Preparing Categorical Features

Module 2 | Chapter 2 | Notebook 5

In this Notebook we'll continue to prepare the data for logistic regression. This time we'll work on the categorical features in our data set. To be precise, you'll learn about these methods for preparing categorical data series:

- Label encoding
- One-hot encoding

Label encoding

Scenario: Pictaglam, a popular social media platform for sharing photos and videos, has received complaints about fake user accounts. Management at Pictaglam's have asked you to create a machine learning model that would help the platform to distinguish between real accounts and fake accounts.

The file *social_media_train.csv* contains data on real and fake Pictaglam user accounts.

Let's get started by importing the data.

In [1]:
```python
import pandas as pd

df = pd.read_csv("social_media_train.csv", index_col=[0])
df.head()
```

Out[1]:

	fake	profile_pic	ratio_numlen_username	len_fullname	ratio_numlen_fullname	sim_name_username
0	0	Yes	0.27	0	0.0	No match
1	0	Yes	0.00	2	0.0	Partial match
2	0	Yes	0.10	2	0.0	Partial match
3	0	Yes	0.00	1	0.0	Partial match
4	0	Yes	0.00	2	0.0	No match

The code for that looks like this:

Column number	Column name	Type	Description
0	`'fake'`	categorical	Whether the user account is real (`0`) or fake (`1`).

Column number	Column name	Type	Description
1	`'profile_pic'`	categorical	Whether the account has a profile picture (`'Yes'`) or not (`'No'`)
2	`'ratio_numlen_username'`	continuous (`float`)	Ratio of numeric characters in the account username to its length
3	`'len_fullname'`	continuous (`int`)	total number of characters in the user's full name
4	`'ratio_numlen_fullname'`	continuous (`float`)	Ratio of numeric characters in the account username to its length
5	`'sim_name_username'`	categorical	Whether the user's name matches their username completely (`'Full match'`), partially (`'Partial match'`) or not at all (`'No match'`)
6	`'len_desc'`	continuous (`int`)	Number of characters in the account's description
7	`'extern_url'`	categorical	Whether the account description contains a URL (`'Yes'`) or not (`'No'`)
8	`'private'`	categorical	Whether the user's contributions are only visible to their followers (`'Yes'`) or to all Pictaglam users (`'No'`)
9	`'num_posts'`	continuous (`int`)	Number of posts by the account
10	`'num_followers'`	continuous (`int`)	Number of Pictaglam users who follow the account
11	`'num_following'`	continuous (`int`)	Number of Pictaglam users the account is following

Each row of `df` represents a user or user account.

First you should create a `list` called `features_cat` so you can start preparing the categorical features. Store the `df` column names of all the categorical columns apart from `'fake'` as `str` entries in it.

In [2]:
```
features_cat = ['profile_pic', 'sim_name_username', 'extern_url', 'private']
print(features_cat)
```

```
['profile_pic', 'sim_name_username', 'extern_url', 'private']
```

Let's look at a few data points of the categorical features directly.

In [3]:
```
df.loc[:, features_cat].head(10)
```

Out[3]:

	profile_pic	sim_name_username	extern_url	private
0	Yes	No match	No	No
1	Yes	Partial match	No	No
2	Yes	Partial match	No	Yes
3	Yes	Partial match	No	No
4	Yes	No match	No	Yes
5	Yes	Partial match	Yes	No
6	Yes	Partial match	No	No
7	Yes	No match	No	No
8	Yes	Partial match	No	No
9	Yes	No match	Yes	No

As long as the categories are indicated with `str` values such as `'Yes'` or `'No'`, we can't use them for logistic regression. We should represent the categories with numbers like `0` and `1` so we can use them to make predictions.

If a categorical feature only has two categories like `'Yes'` and `'No'`, you can represent one category with `1` and the other with `0`. How many categories do the different categorical features have? Write a `for` loop to go through the categorical features and print the unique entries for each one.

In [4]:
```python
for col_name in features_cat:
    unique_values = df.loc[:, col_name].unique()
    print("\nColumn name: {}\nUnique values: {}".format(col_name, unique_values))
```

```
Column name: profile_pic
Unique values: ['Yes' 'No']

Column name: sim_name_username
Unique values: ['No match' 'Partial match' 'Full match']

Column name: extern_url
Unique values: ['No' 'Yes']

Column name: private
Unique values: ['No' 'Yes']
```

The features `'profile_pic'`, `'extern_url'` and `'private'` all have 2 unique values (`'Yes' and `'No'), while `sim_name_username' has 3 values (`['No match', 'Partial match', 'Full match']`).

Let's start by focusing on the features with only 2 unique values. Replacing each `str` with a number is called *label encoding*. The `my_series.replace()` method is well suited to this (link to documentation).

This takes a `dict`, where the *keys* are the entries to be replaced and the *values* are the new value entries. For the `'profile_pic_encoded'` column, that would look like this:

In [5]:
```
df.loc[:, 'profile_pic_encoded'] = df.loc[:, 'profile_pic'].replace({'Yes': 1, 'No': 0
df.loc[:, 'profile_pic_encoded'].unique()
```

Out[5]:
```
array([1, 0])
```

There is now a `1` where there used to be a `'Yes'`. All `'No'` data points in this column are now `0` data points. We could also swap the two numerical values, this makes no difference to the model. However, it is usual to mark existing characteristics with a `1`. Now use `my_series.replace()` to encode the labels of the other binary categorical features (`'extern_url'` and `'private'`). Name the new `df` columns `'extern_url_encoded'` and `'private_encoded'`.

In [6]:
```
df.loc[:, 'extern_url_encoded'] = df.loc[:, 'extern_url'].replace({'Yes': 1, 'No': 0})
df.loc[:, 'private_encoded'] = df.loc[:, 'private'].replace({'Yes': 1, 'No': 0})
```

Check that the `'profile_pic_encoded'`, `'extern_url_encoded'` and `'private_encoded'` columns really only contain numeric entries (`1` and `0`). Print the unique values of these columns.

In [7]:
```
print("Column name: {}\nUnique values: {}".format("profile_pic_encoded",
                                                  df.loc[:, 'profile_pic_encoded'].uni
print("Column name: {}\nUnique values: {}".format("extern_url_encoded",
                                                  df.loc[:, 'extern_url_encoded'].uniq
print("Column name: {}\nUnique values: {}".format("private_encoded",
                                                  df.loc[:, 'private_encoded'].unique(
```

```
Column name: profile_pic_encoded
Unique values: [1 0]
Column name: extern_url_encoded
Unique values: [0 1]
Column name: private_encoded
Unique values: [0 1]
```

Congratulations! Thanks to label encoding, we can now use the information from the binary categorical columns to make predictions. But what about categorical features with more than two categories? How can we prepare them so we can use them for logistic regression? Let's look at that next.

One-hot encoding

One-hot encoding (*one-hot encoding*) is used to represent categorical features with new binary features that only contain `0` and `1`. Why not just use different numbers for different categories? In the case of the `'sim_name_username'` column, for example, a new `'sim_name_username_encoded'` column might look like this:

- `"Full match"` --> `2`
- `"Partial match"` --> `1`
- `"No match"` --> `0`

However if you used a feature encoded like this for logistic regression, the categories would not be interpreted as categories, but as measured values. The categories of `'sim_name_username'` do actually have an order. So this is ordinal data. In the lesson *Polynomial Regression (Module 1, Chapter 4)*, you saw that sometimes it can be advantageous to interpret them as measured values.

However, if ordinal data only has a few ordered categories, as is the case with `'sim_name_username_encoded'`, we recommend that you interpret the data nominally. This means that we should work on the assumption that these categories are not ordered. This is the approach we are pursuing here. Keep in mind that you can use the knowledge here for really nominal data (e.g. colors, countries, names).

So how can we use numbers to represent a categorical variable with nominal categories containing more than two categories? With one-hot encoding, each category becomes a new feature, which contains `1` for the category and `0` otherwise. Let's have a look at this with an example. Country data in `df_example`.

In [8]:
```
df_example = pd.DataFrame({'country': ["United Kingdom", "Italy", "Germany", "France"]
                           'population': [61113205, 58126212, 82329758, 64057792]})
df_example
```

Out[8]:

	country	population
0	United Kingdom	61113205
1	Italy	58126212
2	Germany	82329758
3	France	64057792

`df_example` contains a categorical (`'country'`) and a continuous column (`'population'`). `'country'` contains four categories. With one-hot encoding, this becomes four columns:

`'country'`	`'country_France'`	`'country_Germany'`	`'country_Italy'`	`'country_United Kingdom'`
`'United Kingdom'`	0	0	0	1
`'Italy'`	0	0	1	0
`'Germany'`	0	1	0	0
`'France'`	1	0	0	0

How do you do this in Python? The module `pdpipe` with its `OneHotEncode()` function is a good choice:

```
import pdpipe as pdp
pdpipe.OneHotEncode(
    columns=list, #column labels to be one-hot encoded
    dummy_na=bool, #add column to indicated NaN (True=Yes, False=No)
    exclude_columns=str or list, #name of categorial columns to be excluded
from encoding
    drop_first=bool #Whether to get k-1 dummies out of k categorical levels
by removing the first level (default: True)
    )
```

`dpipe.OneHotEncode()` uses the `sklearn.preprocessing.OneHotEncoder`, but renames the one-hot encoded columns (even) better. Here's a link to a practical introduction to this module.

Now let's try it out with the example data set `df_example`: Import `pdpipe` as `pdp` and apply its function `OneHotEncode()` to the `country` column in `df_example`. To get a good idea about how this works, we'll set the `drop_first` parameter to `False`. This will introduce collinearity to our features, but for pedagogical reasons we'll accept that in this case. Once you have done this, assign the variable name `onehot` to the function. Then fit the model to the data. Don't forget to assign the output to the original `DataFrame` `df_example`. Then print `df_example`.

In [9]:
```
import pdpipe as pdp
onehot = pdp.OneHotEncode(['country'], drop_first=False)
df_example = onehot.fit_transform(df_example)
df_example
```

Out[9]:

	population	country_France	country_Germany	country_Italy	country_United Kingdom
0	61113205	0	0	0	1
1	58126212	0	0	1	0
2	82329758	0	1	0	0
3	64057792	1	0	0	0

As you can see, the `country` column has been split into four separate columns named `country_France`, `country_Germany`, `country_Italy`, and `country_United Kingdom` and filled with the values `0` or `1`. So `pdp.OneHotEncode()` expects a `DataFrame` and then applies one-hot encoding for each column that has the datatype `None`, `object`, or `category`. If you're more interested in the `pdpipe` module, we recommend having a look at the documentation.

Now use `pdp.OneHotEncode()` to convert the `'sim_name_username'` column of `df` to one-hot encoding. Then print the first 5 rows of `df`.

In [10]:
```
onehot = pdp.OneHotEncode(['sim_name_username'], drop_first=False)
df = onehot.fit_transform(df)
df.head()
```

Out[10]:

	fake	profile_pic	ratio_numlen_username	len_fullname	ratio_numlen_fullname	len_desc	extern_url
0	0	Yes	0.27	0	0.0	53	No
1	0	Yes	0.00	2	0.0	44	No
2	0	Yes	0.10	2	0.0	0	No
3	0	Yes	0.00	1	0.0	82	No
4	0	Yes	0.00	2	0.0	0	No

Congratulations! Most machine learning models can only handle numerical features. Now you know how to turn categorical features into numerical features. Next we can calculate a logistic regression model and see if we can really predict which user accounts are fake.

Remember:

- Label encoding uses `my_df.replace()` to create a `0` / `1` feature from a binary categorical feature
- One-hot encoding uses `pdp.OneHotEncode()` to create a set of new `0` / `1` features from a categorical feature with more than two categories

---

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.

This data set was created by Bardiya Bakhshandeh and licensed under Creative Commons Attribution 3.0 Unported (CC BY 3.0).