How can I one hot encode in Python?

Asked 7 years, 9 months ago Modified 1 year, 4 months ago Viewed 565k times

253

Highly active question. You have enough reputation to answer or unprotect this question.

I have a machine learning classification problem with 80% categorical variables. Must I use one hot encoding if I want to use some classifier for the classification? Can i pass the data to a classifier without the encoding?

I am trying to do the following for feature selection:

1. I read the train file:

```
num_rows_to_read = 10000
train_small = pd.read_csv("../../dataset/train.csv",   nrows=num_rows_to_read)
```

2. I change the type of the categorical features to 'category':

```
non_categorial_features = ['orig_destination_distance',
                           'srch_adults_cnt',
                           'srch_children_cnt',
                           'srch_rm_cnt',
                           'cnt']

for categorical_feature in list(train_small.columns):
    if categorical_feature not in non_categorial_features:
        train_small[categorical_feature] =
train_small[categorical_feature].astype('category')
```

3. I use one hot encoding:

```
train_small_with_dummies = pd.get_dummies(train_small, sparse=True)
```

The problem is that the 3'rd part often get stuck, although I am using a strong machine.

Thus, without the one hot encoding I can't do any feature selection, for determining the importance of the features.

What do you recommend?

python pandas machine-learning one-hot-encoding Edit tags

Share Edit Follow Close Flag Unprotect

edited Aug 31, 2020 at 14:54 — yatu — 87.2k 12 90 143

asked May 18, 2016 at 7:26 — avicohen — 2,947 6 16 16

24 Answers

Sorted by: Reset to default — Date modified (newest first)

0

A simple example using vectorize in numpy and apply example in pandas:

```
import numpy as np

a = np.array(['male','female','female','male'])

#define function
onehot_function = lambda x: 1.0 if (x=='male') else 0.0

onehot_a = np.vectorize(onehot_function)(a)
print(onehot_a)
# [1., 0., 0., 1.]

# ----------------------------------------

import pandas as pd

s = pd.Series(['male','female','female','male'])
onehot_s = s.apply(onehot_function)

print(onehot_s)
# 0    1.0
# 1    0.0
# 2    0.0
# 3    1.0
# dtype: float64
```

Share Edit Follow Flag

answered Sep 27, 2022 at 8:08 — tturbo — 777 6 18

0

Lets assume out of 10 variables, you have 3 categorical variables in your data frame named as cname1, cname2 and cname3. Then following code will automatically create one hot encoded variable in the new dataframe.

```
import category_encoders as ce
encoder_var=ce.OneHotEncoder(cols=
['cname1','cname2','cname3'],handle_unknown='return_nan',return_df=True,use_cat_names=Tru
new_df = encoder_var.fit_transform(old_df)
```

Share Edit Follow Flag

answered Oct 19, 2021 at 11:00 — DSBLR — 583 5 10

139

Much easier to use Pandas for basic one-hot encoding. If you're looking for more options you can use `scikit-learn`.

For basic one-hot encoding with **Pandas** you pass your data frame into the **get_dummies** function.

For example, if I have a dataframe called **imdb_movies**:

	imdbRating	Awards	Language	Plot	Genre	Rated
0	4.9	NaN	English	A young girl and her coach overcome adversity ...	Drama, Family, Sport	PG
1	8.2	Won 4 Oscars. Another 33 wins & 67 nominations.	English	After John Nash, a brilliant but asocial mathe...	Biography, Drama	PG-13
2	5.7	1 win & 1 nomination.	English	A beautiful dancer balances on the razor's edg...	Comedy, Drama, Music	R
3	7.8	1 nomination.	English	In a world rapidly being torn asunder by viole...	Drama	NaN
4	7.6	11 wins & 2 nominations.	English	After his wife dies of cancer, an overworked e...	Drama, Fantasy	PG

...and I want to one-hot encode the Rated column, I do this:

```
pd.get_dummies(imdb_movies.Rated)
```

APPROVED	G	NOT RATED	PASSED	PG	PG-13	R	TV-14	TV-G	TV-MA	TV-PG	TV-Y	TV-Y7	TV-Y7-FV	UNRATED
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This returns a new `dataframe` with a column for every "**level**" of rating that exists, along with either a 1 or 0 specifying the presence of that rating for a given observation.

Usually, we want this to be part of the original `dataframe`. In this case, we attach our new dummy coded frame onto the original frame using "**column-binding**.

We can column-bind by using Pandas **concat** function:

```
rated_dummies = pd.get_dummies(imdb_movies.Rated)
pd.concat([imdb_movies, rated_dummies], axis=1)
```

dbRating	Awards	Language	Plot	Genre	Rated	15	APPROVED	G	NOT RATED	PASSED	PG	PG-13	R	TV-14	TV-G	TV-MA	TV-PG	TV-Y	TV-Y7	TV-Y7-FV	UNRATED
4.9	NaN	English	A young girl and her coach overcome adversity ...	Drama, Family, Sport	PG	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8.2	Won 4 Oscars. Another 33 wins & 67 nominations.	English	After John Nash, a brilliant but asocial mathe...	Biography, Drama	PG-13	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
5.7	1 win & 1 nomination.	English	A beautiful dancer balances on the razor's edg...	Comedy, Drama, Music	R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7.8	1 nomination.	English	In a world rapidly being torn asunder by viole...	Drama	NaN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

We can now run an analysis on our full `dataframe`.

SIMPLE UTILITY FUNCTION

I would recommend making yourself a **utility function** to do this quickly:

```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    return(res)
```

Usage:

```
encode_and_bind(imdb_movies, 'Rated')
```

Result:

dbRating	Awards	Language	Plot	Genre	Rated	15	APPROVED	G	NOT RATED	PASSED	PG	PG-13	R	TV-14	TV-G	TV-MA	TV-PG	TV-Y	TV-Y7	TV-Y7-FV	UNRATED
4.9	NaN	English	A young girl and her coach overcome adversity ...	Drama, Family, Sport	PG	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8.2	Won 4 Oscars. Another 33 wins & 67 nominations.	English	After John Nash, a brilliant but asocial mathe...	Biography, Drama	PG-13	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
5.7	1 win & 1 nomination.	English	A beautiful dancer balances on the razor's edg...	Comedy, Drama, Music	R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7.8	1 nomination.	English	In a world rapidly being torn asunder by viole...	Drama	NaN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Also, as per @pmalbu comment, if you would like the function to **remove the original feature_to_encode** then use this version:

```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    res = res.drop([feature_to_encode], axis=1)
    return(res)
```

You can encode multiple features at the same time as follows:

```
features_to_encode = ['feature_1', 'feature_2', 'feature_3',
                      'feature_4']
for feature in features_to_encode:
    res = encode_and_bind(train_set, feature)
```

Share Edit Follow Flag

edited Oct 5, 2020 at 13:51 — Harry B — 2,874 1 24 44

answered Oct 22, 2018 at 18:07 — Cybernetic — 12.9k 16 96 136

3 I would suggest dropping the original feature_to_encode after you concatenate the one hot ended columns with the original dataframe. – pmalbu Feb 1, 2019 at 22:58

Added this option to answer. Thanks. – Cybernetic Feb 5, 2019 at 22:42

Would it also work with the 'Genre' variable , i.e. when there are more than one description in the column? Would that still be one hot encoding? Sorry, for asking this here, but I am not sure it deserves (yet) another question. – Sapiens Aug 27, 2020 at 22:00

@Sapiens Yes, it would still be considered hot encoding, where each level would be the unique genre combination a movie belongs to. Another option is to hot encode each genre a movie belongs to into the encoded vector (so one movie with three genres would have an encoded vector with three 1s and the rest 0s). – Cybernetic Aug 27, 2020 at 22:25

4 To encode multiple features, the last line should be `train_set = encode_and_bind(train_set, feature)` otherwise `res` will store only the last iteration's edit on the original dataframe – rishabhjha Jul 2, 2021 at 9:41

Approach 1: You can use pandas' `pd.get_dummies`.

Example 1:

```
import pandas as pd
s = pd.Series(list('abca'))
pd.get_dummies(s)
Out[]:
   a  b  c
0  1  0  0
1  0  1  0
2  0  0  1
3  1  0  0
```

Example 2:

The following will transform a given column into one hot. Use prefix to have multiple dummies.

```
import pandas as pd

df = pd.DataFrame({
          'A':['a','b','a'],
          'B':['b','a','c']
        })
df
Out[]:
  A  B
0  a  b
1  b  a
2  a  c

# Get one hot encoding of columns B
one_hot = pd.get_dummies(df['B'])
# Drop column B as it is now encoded
df = df.drop('B',axis = 1)
# Join the encoded df
df = df.join(one_hot)
df
Out[]:
    A  a  b  c
0  a  0  1  0
1  b  1  0  0
2  a  0  0  1
```

Approach 2: Use Scikit-learn

Using a `OneHotEncoder` has the advantage of being able to `fit` on some training data and then `transform` on some other data using the same instance. We also have `handle_unknown` to further control what the encoder does with *unseen* data.

Given a dataset with three features and four samples, we let the encoder find the maximum value per feature and transform the data to a binary one-hot encoding.

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> enc = OneHotEncoder()
>>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
OneHotEncoder(categorical_features='all', dtype=<class 'numpy.float64'>,
       handle_unknown='error', n_values='auto', sparse=True)
>>> enc.n_values_
array([2, 3, 4])
>>> enc.feature_indices_
array([0, 2, 5, 9], dtype=int32)
>>> enc.transform([[0, 1, 1]]).toarray()
array([[ 1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.]])
```

Here is the link for this example: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

Share Edit Follow Flag

edited Aug 31, 2020 at 15:00
yatu
87.2k 12 90 143

answered Sep 2, 2016 at 7:55
Sayali Sonawane
12.4k 5 46 47

39 setting `drop_first=True` with `get_dummies` removes the need to drop the original column separately – OverflowingTheGlass Feb 28, 2018 at 15:14

1 In example 2, is there a way to join the new columns to the dataframe without using join? I'm dealing with a really big dataset and get MemoryError when I try to do that. – J.D May 31, 2018 at 12:27

31 @OverflowingTheGlass- drop-first= True does not remove the original column. It drops the first level of the categorical feature so that you end up with k-1 columns instead of k columns, k being the cardinality of the categorical feature. – Garima Jain Feb 28, 2019 at 13:50

3 the df.join() does not work here, it creates more rows... do not know why though. – Chenxi Zeng Oct 6, 2019 at 21:10

2 df.join() creates more rows for me, so I used pd.concat([alldata, cat_encoded], axis=1) to join the encoded columns with the original dataset – Ajay Bhasy Dec 7, 2020 at 13:17

Try this:

```
!pip install category_encoders
import category_encoders as ce

categorical_columns = [...the list of names of the columns you want to one-hot-encode
...]
encoder = ce.OneHotEncoder(cols=categorical_columns, use_cat_names=True)
df_train_encoded = encoder.fit_transform(df_train_small)
```

df_encoded.head()

The resulting dataframe `df_train_encoded` is the same as the original, but the categorical features are now replaced with their one-hot-encoded versions.

More information on `category_encoders` here.

Share Edit Follow Flag

answered Mar 5, 2020 at 10:03
Andrea Araldo
1,382 14 20

Short Answer

Here is a function to do one-hot-encoding *without* using numpy, pandas, or other packages. It takes a list of integers, booleans, or strings (and perhaps other types too).

```
import typing


def one_hot_encode(items: list) -> typing.List[list]:
    results = []
    # find the unique items (we want to unique items b/c duplicate items will have the
same encoding)
    unique_items = list(set(items))
    # sort the unique items
    sorted_items = sorted(unique_items)
    # find how long the list of each item should be
    max_index = len(unique_items)

    for item in items:
        # create a list of zeros the appropriate length
        one_hot_encoded_result = [0 for i in range(0, max_index)]
        # find the index of the item
        one_hot_index = sorted_items.index(item)
        # change the zero at the index from the previous line to a one
        one_hot_encoded_result[one_hot_index] = 1
        # add the result
        results.append(one_hot_encoded_result)

    return results
```

Example:

```
one_hot_encode([2, 1, 1, 2, 5, 3])

# [[0, 1, 0, 0],
#  [1, 0, 0, 0],
#  [1, 0, 0, 0],
#  [0, 1, 0, 0],
#  [0, 0, 0, 1],
#  [0, 0, 1, 0]]


one_hot_encode([True, False, True])

# [[0, 1], [1, 0], [0, 1]]


one_hot_encode(['a', 'b', 'c', 'a', 'e'])

# [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [1, 0, 0, 0], [0, 0, 0, 1]]
```

Long(er) Answer

I know there are already a lot of answers to this question, but I noticed two things. First, most of the answers use packages like numpy and/or pandas. And this is a good thing. If you are writing production code, you should probably be using robust, fast algorithms like those provided in the numpy/pandas packages. But, for the sake of education, I think someone should provide an answer which has a transparent algorithm and not just an implementation of someone else's algorithm. Second, I noticed that many of the answers do not provide a robust implementation of one-hot encoding because they do not meet one of the requirements below. Below are some of the requirements (as I see them) for a useful, accurate, and robust one-hot encoding function:

A one-hot encoding function must:

- handle list of various types (e.g. integers, strings, floats, etc.) as input
- handle an input list with duplicates
- return a list of lists corresponding (in the same order as) to the inputs
- return a list of lists where each list is as short as possible

I tested many of the answers to this question and most of them fail on one of the requirements above.

Share Edit Follow Flag

answered Feb 5, 2020 at 2:04
Floyd
2,362 20 25

Expanding @Martin Thoma's answer

```
def one_hot_encode(y):
    """Convert an iterable of indices to one-hot encoded labels."""
    y = y.flatten() # Sometimes not flattened vector is passed e.g (118,1) in these
cases
    # the function ends up creating a tensor e.g. (118, 2, 1). flatten removes this
issue
    nb_classes = len(np.unique(y)) # get the number of unique classes
    standardised_labels = dict(zip(np.unique(y), np.arange(nb_classes))) # get the
class labels as a dictionary
    # which then is standardised. E.g imagine class labels are (4,7,9) if a vector of y
containing 4,7 and 9 is
    # directly passed then np.eye(nb_classes)[4] or 7,9 throws an out of index error.
    # standardised labels fixes this issue by returning a dictionary;
    # standardised_labels = {4:0, 7:1, 9:2}. The values of the dictionary are mapped to
keys in y array.
    # standardised labels also removes the error that is raised if the labels are
floats. E.g. 1.0; element
    # cannot be called by an integer index e.g y[1.0] - throws an index error.
    targets = np.vectorize(standardised_labels.get)(y) # map the dictionary values to
array.
    return np.eye(nb_classes)[targets]
```

Share Edit Follow Flag

answered Dec 29, 2019 at 12:36
mcagriardic
71 8

You can do the following as well. Note for the below you don't have to use `pd.concat` .

```
import pandas as pd
# intialise data of lists.
data = {'Color':['Red', 'Yellow', 'Red', 'Yellow'], 'Length':[20.1, 21.1, 19.1, 18.1],
        'Group':[1,2,1,2]}

# Create DataFrame
df = pd.DataFrame(data)

for _c in df.select_dtypes(include=['object']).columns:
    print(_c)
    df[_c]  = pd.Categorical(df[_c])
df_transformed = pd.get_dummies(df)
df_transformed
```

You can also change explicit columns to categorical. For example, here I am changing the `Color` and `Group`

```
import pandas as pd
# intialise data of lists.
data = {'Color':['Red', 'Yellow', 'Red', 'Yellow'], 'Length':[20.1, 21.1, 19.1, 18.1],
        'Group':[1,2,1,2]}

# Create DataFrame
df = pd.DataFrame(data)
columns_to_change = list(df.select_dtypes(include=['object']).columns)
columns_to_change.append('Group')
for _c in columns_to_change:
    print(_c)
    df[_c]  = pd.Categorical(df[_c])
df_transformed = pd.get_dummies(df)
df_transformed
```

Share Edit Follow Flag

edited Aug 30, 2019 at 4:23

answered Aug 30, 2019 at 4:18
sushmit
4,487 2 37 39

It can and it should be easy as :

```
class OneHotEncoder:
    def __init__(self,optionKeys):
        length=len(optionKeys)
        self.__dict__={optionKeys[j]:[0 if i!=j else 1 for i in range(length)] for j in
range(length)}
```

Usage :

```
ohe=OneHotEncoder(["A","B","C","D"])
print(ohe.A)
print(ohe.D)
```

Share Edit Follow Flag

answered May 24, 2019 at 5:07

Ofek Ron
8,472 13 55 105

You can pass the data to catboost classifier without encoding. Catboost handles categorical variables itself by performing one-hot and target expanding mean encoding.

3

Share Edit Follow Flag

answered Feb 28, 2019 at 13:53
Garima Jain
1,267 9 6

True but you have to inform catboost first which features are categorical as the algorithm cannot figure them out by itself. – user3103059 Oct 25, 2020 at 16:55

I used this in my acoustic model: probably this helps in ur model.

1

```
def one_hot_encoding(x, n_out):
    x = x.astype(int)
    shape = x.shape
    x = x.flatten()
    N = len(x)
    x_categ = np.zeros((N,n_out))
    x_categ[np.arange(N), x] = 1
    return x_categ.reshape((shape)+(n_out,))
```

Share Edit Follow Flag

edited Dec 10, 2018 at 9:53
remykarem
2,319 24 28

answered Jun 12, 2018 at 18:41
yunus
2,495 1 15 12

This works for me:

1

```
pandas.factorize( ['B', 'C', 'D', 'B'] )[0]
```

Output:

```
[0, 1, 2, 0]
```

Share Edit Follow Flag

answered Dec 6, 2018 at 16:24
scottlittle
19.8k 8 53 72

I know I'm late to this party, but the simplest way to hot encode a dataframe in an automated way is to use this function:

2

```
def hot_encode(df):
    obj_df = df.select_dtypes(include=['object'])
    return pd.get_dummies(df, columns=obj_df.columns).values
```

Share Edit Follow Flag

answered Sep 25, 2018 at 23:33
Rambatino
4,824 1 34 57

---

pandas as has inbuilt function "get_dummies" to get one hot encoding of that particular column/s.

9

one line code for one-hot-encoding:

```
df=pd.concat([df,pd.get_dummies(df['column name'],prefix='column
name')],axis=1).drop(['column name'],axis=1)
```

Share Edit Follow Flag

answered Sep 13, 2018 at 4:36
Arshdeep Singh
517 6 7

Here i tried with this approach :

-1

```
import numpy as np
#converting to one_hot



def one_hot_encoder(value, datal):

    datal[value] = 1

    return datal


def _one_hot_values(labels_data):
    encoded = [0] * len(labels_data)

    for j, i in enumerate(labels_data):
        max_value = [0] * (np.max(labels_data) + 1)

        encoded[j] = one_hot_encoder(i, max_value)

    return np.array(encoded)
```

Share Edit Follow Delete Flag

answered Jun 2, 2018 at 23:53
Aaditya Ura
12.3k 7 52 93

Here is a solution using `DictVectorizer` and the Pandas `DataFrame.to_dict('records')` method.

4

```
>>> import pandas as pd
>>> X = pd.DataFrame({'income': [100000,110000,90000,30000,14000,50000],
                      'country':['US', 'CAN', 'US', 'CAN', 'MEX', 'US'],
                      'race':['White', 'Black', 'Latino', 'White', 'White', 'Black']
                     })

>>> from sklearn.feature_extraction import DictVectorizer
>>> v = DictVectorizer()
>>> qualitative_features = ['country','race']
>>> X_qual = v.fit_transform(X[qualitative_features].to_dict('records'))
>>> v.vocabulary_
{'country=CAN': 0,
 'country=MEX': 1,
 'country=US': 2,
 'race=Black': 3,
 'race=Latino': 4,
 'race=White': 5}

>>> X_qual.toarray()
array([[ 0.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  1.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  1.,  1.,  0.,  0.]])
```

Share Edit Follow Flag

edited Mar 2, 2018 at 20:41
SherylHohman
17.2k 17 90 95

answered Sep 6, 2016 at 12:24
Josh Morel
1,241 12 17

How do I remove redundant features with DictVectorizer? – user3085496 Sep 28, 2020 at 20:26

You can do it with `numpy.eye` and a using the array element selection mechanism:

35

```
import numpy as np
nb_classes = 6
data = [[2, 3, 4, 0]]

def indices_to_one_hot(data, nb_classes):
    """Convert an iterable of indices to one-hot encoded labels."""
    targets = np.array(data).reshape(-1)
    return np.eye(nb_classes)[targets]
```

The the return value of `indices_to_one_hot(nb_classes, data)` is now

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

The `.reshape(-1)` is there to make sure you have the right labels format (you might also have `[[2], [3], [4], [0]]`).

Share Edit Follow Flag

edited Aug 12, 2017 at 5:46

answered Mar 18, 2017 at 13:19
Martin Thoma
130k 161 639 988

1 This will not work for OHE of the columns with string value. – Abhilash Awasthi Jul 31, 2017 at 8:18

3 @AbhilashAwasthi Sure ... but why would you expect it to work then? – Martin Thoma Jul 31, 2017 at 8:26

One hot encoding with pandas is very easy:

25

```
def one_hot(df, cols):
    """
    @param df pandas DataFrame
    @param cols a list of columns to encode
    @return a DataFrame with one-hot encoding
    """
    for each in cols:
        dummies = pd.get_dummies(df[each], prefix=each, drop_first=False)
        df = pd.concat([df, dummies], axis=1)
    return df
```

EDIT:

Another way to one_hot using sklearn's `LabelBinarizer` :

```
from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
label_binarizer.fit(all_your_labels_list) # need to be global or remembered to use it
later

def one_hot_encode(x):
    """
    One hot encode a list of sample labels. Return a one-hot encoded vector for each
label.
    : x: List of sample Labels
    : return: Numpy array of one-hot encoded labels
    """
    return label_binarizer.transform(x)
```

Share Edit Follow Flag

edited Jul 11, 2017 at 7:59

answered Mar 1, 2017 at 4:31
Qy Zuo
2,652 26 21

One-hot encoding requires bit more than converting the values to indicator variables. Typically ML process requires you to apply this coding several times to validation or test data sets and applying the model you construct to real-time observed data. You should store the mapping (transform) that was used to construct the model. A good solution would use the `DictVectorizer` or `LabelEncoder` (followed by `get_dummies`. Here is a function that you can use:

3

```
def oneHotEncode2(df, le_dict = {}):
    if not le_dict:
        columnsToEncode = list(df.select_dtypes(include=['category','object']))
        train = True;
    else:
        columnsToEncode = le_dict.keys()
        train = False;

    for feature in columnsToEncode:
        if train:
            le_dict[feature] = LabelEncoder()
        try:
            if train:
                df[feature] = le_dict[feature].fit_transform(df[feature])
            else:
                df[feature] = le_dict[feature].transform(df[feature])

            df = pd.concat([df,
                            pd.get_dummies(df[feature]).rename(columns=lambda x:
feature + '_' + str(x))], axis=1)
            df = df.drop(feature, axis=1)
        except:
            print('Error encoding '+feature)
            #df[feature]  = df[feature].convert_objects(convert_numeric='force')
            df[feature]  = df[feature].apply(pd.to_numeric, errors='coerce')
    return (df, le_dict)
```

This works on a pandas dataframe and for each column of the dataframe it creates and returns a mapping back. So you would call it like this:

```
train_data, le_dict = oneHotEncode2(train_data)
```

Then on the test data, the call is made by passing the dictionary returned back from training:

```
test_data, _ = oneHotEncode2(test_data, le_dict)
```

An equivalent method is to use `DictVectorizer`. A related post on the same is on my blog. I mention it here since it provides some reasoning behind this approach over simply using get_dummies post (disclosure: this is my own blog).

Share Edit Follow Flag

edited May 13, 2017 at 18:26

answered May 12, 2017 at 12:37
Tukeys
31 2

really, encoding is a half of deal, but decoding back as a result of ML is the final goal... personally I do not see yet the convinient way to do this in TensorFlow, sklearn seems more suitable (for conviniency) yet – JeeyCi Mar 23, 2022 at 4:04

To add to other questions, let me provide how I did it with a Python 2.0 function using Numpy:

0

```
def one_hot(y_):
    # Function to encode output labels from number indexes
    # e.g.: [[5], [0], [3]] --> [[0, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0], [0, 0, 0, 1,
0, 0]]

    y_ = y_.reshape(len(y_))
    n_values = np.max(y_) + 1
    return np.eye(n_values)[np.array(y_, dtype=np.int32)]  # Returns FLOATS
```

The line `n_values = np.max(y_) + 1` could be hard-coded for you to use the good number of neurons in case you use mini-batches for example.

Demo project/tutorial where this function has been used: https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition

Share Edit Follow Flag

answered Mar 28, 2017 at 0:04
Guillaume Chevalier
10.1k 9 51 80

You can use numpy.eye function.

17

```
import numpy as np

def one_hot_encode(x, n_classes):
    """
    One hot encode a list of sample labels. Return a one-hot encoded vector for each
label.
    : x: List of sample Labels
    : return: Numpy array of one-hot encoded labels
     """
    return np.eye(n_classes)[x]

def main():
    list = [0,1,2,3,4,3,2,1,0]
    n_classes = 5
    one_hot_list = one_hot_encode(list, n_classes)
    print(one_hot_list)

if __name__ == "__main__":
    main()
```

Result

```
D:\Desktop>python test.py
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.]]
```

Share Edit Follow Flag

edited Mar 18, 2017 at 21:16

answered Mar 18, 2017 at 21:00
Dieter
2,559 1 23 42

2 Did you just copy my answer? – Martin Thoma Jun 3, 2018 at 21:39

@Martin Thoma - I think, I didn't – Dieter Jun 4, 2018 at 10:46

Firstly, easiest way to one hot encode: use Sklearn.

http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

23

Secondly, I don't think using pandas to one hot encode is that simple (unconfirmed though)

Creating dummy variables in pandas for python

Lastly, is it necessary for you to one hot encode? One hot encoding exponentially increases the number of features, drastically increasing the run time of any classifier or anything else you are going to run. Especially when each categorical feature has many levels. Instead you can do dummy coding.

Using dummy encoding usually works well, for much less run time and complexity. A wise prof once told me, 'Less is More'.

Here's the code for my custom encoding function if you want.

```
from sklearn.preprocessing import LabelEncoder

#Auto encodes any dataframe column of type category or object.
def dummyEncode(df):
        columnsToEncode = list(df.select_dtypes(include=['category','object']))
        le = LabelEncoder()
        for feature in columnsToEncode:
            try:
                df[feature] = le.fit_transform(df[feature])
            except:
                print('Error encoding '+feature)
        return df
```

EDIT: Comparison to be clearer:

One-hot encoding: convert n levels to n-1 columns.

```
Index  Animal          Index  cat  mouse
  1     dog              1     0     0
  2     cat       -->    2     1     0
  3    mouse             3     0     1
```

You can see how this will explode your memory if you have many different types (or levels) in your categorical feature. Keep in mind, this is just ONE column.

Dummy Coding:

```
Index  Animal          Index  Animal
  1     dog              1     0
  2     cat       -->    2     1
  3    mouse             3     2
```

Convert to numerical representations instead. Greatly saves feature space, at the cost of a bit of accuracy.

Share Edit Follow Flag

edited May 23, 2017 at 12:26
Community BOT
1 1

answered May 18, 2016 at 7:46
Wboy
2,482 2 26 47

1 1. I have a data set which has 80% categorical variables. To my understanding i must use one hot encoding if i want to use a classifier for this data, else in the case of not doing the one hot encoding the classifier won't treat the categorical variables in the correct way? Is there an option not to encode? 2. If i use pd.get_dummies(train_small, sparse=True) with the saprse=True - doesn't that solves the memory problem? 3. How should i approach such a problem? – avicohen May 18, 2016 at 9:15

As I said, there are two options. 1) One hot encode --> convert every level in categorical features to a new column. 2)Dummy coding --> convert every column to numeric representations. I'll edit my answer above to be clearer. But you can just run the function i provided and it should work – Wboy May 18, 2016 at 9:51

23 "at the cost of a bit of accuracy." How can you say "a bit"? Maybe in some cases, but in others, the accuracy could be hurt a lot. This solution results in treating qualitative features as continuous which means your model will not learn from the data properly. – Josh Morel Sep 6, 2016 at 12:23

6 As Josh said above, in your second example you end up telling the model that `mouse > cat > dog` but this is not the case. `get_dummies` is the most straight forward way of transferring categorical variables into model friendly data from my experience (albeit very limited) – Martin O Leary Jan 16, 2017 at 20:06

12 This solution is very dangerous as pointed out by some other comments. It arbitrarily assigns orders and distances to categorical variables. Doing so reduces model flexibility in a random way. For tree based models, such encoding reduces possible subsetting possibilities. For example, you can only get two possible splittings now [(0), (1,2)] and [(0,1),(2)], and the split [(0,2), (1)] is impossible. The loss is much more significant when the number of categories is high. – Random Certainty Dec 28, 2017 at 1:48

0

This post is hidden. It was deleted 3 years ago by the post author.

For One Hot Encoding I recommend using ColumnTransformer and OneHotEncoder instead of get_dummies. That's because OneHotEncoder returns an object which can be used to encode unseen samples using the same mapping that you used on your training data.

This code encodes all the columns provided in the *del_idxs_before* variable:

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'cat_1': ['A1', 'B1', 'C1'], 'num_1': [100, 200, 300],
                   'cat_2': ['A2', 'B2', 'C2'], 'cat_3': ['A3', 'B3', 'C3'],
                   'label': [1, 0, 0]})

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
del_idxs_before = [0, 2, 3] # Change here
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), del_idxs_before)],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

X:

```
array([[1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 100],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 200],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 300]], dtype=object)
```

To avoid multicollinearity due to the dummy variable trap, I would also suggest to remove one of the columns returned by your encoding. This code encodes all the columns provided in the *del_idxs_before* variable, **AND** it removes the last column of every one hot encoded column:

```
import pandas as pd
import numpy as np

def sum_prev (l_in):
    l_out = []
    l_out.append(l_in[0])
    for i in range(len(l_in)-1):
        l_out.append(l_out[i] + l_in[i+1])
    return [e - 1 for e in l_out]

df = pd.DataFrame({'cat_1': ['A1', 'B1', 'C1'], 'num_1': [100, 200, 300],
                   'cat_2': ['A2', 'B2', 'C2'], 'cat_3': ['A3', 'B3', 'C3'],
                   'label': [1, 0, 0]})

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
del_idxs_before = [0, 2, 3] # Change here
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(sparse=False),
del_idxs)], remainder='passthrough')

del_idxs_after = [df.iloc[:, del_idx].nunique() for del_idx in del_idxs]
del_idxs_after = sum_prev(del_idxs_after)
X = np.array(ct.fit_transform(X))
X = np.delete(X, cols_idxs_to_del, 1)
```

X:

```
array([[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 100],
       [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 200],
       [0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 300]], dtype=object)
```

Share Edit Follow **Undelete** Flag

answered Jun 17, 2020 at 0:34
Matheus Schaly
165 4 23

1 This appears to be a good answer. But you've posted almost identical text to three different questions within a ten minute period. That can start to look spammy. If the same answer works for multiple questions, that's also a good indication that those questions *might* be duplicates of one another, in which case the better solution is to flag them as such, and point to a version of the question that has been answered before. – Jeremy Caney Jun 17, 2020 at 1:08

Comments disabled on deleted / locked posts / reviews

0

This post is hidden. It was deleted 7 years ago by the post author.

I suggest a simpler way than the previous answers that needs nothing but numpy. If y is a one-dimensional numpy array, then:

```
y_one_hot = np.zeros((len(y), np.amax(y)+1))
for i in range(len(y)):
    y_one_hot[i][y[i]] = 1
```

where the number of columns of y_one_hot will be equal to the largest value in y (which presumably is equal to the number of classes of your classification task) and each row of y will consist of all zeros except for the column that corresponds to the value of the respective entry in y, which will be 1.

Share Edit Follow **Undelete** Flag

answered Jan 31, 2017 at 23:54
Luigi
1 1

Comments disabled on deleted / locked posts / reviews