Perceptron

Machine Learning Techniques

Dr. Ashish Tendulkar
IIT Madras

Perceptron is a binary classification algorithm.

We will cover

- Five different components of this algorithm, just like any other ML algorithm along with mathematical details.
- **Implementation** from scratch in Python with Numpy.

Perceptron was motivated from neurons

- Invented in 1958 by Frank Rosenblatt and was intended to be a machine, rather than a program.

- Perceptron was meant to be a rough model of how individual neurons work in the brain.

Let's look at the first component that is training data, which is very similar to other classification algorithms.

Training Data

- Feature matrix: $\mathbf{X}_{n \times m}$
- Label vector: $\mathbf{y}_{n \times 1}$

Note that perceptron can solve only binary classification problems. Hence $y^{(i)} \in \{-1, +1\}$

Let's look at the second component that is model, which is inspired from brain neurons.

Model $h_{\mathbf{w}}(\mathbf{x})$

Weight vector

Non-linear activation function

Feature transformation

$$y = f(\underbrace{\mathbf{w}^T \phi(\mathbf{x})}_{\text{Linear combination of features}})$$

Label

Feature vector

Linear combination of features

Model $h_{\mathbf{w}}(\mathbf{x})$

$$y = f(\mathbf{w}^T \phi(\mathbf{x})) = f(z)$$

where, $f(.)$ is a non-linear activation function. Here we use sign or threshold function as $f(.)$:

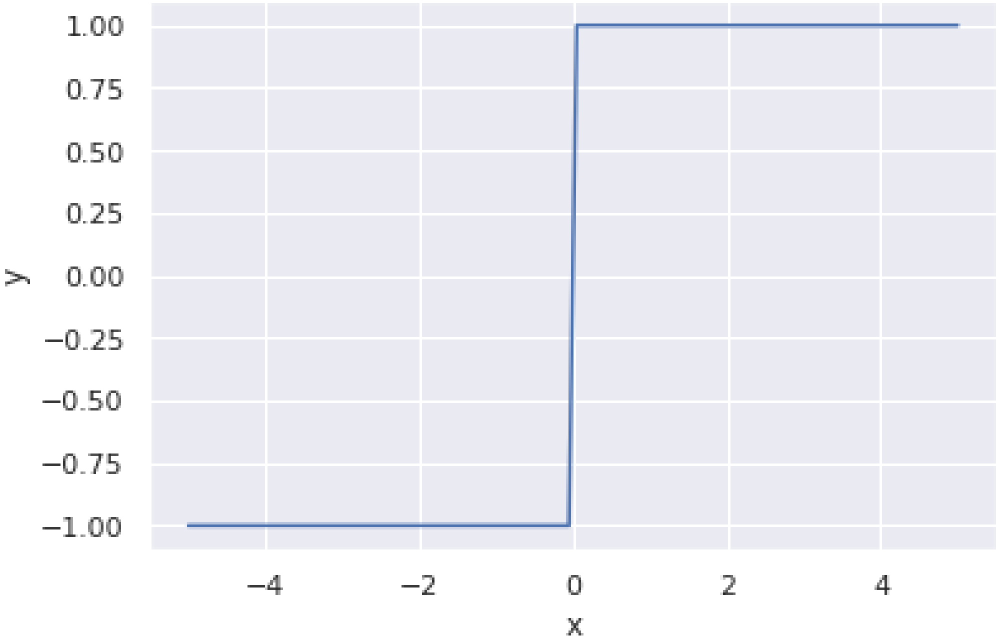
$$f(z) = \begin{cases} +1, \text{ if } z \geq 0 \\ -1, \text{ otherwise (i.e. } z < 0) \end{cases}$$

It can also be written as

$$y = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}))$$

Model visualization

Remember $y = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}))$



Note that for values of $x < 0$, we have $y = -1$

And for values of $x \geq 0$, we have $y = +1$

Now that we have seen the model of perceptron, let's look at the third component that is loss function.

Loss function

Let $\widehat{y}^{(i)} \in \{-1, +1\}$ be the prediction from perceptron and $y^{(i)}$ be the actual label for i-th example. The error $e^{(i)}$ is calculated as

$$e^{(i)} = \begin{cases} 0, \text{ if } \widehat{y}^{(i)} = y^{(i)} \\ -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}, \text{ otherwise (i.e. } \widehat{y}^{(i)} \neq y^{(i)}) \end{cases}$$

For correctly classified examples, the error is 0

For misclassified examples, the error is $-\mathbf{w}^T \phi(x^{(i)}) y^{(i)}$

Loss function

The error can be compactly written as:

$$
\begin{aligned}
e^{(i)} &= \max(0, -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}) \\
&= \max(0, -h_{\mathbf{w}}(x^{(i)}) y^{(i)})
\end{aligned}
$$

Loss function Illustration

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) : y = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{x}^{(i)})\right)$ is either +1 or -1.

If the decision is correct, then

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = +1$ and $y^{(i)} = +1$ or : $h_{\mathbf{w}}(\mathbf{x}^{(i)}) = -1$ and $y^{(i)} = -1$

$$
\begin{aligned}
e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\
&= \max(0, -(1 \times 1)) \\
&= \max(0, -1) \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\
&= \max(0, -(-1 \times -1)) \\
&= \max(0, -1) \\
&= 0
\end{aligned}
$$

Note that in both the cases the error is 0.

Loss function Illustration

If the decision is wrong, then

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = +1$ and $y^{(i)} = -1$

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = -1$ and $y^{(i)} = +1$:

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(+1 \times -1)) \\ &= \max(0, 1) = 1 \end{aligned}$$

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(-1 \times +1)) \\ &= \max(0, 1) = 1 \end{aligned}$$

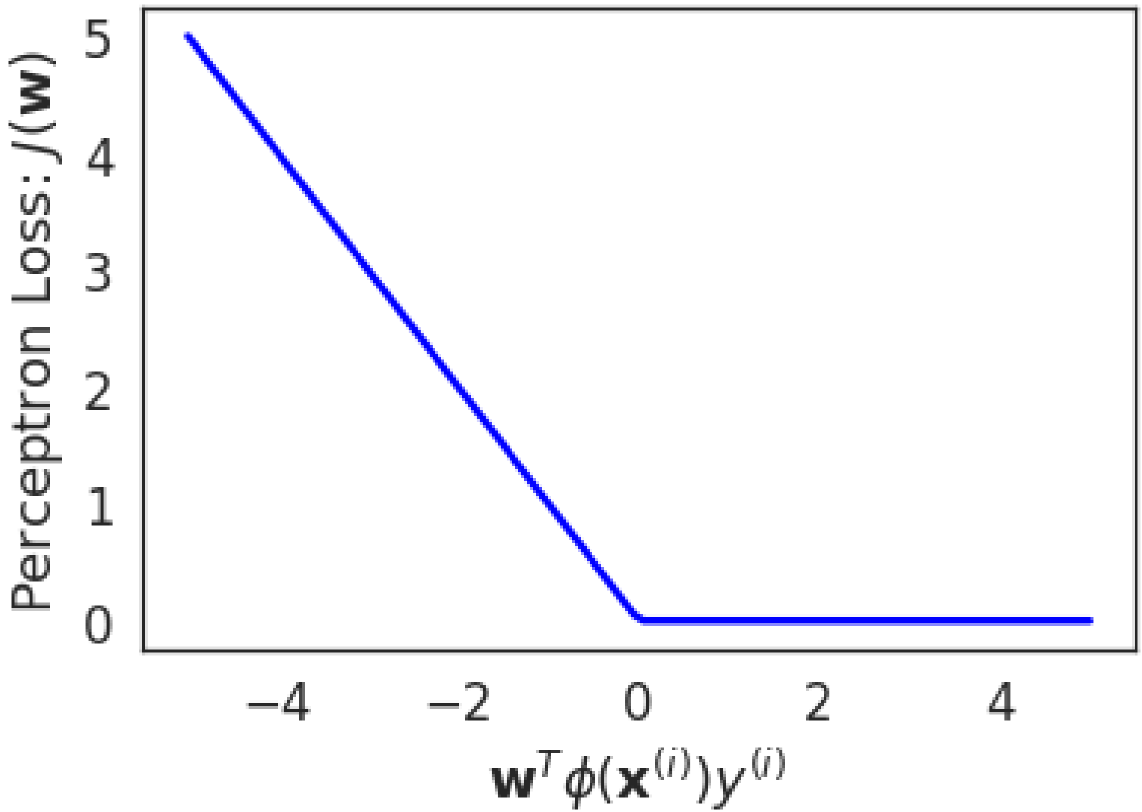
Note that in both the cases the error is 1.

Loss function

$$
\begin{aligned}
J(\mathbf{w}) &= \sum_{i=1}^{n} e^{(i)} \\
&= \sum_{i=1}^{n} \max(0, -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}) \\
&= \sum_{i=1}^{n} \max(0, -y^{(i)} h_{\mathbf{w}}(x^{(i)}))
\end{aligned}
$$

The error is a piecewise linear function: it is zero in the correctly classified regions and a linear function of $\mathbf{w}$ in mis-classified region.

$J(\mathbf{w})$ is not differentiable in $\mathbf{w}$.



We can control the loss is by adjusting the value of $\mathbf{w}$.

The loss is directly proportional to $\mathbf{w}$.

Thus for misclassified example, we can reduce loss by reducing $\mathbf{w}$.

And for correctly classified examples, we leave $\mathbf{w}$ unchanged.

The next task is to obtain the weight vector that minimizes the loss.

We will look at the optimization procedure used in perceptron. This procedure is known as perceptron update rule.

Optimization procedure

1. Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2. For each training example $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$:

$$\widehat{y}^{(i)} = \operatorname{sign}\left(\mathbf{w}^T \phi(\mathbf{x}^{(i)})\right)$$

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \left(y^{(i)} - \widehat{y}^{(i)}\right) \phi(\mathbf{x}^{(i)})$$

Note for correctly classified examples, $\left(y^{(i)} - \widehat{y}^{(i)}\right)$ = 0 and hence there is no change in the weight vector.

Linear separable examples lead to convergence of the algorithm with zero training loss, else it oscillates.

Let's understand perceptron update rule for various values of $\hat{y}^{(i)}$ and $y^{(i)}$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right) \phi(\mathbf{x}^{(i)})$$

(Case 1: Correct classification) $\hat{y}^{(i)} = y^{(i)}$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times 0 \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} + 0 = \mathbf{w}^{(t)}$$

(Case 2: Negative class misclassification) $y^{(i)} = -1$ and $\hat{y}^{(i)} = 1$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times (-1 - 1) \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} - 2\alpha\phi(\mathbf{x}^{(i)})$$

(Case 3: Positive class misclassification) $y^{(i)} = 1$ and $\hat{y}^{(i)} = -1$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times (1 - (-1)) \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} + 2\alpha\phi(\mathbf{x}^{(i)})$$

Optimization procedure: Convergence

- On linearly separable data, the optimization procedure eventually converges.

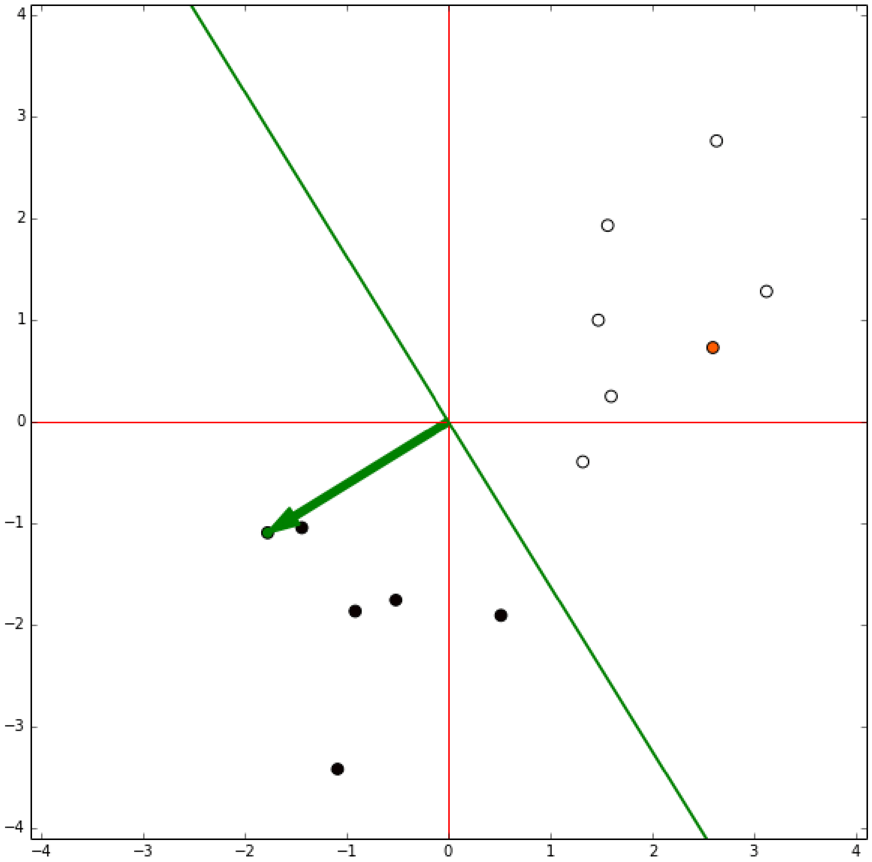
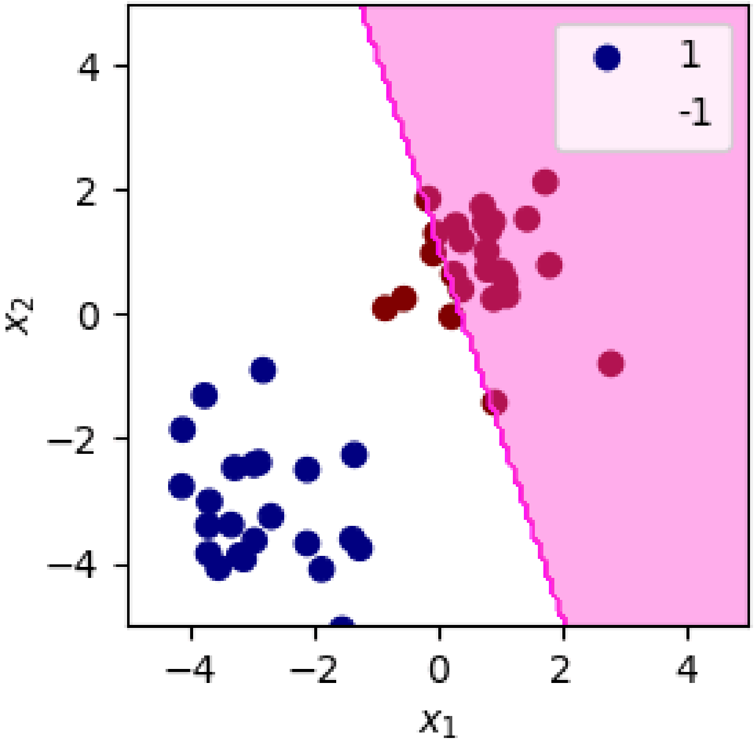
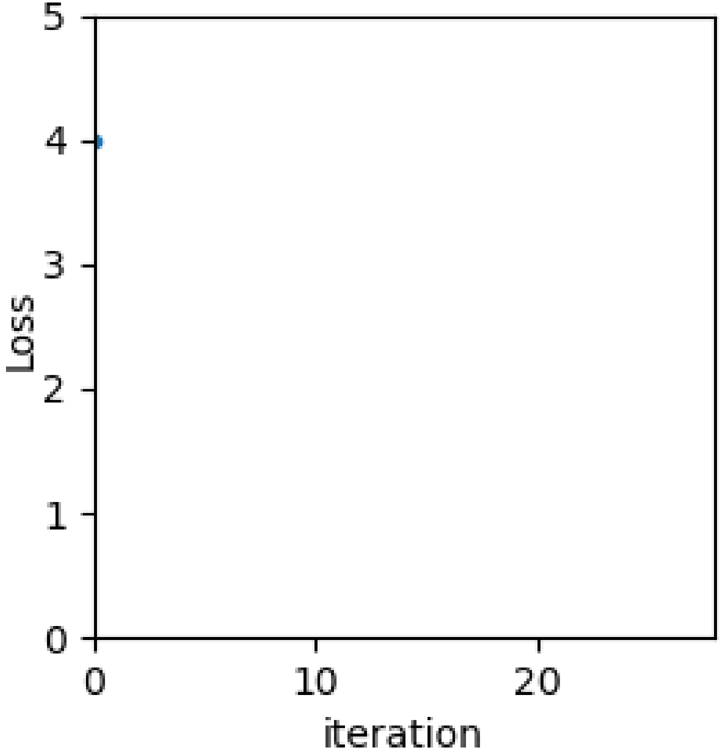


Image source: Wikipedia.org

Optimization procedure: Convergence

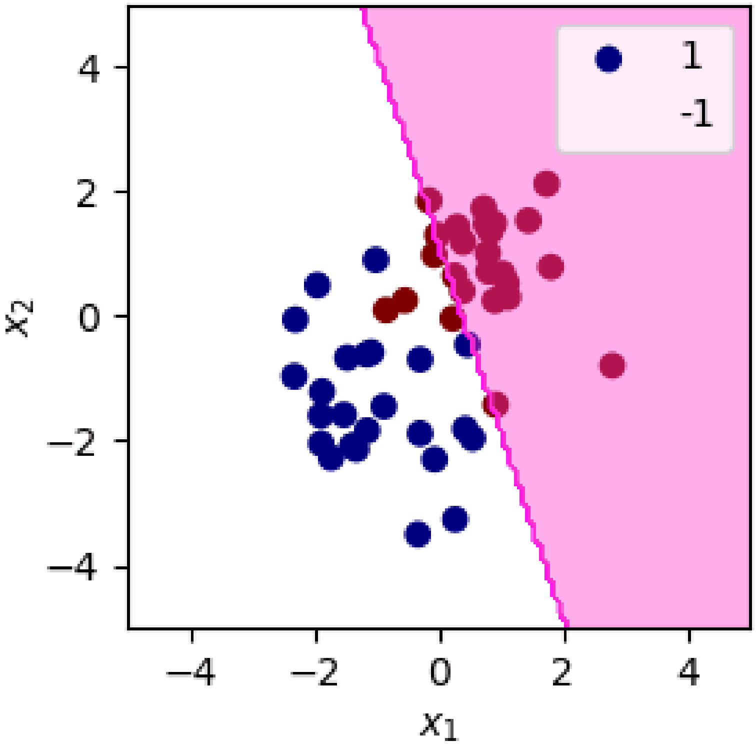
- On linearly separable data, the optimization procedure converges.

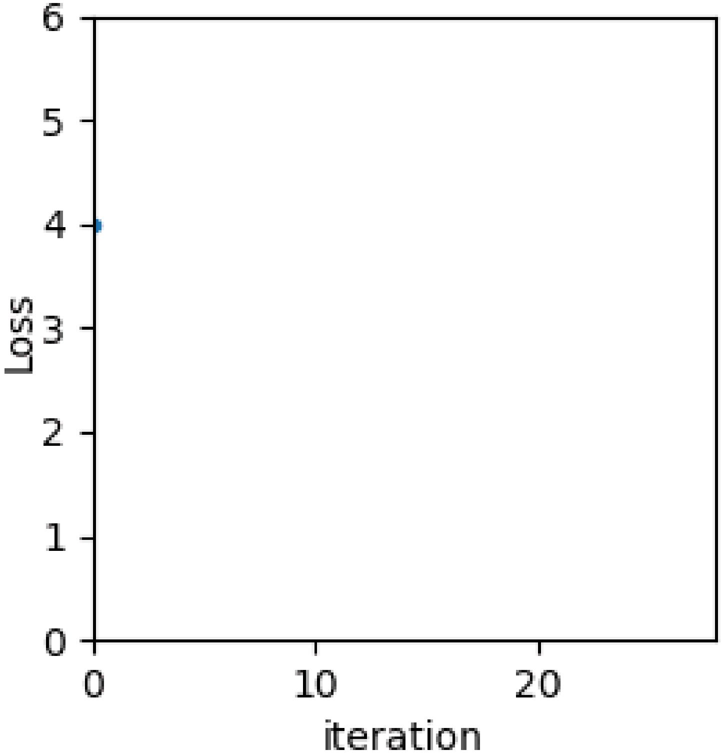




Optimization procedure: Oscillations

- On **non** linearly separable data, the optimization procedure **never** converges.



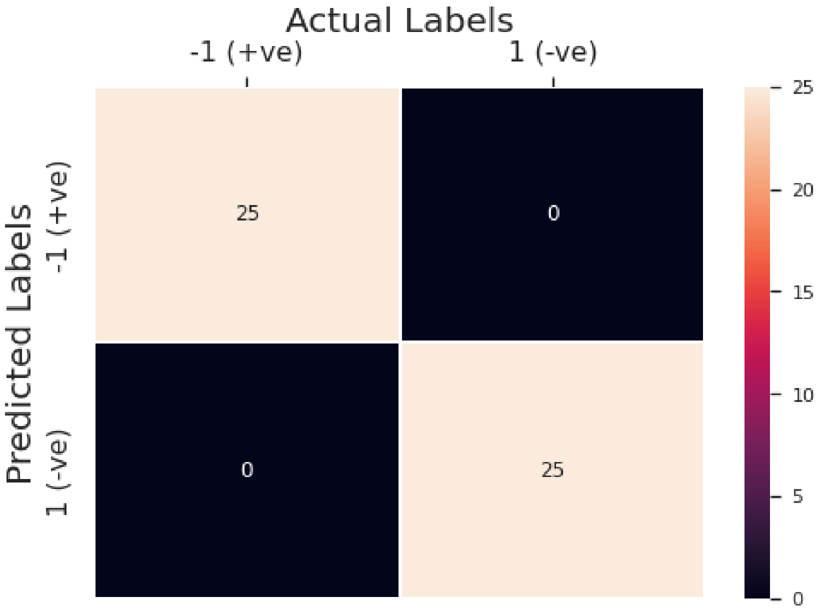


Finally let's look at the evaluation metrics.
They are same as other classification algorithms.

Evaluation metrics

- Calculate confusion matrix based on predicted and actual labels.
- Calculate classification metrics like precision, recall, accuracy, F1-score from confusion matrix.

Confusion matrix and evaluation metrics: Example



- TP = 25
- FP = 0
- TN = 25
- FN = 0

Precision (P) = $\frac{TP}{TP+FP} = \frac{25}{25+0} = 1.0$

Recall (R) = $\frac{TP}{TP+FN} = \frac{25}{25+0} = 1.0$

F1 Score = $\frac{2 \times P \times R}{P+R} = \frac{2 \times 1 \times 1}{1+1} = 1.0$

Confusion matrix implementation

```python
def compute_confusion_matrix(y, y_predicted):

  # Create a 2D matrix of size n by n
  confusion_matrix = np.zeros((n,n))

  # Populate entries of confusion matrix
  for i_x,i_y in zip(y_predicted, y):
    confusion_matrix[i_x,i_y] +=1

  return confusion_matrix
```