Photo credit: Pixabay

Multi-Class Text Classification Model Comparison and Selection

Natural Language Processing, word2vec, Support Vector Machine, bag-of-words, deep learning

Susan Li Follow
Sep 25, 2018 · 7 min read

When working on a supervised machine learning problem with a given data set, we try different algorithms and techniques to search for models to produce general hypotheses, which then make the most accurate predictions possible about future instances. The same principles apply to text (or document) classification where there are many models can be used to train a text classifier. The answer to the question "What machine learning model should I use?" is always "It depends." Even the most experienced data scientists can't tell which algorithm will perform best before experimenting them.

This is what we are going to do today: use everything that we have presented about text classification in the previous articles (and more) and comparing between the text classification models we trained in order to choose the most accurate one for our problem.

The Data

We are using a relatively large data set of Stack Overflow questions and tags. The data is available in Google BigQuery, it is also publicly

available at this Cloud Storage URL:
https://storage.googleapis.com/tensorflow-workshop-examples/stack-overflow-data.csv.

Exploring the Data

```
import logging
import pandas as pd
import numpy as np
from numpy import random
import gensim
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVecto
from sklearn.metrics import accuracy_score, confusion_
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
from bs4 import BeautifulSoup
```

explore

	post	tags
0	what is causing this behavior in our c# datet...	c#
1	have dynamic html load as if it was in an ifra...	asp.net
2	how to convert a float value in to min:sec i ...	objective-c
3	.net framework 4 redistributable just wonderi...	.net
4	trying to calculate and print the mean and its...	python
5	how to give alias name for my website i have ...	asp.net
6	window.open() returns null in angularjs it wo...	angularjs
7	identifying server timeout quickly in iphone ...	iphone
8	unknown method key error in rails 2.3.8 unit ...	ruby-on-rails
9	from the include how to show and hide the con...	angularjs

Figure 1

10276752

We have over 10 million words in the data.

```
my_tags = ['java','html','asp.net','c#','ruby-on-
rails','jquery','mysql','php','ios','javascript','python','c
','css','android','iphone','sql','objective-
```

```
c','c++','angularjs','.net']
plt.figure(figsize=(10,4))
df.tags.value_counts().plot(kind='bar');
```

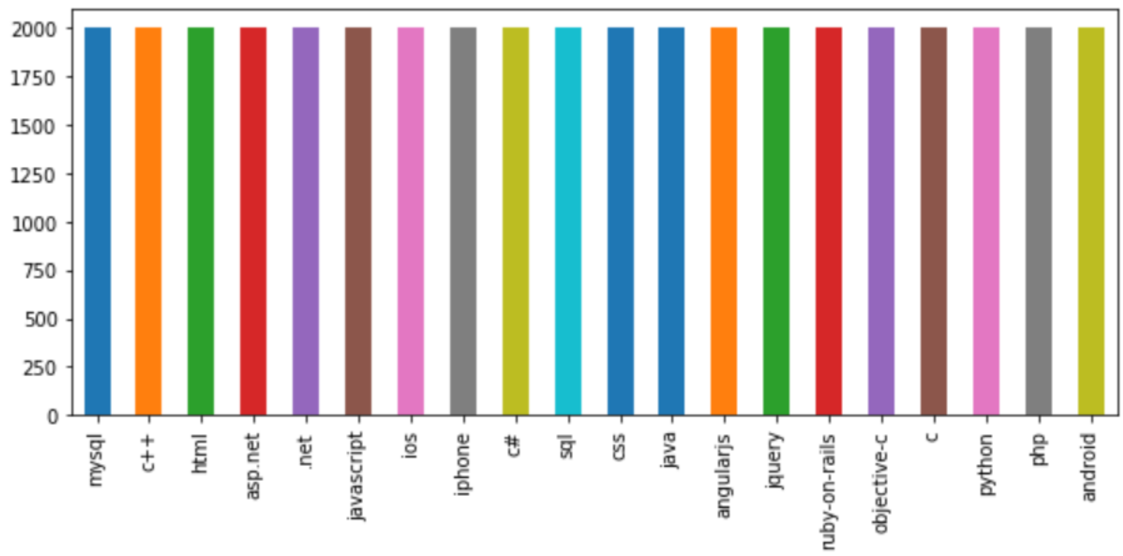

Figure 2

The classes are very well balanced.

We want to have a look a few post and tag pairs.

```
def print_plot(index):
    example = df[df.index == index][['post',
'tags']].values[0]
    if len(example) > 0:
        print(example[0])
        print('Tag:', example[1])
```

```
print_plot(10)
```

```
when we need interface c# <blockquote>    <strong>possible duplicate:</strong><br>    <a href= https://stackoverflow.com/questio
ns/240152/why-would-i-want-to-use-interfaces >why would i want to use interfaces </a>    <a href= https://stackoverflow.com/ques
tions/9451868/why-i-need-interface >why i need interface </a>     </blockquote>     i want to know where and when to use it
for example    <pre><code>interface idemo {  // function prototype  public void show(); }  // first class using the interface c
lass myclass1 : idemo {  public void show()  {   // function body comes here   response.write( i m in myclass );  }  }  // seco
nd class using the interface class myclass2 : idemo {  public void show()   {   // function body comes here   response.write( i
m in myclass2 );   response.write( so  what  );  } </code></pre>   these two classes has the same function name with different
body. this can be even achieved without interface. then why we need an interface where and when to use it
Tag: c#
```

Figure 3

```
print_plot(30)
```

```
how to chain expressions inside ngclass when using the {...}[] form  how can i add another expression to an <code>ng-class</cod
e> directive that uses this form:   <pre><code>ng-class= {true: loading   false: loading-done }[data.loader===null]  </code></p
re>   i d like to add something like this to the list:   <pre><code>{highlight:isspecial} </code></pre>   is it possible withou
t expanding the first expression     thanks.
Tag: angularjs
```

Figure 4

As you can see, the texts need to be cleaned up.

Text Pre-processing

The text cleaning techniques we have seen so far work very well in practice. Depending on the kind of texts you may encounter, it may be relevant to include more complex text cleaning steps. But keep in mind that the more steps we add, the longer the text cleaning will take.

For this particular data set, our text cleaning step includes HTML decoding, remove stop words, change text to lower case, remove punctuation, remove bad characters, and so on.

```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """
        text: a string

        return: modified initial string
    """
    text = BeautifulSoup(text, "lxml").text # HTML dec
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text) # replac
```

clean_text

Now we can have a look a cleaned post:

```
need interface c# possible duplicate would want use interfaces need interface want know use example interface idemo function pr
ototype public void show first class using interface class myclass1 idemo public void show function body comes responsewrite my
class second class using interface class myclass2 idemo public void show function body comes responsewrite myclass2 responsewri
te two classes function name different body even achieved without interface need interface use
Tag: c#
```

Figure 5

Way better!

```
df['post'].apply(lambda x: len(x.split(' '))).sum()
```

3421180

After text cleaning and removing stop words, we have only over 3 million words to work with!

After splitting the data set, the next steps includes feature engineering. We will convert our text documents to a matrix of token counts (CountVectorizer), then transform a count matrix to a normalized tf-idf representation (tf-idf transformer). After that, we train several classifiers from Scikit-Learn library.

```
X = df.post
y = df.tags
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state = 42)
```

Naive Bayes Classifier for Multinomial Models

After we have our features, we can train a classifier to try to predict the tag of a post. We will start with a Naive Bayes classifier, which provides a nice baseline for this task. `scikit-learn` includes several variants of this classifier; the one most suitable for text is the multinomial variant.

To make the vectorizer => transformer => classifier easier to work with, we will use `Pipeline` class in Scilkit-Learn that behaves like a compound classifier.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTrans

nb = Pipeline([('vect', CountVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
              ])
nb.fit(X_train, y_train)

%%time
```

nb

```
accuracy 0.7395
               precision    recall  f1-score   support

         java       0.63      0.65      0.64       613
         html       0.94      0.86      0.90       620
      asp.net       0.87      0.92      0.90       587
           c#       0.70      0.77      0.73       586
ruby-on-rails       0.73      0.87      0.79       599
       jquery       0.72      0.51      0.60       589
        mysql       0.77      0.74      0.75       594
          php       0.69      0.89      0.78       610
          ios       0.63      0.59      0.61       617
   javascript       0.57      0.65      0.61       587
       python       0.70      0.50      0.59       611
            c       0.79      0.79      0.79       594
          css       0.84      0.59      0.69       619
      android       0.66      0.84      0.74       574
       iphone       0.64      0.83      0.72       584
          sql       0.66      0.64      0.65       578
  objective-c       0.79      0.77      0.78       591
          c++       0.89      0.83      0.86       608
    angularjs       0.94      0.89      0.91       638
         .net       0.74      0.66      0.70       601

  avg / total       0.75      0.74      0.74     12000

Wall time: 955 ms
```

Figure 6

We achieved 74% accuracy.

Linear Support Vector Machine

Linear Support Vector Machine is widely regarded as one of the best text classification algorithms.

```
from sklearn.linear_model import SGDClassifier

sgd = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', SGDClassifier(loss='hinge', pe
               ])
sgd.fit(X_train, y_train)

%%time

```

svm

```
accuracy 0.7891666666666667
               precision    recall  f1-score   support

         java       0.74      0.68      0.71       613
         html       0.85      0.93      0.89       620
      asp.net       0.87      0.95      0.91       587
           c#       0.81      0.80      0.80       586
ruby-on-rails       0.74      0.88      0.80       599
       jquery       0.77      0.41      0.53       589
        mysql       0.82      0.68      0.74       594
          php       0.70      0.95      0.81       610
          ios       0.82      0.56      0.66       617
   javascript       0.72      0.59      0.65       587
       python       0.71      0.65      0.68       611
            c       0.81      0.87      0.84       594
          css       0.77      0.79      0.78       619
      android       0.83      0.86      0.85       574
       iphone       0.81      0.80      0.81       584
          sql       0.71      0.68      0.69       578
  objective-c       0.81      0.90      0.85       591
          c++       0.84      0.96      0.89       608
    angularjs       0.87      0.95      0.91       638
         .net       0.77      0.89      0.83       601

  avg / total       0.79      0.79      0.78     12000

Wall time: 1.26 s
```

Figure 7

We achieve a higher accuracy score of 79% which is 5% improvement over Naive Bayes.

Logistic Regression

Logistic regression is a simple and easy to understand classification algorithm, and Logistic regression can be easily generalized to multiple classes.

```
from sklearn.linear_model import LogisticRegression

logreg = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', LogisticRegression(n_jobs=1, C
               ])
logreg.fit(X_train, y_train)

%%time

```

logreg

```
accuracy 0.783
               precision    recall  f1-score   support

         java       0.70      0.62      0.66       613
         html       0.91      0.91      0.91       620
      asp.net       0.97      0.94      0.95       587
           c#       0.78      0.77      0.78       586
ruby-on-rails       0.77      0.81      0.79       599
       jquery       0.59      0.58      0.58       589
        mysql       0.77      0.76      0.76       594
          php       0.82      0.86      0.84       610
          ios       0.70      0.72      0.71       617
   javascript       0.61      0.59      0.60       587
       python       0.64      0.63      0.64       611
            c       0.83      0.83      0.83       594
          css       0.78      0.78      0.78       619
      android       0.85      0.85      0.85       574
       iphone       0.80      0.83      0.81       584
          sql       0.65      0.65      0.65       578
  objective-c       0.82      0.84      0.83       591
          c++       0.91      0.91      0.91       608
    angularjs       0.96      0.94      0.95       638
         .net       0.78      0.83      0.80       601

  avg / total       0.78      0.78      0.78     12000

Wall time: 981 ms
```

Figure 8

We achieve an accuracy score of 78% which is 4% higher than Naive Bayes and 1% lower than SVM.

As you can see, following some very basic steps and using a simple linear model, we were able to reach as high as an 79% accuracy on this multi-class text classification data set.

Using the same data set, we are going to try some advanced techniques such as word embedding and neural networks.

Now, let’s try some complex features than just simply counting words.

Word2vec and Logistic Regression

Word2vec, like doc2vec, belongs to the text preprocessing phase. Specifically, to the part that transforms a text into a row of numbers.

Word2vec is a type of mapping that allows words with similar meaning to have similar vector representation.

The idea behind Word2vec is rather simple: we want to use the surrounding words to represent the target words with a Neural Network whose hidden layer encodes the word representation.

First we load a word2vec model. It has been pre-trained by Google on a 100 billion word Google News corpus.

```
from gensim.models import Word2Vec
```

```
wv = 
gensim.models.KeyedVectors.load_word2vec_format("GoogleNews-
vectors-negative300.bin.gz", binary=True)
wv.init_sims(replace=True)
```

We may want to explore some vocabularies.

```
from itertools import islice
list(islice(wv.vocab, 13030, 13050))
```

```
['Memorial_Hospital',
 'Seniors',
 'memorandum',
 'elephant',
 'Trump',
 'Census',
 'pilgrims',
 'De',
 'Dogs',
 '###-####_ext',
 'chaotic',
 'forgive',
 'scholar',
 'Lottery',
 'decreasing',
 'Supervisor',
 'fundamentally',
 'Fitness',
 'abundance',
 'Hold']
```

Figure 9

BOW based approaches that includes averaging, summation, weighted addition. The common way is to average the two word vectors. Therefore, we will follow the most common way.

```
def word_averaging(wv, words):
    all_words, mean = set(), []

    for word in words:
        if isinstance(word, np.ndarray):
            mean.append(word)
        elif word in wv.vocab:
            mean.append(wv.syn0norm[wv.vocab[word].ind
            all_words.add(wv.vocab[word].index)

    if not mean:
        logging.warning("cannot compute similarity wit
        # FIXME: remove these examples in pre-processi
        return np.zeros(wv.vector_size,)
```

word_averaging

We will tokenize the text and apply the tokenization to "post" column, and apply word vector averaging to tokenized text.

```
def w2v_tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text, language='eng
        for word in nltk.word_tokenize(sent, language=
            if len(word) < 2:
                continue
            tokens.append(word)
    return tokens

train, test = train_test_split(df, test_size=0.3, rand

```

w2v_tokenize_text

Its time to see how logistic regression classifiers performs on these word-averaging document features.

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(n_jobs=1, C=1e5)
```

```
logreg = logreg.fit(X_train_word_average, train['tags'])
y_pred = logreg.predict(X_test_word_average)
print('accuracy %s' % accuracy_score(y_pred, test.tags))
print(classification_report(test.tags,
y_pred,target_names=my_tags))
```

```
accuracy 0.6379166666666667
               precision    recall  f1-score   support

         java       0.63      0.59      0.61       613
         html       0.73      0.76      0.75       620
      asp.net       0.65      0.67      0.66       587
           c#       0.53      0.52      0.52       586
ruby-on-rails       0.70      0.77      0.73       599
       jquery       0.44      0.39      0.41       589
        mysql       0.65      0.61      0.63       594
          php       0.73      0.80      0.76       610
          ios       0.60      0.61      0.61       617
   javascript       0.56      0.52      0.54       587
       python       0.55      0.50      0.52       611
            c       0.61      0.61      0.61       594
          css       0.65      0.65      0.65       619
      android       0.60      0.57      0.59       574
       iphone       0.70      0.71      0.71       584
          sql       0.42      0.42      0.42       578
  objective-c       0.68      0.71      0.70       591
          c++       0.76      0.78      0.77       608
    angularjs       0.82      0.83      0.82       638
         .net       0.66      0.71      0.68       601

  avg / total       0.63      0.64      0.64     12000
```

Figure 10

It was disappointing, worst we have seen so far.

Doc2vec and Logistic Regression

The same idea of word2vec can be extended to documents where instead of learning feature representations for words, we learn it for sentences or documents. To get a general idea of a word2vec, think of it as a mathematical average of the word vector representations of all the words in the document. Doc2Vec extends the idea of word2vec, however words can only capture so much, there are times when we need relationships between documents and not just words.

The way to train doc2vec model for our Stack Overflow questions and tags data is very similar with when we train Multi-Class Text Classification with Doc2vec and Logistic Regression.

First, we label the sentences. Gensim's Doc2Vec implementation requires each document/paragraph to have a label associated with it. and we do this by using the `TaggedDocument` method. The format will be "TRAIN_i" or "TEST_i" where "i" is a dummy index of the post.

```
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
import gensim
from gensim.models.doc2vec import TaggedDocument
import re

def label_sentences(corpus, label_type):
    """
    Gensim's Doc2Vec implementation requires each docu
    We do this by using the TaggedDocument method. The
    a dummy index of the post.
    """
    labeled = []
```

label_sentences

According to Gensim doc2vec tutorial, its doc2vec class was trained on the entire data, and we will do the same. Let's have a look what the tagged document looks like:

```
all_data[:2]
```

```
[TaggedDocument(words=['fulltext', 'search', 'php', 'pdo', 'returning', 'result', 'searched', 'lot', 'matter', 'find', 'wrong',
'setup', 'trying', 'fulltext', 'search', 'using', 'pdo', 'php', 'get', 'results', 'error', 'messages', 'table', 'contains', 'cu
stomer', 'details', 'id', 'int', '11', 'auto_increment', 'name', 'varchar', '150', 'lastname', 'varchar', '150', 'company', 'va
rchar', '250', 'adress', 'varchar', '150', 'postcode', 'int', '5', 'city', 'varchar', '150', 'email', 'varchar', '250', 'phon
e', 'varchar', '20', 'orgnr', 'varchar', '15', 'timestamp', 'timestamp', 'current_timestamp', 'run', 'sqlquery', 'alter', 'tabl
e', 'system_customer', 'add', 'fulltext', 'name', 'lastname', 'except', 'columns', 'id', 'postcode', 'timestamp', 'signs', 'tro
uble', 'far', 'idea', 'problem', 'lies', 'db', 'configuration', 'php', 'code', 'goes', 'php', 'sth', 'dbhprepare', 'select', 'n
ame', 'lastname', 'company', 'adress', 'city', 'phone', 'email', 'orgnr', 'db_pre', 'customer', 'match', 'name', 'lastname', 'c
ompany', 'adress', 'city', 'phone', 'email', 'orgnr', 'search', 'boolean', 'mode', 'bind', 'placeholders', 'sthbindparam', 'sea
rch', 'data', 'sthexecute', 'rows', 'sthfetchall', 'testing', 'print_r', 'dbherrorinfo', 'empty', 'rows', 'echo', 'else', 'ech
o', 'foreach', 'rows', 'row', 'echo', 'tr', 'datahref', 'new_orderphp', 'cid', 'row', 'id', 'echo', 'td', 'row', 'name', 'td',
'echo', 'td', 'row', 'lastname', 'td', 'echo', 'td', 'row', 'company', 'td', 'echo', 'td', 'row', 'phone', 'td', 'echo', 'td',
'row', 'email', 'td', 'echo', 'td', 'date', 'ymd', 'strtotime', 'row', 'timestamp', 'td', 'echo', 'tr', 'echo', 'tried', 'chang
e', 'parameter', 'searchquery', 'string', 'like', 'testcompany', 'somename', 'boolean', 'mode', 'also', 'read', 'word', 'foun
d', '50', 'rows', 'counts', 'common', 'word', 'pretty', 'sure', 'case', 'uses', 'specific', 'words', 'table', 'uses', 'myisam',
'engine', 'get', 'results', 'error', 'messages', 'please', 'help', 'point', 'wrong', 'thank'], tags=['Train_0']),
 TaggedDocument(words=['select', 'everything', '1', 'table', 'x', 'rows', 'another', 'im', 'making', 'join', 'query', 'like',
'select', 'clothes', 'c', 'join', 'style', 'cstyleid', 'ssylelid', 'clothesid', '19', 'dont', 'want', 'select', 'everything',
'style', 'want', 'select', 'everything', 'clothes', '20', 'rows', 'select', '1', 'row', '10', 'style', 'easyest', 'way', 'witho
ut', 'select', 'every', 'row', 'clothes', '20', 'things', 'select', 'like', 'select', 'cid', 'cdescription', 'cname', 'csize',
'cbrand', 'sname', 'clothes', 'c', 'join', 'style', 'cstyleid', 'stsylelid', 'clothesid', '19', 'would', 'fastest', 'way', 'pos
sibillity'], tags=['Train_1'])]
```

Figure 11

When training the doc2vec, we will vary the following parameters:

- `dm=0` , distributed bag of words (DBOW) is used.
- `vector_size=300` , 300 vector dimensional feature vectors.
- `negative=5` , specifies how many "noise words" should be drawn.

- `min_count=1` , ignores all words with total frequency lower than this.
- `alpha=0.065` , the initial learning rate.

We initialize the model and train for 30 epochs.

```
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5,
model_dbow.build_vocab([x for x in tqdm(all_data)])

for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(all
    model_dbow.alpha -= 0.002
```

train_doc2vec

Next, we get vectors from trained doc2vec model.

```
def get_vectors(model, corpus_size, vectors_size, vect
    """
    Get vectors from trained doc2vec model
    :param doc2vec_model: Trained Doc2Vec model
    :param corpus_size: Size of the data
    :param vectors_size: Size of the embedding vectors
    :param vectors_type: Training or Testing vectors
    :return: list of vectors
    """
    vectors = np.zeros((corpus_size, vectors_size))
    for i in range(0, corpus_size):
        prefix = vectors_type + '_' + str(i)
```

get_vectors

Finally, we get a logistic regression model trained by the doc2vec features.

```
logreg = LogisticRegression(n_jobs=1, C=1e5)
logreg.fit(train_vectors_dbow, y_train)
logreg = logreg.fit(train_vectors_dbow, y_train)
y_pred = logreg.predict(test_vectors_dbow)
print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test,
y_pred,target_names=my_tags))
```

```
accuracy 0.8045
               precision    recall  f1-score   support

         java       0.73      0.68      0.70       589
         html       0.89      0.91      0.90       661
      asp.net       0.93      0.94      0.94       606
           c#       0.80      0.80      0.80       613
ruby-on-rails       0.83      0.90      0.86       601
       jquery       0.72      0.71      0.72       585
        mysql       0.87      0.81      0.84       621
          php       0.81      0.84      0.82       587
          ios       0.68      0.67      0.67       560
   javascript       0.69      0.63      0.66       611
       python       0.63      0.65      0.64       593
            c       0.81      0.83      0.82       581
          css       0.81      0.77      0.79       608
      android       0.84      0.85      0.84       593
       iphone       0.84      0.82      0.83       592
          sql       0.68      0.65      0.66       597
  objective-c       0.84      0.86      0.85       604
          c++       0.90      0.95      0.92       610
    angularjs       0.93      0.96      0.95       595
         .net       0.81      0.84      0.82       593

  avg / total       0.80      0.80      0.80     12000
```

Figure 12

We achieve an accuracy score of 80% which is 1% higher than SVM.

BOW with Keras

Finally, we are going to do a text classification with Keras which is a Python Deep Learning library.

The following code were largely taken from a Google workshop. The process is like this:

- Separate the data into training and test sets.
- Use `tokenizer` methods to count the unique words in our vocabulary and assign each of those words to indices.
- Calling `fit_on_texts()` automatically creates a word index lookup of our vocabulary.
- We limit our vocabulary to the top words by passing a `num_words` param to the tokenizer.
- With our tokenizer, we can now use the `texts_to_matrix` method to create the training data that we'll pass our model.
- We feed a one-hot vector to our model.
- After we transform our features and labels in a format Keras can read, we are ready to build our text classification model.
- When we build our model, all we need to do is tell Keras the shape of our input data, output data, and the type of each layer. keras will look after the rest.

- When training the model, we'll call the `fit()` method, pass it our training data and labels, batch size and epochs.

```
import itertools
import os

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.preprocessing import LabelBinarizer, Labe
from sklearn.metrics import confusion_matrix

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.preprocessing import text, sequence
from keras import utils

train_size = int(len(df) * .7)
train_posts = df['post'][:train_size]
train_tags = df['tags'][:train_size]

test_posts = df['post'][train_size:]
test_tags = df['tags'][train_size:]

max_words = 1000
tokenize = text.Tokenizer(num_words=max_words, char_le
tokenize.fit_on_texts(train_posts) # only fit on train

x_train = tokenize.texts_to_matrix(train_posts)
x_test = tokenize.texts_to_matrix(test_posts)

encoder = LabelEncoder()
encoder.fit(train_tags)
y_train = encoder.transform(train_tags)
y_test = encoder.transform(test_tags)

num_classes = np.max(y_train) + 1
y_train = utils.to_categorical(y_train, num_classes)
```

keras_training

```
Train on 25200 samples, validate on 2800 samples
Epoch 1/2
25200/25200 [==============================] - 11s 442us/step - loss: 1.0261 - acc: 0.7180 - val_loss: 0.6658 - val_acc: 0.7975
Epoch 2/2
25200/25200 [==============================] - 11s 434us/step - loss: 0.5675 - acc: 0.8190 - val_loss: 0.6625 - val_acc: 0.7868
```

Figure 13

The accuracy is:

```
score = model.evaluate(x_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
12000/12000 [==============================] - 1s 76us/step
Test accuracy: 0.7955833333333333
```

Figure 14

So, which model is the best for this particular data set? I will leave it to you to decide.

Jupyter notebook can be found on Github. Have a productive day!

References:

https://github.com/RaRe-Technologies/movie-plots-by-genre/blob/master/ipynb_with_output/Document%20classification%20with%20word%20embeddings%20tutorial%20-%20with%20output.ipynb

https://github.com/tensorflow/workshops/blob/master/extras/keras-bag-of-words/keras-bow-model.ipynb

https://datascience.stackexchange.com/questions/20076/word2vec-vs-sentence2vec-vs-doc2vec