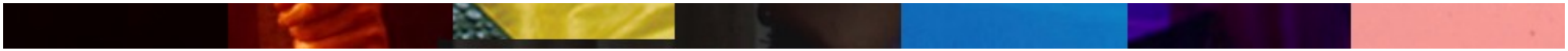

Implementing multi-class text classification with Doc2Vec

Dipika Baad Feb 3, 2019 · 6 min read

Introduction

In this post, you will learn how to classify text documents into different categories while using Doc2Vec to represent the documents. We will learn this with an easy to understand example of classifying the movie plots by genre using Doc2vec for feature representation and using logistic regression as a classification algorithm. The **movie dataset** contains short movie plot descriptions and the labels for them represents the genre. There are **six genres** present in the dataset:

1. Sci-fi
2. Action
3. Comedy

4. Fantasy
5. Animation
6. Romance

What is Doc2Vec and why do we need it?

Well, why should we choose **doc2vec** representation method rather than using widely known bag-of-words(**BOW**) method. For complex text classification algorithms, **BOW** would not be suitable as it lacks the capability to capture the semantics and syntactic order of words in the text. Thus using them as feature input to machine learning algorithm will not yield significant performance. Doc2Vec on the other hand is able to detect relationships among words and understands the semantics of the text. Doc2Vec is an unsupervised algorithm that learns fixed-length feature vectors for paragraphs/documents/texts. For understanding the basic working of **doc2vec** , how the **word2vec** works needs to be understood as it uses the same logic except the document specific vector is the added feature vector. For more details on this, you can read this blog. Now that we know why we are using it and how doc2vec will be used in this program, we can go on to the next stage of actually implementing the classifier.

Let's get started on building a movie plot classifier!!

Implementation

Three main parts involved for this are as follows:

1. Loading and preparing the text data
2. Getting the feature vector using **doc2vec** model
3. Training the Classifier

Here are the first three rows of the input csv file:

```
,movieId,plot,tag


0,1,"A little boy named Andy loves to be in his room, playing with
his toys, especially his doll named ""Woody"". But, what do the toys
do when Andy is not with them, they come to life. Woody believes that
he has life (as a toy) good. However, he must worry about Andy's
family moving, and what Woody does not know is about Andy's birthday
party. Woody does not realize that Andy's mother gave him an action
figure known as Buzz Lightyear, who does not believe that he is a
toy, and quickly becomes Andy's new favorite toy. Woody, who is now
consumed with jealousy, tries to get rid of Buzz. Then, both Woody
and Buzz are now lost. They must find a way to get back to Andy
before he moves without them, but they will have to pass through a
ruthless toy killer, Sid Phillips.",animation


1,2,"When two kids find and play a magical board game, they release a
man trapped for decades in it and a host of dangers that can only be
stopped by finishing the game.",fantasy
```

Importing the required libraries:

We use multiprocessing for utilizing all the cores for faster training with Doc2Vec. Package tqdm is used for showing the progress bar while training. We use gensim package for Doc2Vec. For classification purpose Logistic regression from scikit-learn is used. NLTK package is used for tokenizing task.

```
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split


from sklearn.linear_model import LogisticRegression
from sklearn import utils
import csv
from tqdm import tqdm
import multiprocessing


import nltk
from nltk.corpus import stopwords
```

1. Reading and Preparing the text data

The following code is for reading the data from the csv and function for tokenizing which will be used while creating training and testing

documents as input to the **doc2vec** model. The data has 2448 rows, we have chosen first 2000 rows for training and the rest for testing.

```
tqdm.pandas(desc="progress-bar")


# Function for tokenizing


def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens


# Initializing the variables


train_documents = []
test_documents = []
i = 0


# Associating the tags(labels) with numbers


tags_index = {'sci-fi': 1 , 'action': 2, 'comedy': 3, 'fantasy': 4,
'animation': 5, 'romance': 6}


#Reading the file


FILEPATH = 'data/tagged_plots_movielens.csv'
with open(FILEPATH, 'r') as csvfile:
with open('data/tagged_plots_movielens.csv', 'r') as csvfile:
```

```
    moviereader = csv.reader(csvfile, delimiter=',', quotechar='"')
    for row in moviereader:
        if i == 0:
            i += 1
            continue
        i += 1
        if i <= 2000:


            train_documents.append(
TaggedDocument(words=tokenize_text(row[2]), tags=
[tags_index.get(row[3], 8)] ))


        else:
            test_documents.append(
TaggedDocument(words=tokenize_text(row[2]),
 tags=[tags_index.get(row[3], 8)]))


print(train_documents[0])
```

Output which is the training_document for the first row is the TaggedDocument object. This shows the tokens as the first argument which are tokens and labelID as the second argument (5: Animation) for TaggedDocument.

```
TaggedDocument(['little', 'boy', 'named', 'andy', 'loves', 'to',
'be', 'in', 'his', 'room', 'playing', 'with', 'his', 'toys',
'especially', 'his', 'doll', 'named', '``', 'woody', "''", 'but',
'what', 'do', 'the', 'toys', 'do', 'when', 'andy', 'is', 'not',
'with', 'them', 'they', 'come', 'to', 'life', 'woody', 'believes',
'that', 'he', 'has', 'life', 'as', 'toy', 'good', 'however', 'he',
'must', 'worry', 'about', 'andy', "'s", 'family', 'moving', 'and',
```

```
'what', 'woody', 'does', 'not', 'know', 'is', 'about', 'andy', "'s",
'birthday', 'party', 'woody', 'does', 'not', 'realize', 'that',
'andy', "'s", 'mother', 'gave', 'him', 'an', 'action', 'figure',
'known', 'as', 'buzz', 'lightyear', 'who', 'does', 'not', 'believe',
'that', 'he', 'is', 'toy', 'and', 'quickly', 'becomes', 'andy', "'s",
'new', 'favorite', 'toy', 'woody', 'who', 'is', 'now', 'consumed',
'with', 'jealousy', 'tries', 'to', 'get', 'rid', 'of', 'buzz',
'then', 'both', 'woody', 'and', 'buzz', 'are', 'now', 'lost', 'they',
'must', 'find', 'way', 'to', 'get', 'back', 'to', 'andy', 'before',
'he', 'moves', 'without', 'them', 'but', 'they', 'will', 'have',
'to', 'pass', 'through', 'ruthless', 'toy', 'killer', 'sid',
'phillips'], [5])
```

2. Getting the feature vector from doc2vec model

Next, we initialize the gensim doc2vec model and train for 30 epochs. This process is pretty simple. Doc2Vec architecture also has two algorithms like word2vec and they are the corresponding algorithms for those two algorithms namely 'Continuous Bag of Words'(CBOW) and 'Skip-Gram'(SG). One of the algotihms in doc2vec is called Paragraph Vector - Distributed Bag of Words (PV-DBOW) which is similar to SG model in word2vec except that additional paragraph id vector is added. Here neural network is trained to predict the vector of surrounding words in the given paragraph and paragraph id vector based on a given word in the paragraph. The second algorithm is Paragraph Vector (PV-DM) which is similar to CBOW in word vector.

A few important parameters for the **doc2vec** model consists of:

`dm` ({0,1}, optional) 1: PV-DM, 0: PV-DBOW

`vector_size` Dimensionality of the feature vector (we chose 300)

`workers` this is number of threads which we assigned number of cores

The rest of the parameter details could be found here. After initializing, we build the vocabulary using the `train_documents`

```
cores = multiprocessing.cpu_count()

model_dbow = Doc2Vec(dm=1, vector_size=300, negative=5, hs=0,
min_count=2, sample = 0, workers=cores, alpha=0.025, min_alpha=0.001)
model_dbow.build_vocab([x for x in tqdm(train_documents)])


train_documents  = utils.shuffle(train_documents)
model_dbow.train(train_documents,total_examples=len(train_documents),
epochs=30)


def vector_for_learning(model, input_docs):
    sents = input_docs
    targets, feature_vectors = zip(*[(doc.tags[0],
model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, feature_vectors


model_dbow.save('./movieModel.d2v')
```

3. Training the Classifier

Finally, using the feature vector building function above we train the Logistic Regression Classifier. Here we used the feature vector generated for the `train_documents` while training and used the feature vectors of `test_documents` in the prediction phase.

```
y_train, X_train = vector_for_learning(model_dbow, train_documents)
y_test, X_test = vector_for_learning(model_dbow, test_documents)

logreg = LogisticRegression(n_jobs=1, C=1e5)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)


print('Testing accuracy for movie plots%s' % accuracy_score(y_test,
y_pred))
print('Testing F1 score for movie plots: {}'.format(f1_score(y_test,
y_pred, average='weighted')))
```

The output when `dm=1` is as follows:

```
Testing accuracy 0.42316258351893093
Testing F1 score: 0.41259684559985876
```

This accuracy is obtained only with just few records with very short text and hence, this can be improved with adding better features like n-grams, using stopwords to remove noise.

Further Developments

Building on this, you can experiment more easily using with other datasets and changing the algorithms for doc2vec is as easy as changing the `dm` paramter. Hope this helps you get started on your first project on text classification with Doc2Vec!

Author: Dipika Baad

Machine Learning Naturallanguageprocessing Classification Problem Doc2vec