Encoding using sklearn

Encoding in sklearn is done using the **preprocessing module** which comes with a variety of options of manipulating data before going into the analysis of data. We will focus on two forms of encoding for now, the **LabelEncoder** and the **OneHotEncoder**.

Label Encoder

First, we have to import the preprocessing library.

```
from sklearn import preprocessing
```

Let's create a dummy dataframe named `data` with a column whose values we want to transform from categories to integers.

```
# creating sample data
sample_data = {'name': ['Ray', 'Adam', 'Jason', 'Varun', 'Xiao'],
'health':['fit', 'slim', 'obese', 'fit', 'slim']}
# storing sample data in the form of a dataframe
data = pandas.DataFrame(sample_data, columns = ['name', 'health'])
```

We have 3 different labels that we are looking to categorize: slim, fit, obese. To do this, we will call `LabelEncoder()` and fit it to the column we are looking to categorize.

```
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(data['health'])
```

Once you have fit the label encoder to the column you want to encode, you can then transform that column to integer data based on the categories found in that column. That can be done as follows:

```
label_encoder.transform(data['health'])
```

This will give you the output:

```
array([0, 2, 1, 0, 2])
```

You can combine the `fit` and `transform`statements above by using `label_encoder.fit_transform(data['health'])`.

The string categorical health data has been mapped as follows:

```
fit : 0
obese: 1
slim: 2
```

One thing to keep in mind when encoding data is the fact that you do not want to skew your analysis because of the numbers that are assigned to your categories. For example, in the above example, slim is assigned a value 2 and obese a value 1. This is not to say that the intention here is to have `slim` be a value that is empirically twice is likely to affect your analysis as compared to `obese`. In such situations it is better to one-hot encode your data as all categories are assigned a `0` or a `1` value thereby removing any unwanted biases that may creep in if you simply label encode your data.

One-hot Encoder

If we were to apply the one-hot transformation to the same example we had above, we'd do it in Pandas using **get_dummies** as follows:

```
pandas.get_dummies(data['health'])
```

We could do this in sklearn on the label encoded data using **OneHotEncoder** as follows:

```
ohe = preprocessing.OneHotEncoder() # creating OneHotEncoder object
label_encoded_data = label_encoder.fit_transform(data['health'])
ohe.fit_transform(label_encoded_data.reshape(-1,1))
```