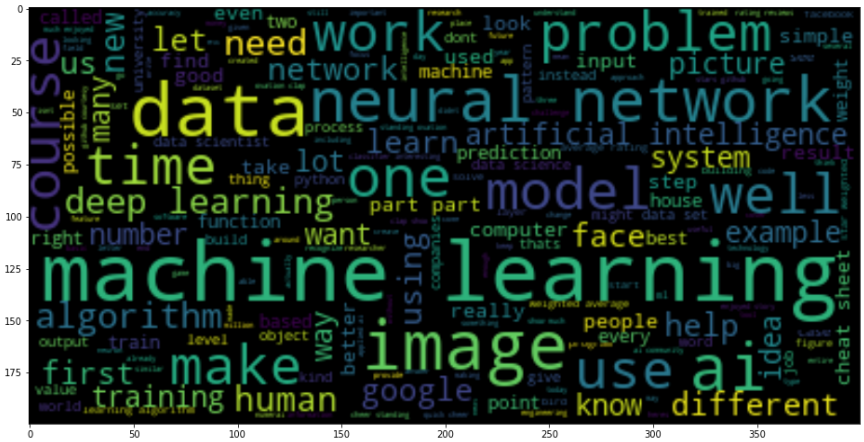
Multi-Label Classification(Blog Tags Prediction)using NLP

Indresh Bhattacharyya Follow
Jul 10, 2018 · 8 min read



The difference between a multi-class classification and a multi-label classification?

A multi class classification is where there are multiple categories associated in the Y axis or the target variable but each row of data falls under single category.

Where as in multi-label classification multiple categories are associated with the same data. Simply each row may have multiple categorical values.

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

This is a binary classification problem

See the above data set our categorical values is basically

'LEGENDARY' → it has a binary value(True or False)

Now let us have a look at the data we are going to work on:

	Body	Title	0	1	2	3	4
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	0	0	0	0
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	Artificial Intelligence	Tech	Programming	Technology
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	Algorithms	Data Science	Towards Data Science	0
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	Deep Learning	Artificial Intelligence	Neural Networks	Big Data
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	Artificial Intelligence	Deep Learning	0	0
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Machine Learning	Artificial Intelligence	Deep Learning	0	0
0	Update: This article is part of a series. Che...	Machine Learning is Fun!	Artificial Intelligence	Machine Learning	Startups	Tech	Technology

This is a multi-label dataset

See this data set it has multiple values associated with each row.

The Work Flow goes like this.

Scrape data from web

Clean and Preprocess

Visualize

Classify

Objective:

In this project we are going to scrape data from medium and identify the tags given make a Data Frame out of it and in OneHotEncoding format and then.Classify which Blog post Fall under which tags.

So Let's Start:

Ok, I am scraping the data from medium itself:

```
import pandas as pd
from bs4 import BeautifulSoup
```

```
import urllib3
```

Import the needed libraries ***pandas*** for Data Frame and ***urllib3*** for connecting to the web and fetching the data. ***Beautiful Soup*** is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

```
http=urllib3.PoolManager()
from pandas import DataFrame
column=['Title','Body']
dfBA=DataFrame(columns=column)
dfT=DataFrame(columns=[0,1,2,3,4])
```

Creating empty dataFrame for title and body **column= ['Title','Body'] dfBA=DataFrame(columns=column)**

Also one more for the tags **dfT=DataFrame(columns=[0,1,2,3,4])**

```
def spider(link):
print(link)
blogData=http.request('GET',link)
soup=BeautifulSoup(blogData.data,'html.parser')
for links in soup.find_all('div',{'class':'postArticle-readMore'}):
link=links.find('a').get('href')
CrawlAndFrame(link)
```

The spider function will go into the web pages and get the links for all the posts in the web page.

Note: As urllib3 is not good at dynamic scraping it will only fetch 7 posts per page. Use selenium for dynamic scraping

```
def CrawlAndFrame(link):
try:
print(link)
blogData=http.request('GET',link)
soup=BeautifulSoup(blogData.data,'html.parser')
article=''
tags=[]
heading=soup.find('h1').text
for para in soup.find_all('p'):
```

```
p=para.text
p=p.strip('\u')
article=article+' '+p
for mtags in soup.find_all('a',{'class':'link u-baseColor—link'}):
tags.append(mtags.text)
#CreateDataFrame(list())
someList=[heading,article,tuple(tags)]
#print(someList[0])
CreateDataFrame(someList)
except:
pass
```

CrawlAndFrame() function in going into every link collected by the spider and from there it collects all the blog ***article*** and ***title*** of the blog with the ***tags*** involved in the blog and passes them ***CreateDataFrame(someList)*** which creates a dataframe for the head and article(**dfBA**) and another for the tags(dfT).

```
def CreateDataFrame(someList):
t={}
d={'Title':[someList[0]],'Body':[someList[1]]}
for n in range(5):
if len(someList[2])>n:
t[n]=[someList[2][n]]
else:
t[n]=['0']
toDf=DataFrame(data=d)
global dfBA,dfT
#print(dfBA)
dfBA=dfBA.append(toDf)
#print(dfBA)
dfT=dfT.append(DataFrame(data=t))
```

CreateDataFrame() creates the dataframe for **dfDA** and **dfT** respectively for (Title and Body) and (Tags)

Out[7]:

	Body	Title
0	Update: This article is part of a series. Che...	Machine Learning is Fun!
0	A year and a half ago, I dropped out of one o...	Every single Machine Learning course on the in...
0	In machine learning, there's something called...	A Tour of The Top 10 Algorithms for Machine Le...
0	Over the past few months, I have been collect...	Cheat Sheets for AI, Neural Networks, Machine ...
0	Update: This article is part of a series. Che...	Machine Learning is Fun! Part 4: Modern Face R...
0	Update: This article is part of a series. Che...	Machine Learning is Fun! Part 3: Deep Learning...
0	I have a challenge for you. In a few seconds,...	The Non-Technical Guide to Machine Learning & ...
0	Learning machine learning and deep learning i...	Essential Cheat Sheets for Machine Learning an...
0	Update: This article is part of a series. Che...	Machine Learning is Fun! Part 2
0	For the past year, we've compared nearly 8,80...	30 Amazing Machine Learning Projects for the P...
0	Artificial Intelligence (AI) and Machine Lear...	Want to know how Deep Learning works? Here's a...
0	Over the past few months, I have been collect...	Cheat Sheets for AI, Neural Networks, Machine ...
0	(An alternate version of this article was ori...	Deep Learning Is Going to Teach Us All the Les...
0	Update: This article is part of a series. Che...	Machine Learning is Fun! Part 4: Modern Face R...
0	Update: This article is part of a series. Che...	Machine Learning is Fun! Part 3: Deep Learning...
0	Learning machine learning and deep learning i...	Essential Cheat Sheets for Machine Learning an...
0	[Update—Feb 2nd 2018: When this blog post w...	Thoughts after taking the Deeplearning.ai courses

dfBA

	0	1	2	3	4
0	Machine Learning	0	0	0	0
0	Machine Learning	Artificial Intelligence	Tech	Programming	Technology
0	Machine Learning	Algorithms	Data Science	Towards Data Science	0
0	Machine Learning	Deep Learning	Artificial Intelligence	Neural Networks	Big Data
0	Machine Learning	Artificial Intelligence	Deep Learning	0	0
0	Machine Learning	Artificial Intelligence	Deep Learning	0	0
0	Artificial Intelligence	Machine Learning	Startups	Tech	Technology
0	Machine Learning	Artificial Intelligence	Deep Learning	Technology	Computer Science
0	Artificial Intelligence	Machine Learning	Nintendo	0	0
0	Machine Learning	Data Science	Artificial Intelligence	Programming	Tech
0	Jonathan Sugumar	Technology	Startup	Business	Artificial Intelligence
0	Machine Learning	Deep Learning	Artificial Intelligence	Neural Networks	Big Data

dfT

Looking at **dfT** You can probably tell that this is not a optimal way to the tag data. So we need to change it into OneHotEncoding format(***which is basically creating a sparse matrix of 0 and 1 where 1 represents that the index tag is present 0 represents that it doesnt***)

Lets do it:

```
okList=[]
for cl in dfT.columns:
 for n in dfT[cl]:
 okList.append(n)
```

```
okList=list(set(okList))
del(okList[okList.index('0')])
newDF=DataFrame(columns=okList)
for x in range(dfT.count()[0]):
 someDict={}
 for d in okList:
 rowdata=list(dfT.iloc[x])
 if d in rowdata:
 someDict[d]=1
 else:
 someDict[d]=0
 newDF=newDF.append(someDict,ignore_index=True)
newDF
```

In a nutshell what I did here was just taking all the unique tags in the List and made them the columns of my dataFrame and put 1 if that tag is present in the row and 0 if not

	Bitcoin	Venture Capital	GoogleIO	Resources	Recruitment	Big Data	Blockchain	Towards Data Science	Insights	Nintendo	...	Neural Networks	Technology	Jonathan Sugumar	Startups	I
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	1	0	0	
2	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	
3	0	0	0	0	0	1	0	0	0	0	...	1	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	1	0	1	
7	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	
8	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	...	0	1	1	0	
11	0	0	0	0	0	1	0	0	0	0	...	1	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	
16	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	

We thus got a OneHotEncoded Data frame from the tags

NOW THAT OUR DATA IS READY LETS START PREPROCESSING IT:

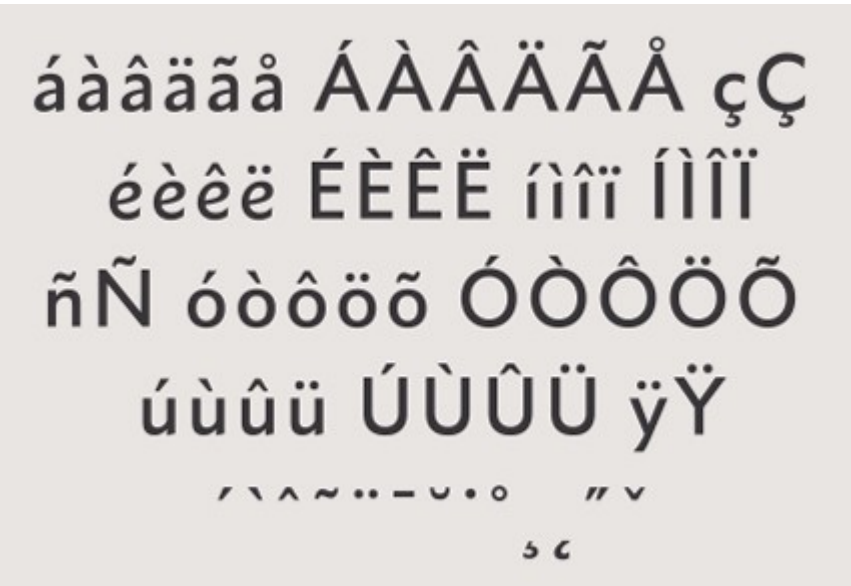
```
from nltk.corpus import stopwords
stopWordList=stopwords.words('english')
```

```
stopWordList.remove('no')
stopWordList.remove('not')
def removeTags(data):
 soup=BeautifulSoup(data,'html.parser')
 text=soup.get_text()
 return text
import unicodedata
```

This removes every html tags. If there are any. Sometimes even after scraping some tags remain. We are removing that.

```
def removeAscendingChar(data):
 data=unicodedata.normalize('NFKD', data).encode('ascii',
'ignore').decode('utf-8', 'ignore')
 return data
```

This function transforms all the accented characters into normal English .



Accented characters

```
def removeCharDigit(text):
 str='`1234567890-=~@#$%^&*()_+[!{;":\'><.,/?"}]'
 for w in text:
 if w in str:
 text=text.replace(w,'')
 return text
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import ToktokTokenizer
lemma=WordNetLemmatizer()
token=ToktokTokenizer()
```

Removes all special characters and digits

```
def lemitizeWords(text):
words=token.tokenize(text)
 listLemma=[]
 for w in words:
 x=lemma.lemmatize(w,'v')
 listLemma.append(x)
 return text
def stopWordsRemove(text):
 wordList=[x.lower().strip() for x in token.tokenize(text)]
 removedList=[x for x in wordList if not x in stopWordList]
 text=' '.join(removedList)
 return text
```

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is →be
car, cars, car's, cars →car

Stemming and lemmatization

Stemming and
lemmatizationnlp.stanford.edu

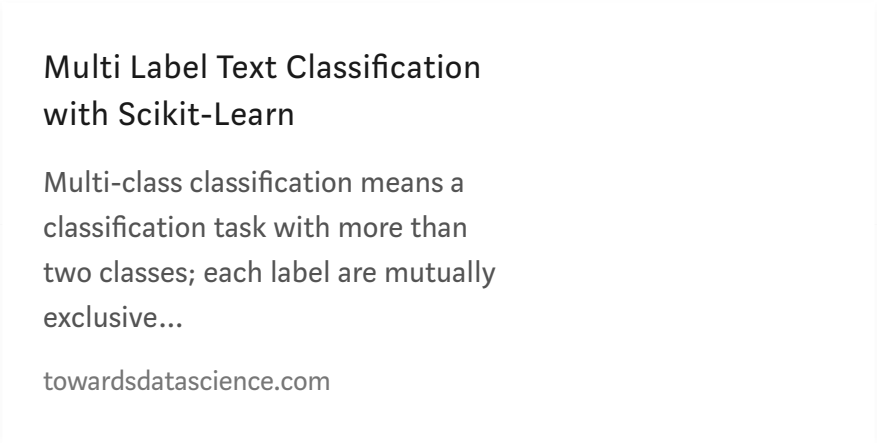
```
def PreProcessing(text):
text=removeTags(text)
text=removeCharDigit(text)
text=removeAscendingChar(text)
text=lemitizeWords(text)
text=stopWordsRemove(text)
return(text)
```

```
import re
def clean_text(text):
text = text.lower()
text = re.sub(r"what's", "what is ", text)
text = re.sub(r"\'s", " ", text)
text = re.sub(r"\'ve", " have ", text)
text = re.sub(r"can't", "can not ", text)
text = re.sub(r"n't", " not ", text)
text = re.sub(r"i'm", "i am ", text)
text = re.sub(r"\'re", " are ", text)
text = re.sub(r"\'d", " would ", text)
text = re.sub(r"\'ll", " will ", text)
text = re.sub(r"\'scuse", " excuse ", text)
```

```
text = re.sub('\W', ' ', text)
text = re.sub('\s+', ' ', text)
text = text.strip(' ')
return text
```

> *Now lets change the words like i'm to i am or what's to what is and thats exactly what clean_text is doing.*

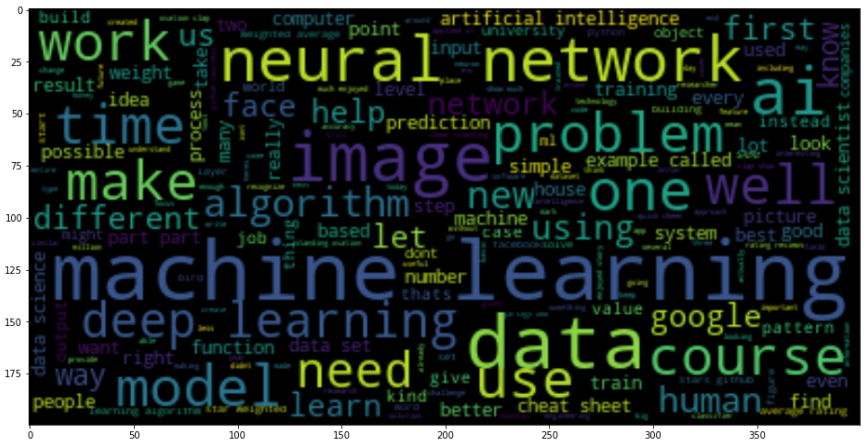
For more check:




```
df['Body'] = df['Body'].map(lambda com : clean_text(com))
df['Body'] = df['Body'].map(lambda com : PreProcessing(com))
```

Lets visualize:

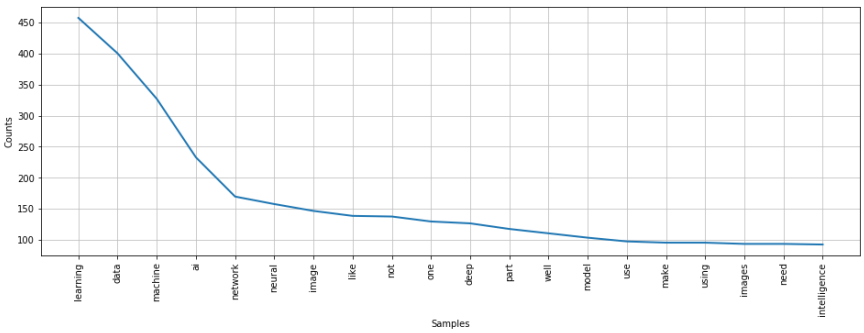
```
totalText=''
for x in df['Body']:
 ps=PreProcessing(x)
 totalText=totalText+' '+ps
from wordcloud import WordCloud
wc=WordCloud(background_color='black',max_font_size=50).genera
te(totalText)
plt.figure(figsize=(16,12))
plt.imshow(wc, interpolation='bilinear')
```



World cloud Representaion

What about frequency of the words?

```
import nltk
from nltk.tokenize import ToktokTokenizer
x=nltk.FreqDist(ToktokTokenizer().tokenize(totalText))
plt.figure(figsize=(16,5))
x.plot(20)
```



Word frequency Plot

As you can see the most frequent words in the article is Learning,Data,Machine,ai etc

Classification(Training and Testing the model):

```
# using binary relevance
import pandas as DataFrame
from skmultilearn.problem_transform import BinaryRelevance
```

```
from sklearn.naive_bayes import GaussianNB
```

```
# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
classifier = BinaryRelevance(GaussianNB())
```

```
# train
classifier.fit(x, y)
```

```
# predict
predictions = classifier.predict(x)
print(predictions.toarray())
print(accuracy_score(y, predictions))
```

What is binary relevance??

This is the simplest technique, which basically treats each label as a separate single class classification problem.

Say we have **x** as a independent variable and y1,y2,y3 as the labels of dependent variable . So what ***binary relevance*** does is that is it treats each independent variable as a separate class in consideration to the independent variable.

so it maps

$$x \rightarrow y1 \text{ and } x \rightarrow y2 \text{ and } x \rightarrow y3$$

```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

```
# initialize classifier chains multi-label classifier
# with a gaussian naive bayes base classifier use any other classifier if u
```

```
wish
#classifier = ClassifierChain(GaussianNB())
classifier = ClassifierChain(DecisionTreeClassifier())
# train
classifier.fit(x, y)
```

```
# predict
predictions = classifier.predict(x)
```

```
accuracy_score(y,predictions)
```

What is classifier chains??

In this, the first classifier is trained just on the input data and then each next classifier is trained on the input space and all the previous classifiers in the chain.

Say we have **x** as a independent variable and y1,y2,y3 as the labels of dependent variable . So we basically have 3 subsets of classification

$$x \rightarrow y1 \text{ and } x \rightarrow y1,y2 \text{ and } x \rightarrow y1,y2,y3$$

```
# using Label Powerset
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
```

```
# initialize Label Powerset multi-label classifier
# with a gaussian naive bayes base classifier
classifier = LabelPowerset(GaussianNB())
#OR
#classifier = ClassifierChain(DecisionTreeClassifier())
# train
classifier.fit(x, y)
```

```
# predict
```

```
predictions = classifier.predict(x)
```

```
accuracy_score(y,predictions)
```

What is Label Powerset??

In this, we transform the problem into a multi-class problem with one multi-class classifier is trained on all unique label combinations found in the training data.

```
from skmultilearn.adapt import MLkNN
```

```
classifier = MLkNN(k=20)
```

```
classifier.fit(x, y)
```

```
# predict
predictions = classifier.predict(x)
print(predictions.toarray())
print(y)
accuracy_score(y,predictions)
from sklearn.metrics import f1_score
print(f1_score(y,predictions,average='micro'))
```

MLkNN:

Adapted algorithm, as the name suggests, adapting the algorithm to directly perform multi-label classification, rather than transforming the problem into different subsets of problems.

For example, multi-label version of kNN is represented by MLkNN.

Last but not the least we have oneVSRest Classifier

Also known as one-vs-all, this strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency

```
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score

clf=OneVsRestClassifier(MultinomialNB())

clf.fit(x,y)

pred=clf.predict(x)

accuracy_score(y,pred)
```

DOWNLOAD THE WHOLE CODE HERE

neelindresh/NeelBlog

Contains the code and csv from my blog. Contribute to neelindresh/NeelBlog development by creating an account on...

github.com

Solving Multi-Label Classification problems (Case studies included)

Introduction For some reason, Regression and Classification problems end up taking most of the attention in machine...

sklearn.multiclass.OneVsRestClassifier - scikit-learn 0.19.1 documentation

Also known as one-vs-all, this strategy consists in fitting one classifier per class. For each classifier, the class is...

scikit-learn.org

Selecting a multi-label classifier - scikit-multilearn 0.0.5 documentation

The algorithm adaptation approach is based on a single-label classification method adapted for multi-label...

scikit.ml

. . .

For More follow my blog:

Data Science for Everyone

This post is about SUPPORT VECTOR REGRESSION. Those who are in Machine Learning or Data Science are quite familiar with...

dataneel.wordpress.com