Converting categorical data into numbers with Pandas and Scikit-learn

2014-04-30

Many machine learning tools will only accept numbers as input. This may be a problem if you want to use such tool but your data includes categorical features. To represent them as numbers typically one converts each categorical feature using “one-hot encoding”, that is from a value like “BMW” or “Mercedes” to a vector of zeros and one *1*.

This functionality is available in some software libraries. We load data using Pandas, then convert categorical columns with *DictVectorizer* from scikit-learn.

Pandas is a popular Python library inspired by data frames in R. It allows easier manipulation of tabular numeric and non-numeric data. Downsides: not very intuitive, somewhat steep learning curve. For any questions you may have, Google + StackOverflow combo works well as a source of answers.

UPDATE: Turns out that Pandas has get_dummies() function which does what we’re after.

We’ll use Pandas to load the data, do some cleaning and send it to Scikit-learn’s DictVectorizer. OneHotEncoder is another option. The difference is as follows:

OneHotEncoder takes as input categorical values encoded as integers - you can get them from LabelEncoder.

DictVectorizer expects data as a list of dictionaries, where each dictionary is a data row with column names as keys:

```
[ { 'foo': 1, 'bar': 'z' },
  { 'foo': 2, 'bar': 'a' },
  { 'foo': 3, 'bar': 'c' } ]
```

After vectorizing and saving as CSV it would look like this:

```
foo,bar=z,bar=a,bar=c
1,1,0,0
2,0,1,0
3,0,0,1
```

Notice the column names and that *DictVectorizer* doesn't touch numeric values.

The representation above is redundant, because **to encode three values you need two indicator columns. In general, one needs *d - 1* columns for *d* values**. This is not a big deal, but apparently some methods will complain about collinearity. The solution is to drop one of the columns. It won't result in information loss, because in the redundant scheme with *d* columns one of the indicators must be non-zero, so if two out of three are zeros then the third must be *1*. And if one among the two is positive than the third must be zero.

Pandas

Before

The question is how to convert some columns from a data frame to a list of dicts. First we need a list of column names. Then we create a new data frame containing only these columns. DataFrame has a `to_dict()` method, but the keys are columns. Therefore we transpose the data frame and then call `.to_dict().values()`:

```
cols_to_retain = [ 'a', 'list', 'of', 'categorical', 'column', 'names' ]
cat_df = df[ cols_to_retain ]
cat_dict = cat_df.T.to_dict().values()
```

If you have a few categorical columns, you can list them as above. In the Analytics Edge competition, there are about 100 categorical columns, so in this case it's easier to drop columns which are not categorical:

```
cols_to_drop = [ 'UserID, 'YOB', 'votes', 'Happy']
```

```
cat_df = df.drop( cols_to_drop, axis = 1 ]
```

For those who like it dense - you could chain the manipulations in a one-liner:

```
cat_dict = df.drop( [ 'UserID', 'YOB', 'votes', 'Happy' ], axis = 1 ).T.to_dict().values()
```

After

Using the vectorizer

```
from sklearn.feature_extraction import DictVectorizer as DV

vectorizer = DV( sparse = False )
vec_x_cat_train = vectorizer.fit_transform( x_cat_train )
vec_x_cat_test = vectorizer.transform( x_cat_test )
```

If the data has missing values, they will become NaNs in the resulting Numpy arrays. Therefore it's advisable to fill them in with Pandas first:

```
cat_data = cat_data_with_missing_values.fillna( 'NA' )
```

This way, the vectorizer will create additional column `<feature>=NA` for each feature with NAs.

Handling binary features with missing values

If you have missing values in a binary feature, there's an alternative representation:

- *-1* for negatives
- *0* for missing values
- *1* for positives

It worked better in case of the Analytics Edge competition: an SVM trained on one-hot encoded data with *d* indicators scored 0.768 in terms of AUC, while the alternative representation yielded 0.778. That simple solution would give you 30th place out of 1686 contenders.

Code

We provide a sample script that loads data from CSV and vectorizes selected columns. Copy and paste the parts you find useful. If you'd like to run the script, you'll need:

1. data from the Analytics Edge competition.
2. to split the training set into two files for validation, for example with split.py

`split.py train.csv train_v.csv test_v.csv 0.8`

Make sure to have headers in both files. With `split.py`, headers from the source file will end up in one of the output files, probably in *train*. Just copy the header line to the other file using a text editor or `head -n 1` and `cat` on Unix.

Posted by Zygmunt Z. 2014-04-30 Kaggle, basics, code, software

Tweet G+1 9

« Predicting happiness from demographics and poll answers Impute missing values with Amelia »

Comments

16 Comments FastML - machine learning made easy Login

Recommend Share Sort by Best

Join the discussion…

brian · 2 years ago
Did you find any way to create one hot encoding with a reference level excluded - i.e. to use with linear models with an intercept?
2 · Reply · Share ›

ZygmuntZ Mod → brian · 2 years ago
I don't understand what you mean.
Reply · Share ›

Brian → ZygmuntZ · 2 years ago
I was experimenting with one-hot-encoding and if you have a variable, called Var1 which takes on values 'A', 'B' and 'C' after one hot you will end up with three indicator variables, one for each of the levels of Var1. You need to drop one if these level (making it the reference) if you fit a linear model (regression) with the data and include an intercept.
Reply · Share ›

ZygmuntZ Mod → Brian · 2 years ago
Ah, yes. You need d - 1 indicator variables to represent d levels. I think d should work just as well, I wouldn't bother with dropping one except the binary case when one column is the most natural.
Reply · Share ›

Brian → ZygmuntZ · 2 years ago
In R and other languages the model will not be fit - a linear dependence will be detected. Not sure if scikit learn will be effected. Worth an experiment.
Reply · Share ›

jmrosal → Brian · 2 years ago
Linear dependece will arise if you add a constant to your model and keep d indicator variables. If work with d-1, it should work just fine.
Reply · Share ›

Michael ➦ Brian • 2 years ago

Does anyone know how how this can be balanced with the need for feature selection? For example, I'm working with a dataset of 250 variables which, with one-hot-encoding becomes around 300 variables. Using recursive feature elimination I've learned an optimal model is fit keeping only 24 variables... what if one of the 24 vars is the "reference" variable that would be removed? On a variable with [A, B, C, D] options you need to keep all 3 A,B,C columns and see them at 0 to infer that D is the feature in use. It seems like this reference approach doesn't account for the case where D is the most predictive category and would be kept in an optimal model. Not debating, just trying to understand this better. Appreciate any advice.

• Reply • Share ›

ZygmuntZ Mod ➦ Michael • 2 years ago

Since there's no loss of information with d-1 columns, I think that a feature selection algorithm would find another, just as good, combination.

• Reply • Share ›

ZygmuntZ Mod ➦ Brian • 2 years ago

Well, I guess you could for each feature just drop one indicator variable to get d - 1 of them per feature. The vectorizer has a list of indicator column names.

• Reply • Share ›

disqus_0HlmhRwEjt • a year ago

What if my df has one column that is text which I intend to use bag of words (TfidfVectorizer) on... How do I still do the above for categorical data but then only apply TfidfVectorizer to one column.

• Reply • Share ›

Michael • 2 years ago

Thanks for the post. I found this extremely helpful and have referred to it 3+ times. One question: the approach seems to lose column names when converted to vector. Specifically at: vec_x_cat_train = vectorizer.fit_transform(x_cat_train)

This converts from a Pandas DataFrame with x_cat_train.columns to a numpy array with loss of identity for original variables. Looking around on StackOverflow one posts suggests numpy supports column names via vec_x_cat_train.dtype.names, but this returns None for me. Any tips on vectorizing without losing feature identity? I wrote a method that pulls it off using Pandas get_dummies(), but am keen to understand it via sklearn DictVectorizer if possible.

Re-applying original column names doesn't seem to fly since the shape expands to incorporate all the new column names. We might be able to extrapolate which columns are which, but it seems risky to assume numpy formatted the columns in a specific order without some sort of id.

^ | ⌄ • Reply • Share ›

ZygmuntZ Mod ➦ Michael • 2 years ago

Look at DictVectorizer object attributes in the scikit-learn documentation.

^ | ⌄ • Reply • Share ›

Michael ➦ ZygmuntZ • 2 years ago

Thank you!

^ | ⌄ • Reply • Share ›

Anuj • 2 years ago

Very helpful post. I am a beginner in ML using python(or using anything for that matter). I have one doubt - I have a large dataset and I am facing the same problem of converting string categories to int so that I can apply ML models on it. I have around 500k rows and 18 features with some features having as much as 5k unique values. In such as I get a 'memory error' when I use DV. I get that it is because it will require 5k bits for a single entry for that feature and hence I get the error. Is there any substitute for DV like OneHotEncoder but which also takes strings as categories and uses as many int categories as there are unique string categories. For eg if I have ['a','b', 'c','c'] as a column then I get [0,1,2,2] as the output.

Cheers

^ | ⌄ • Reply • Share ›

David Wihl • 2 years ago

for the same competition, I used scikit's LabelEncoding class to turn the column values into the equivalent of R factors. (Warning disqus flattens Python's indentation.)

```
for i in xrange(len(headers)):

factors.append(set([]))

self.labelEncoders.append(preprocessing.LabelEncoder())

#...and then

# Encode Labels

colnum = 0
```

for f in factors:

self.labelEncoders[colnum].fit(list(f))

colnum += 1

# and then

#demographics

for col in row[2:7]:

newRow.append(self.labelEncoders[colnum].transform([col])[0])

colnum += 1

• Reply • Share ›

xag • 2 years ago

Thank you Adblock to help me hide these cat pictures.

• Reply • Share ›

ALSO ON **FASTML - MACHINE LEARNING MADE EASY**

Deep nets generating stuff

5 comments • a year ago•

Avat ZygmuntZ — Dis is why! https://twitter.com/molleindus...

Coming out - FastML

2 comments • 4 months ago•

Avat zankbennett — April Fools! Nicely done :)

Real-time interactive movie recommendation

12 comments • a year ago•

Avat Zenon Eleates — Video with Cris, @27min http://www.youtube.com/watch?v...

What you wanted to know about AI, part II

5 comments • a year ago•

Avat Noone — First off, love your blog - thank you for years of interesting material! >> To sum up: creating real intelligence would be an enormous ego boost. That's ...

Subscribe Add Disqus to your site Add Disqus Add Privacy

Recent Posts

- Adversarial validation, part two
- ^one weird trick for training char-^r^n^ns
- Adversarial validation, part one
- Coming out
- Bayesian machine learning
- What next?
- What is better: gradient-boosted trees, or a random forest?

Twitter

Follow @fastml for notifications about new posts.

- Status updating...

Follow @fastml 36.7K followers

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

Cubert

Visualize your data in 3D, as described in Interactive in-browser 3D visualization of datasets.

http://cubert.fastml.com/

Jobs

Machine learning, data science and AI related jobs:

- PhD position (Gaussian processes) at Imperial College - London, England
- Post Doc Machine Learning at TELECOM ParisTech - Paris, France
- Research Fellow at AYLIEN (multiple openings) - Dublin, Ireland

More