A Descriptive Framework of Stories of Algorithms

Supplementary Material

J. Liem, R. Henkin, J. Wood, and C. Turkay
City, University of London

Example 1: A visual introduction to machine learning

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Example 2: The Beginner's Guide to Dimensionality Reduction

https://idyll-lang.org/gallery/the-beginner-s-guide-to-dimensionality-reduction

Example 3: t-SNE explained in plain javascript

https://observablehq.com/@nstrayer/t-sne-explained-in-plain-javascript

Example 4: Roads from Above

https://roadsfromabove.netlify.com/

Legend

Elements of algorithms and models

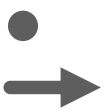
Input

Logic

Parameters

Output

Progress depiction



Snapshot

Continuous

Thematic focus

Focus

Support

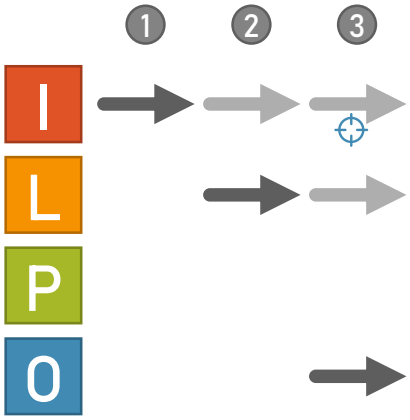
Sequencing of parts of a story

Element linkage

Indication of sources and targets of transformations

A visual introduction to machine learning

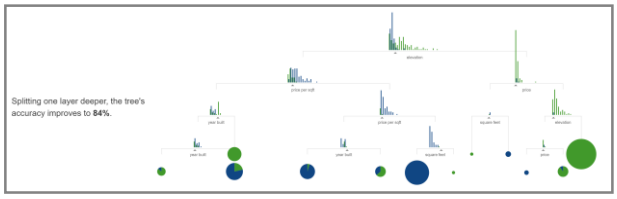
http://www.r2d3.us/visual-intro-to-machine-learning-part-1/



1 I

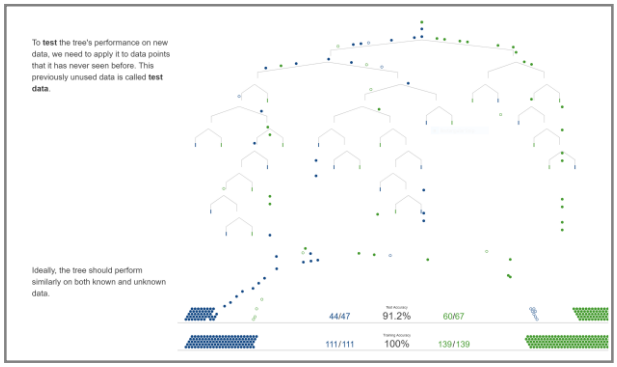
Explaining input records and features.
The progression is continuously represented.

2 I L



Building the decision tree.
The progression of building the tree is continuously represented. The input has a supporting role.

3 I L O

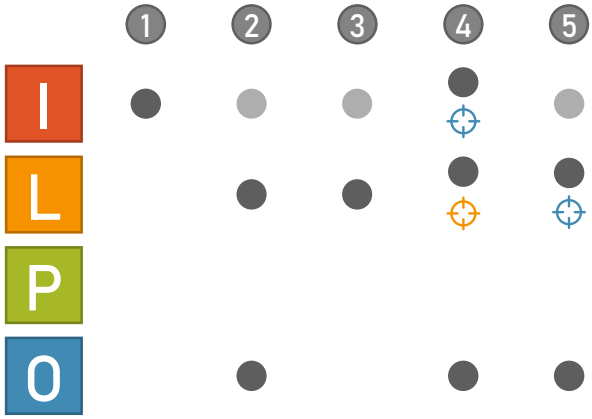


Running the model with a training set and a test set.
Defined by the author, first the user can see how the training set is used to run the model. Then an input change has an effect on the output in an additional model run.

The example follows a logic build up pattern structuring the explanation along the logical order of algorithmic execution.

The Beginner's Guide to Dimensionality Reduction (1)

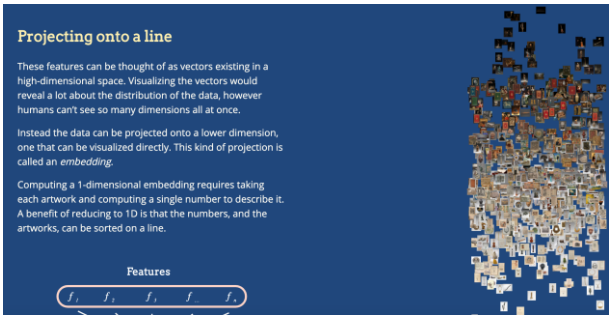
https://idyll-lang.org/gallery/the-beginner-s-guide-to-dimensionality-reduction



1 I

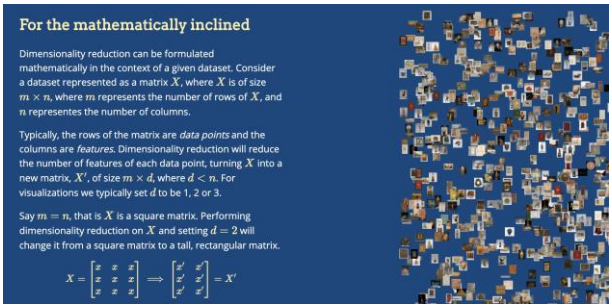
Explaining the records and features used as input.
A snapshot of the input that will be used in an algorithm is presented.

2 I L O



Explaining the logic and output of the algorithm.
A simple model maps the input (in a supporting role) to the output. The text also includes an overview of the logic used.

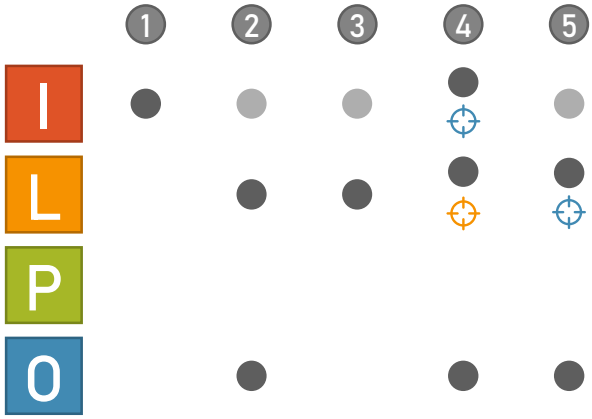
3 I L



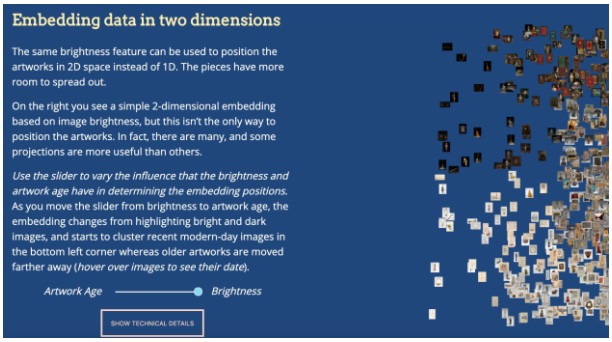
Explaining the logic of the algorithm.
The input is still present in a supporting role, yet in this case the output is not related to the algorithm. The focus is on the textual explanation of the algorithm.

The Beginner's Guide to Dimensionality Reduction (2)

https://idyll-lang.org/gallery/the-beginner-s-guide-to-dimensionality-reduction

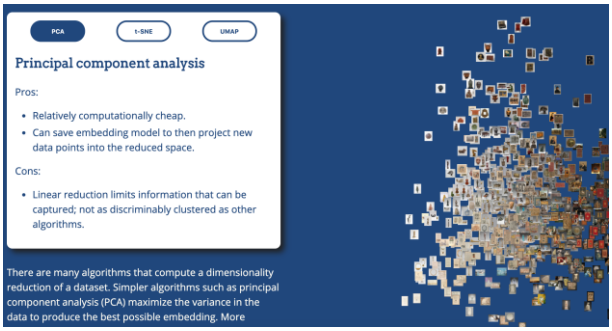


4 I L O



Explaining the relationship between input and output.
A snapshot of the output can be modified by interacting with the input. Here, an optional interaction stage with the underlying mechanism enables inspecting it (as seen in the orange target above).

5 I L O

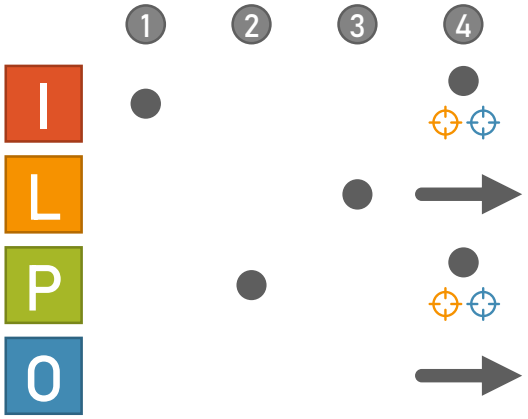


Explaining the logic of the algorithm.
Here it's possible to explore different implementations of dimensionality reduction algorithms. Input is present in a supporting role, whereas interacting with the logic affects the output.

The story has a logic complexity build up pattern, where the complexity of the algorithm increases from 1D, a simple 2D and finally the choice between three implementations.

t-SNE explained in plain javascript (1)

https://observablehq.com/@nstrayer/t-sne-explained-in-plain-javascript



1 I

With 50 total datapoints or observations, we can calculate the total number of permutations:

Set number of observations in your dataset 50

$n^2 = 50^2 = \boxed{2500 \text{ pairs}}$

Now if we're efficient about it and only look at the combinations of the data we can cut this down. The combination equation is a bit more complicated...

$$\frac{n!}{2!(n-2)!} = \frac{50!}{2 \cdot 48!} \frac{...47 \cdot 48 \cdot 49 \cdot 50}{2 \cdot (...47 \cdot 48)} \frac{49 \cdot 50}{2} \boxed{1225 \text{ pairs}}$$

So we can see that by taking advantage of this combination vs permutation distinction we are saving ourselves **1275** total distance calculations. Kinda nice, especially when the data gets larger.

Explaining how the input affects the algorithm.
An interactive slider helps with a textual explanation.

2 P

What is it?

Perplexity a measure related to the entropy (or dispersed-ness) of the system of points. A valuable way of thinking about perplexity is to think of it as the effective number of neighbors for each point. Aka how many points that the distribution over point `i` capture with a non-trivial probability.

The equation is relatively simple. (*The calculations for $p_{i|j}$ are coming up next, so don't worry about them yet*)...

$$\log(\text{perplexity for point } i) = (\text{entropy for point } i) = \sum_{j \neq i} -p_{j|i} \log(p_{j|i})$$

```
calc_entropy = f(probs)
```

```
function calc_entropy(probs){
 return probs.reduce(
   (sum, p) => sum - (p>1e-7 ? p*log(p): 0), // super small probabilities can be ignored
   0
 )
}
```

Explanation of a parameter.
A textual snapshot represents the parameter.

3 L

Actually running it

Now that we have functions for calculating the cost and gradient for a given mapping we can (*finally*) proceed with the actual algorithm.

The steps we need to do for t-SNE are as follows:

1. Randomly initialize mappings mappings: Y_{init}
2. Calculate the cost and gradient for the current mapping
3. Use the gradient to nudge all our mappings to slightly better postions
4. Repeat 2 and 3 until we've lowered our cost function a satisfactory amount.

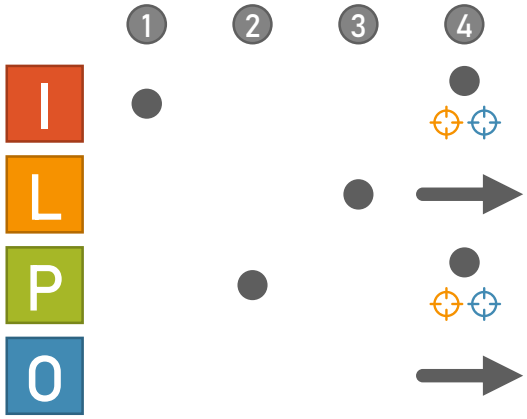
First we initialize a random starting position for the mappings (`Y_new`) and some vectors to help with the gradient descent (which we have not gone into much but it uses momentum and gain to increase performance).

We initialize the points purely randomly with a uniform random number generator, but it would be interesting to compare the performance of initializations using different distributions or patterns (such as the phylotaxisis initialization patterns commonly used in network diagrams.

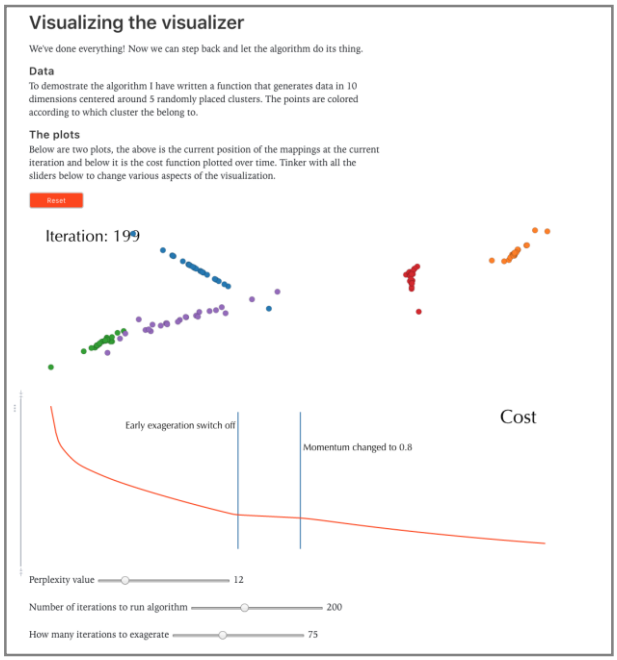
A textual explanation of the logic of the algorithm.

t-SNE explained in plain javascript (2)

https://observablehq.com/@nstrayer/t-sne-explained-in-plain-javascript



4 I L P O

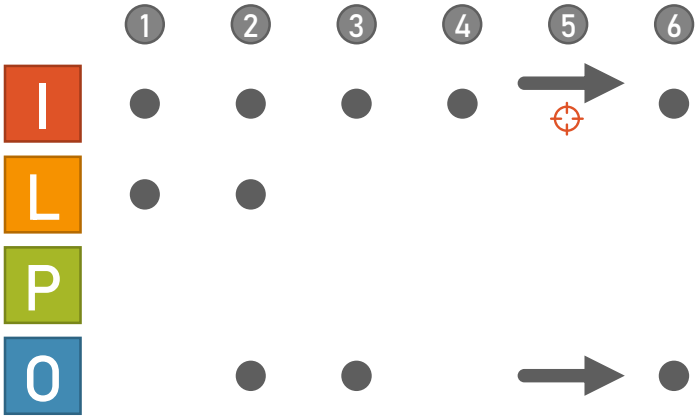


An interactive sandbox combining all elements. Here, input and parameters can be modified with the sliders, while the resulting output is *progressively* shown in the visualization. The lower graphic also contains an output related to the algorithm used – the Cost function.

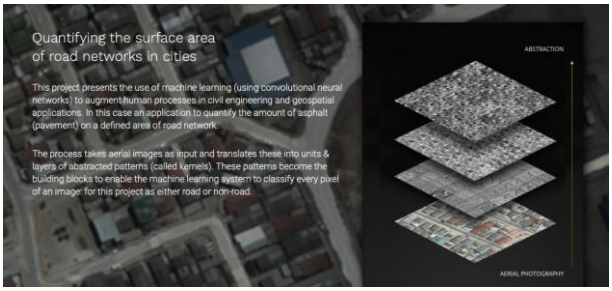
The story has a build up and sandbox pattern: all the elements are described progressively with snapshots, concluding with a *sandbox* area that allows readers to play around with the elements and see the computation of the results.

Roads from Above (1)

https://roadsfromabove.netlify.com/

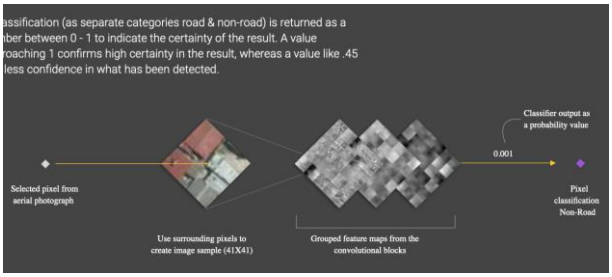


1 I L



Explaining input and intermediate steps of the model.
In two visualisations the input imagery can be compared to different layers of the neural network.

2 I L O

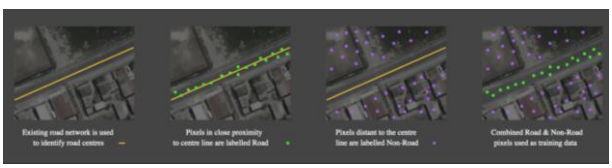


Algorithm overview.
This part explains the basic idea of the algorithm, including input (photo imagery), logic (convolutional neural network) and the output (classification result).

3 I O

Comparing input and output.
A subset of the input pixels can be compared to their output classification in form of snapshots using an interactive swipe tool.

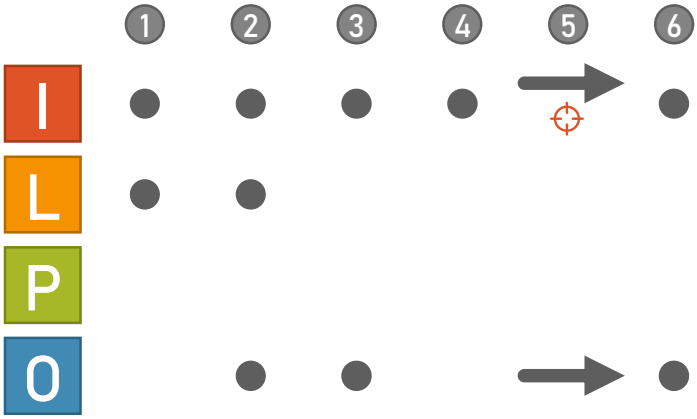
4 I



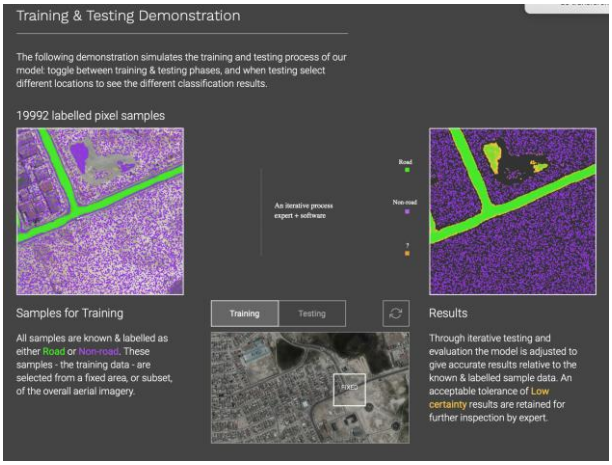
Explaining training set.
This part explains how the training data was generated and labelled.

Roads from Above (2)

https://roadsfromabove.netlify.com/

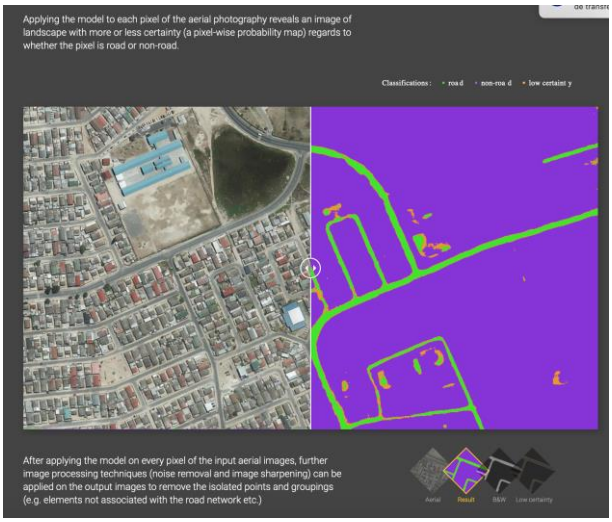


5 I O



From input to output.
An animation shows which input pixels result in which output class. A use can follow the animation for the training set or can select a new test area interactively. The animation then scans the input imagery from top to bottom selecting sample locations, which move from left to right while being applied to a class (watch online!). Compare to the other parts the story this part is presented in a continuous way.

6 I O



Comparing input and output.
Again, using an interactive swipe tool the user can compare the input imagery with the output classification (roads, non-road, low certainty) in the manner of snapshots

The story follows an input-effect pattern, where each part of the story explains how the input influences other elements such as intermediate stages of the model logic or the model output.