CSE419 – Artificial Intelligence and Machine Learning 2018

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/CSE419_2018

Lecture 5

Perceptron Learning

Based on Asst. Prof. Dr. David Kauchak (Pomona College) Lecture Slides

Machine learning models

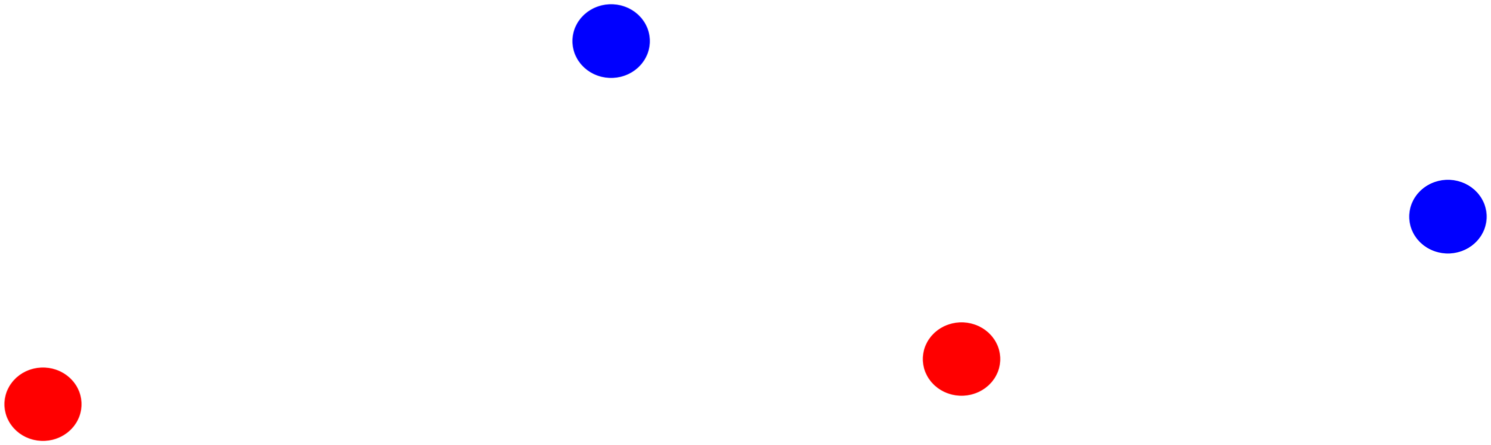
Some machine learning approaches make strong assumptions about the data

- If the assumptions are true this can often lead to better performance
- If the assumptions aren't true, they can fail miserably

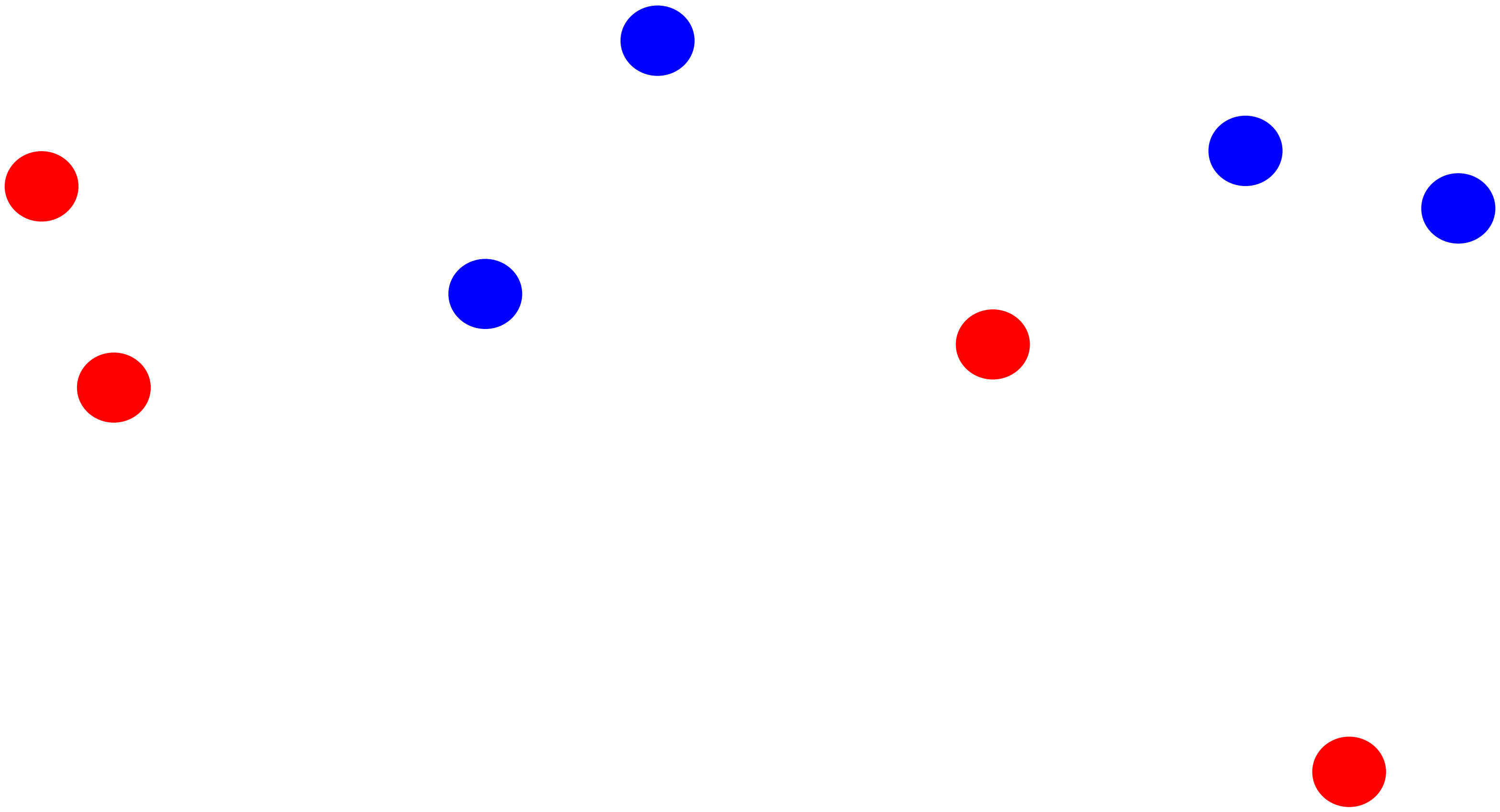
Other approaches don't make many assumptions about the data

- This can allow us to learn from more varied data
- But, they are more prone to overfitting
- and generally require more training data

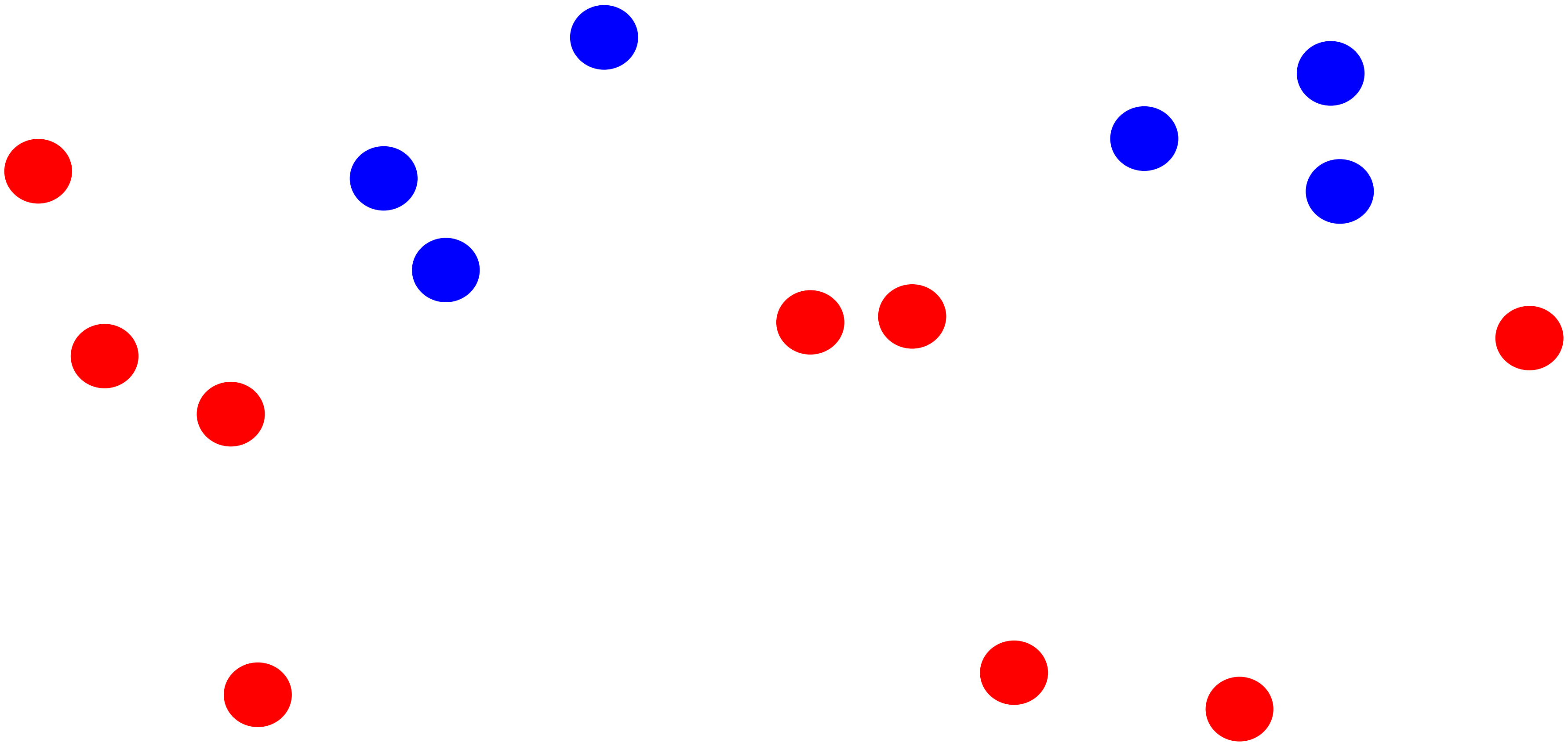
What is the data generating distribution?



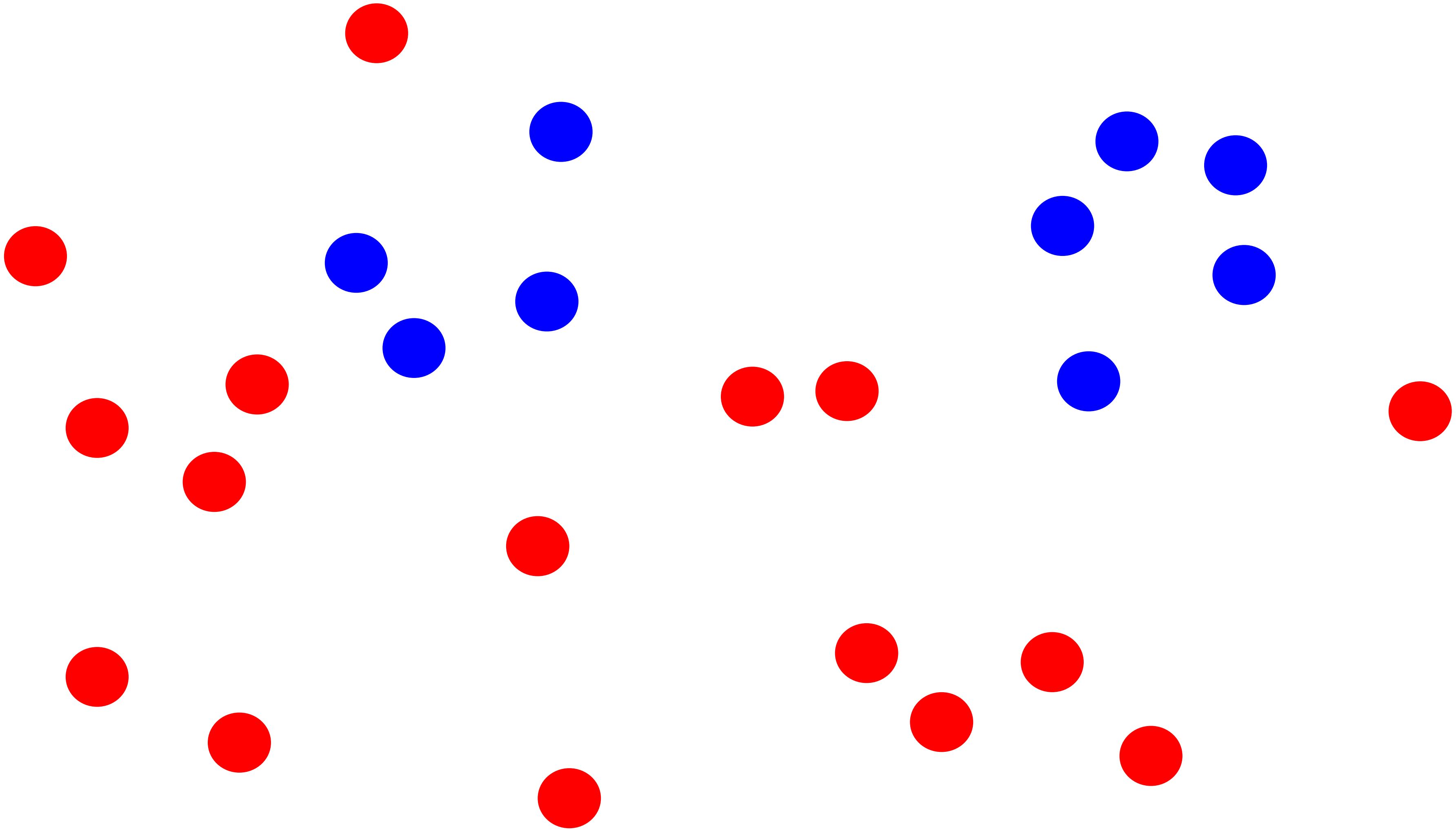
What is the data generating distribution?



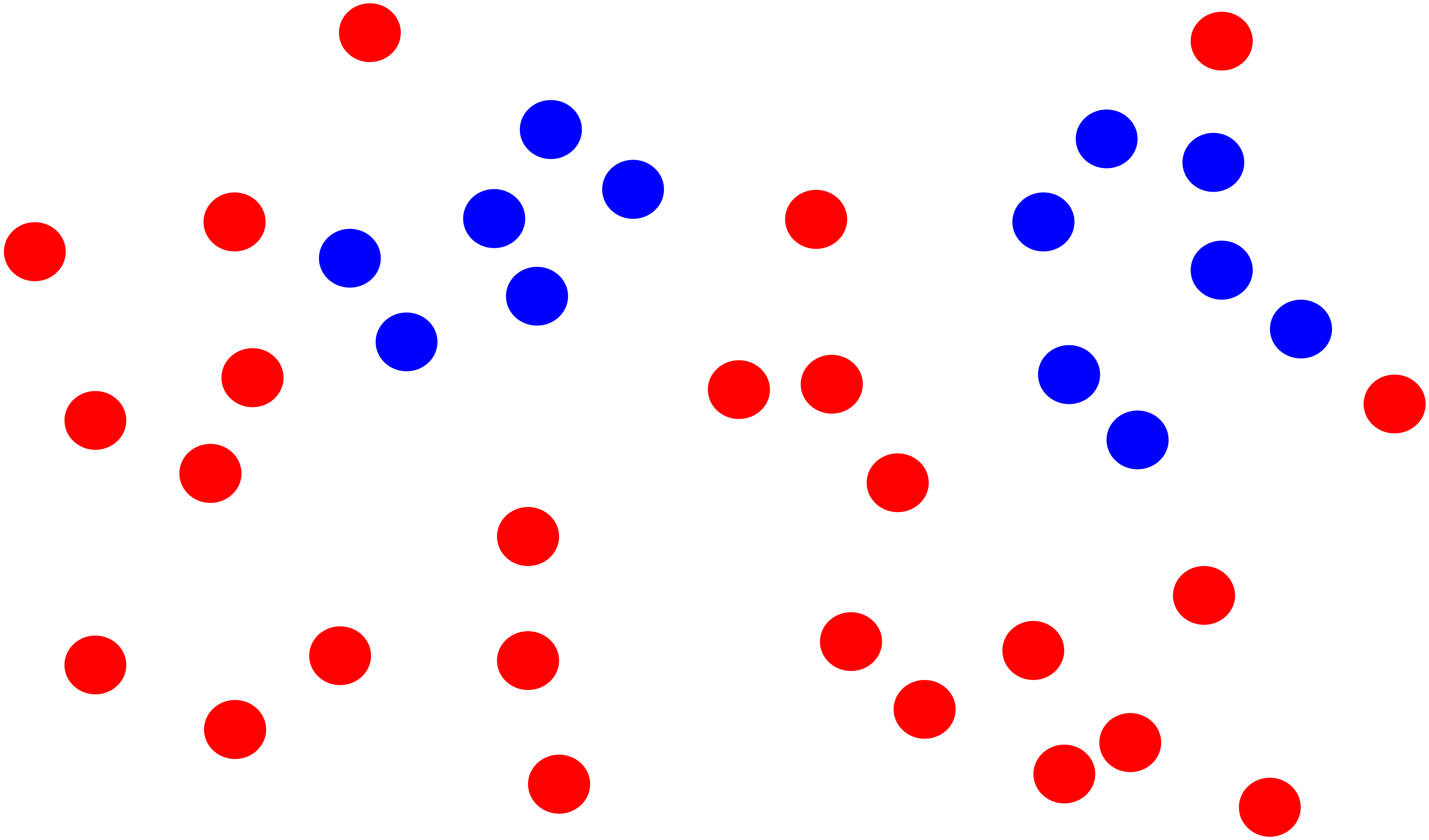
What is the data generating distribution?



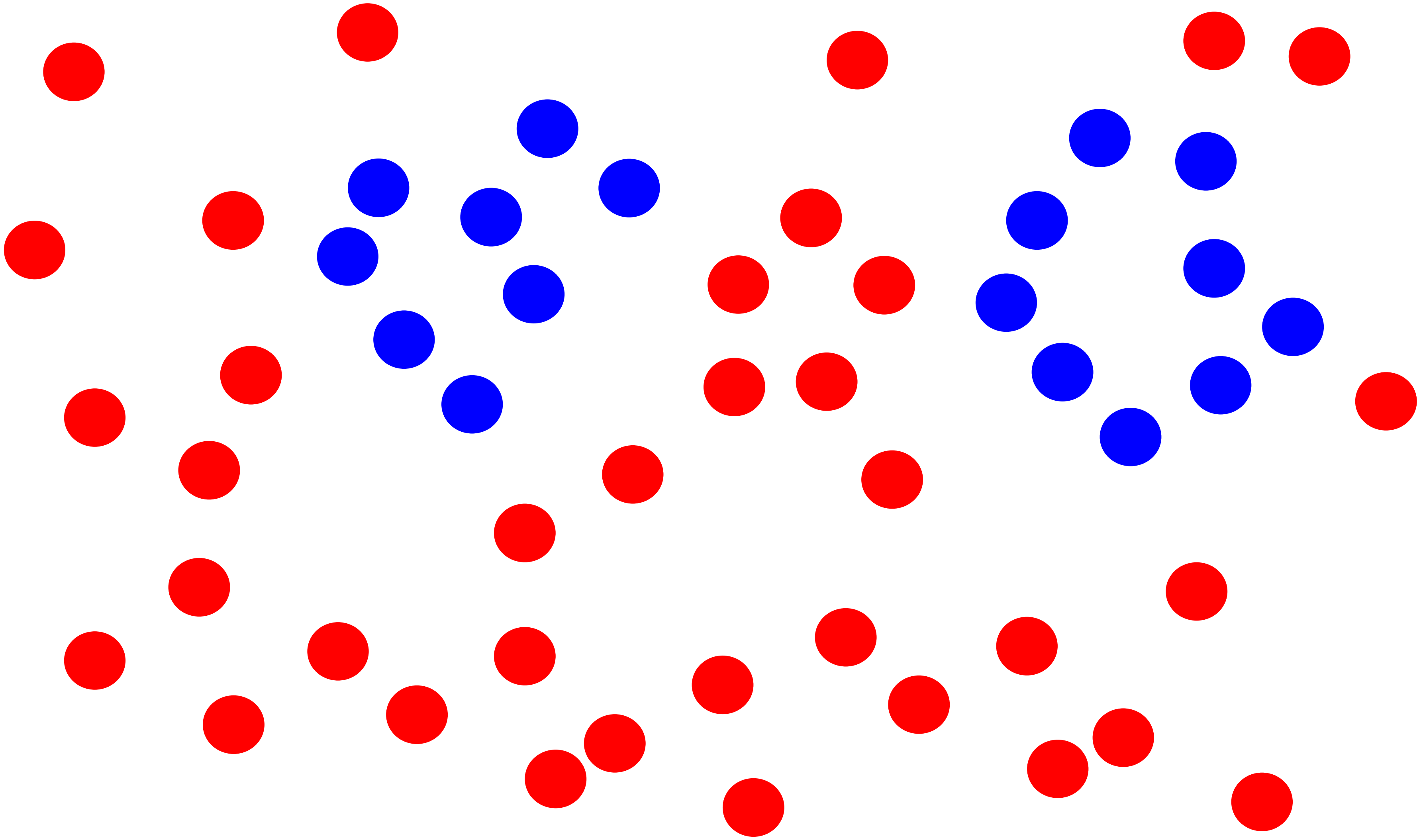
What is the data generating distribution?



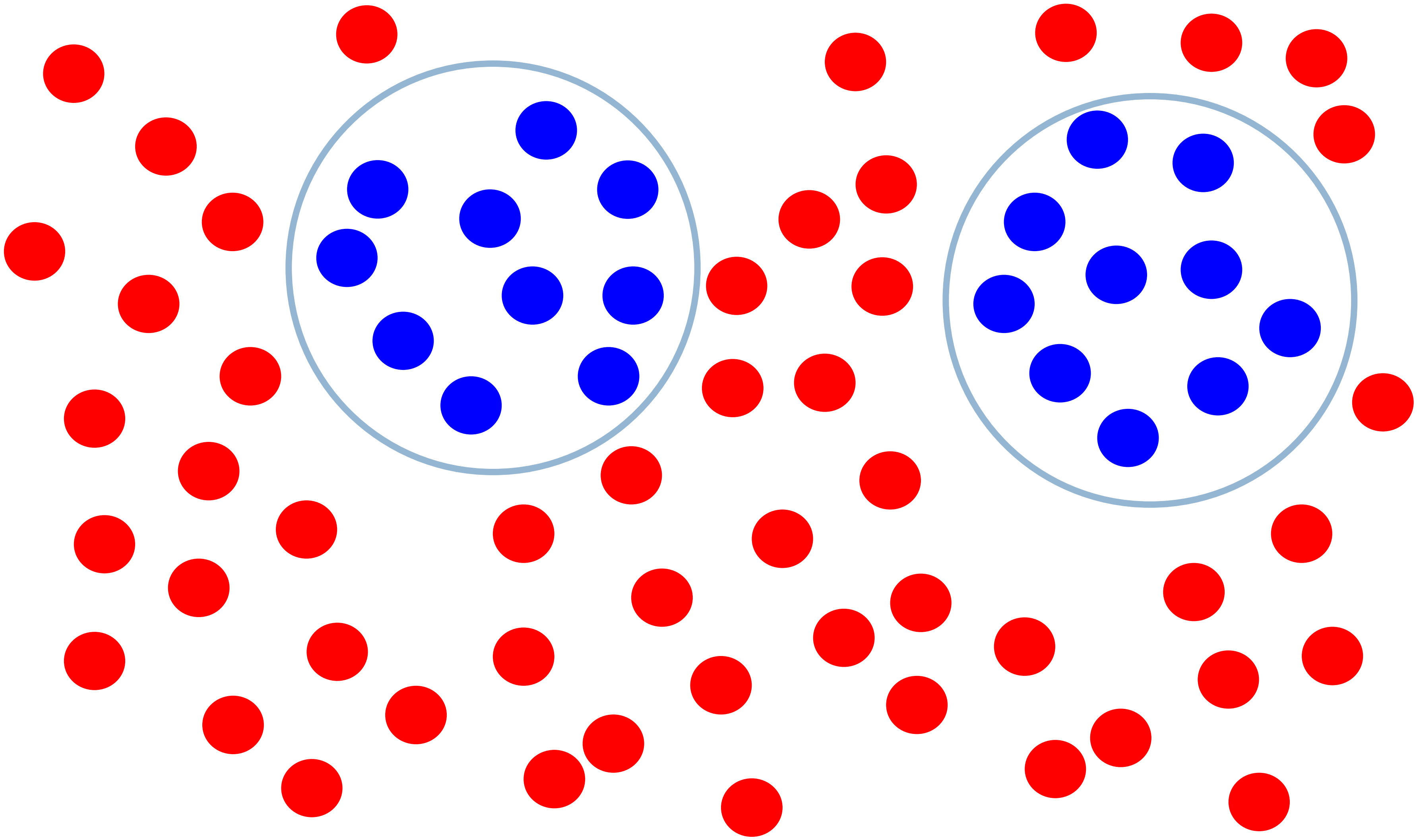
What is the data generating distribution?



What is the data generating distribution?



Actual model

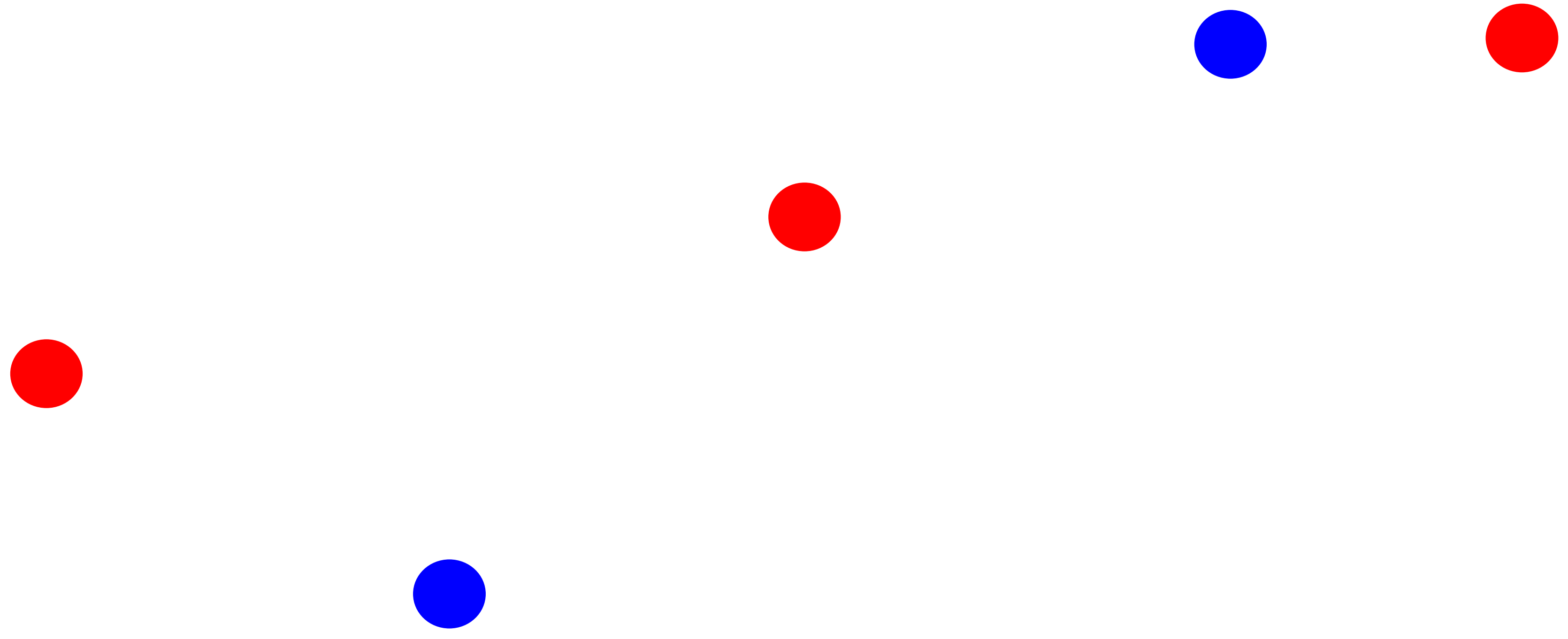


Model assumptions

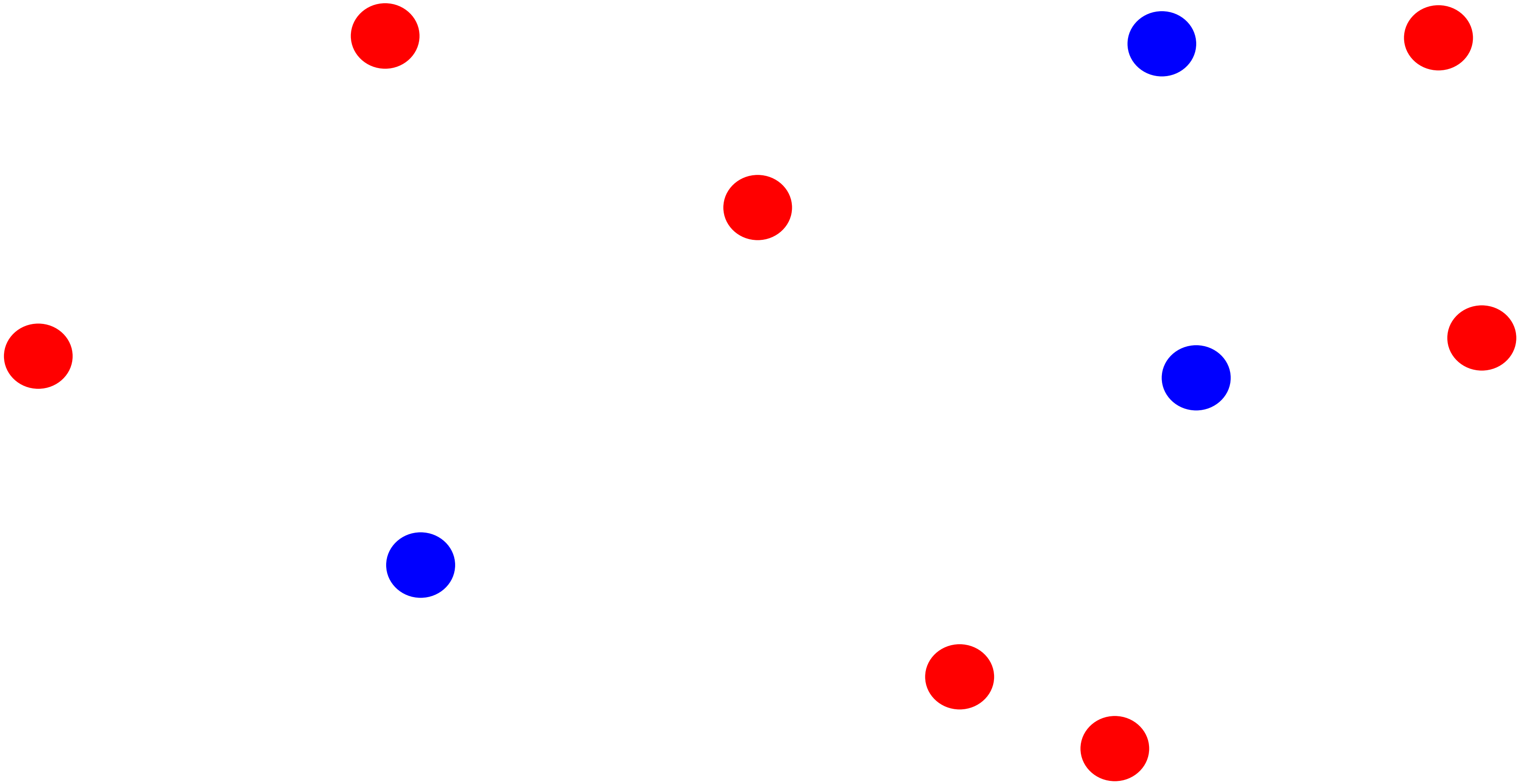
If you don’t have strong assumptions about the model, it can take you a longer to learn

Assume now that our model of the blue class is two circles

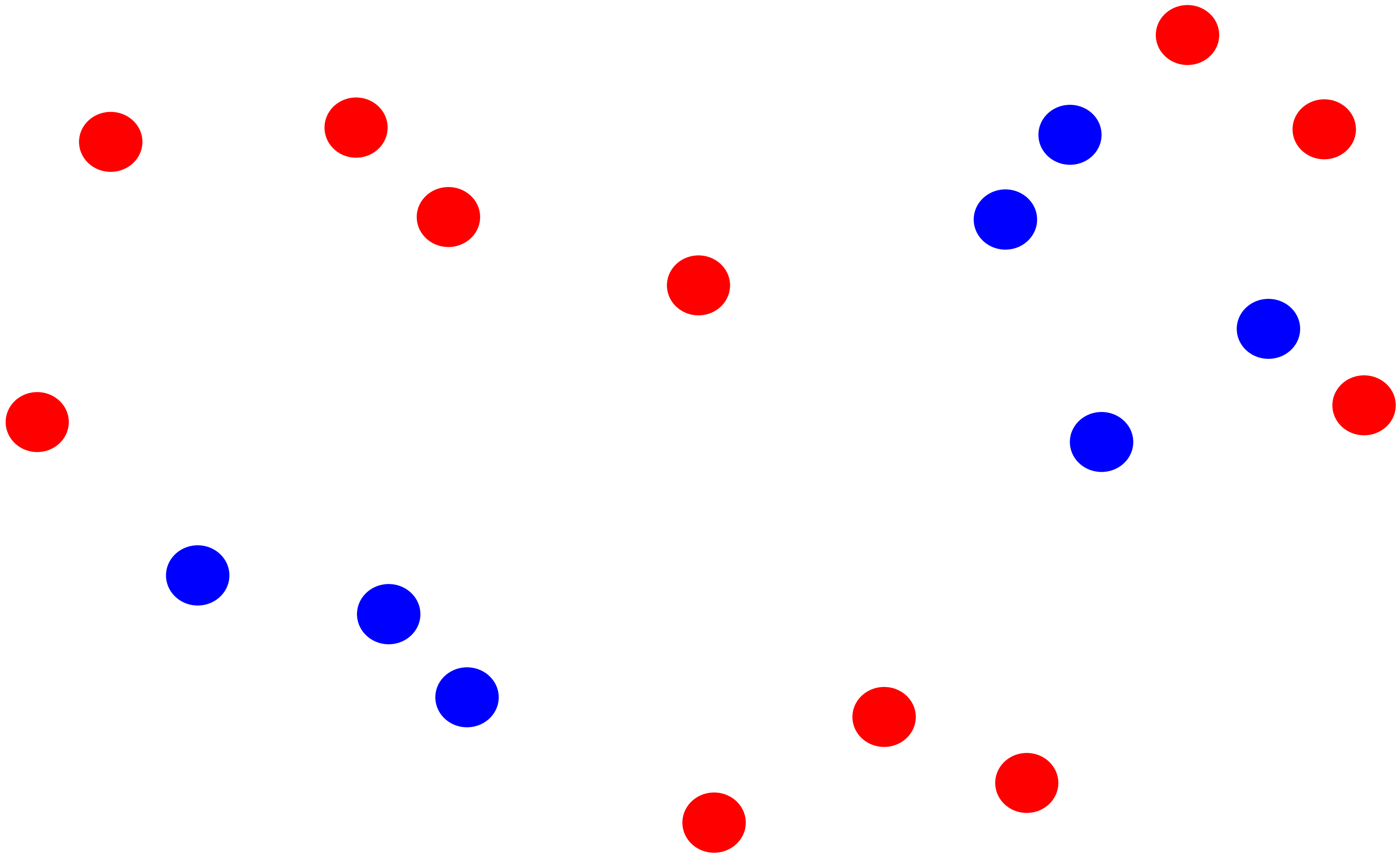
What is the data generating distribution?



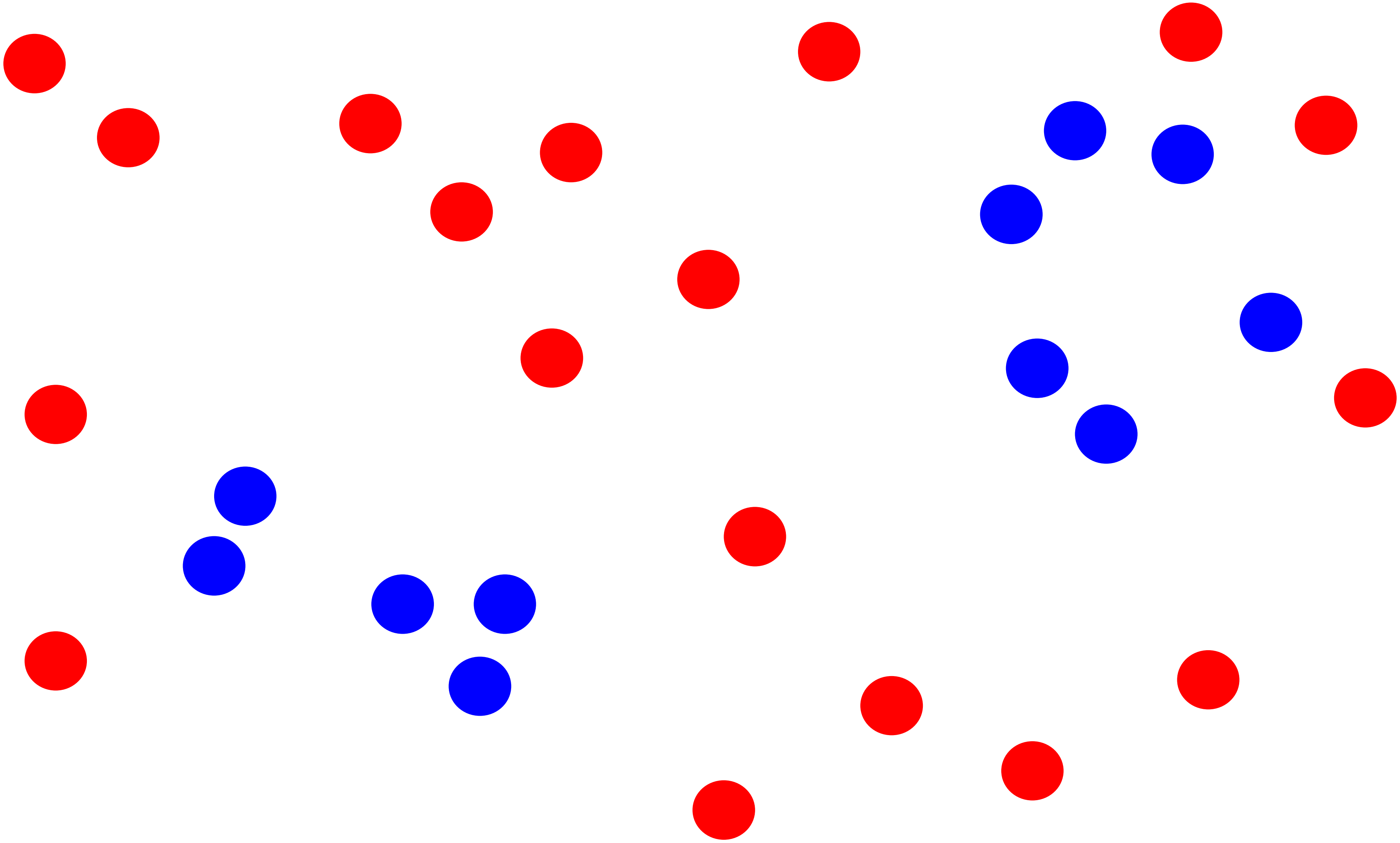
What is the data generating distribution?



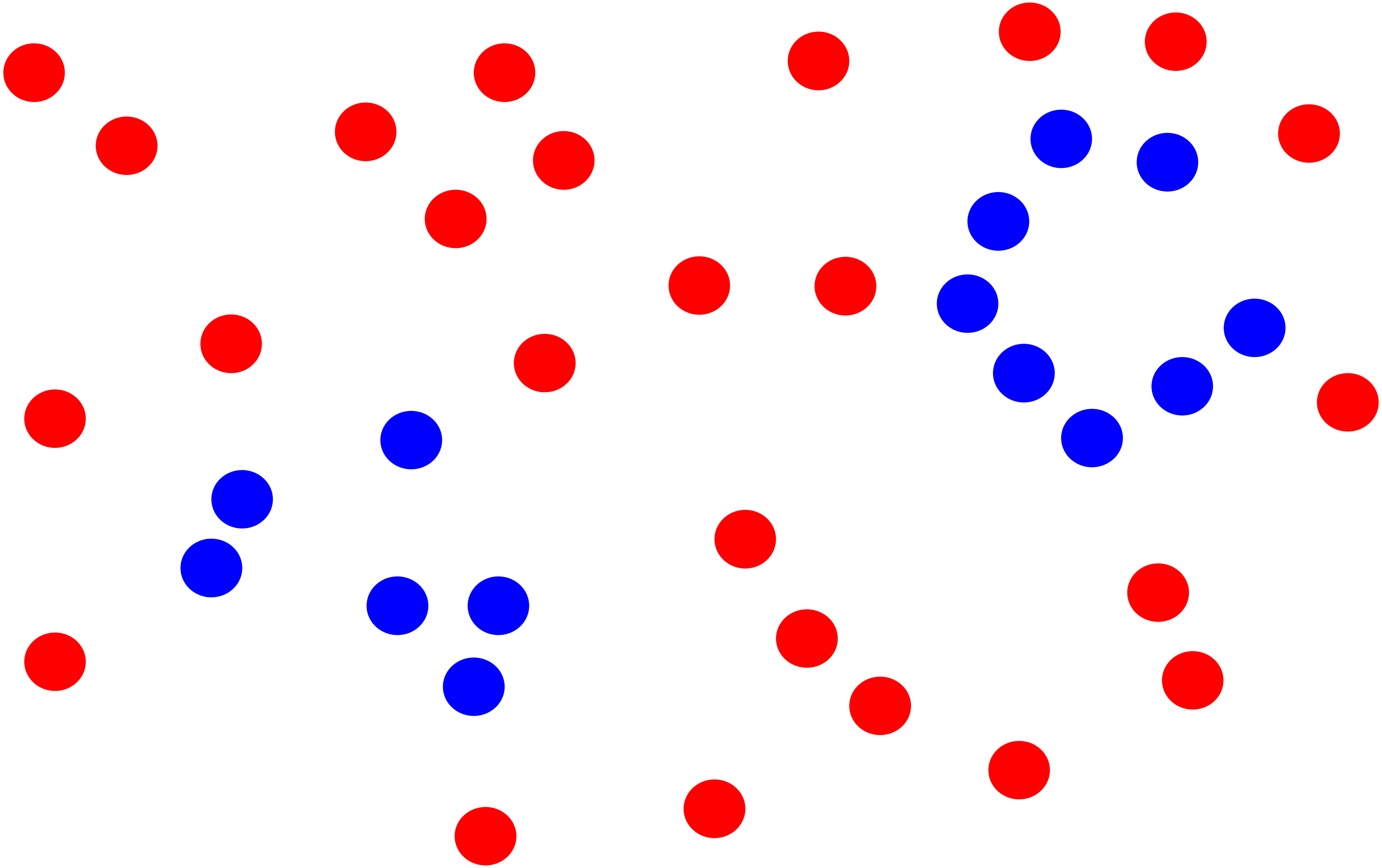
What is the data generating distribution?



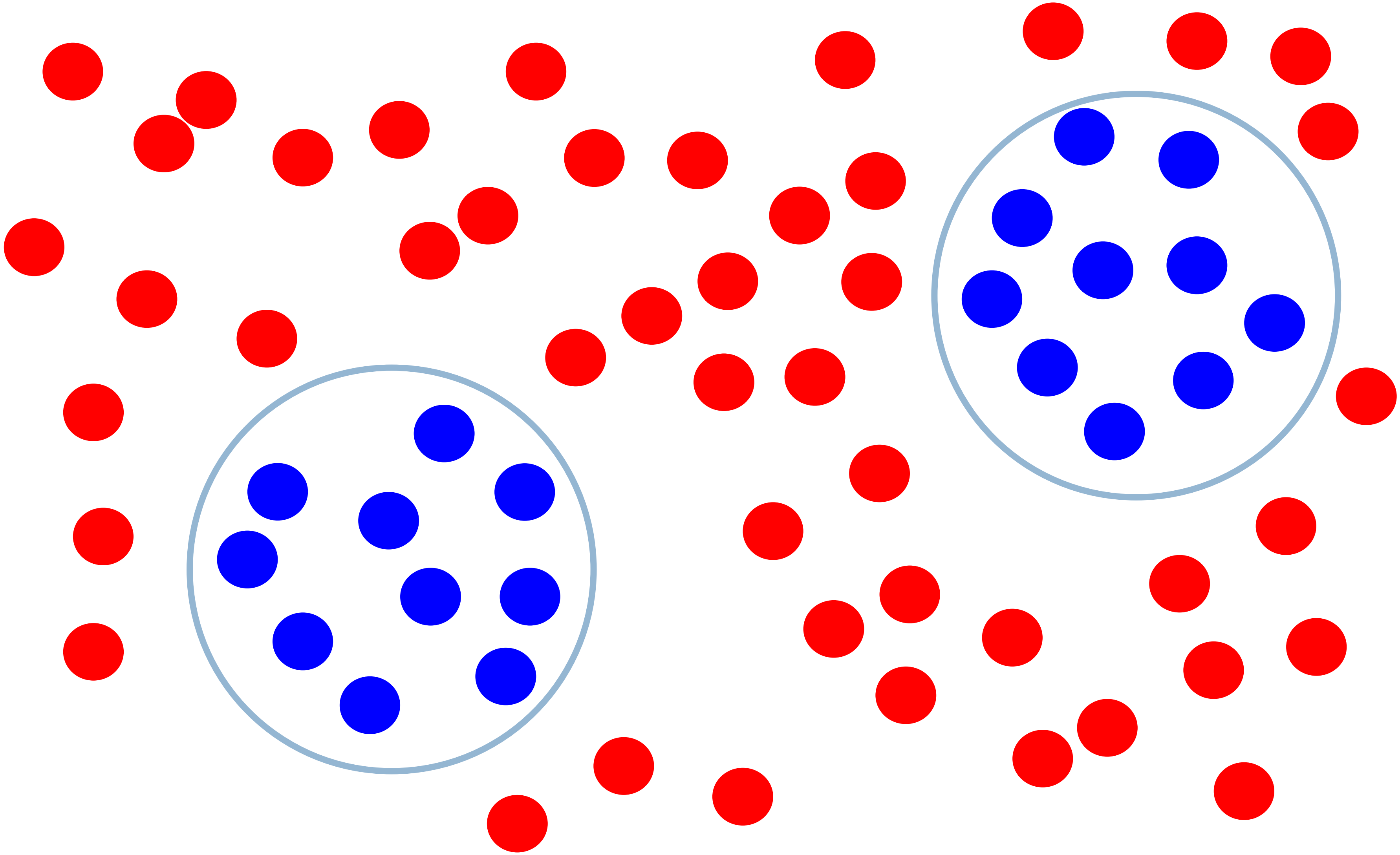
What is the data generating distribution?



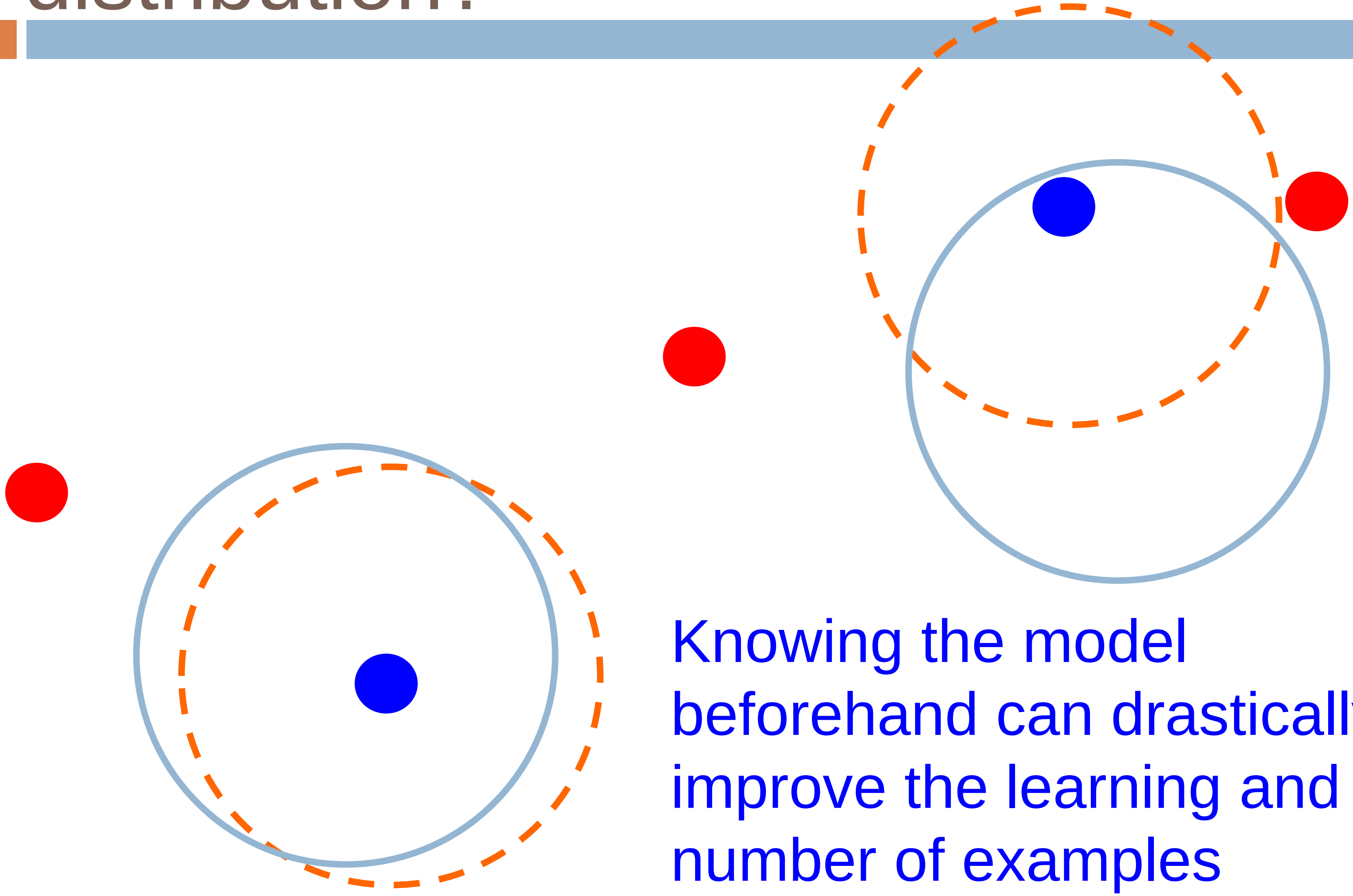
What is the data generating distribution?



Actual model

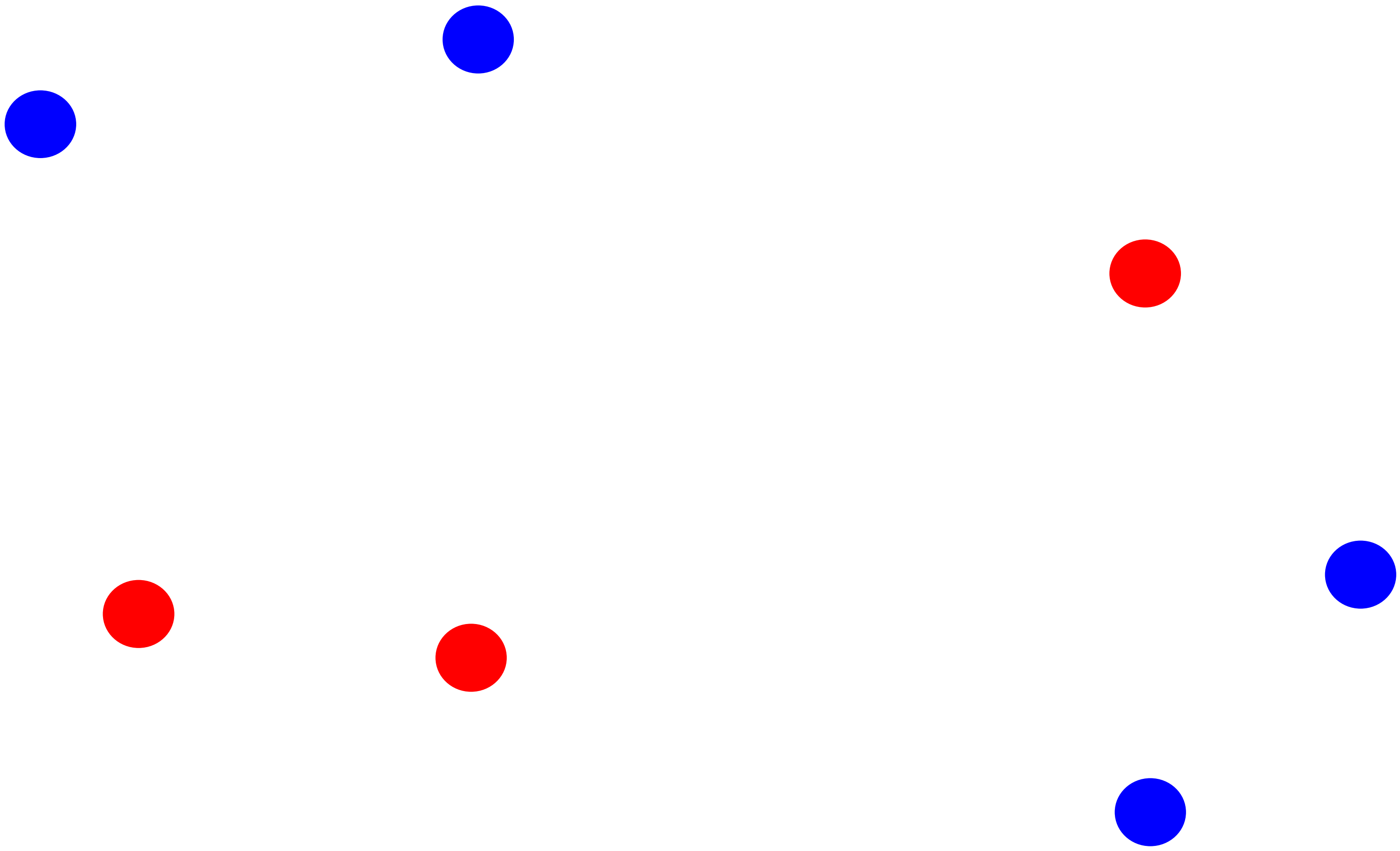


What is the data generating distribution?

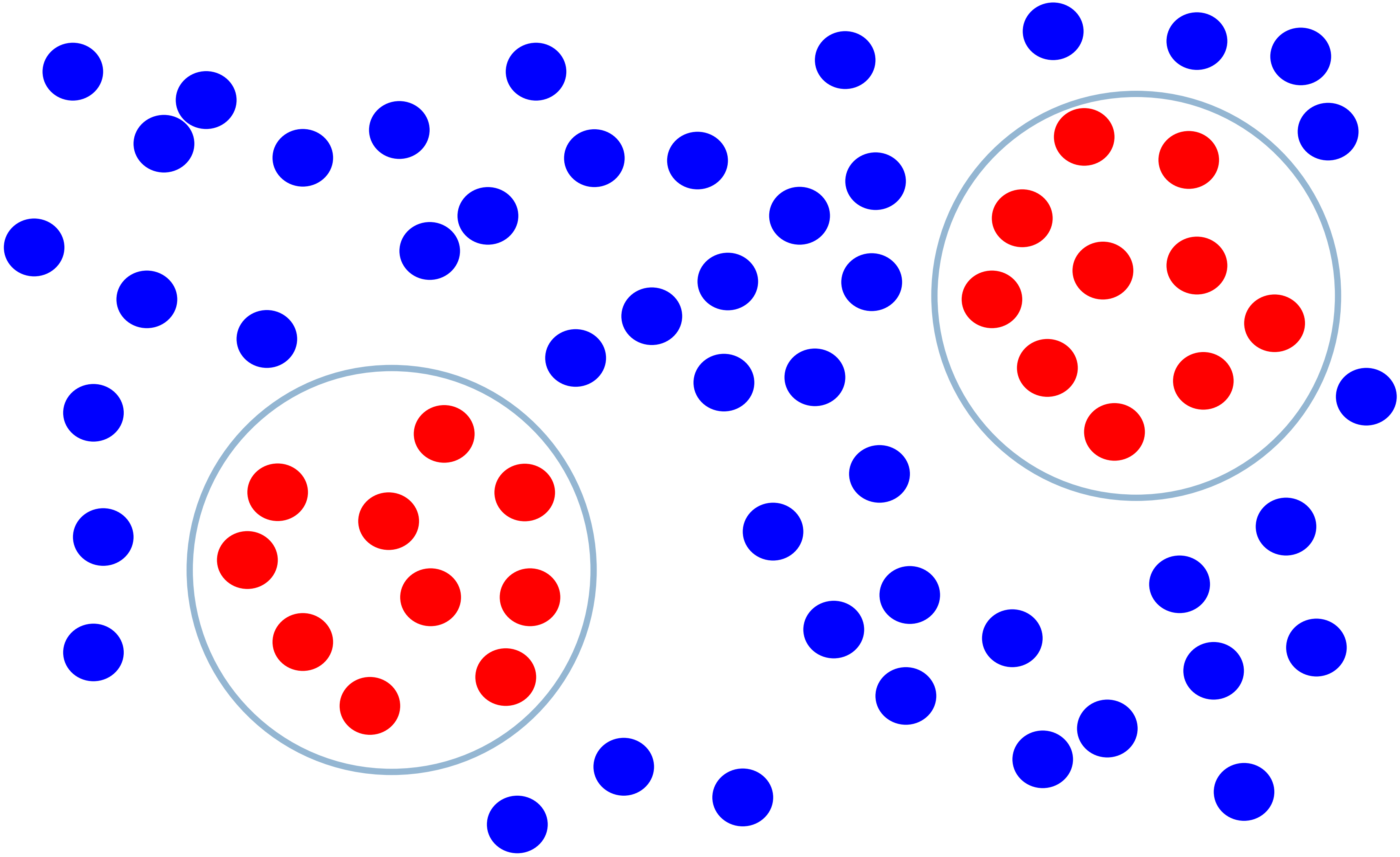


Knowing the model beforehand can drastically improve the learning and the number of examples required

What is the data generating distribution?



Make sure your assumption is correct, though!



Machine learning models

- What were the *model* assumptions (if any) that *k*-NN and decision trees make about the data?
  - KNN is an non parametric lazy learning algorithm.
  - That is a pretty concise statement.
  - When you say a technique is non parametric, it means that it does not make any assumptions on the underlying data distribution.

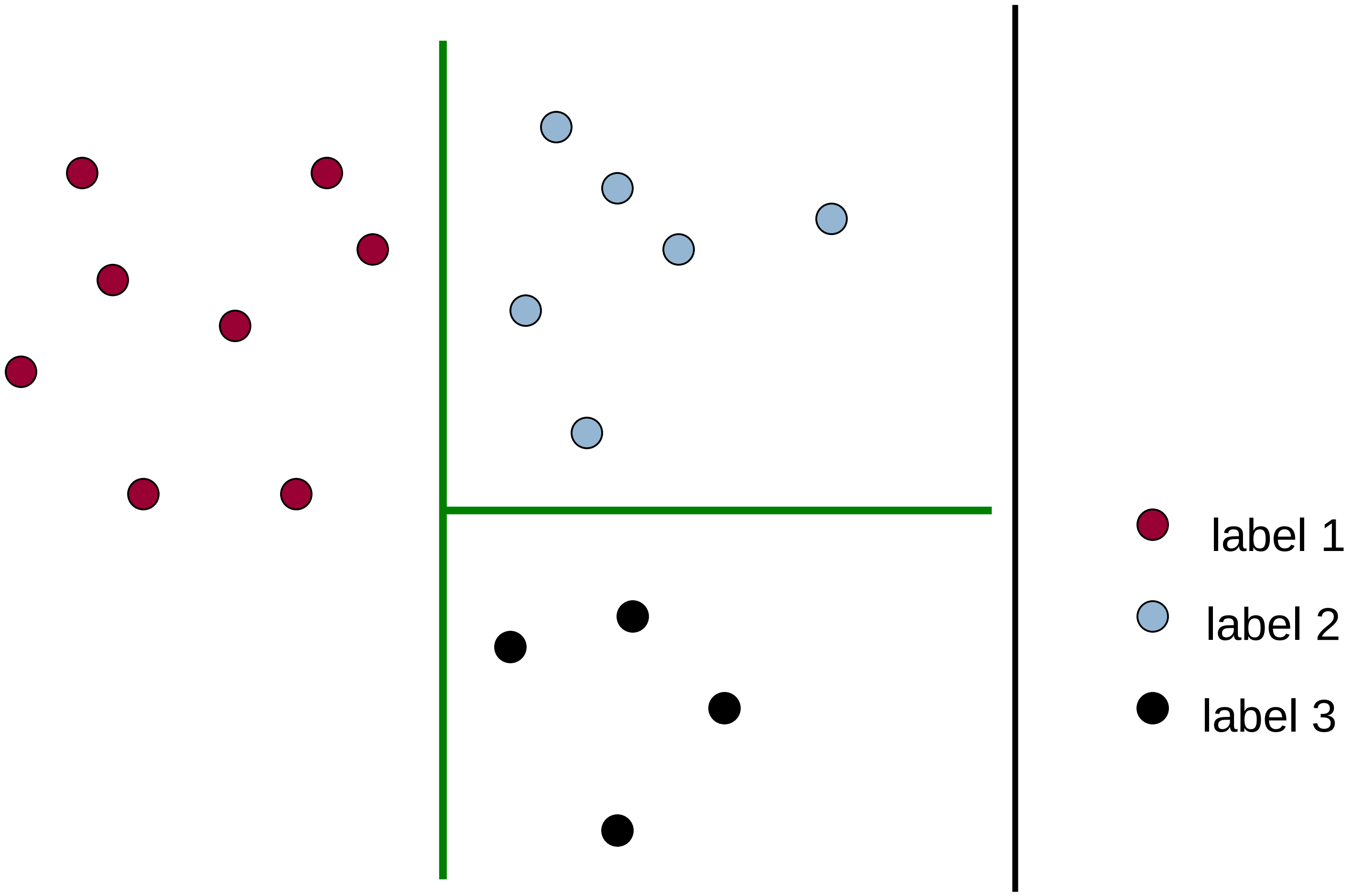
KNN-Pros

- The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms.
- There is no need to train a model for generalization, that is why KNN is known as the simple and instance-based learning algorithm.
- KNN can be useful in case of nonlinear data.
- It can be used with the regression problem.
- Output value for the object is computed by the average of k closest neighbors value.

KNN-Cons

- The testing phase of K-nearest neighbor classification is slower and costlier in terms of time and memory.
- It requires large memory for storing the entire training dataset for prediction.
- KNN requires scaling of data because KNN uses the Euclidean distance between two data points to find nearest neighbors.
- Euclidean distance is sensitive to magnitudes.
- The features with high magnitudes will weight more than features with low magnitudes.
- KNN also not suitable for large dimensional data.

Decision tree model



Axis-aligned splits/cuts of the data

Bias

The “bias” of a model is how strong the model assumptions are.

low-bias classifiers make minimal assumptions about the data (*k*-NN and DT are generally considered low bias

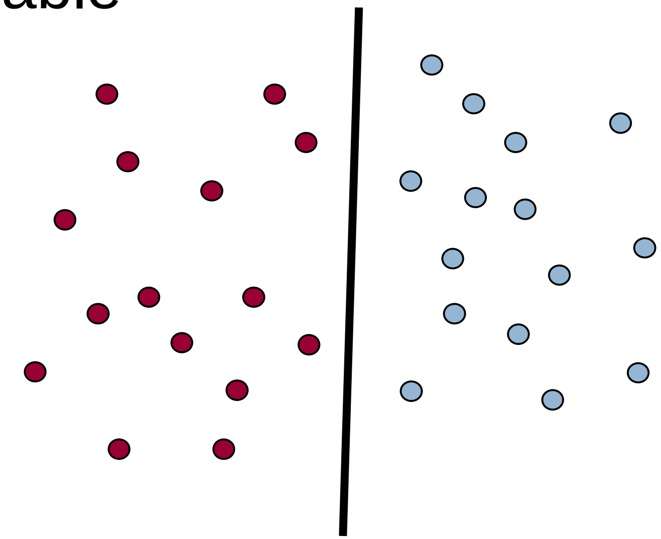
high-bias classifiers make strong assumptions about the data

Linear models

A strong high-bias assumption is *linear separability*:

- in 2 dimensions, can separate classes by a line
- in higher dimensions, need hyperplanes

A *linear model* is a model that assumes the data is linearly separable



Hyperplanes

A hyperplane is line/plane in a high dimensional space

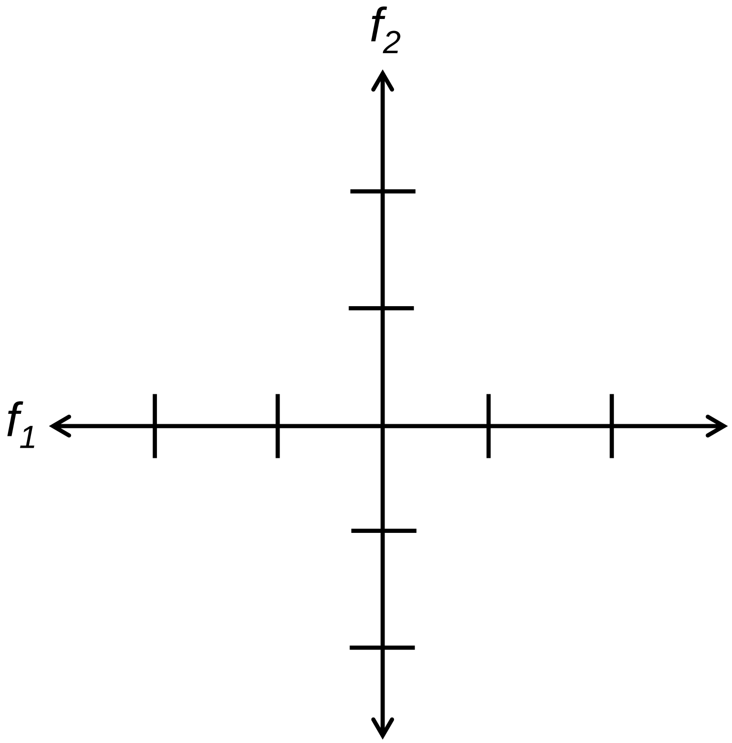
What defines a line?

What defines a hyperplane?

Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$



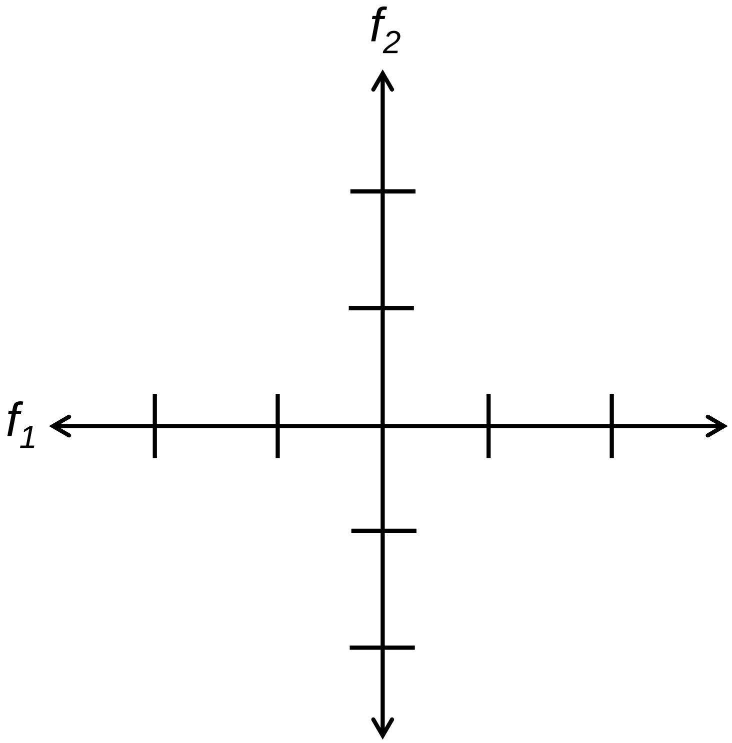
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

-2	1
-1	0.5
0	0
1	-0.5
2	-1



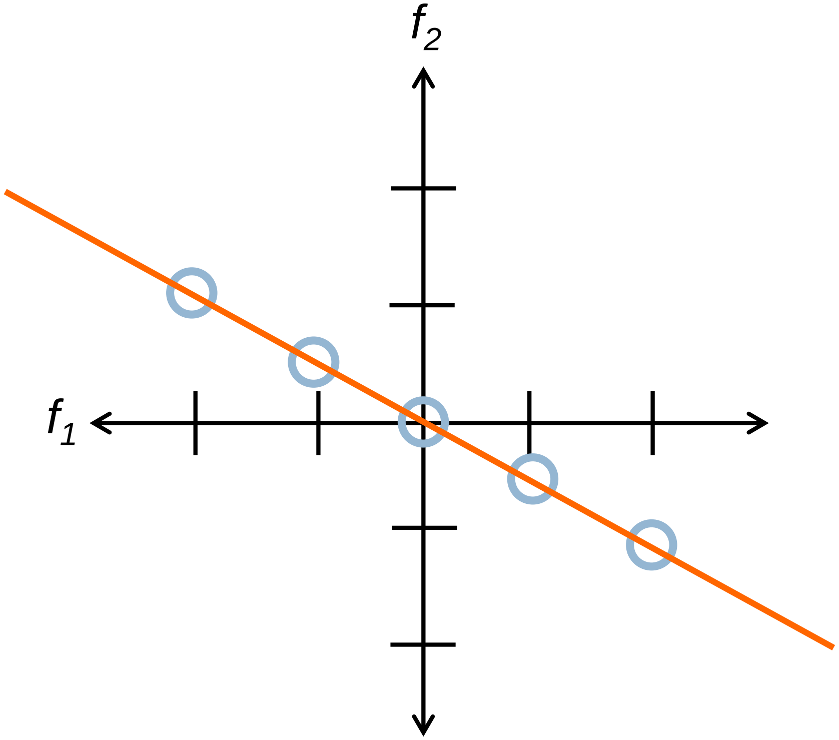
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

-2	1
-1	0.5
0	0
1	-0.5
2	-1



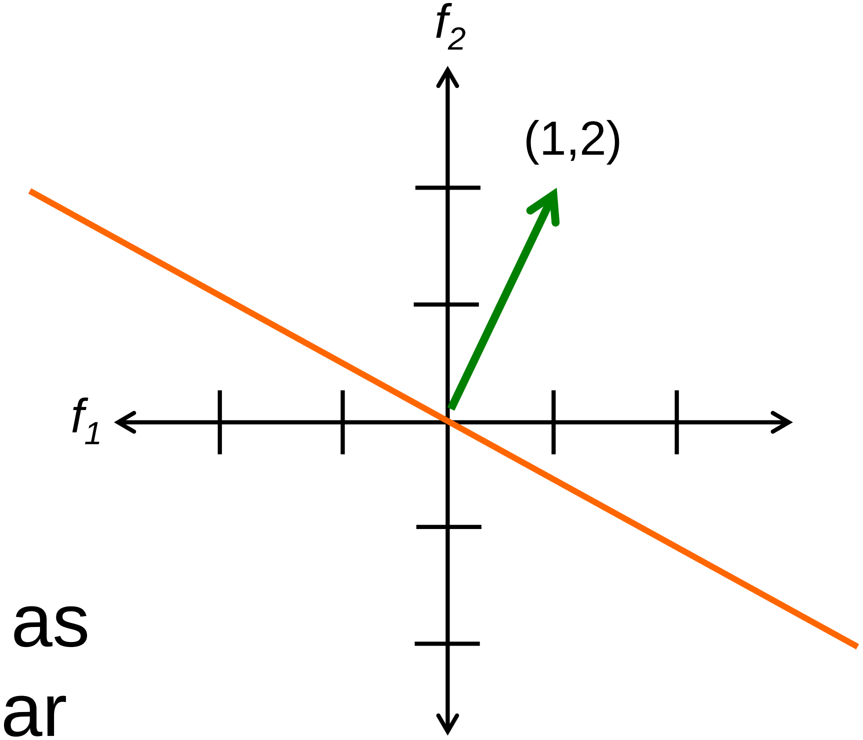
Defining a line

Any pair of values (*w_1*,*w_2*) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

w=(1,2)

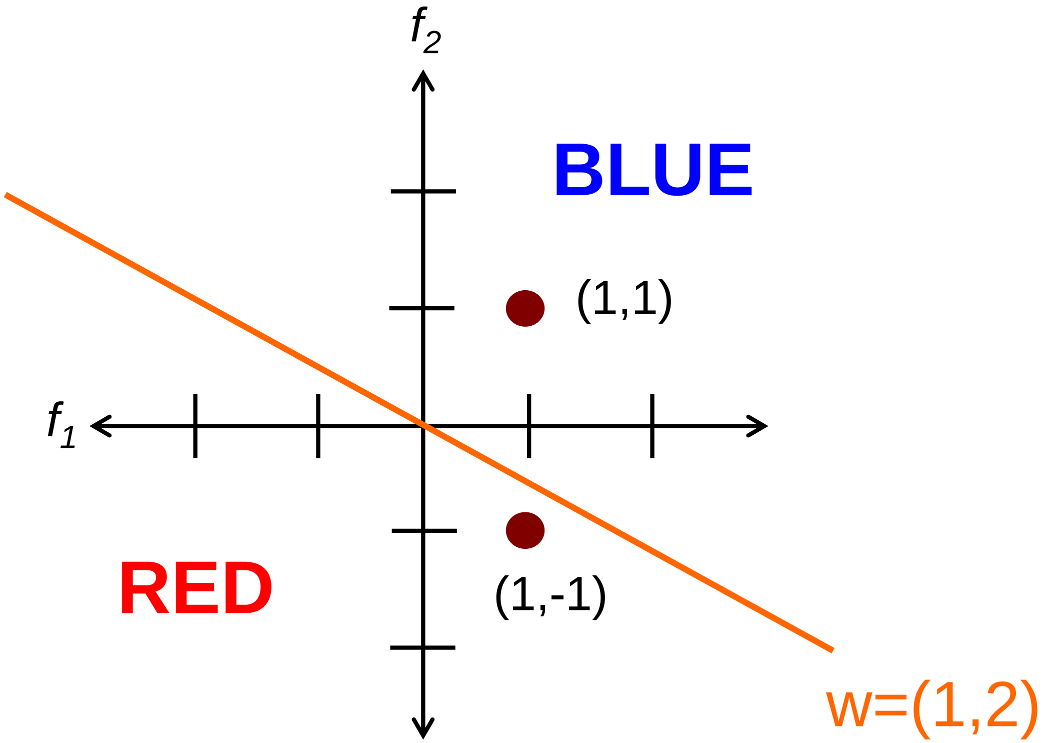


We can also view it as the line perpendicular to the *weight vector*

Classifying with a line

Mathematically, how can we classify points based on a line?

$$0 = 1f_1 + 2f_2$$



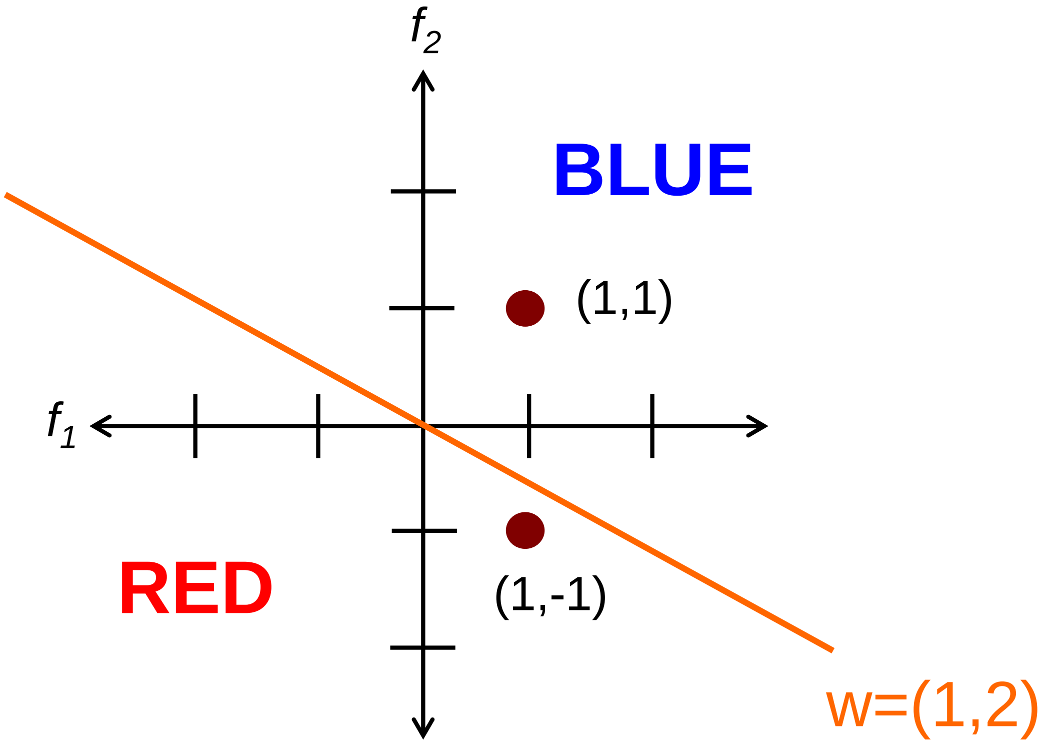
Classifying with a line

Mathematically, how can we classify points based on a line?

$$0 = 1f_1 + 2f_2$$

(1,1): $1*1+2*1=3$

(1,-1): $1*1+2*-1=-1$



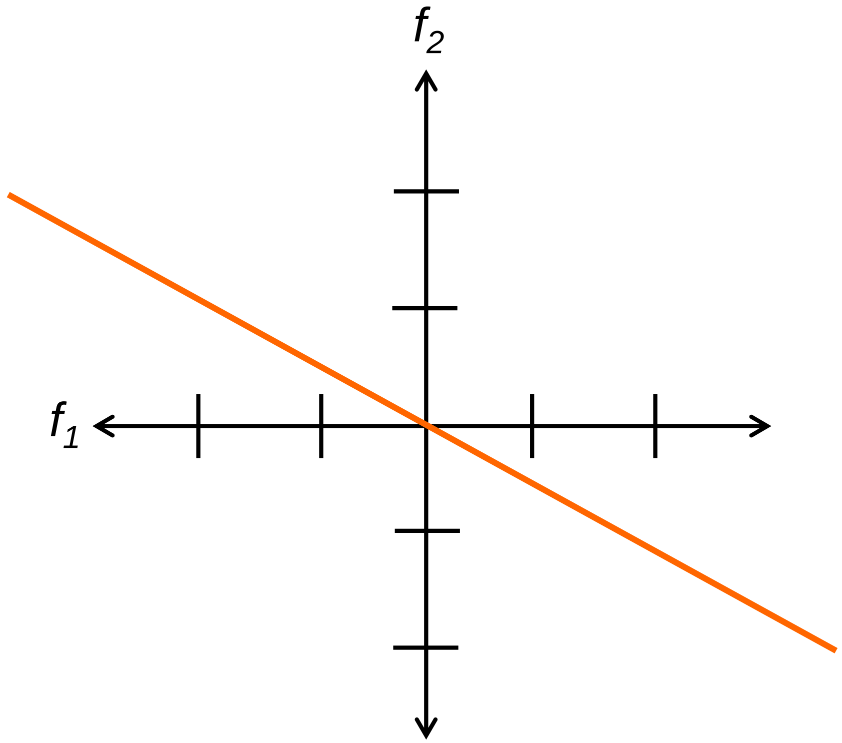
The sign indicates which side of the line

Defining a line

Any pair of values (*w_1*,*w_2*) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$



How do we move the line off of the origin?

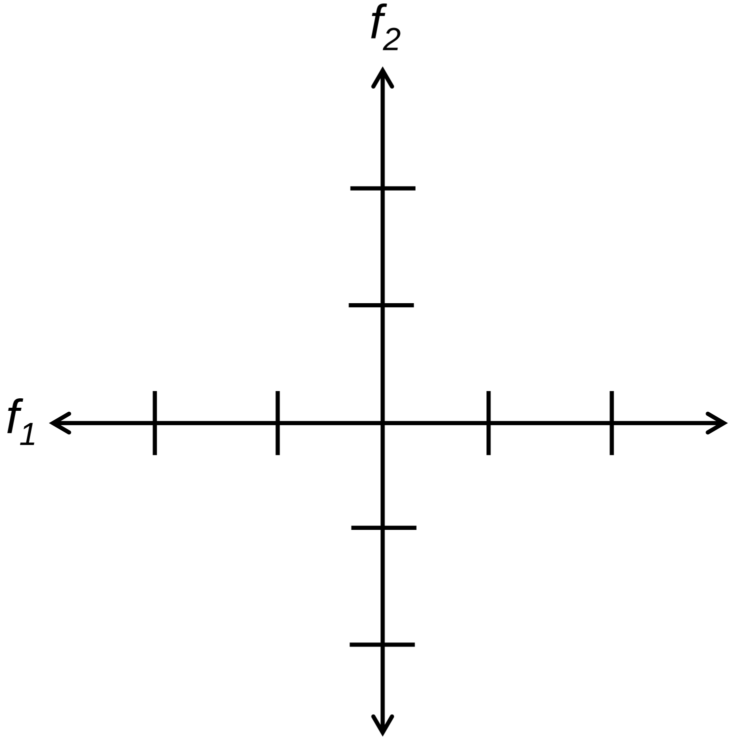
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

$$-1 = 1 f_1 + 2 f_2$$

-2
-1
0
1
2



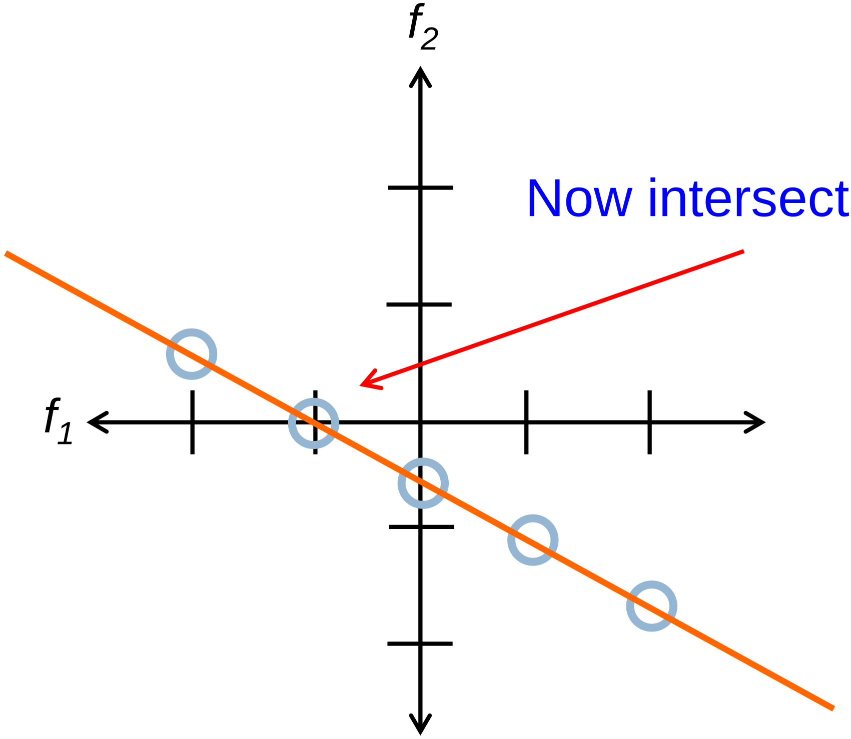
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

$$-1 = 1 f_1 + 2 f_2$$

-2	0.5
-1	0
0	-0.5
1	-1
2	-1.5



Now intersects at -1

Linear models

A linear model in *n*-dimensional space (i.e. *n* features) is define by *n*+1 weights:

In two dimensions, a line:

$$0 = w_1 f_1 + w_2 f_2 + b$$ (where b = -a)

In three dimensions, a plane:

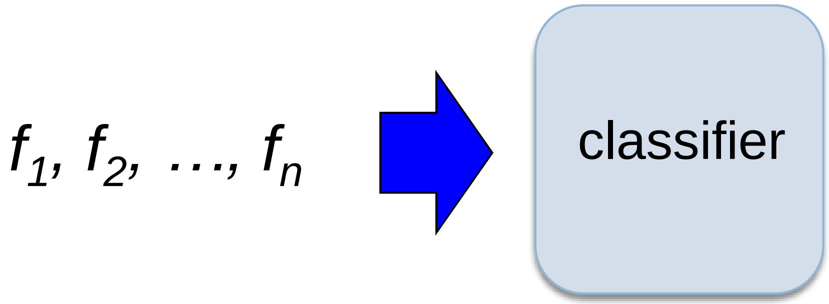
$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$$

In *n*-dimensions, a *hyperplane*

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

Classifying with a linear model

We can classify with a linear model by checking the sign:



$f_1, f_2, \ldots, f_n$

classifier

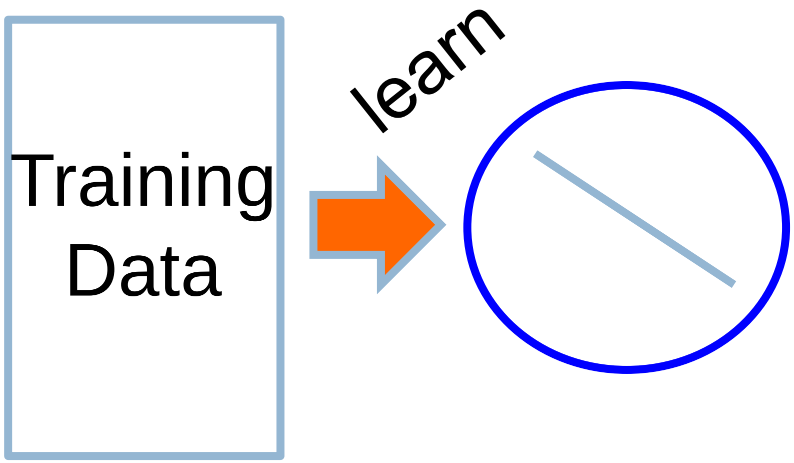
$$b + \sum_{i=1}^{n} w_i f_i > 0$$ Positive example

$$b + \sum_{i=1}^{n} w_i f_i < 0$$ Negative example

Learning a linear model

Geometrically, we know what a linear model represents

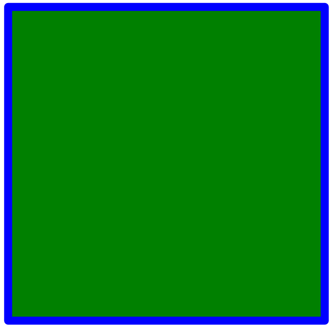
Given a linear model (i.e. a set of weights and b) we can classify examples



(data with labels)

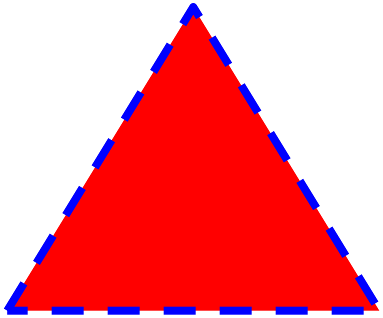
How do we learn a linear model?

Positive or negative?



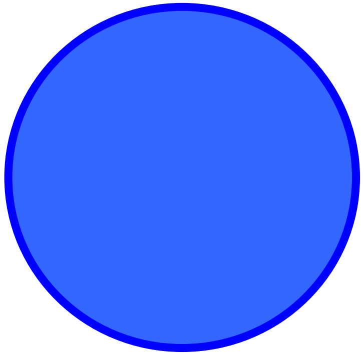
NEGATIVE

Positive or negative?



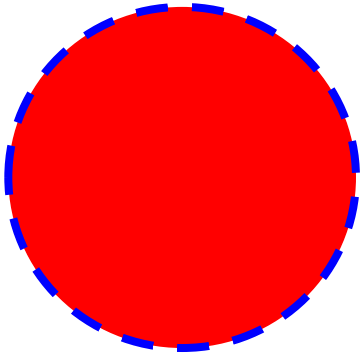
NEGATIVE

Positive or negative?



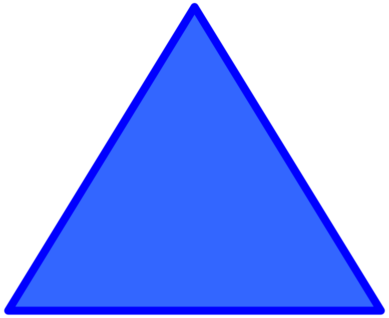
POSITIVE

Positive or negative?



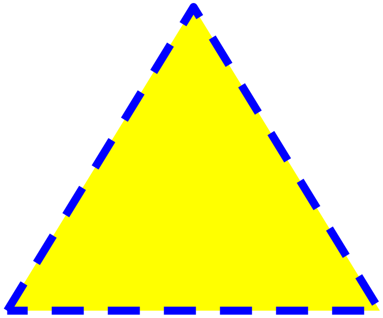
NEGATIVE

Positive or negative?



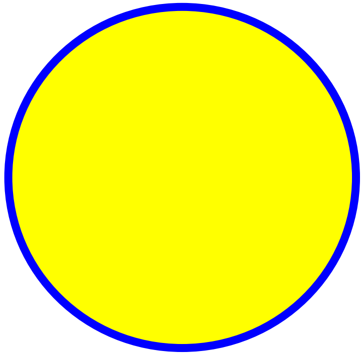
POSITIVE

Positive or negative?



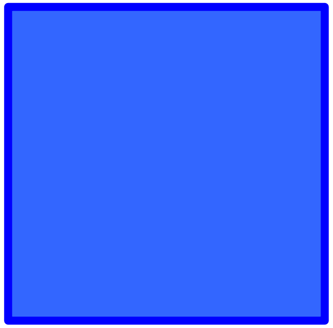
POSITIVE

Positive or negative?



NEGATIVE

Positive or negative?



POSITIVE

A method to the madness

blue = positive

yellow triangles = positive

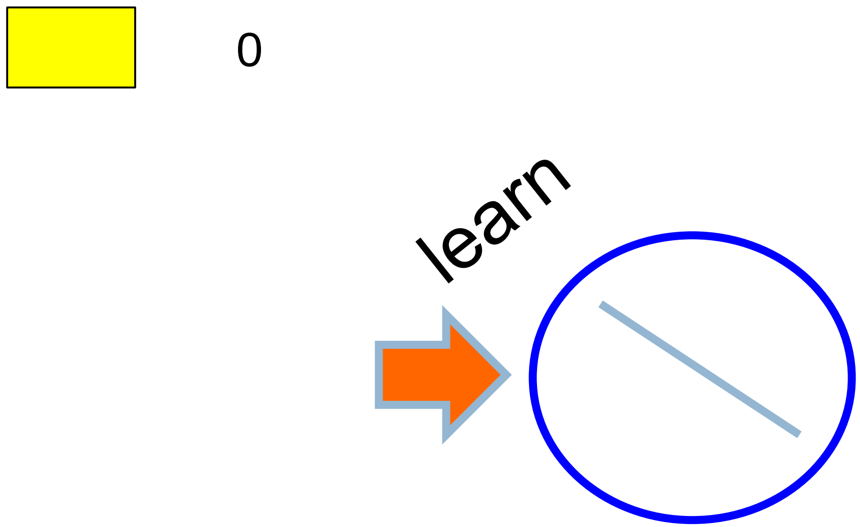
all others negative

How is this learning setup different than the learning we've done before?

When might this arise?

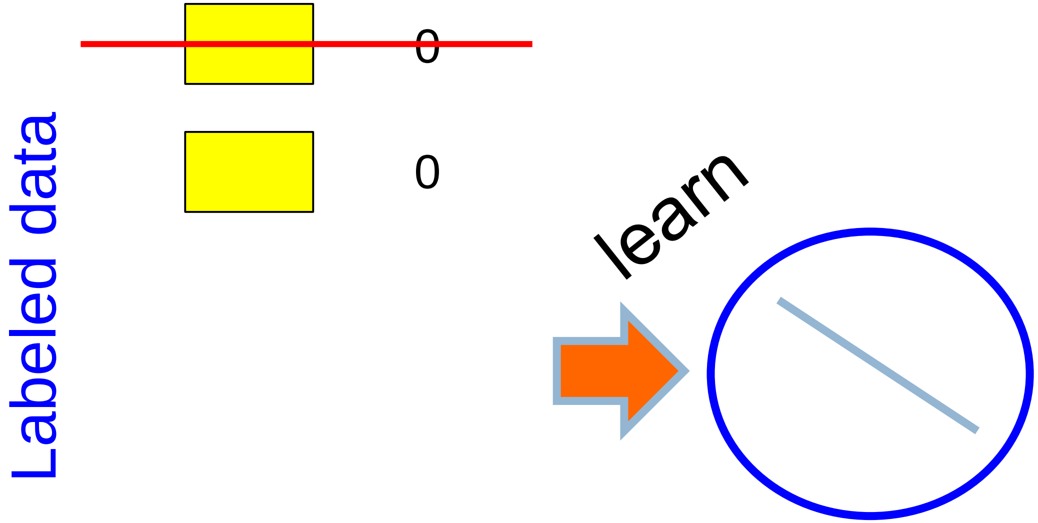
Online learning algorithm

Labeled data



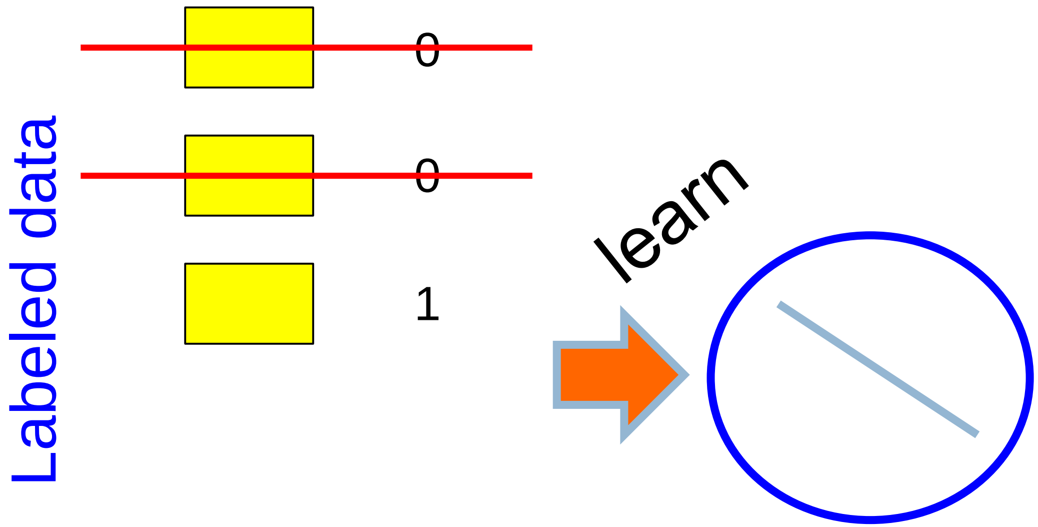
Only get to see one example at a time!

Online learning algorithm



Only get to see one example at a time!

Online learning algorithm



Labeled data

~~0~~

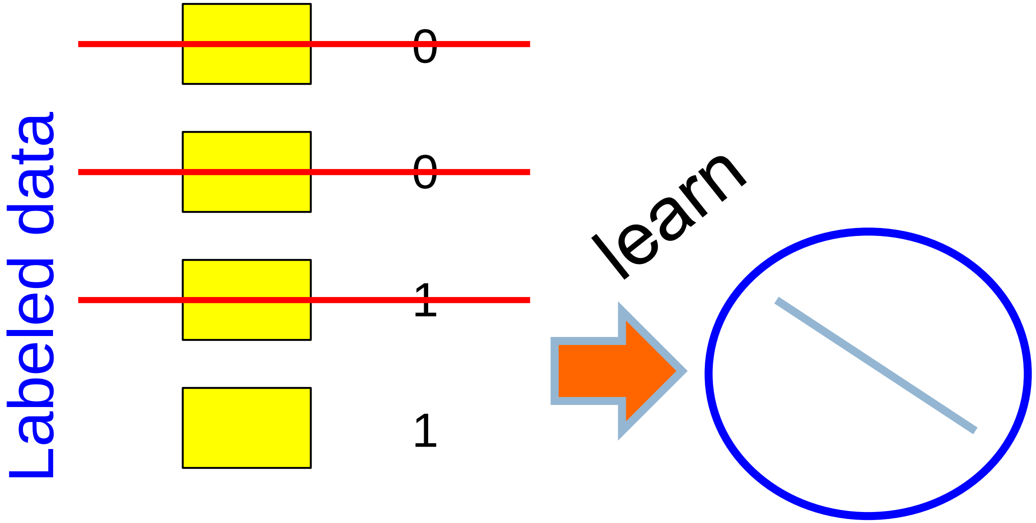
~~0~~

1

learn

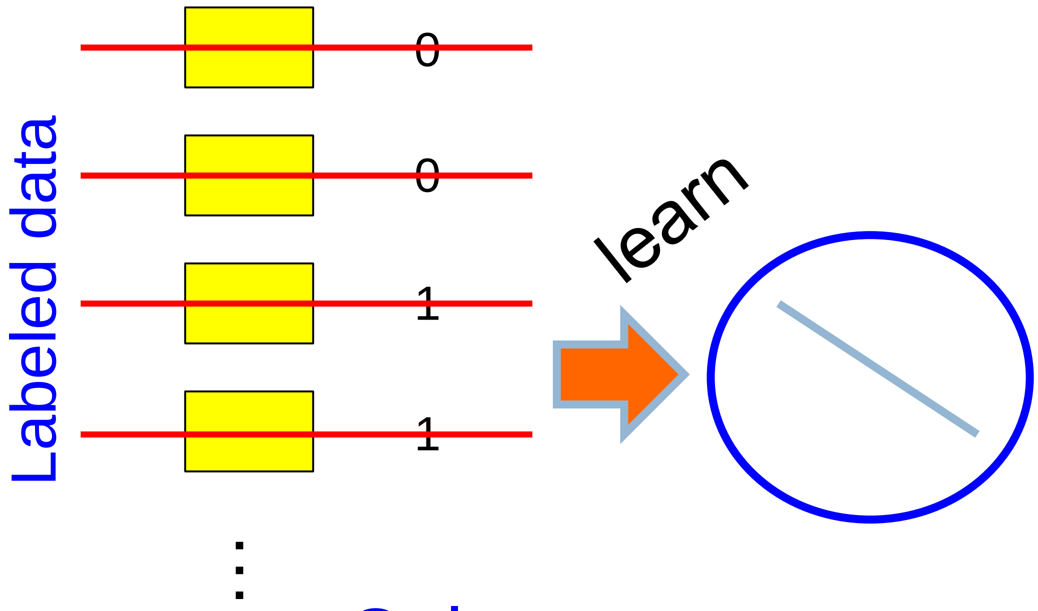
Only get to see one example at a time!

Online learning algorithm



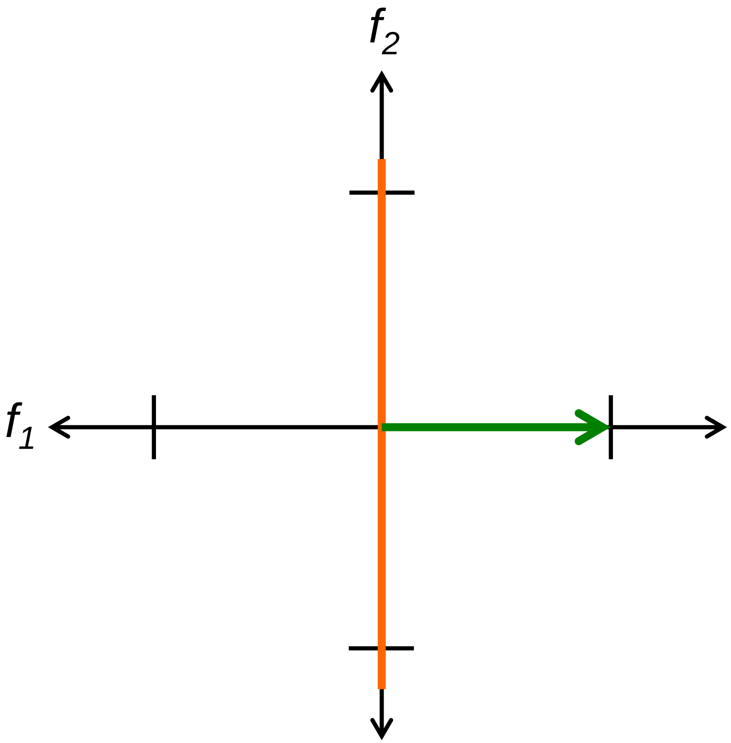
Only get to see one example at a time!

Online learning algorithm



Only get to see one example at a time!

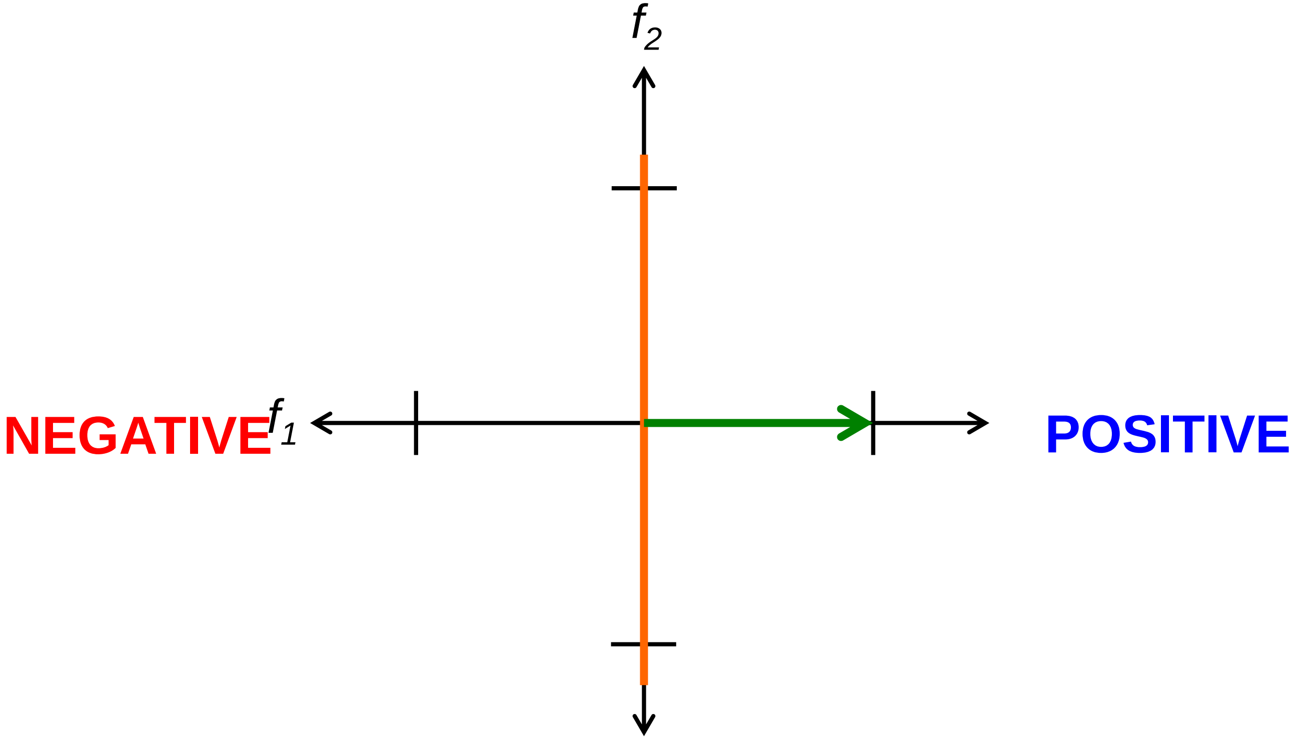
Learning a linear classifier



What does this model currently say?

w=(1,0)

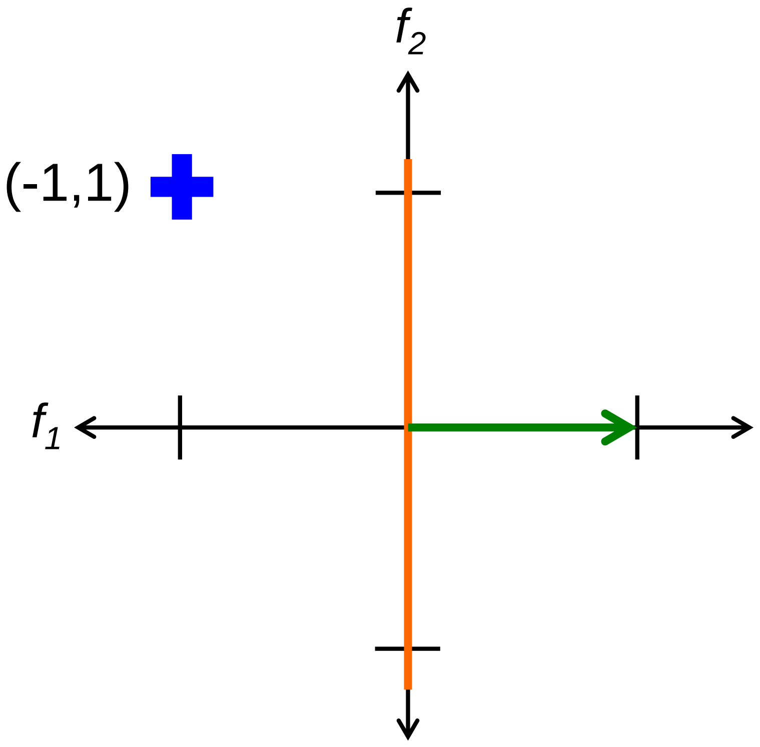
Learning a linear classifier



$w=(1,0)$

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess: right or wrong?

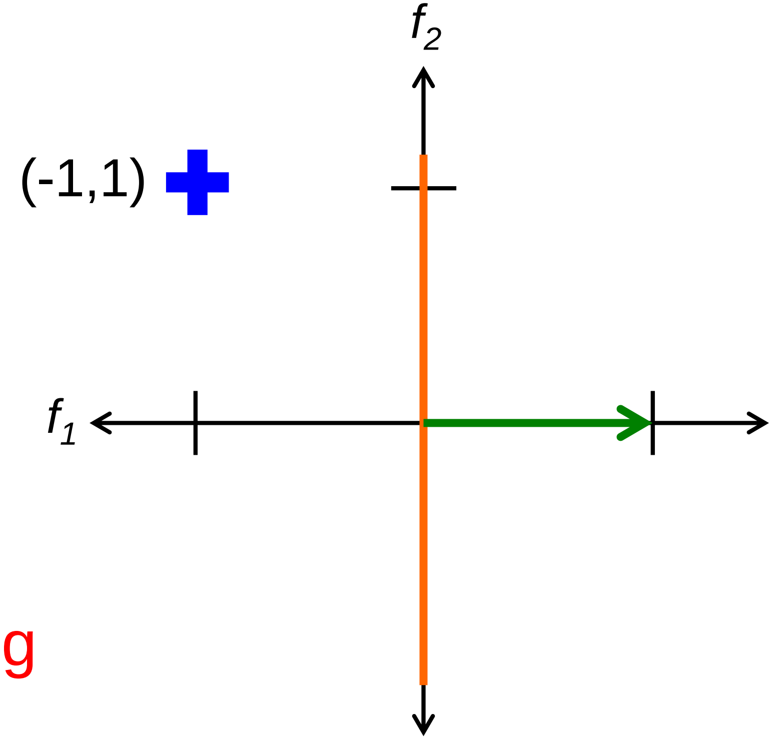
w=(1,0)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$



predicts negative, wrong

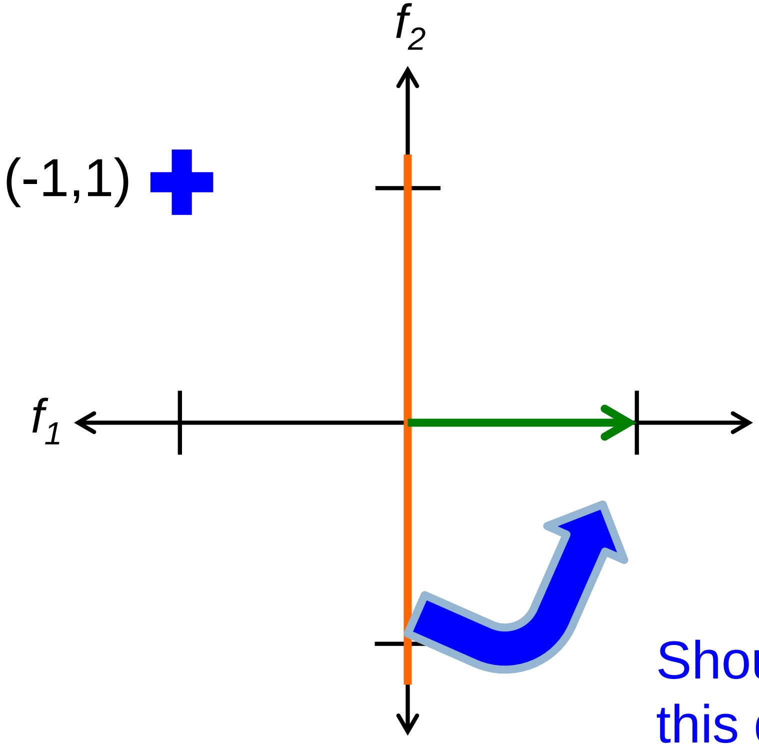
How should we update the model? w=(1,0)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$



Should move this direction

w=(1,0)

A closer look at why we got it wrong

(-1, 1, positive)

$$\overset{w_1}{1} * f_1 + \overset{w_2}{0} * f_2 =$$

$$\underbrace{1 * -1 + 0 * 1}_{} = -1 \leftarrow$$

We'd like this value to be positive since it's a positive value

Which of these contributed to the mistake?

A closer look at why we got it wrong

w_1 w_2

(-1, 1, positive)

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$

We'd like this value to be positive since it's a positive value

contributed in the wrong direction

could have contributed (positive feature), but didn't

How should we change the weights?

A closer look at why we got it wrong

w_1 w_2

(-1, 1, positive)

$$1*f_1+0*f_2=$$

$$1*-1+0*1=-1$$

We'd like this value to be positive since it's a positive value

contributed in the wrong direction

decrease

1 -> 0

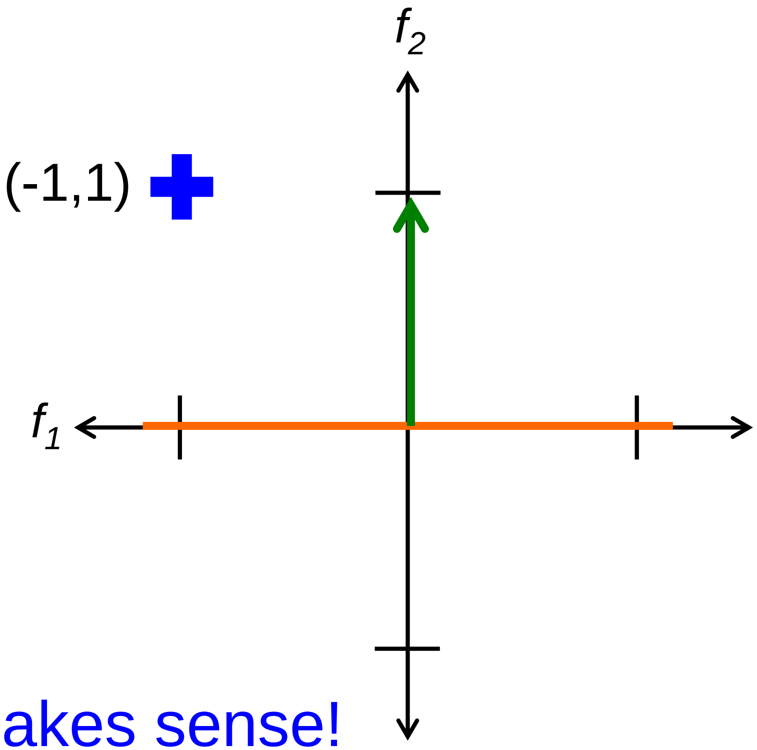
could have contributed (positive feature), but didn't

increase

0 -> 1

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

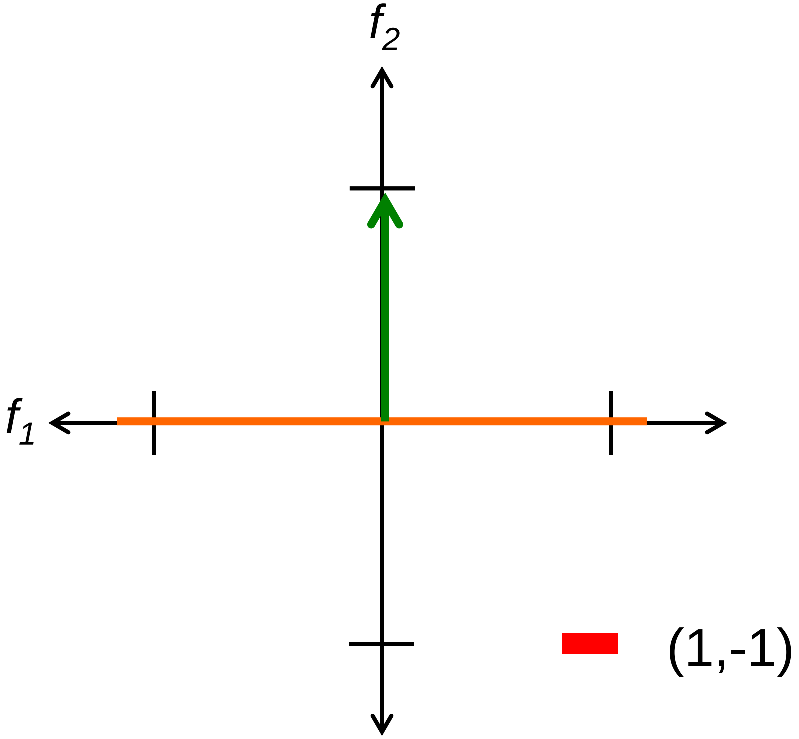


Graphically, this also makes sense!

w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess: right or wrong?

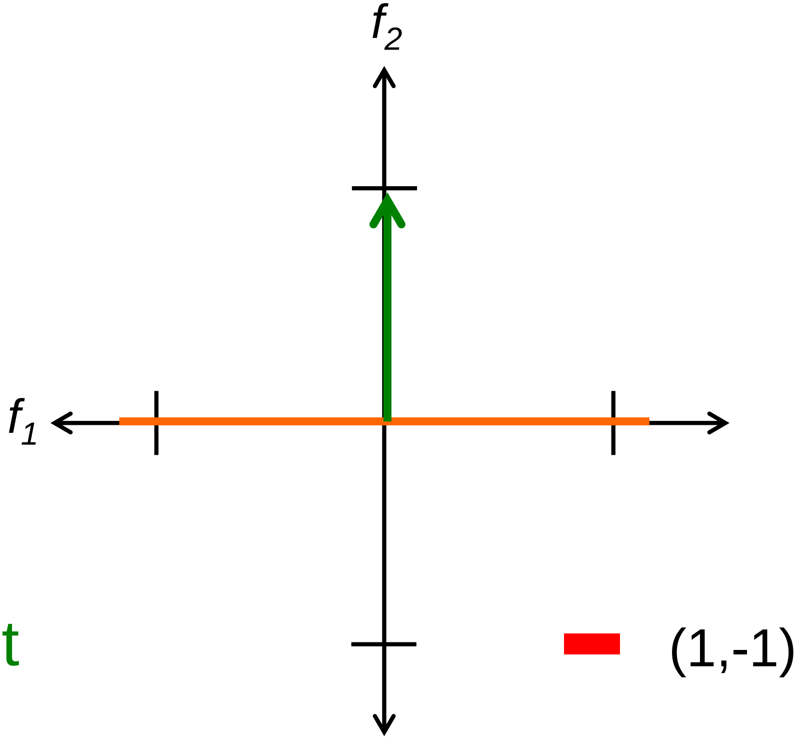
w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0*1+1*-1 = \boxed{-1}$$



predicts negative, correct

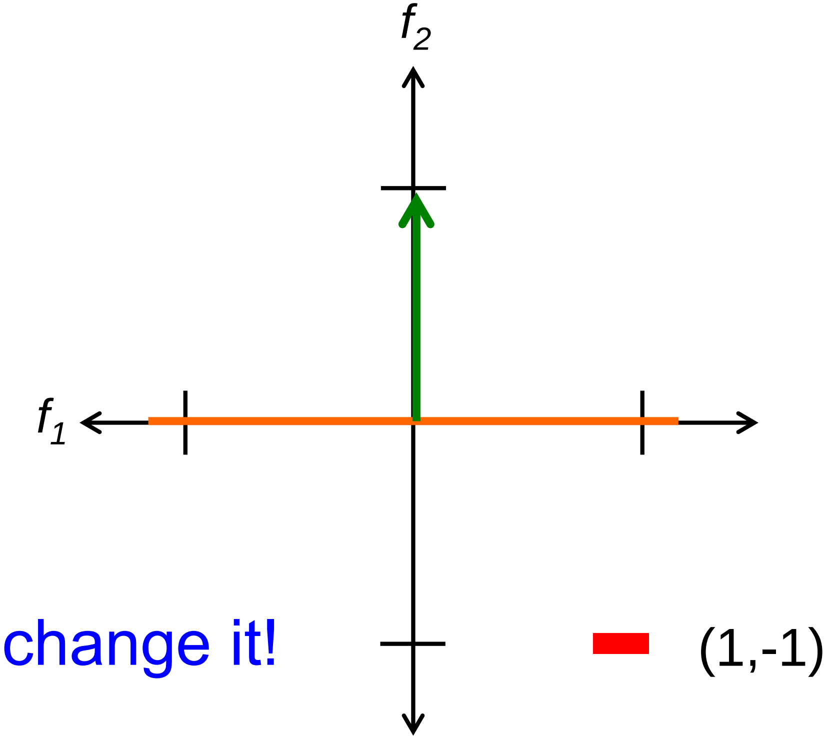
How should we update the model? w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0 * 1 + 1 * -1 = -1$$

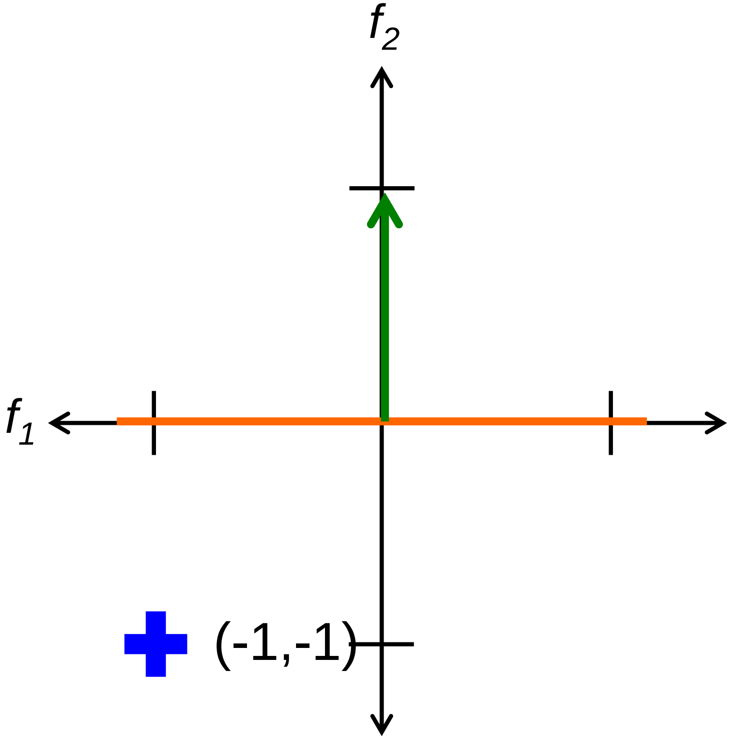


Already correct… don't change it!

w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess: right or wrong?

w=(0,1)

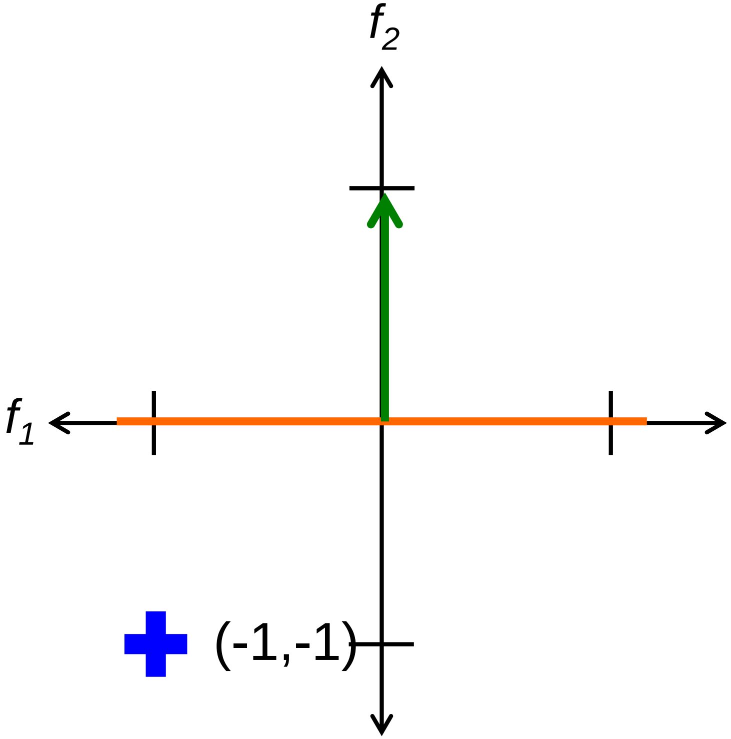
Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

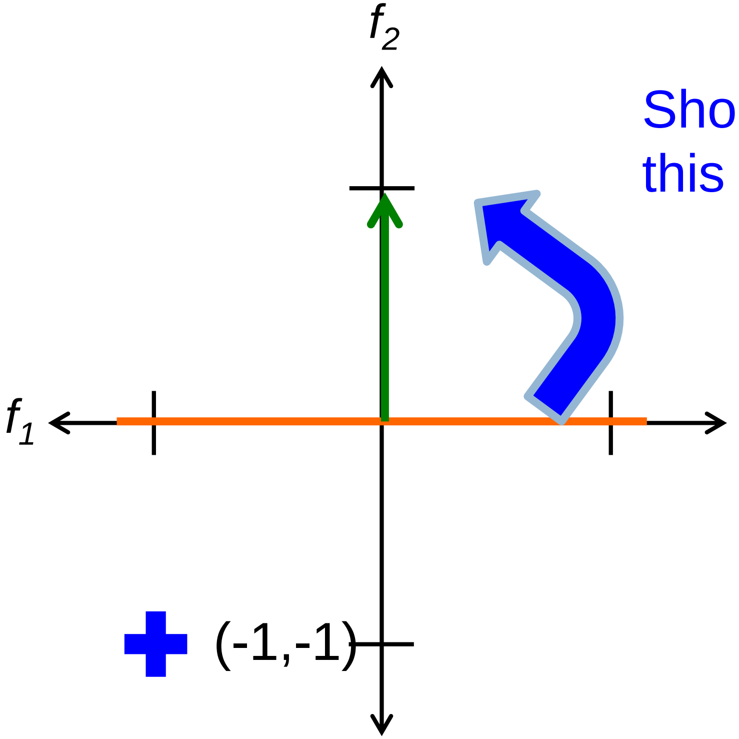
predicts negative, wrong



How should we update the model? w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Should move this direction

w=(0,1)

A closer look at why we got it wrong

w_1 w_2

(-1, -1, positive)

$$0*f_1+1*f_2=$$

$$\underbrace{0*-1+1*-1}=-1 \leftarrow$$

We’d like this value to be positive since it’s a positive value

Which of these contributed to the mistake?

A closer look at why we got it wrong

w_1 w_2

(-1, -1, positive)

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

contributed in the wrong direction

How should we change the weights?

A closer look at why we got it wrong

w_1 w_2

(-1, -1, positive)

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

contributed in the wrong direction

decrease

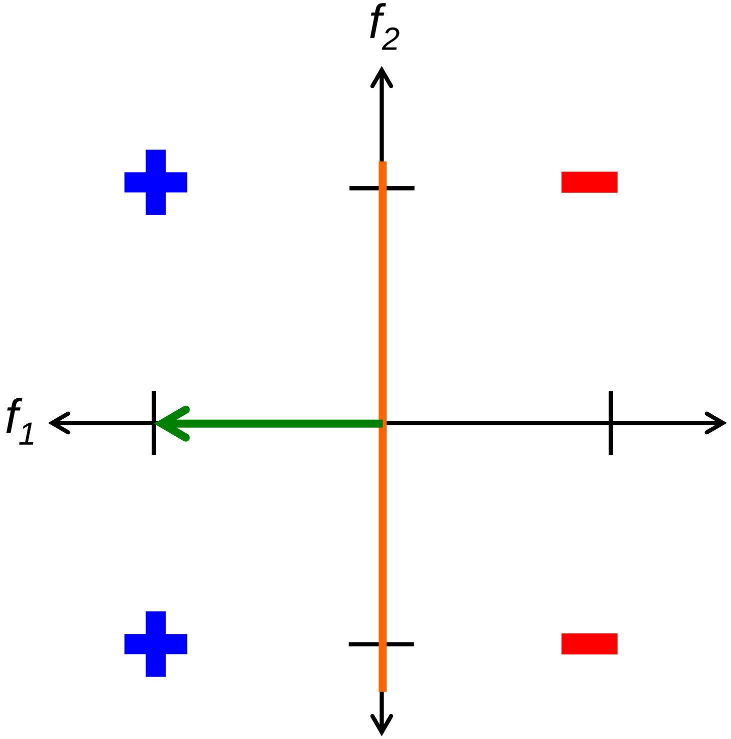
0 -> -1

decrease

1 -> 0

Learning a linear classifier

f_1	*f_2*	*label*
-1	-1	positive
-1	1	positive
1	1	negative
1	-1	negative



w=(-1,0)

Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    check if it's correct based on the current model

    if not correct, update all the weights:          $w_i f_i$
      if label positive and feature positive:
        increase weight (increase weight = predict more positive)
      if label positive and feature negative:
        decrease weight (decrease weight = predict more positive)
      if label negative and feature positive:
        decrease weight (decrease weight = predict more negative)
      if label negative and negative weight:
        increase weight (increase weight = predict more negative)
```

A trick…

Let positive label = 1 and negative label = -1

if label positive and feature positive:

increase weight (increase weight = predict more positive)

if label positive and feature negative:

decrease weight (decrease weight = predict more positive)

if label negative and feature positive:

decrease weight (decrease weight = predict more negative)

if label negative and negative weight:

increase weight (increase weight = predict more negative)

label * f_i

1*1=1

1*-1=-1

-1*1=-1

-1*-1=1

A trick…

Let positive label = 1 and negative label = -1

if label positive and feature positive:

increase weight (increase weight = predict more positive)

if label positive and feature negative:

decrease weight (decrease weight = predict more positive)

if label negative and feature positive:

decrease weight (decrease weight = predict more negative)

if label negative and negative weight:

increase weight (increase weight = predict more negative)

label * f_i

1*1=1

1*-1=-1

-1*1=-1

-1*-1=1

Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    check if it’s correct based on the current model

    if not correct, update all the weights:
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

How do we check if it’s correct?

Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
      $prediction = b + \sum_{i=1}^{n} w_i f_i$
    if $prediction$ * $label \le 0$:  // they don’t agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Perceptron learning algorithm

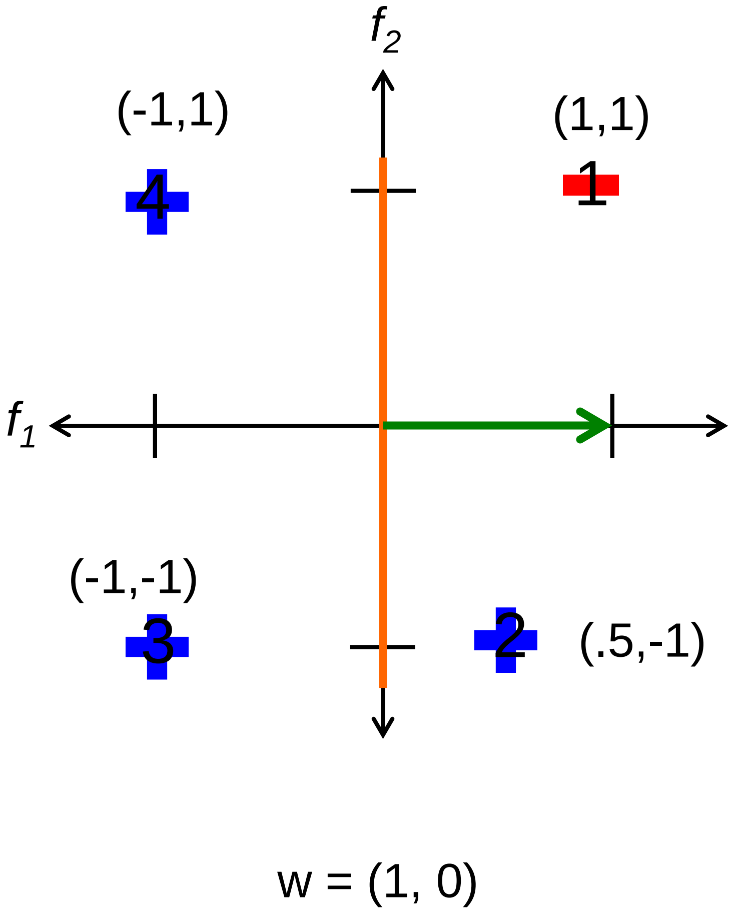
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1, f_2, \ldots, f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$
    if $prediction * label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Would this work for non-binary features, i.e. real-valued?

Your turn ☺

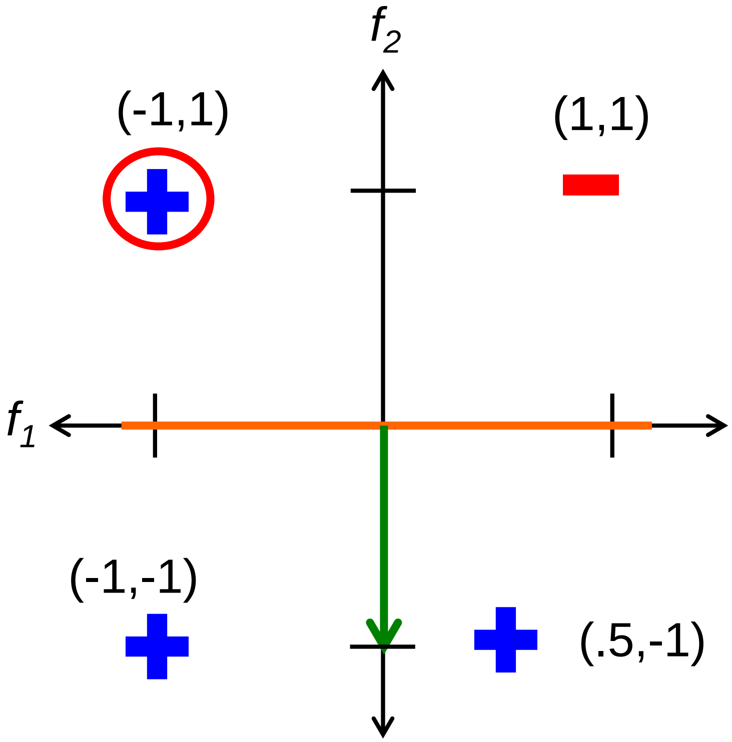
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1, f_2, \ldots, f_n$, label):
    $prediction = \text{å}_{i=1}^{n} w_i f_i$
    if *prediction* * *label* ≤ 0:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
```

- Repeat until convergence
- Keep track of w_1, w_2 as they change
- Redraw the line after each step



Your turn ☺

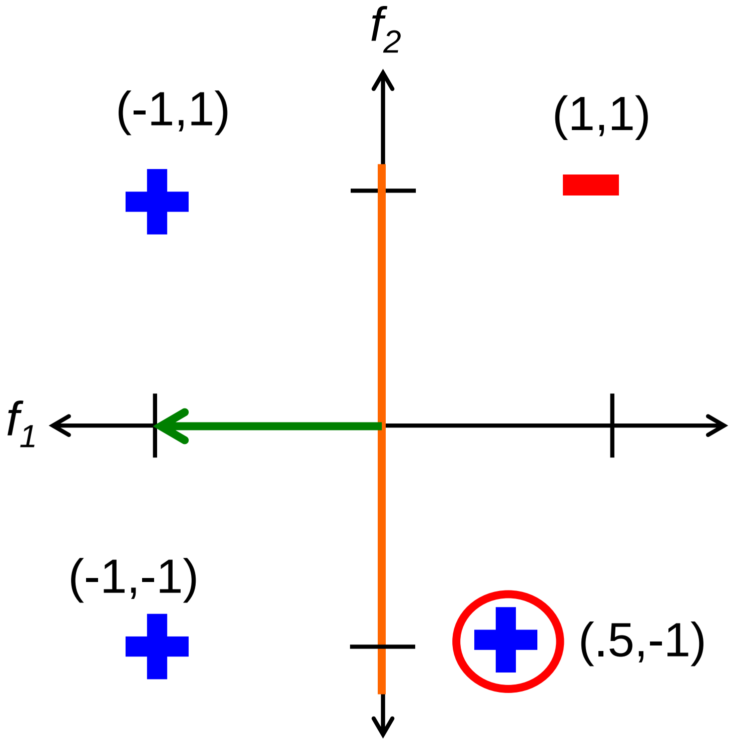
```
repeat until convergence (or for some # of iterations):
   for each training example ($f_1$, $f_2$, …, $f_n$, label):

      $prediction = \text{å}_{i=1}^{n} w_i f_i$

    if $prediction * label \leq 0$:  // they don't agree
           for each $w_i$:
             $w_i = w_i + f_i$*label
```



w = (0, -1)

Your turn ☺

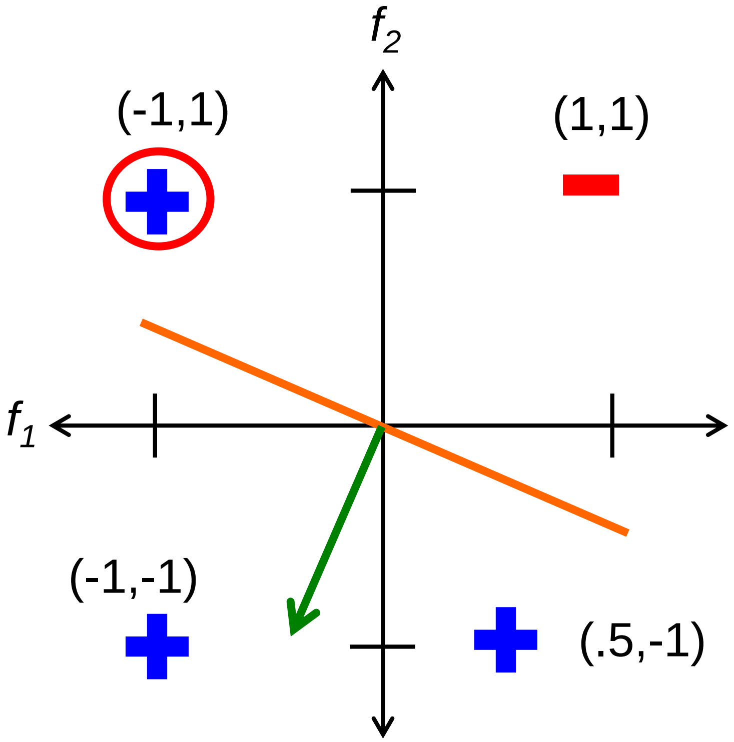
```
repeat until convergence (or for some # of iterations):
   for each training example ($f_1, f_2, \ldots, f_n$, label):

      $prediction = \sum_{i=1}^{n} w_i f_i$

      if $prediction * label \leq 0$:  // they don't agree
            for each $w_i$:
               $w_i = w_i + f_i$*label
```



w = (-1, 0)

Your turn ☺

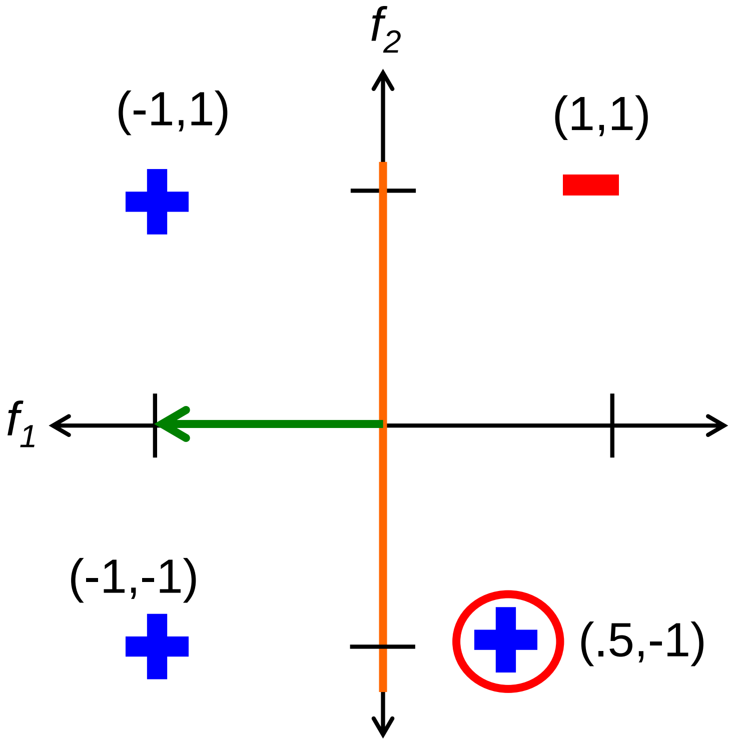
```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):
        $prediction = \sum_{i=1}^{n} w_i f_i$
        if $prediction * label \le 0$:  // they don't agree
            for each $w_i$:
                $w_i = w_i + f_i$*label
```



w = (-.5, -1)

Your turn ☺

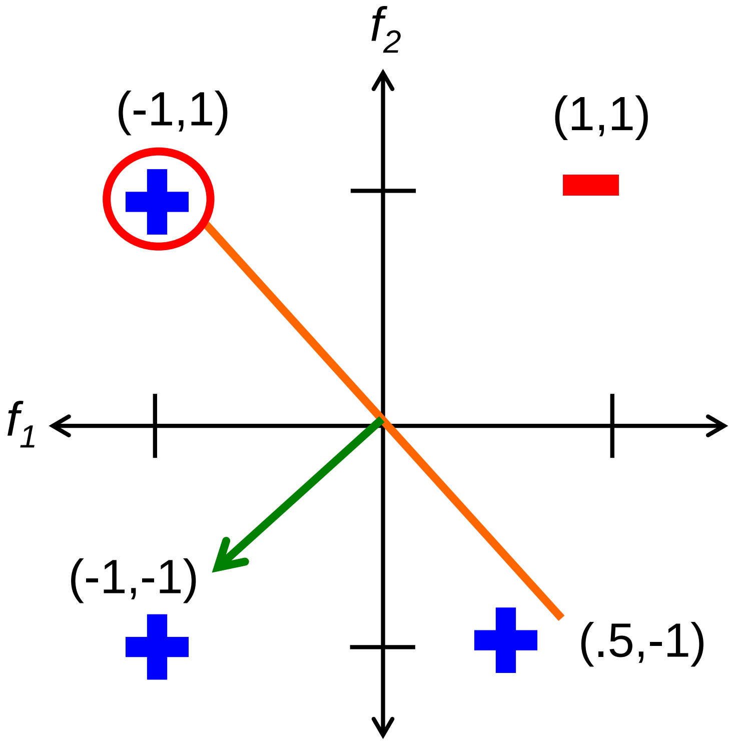
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):

    $prediction = å^{n}_{i=1} w_i f_i$

    if $prediction$ * $label \leq 0$:  // they don't agree
        for each $w_i$:
          $w_i = w_i + f_i$*label
```



w = (-1.5, 0)

Your turn ☺

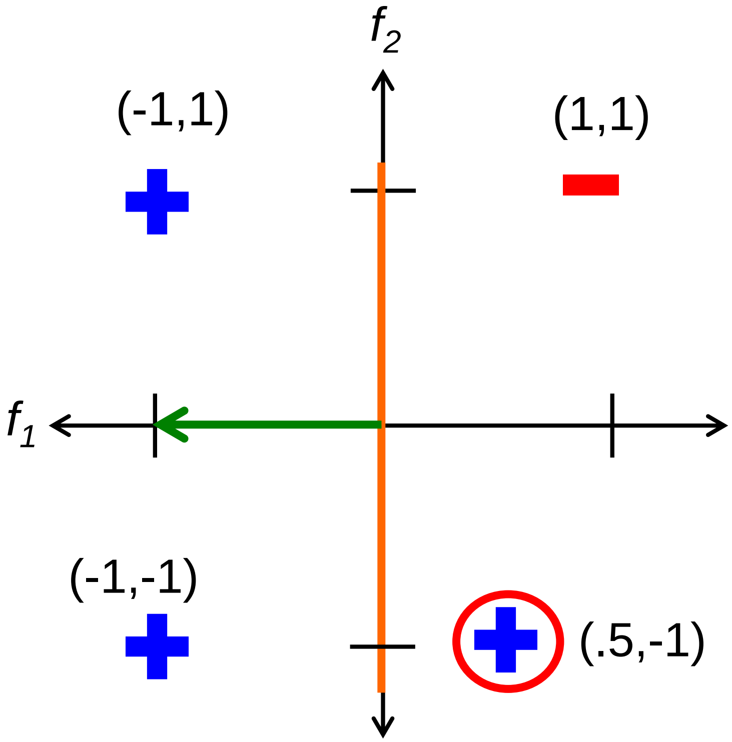
```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):
        $prediction = \sum_{i=1}^{n} w_i f_i$
        if $prediction$ * $label \le 0$:  // they don't agree
            for each $w_i$:
                $w_i = w_i + f_i$*label
```



w = (-1, -1)

Your turn ☺

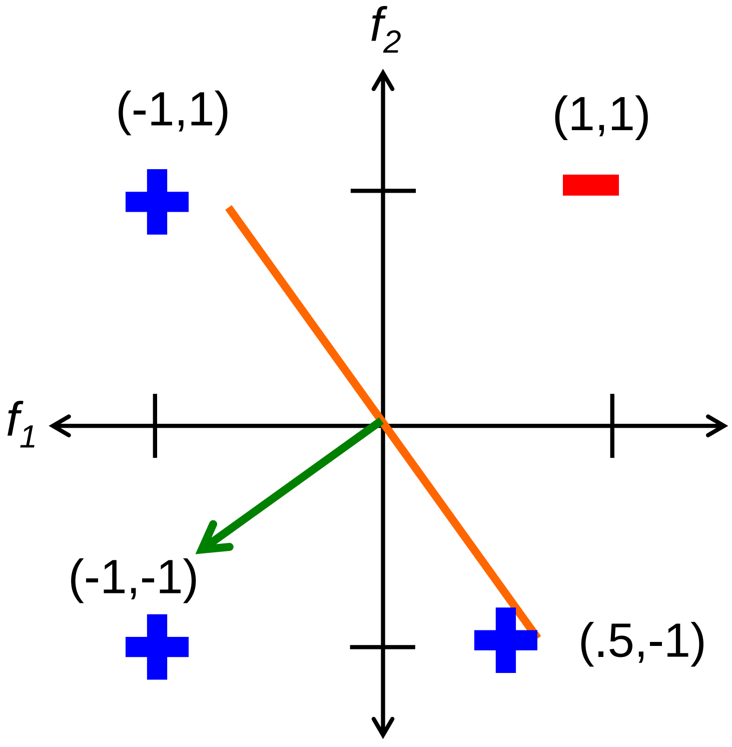
```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):
        $prediction = \sum_{i=1}^{n} w_i f_i$
        if prediction * label ≤ 0:  // they don't agree
            for each $w_i$:
                $w_i = w_i + f_i$*label
```



w = (-2, 0)

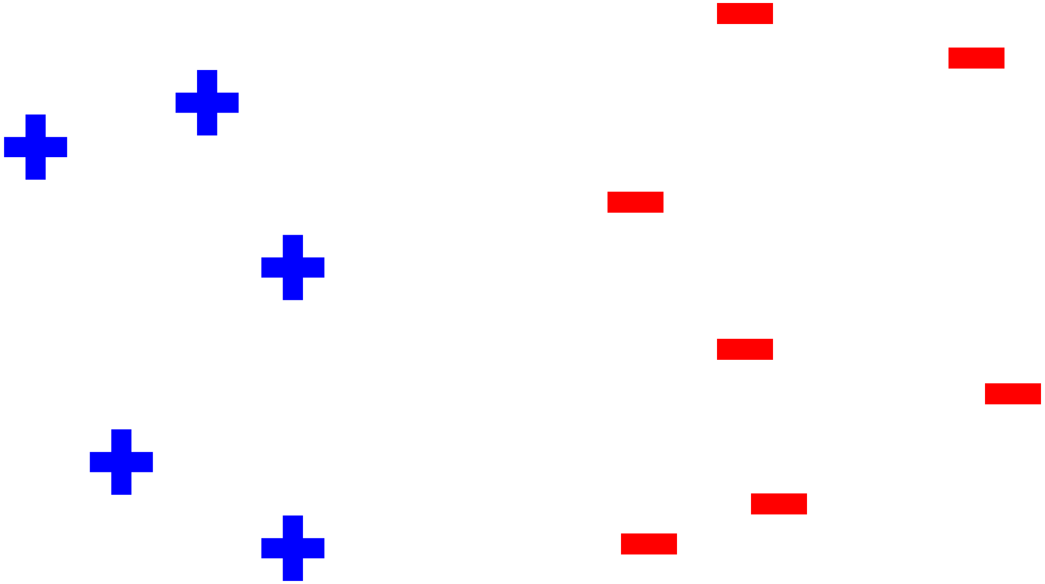
Your turn ☺

```
repeat until convergence (or for some # of iterations):
    for each training example ($f_1$, $f_2$, …, $f_n$, label):

        $prediction = \text{å}_{i=1}^{n} w_i f_i$

        if $prediction$ * $label \leq 0$:  // they don't agree
            for each $w_i$:
                $w_i = w_i + f_i$*label
```

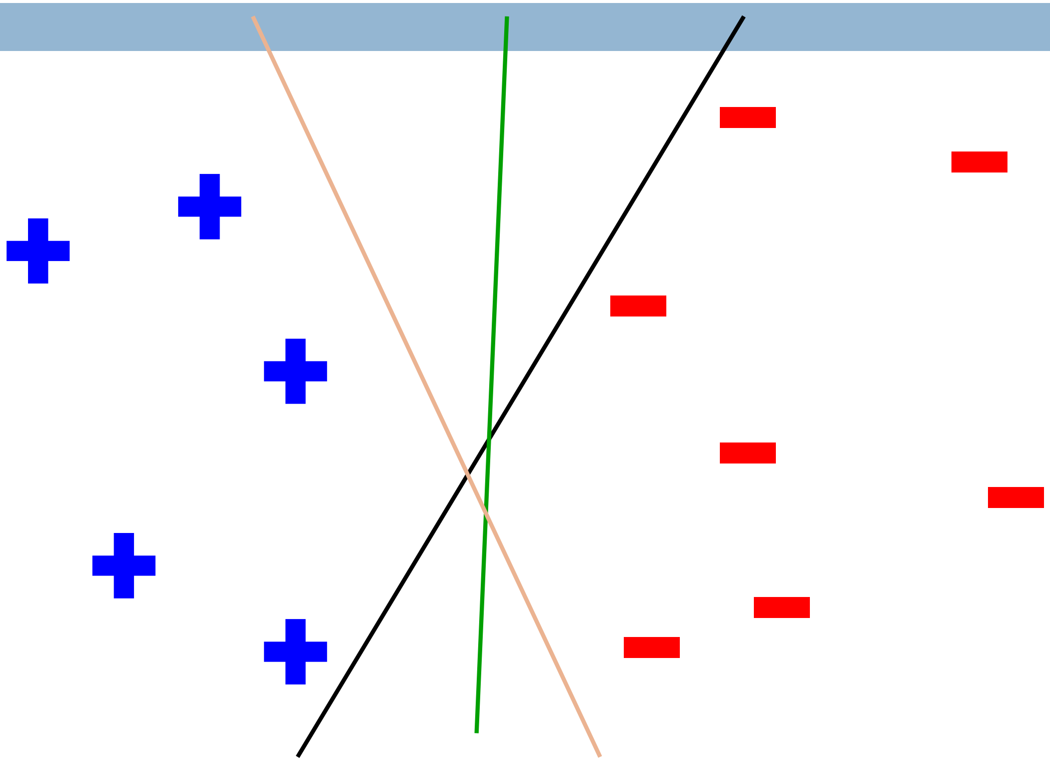


w = (-1.5, -1)

Which line will it find?



Which line will it find?



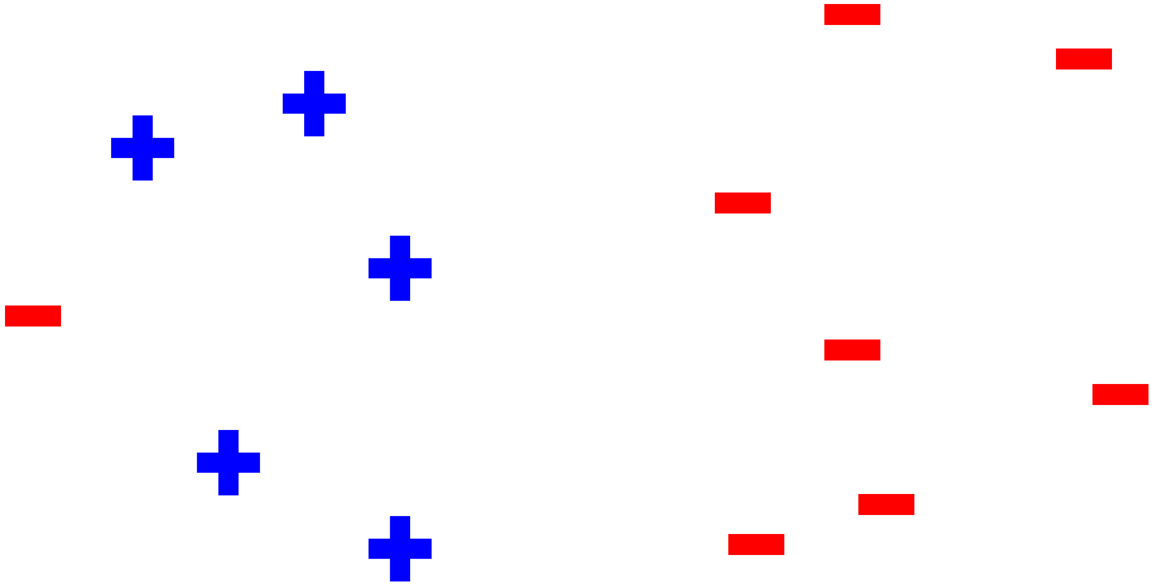
Only guaranteed to find ***some*** line that separates the data

Convergence

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1, f_2, \ldots, f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction * label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Why do we also have the "some # iterations" check?

Handling non-separable data



If we ran the algorithm on this it would never converge!

Convergence

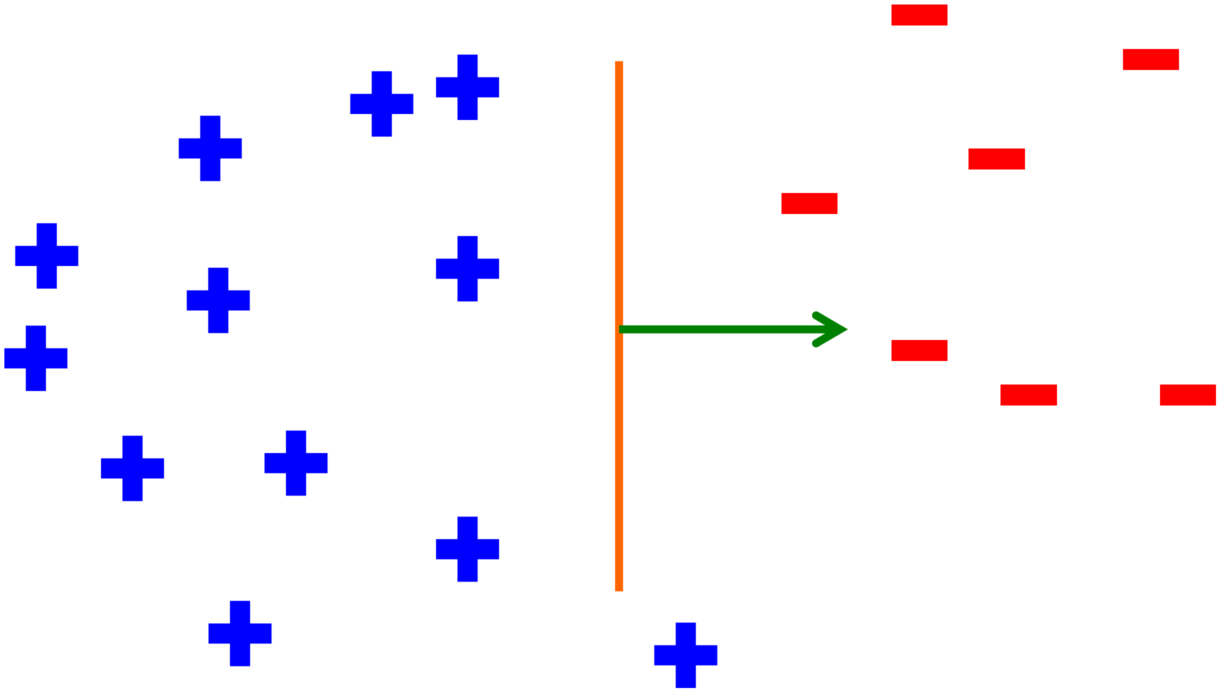
```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1, f_2, …, f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction * label \le 0$:  // they don’t agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Ordering

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1, f_2, \ldots, f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction * label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i * \text{label}$
      $b = b + \text{label}$
```

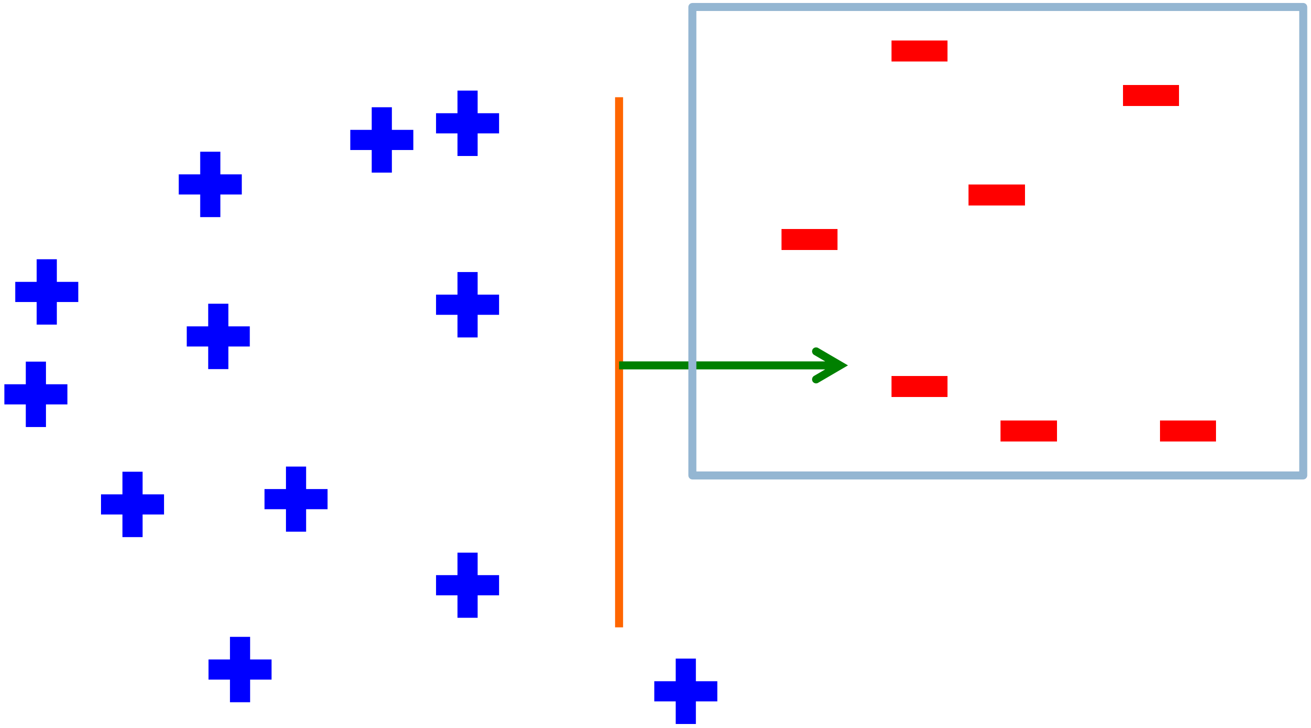
What order should we traverse the examples? Does it matter?

Order matters

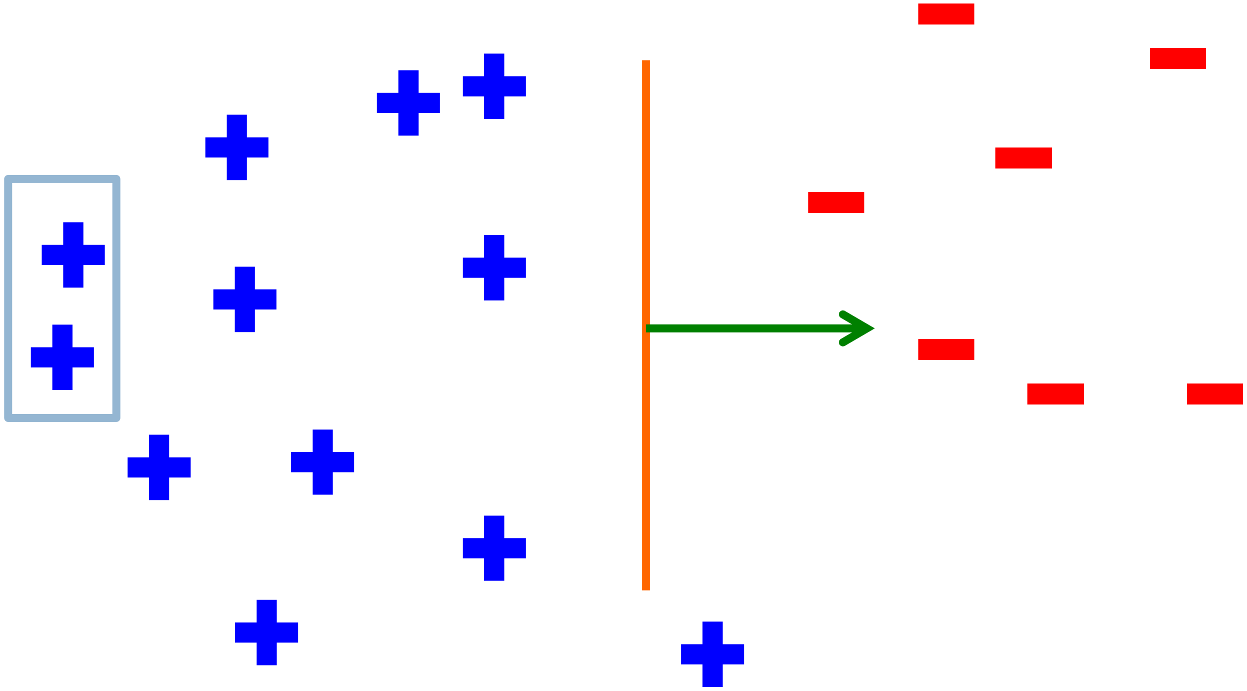


What would be a good/bad order?

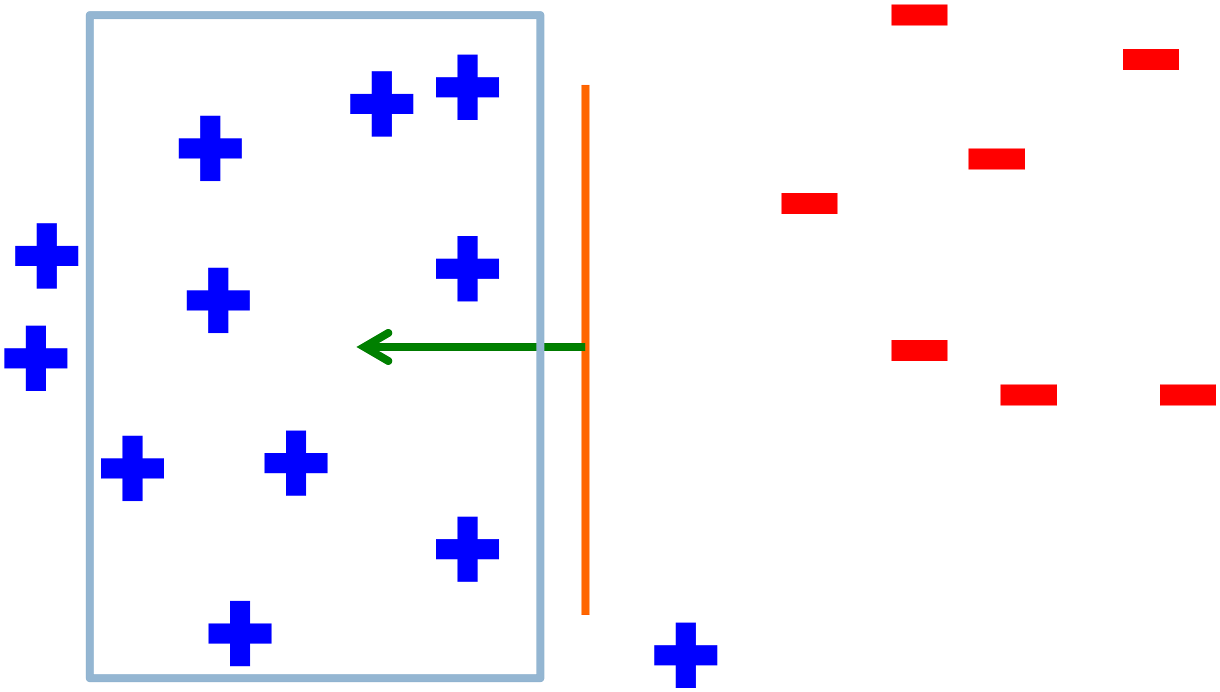
Order matters: a bad order



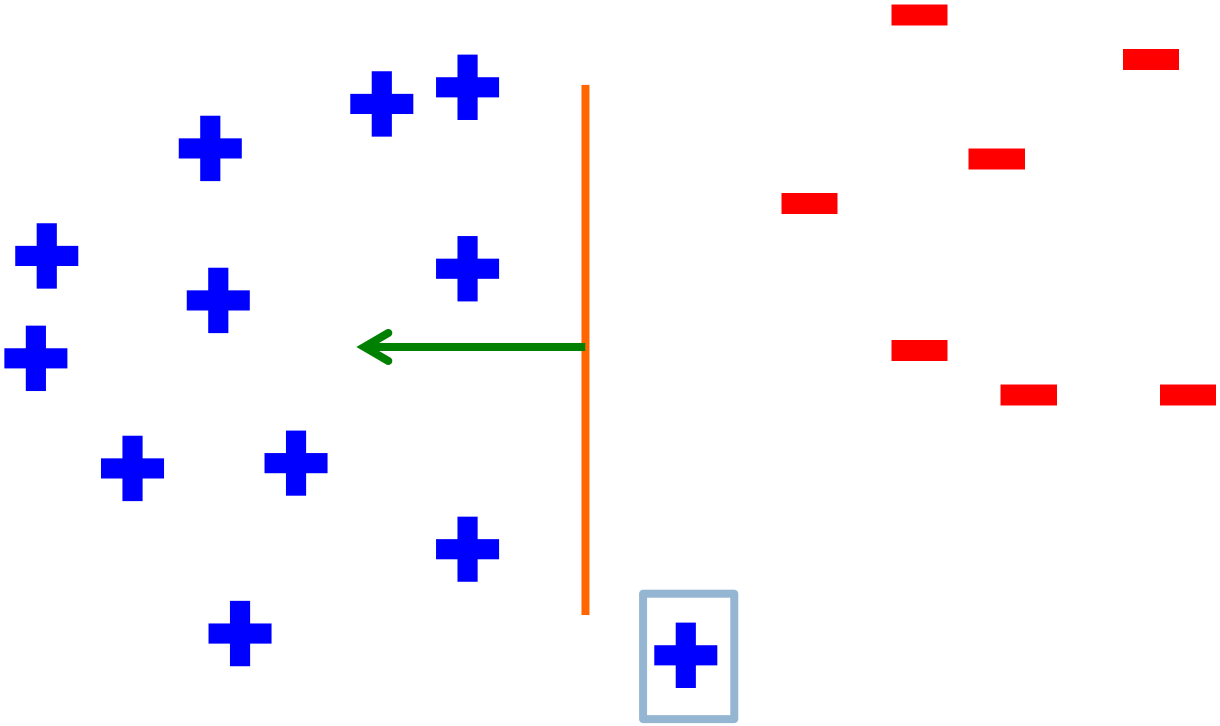
Order matters: a bad order



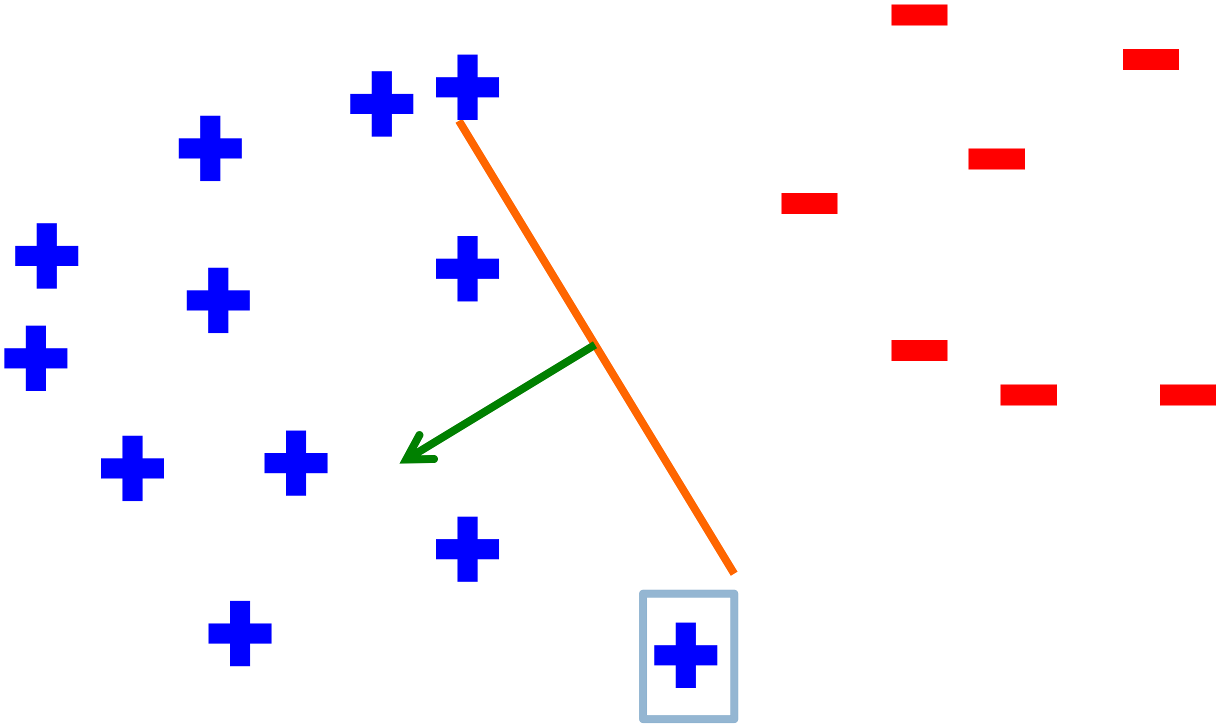
Order matters: a bad order



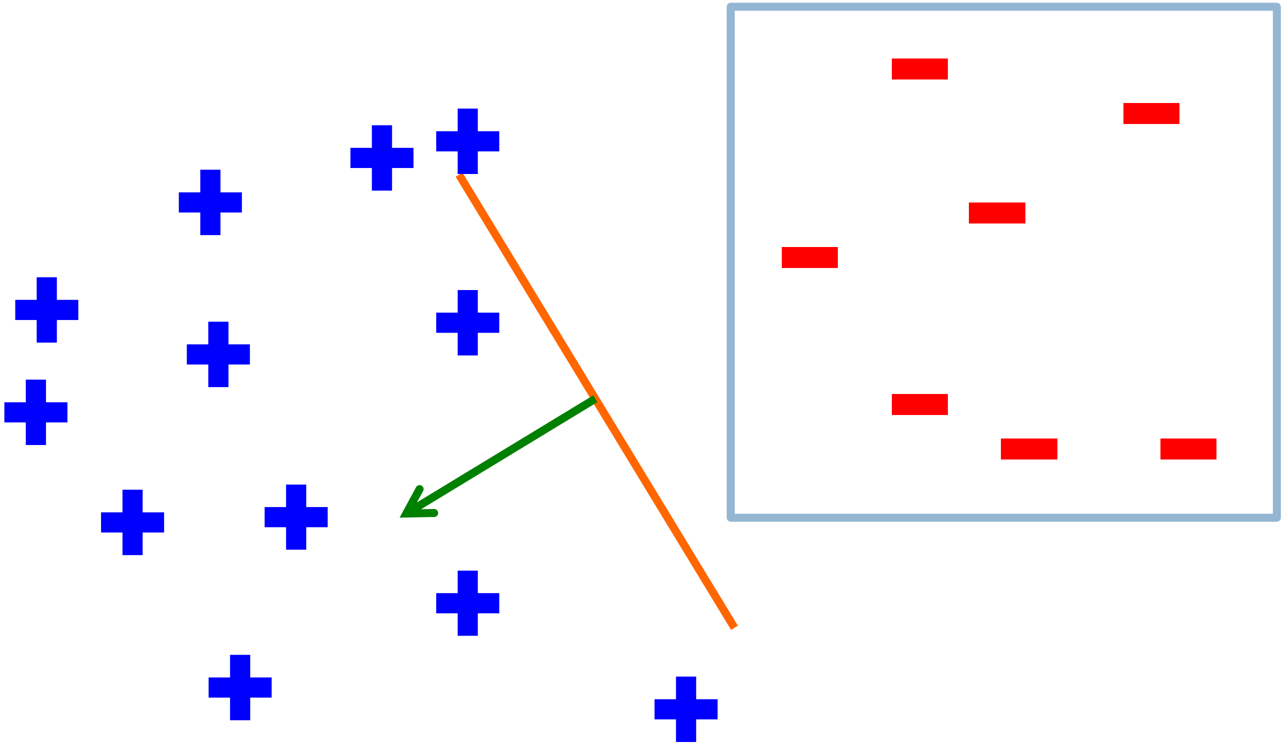
Order matters: a bad order



Order matters: a bad order



Order matters: a bad order

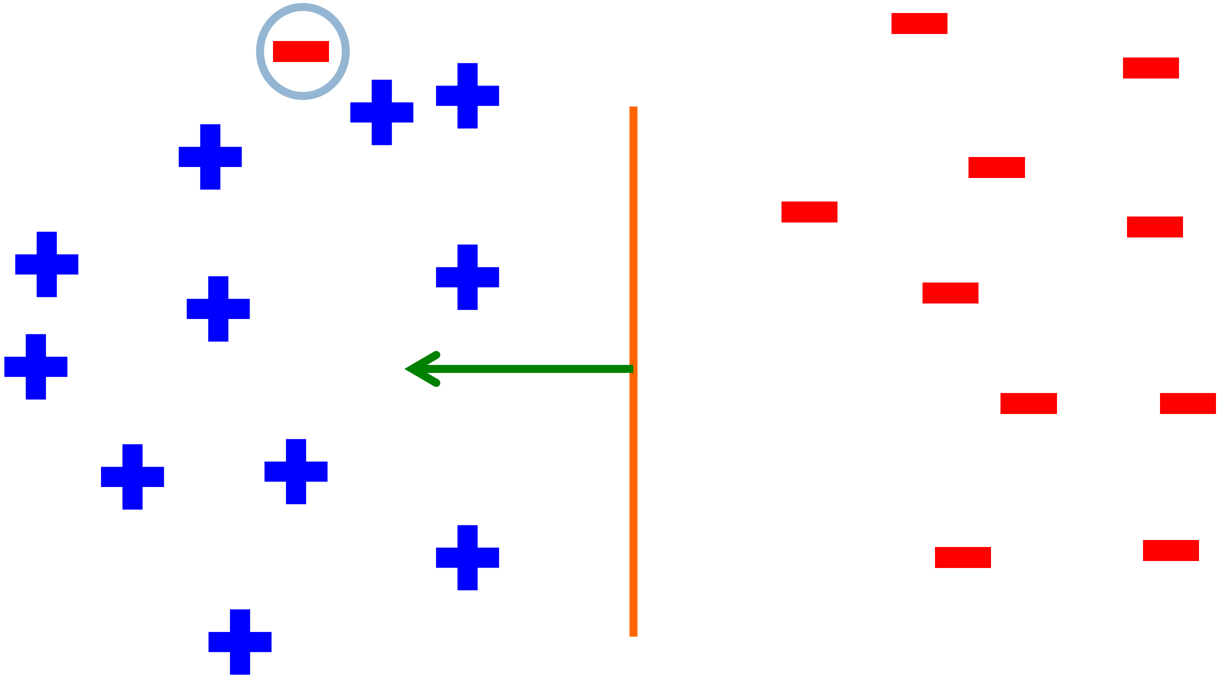


Solution?

Ordering

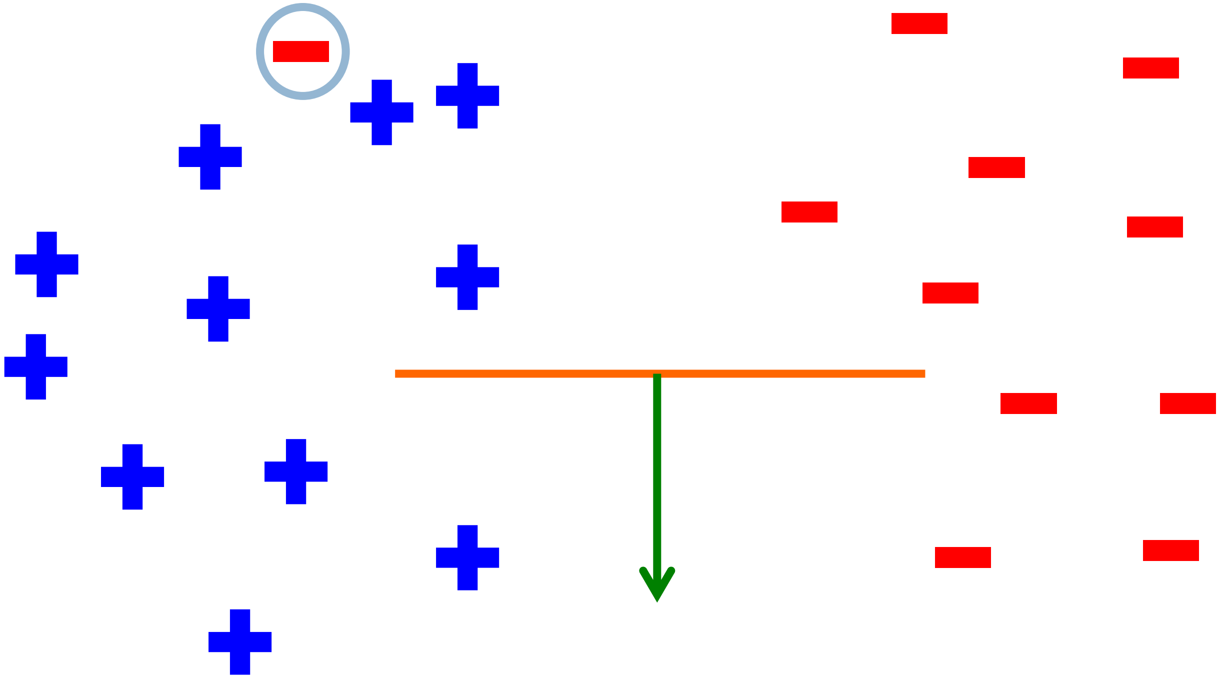
```
repeat until convergence (or for some # of iterations):
  randomize order or training examples
  for each training example ($f_1, f_2, \ldots, f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$
    if $prediction * label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Improvements



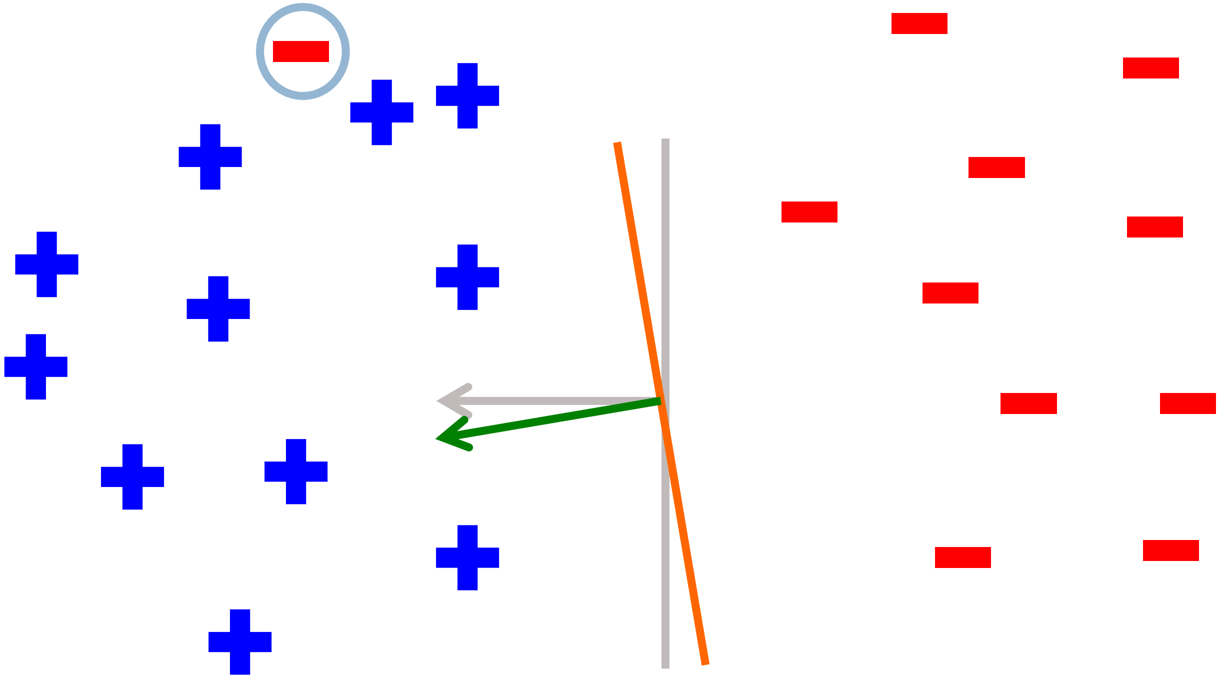
What will happen when we examine this example?

Improvements



Does this make sense? What if we had previously gone through ALL of the other examples correctly?

Improvements



Maybe just move it slightly in the direction of correction

Voted perceptron learning

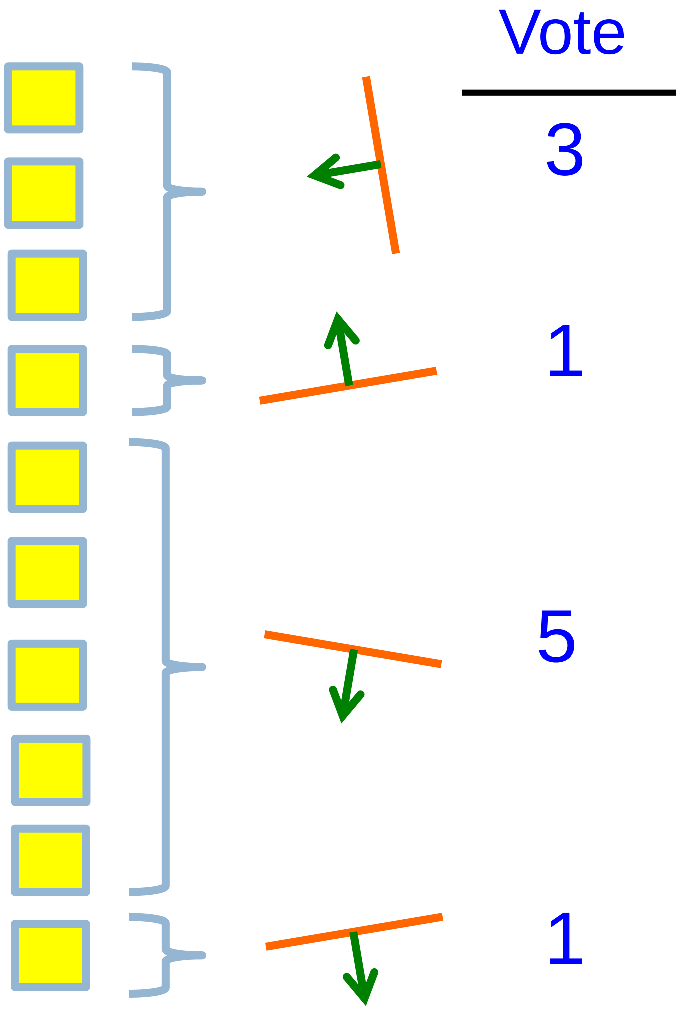
Training

- every time a mistake is made on an example:
    - store the weights (i.e. before changing for current example)
    - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes

Voted perceptron learning

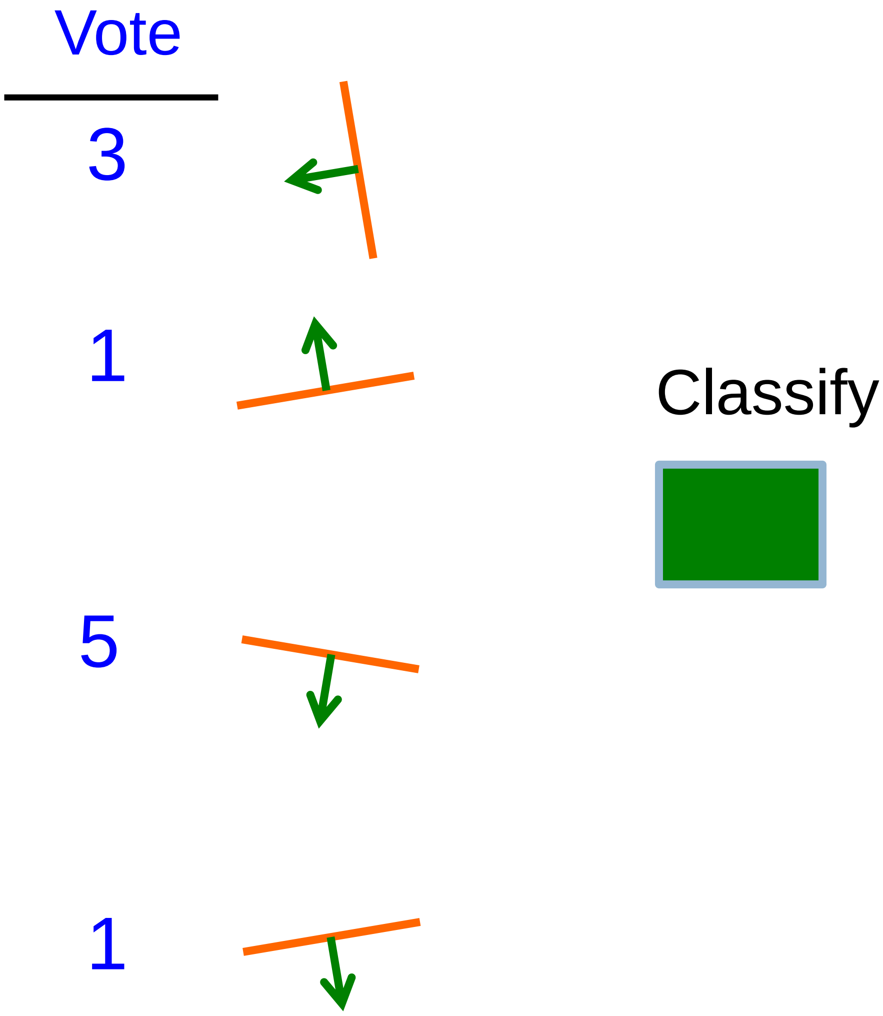


Training

every time a mistake is made on an example:

- store the weights
- store the number of examples that set of weights got correct

Voted perceptron learning



Vote

3

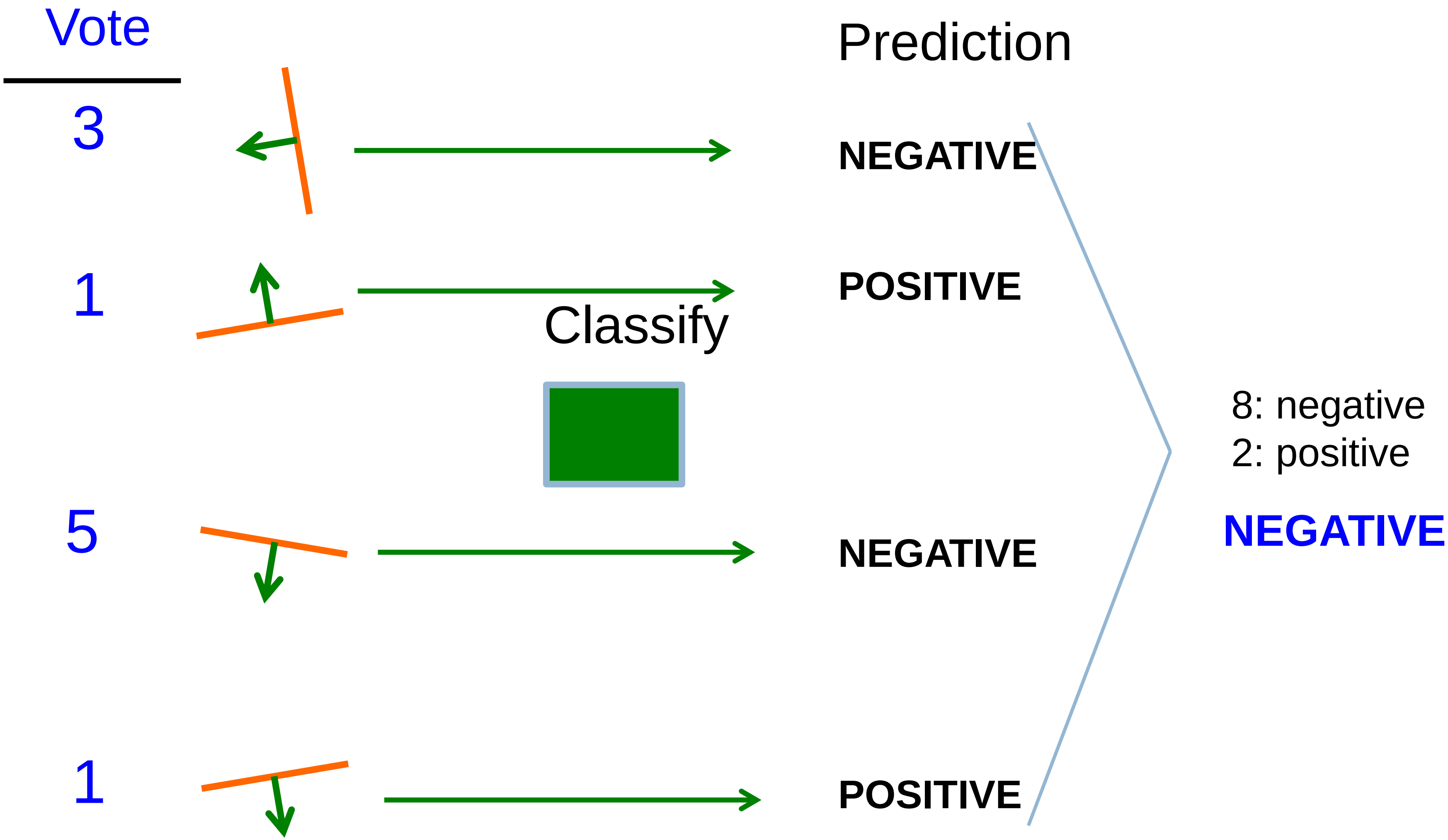
1

5

1

Classify

Voted perceptron learning



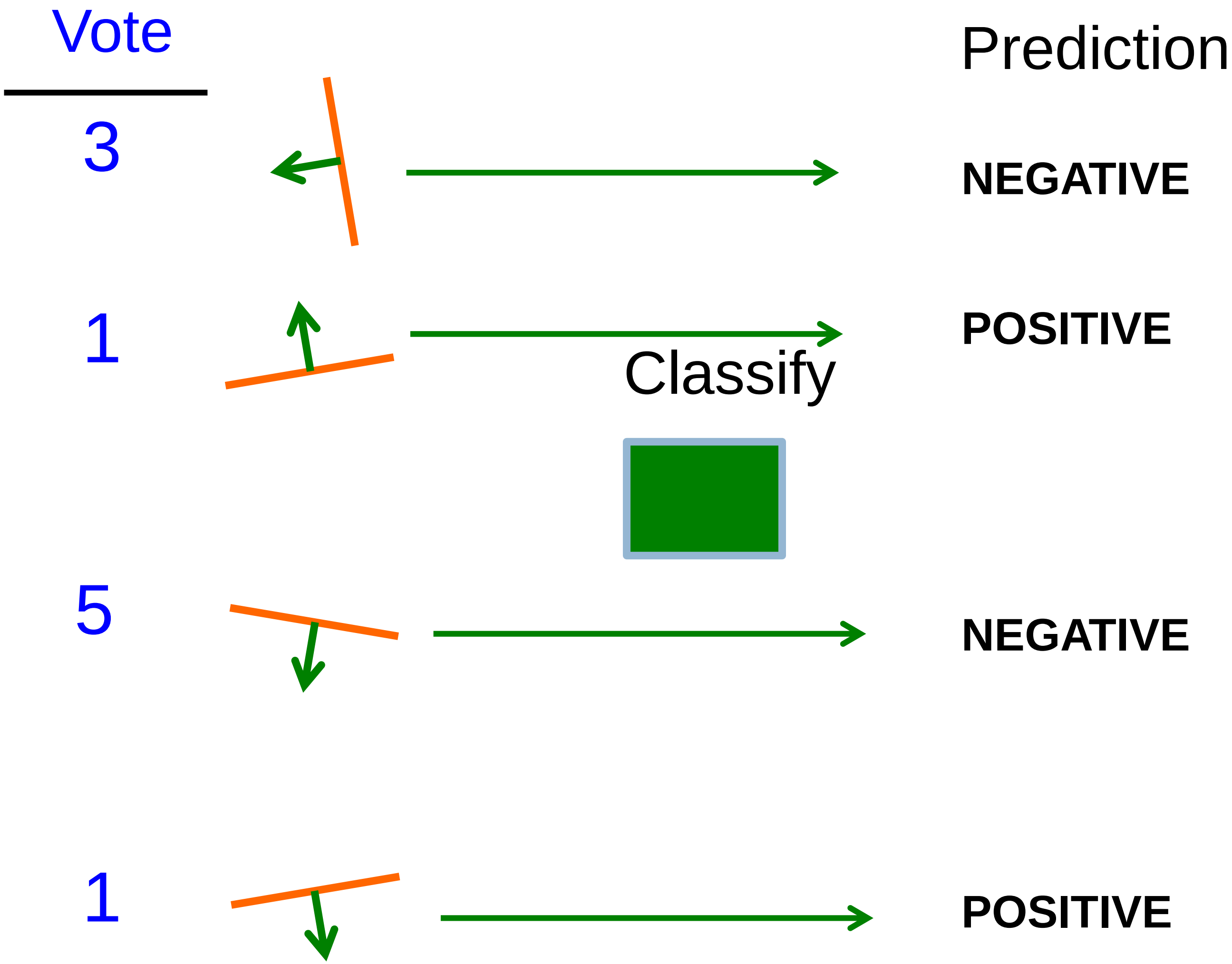
Vote	Prediction
3	NEGATIVE
1	POSITIVE
5	NEGATIVE
1	POSITIVE

Classify

8: negative
2: positive

NEGATIVE

Voted perceptron learning



Vote	Prediction
3	**NEGATIVE**
1	**POSITIVE**
5	**NEGATIVE**
1	**POSITIVE**

Classify 🟩

Voted perceptron learning

Works much better in practice

Avoids overfitting, though it can still happen

Avoids big changes in the result by examples examined at the end of training

Voted perceptron learning

Training

- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes

Any issues/concerns?

Voted perceptron learning

Training

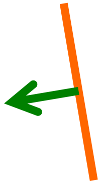
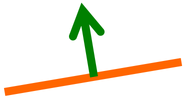
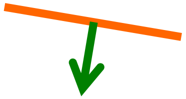
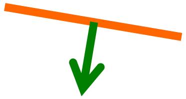
- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

Classify

- calculate the prediction from ALL saved weights
- multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
- said another way: pick whichever prediction has the most votes

1. Can require a lot of storage
2. Classifying becomes very, very expensive

Average perceptron

Vote		
3		$w_1^1, w_2^1, \ldots, w_n^1, b^1$
1		$w_1^2, w_2^2, \ldots, w_n^2, b^2$
5		$w_1^3, w_2^3, \ldots, w_n^3, b^3$
1		$w_1^4, w_2^4, \ldots, w_n^4, b^4$

$$\overline{w_i} = \frac{3w_i^1 + 1w_i^2 + 5w_i^3 + 1w_i^4}{10}$$

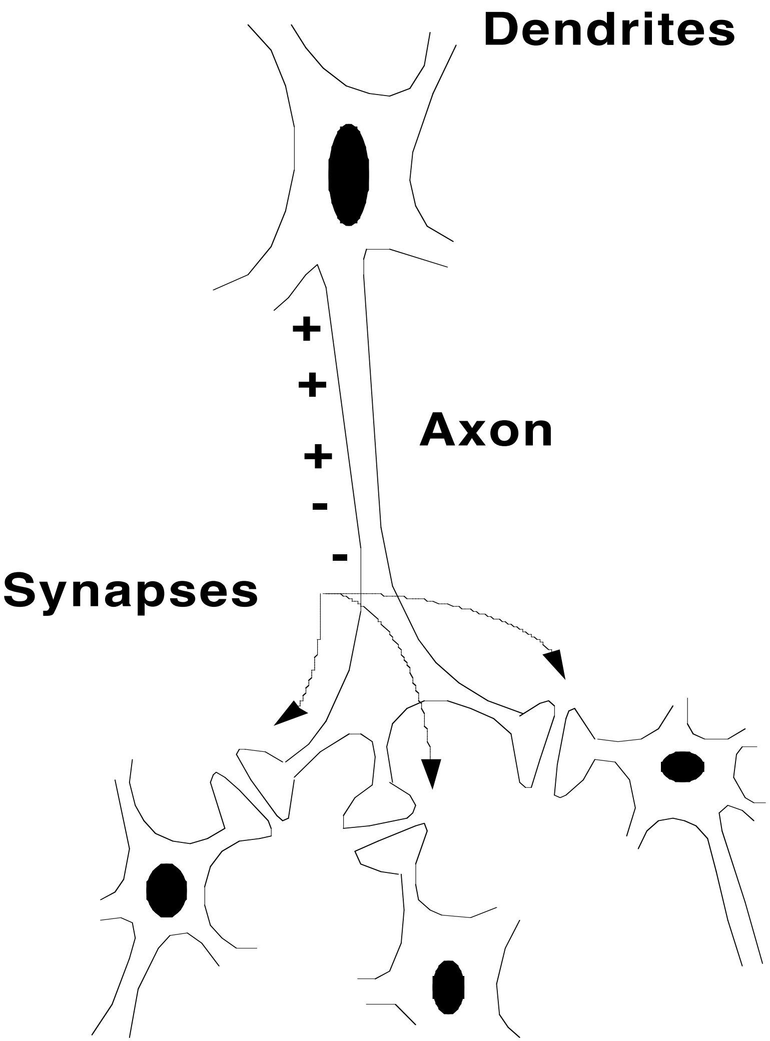
The final weights are the *weighted average* of the previous weights

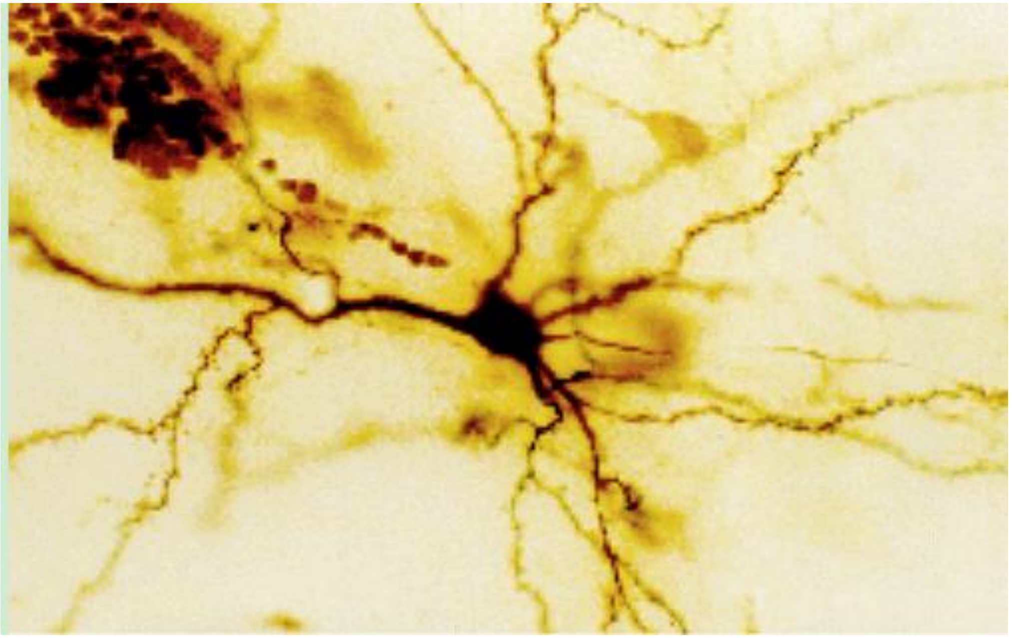
Perceptron learning algorithm

```
repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_n$, label):
    $prediction = b + \sum_{i=1}^{n} w_i f_i$

    if $prediction * label \leq 0$:  // they don't agree
      for each $w_i$:
        $w_i = w_i + f_i$*label
      $b = b$ + label
```

Why is it called the “perceptron” learning algorithm if what it learns is a line? Why not “line learning” algorithm?

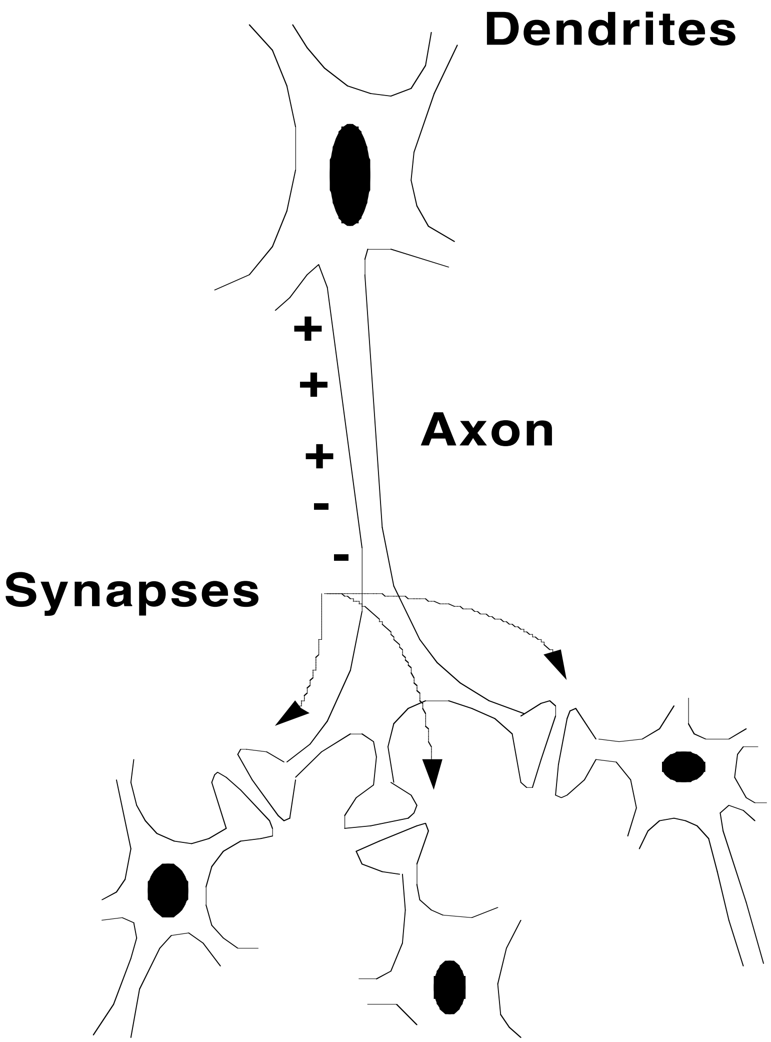
Our Nervous System





Neuron

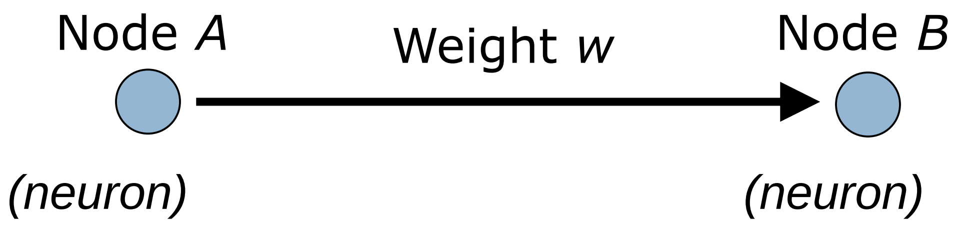
Our nervous system: *the computer science view*



the human brain is a large collection of interconnected neurons

a NEURON is a brain cell

- collect, process, and disseminate electrical signals
- Neurons are connected via synapses
- They FIRE depending on the conditions of the neighboring neurons



w is the strength of signal sent between A and B.

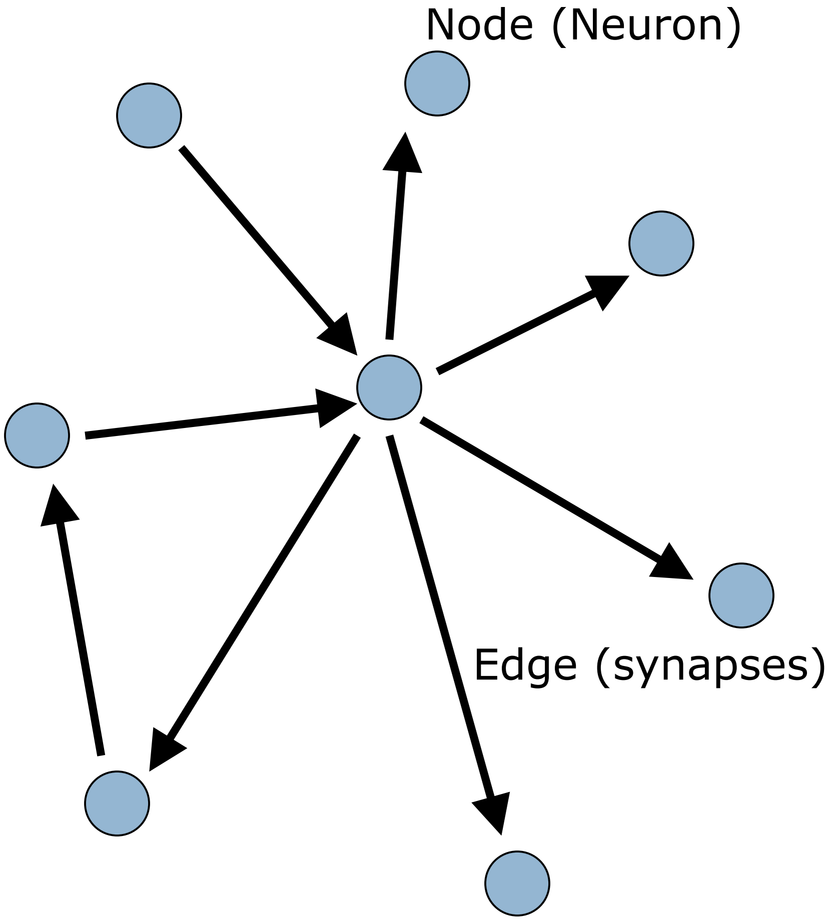
If *A* fires and *w* is **positive**, then *A* **stimulates** *B*.

If *A fires* and *w* is **negative**, then *A* **inhibits** *B*.

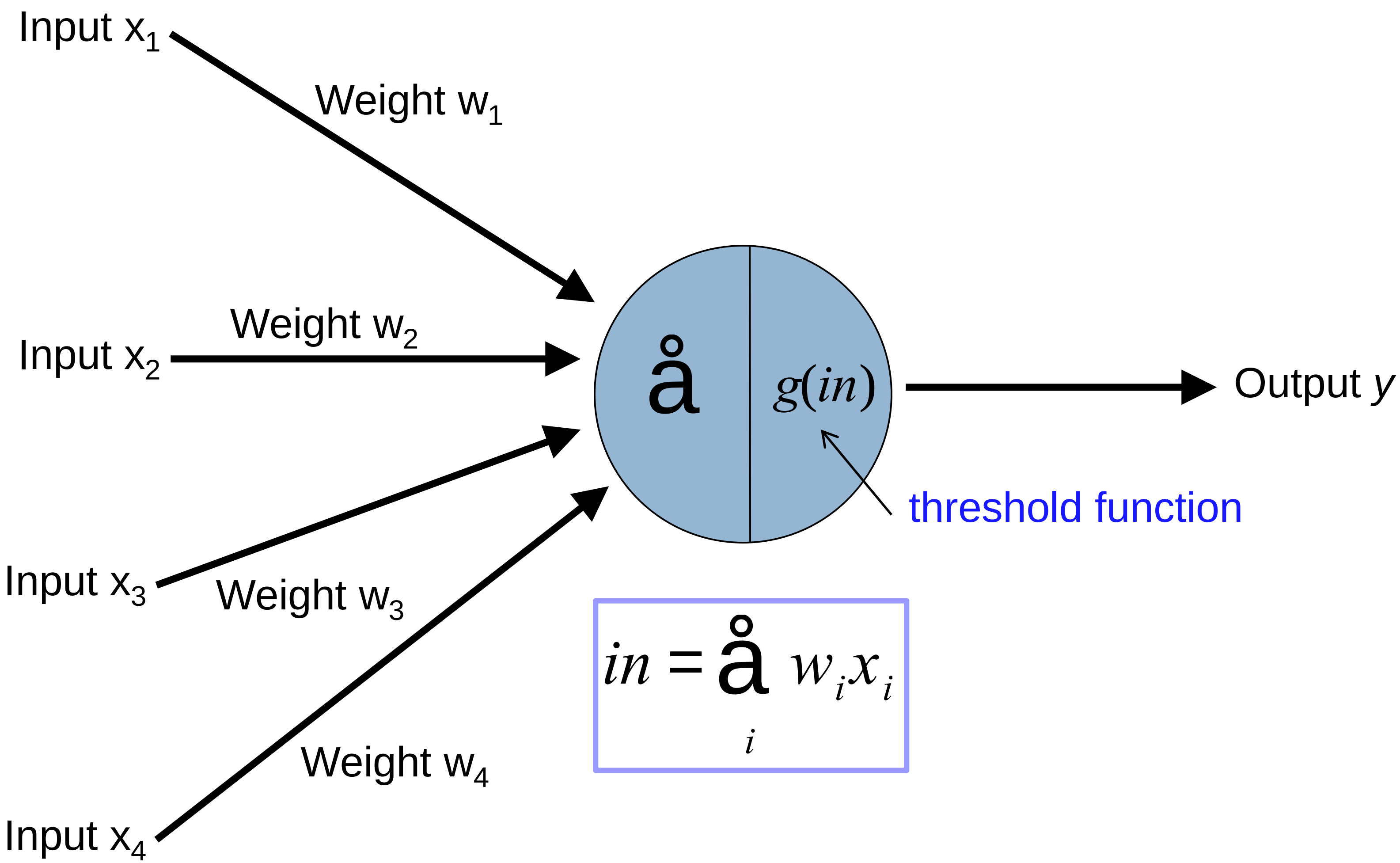
If a node is stimulated enough, then it also fires.

How much stimulation is required is determined by its **threshold**.

Neural Networks



A Single Neuron/Perceptron

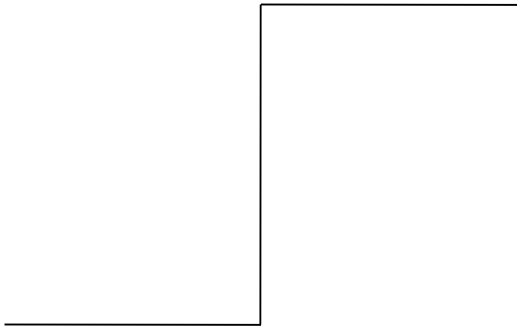


$$in = \underset{i}{\text{å}}\, w_i x_i$$

Possible threshold functions

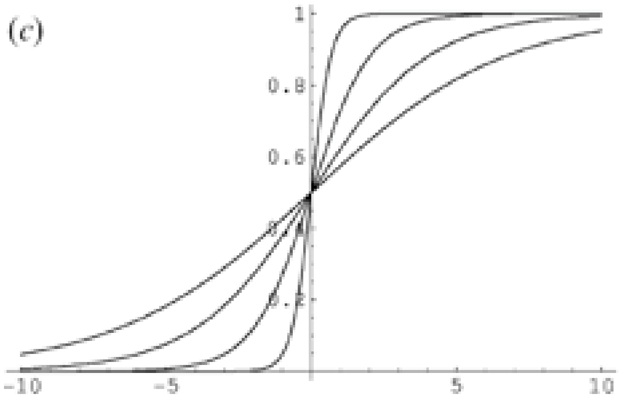
hard threshold:

if *in* (the sum of weights) >= *threshold* 1, 0 otherwise

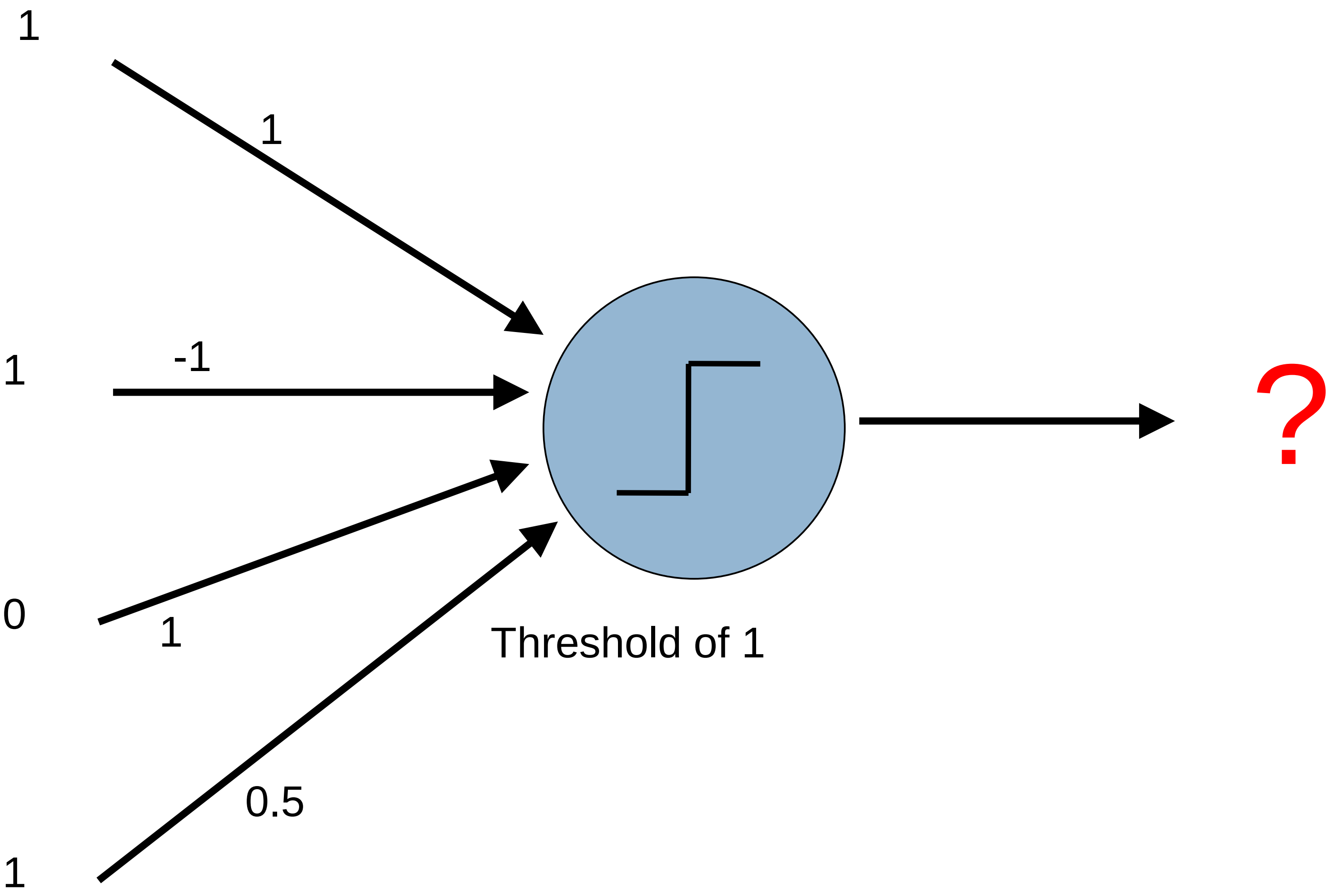


Sigmoid

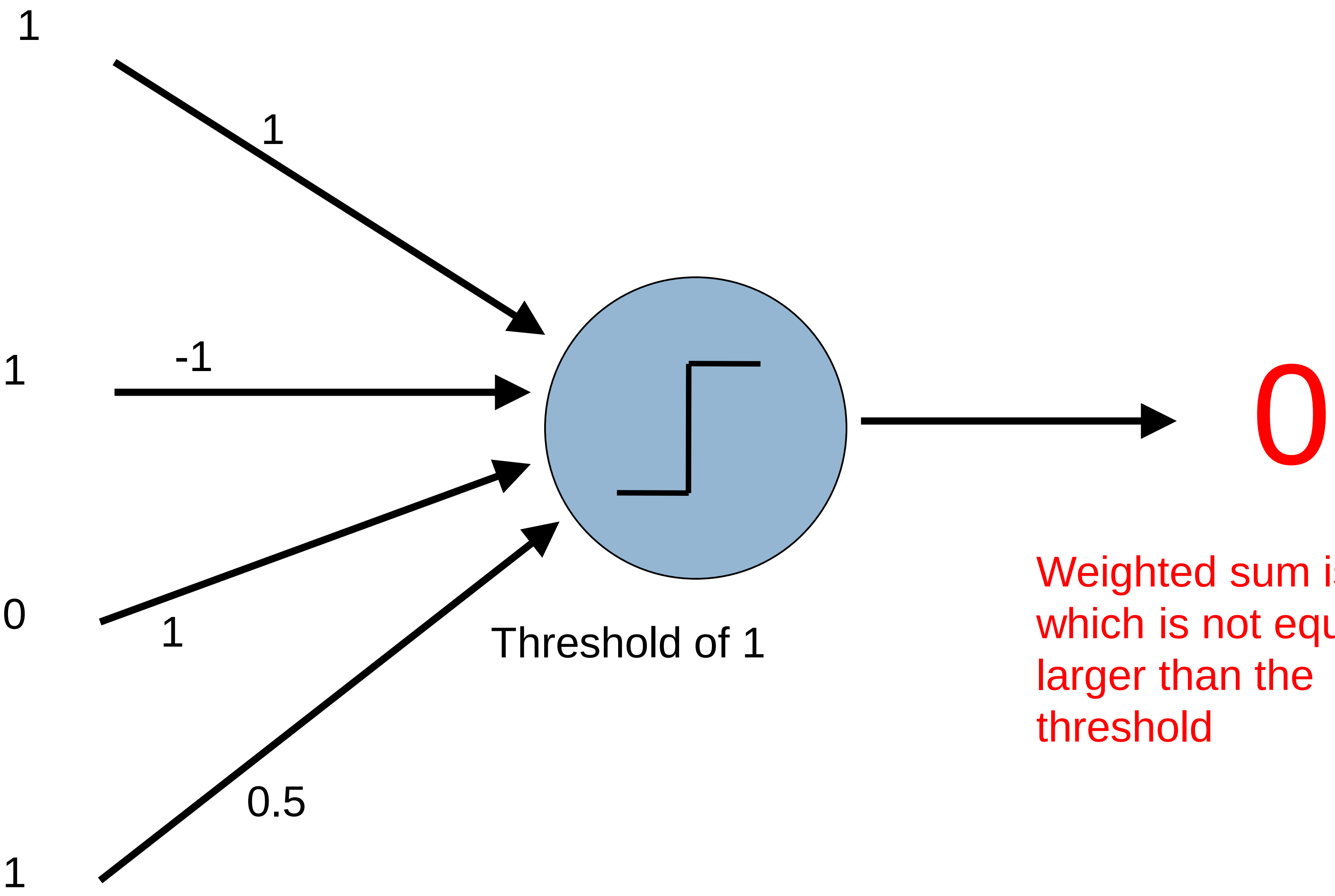
$$g(x) = \frac{1}{1 + e^{-ax}}$$



A Single Neuron/Perceptron

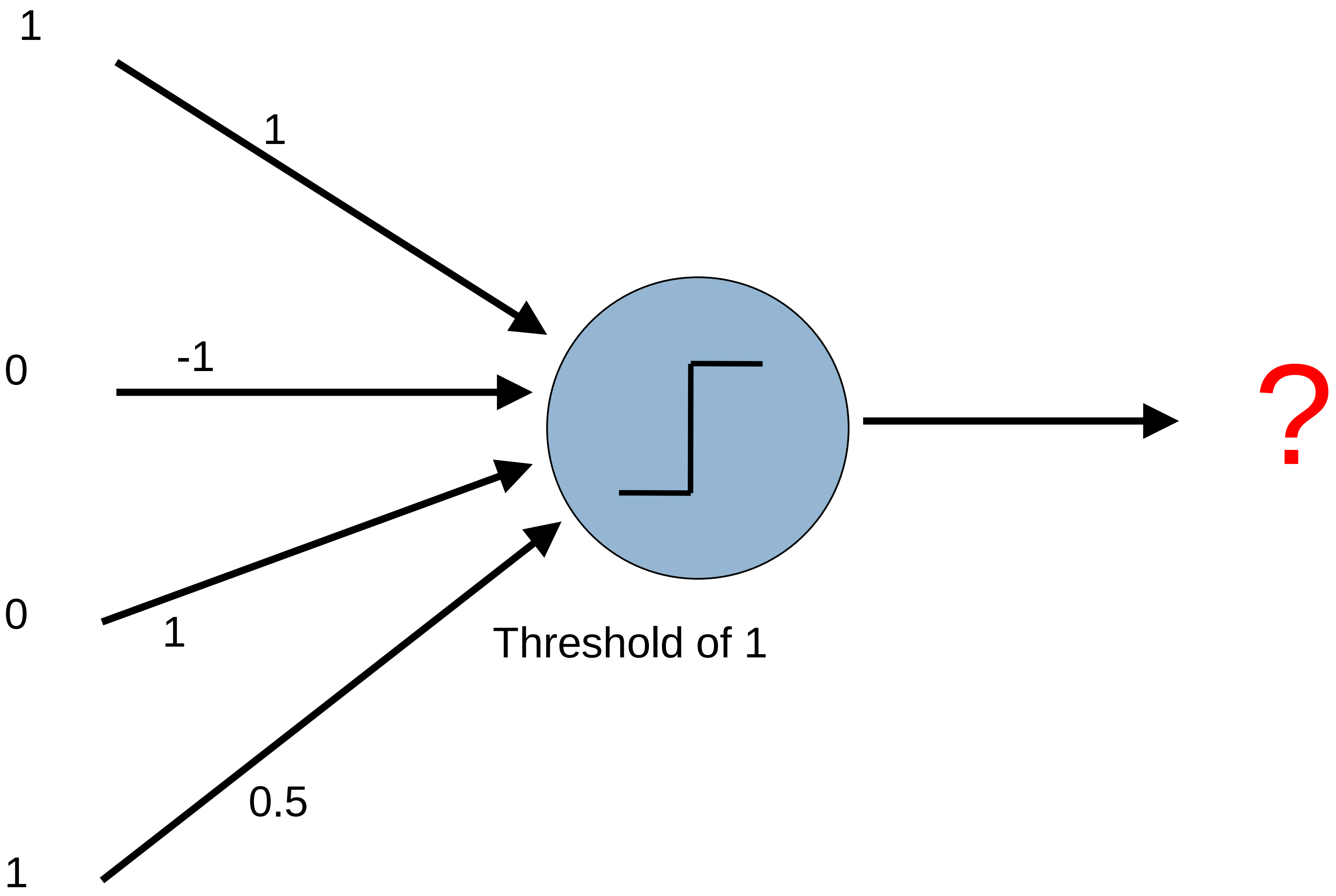


A Single Neuron/Perceptron

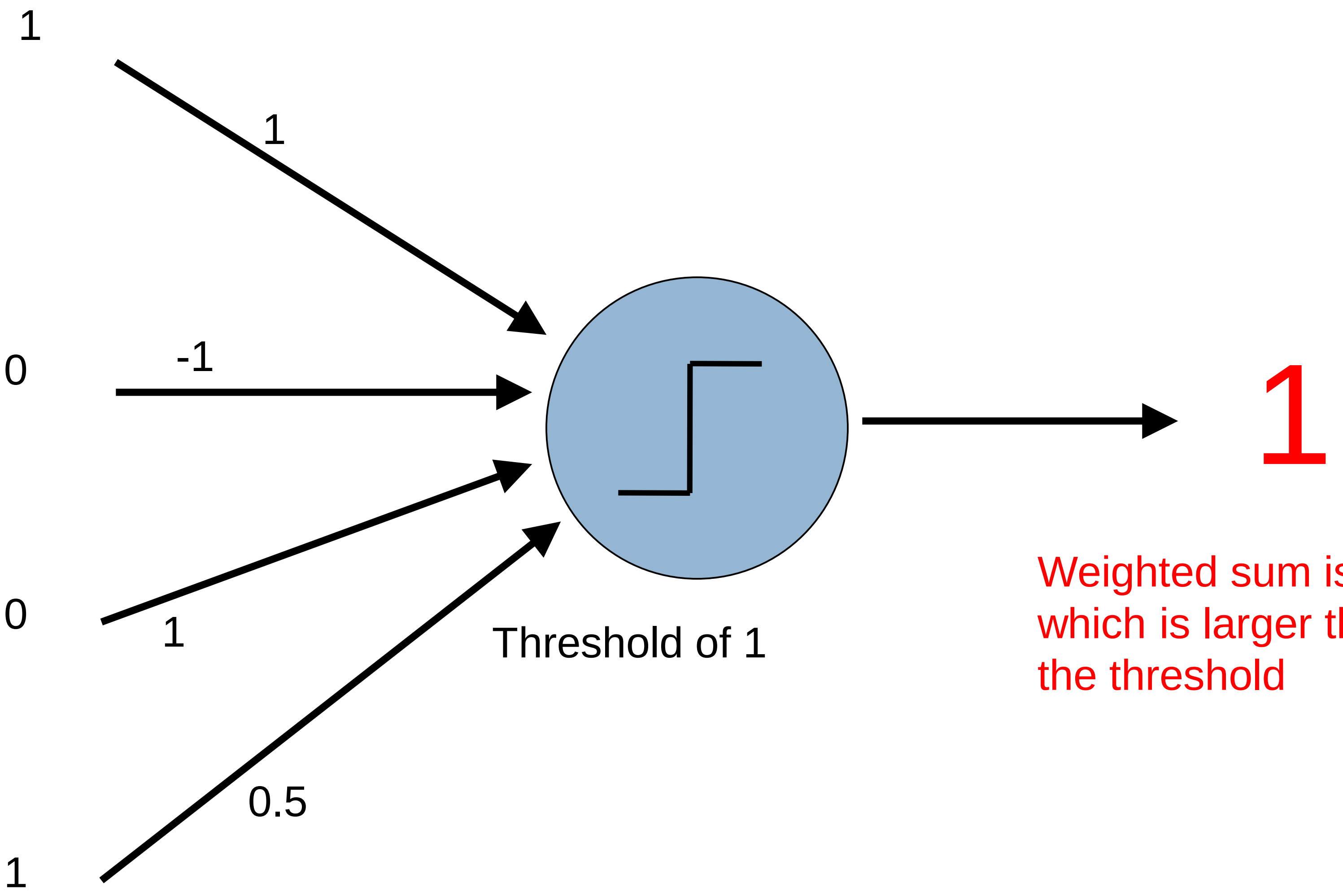


Weighted sum is 0.5, which is not equal or larger than the threshold

A Single Neuron/Perceptron



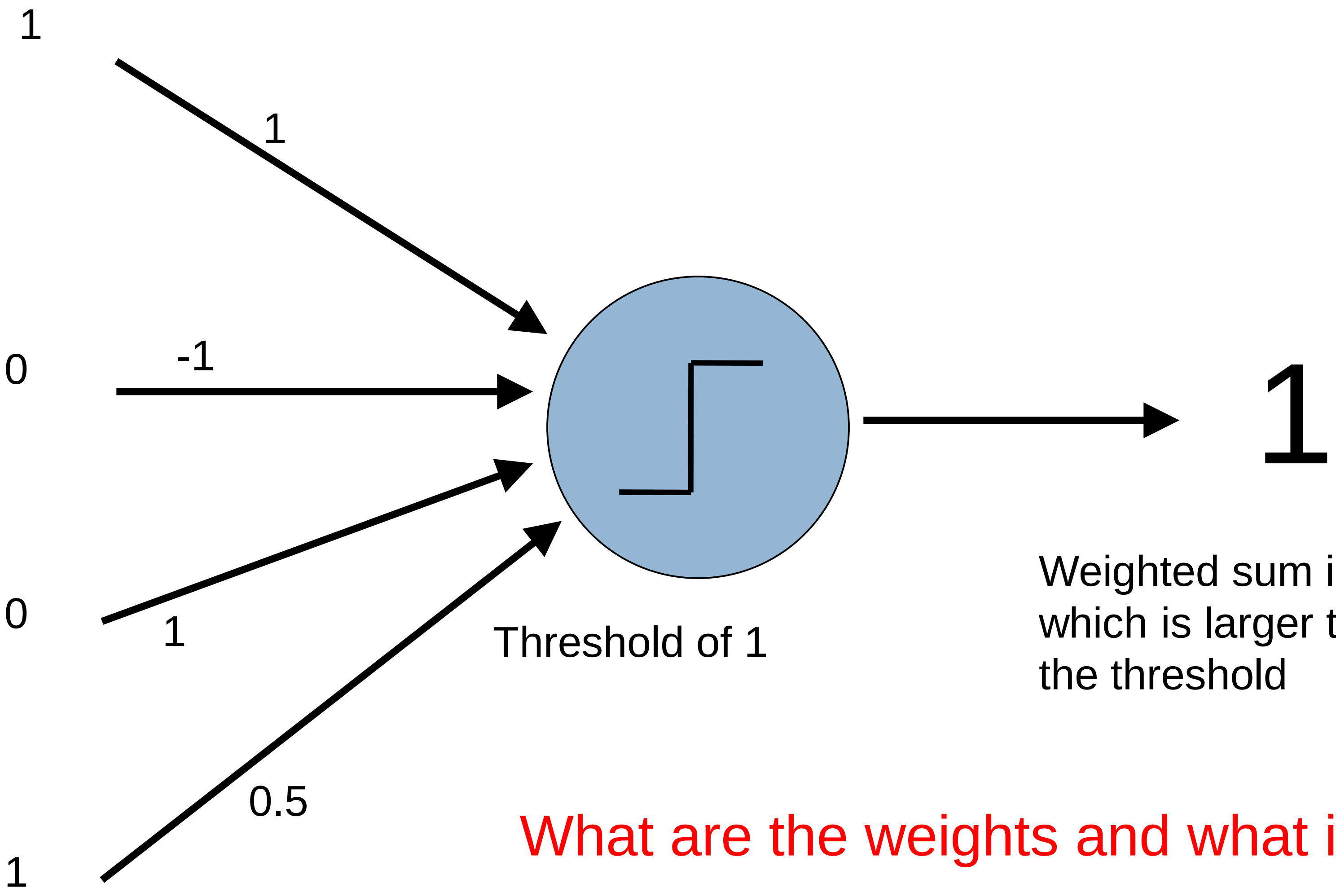
A Single Neuron/Perceptron



Threshold of 1

Weighted sum is 1.5,
which is larger than
the threshold

A Single Neuron/Perceptron



Weighted sum is 1.5, which is larger than the threshold

What are the weights and what is b?

History of Neural Networks

McCulloch and Pitts (1943) – introduced model of artificial neurons and suggested they could learn

Hebb (1949) – Simple updating rule for learning

Rosenblatt (1962) - the *perceptron* model

Minsky and Papert (1969) – wrote *Perceptrons*

Bryson and Ho (1969, but largely ignored until 1980s) – invented back-propagation learning for multilayer networks