autonomio Documentation

Release latest

Jun 09, 2017

Contents

This document covers in detail every function of Autonomio. If you're looking for a high level overview of the capabilities, you might find *[Autonomio_Overview]* more useful.

Autonomio is very easy to use and it's highly recommended to memorize the namespace which is less just 3 commands and less than 20 arguments combined. Yet you have an infinite number of network configurations available. To have 100% control over Autonomio's powerful features, you just have to know three commands.

To train (and save) model:

```
train()
```

To test (and load) model:

```
test()
```

To load a dataset:

```
data()
```

CHAPTER 1

DATA INPUTS

The expected input dataformat is Pandas dataframe. Deep learning is most useful in solving classification problems, and for that we are providing two modes 'binary' and 'categorical'.

BINARY (default)

- X can be text, integer
- Y can be an integer

The default settings are optimized for making a 1 or 0 prediction and for example in the case of predicting sentiment from tweets, Autonomio gives 85% accuracy out-of-the-box for classifying tweets that rank in the most negative 20% according to NLTK Vader sentiment analysis.

CATEGORICAL

- X can be text, integer
- Y can be an integer or text
- output layer neurons must match number of categories
- change activation_out

It's not a good idea to have too many categories, maybe 10 is pushing it in most cases.

CHAPTER 2

TRAIN

The absolute minimum use case using an Autonomio dataset is:

```
from autonomio.commands import *
%matplotlib inline
train('text','neg',data('random_tweets'))
```

Using this example and NLTK's sentiment analyzer as an input for the ground truth, Autonomio yields 85% prediction result out of the box with with nothing but:

```
train('text','neg',data('random_tweets'))
```

There are multiple ways you can input 'x' with single input:

```
train('text' ,'neg', data) # a single column where data is string

train(5, 'neg', data) # a single column by index

train(['quality_score'], 'neg', data) # a single column by label
```

And few more ways where you input a list for 'x':

```
train([1,5], 'neg', data) # a range of column index

train(['quality_score', 'reach_score'], 'neg', data) # set of column labels

train([1,2,4,6,18], 'neg', data) # a list of column index
```

A slightly more involving example may include changing the number of epochs:

```
train('text','neg',data('random_tweets'),epoch=20)
```

For flattening the options are 'mean', 'median', 'none' and IQR. IQR is invoked by inputting a float:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3)
```

Dropout is one of the most important aspects of neural network:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,dropout=.5)
```

You might want to change the number of layers in the network:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,dropouts=.5,layers=4)
```

Or change the loss of the model:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,dropouts=.5,layers=4,
↪loss='kullback_leibler_divergence')
```

For a complete list of supported losses see *[Keras_Losses]*

If you want to save the model, be mindful of using .json ending:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,save_model='model.json')
```

Control the neuron size by setting the number of neurons on the input layer:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,neuron_first=50)
```

Sometimes changing the batch size can improve the model significantly:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,batch_size=15)
```

By default verbosity from Keras is at mimimum, and you may want the live mode for training:

```
train('text','neg',data('random_tweets'),epoch=20,flatten=.3,verbose=1)
```

TRAIN ARGUMENTS

Even though it's possible to use Autonomio mostly with few arguments, there are a total 11 arguments that can be used to improving model accuracy:

```
def train(X,Y,data,
            dims=300,
            epoch=5,
            flatten='mean',
            dropout=.2,
            layers=3,
            model='train',
            loss='binary_crossentropy',
            save_model=False,
            neuron_first='auto',
            neuron_last=1,
            batch_size=10,
            verbose=0):
```

ARGUMENT	REQUIRED INPUT	DEFAULT
X	string, int, float	NA
Y	int,float,categorical	NA
data	data object	NA
epoch	int	5
flatten	string, float	'mean'
dropout	float	.2
layers	int (2 through 5	3
model	int	'train' (OBSOLETE)
loss	string *[Keras_Losses]*	'binary_crossentropy'
save_model	string,	False
neuron_first	int,float,categorical	300
neuron_last	data object	1
batch_size	int	10
verbose	0,1,2	0

Note that the network shape is roughly an upside-down pyramind. To change this you would want to change the code in train_new.py.

CHAPTER 3

TEST

Once you've trained a model with train(), you can use it easily on any dataset:

```
test('text',data,'handle','model.json')
```

Or if you want to see an interactive scatter plot visualization with new y variable:

```
test('text',data,'handle','model.json',y_scatter='influence_score')
```

Whatever y_scatter is set as, will be set as the y-axis for the scatter plot.

To yield the scatter plot, you have to call it specifically:

```
test_result = test('text',data,'handle','model.json',y_scatter='influence_score')
test_result[1]
```

TEST ARGUMENTS

The only difference between the two modes of test() is if a scatter plot is called:

```
def test(X,data,labels,saved_model,y_scatter=False)
```

ARGUMENT	REQUIRED INPUT	DEFAULT
X	variable/s in dataframe	NA
data	pandas dataframe	NA
labels	variable/s in dataframe	NA
saved_model	filename	5
y_scatter	variable in dataframe	'mean'

CHAPTER 4

DATA

Dataset consisting of 10 minute samples of 80 million tweets:

```
data('election_in_twitter')
```

4,000 ad funded websites with word vectors and 5 categories:

```
data('sites_category_and_vec')
```

Data from both buy and sell side and over 10 other sources:

```
data('programmatic_ad_fraud')
```

9 years of monthly poll and unemployment numbers:

```
data('parties_and_employment')
```

120,000 tweets with sentiment classification from NLTK:

```
data('tweet_sentiment')
```

20,000 random tweets:

```
data('random_tweets')
```

DATA ARGUMENTS

The data command is provided for both convinience, and to give the user access to unique deep learning datasets. In addition to allowing access to Autonomio datasets, the function also supports importing from csv, json, and excel. The data importing function is for most cases we face, but is not intended as a replacement to pandas read functions:

```
def data(name,mode='default')
```

ARGUMENT	REQUIRED INPUT	DEFAULT
name	dataset or filename	NA
mode	string ('file')	NA

CHAPTER 5

TROUBLESHOOTING

One of the most common errors you get working with Keras is related with your output layer:

```
ValueError: Error when checking model target: expected dense_22 to have shape (None,␣
↪2) but got array with shape (1000,
```

This means that your neuron_last does not match the number of categories in 'y'. Usually you would only see this with in cases where you have an output other than 1 or 0, or when you do have that but for some reason changed neuron_last to something else than 1 from train().

You could have a very similar error message from Keras if your dims is not same as the number of features:

```
ValueError: Error when checking model input: expected dense_1_input to have shape␣
↪(None, 300) but got array with shape (1000, 1)
```

NOTE: Your dims number must be exactly the same as the number of features in your mode ('x') except with series of text as an input where the default setting 300 is correct.

If your dims (input layer) is smaller than output layer (neuron_last):

```
ValueError: Input arrays should have the same number of samples as target arrays.␣
↪Found 100 input samples and 1 target samples.
```

CHAPTER 6

LINKS

Bibliography

[Keras_Losses] https://keras.io/losses/

[Autonomio_Overview] https://github.com/botlabio/autonomio/blob/master/README.md