Machine Learning

|Handling Text and Categorical Attributes

Prof. Dr. Mostafa Elhosseini

https://youtube.com/drmelhosseini

Agenda

- Handling Text and Categorical AttributesMissing Values

Handling Text and Categorical Attributes

- Most Machine Learning algorithms **prefer to work with numbers** anyway, so let's convert these text labels to numbers.
- In this dataset, there is just one: the **ocean_proximity** attribute.

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(8)
```

	ocean_proximity
13096	NEAR BAY
14973	<1H OCEAN
3785	INLAND
14689	INLAND
20507	NEAR OCEAN
1286	INLAND
18078	<1H OCEAN
4396	NEAR BAY

Handling Text and Categorical Attributes

- We can use Scikit-Learn's OrdinalEncoder class.

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
```

```
housing_cat_encoded[:8]
```

```
array([[3.],
       [0.],
       [1.],
       [1.],
       [4.],
       [1.],
       [0.],
       [3.]])
```

Handling Text and Categorical Attributes

- You can get the list of categories using the **categories_ instance** variable.

```
ordinal_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

- One issue with this representation is that ML algorithms will assume that **two nearby values are more similar than two distant values**. This may be fine in some cases (e.g., for **ordered categories** such as “bad”, “average”, “good”, and “excellent”)

Handling Text and Categorical Attributes

- To fix this issue, a common solution is to create one binary attribute per category: one attribute equal to 1 (and 0 otherwise)
  - This is called **one-hot encoding**
  - Only one attribute will be equal to 1 (hot), while the others
  - will be 0 (cold).
  - The new attributes are sometimes called dummy attributes
- Scikit-Learn provides a OneHotEncoder encoder to convert integer categorical values into one-hot vectors.

Handling Text and Categorical Attributes

- Scikit-Learn provides a **OneHotEncoder** class to convert categorical values into one-hot vectors.

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

```
housing_cat_1hot
```

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

Handling Text and Categorical Attributes

- By default, the output of a OneHotEncoder is a **SciPy sparse matrix**, instead of a NumPy array.
- A **sparse** matrix is a matrix in which most of the elements are zero.
- A sparse matrix is a **very efficient representation** for matrices that contain mostly zeros. Indeed, internally it only stores the nonzero values and their positions.
- When a categorical attribute has hundreds or thousands of categories, one-hot encoding it results in a **very large matrix full of 0s** except for a single 1 per row. In this case, a **sparse matrix** is exactly what you need.

Handling Text and Categorical Attributes

- Storing only the non-zero elements significantly reduces memory requirements, especially for very large matrices.
- Operations such as matrix multiplication, addition, and subtraction can be faster if they are optimized to skip the zero elements.
- Sparse matrices are particularly useful in certain domains such as natural language processing, network analysis, and scientific computing

Handling Text and Categorical Attributes

- if you want to convert it to a (dense) NumPy array, just call the **toarray()** method

```
housing_cat_1hot.toarray()
```

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

- Alternatively, you can **set sparse=False** when creating the OneHotEncoder

```
cat_encoder = OneHotEncoder(sparse_output=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

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

Handling Text and Categorical Attributes

- As with the OrdinalEncoder, you can get the list of categories using the encoder's **categories_** instance variable

```
cat_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Handling Text and Categorical Attributes

- Pandas has a function called **get_dummies()**, which also converts each categorical feature into a one-hot representation, with one binary feature per category.

```
df_test = pd.DataFrame({"ocean_proximity": ["INLAND", "NEAR BAY"]})
pd.get_dummies(df_test)
```

	ocean_proximity_INLAND	ocean_proximity_NEAR BAY
0	1	0
1	0	1

- It looks nice and simple, so why not use it instead of OneHotEncoder?

Handling Text and Categorical Attributes

- The **advantage of OneHotEncoder** is that it **remembers** which categories it was trained on. This is very important because once your model is in **production**, it should be **fed exactly the same features as during training**: no more, no less.

```
df_test = pd.DataFrame({"ocean_proximity": ["INLAND", "NEAR BAY"]})
```

```
cat_encoder.transform(df_test)
```

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

Handling Text and Categorical Attributes

- Moreover, if you feed get_dummies() a DataFrame containing an **unknown category** (e.g., "<2H OCEAN"), it will happily generate a column for it:.

```
df_test_unknown = pd.DataFrame({"ocean_proximity": ["<2H OCEAN", "ISLAND"]})
pd.get_dummies(df_test_unknown)
```

	ocean_proximity_<2H OCEAN	ocean_proximity_ISLAND
0	1	0
1	0	1

Handling Text and Categorical Attributes

- **OneHotEncoder** is smarter: it will **detect** the unknown category and **raise an exception**. If you prefer, you can set the handle_unknown hyperparameter to "ignore", in which case it will just represent the unknown category with zeros:

```
df_test_unknown = pd.DataFrame({"ocean_proximity": ["<2H OCEAN", "ISLAND"]})
```

```
cat_encoder.handle_unknown = "ignore"
cat_encoder.transform(df_test_unknown)
```

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

Handling Text and Categorical Attributes

- When you fit any Scikit-Learn estimator using a DataFrame, the **estimator** stores the column names in the **feature_names_in_** attribute. Scikit-Learn then ensures that any DataFrame fed to this estimator after that (e.g., to transform() or predict()) has the same column names

```
cat_encoder.feature_names_in_
```

```
array(['ocean_proximity'], dtype=object)
```

Handling Text and Categorical Attributes

- Transformers also provide a **get_feature_names_out()** method that you can use to build a DataFrame around the transformer's output

```
cat_encoder.get_feature_names_out()
```

```
array(['ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN'], dtype=object)
```

```
df_output = pd.DataFrame(cat_encoder.transform(df_test_unknown),
                         columns=cat_encoder.get_feature_names_out(),
                         index=df_test_unknown.index)
```

```
df_output
```

	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY	ocean_proximity_NEAR OCEAN
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0

Handling Text and Categorical Attributes

- If a categorical attribute has a **large number of possible categories** (e.g., country code, profession, species), then **one-hot encoding** will result in a large number of input features. This may slow down training and degrade performance.
- If this happens, you may want to replace the categorical input with useful numerical features related to the categories: for example, you could replace the **ocean_proximity** feature with the **distance to the ocean** (similarly, a **country code** could be replaced with the **country's population and GDP per capita**.
- Alternative strategies like numerical feature engineering, advanced encoders, and representation learning through embeddings can provide more efficient and potentially more effective solutions for handling such data.

Handling Text and Categorical Attributes

- Instead of one-hot encoding, which creates a binary vector for each category with a size equal to the number of categories (resulting in a large, sparse matrix), embeddings map each category to a dense vector of a much lower dimension. This mapping is not fixed; instead, the values of the embedding vectors are parameters that the neural network learns during training, which can lead to a more expressive and efficient representation of categorical data.
- Unlike one-hot encoding, these vectors are not binary and can contain any real-valued numbers. These numbers are learned during the training process.

Handling Text and Categorical Attributes

- Let's assume that after training, the embeddings for our colors are as follows (these are hypothetical values for the sake of illustration):
  - "Red" is mapped to [0.8, 0.2]
  - "Green" is mapped to [0.1, 0.9]
  - "Blue" is mapped to [0.5, 0.5]