Encoding Categorical Data

There are three common approaches for converting ordinal and categorical variables to numerical values. They are:

- Ordinal Encoding
- One-Hot Encoding
- Dummy Variable Encoding

Let's take a closer look at each in turn.

Ordinal Encoding

In ordinal encoding, each unique category value is assigned an integer value.

For example, "*red*" is 1, "*green*" is 2, and "*blue*" is 3.
This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

For some variables, an ordinal encoding may be enough. The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.

It is a natural encoding for ordinal variables. For categorical variables, it imposes an ordinal relationship where no such relationship may exist. This can cause problems and a one-hot encoding may be used instead.

This ordinal encoding transform is available in the scikit-learn Python machine learning library via the OrdinalEncoder class.
By default, it will assign integers to labels in the order that is observed in the data. If a specific order is desired, it can be specified via the "*categories*" argument as a list with the rank order of all expected labels.
We can demonstrate the usage of this class by converting colors categories "red", "green" and "blue" into integers. First the categories are sorted then numbers are applied. For strings, this means the labels are sorted alphabetically and that **blue=0, green=1 and red=2**.

The complete example is listed below.

```
# example of a ordinal encoding
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```

Running the example first reports the 3 rows of label data, then the ordinal encoding.

We can see that the numbers are assigned to the labels as we expected.

```
[['red']
 ['green']
 ['blue']]
[[2.]
 [1.]
 [0.]]
```

This OrdinalEncoder class is intended for input variables that are organized into rows and columns, e.g. a matrix.

If a categorical target variable needs to be encoded for a classification predictive modelling problem, then the LabelEncoder class can be used. It does the same thing as the OrdinalEncoder, although it expects a one-dimensional input for the single target variable.

One-Hot Encoding

For categorical variables where no ordinal relationship exists, the integer encoding may not be enough, at best, or misleading to the model at worst.

Forcing an ordinal relationship via an ordinal encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the ordinal representation. This is where the integer encoded variable is removed and one new binary variable is added for each unique integer value in the variable.

Each bit represents a possible category. If the variable cannot belong to multiple categories at once, then only one bit in the group can be "on." This is called one-hot encoding …

— Page 78, Feature Engineering for Machine Learning, 2018.

In the "*colour*" variable example, there are three categories, and, therefore, three binary variables are needed. A "1" value is placed in the binary variable for the colour and "0" values for the other colours.

This one-hot encoding transform is available in the scikit-learn Python machine learning library via the OneHotEncoder class.

We can demonstrate the usage of the OneHotEncoder on the colour categories. First the categories are sorted, in this case alphabetically because they are strings, then binary variables are created for each category in turn. This means blue will be represented as [1, 0, 0] with a "1" in for the first binary variable, then green, then finally red.

```
# example of a one hot encoding

from numpy import asarray

from sklearn.preprocessing import OneHotEncoder

# define data

data = asarray([['red'], ['green'], ['blue']])

print(data)

# define one hot encoding

encoder = OneHotEncoder(sparse=False)

# transform data

onehot = encoder.fit_transform(data)

print(onehot)
```

Running the example first lists the three rows of label data, then the one hot encoding matching our expectation of 3 binary variables in the order “blue”, “green” and “red”.

[['red']

['green']

['blue']]

[[0. 0. 1.]

[0. 1. 0.]

[1. 0. 0.]]

If you know all of the labels to be expected in the data, they can be specified via the “*categories*” argument as a list.

The encoder is fit on the training dataset, which likely contains at least one example of all expected labels for each categorical variable if you do not specify the list of labels. If new data contains categories not seen in the training dataset, the “*handle_unknown*” argument can be set to “*ignore*” to not raise an error, which will result in a zero value for each label.