has the lines too close together (con), but it classifies all the points correctly (pro).

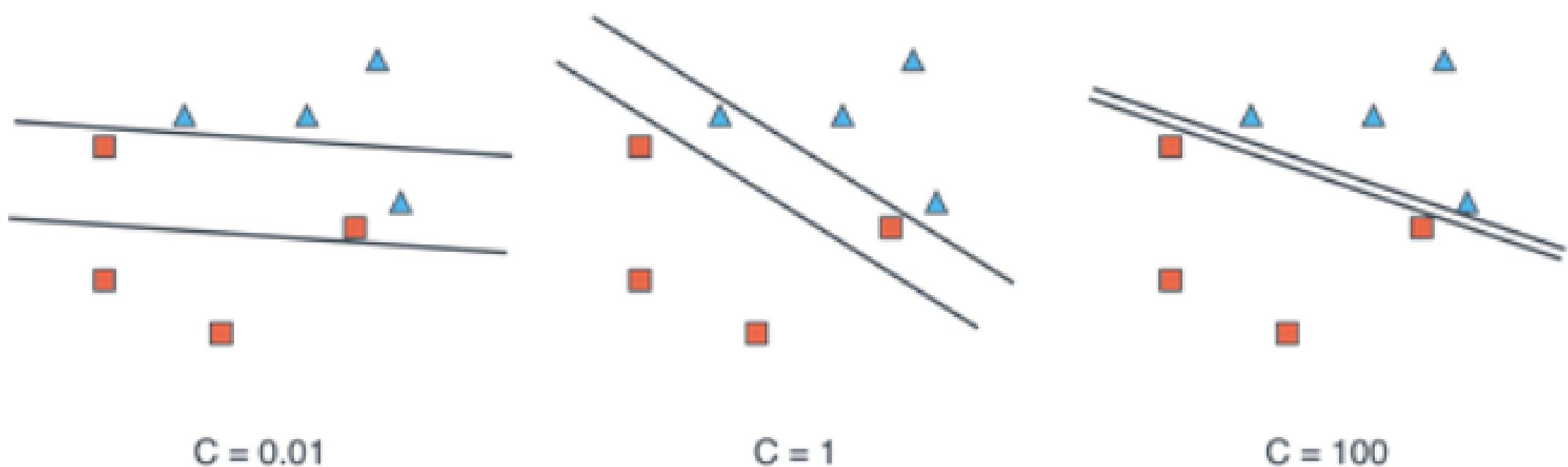


Lines are far apart  
Makes errors



Lines are too close  
Doesn't make errors

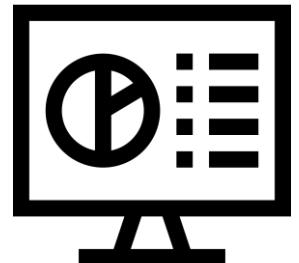
Figure . Different values of C toggle between a classifier with well spaced lines, and one that classifies points correctly. The classifier on the left has a small value of C (0.01), and the lines are well spaced, but it makes mistakes. The classifier on the right has a large value of C (100), and it classifies points correctly, but the lines are too close together. The classifier in the middle has a medium value of C (1), is a good middle ground between the other two.



# Coding a simple SVM

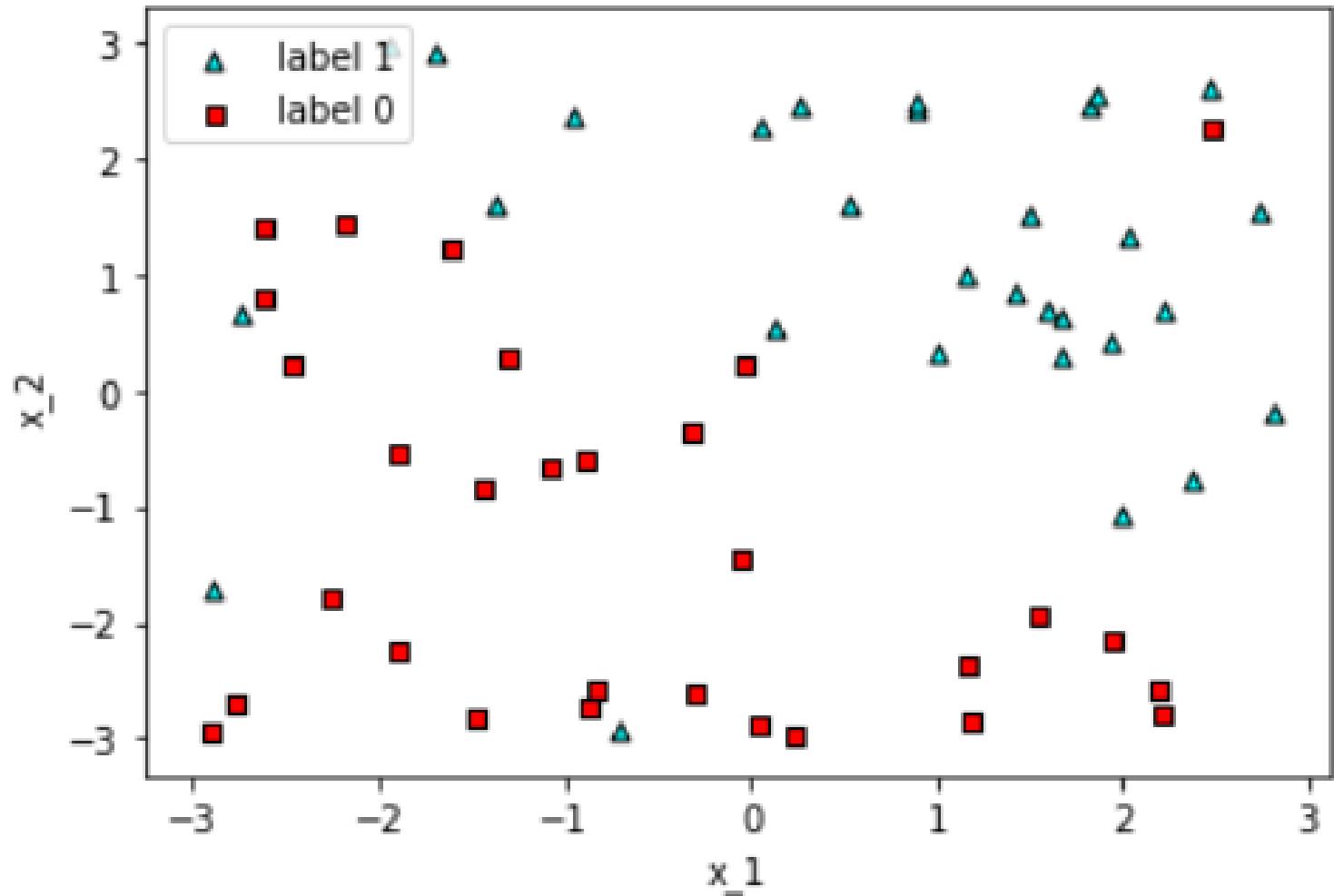
- We use the dataset called ‘linear.csv’, which we load and plot (Next Figure) as follows:

```
df = pd.read_csv('linear.csv')
X = np.array(df[['x_1', 'x_2']])
y = np.array(df['y']).astype(int)
plot_points(X,y)
```



# Coding a simple SVM

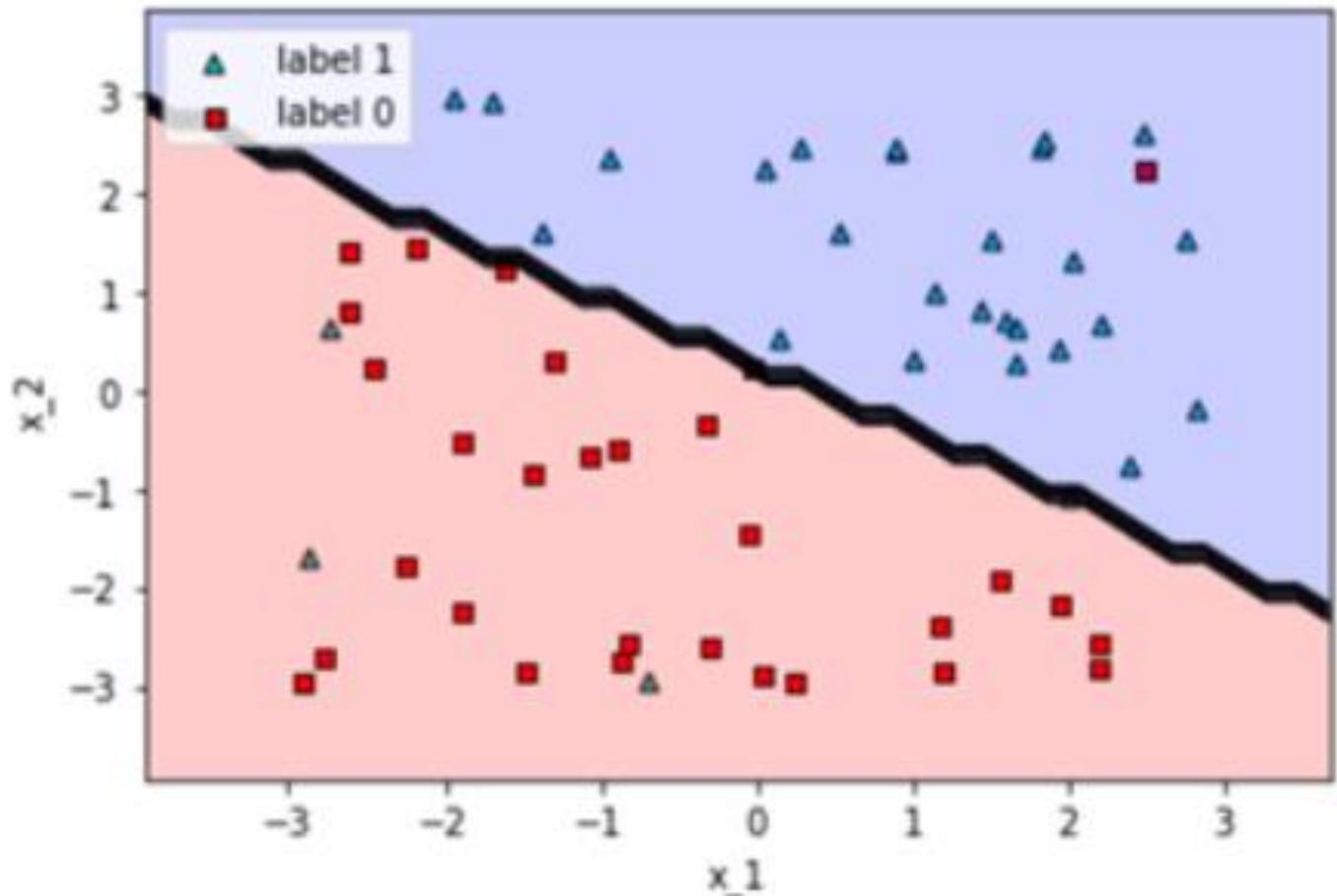
An linearly separable dataset, with noise.



# Coding a simple SVM

The SVM classifier we've built in sklearn consists of a line.

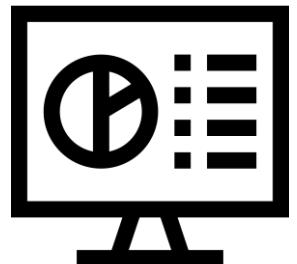
Accuracy: 0.9333333333333333



# Introducing the C parameter

```
C = 0.01
svm_c_001 = SVC(kernel='linear', C=0.01)
svm_c_001.fit(X,y)
print("C = 0.1")
print("Accuracy:", svm_c_001.score(X, y))
plot_model(X,y,svm_c_001)

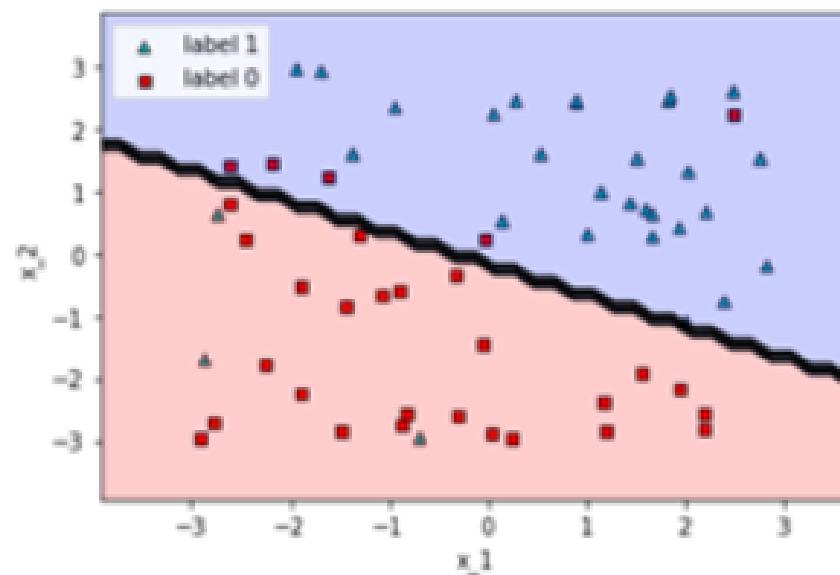
C = 100
svm_c_100 = SVC(kernel='linear', C=100)
svm_c_100.fit(X,y)
print("C = 100")
print("Accuracy:", svm_c_100.score(X, y))
plot_model(X,y,svm_c_100)
```



# Introducing the C parameter

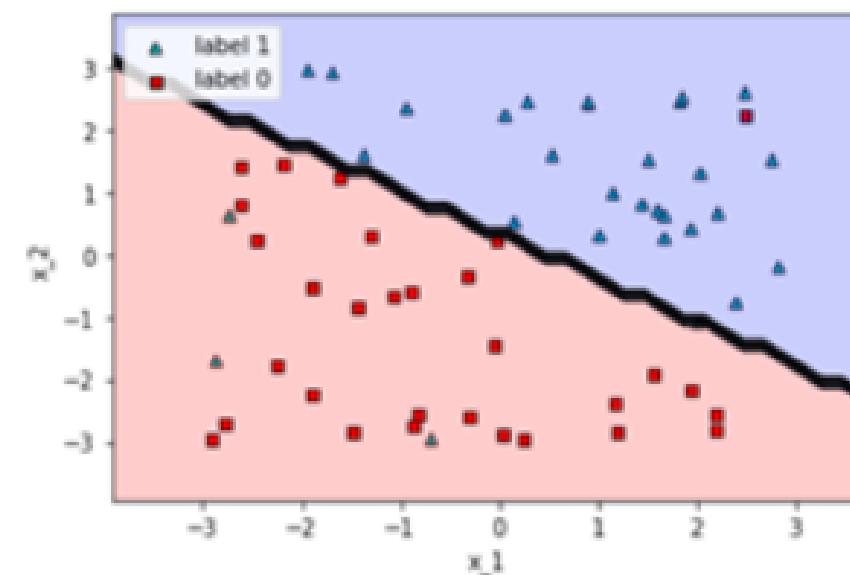
Figure The classifier on the left has a small value of C, and it spaced the line well between the points, but it makes some mistakes. The classifier on the right has a large value of C, and it makes no mistakes, although the line passes too close to some of the points.

Accuracy: 0.8666666666666667



C = 0.01

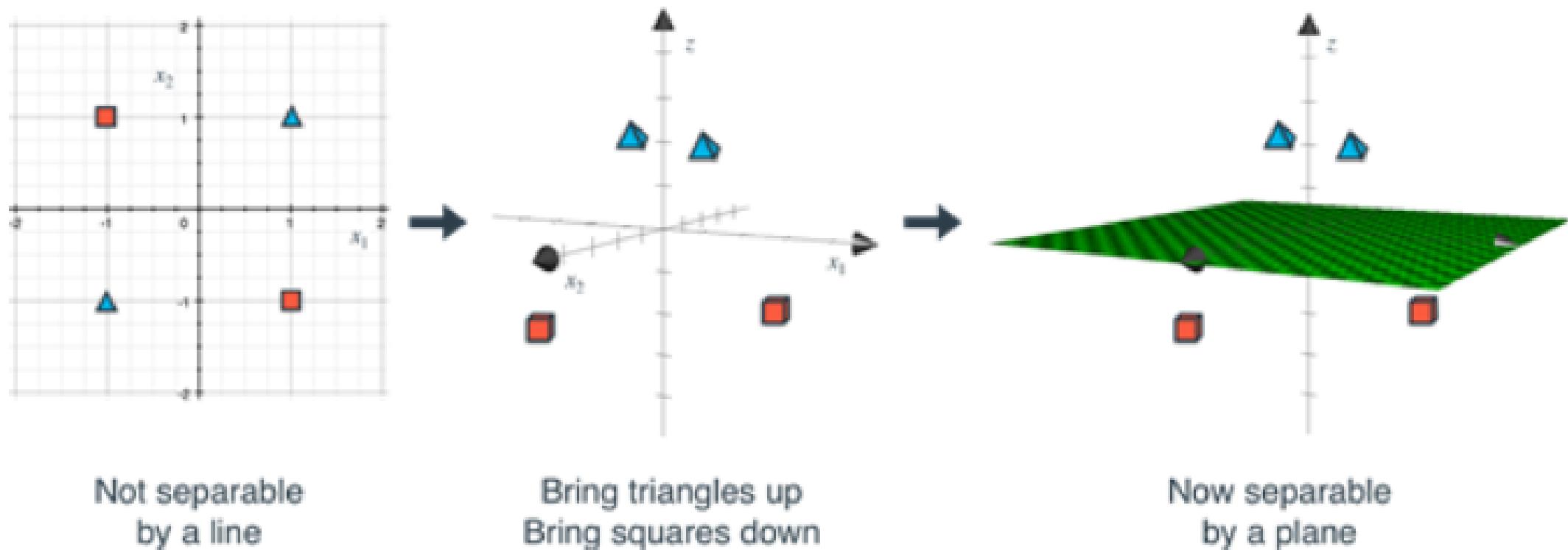
Accuracy: 0.9166666666666666



C = 100

# The kernel method

Figure Left: The set is not separable by a line. Middle: We look at it in three dimensions, and proceed to raise the two triangles and lower the two squares. Right: Our new dataset is now separable by a plane.



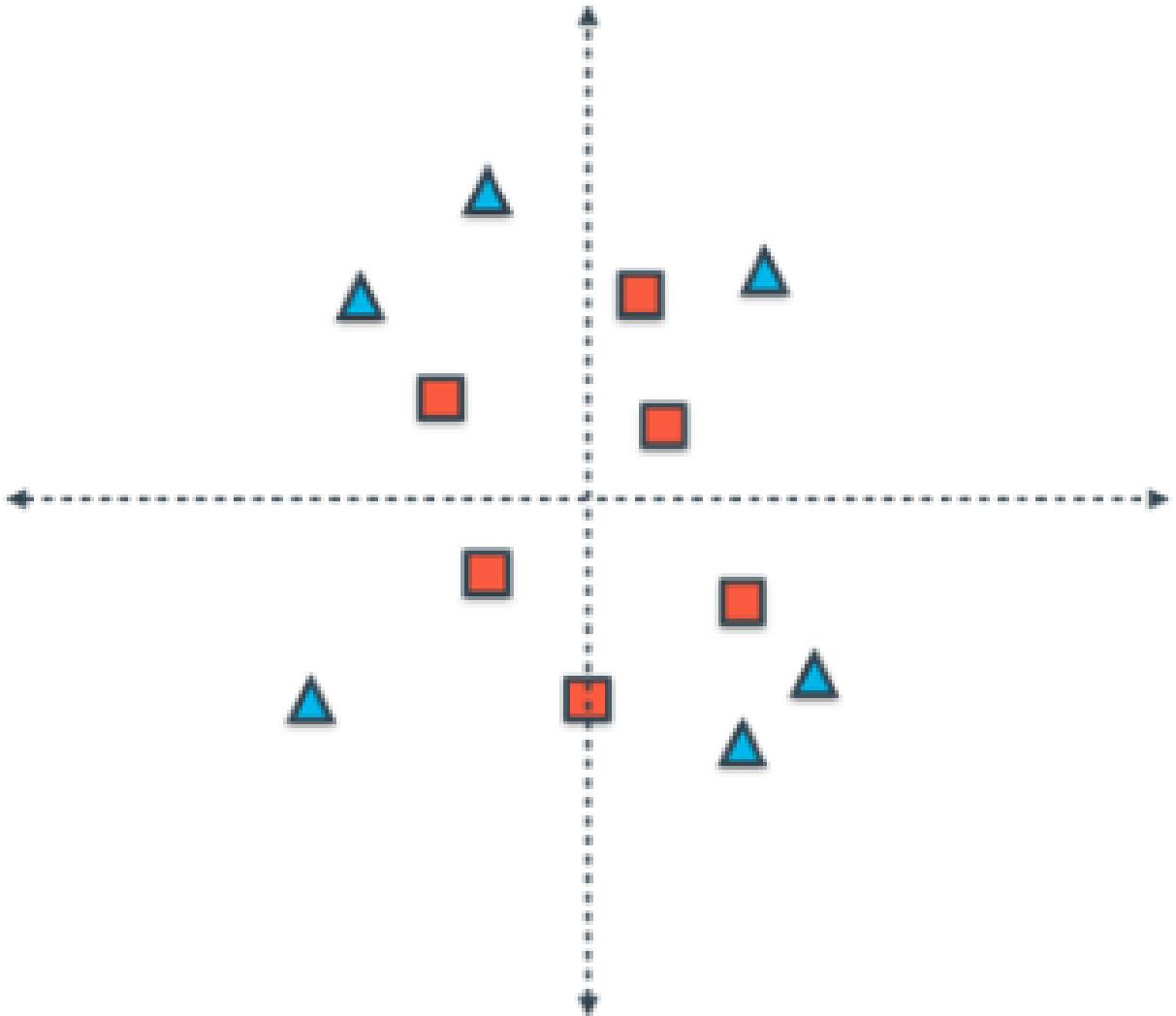
# Example 1: A circular data set

| X <sub>1</sub> | X <sub>2</sub> | y |
|----------------|----------------|---|
| 0.3            | 0.3            | 0 |
| 0.2            | 0.8            | 0 |
| -0.6           | 0.4            | 0 |
| 0.6            | -0.4           | 0 |
| -0.4           | -0.3           | 0 |
| 0              | -0.8           | 0 |

|      |      |   |
|------|------|---|
| -0.4 | 1.2  | 1 |
| 0.9  | -0.7 | 1 |
| -1.1 | -0.8 | 1 |
| 0.7  | 0.9  | 1 |
| -0.9 | 0.8  | 1 |
| 0.6  | -1   | 1 |

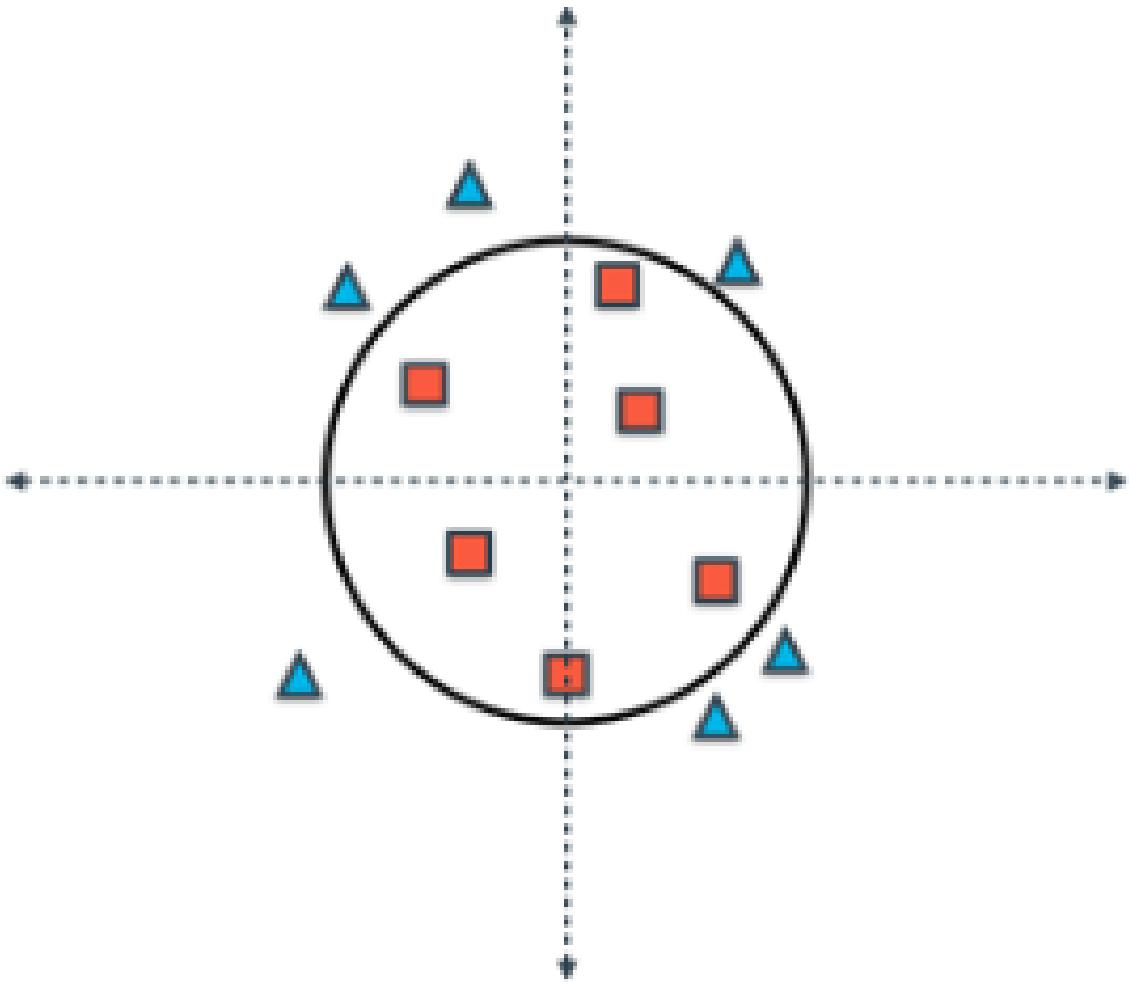
# The kernel method

- Plot of the dataset in Previous Table.
- Note that it is not separable by a line.



# The kernel method

The kernel method gives us a classifier with a circular boundary, which separates these points well..

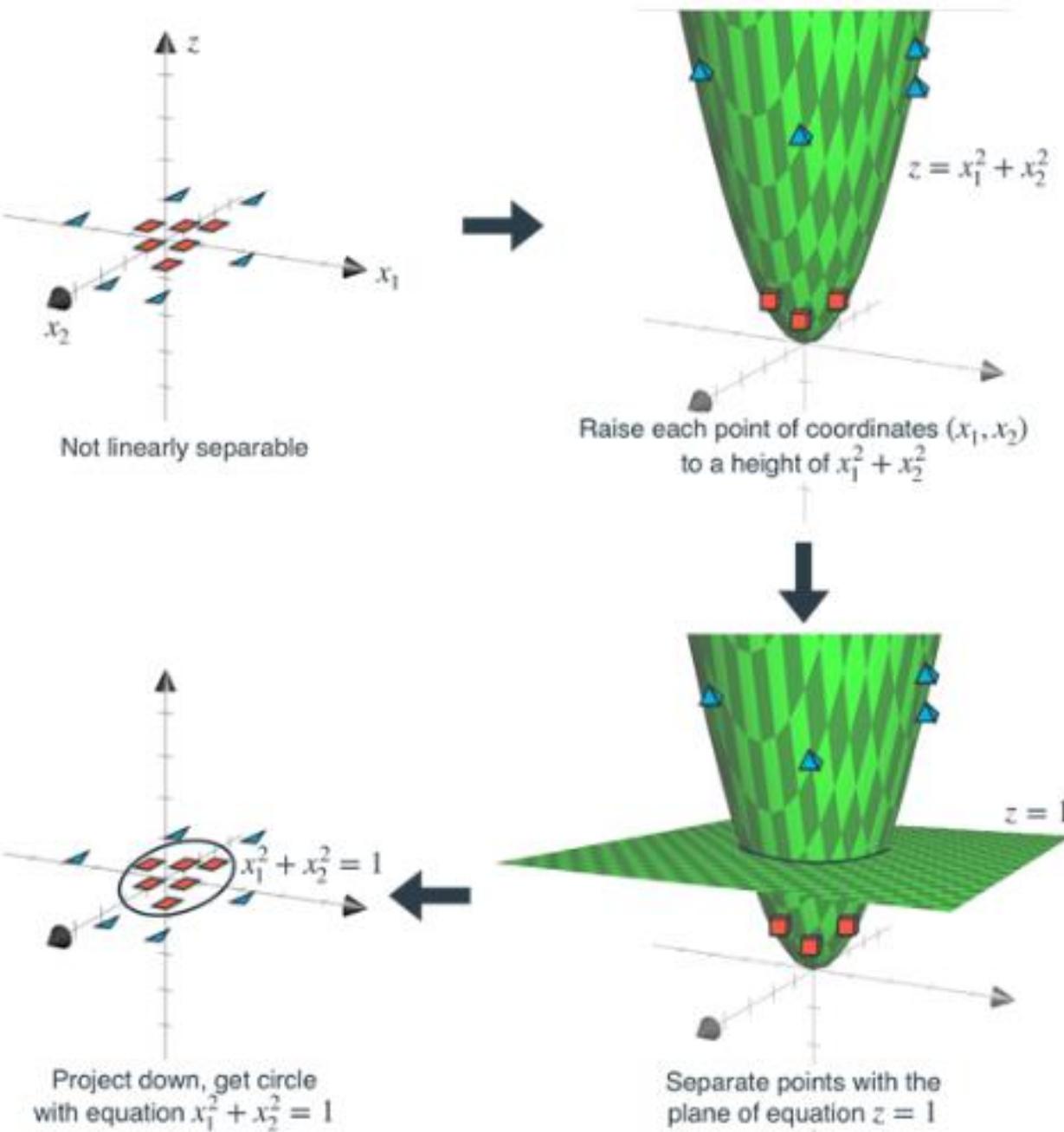


# The kernel method

| $x_1$ | $x_2$ | $x_1^2 + x_2^2$ | $y$ |
|-------|-------|-----------------|-----|
| 0.3   | 0.3   | 0.18            | 0   |
| 0.2   | 0.8   | 0.68            | 0   |
| -0.6  | 0.4   | 0.52            | 0   |
| 0.6   | -0.4  | 0.52            | 0   |
| -0.4  | -0.3  | 0.25            | 0   |

|      |      |      |   |
|------|------|------|---|
| 0    | -0.8 | 0.64 | 0 |
| -0.4 | 1.2  | 1.6  | 1 |
| 0.9  | -0.7 | 1.3  | 1 |
| -1.1 | -0.8 | 1.85 | 1 |
| 0.7  | 0.9  | 1.3  | 1 |
| -0.9 | 0.8  | 1.45 | 1 |
| 0.6  | -1   | 1.36 | 1 |

Step 1: We start a dataset  
that is not linearly  
separable



## Example 2: The AND operator on a recommendation system

- When a customer leaves the page, they are asked to fill in a feedback through a form (and somehow, all of the customers filled it in).
- The two questions in the feedback form are:

Did you like the product?

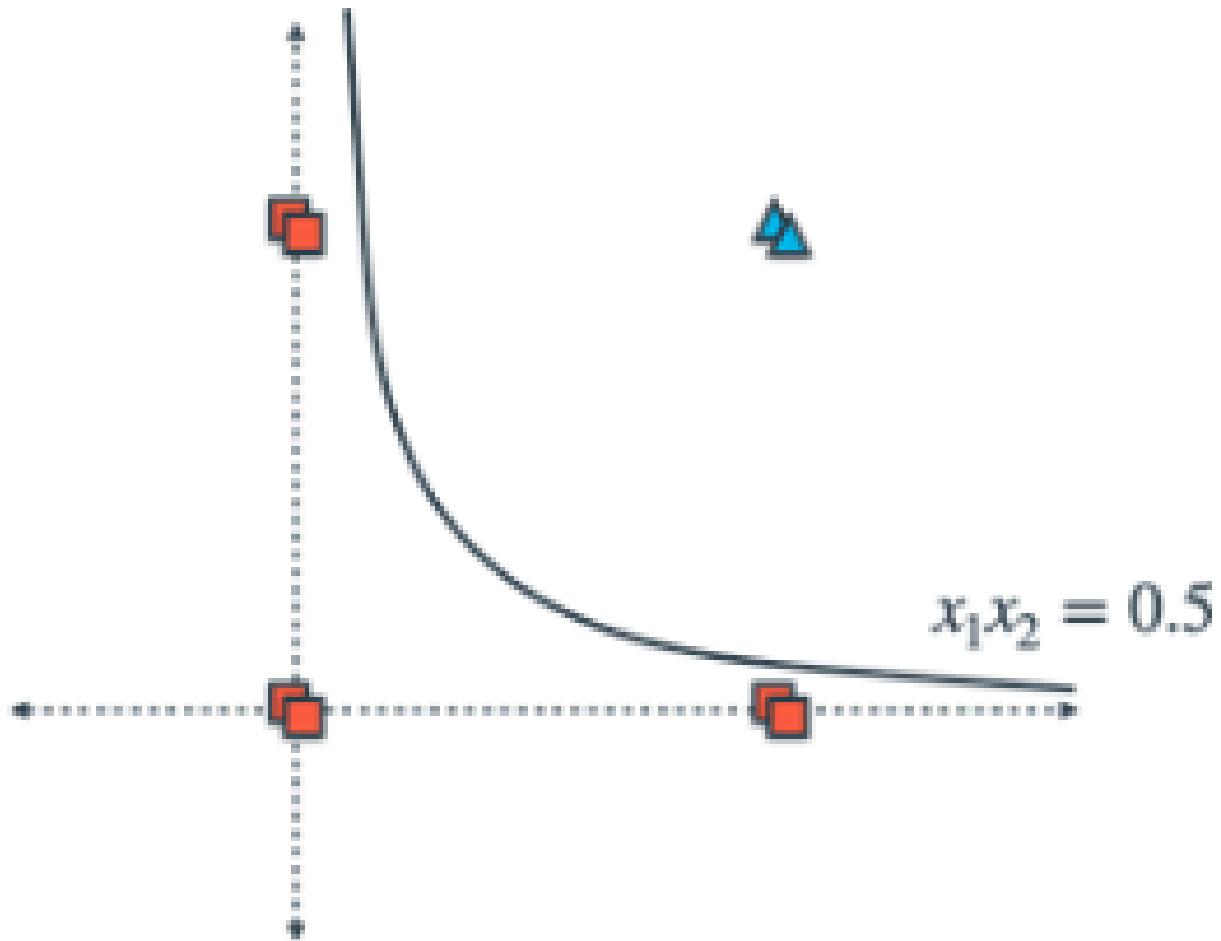
Did you like the price of the product?

- A table of customers.
- For each customer, we record if they liked the product, if they liked the price, and if they bought the product.

| $x_1$ (Liked the product) | $x_2$ (Liked the price) | $y$ (Bought the product) |
|---------------------------|-------------------------|--------------------------|
| 0                         | 0                       | 0                        |
| 0                         | 0                       | 0                        |
| 0                         | 1                       | 0                        |
| 0                         | 1                       | 0                        |
| 1                         | 0                       | 0                        |
| 1                         | 0                       | 0                        |
| 1                         | 1                       | 1                        |
| 1                         | 1                       | 1                        |

| $x_1$ (Liked the product) | $x_2$ (Liked the price) | $x_1x_2$ (Liked the price and the product) | $y$ (Bought the product) |
|---------------------------|-------------------------|--------------------------------------------|--------------------------|
| 0                         | 0                       | 0                                          | 0                        |
| 0                         | 0                       | 0                                          | 0                        |
| 0                         | 1                       | 0                                          | 0                        |
| 0                         | 1                       | 0                                          | 0                        |
| 1                         | 0                       | 0                                          | 0                        |
| 1                         | 0                       | 0                                          | 0                        |
| 1                         | 1                       | 1                                          | 1                        |
| 1                         | 1                       | 1                                          | 1                        |

# The kernel method



| $x_1$ | $x_2$ | $x_3 = x_1^2$ | $x_4 = x_1x_2$ | $x_5 = x_2^2$ | y |
|-------|-------|---------------|----------------|---------------|---|
| 0.3   | 0.3   | 0.09          | 0.09           | 0.09          | 0 |
| 0.2   | 0.8   | 0.04          | 0.16           | 0.64          | 0 |
| -0.6  | 0.4   | 0.36          | -0.24          | 0.16          | 0 |
| 0.6   | -0.4  | 0.36          | -0.24          | 0.16          | 0 |
| -0.4  | -0.3  | 0.16          | 0.12           | 0.09          | 0 |
| 0     | -0.8  | 0             | 0              | 0.64          | 0 |

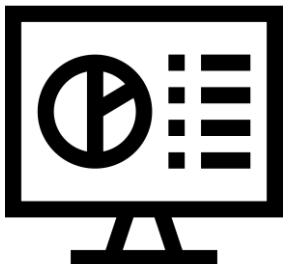
# The kernel method

|      |      |      |       |      |   |
|------|------|------|-------|------|---|
| -0.4 | 1.2  | 0.16 | -0.48 | 1.44 | 1 |
| 0.9  | -0.7 | 0.81 | -0.63 | 0.49 | 1 |
| -1.1 | -0.8 | 1.21 | 0.88  | 0.64 | 1 |
| 0.7  | 0.9  | 0.49 | 0.63  | 0.81 | 1 |
| -0.9 | 0.8  | 0.81 | -0.72 | 0.64 | 1 |
| 0.6  | -1   | 0.36 | -0.6  | 1    | 1 |

# Exercise:

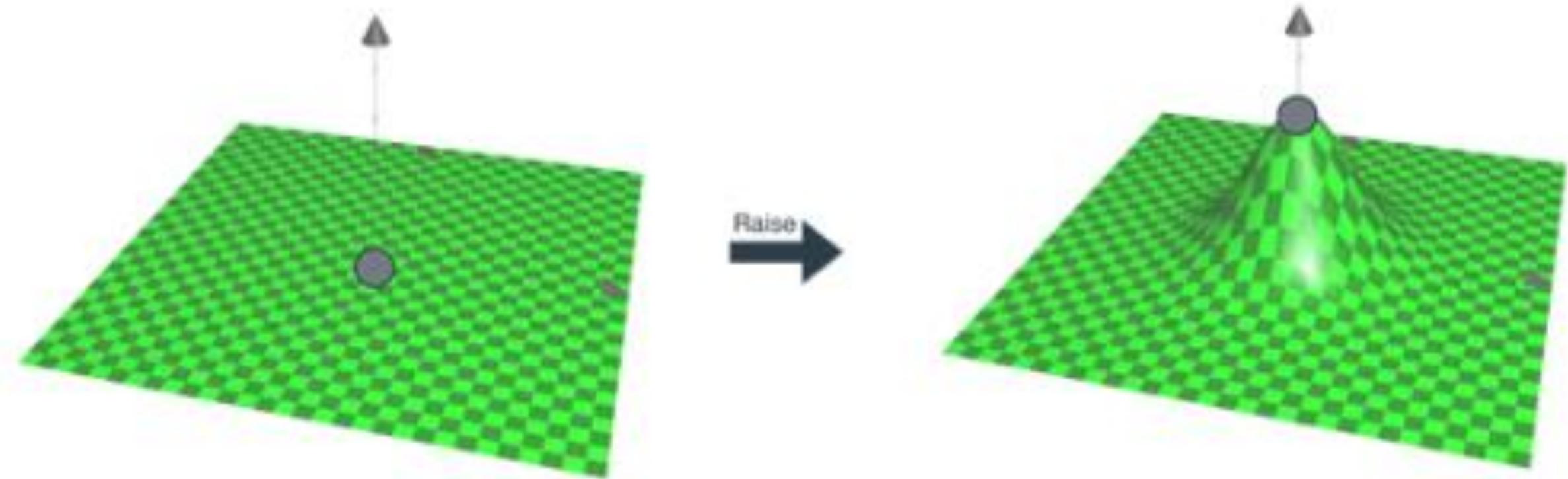


- How many elements does the polynomial kernel of degree 4 have, on the variables  $x_1, x_2$ , and  $x_3$ ?



# The radial basis function (rbf) kernel

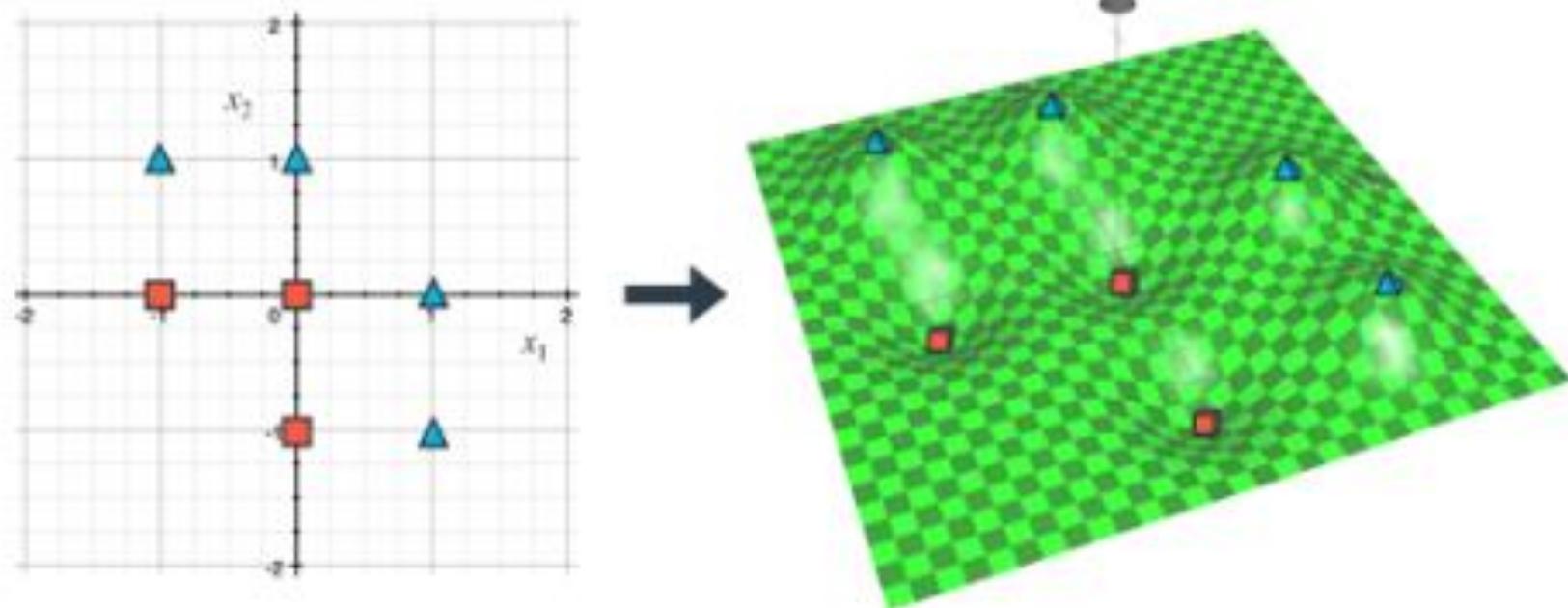
- A radial basis function consists of raising the plane at a particular point.



# The radial basis function (rbf) kernel

Right: We have used the radial basis functions to raise each of the triangles and lower each of the squares.

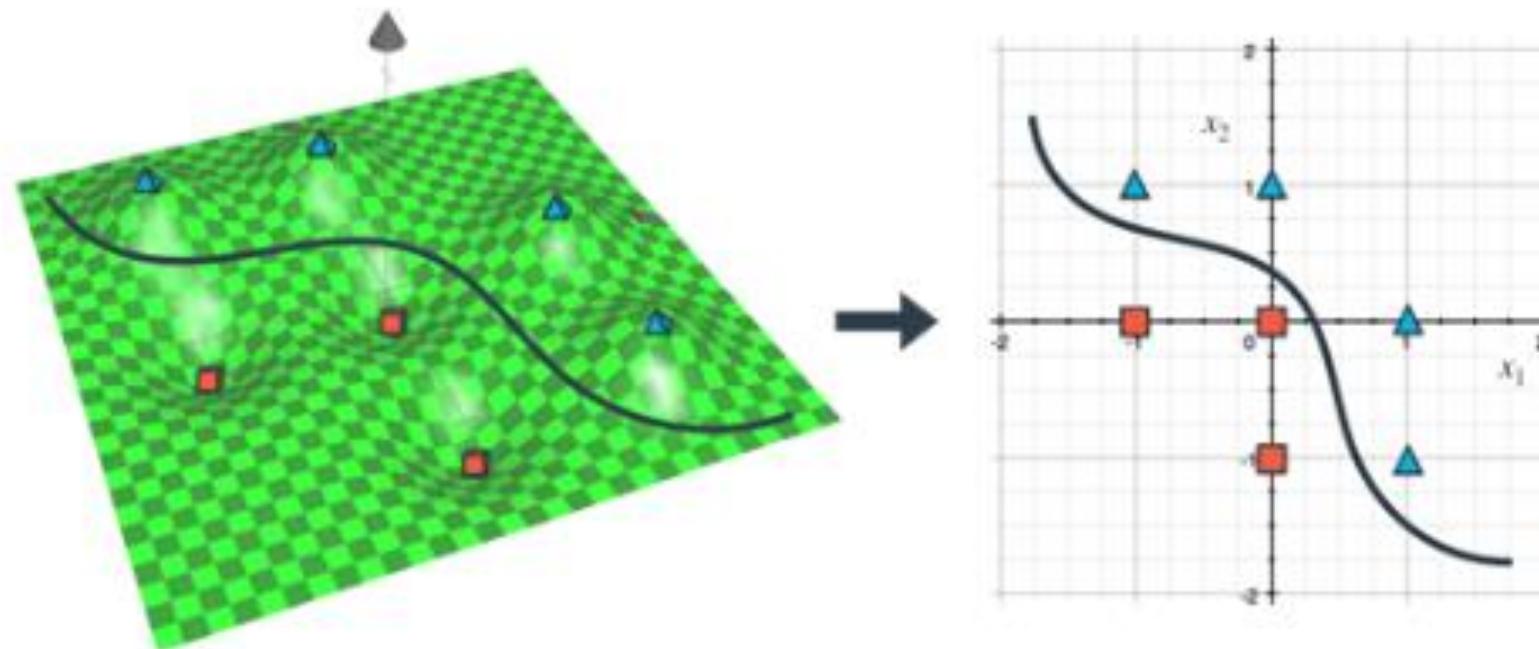
Figure 1. Left: A dataset in the plane.



# The radial basis function (rbf) kernel

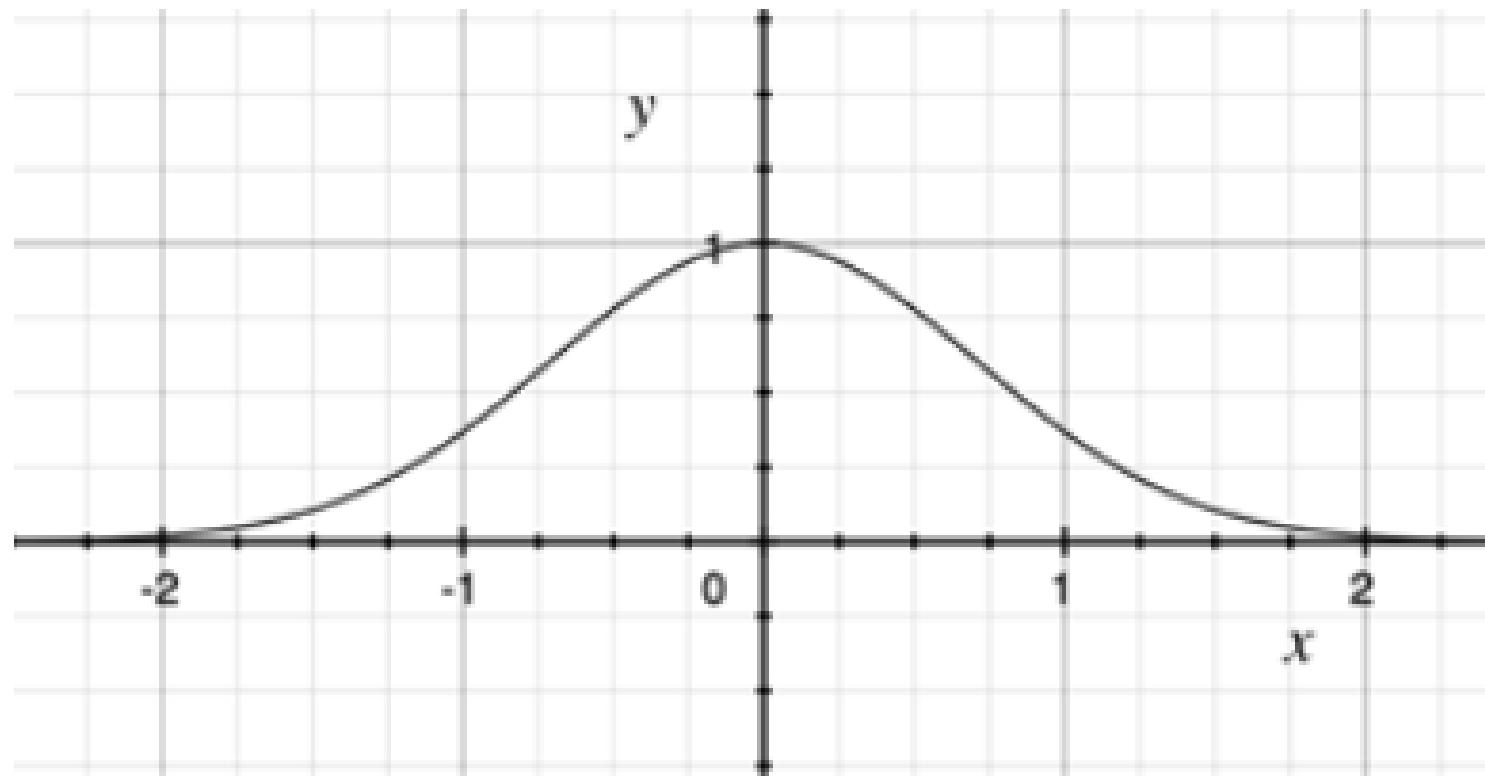
Right: When we project (flatten) the points back to the plane, the coastline is now our classifier that separates the triangles from the squares.

Figure . Left: If we look at the points at height 0, they form a curve. If we think of the high points as land and the low points as the sea, this curve is the coastline.



# Radial basis functions

An example of a radial basis function. It looks a lot like a Gaussian (normal)

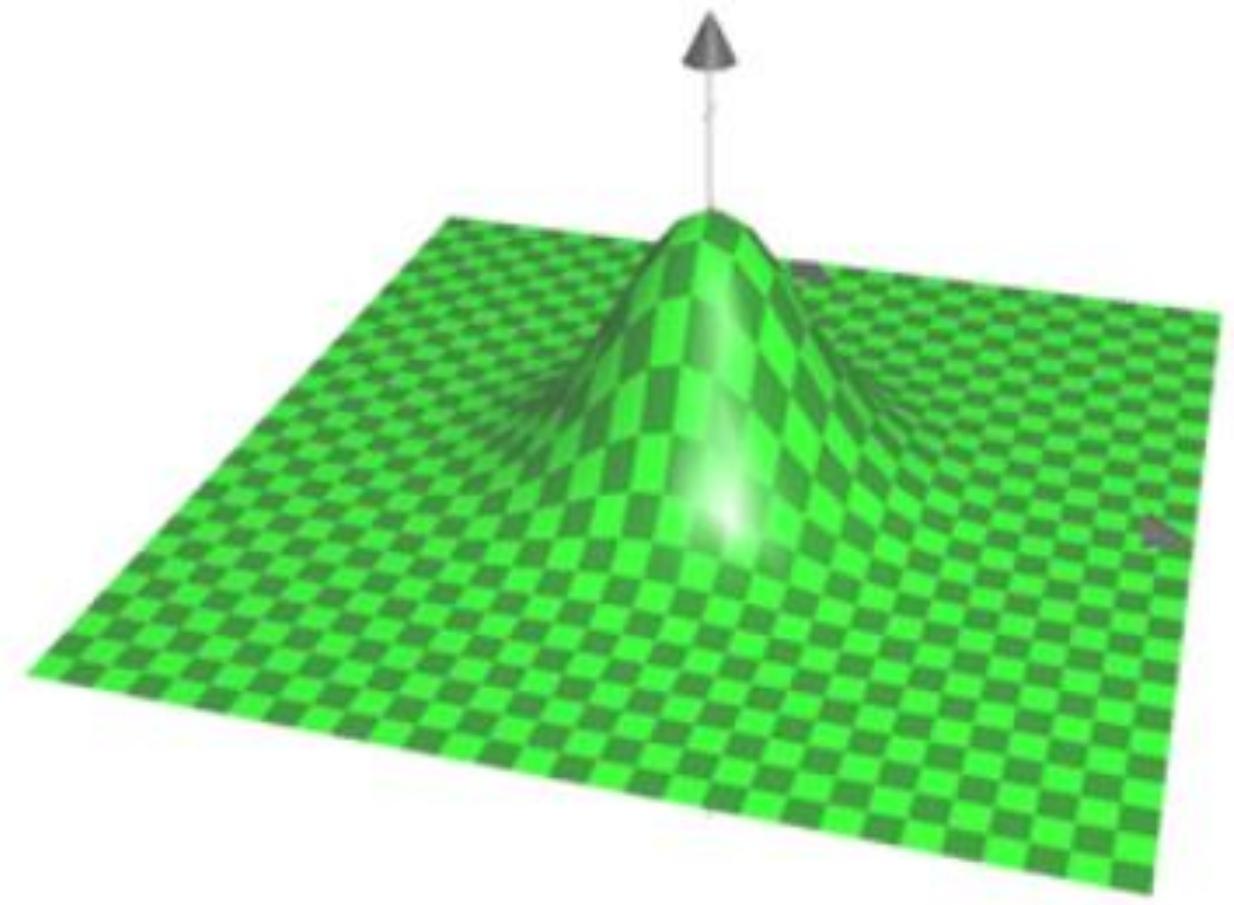


# Radial basis functions

- Notice that this bump happens at 0. If we wanted it to appear at any different point, say  $y = e^{-(x-p)^2}$  simply translate the formula, and get  $y = e^{-(x-5)^2}$ . Thus, if I want to obtain the radial basis function  $y = e^{-(x-5)^2}$  the point 5 is precisely
- For two variables, the formula for the most basic radial basis function is  $z = e^{-(x^2+y^2)}$ , and it looks like the plot in Next Figure

# Radial basis functions

- A radial basis function on two variables.
- It again looks a lot like a Gaussian distribution.



# Radial basis functions

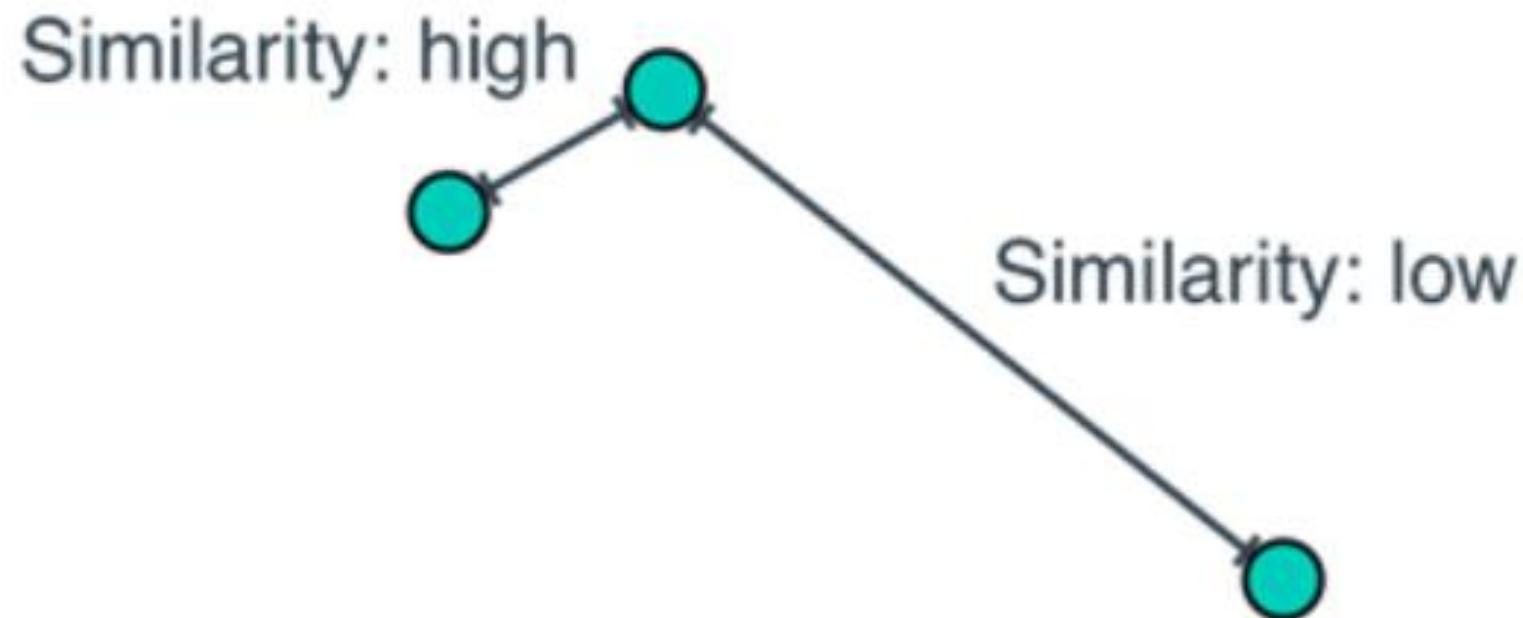
- Again, this bump happens exactly at the point (0,0). If we wanted it to appear at any different point, say (p,q), we simply translate the formula, and get  
 $y = e^{-[(x-p)^2 + (y-q)^2]}$  want to obtain the radial basis function centered at the point (2,-3),  
the formula is precisely  
 $y = e^{-[(x-2)^2 + (y+3)^2]}$

- For n variables, the formula for the basic radial basis function is  
 $y = e^{-(x_1^2 + \dots + x_n^2)}$

# Radial basis functions

## Similarity

Figure Two points that are close by are defined to have high similarity. Two points that are far away are defined to have low similarity.



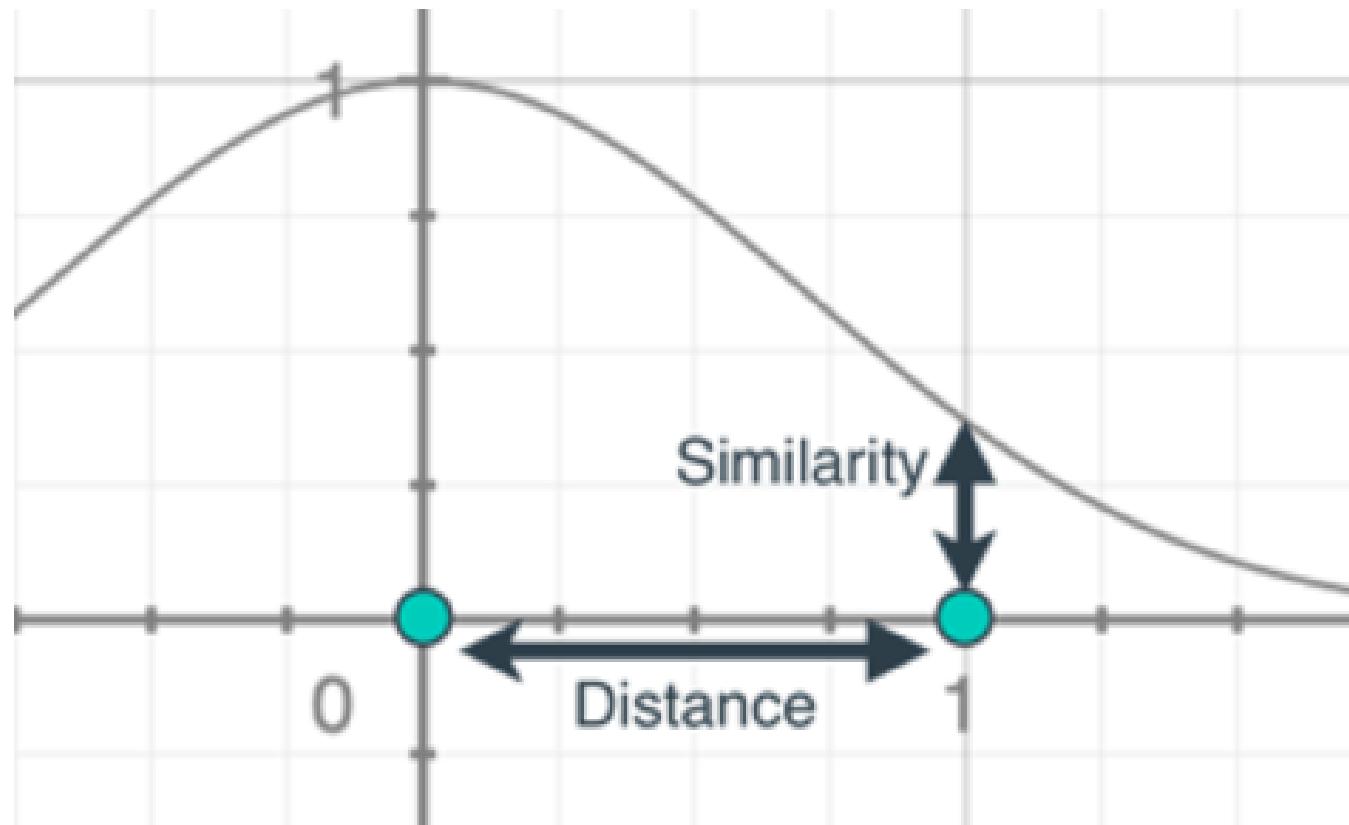
# Radial basis functions

- Now we need to find a formula for similarity.
- As you can see, similarity grows inversely as distance.
- Thus, many formulas for similarity would work, as long as the similarity increases while the distance decreases. Since we are using exponential functions a lot in this section, let's define it as follows.
- For points p and q, the similarity between p and q is:

$$\text{similarity}(p, q) = e^{-\text{distance}(p,q)^2}.$$

# Radial basis functions

Figure . The similarity is defined as the height of a point in the radial basis function, where the input is the distance. Note that the higher the distance, the lower the similarity, and vice versa.



## Training an SVM with the rbf kernel

| Point | X <sub>1</sub> | X <sub>2</sub> | y |
|-------|----------------|----------------|---|
| 1     | 0              | 0              | 0 |
| 2     | -1             | 0              | 0 |
| 3     | 0              | -1             | 0 |
| 4     | 0              | 1              | 1 |
| 5     | 1              | 0              | 1 |
| 6     | -1             | 1              | 1 |
| 7     | 1              | -1             | 1 |

# Radial basis functions

- The distance between point 1 and point 6, by the pythagorean theorem is:

$$\text{distance}(\text{point 1}, \text{point 6}) = \sqrt{(0+1)^2 + (0-1)^2} = \sqrt{2}.$$

Therefore, the similarity is precisely:

$$\text{similarity}(\text{point 1}, \text{point 2}) = e^{-\text{distance}(q,p)^2} = e^{-2} = 0.135.$$

| Point | X <sub>1</sub> | X <sub>2</sub> | Sim 1 | Sim 2 | Sim 3 | Sim 4 | Sim 5 | Sim 6 | Sim 7 | y |
|-------|----------------|----------------|-------|-------|-------|-------|-------|-------|-------|---|
| 1     | 0              | 0              | 1     | 0.135 | 0.135 | 0.135 | 0.135 | 0.368 | 0.368 | 0 |
| 2     | -1             | 0              | 0.135 | 1     | 0.368 | 0.368 | 0.018 | 0.135 | 0.018 | 0 |
| 3     | 0              | -1             | 0.135 | 0.368 | 1     | 0.018 | 0.135 | 0.007 | 0.368 | 0 |
| 4     | 0              | 1              | 0.135 | 0.368 | 0.018 | 1     | 0.135 | 0.368 | 0.007 | 1 |
| 5     | 1              | 0              | 0.135 | 0.018 | 0.135 | 0.135 | 1     | 0.007 | 0.368 | 1 |
| 6     | -1             | 1              | 0.368 | 0.135 | 0.007 | 0.368 | 0.007 | 1     | 0     | 1 |
| 7     | 1              | -1             | 0.368 | 0.018 | 0.368 | 0.007 | 0.368 | 0     | 1     | 1 |

## Overfitting and underfitting with the rbf kernel - The gamma parameter

Figure . The gamma parameter determines how wide the curve is. Notice that for small values of gamma, the curve is very wide, and for large values of gamma, the curve is very narrow.

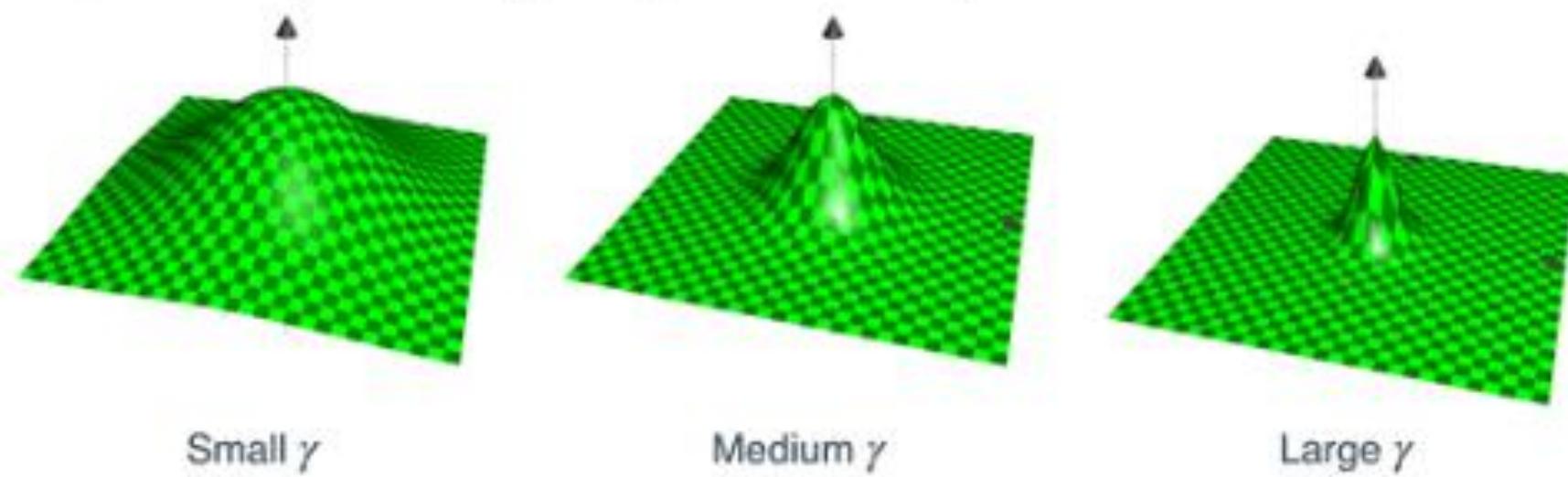
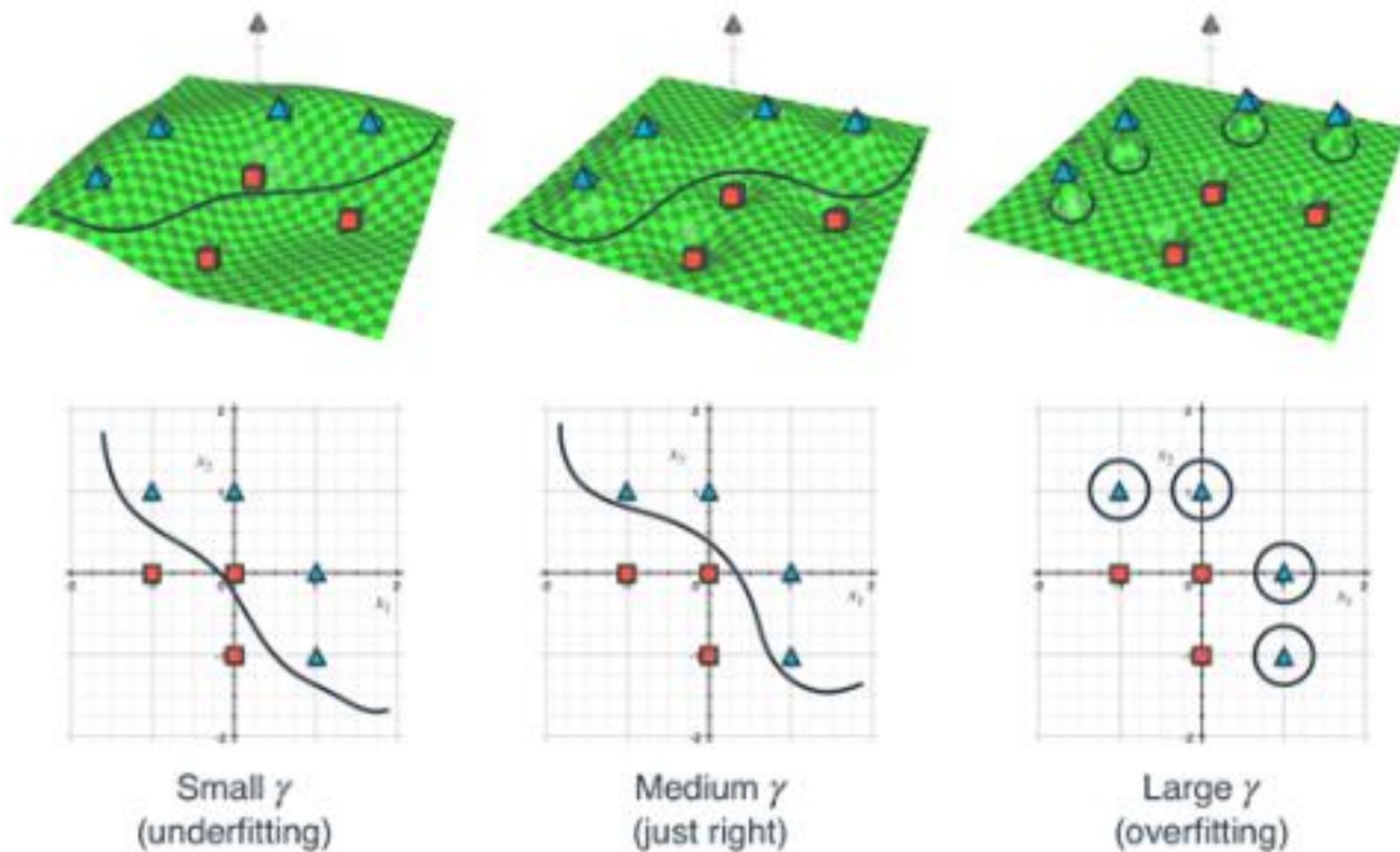


Figure . Three SVM classifiers with an rbf kernel and different values of gamma. Notice that the classifier on the left (small value of gamma) underfits, since it doesn't classify all the points well. The classifier on the right (large value of gamma) overfits, as it only manages to draw a small circle around each of the triangles, while classifying everything else as a square. The classifier in the middle is good, as it draws a boundary that is simple enough, yet classifies the points correctly.



# Radial basis functions

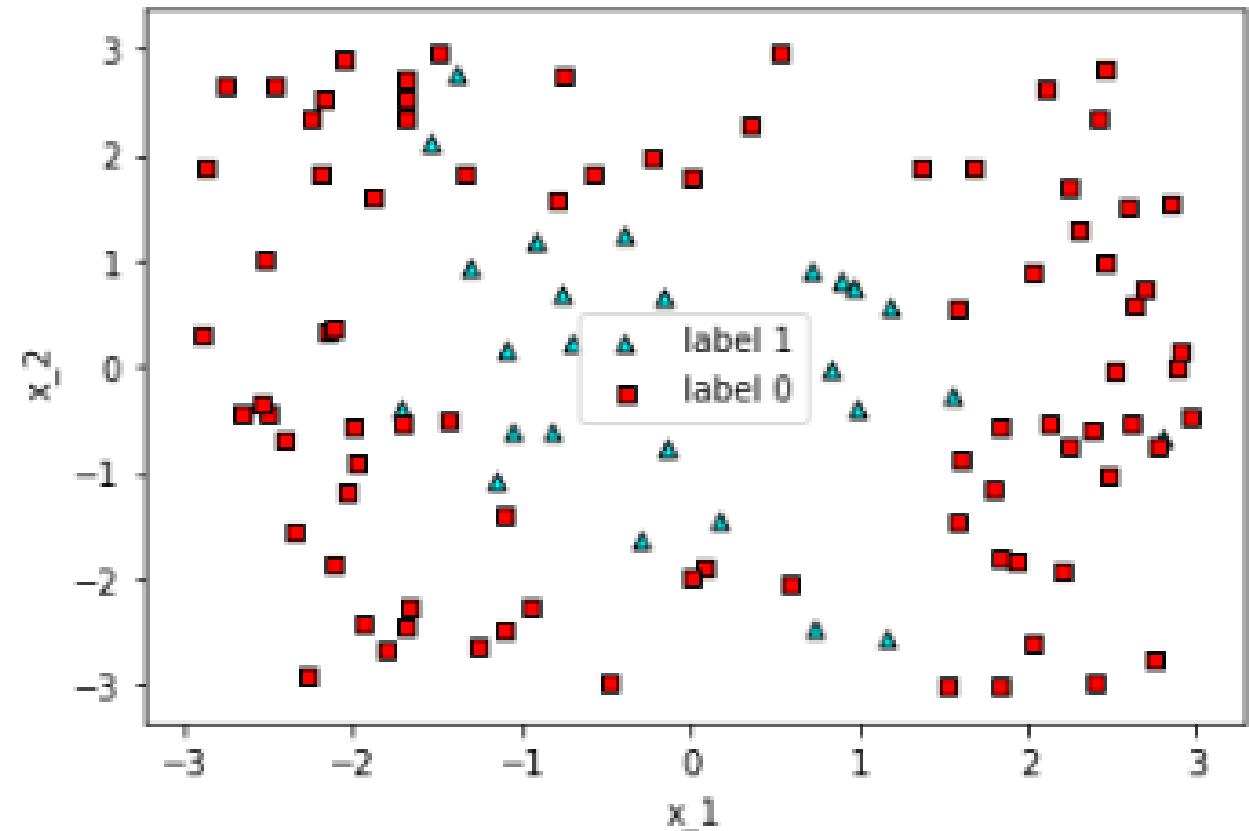
- The equation for the radial basis function doesn't change much when we add the gamma parameter, all we have to do is multiply the exponent by gamma, to get the following equation.
- In the general case, we get the following:

$$y = e^{-\gamma[(x_1-p_1)^2 + \dots + (x_n-p_n)^2]}.$$

# Radial basis functions

## Coding the polynomial kernel

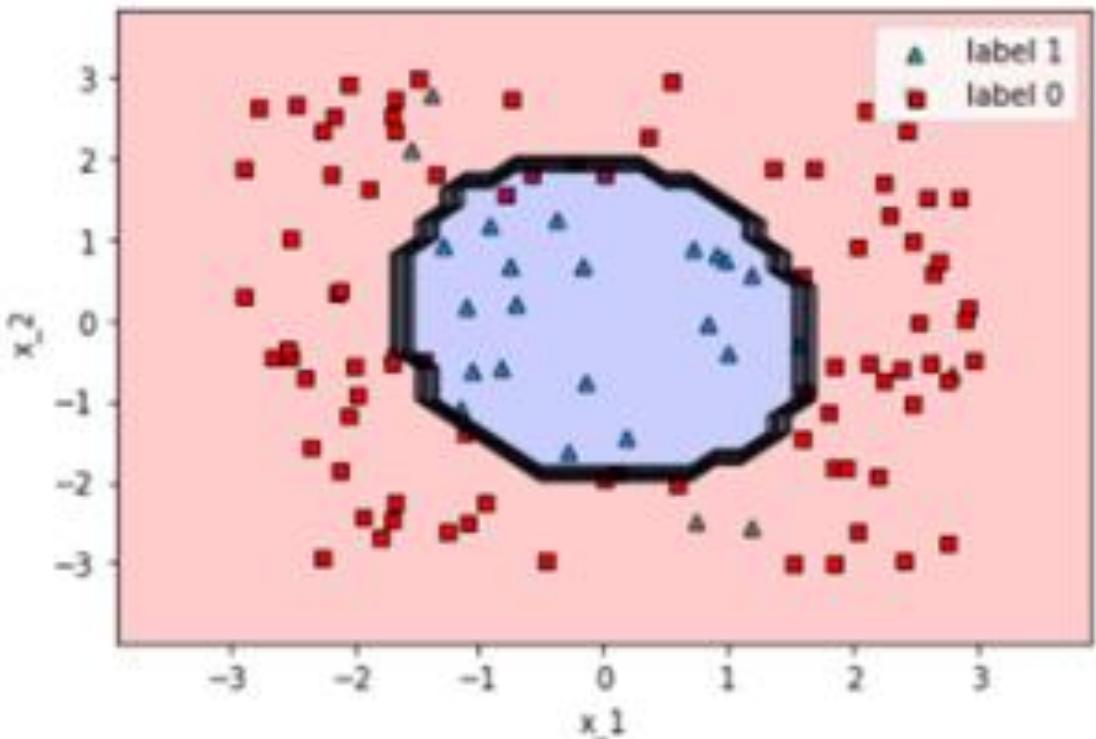
- For this, we use the following circular dataset called 'one\_circle.csv'



# Radial basis functions

- An SVM classifier with a polynomial kernel of degree 2.
- Notice that it draws a circular region.

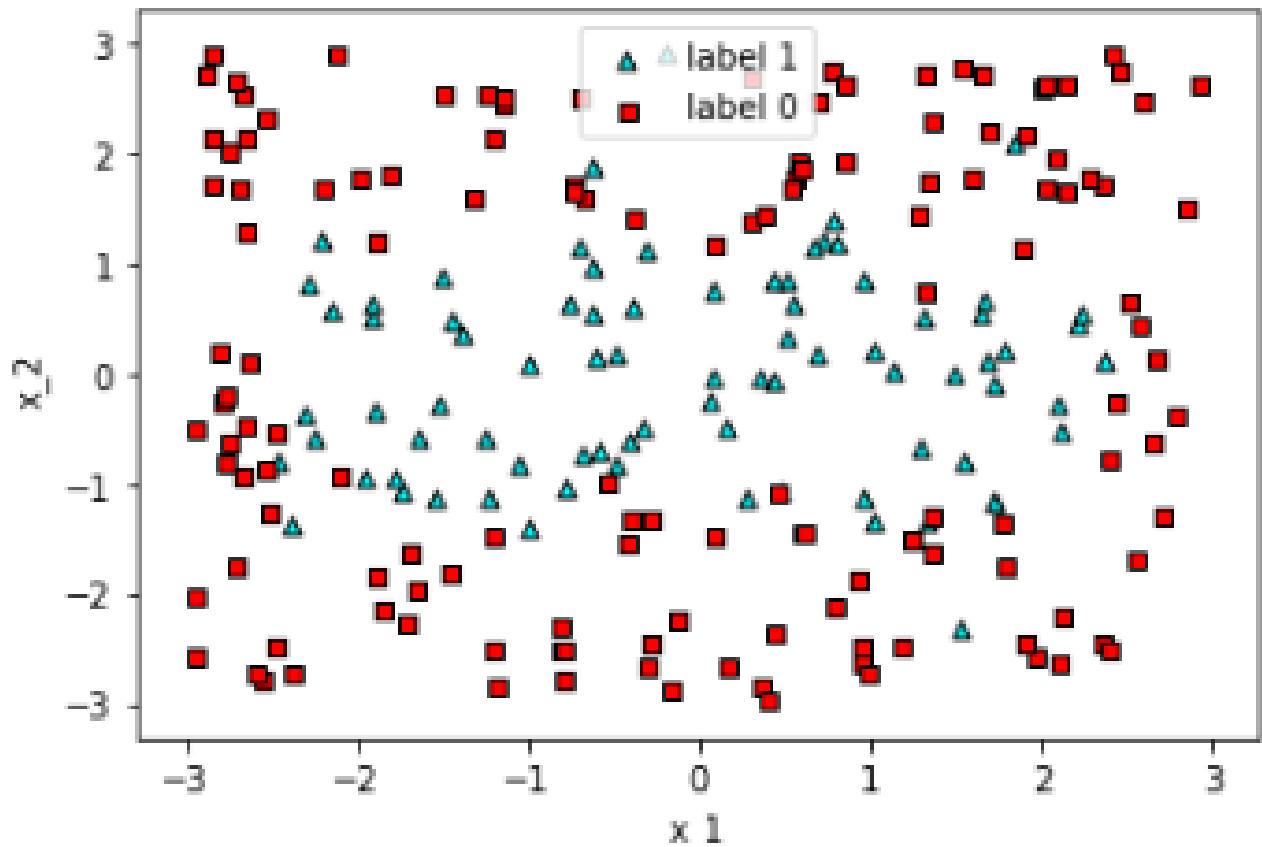
Degree = 2  
Accuracy: 0.8909090909090909



# Radial basis functions

## Coding the rbf kernel

- A dataset consisting of two intersecting circles, with noise.



# Radial basis functions

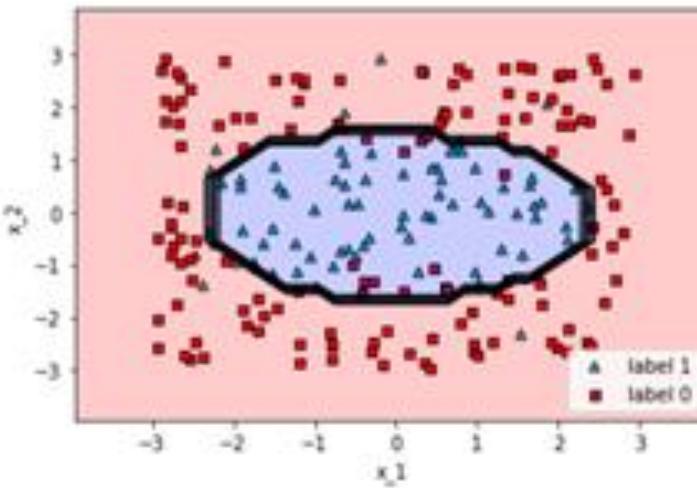
```
gamma = 0.1
svm_gamma_01 = SVC(kernel='rbf', gamma=0.1)
svm_gamma_01.fit(X, y)

gamma = 1
svm_gamma_1 = SVC(kernel='rbf', gamma=1)
svm_gamma_1.fit(X, y)

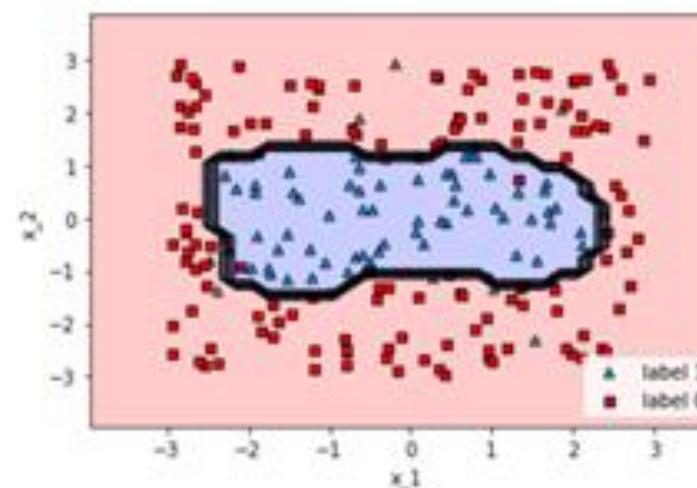
gamma = 10
svm_gamma_10 = SVC(kernel='rbf', gamma=10)
svm_gamma_10.fit(X, y)

gamma = 100
svm_gamma_100 = SVC(kernel='rbf', gamma=100)
svm_gamma_100.fit(X, y)
```

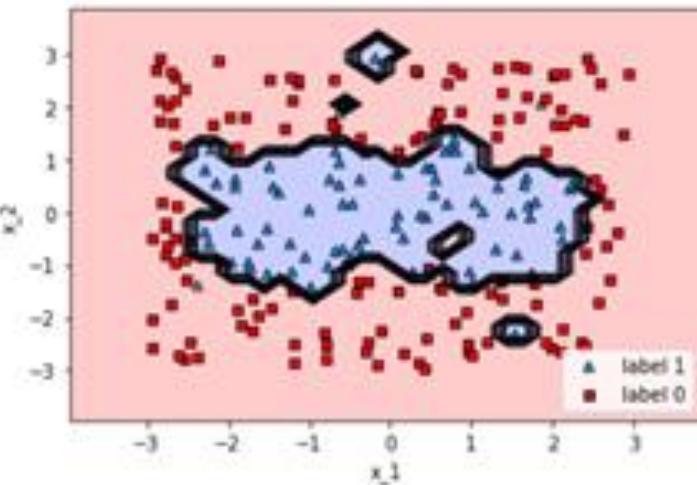
Gamma = 0.1  
Accuracy: 0.8772727272727273



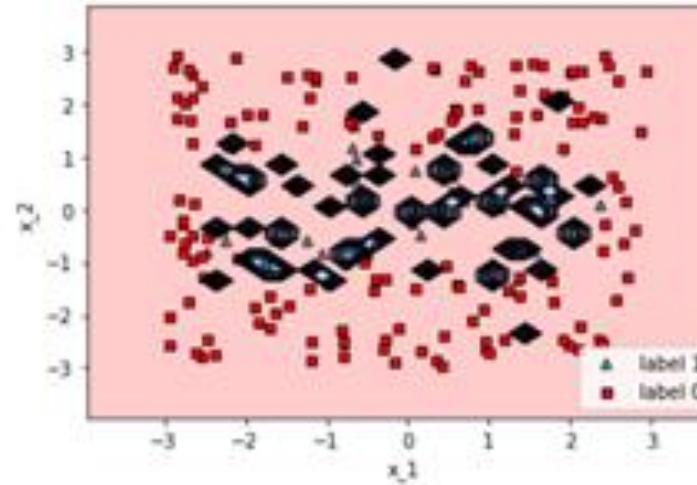
Gamma = 1  
Accuracy: 0.9045454545454545



Gamma = 10  
Accuracy: 0.9636363636363636



Gamma = 100  
Accuracy: 0.990909090909091



# Summary



- The C parameter is used to regulate between trying to classify the points correctly and trying to space out the lines.
- The kernel method is a very useful and very powerful method used to build non-linear classifiers.
- The kernel method consists of applying several functions to our data to build more columns.
- This puts our data inside a higher dimensional space, in which the points may be easier to classify with a linear classifier.

# 13: Putting it in practice: A real-life example of data engineering and machine learning

# Putting it in practice

This lesson covers

- Using pandas to read datasets.
- Cleaning up data.
- Preprocessing data in order to make it readable by our model.
- Dealing with missing data.
- Turning categorical features into numerical features, and vice versa.
- Removing unnecessary features from our dataset.
- Splitting data into training, validation, and testing sets.

# The features of our dataset

The Titanic dataset that we'll use in this class contains the names and information of 891 passengers in the Titanic, including if they survived or not. Here are the columns of the dataset:

- **Name:** The full name of the passenger.
- **Sex:** The gender of the passenger as male or female.
- **Age:** The age of the passenger as an integer.
- **Pclass:** The class in which the passenger was traveling: first, second, or third.
- **SibSP:** The number of siblings and spouse of the passenger (0 if the passenger is travelling alone).

# Using pandas to load the dataset

- We can load the dataset as a DataFrame using the following command.

```
raw_data = pd.read_csv('./titanic.csv')
```

- This command stores the dataset into a pandas DataFrame called ‘raw\_data’.
- We call it raw data because our goal is to clean it and preprocess it later.

# Using pandas to load the dataset

| PassengerId | Survived | Pclass | Name                                                                        | Sex    | Age  | SibSp | Parch | Ticket                       | Fare    | Cabin | Embarked |
|-------------|----------|--------|-----------------------------------------------------------------------------|--------|------|-------|-------|------------------------------|---------|-------|----------|
| 0           | 1        | 0      | Braund, Mr. Owen Harris                                                     | male   | 22.0 | 1     | 0     | A/5 21171                    | 7.2500  | NaN   | S        |
| 1           | 2        | 1      | Cumings, Mrs. John Bradley (Florence Briggs Th...<br>Heikkinen, Miss. Laina | female | 38.0 | 1     | 0     | PC 17599<br>STON/O2. 3101282 | 71.2833 | C85   | C        |
| 2           | 3        | 1      | Allen, Mr. William Henry                                                    | male   | 35.0 | 0     | 0     | 113803<br>373450             | 53.1000 | C123  | S        |
| 3           | 4        | 1      | Futrelle, Mrs. Jacques Heath (Lily May Peel)                                | female | 35.0 | 1     | 0     |                              | 8.0500  | NaN   | S        |
| 4           | 5        | 0      | Montvila, Rev. Juozas                                                       | male   | 27.0 | 0     | 0     |                              |         | NaN   | S        |
| ...         | ...      | ...    | Graham, Miss. Margaret Edith                                                | female | 19.0 | 0     | 0     | 211536<br>W./C. 6607         | 13.0000 | NaN   | S        |
| 886         | 887      | 0      | Johnston, Miss. Catherine Helen "Carrie"                                    | female | NaN  | 1     | 2     | 112053<br>111369             | 30.0000 | B42   | S        |
| 887         | 888      | 1      | Behr, Mr. Karl Howell                                                       | male   | 26.0 | 0     | 0     | 370376                       | 23.4500 | NaN   | C        |
| 888         | 889      | 1      | Dooley, Mr. Patrick                                                         | male   | 32.0 | 0     | 0     |                              | 30.0000 | C148  | Q        |
| 889         | 890      | 0      |                                                                             |        |      |       |       |                              |         |       |          |
| 890         | 891      | 0      |                                                                             |        |      |       |       |                              |         |       |          |

# Using pandas to study our dataset

- In this section I teach you some useful pandas methods for studying our dataset.
- The first one is the length function, or len.
- This function returns the number of rows in the dataset, as follows.

```
len(raw_data)
```

Output: 891

# Using pandas to study our dataset

- This means our dataset has 891 rows.
- To output the columns, we use the `columns` property of a `DataFrame`, as follows.

```
raw_data.columns
```

```
Output: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex',
 'Age', 'SibSp', 'Parch',
 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype='object')
```

# Using pandas to study our dataset

```
raw_data[‘Survived’]
Output:
0 0
1 1
2 1
3 1
4 0
..
886 0
887 1
888 0
889 1
890 0
Name: Survived, Length: 891, dtype: int64
```

# Using pandas to study our dataset

- Now let's say we want to find out how many passengers survived.
- We can simply sum up the values in the 'Survived' column using the 'sum' function, as follows.

```
sum(raw_data['Survived'])
```

Output: 342

# Cleaning up our dataset

```
raw_data['Cabin']
```

**Output:**

```
0 NaN
```

```
1 C85
```

```
2 NaN
```

```
3 C123
```

```
4 NaN
```

```
...
...
```

```
886 NaN
```

```
887 B42
```

```
888 NaN
```

```
889 C148
```

```
890 NaN
```

```
Name: Cabin, Length: 891, dtype: object
```

# Cleaning up our dataset

```
raw_data.isna().sum()
```

**Output:**

|             |     |
|-------------|-----|
| PassengerId | 0   |
| Survived    | 0   |
| Pclass      | 0   |
| Name        | 0   |
| Sex         | 0   |
| Age         | 177 |
| SibSp       | 0   |
| Parch       | 0   |
| Ticket      | 0   |
| Fare        | 0   |
| Cabin       | 687 |
| Embarked    | 2   |

# Dropping columns with missing data

- In this case, ‘Cabin’ is not looking like a good feature. Out of 891 rows, 687 don’t have a value. This feature should be removed.
- We can do it with the drop function in pandas as follows.
- Let’s make a new DataFrame called clean\_data in order to store the data we’re about to clean up.

```
clean_data = raw_data.drop('Cabin', axis=1)
```

# How to not lose the entire column - filling in missing data

```
clean_data['Age']
Output:
0 22.0
1 38.0
2 26.0
3 35.0
4 35.0
 ...
886 27.0
887 19.0
888 NaN
889 26.0
890 32.0
Name: Age, Length: 891, dtype: float64
```

# How to not lose the entire column - filling in missing data

- The following code will first calculate the median, using the ‘median’ function (we get 28).
- Next, we use the ‘fillna’ function, which will fill in the missing values with the value we give it.

```
median_age = clean_data["Age"].median() #A
clean_data["Age"] = clean_data["Age"].fillna(median_age) #B
```

# Looking at our data and saving it for future use

| PassengerId | Survived | Pclass | Name                                                                        | Sex    | SibSp | Parch | Ticket                          | Embarked | Centered_Normalized_Fare | Centered_Normalized_Age |
|-------------|----------|--------|-----------------------------------------------------------------------------|--------|-------|-------|---------------------------------|----------|--------------------------|-------------------------|
| 0           | 1        | 0      | Braund, Mr. Owen Harris                                                     | male   | 1     | 0     | A/5 21171                       | S        | -0.971698                | -0.442427               |
| 1           | 2        | 1      | Cumings, Mrs. John Bradley (Florence Briggs Th...<br>Heikkinen, Miss. Laina | female | 1     | 0     | PC 17599<br>STON/O2.<br>3101282 | C<br>S   | -0.721729<br>-0.969063   | -0.044516<br>-0.342950  |
| 2           | 3        | 1      | Futrelle, Mrs. Jacques Heath (Lily May Peel)                                | female | 0     | 0     | 113803                          | S        | -0.792711                | -0.119125               |
| 3           | 4        | 0      | Allen, Mr. William Henry                                                    | male   | 0     | 0     | 373450                          | S        | -0.968575                | -0.119125               |
| ...         | ...      | ...    | ...                                                                         | ...    | ...   | ...   | ...                             | ...      | ...                      | ...                     |
| 886         | 887      | 0      | Montvila, Rev. Juozas                                                       | male   | 0     | 0     | 211536                          | S        | -0.949251                | -0.318080               |
| 887         | 888      | 1      | Graham, Miss. Margaret Edith                                                | female | 0     | 0     | 112053                          | S        | -0.882888                | -0.517036               |
| 888         | 889      | 0      | Johnston, Miss. Catherine Helen "Carrie"                                    | female | 1     | 2     | W.C. 6607                       | S        | -0.908457                | -0.293211               |
| 889         | 890      | 1      | Behr, Mr. Karl Howell                                                       | male   | 0     | 0     | 111369                          | C        | -0.882888                | -0.342950               |
| 890         | 891      | 0      | Dooley, Mr. Patrick                                                         | male   | 0     | 0     | 370376                          | Q        | -0.969746                | -0.193733               |

891 rows x 11 columns

# Feature engineering

- In this section, we'll call the DataFrame preprocessed\_data.
- Since we have stored the cleaned data in a CSV in subsection, we can recover it with the following line of code.

```
preprocessed_data = pd.read_csv('clean_titanic_data.csv')
```

# Turning categorical data into numerical data

Let's look at the 'gender' column. It has two categories, 'female' and 'male'. One way to turn them into numbers is to simply assign a 0 to all the females and a 1 to all the males. This technique will work, since the machine learning will conclude one of the following two things:

- Passengers are more likely to survive if they are female than male.
- Passengers are more likely to survive if they are male than female.

|             | gender |
|-------------|--------|
| Passenger 1 | F      |
| Passenger 2 | M      |
| Passenger 3 | M      |
| Passenger 4 | F      |



|             | gender_female | gender_male |
|-------------|---------------|-------------|
| Passenger 1 | 1             | 0           |
| Passenger 2 | 0             | 1           |
| Passenger 3 | 0             | 1           |
| Passenger 4 | 1             | 0           |

|             | embarked |
|-------------|----------|
| Passenger 1 | Q        |
| Passenger 2 | S        |
| Passenger 3 | C        |
| Passenger 4 | S        |



|             | embarked_c | embarked_q | embarked_s |
|-------------|------------|------------|------------|
| Passenger 1 | 0          | 1          | 0          |
| Passenger 2 | 0          | 0          | 1          |
| Passenger 3 | 1          | 0          | 0          |
| Passenger 4 | 0          | 0          | 1          |

# Turning categorical data into numerical data

```
gender_columns = pd.get_dummies(data['Sex'], prefix='Sex') #A
embarked_columns = pd.get_dummies(data["Pclass"], prefix="Pclass")

preprocessed_data = pd.concat([preprocessed_data, gender_columns], axis=1) #B
preprocessed_data = pd.concat([preprocessed_data, embarked_columns], axis=1)

preprocessed_data = preprocessed_data.drop(['Sex', 'Embarked'], axis=1) #C
```

# Binning

Let's look at the age column, for example. It's very nice and numerical. However, the machine learning model is interested in answering the following question: "How much does age determine survival in the Titanic?". In other words, the model would end up with one of the following two conclusions:

- The older the passenger is, the more likely they are to survive.
- The older the passenger is, the less likely they are to survive.

# Binning

- Previous was very similar to one-hot encoding, in the sense that it will turn the age column into 9 new columns.
- The code to do this is the following.

```
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80]
categorized_age = pd.cut(preprocessed_data['Age'], bins)
preprocessed_data['Categorized_age'] = categorized_age
preprocessed_data = preprocessed_data.drop(["Age"], axis=1)
```

# Binning

- It seems that neither one of statements 1 or 2 above are true, since the lowest probability of survival is in the second class.
- Therefore, we should treat this feature as categorical instead, and one-hot encode it. The following lines of code will do it.

```
categorized_pclass_columns = pd.get_dummies(preprocessed_data['Pclass'], prefix='Pclass')
preprocessed_data = pd.concat([preprocessed_data, categorized_pclass_columns], axis=1)
preprocessed_data = preprocessed_data.drop(['Pclass'], axis=1)
```

# Feature selection

- The ticket and the PassengerID features have the same problem as the name feature, since there is a unique one for each passenger.
- We'll remove those two columns as well. The drop function will help us do this.

```
preprocessed_data = preprocessed_data.drop(['Name', 'Ticket',
'PassengerId'], axis=1)
```

# Saving for future use

|   | Survived | SibSp | Parch | Fare    | Sex_female | Sex_male | Pclass_C | Pclass_Q | Pclass_S | Pclass_U | ... | Categorized_age_(10, 20] | Categorized_age_(20, 30] |
|---|----------|-------|-------|---------|------------|----------|----------|----------|----------|----------|-----|--------------------------|--------------------------|
| 0 | 0        | 1     | 0     | 7.2500  | 0          | 1        | 0        | 0        | 1        | 0        | ... | 0                        | 1                        |
| 1 | 1        | 1     | 0     | 71.2833 | 1          | 0        | 1        | 0        | 0        | 0        | ... | 0                        | 0                        |
| 2 | 1        | 0     | 0     | 7.9250  | 1          | 0        | 0        | 0        | 1        | 0        | ... | 0                        | 1                        |
| 3 | 1        | 1     | 0     | 53.1000 | 1          | 0        | 0        | 0        | 1        | 0        | ... | 0                        | 0                        |
| 4 | 0        | 0     | 0     | 8.0500  | 0          | 1        | 0        | 0        | 1        | 0        | ... | 0                        | 0                        |

5 rows × 27 columns

# Saving for future use

- As usual, we can save this dataset in its own CSVfile for later use, with the following line of

```
preprocessed_data.to_csv('./preprocessed_titanic_data.csv',
index=None)
```

# Training our models

- As usual, we load our data from the file where we saved it in the previous section. We'll call it data.

```
data = pd.read_csv('./preprocessed_titanic_data.csv')
```

- Remark: If you run the code from the repo by yourself, you may get different numbers.

# Training our models

- Splitting the dataset into features and labels is very simple in pandas, and we've seen how to do it before. We use the drop feature.
- We'll call the resulting DataFrames features and labels.
- Since the labels only has one column, it is a Series instead of a DataFrame.

```
features = data.drop(["Survived"], axis=1)
labels = data["Survived"]
```

# Training our models

- If we wanted to split our data into 80% training and 20% testing, we would use the following code.

```
from sklearn.model_selection import train_test_split
features_train, features_test, labels_train, labels_test =
train_test_split(features, labels, test_size=0.2)
```

# Training our models

```
features_train, features_validation_test, labels_train, labels_validation_test =
 train_test_split(features, labels, test_size=0.4)

features_validation, features_test, labels_validation, labels_test =
 train_test_split(features_validation_test, labels_validation_test, test_size=0.5)
```

# Training several models on our dataset

- First, we'll start by training a logistic regression model.
- We can do this using the LogisticRegression object from sklearn, and the fit function, as follows.

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression() #A
lr_model.fit(features_train, labels_train) #B
```

# Training several models on our dataset

```
from sklearn.tree import DecisionTreeClassifier, GaussianNB, SVC, RandomForestClassifier,
 GradientBoostingClassifier, AdaBoostClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(features_train, labels_train)

nb_model = GaussianNB()
nb_model.fit(features_train, labels_train)

svm_model = SVC()
svm_model.fit(features_train, labels_train)

rf_model = RandomForestClassifier()
rf_model.fit(features_train, labels_train)

gb_model = GradientBoostingClassifier()
gb_model.fit(features_train, labels_train)

ab_model = AdaBoostClassifier()
ab_model.fit(features_train, labels_train)
```

# Which model is better? - Evaluating the models

- **Accuracy:** The ratio between the number of correctly labeled points and the total number of points.
- **Recall:** Among the positively labeled points, the ratio between the number of correctly labeled points and the total number of points.
- **Precision:** Among the points that have been predicted as positive, the ratio between the number of correctly labeled points and the total number of points.
- **F1-score:** The harmonic mean of the precision and recall. This is a number which is in between precision and recall, but it is closer to the smaller one of the two.

# TESTING EACH MODEL'S ACCURACY

```
lr_model.score(features_validation, labels_validation)
Output:
0.7932960893854749
```

We calculate it for all the other models, and get the following results, which I've rounded to two figures (see repo for the whole procedure):

## Accuracy

**Logistic regression:** 0.77

**Decision tree:** 0.78

**Naive Bayes:** 0.72

**SVM:** 0.68

**Random forest:** 0.7875

**Gradient boosting:** 0.81

**AdaBoost:** 0.76

# TESTING EACH MODEL'S F1-SCORE

```
lr_predicted_labels = lr_model.predict(features_validation) #A
f1_score(labels_validation, lr_predicted_labels) #B
Output:
0.6870229007633588
```

#A Using the model to make the predictions  
#B Calculating the f1 score

As before, we can do this for all the models, and get the following:

## F1-Score

**Logistic regression:** 0.69  
**Decision Tree:** 0.71  
**Naive Bayes:** 0.63  
**Support Vector Machine:** 0.42  
**Random Forest:** 0.68  
**Gradient boosting:** 0.74  
**AdaBoost:** 0.69

# Testing the model

First, let's evaluate the accuracy.

```
gb_model.score(features_test, labels_test)
Output:
0.8324022346368715
```

And now let's look at the F1-score.

```
gb_predicted_test_labels = gb_model.predict(features_test)
f1_score(labels_test, gb_predicted_test_labels)
Output:
0.8026315789473685
```

These scores are quite good for the Titanic dataset. Thus, we can comfortably say that our model is good.

# Grid search

Let's also try to train models with  $C=1$  and  $C=10$ . This gives us 4

possible models to train:

Model 1: `kernel=rbf, gamma=1, C=1.`

Model 2: `kernel=rbf, gamma=1, C=10.`

Model 3: `kernel=rbf, gamma=10, C=1.`

Model 4: `kernel=rbf, gamma=10, C=10.`

We can easily train all of those in our training set with the following 8 lines of code.

# Grid search

```
print("SVM grid search with a radial basis function kernel")

svm_1_1 = SVC(kernel='rbf', C=1, gamma=1)
svm_1_1.fit(features_train, labels_train)

svm_1_10 = SVC(kernel='rbf', C=1, gamma=10)
svm_1_10.fit(features_train, labels_train)

svm_10_1 = SVC(kernel='rbf', C=10, gamma=1)
svm_10_1.fit(features_train, labels_train)

svm_10_10 = SVC(kernel='rbf', C=10, gamma=10)
svm_10_10.fit(features_train, labels_train)
```

# Grid search

|             | C=1  | C=10        |
|-------------|------|-------------|
| gamma = 0.1 | 0.69 | <b>0.72</b> |
| gamma = 1   | 0.70 | 0.70        |
| gamma = 10  | 0.67 | 0.65        |

# Grid search

- In this case, let's explore the following space:

**Kernel:** rbf

**C:** 0.01, 0.1, 1, 10, 100

**gamma:** 0.01, 0.1, 1, 10, 100

The following code will do it.

```
svm_parameters = {'kernel': ['rbf'],
 'C': [0.01, 0.1, 1 , 10, 100],
 'gamma': [0.01, 0.1, 1, 10, 100]
 } #A
svm = SVC() #B

svm_gs = GridSearchCV(estimator = svm,
 param_grid = svm_parameters) #C

svm_gs.fit(features_train, labels_train) #D
```

# Grid search

```
svm_winner
```

**Output:**

```
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
 decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
 max_iter=-1, probability=False, random_state=None, shrinking=True,
 tol=0.001, verbose=False)
```

The winning model used an rbf kernel with gamma=0.001 and C=10.

# Using K-fold cross-validation

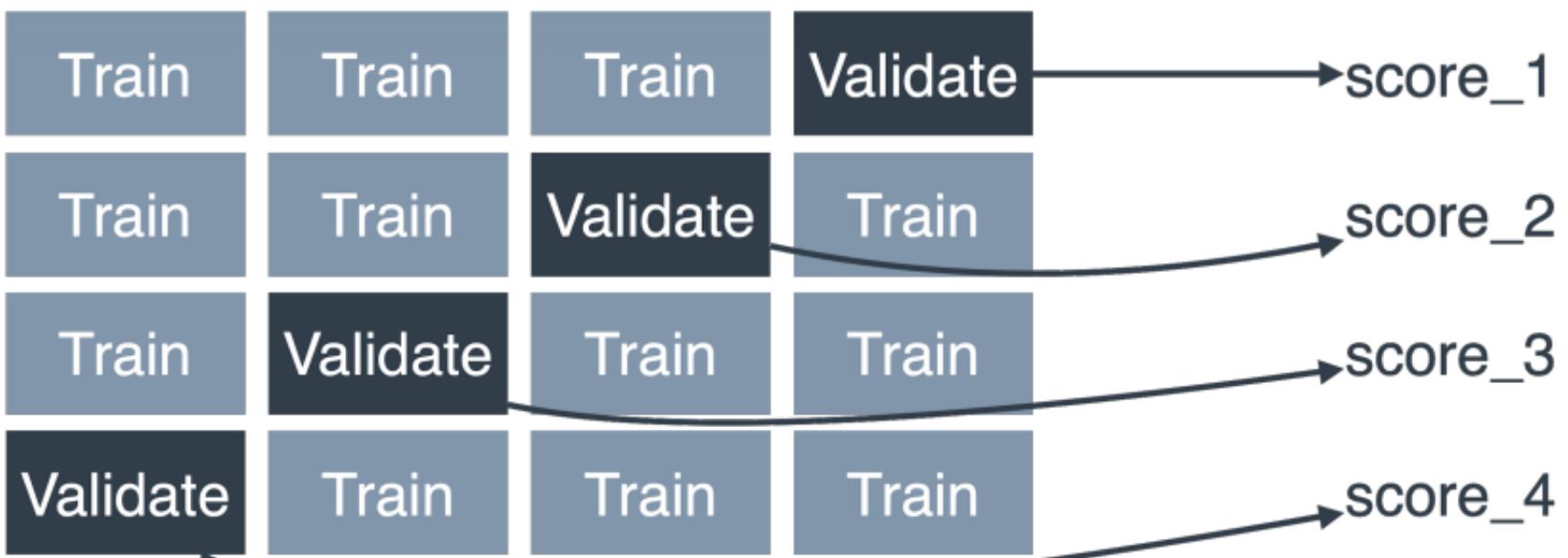
It works as follows:

1. Split the data into  $k$  equal (or almost equal) portions.
2. Train the model  $k$  times, using the union of  $k-1$  of the portions as the training set, and the remaining one as a validation set.
3. The final score of that model is the average of the validation scores from the  $k$  steps.

## Validation



## 4-fold Cross validation



$$\text{score} = \frac{\text{score}_1 + \text{score}_2 + \text{score}_3 + \text{score}_4}{4}$$

# Summary



- Pandas is a very useful Python package to open, manipulate, and save datasets.
- Cleaning up our data is necessary, as it may come with problems such as missing values.
- Features can be numerical or categorical. Numerical features are numbers, such as age. Categorical features are categories, or types, such as male/female or dog/cat/bird.