

Fasttext & Doc2Vec for Text Classification



Hatice Şeyma Koç · Following

5 min read · Jul 7



11



Hello everyone,

In this article, I will show you how to build a multi-class classifier for any type of text with Fasttext and Doc2Vec. This document will be a guide for those who will work with the text classifier for the first time. Starting with data loading, I will follow the steps of data cleaning, training data preparation and model building. Finally, I will end my article by making predictions for the target. If you're ready, let's start with data loading!

Load Data

First of all, we need a classified data to do this. For this we will be using [this classified data set](#). The dataset contains 7613 Tweets that have been flagged as related to real disasters. There are 5 columns in this data. The important ones are “text” and “target” columns. The “text” column includes text of tweet, and the target column indicates whether the text is about a real

disaster. Using the data, we will create a multi-class classifier with Fasttext and Doc2vec.

Let's load in the data and look at it.

```
import pandas as pd

tweets = pd.read_csv('train.csv')
tweets.head()
```

```
>>> tweets.head()
   id keyword location          text  target
0    1     NaN     NaN  Our Deeds are the Reason of this #earthquake M...      1
1    4     NaN     NaN  Forest fire near La Ronge Sask. Canada          1
2    5     NaN     NaN  All residents asked to 'shelter in place' are ...      1
3    6     NaN     NaN  13,000 people receive #wildfires evacuation or...      1
4    7     NaN     NaN  Just got sent this photo from Ruby #Alaska as ...      1
```

Clean Text

As seen in the “text” column, there are some unwanted characters, therefore before starting the build process, “text” column should be cleaned from some characters. I will remove the following list of characters:

- Punctuations
- Numeric characters
- Stopwords like I, me, and etc.
- New line character: “\n”

Let's write clean code and see difference:

```
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
sw = stopwords.words("english")

# convert lower
tweets['cleaned_text'] = tweets['text'].str.lower()
# remove punctuations
tweets['cleaned_text'] = tweets['cleaned_text'].str.replace('[^\w\s]', '', regex=True)
# remove numeric characters
tweets['cleaned_text'] = tweets['cleaned_text'].str.replace('\d', '', regex=True)
# remove stopwords
tweets['cleaned_text'] = tweets['cleaned_text'].apply(lambda row: " ".join(x for x in row.split() if x not in sw))
# remove new line
tweets['cleaned_text'] = tweets['cleaned_text'].apply(lambda row: row.replace('\n', ' '))

tweets[['text', 'cleaned_text']].head()
```

	text	cleaned_text
0	Our Deeds are the Reason of this #earthquake M...	deeds reason earthquake may allah forgive us
1	Forest fire near La Ronge Sask. Canada	forest fire near la ronge sask canada
2	All residents asked to 'shelter in place' are ...	residents asked shelter place notified officer...
3	13,000 people receive #wildfires evacuation or...	people receive wildfires evacuation orders cal...
4	Just got sent this photo from Ruby #Alaska as ...	got sent photo ruby alaska smoke wildfires pou...

Split Data

Now since the main column is cleaned, the learning process can be started. As we know from all machine learning models, in order to calculate model success, all training data must be divided into 2 parts: train and test.

train_test_split function of *sklearn* library will be used for the split process. As a general usage, 20% of all data will be test data. The split code can be seen below:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(tweets, test_size=0.2)

print(f'Train data set has {train.shape[0]} data point.')
print(f'Test data set has {test.shape[0]} data point.')
```

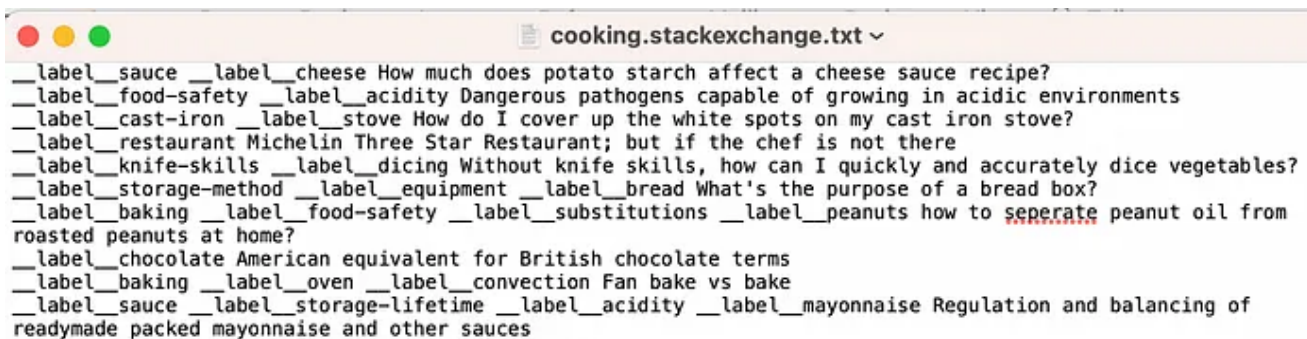
```
Train data set has 6090 data point.  
Test data set has 1523 data point.
```

Prepare Train Data

I will use the supervised model of fasttext library and doc2vec model of gensim library. To use the “cleaned_text” column in these models, we should prepare it in desired format. Let’s prepare both formats in turn.

- *Fasttext supervised model*

When you visit this website which is formal tutorial of fasttext supervised model, you can see that the following format is requested.



```
cooking.stackexchange.txt ~  
__label__sauce __label__cheese How much does potato starch affect a cheese sauce recipe?  
__label__food-safety __label__acidity Dangerous pathogens capable of growing in acidic environments  
__label__cast-iron __label__stove How do I cover up the white spots on my cast iron stove?  
__label__restaurant Michelin Three Star Restaurant; but if the chef is not there  
__label__knife-skills __label__dicing Without knife skills, how can I quickly and accurately dice vegetables?  
__label__storage-method __label__equipment __label__bread What's the purpose of a bread box?  
__label__baking __label__food-safety __label__substitutions __label__peanuts how to seperate peanut oil from  
roasted peanuts at home?  
__label__chocolate American equivalent for British chocolate terms  
__label__baking __label__oven __label__convection Fan bake vs bake  
__label__sauce __label__storage-lifetime __label__acidity __label__mayonnaise Regulation and balancing of  
readymade packed mayonnaise and other sauces
```

That is, each text should be labeled with its own tag, as seen in the example. In our data each text has only one label, so we will not see more than one label in a row. Also, you can see on the website, after the labeled texts prepared, you should save it as a train file. This file will be used later when creating the model. The following codes are doing these operations.

```
train['label'] = train['target'].apply(lambda row: '__label__' + str(row))

fasttext_train = train['label'] + ' ' + train['cleaned_text']
fasttext_train.to_csv('fasttext_train.train', index=False, header=False)
```

```
fast vs doc2.py x fasttext_train.train x
1 __label__0 cloydrivers plenty black people rioting tosu championship well
2 __label__0 wow crackdown uses multiple servers multiplayer u destroy whole buildings copped
3 __label__0 youre loving like water slipping fingers natural disaster love
4 __label__0 duchovbutt starbuck_scully madmakny davidduchovny yeah survived seasons movies lets hope good theres hope
5 __label__0 blew mentions
6 __label__1 cyhitheprynce bombed kanye elephantinthetroom
7 __label__1 stunckle gordon_r crazydoctorlady im expert raw uranium nuclear reactor fuel rods different creatures
8 __label__1 police kill hatchetwielding gunman opened fire inside nashville movie theater aâ€œmiddleaged manâ€œarmed wi httpcotydnflx
9 __label__0 imagine getting flattened kurt zouma
10 __label__0 hey girl must toe hobbit part two ghe desolation smaug im interested seeing sorry
```

- *Doc2Vec model*

Doc2Vec model, on the other hand, wants to create the tags of the data itself by using the *TaggedDocument* function in its own library. Let's take a look at the code for it:

```
from gensim.models.doc2vec import TaggedDocument
doc2vec_train = [TaggedDocument(doc, [i]) for i, doc in train[['target', 'cleaned_text']].itertuples(index=False)]
print(doc2vec_train[0:3])
|
```

```
[TaggedDocument(words='cloydrivers plenty black people rioting tosu championship well', tags=[0])]
[TaggedDocument(words='wow crackdown uses multiple servers multiplayer u destroy whole buildings copped', tags=[0])]
[TaggedDocument(words='youre loving like water slipping fingers natural disaster love', tags=[0])]
```

As you can see above results, this function adds the target as a tag to each text.

Build Model

Now we are ready to create the model. First let's import model's libraries, create models with default hyperparameters and take a look at it.

```
from fasttext import train_supervised
from gensim.models.doc2vec import Doc2Vec

fasttext_model = train_supervised(input='fasttext_train.train')

doc2vec_model = Doc2Vec()
doc2vec_model.build_vocab(doc2vec_train)
doc2vec_model.train(doc2vec_train, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.epochs)
```

Now we have 2 classifier models. They are trained using the same train data and will be tested with same test data. *But remember, no settings have been made yet.* These are just models that work and predict classes, but whose results are not trusted!

Predict Test Data

We are nearing the end. There are just a few steps. Since our models are ready, we can predict the class in test data. Let's get results from two models for a random row and see how it works, before estimating for all test data. Then we will also make predictions for the entire dataset.

```
test_index = test.sample().index[0]
sample_text = test.loc[test_index]['cleaned_text']
sample_target = test.loc[test_index]['target']

fasttext_prediction = fasttext_model.predict(sample_text)
doc2vec_prediction = doc2vec_model.dv.most_similar(doc2vec_model.infer_vector([sample_text]))[0]
```

```
The sample text is           : phone worstoverdose screams jaileens caked phone everyone looks
The sample target is         : 0
The prediction from fasttext is: (('__label__0',), array([0.97515267]))
The prediction from Doc2Vec is: (0, 0.0901530534029007)
```

The array in the fasttext result represents the probability of the predicted tag. The second value in the doc2vec result represents the distance of the vector of the predicted label to the vector of the text. In other words, we want the array value to be high in the fasttext output, while we want it to be low in the doc2vec output.

Now let's make our predictions for all test data and compare the success values.

```
test['pred_ft'] = test['cleaned_text'].apply(lambda row: int(fasttext_model.predict(row)[0][0][9:]))
test['pred_dv'] = test['cleaned_text'].apply(lambda row:
                                             doc2vec_model.dv.most_similar(doc2vec_model.infer_vector([row]))[0][0])

ft_success = test[test['target'] == test['pred_ft']].shape[0] / test.shape[0]
dv_success = test[test['target'] == test['pred_dv']].shape[0] / test.shape[0]
```

```
The success rate of fasttext model is: 0.8076165462902167
The success rate of Doc2Vec model is: 0.5147734734077478
```

As I said above, the results should be improved by tuning the hyperparameters of these models. In this article, I explained how to create a classifier for texts only. Improving the results by adjusting the parameters will be the subject of another article.

NLP

Machine Learning

Fasttext

Doc2vec

Text Classification

More from the list: "NLP"

Curated by Himanshu Birla



Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

★ · 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

★ · 6 min read · Sep 3, 2021



Jon Gi... in

The Word2vec

★ · 15 min read



[View list](#)



Written by Hatice Şeyma Koç

10 Followers

Following



More from Hatice Şeyma Koç



Hatice Şeyma Koç

How to Deploy Projects?



Hatice Şeyma Koç

Data Science in Global Trade

I am back with a new article after a long break.
As you know, a project consists of many...

5 min read · Sep 7



1



Hello everyone,

2 min read · Jun 12



31



Hatice Şeyma Koç

How Does Random State Work?

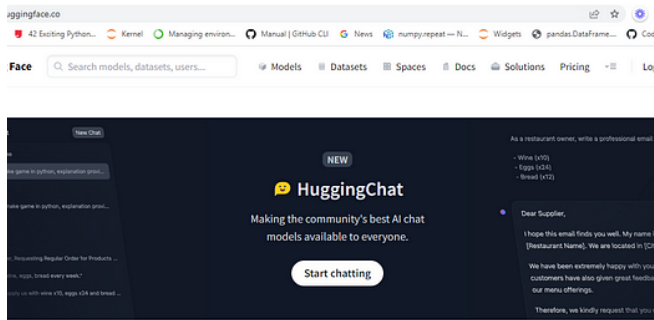
Hello everyone,

4 min read · Jul 13



See all from Hatice Şeyma Koç

Recommended from Medium



 Bright Eshun

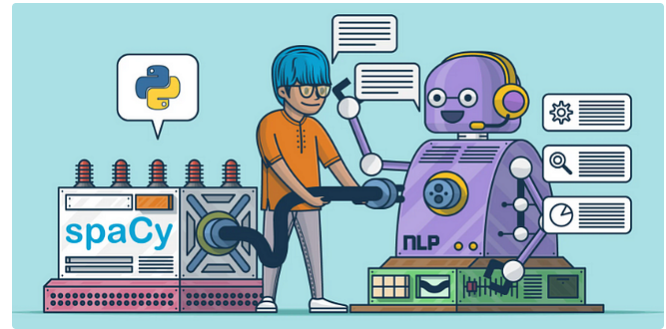
Sentiment Analysis (Part 1): Finetuning DistilBert for Text...


I. Introduction

9 min read · May 8



2



 HasancanÇakıcıoğlu

Comprehensive Text Preprocessing NLP (Natural Language...

Text preprocessing plays a crucial role in Natural Language Processing (NLP) by...

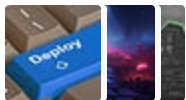
12 min read · Jul 9



12



Lists



Predictive Modeling w/ Python

20 stories · 452 saves



Natural Language Processing

669 stories · 283 saves



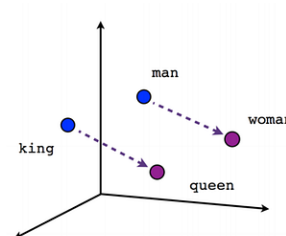
Practical Guides to Machine Learning

10 stories · 519 saves

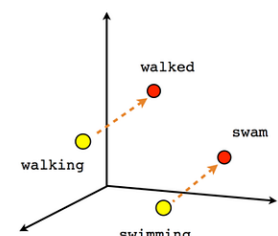


The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves



Male-Female



Verb tense



Ahmet Taşdemir

Fine-Tuning DistilBERT for Emotion Classification

In this post, we will walk through the process of fine-tuning the DistilBERT model for...

8 min read · Jun 14



Maninder Singh

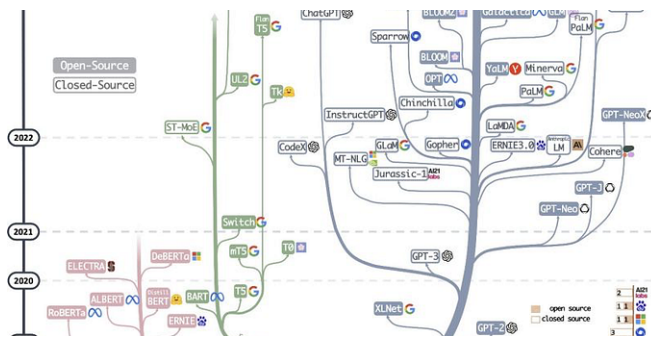
Accelerate Your Text Data Analysis with Custom BERT Word...

One thing is for sure the way humans interact with each other naturally is one of the most...

4 min read · Apr 24



156



Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14



372



Nimrita Koul

Natural Language Processing with Python Part 5: Text Classification

This article is the fifth in the series of my articles covering the sessions I delivered for...

8 min read · May 4



1



See more recommendations