2017-01-30

# How to use pd.get_dummies() with the test set

It turns out that Converting categorical data into numbers with Pandas and Scikit-learn has become the most popular article on this site. Let's revisit the topic and look at Pandas' get_dummies() more closely.

Using the function is straightforward - you specify which columns you want encoded and get a dataframe with original columns replaced with one-hot encodings.

```
df_with_dummies = pd.get_dummies( df, columns = cols_to_transform )
```

Naturally, there will be more columns in the new frame. They will have names corresponding to the original column and its values. For example, `car` will be replaced with `car_Audi`, `car_BMW`, `car_Mercedes` etc.

What if the test set is small and some values are absent? Or it has new values not present in the training set, for example *Volkswagen*?

Two solutions come to mind. One is two `pd.concat(( train, test ))`, `get_dummies()` and then split the set back. If columns sets in train and test differ, you can extract and concatenate just the categorical columns to encode.

Another way is to add the missing columns, filled with zeros, and delete any extra columns. For this to work, one first needs a list of original `columns`. We pass the frame to fix and a list of columns to the following function:

```
def add_missing_dummy_columns( d, columns ):
    missing_cols = set( columns ) - set( d.columns )
    for c in missing_cols:
        d[c] = 0
```

By default Python passes non-scalar object by reference - meaning the function operates on the original. And so it modifies `d` in place and returns nothing. This is a matter of taste and can be easily changed.

We also need to remove any extra columns and reorder the remaining ones to match the original setup:

```
def fix_columns( d, columns ):

    add_missing_dummy_columns( d, columns )

    # make sure we have all the columns we need
    assert( set( columns ) - set( d.columns ) == set())

    extra_cols = set( d.columns ) - set( columns )
    if extra_cols:
        print "extra columns:", extra_cols

    d = d[ columns ]
    return d
```

Now for some informal testing.

```
def fix_columns_test():

    n_cols = 4
    n_rows = 5

    columns = [ "col_{}".format( x ) for x in range( n_cols )]

    # create the "new" set of columns
    new_columns = columns[:]            # copy
    new_columns.pop()
    new_columns.append( 'col_new' )

    # create the "new" dataframe
    n = np.random.random(( n_rows, n_cols ))
    d = pd.DataFrame( n, columns = new_columns )

    print d
    print "\n", columns

    fixed_d = fix_columns( d.copy(), columns )
    print "\n", fixed_d

    assert( list( fixed_d.columns ) == columns )
```

By the way, if you happen to be using `get_dummies( ..., drop_first = True )`, you might want to think over the process described above to make sure everything works as expected.



Posted by Zygmunt Z. • 2017-01-30 • basics, code, software

Tweet
« Data in, predictions out
Tuning hyperparams fast with Hyperband »

# Comments



## Recent Posts



## Twitter

Follow @fastml for notifications about new posts.

Status updating...

Follow @fastml

Also check out @fastml_extra for things related to machine learning and data science in general.

## GitHub

Most articles come with some code. We push it to Github.

https://github.com/zygmuntz