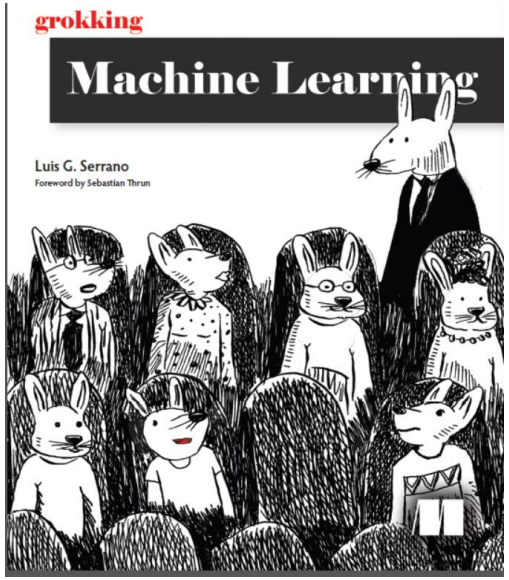


Machine Learning

| Perceptron – Error Function

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ElhosseiniAcademy

Agenda

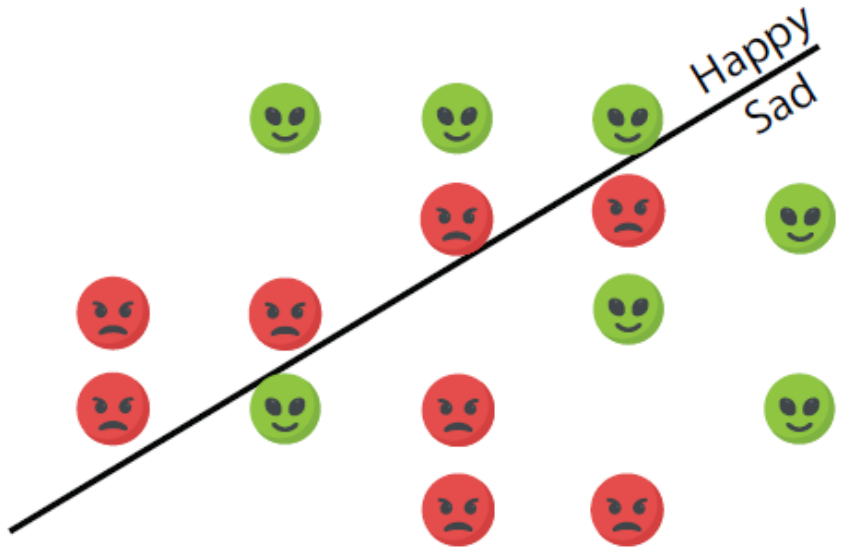
- The error function
- The perceptron trick

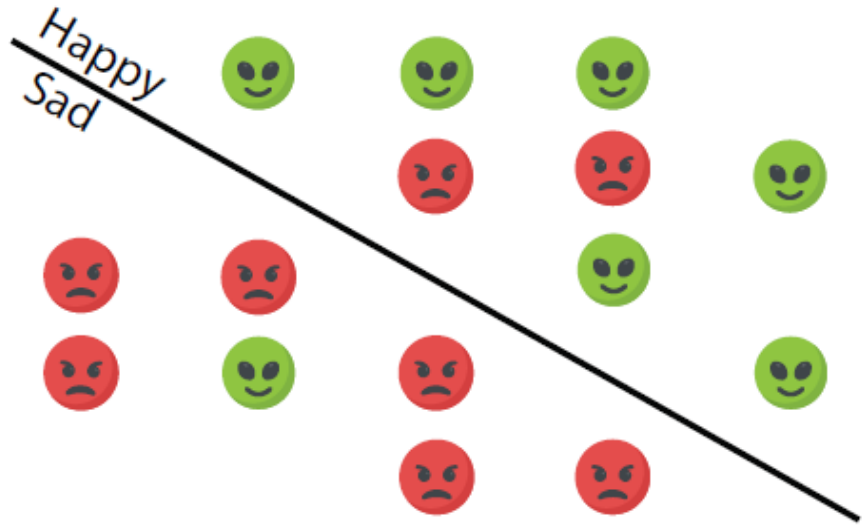
The error function

- How do we determine whether a classifier is **good or bad**?
- How do we **find** the perceptron classifier that **best fits** our data?
- **Error function** tell us whether a perceptron classifier fits our data well

The error function

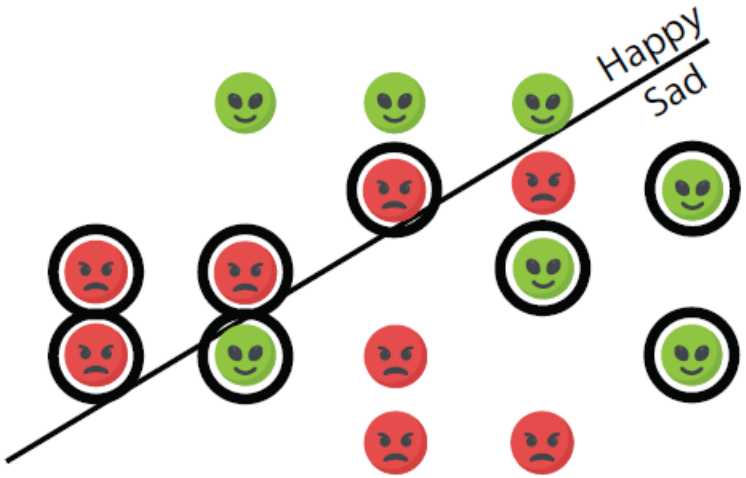
- The one on the left is a bad classifier, and the one on the right is good



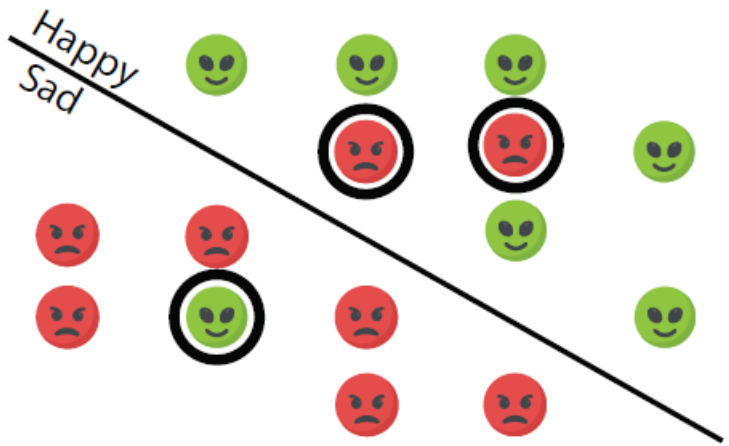


Error function 1: Number of errors

- Counting the number of points that it classifies incorrectly



Bad classifier
Error: 8



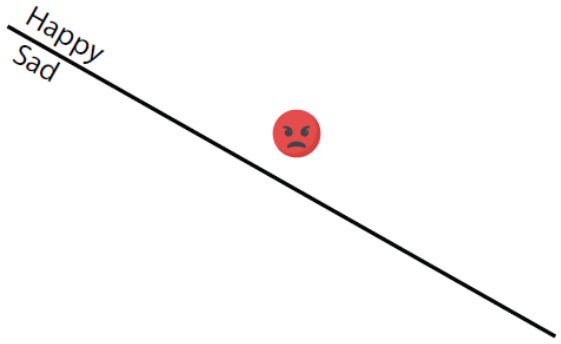
Good classifier
Error: 3

Error function 1: Number of errors

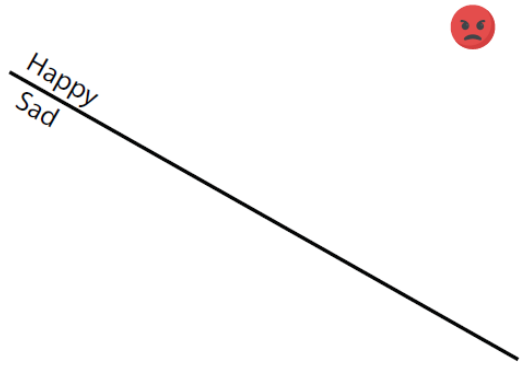
- This is a good error function, but it's not a great error function. Why?
  - It tells us when there is an error, but it doesn't measure the **gravity** of the error
  - if a sentence is **sad**, and the classifier gave it a **score of 1**, the classifier made a mistake. However, if **another classifier** gave it a score of **100**, this classifier made a much bigger mistake

Error function 1: Number of errors

- both classifiers misclassified a sad point by predicting that it is happy.
- However, the classifier on the **left located the line close** to the point, which means that the sad point is not too far from the sad zone.



Poorly classified



Very poorly classified

Error function 1: Number of errors

- Why **do we care about** measuring how bad an error is?
- The way to reduce an error is by decreasing it in small amounts, until we reach a point where the error is small

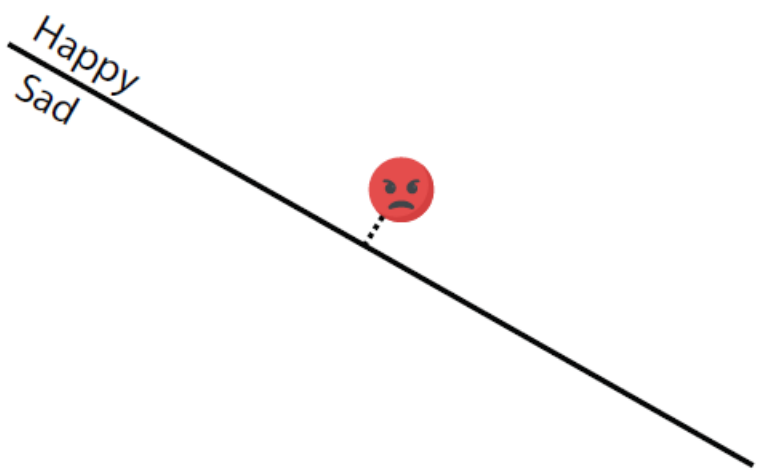
- If our error is calculated by counting the number of misclassified points, then this error will take only integer values. If we wiggle the line a small amount, the error **may not decrease at all**, and we **don't know** in which **direction** to move.

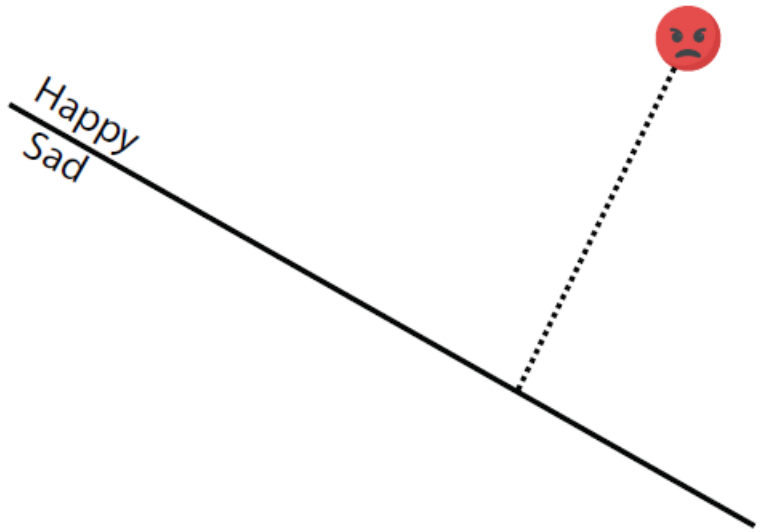
Error function 1: Number of errors

- We need a function that measures the magnitude of an error and that assigns a higher error to misclassified points that are far from the boundary than to those that are close to it.

Error function 2: Distance

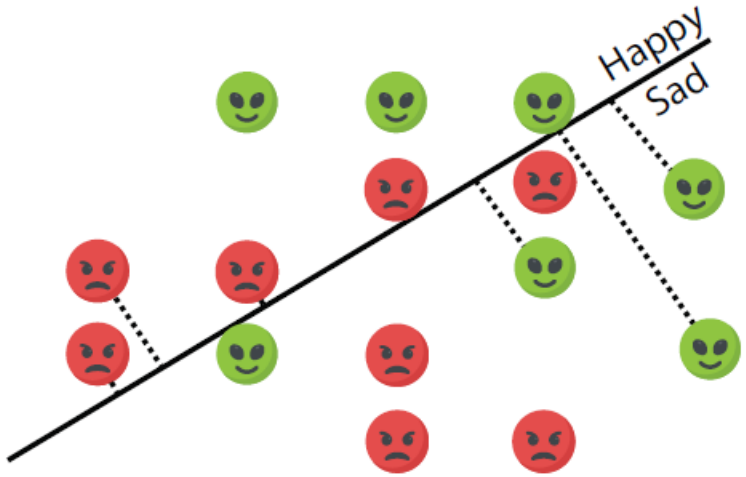
- Considering the **perpendicular distance** from the point to the line
- Points that are correctly classified produce an error of 0.
- Points that are misclassified produce an error equal to the distance from that point to the line.





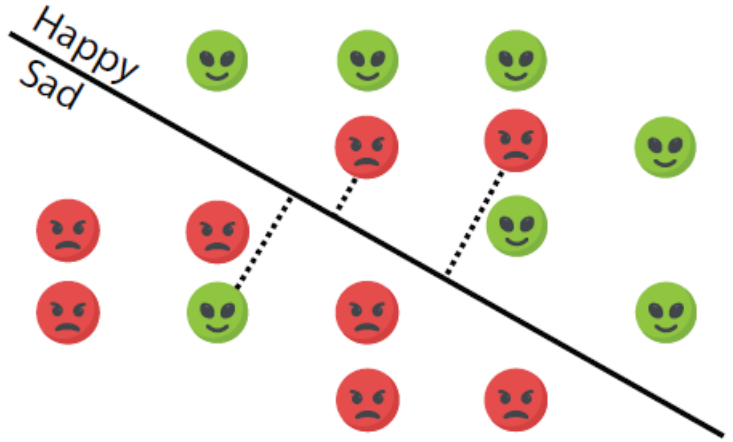
Error function 2: Distance

- Look only at the misclassified points and add the perpendicular distances from these points to the line



Bad classifier

Error: .....



Good classifier

Error: .....

Error function 2: Distance

- This is almost the error function we will use. Why don't we use this one?

- The distance from a point (x_1, y_1) to a line given by the equation $ax + by + c = 0$ can be calculated using the following formula:

$$\frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

- It contains a **square root – complicated derivatives**,

Error function 3: Score

let’s summarize the properties we want in an error function

- The error function of a **correctly** classified point is **0**.

- The error function of an incorrectly classified point is a **positive number**.
  - For misclassified points **close to the boundary**, the error function is **small**.
  - For misclassified points **far** from the boundary, the error function is **large**.

- It is given by a **simple formula**.

Error function 3: Score

- The concept of scoring a point with respect to a decision boundary simplifies the calculation of the model's error function
  - The points in the boundary have a score of 0.
  - The points in the positive zone have positive scores.
  - The points in the negative zone have negative scores.
  - The points close to the boundary have scores of low magnitude (i.e., positive or negative scores of low absolute value).
  - The points far from the boundary have scores of high magnitude (i.e., positive or negative scores of high absolute value).

Error function 3: Score

- Thus, We define the **error** as the **absolute value of the score** for misclassified points.

Perceptron error for a point (general)

- If the point is correctly classified, the error is 0.

- If the point is misclassified, the error is $|\boldsymbol{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b}|$

Mean perceptron error: Entire dataset

- Take the **average** of all the errors corresponding to all the points.

- We can also **take the sum** if we choose, although in this chapter we choose the **average** and call it the **mean perceptron error**.

Example

- Consider the dataset made of four sentences, two labeled happy and two labeled sad

Sentence	*Aack*	*Beep*	Label (mood)
Aack	1	0	Sad
Beep	0	1	Happy
Aack beep beep beep	1	3	Happy
Aack beep beep aack aack	3	2	Sad

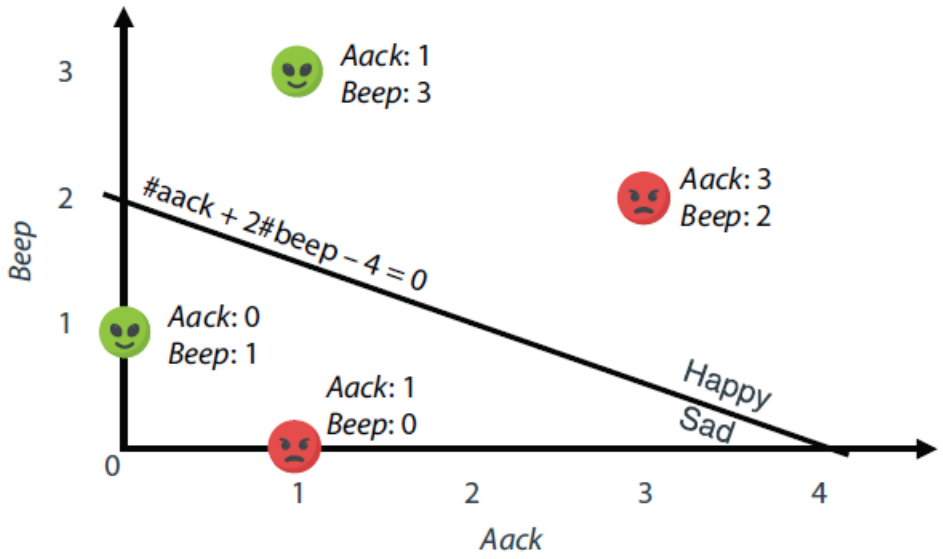
- We'll compare the following two classifiers on this dataset

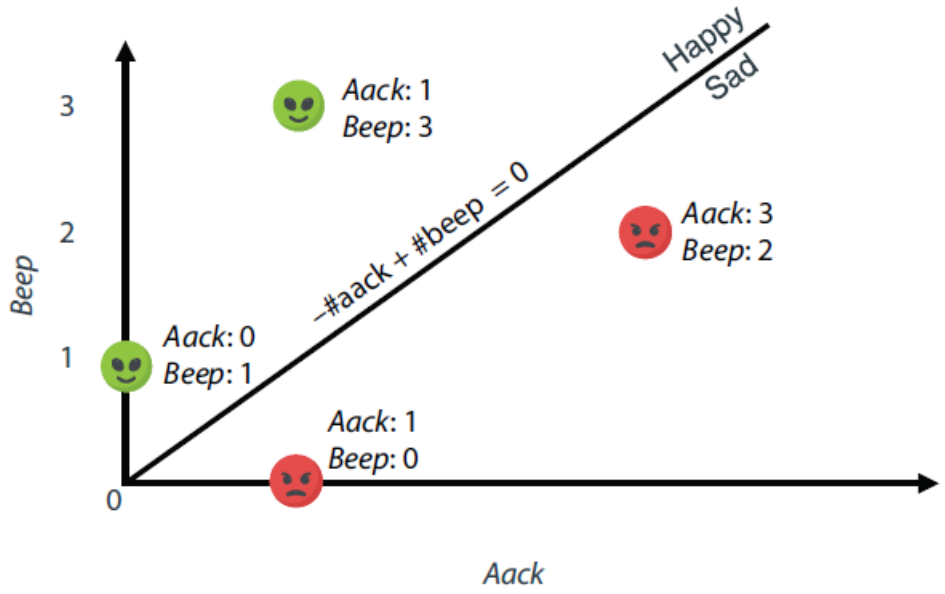
Example

Sentence	*Aack*	*Beep*	Label (mood)
Aack	1	0	Sad
Beep	0	1	Happy
Aack beep beep beep	1	3	Happy
Aack beep beep aack aack	3	2	Sad

- We'll compare the following two classifiers on this dataset
- $1x_{aack} + 2x_{beep} - 4$ $\quad \hat{y} = step(1x_{aack} + 2x_{beep} - 4)$
- $-1x_{aack} + 1x_{beep}$ $\quad \hat{y} = step(-1x_{aack} + 1x_{beep})$

Example


Aack: 1
Beep: 3
Aack: 3
Beep: 2
#aack + 2#beep – 4 = 0
Aack: 0
Beep: 1
Aack: 1
Beep: 0
Happy
Sad
Beep
Aack
0
1
2
3
4

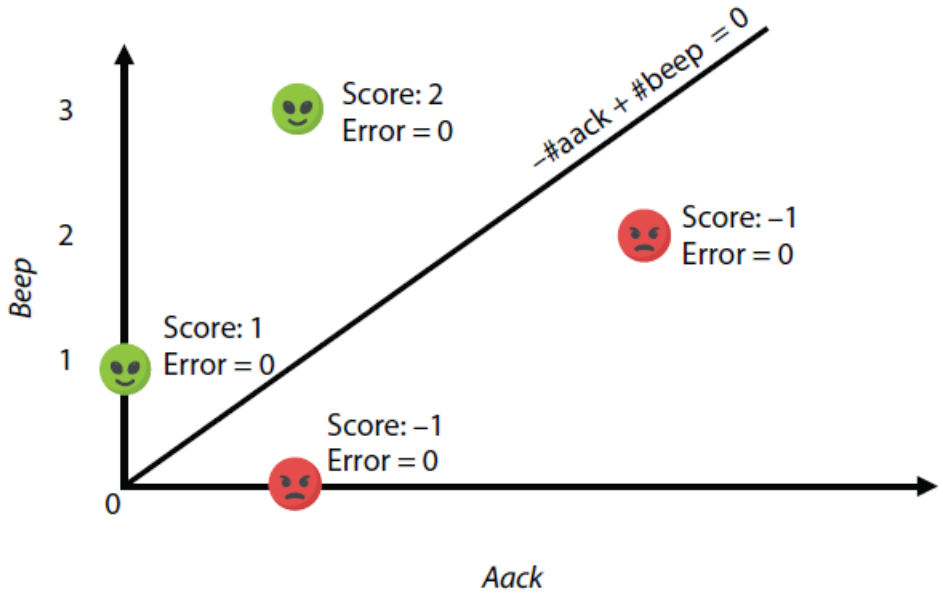

Aack: 1
Beep: 3
Happy
Sad
–#aack + #beep = 0
Aack: 3
Beep: 2
Aack: 0
Beep: 1
Aack: 1
Beep: 0
Beep
Aack
0
1
2
3

Example

Datapoint	Label	Classifier 1 score	Classifier 1 prediction	Classifier 1 error	Classifier 2 score	Classifier 2 prediction	Classifier 2 error
(1, 0)	Sad	-3	0	0	-1	0	0
(0, 1)	Happy	-2	0	2	1	1	0
(1, 3)	Happy	3	1	0	2	1	0
(3, 2)	Sad	3	1	3	-1	0	0
Mean Perceptron error				1.25			0

Example

Mean perceptron error = $\frac{1}{4}(0+3+0+2) = 1.25$



Mean perceptron error = $\frac{1}{4}(0+0+0+0) = 0$

The perceptron algorithm

- Start with a random perceptron classifier.
- Slightly improve the classifier. (Repeat many times.)
- Measure the perceptron error to decide when to stop running the loop.

The perceptron trick

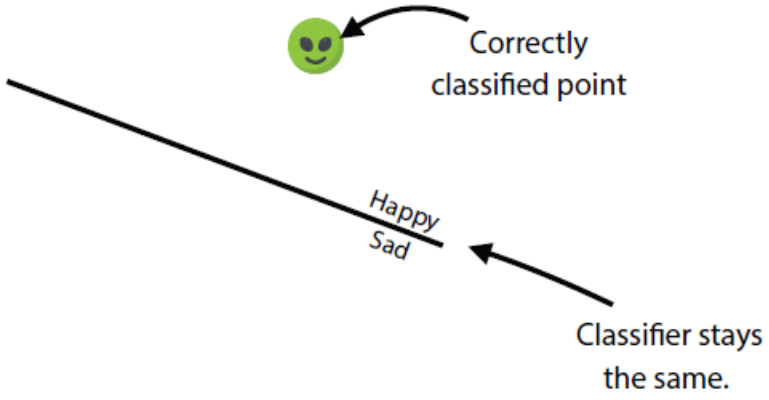
- The perceptron trick is a **tiny step** that helps us go from a perceptron classifier to a slightly better perceptron classifier

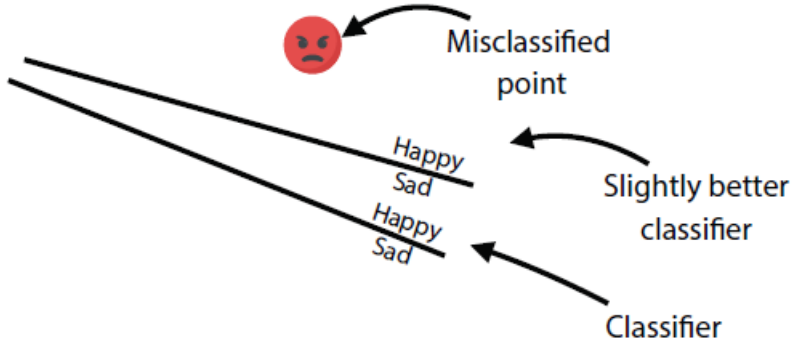
- Case 1: If the point is correctly classified, leave the line as it is.

- Case 2: If the point is **incorrectly classified**, **move** the line a **little closer** to the point.

The perceptron trick

- Moving the line closer to it may not put it on the right side, but at least it gets it closer to the line and, thus, closer to the correct side of the line.
- We repeat this process many times, so it is imaginable that one day we'll be able to move the line past the point, thus correctly classifying it





The perceptron trick

- we use a learning rate of $\eta = 0.01$
- $1x_{aack} + 2x_{beep} - 4$ $\quad\quad$ $\hat{y} = step(1x_{aack} + 2x_{beep} - 4)$

- Sentence 1: “Beep aack aack beep beep beep beep.”
- Label: **Sad** (0)

- Classifier 1 score: 8
- Classifier 1 prediction: 1 (**Happy**)

The perceptron trick

- we use a learning rate of $\eta = 0.01$
- $1x_{aack} + 2x_{beep} - 4$ $\hat{y} = step(1x_{aack} + 2x_{beep} - 4)$

- The sentence should have had a **negative score**, to be classified as sad. However, the classifier gave it a **score of 8**, which is **positive**. We **need to decrease** this score.
- One way to decrease it is to **subtract the learning rate** to the weight of aack to the weight of beep and to the bias, thus obtaining new weights
- Weight for Aack: 1−0.01=0.99
- Weight for Beep: 2-−0.01=1.99
- Bias: -4−0.01=−4.01

The perceptron trick

- we use a learning rate of $\eta = 0.01$
- $1x_{aack} + 2x_{beep} - 4$ $\hat{y} = step(1x_{aack} + 2x_{beep} - 4)$

- The word **beep appeared** many **more times** than the word aack. In some way, beep is more crucial to the score of the sentence than aack. We should probably decrease the weight of beep more than the score of aack

The perceptron trick

Inputs:

- A perceptron with weights a, b, and bias c
- A point with coordinates (x_1, x_2) and label y
- A small positive value η (the learning rate)

Output:

- A perceptron with new weights a', b', and bias c'

Procedure:

- The prediction the perceptron makes at the point is $\hat{y} = step(ax_1 + bx_2 + c)$.
- **Case 1:** If $\hat{y} = y$:
  - **Return** the original perceptron with weights a, b, and bias c.
- **Case 2:** If $\hat{y} = 1$ and $y = 0$:
  - **Return** the perceptron with the following weights and bias:
    - $a' = a - \eta x_1$
    - $b' = b - \eta x_2$
    - $c' = c - \eta x_1$.
- **Case 3:** If $\hat{y} = 0$ and $y = 1$:
  - **Return** the perceptron with the following weights and bias:
    - $a' = a + \eta x_1$
    - $b' = b - \eta x_2$
    - $c' = c + \eta x_1$.

The perceptron trick

- Note that the quantity $y - \hat{y}$ is 0, –1, and +1 for the three cases in the perceptron trick. Thus, we can summarize it as follows:

Procedure:

- The prediction the perceptron makes at the point is $\hat{y} = step(ax_1 + bx_2 + c)$.
- **Return** the perceptron with the following weights and bias:
  - $a' = a + \eta(y - \hat{y})x_1$
  - $b' = b + \eta(y - \hat{y})x_2$
  - $c' = c + \eta(y - \hat{y})$

Perceptron Algorithm

- Next Lecture