3.1. Example

Let's say we have a Pokemon dataset:

Name	Total	HP	Attack	Defence	Type
Beedrill	395	65	90	40	Poison
Gastly	310	30	35	30	Poison
Pidgey	251	40	45	40	Flying
Wigglytuff	435	140	70	45	Fairy

Two features are categorical. The *Name* column acts as the id, so we'll disregard it when training machine-learning models. However, the *Type* column contains information that is relevant to learning tasks.

To use it, we apply one-hot encoding. As there are three categories, the vectors will have three dimensions. In each dataset row, we replace the *Type* category with an encoded vector that has 1 in the position corresponding to the category and contains zeroes in the other two dimensions:

Name	Total	HP	Attack	Defence	Type-Poison	Type-Flying	Type-Fairy
Beedrill	395	65	90	40	1	0	0
Gastly	310	30	35	30	1	0	0
Pidgey	251	40	45	40	0	1	0
Wigglytuff	435	140	70	45	0	0	1

This way, **we enlarge our data by adding new columns, one per category.** We assign descriptive names to the new dummy columns to better navigate the processed dataset.

3.2. Dimensionality

If we have a categorical column with a lot of categories, one-hot encoding will add an excessive number of new features. That will take a lot of space and make learning algorithms slow.

To address that issue, we can find the top most frequent categories and encode only them. For the rest, we create a special column "other" or ignore them. The exact choice of   depends on our processing power. In practice, we usually go with 10 or 20.

This issue can also occur if we have multiple categorical variables that, in total, produce too many new columns.

4. Advantages and Disadvantages of One-Hot Encoding

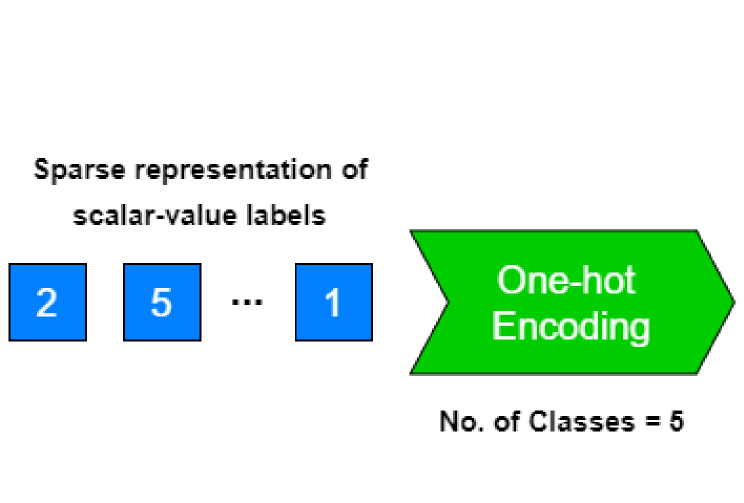
It's relatively straightforward to implement and enables us to apply machine-learning algorithms to data with categorical columns.

The problem is that it **increases dimensionality so training becomes slower and more complex**. It can also create sparse data since most entries in the new columns will be zero. Additionally, one-hot encoding takes more space but adds no new information since it only changes data representation.

Even if there are relatively few categories, one-hot encoding may cause these problems if the data contain a lot of rows.

5. Conclusion

In this article, we explored one-hot encoding and the motivations to apply it. It enables us to use standard machine-learning algorithms on categorical data but may increase dimensionality and slow down the training.



We need to convert scalar-value labels into a ***one-hot vector*** before using them in deep learning models.

This is required for multiclass classification models that output probabilities per class when using the ***categorical_crossentropy*** loss function.

Sparse scalar representation

The values in the label column are usually represented as sparse scalars (i.e. in single-digit format). For example, the training and test labels in the MNIST digits dataset are represented in single-digit format ranging from 0 to 9. Each digit represents a class label.

```
Training labels
[5 0 4 ... 5 6 8]
(60000,)
<class 'numpy.ndarray'>
-----------------------
Test labels
[7 2 1 ... 4 5 6]
(10000,)
<class 'numpy.ndarray'>
```

The training and test labels are in two separate one-dimensional vectors.

One-hot representation

As I explained at the beginning of this article, sparse scalar representation is not suitable for multiclass classification models that output probabilities per class. So, it is necessary to perform one-hot encoding for scalar-value labels before using them in deep learning models.

After performing one-hot encoding, we get a one-hot vector.

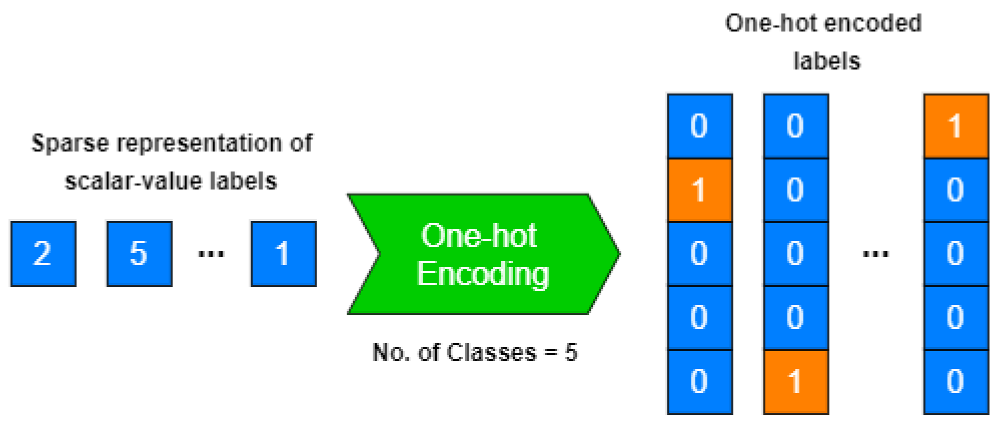
In one-hot encoding, we need an ***n-element vector*** (**n** is the number of classes) for each scalar-value label. All elements in the vector are zeros (denote 0% probability) except the one that we use to mark the index of the scalar value. For example, the one-hot representation of the scalar-value 5 can be repeated as in the following when the number of classes is 10.

```
#5
[0,0,0,0,0,1,0,0,0,0]
```

The value 1 at the fifth index (indices is zero-based) denotes 100% probability for the label value at that index. That’s digit 5.

We get that type of vector for each scalar-value label and stack them vertically to get the final one-hot vector.

Here is an example of a one-hot vector of three scalar-value labels when the number of classes is 5.



After performing one-hot encoding, the training and test labels in the MNIST digits dataset look as follows.

```
Training labels
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]]
(60000, 10)
<class 'numpy.ndarray'>
----------------------
Test labels
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
(10000, 10)
<class 'numpy.ndarray'>
```

Now, the training and test labels are in two separate two-dimensional matrices.