Multi-Class Text Classification with Doc2Vec & Logistic Regression

The goal is to classify consumer finance complaints into 12 pre-defined classes using Doc2Vec and Logistic Regression

Susan Li · Follow

Published in Towards Data Science · 6 min read · Sep 17, 2018

Doc2vec is an NLP tool for representing documents as a vector and is a generalizing of the word2vec method.

In order to understand doc2vec, it is advisable to understand word2vec approach. However, the complete mathematical details is out of scope of this article. If you are new to word2vec and doc2vec, the following resources can help you to get start:

- Distributed Representations of Words and Phrases and their Compositionality
- Distributed Representations of Sentences and Documents
- A gentle introduction to Doc2Vec
- Gensim Doc2Vec Tutorial on the IMDB Sentiment Dataset
- Document classification with word embeddings tutorial

Using the same data set when we did Multi-Class Text Classification with Scikit-Learn, In this article, we'll classify complaint narrative by product using doc2vec techniques in Gensim. Let's get started!

The Data

The goal is to classify consumer finance complaints into 12 pre-defined classes. The data can be downloaded from data.gov.

```
import pandas as pd
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
import gensim
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('Consumer_Complaints.csv')
df = df[['Consumer complaint narrative','Product']]
df = df[pd.notnull(df['Consumer complaint narrative'])]
df.rename(columns = {'Consumer complaint narrative':'narrative'},
inplace = True)
df.head(10)
```

	narrative	Product
1	When my loan was switched over to Navient i wa...	Student loan
2	I tried to sign up for a spending monitoring p...	Credit card or prepaid card
7	My mortgage is with BB & T Bank, recently I ha...	Mortgage
14	The entire lending experience with Citizens Ba...	Mortgage
15	My credit score has gone down XXXX points in t...	Credit reporting
17	I few months back I contacted XXXX in regards...	Credit reporting, credit repair services, or o...
28	I " m a victim of fraud and I have a file wit...	Credit reporting, credit repair services, or o...
30	My mortgage is owned by XXXX, we have painfull...	Mortgage
32	I have been disputing a Bankruptcy on my credi...	Credit reporting, credit repair services, or o...
54	Today I received a phone call from a number li...	Debt collection

Figure 1

After remove null values in narrative columns, we will need to re-index the data frame.

```
df.shape
```

(318718, 2)

```
df.index = range(318718)
df['narrative'].apply(lambda x: len(x.split(' '))).sum()
```

63420212

We have over 63 million words, it is a relatively large data set.

Exploring

```
cnt_pro = df['Product'].value_counts()

plt.figure(figsize=(12,4))
sns.barplot(cnt_pro.index, cnt_pro.values, alpha=0.8)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Product', fontsize=12)
plt.xticks(rotation=90)
plt.show();
```

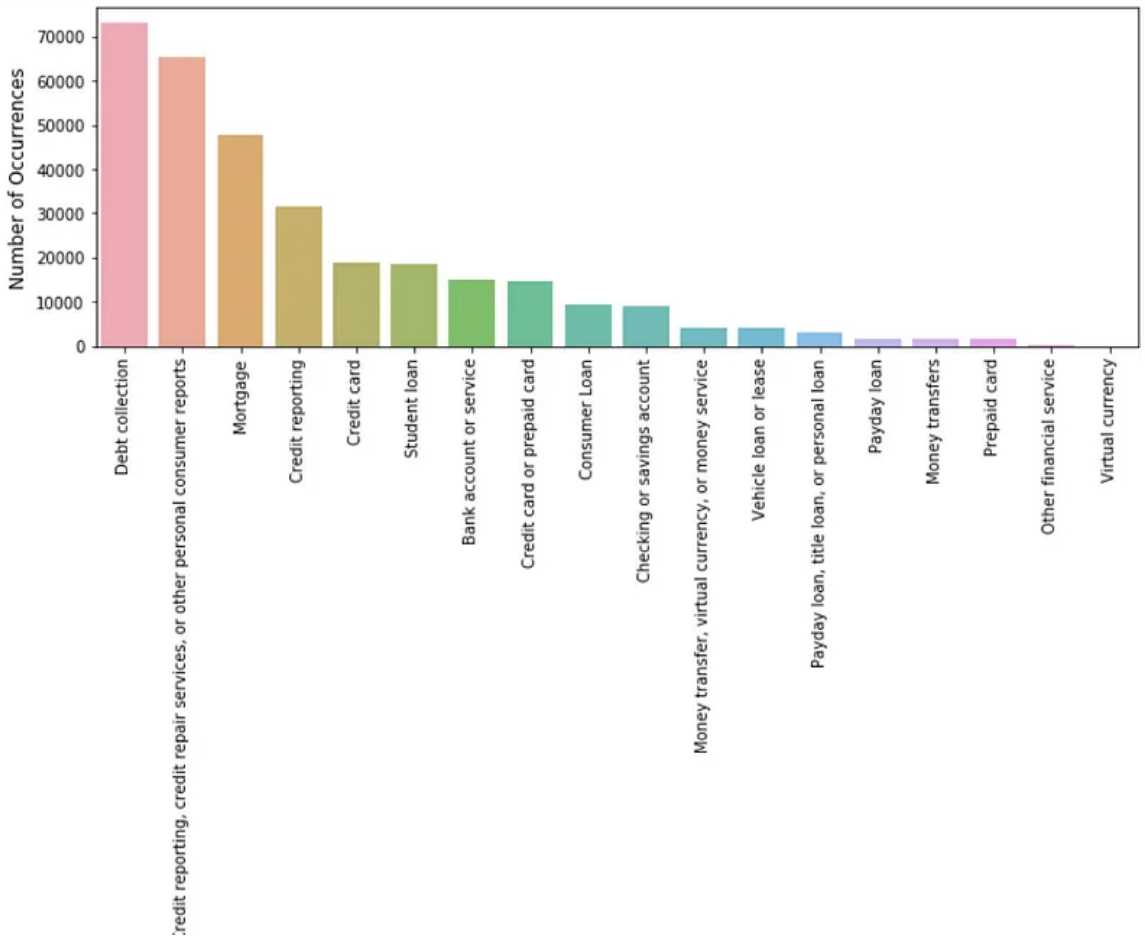

Figure 2

The classes are imbalanced, however, a naive classifier that predicts everything to be Debt collection will only achieve over 20% accuracy.

Let's have a look a few examples of complaint narrative and its associated product.

```
def print_complaint(index):
    example = df[df.index == index][['narrative',
'Product']].values[0]
    if len(example) > 0:
        print(example[0])
        print('Product:', example[1])

print_complaint(12)
```

```
I APPARENTLY HAVE AN OUTSTANDING DEBT WITH XXXX XXXX. THEY SENT THE ACCOUNT TO XXXX  XXXX XXXX. THIS COLLECTION COMPANY CALLS
Y WORK AND CELL SEVERAL TIMES A DAY. UP TO 6 TIMES A DAY. I TRY TO TELL THE COMPANY TO STOP CALLING ME AND THEY HANG UP LAUGH
G. I HAVE NOT RECEIVED ANY PAPERWORK FROM THIS COMPANY. AND XXXX XXXX REFUSE TO SPEAK WITH ME. I HAVE MAILED IN A CEASE AND D
IST, AND I HAVE CALLED THEM TO GET THEIR FAX NUMBER TO SEND IN THE PAPERWORK-THEY KEEP GIVING ME DIFFERENT ONES ( FAX # XXXX
I DON'T OWE ANY MONEY TO XXXX, I CLOSED THE ACCOUNT WITH A XXXX BALANCE AND TURNED IN ALL OF MY EQUIPMENT THAT I WAS RENTING.
ND SO DID MY WIFE.
Product: Debt collection
```

Figure 3

```
print_complaint(20)
```

```
In late XXXX of 2017 I requested a balance transfer from my Comenity Bank Visa card to pay off a high balance on another cred
card I have. On XXXX XXXX, 2017 the charge posted to my Comenity Bank Visa card account however, as of today ( XXXX XXXX, 201
7 ) they have never sent the money to pay off the other credit card. I have tried to resolve the problem with several phone c
ls to Comenity Bank but no one has been able to give a reasonable explanation of why they are charging my credit card account
ithout actually sending the money to pay off the other credit card. I have also sent them a written dispute of the charge but
ave received no response yet.
Product: Credit card or prepaid card
```

Figure 4

Text Preprocessing

Below we define a function to convert text to lower-case and strip punctuation/symbols from words and so on.

```
from bs4 import BeautifulSoup
def cleanText(text):
    text = BeautifulSoup(text, "lxml").text
    text = re.sub(r'\|\|\|', r' ', text)
    text = re.sub(r'http\S+', r'<URL>', text)
    text = text.lower()
    text = text.replace('x', '')
    return text
df['narrative'] = df['narrative'].apply(cleanText)
```

The following steps include train/test split of 70/30, remove stop-words and tokenize text using NLTK tokenizer. For our first try we tag every complaint narrative with its product.

```
train, test = train_test_split(df, test_size=0.3, random_state=42)
```

```
import nltk
from nltk.corpus import stopwords
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
```

```
train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['narrative']),
tags=[r.Product]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['narrative']),
tags=[r.Product]), axis=1)
```

This is what a training entry looks like — an example complaint narrative tagged by “Credit reporting”.

```
train_tagged.values[30]
```

```
TaggedDocument(words=['had', 'bankruptcy', 'years', 'ago', 'and', 'it', 'is', 'still', 'showing', 'up', 'on', 'equifa', 'whi
h', 'is', 'preventing', 'me', 'from', 'buying', 'home', 'at', 'good', 'rate', 'they', 'need', 'to', 'take', 'it', 'off', 'li
e', 'did', 'so', 'my', 'score', 'will', 'be'], tags=['Credit reporting'])
```

Figure 5

Set-up Doc2Vec Training & Evaluation Models

First, we instantiate a doc2vec model — Distributed Bag of Words (DBOW). In the word2vec architecture, the two algorithm names are “continuous bag of words” (CBOW) and “skip-gram” (SG); in the doc2vec architecture, the corresponding algorithms are “distributed memory” (DM) and “distributed bag of words” (DBOW).

Distributed Bag of Words (DBOW)

DBOW is the doc2vec model analogous to Skip-gram model in word2vec. The paragraph vectors are obtained by training a neural network on the task of predicting a probability distribution of words in a paragraph given a randomly-sampled word from the paragraph.

We will vary the following parameters:

- If `dm=0`, distributed bag of words (PV-DBOW) is used; if `dm=1`,‘distributed memory’ (PV-DM) is used.
- 300- dimensional feature vectors.
- `min_count=2`, ignores all words with total frequency lower than this.
- `negative=5`, specifies how many “noise words” should be drawn.
- `hs=0`, and negative is non-zero, negative sampling will be used.
- `sample=0`, the threshold for configuring which higher-frequency words are randomly down sampled.
- `workers=cores`, use these many worker threads to train the model (=faster training with multicore machines).

```
import multiprocessing
cores = multiprocessing.cpu_count()
```

Building a Vocabulary

```
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0,
min_count=2, sample = 0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 223102/223102 [00:00<00:00, 2855261.21it/s]
```

Figure 6

Training a doc2vec model is rather straight-forward in Gensim, we initialize the model and train for 30 epochs.

```
%%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in
tqdm(train_tagged.values)]),
total_examples=len(train_tagged.values), epochs=1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

```
100%|██████████| 223102/223102 [00:00<00:00, 2379615.48it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2849340.50it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2100357.58it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1596907.77it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2576198.16it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1961591.50it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1799270.90it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1216525.65it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1879469.14it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2159058.65it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1851940.21it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1868324.12it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2222950.33it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2035760.37it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1791036.78it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039549.64it/s]
100%|██████████| 223102/223102 [00:00<00:00, 3569332.45it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2855374.47it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2855418.04it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2379567.07it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039580.76it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2322399.87it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2517534.79it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1888184.11it/s]
100%|██████████| 223102/223102 [00:00<00:00, 919457.47it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2060277.88it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1885517.07it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1539867.75it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2726717.42it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2033057.94it/s]
Wall time: 18min 24s
```

Figure 7

Building the Final Vector Feature for the Classifier

```
def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, regressorsdef vec_for_learning(model,
tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, regressors
```

Train the Logistic Regression Classifier.

```
y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)

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

from sklearn.metrics import accuracy_score, f1_score

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

Testing accuracy 0.6683609437751004

Testing F1 score: 0.651646431211616

Distributed Memory (DM)

Distributed Memory (DM) acts as a memory that remembers what is missing from the current context — or as the topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document. We again instantiate a Doc2Vec model with a vector size with 300 words and iterating over the training corpus 30 times.

```
model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10,
negative=5, min_count=1, workers=5, alpha=0.065, min_alpha=0.065)
model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 223102/223102 [00:00<00:00, 1886113.74it/s]
```

Figure 8

```
%%time
for epoch in range(30):
    model_dmm.train(utils.shuffle([x for x in
tqdm(train_tagged.values)]),
total_examples=len(train_tagged.values), epochs=1)
    model_dmm.alpha -= 0.002
    model_dmm.min_alpha = model_dmm.alpha
```

```
100%|██████████| 223102/223102 [00:00<00:00, 2855531.31it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1578083.26it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1745168.51it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1854244.79it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1893698.00it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1885653.85it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1869836.11it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1869832.37it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2445159.40it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2022787.35it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1165813.40it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2379542.86it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2379585.22it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1784628.78it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1784628.78it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1940508.05it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039602.98it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1104039.77it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039647.44it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039562.97it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2379573.12it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2855418.04it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2379506.56it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2039554.08it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1784632.19it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1751177.79it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1892361.48it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1959722.32it/s]
100%|██████████| 223102/223102 [00:00<00:00, 1649827.85it/s]
100%|██████████| 223102/223102 [00:00<00:00, 2778023.03it/s]

Wall time: 33min 28s
```

Figure 9

Train the Logistic Regression Classifier

```
y_train, X_train = vec_for_learning(model_dmm, train_tagged)
y_test, X_test = vec_for_learning(model_dmm, test_tagged)

logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

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

Testing accuracy 0.47498326639892907

Testing F1 score: 0.4445833078167434

Model Pairing

According to Gensim doc2vec tutorial on the IMDB sentiment data set, combining a paragraph vector from Distributed Bag of Words (DBOW) and Distributed Memory (DM) improves performance. We will follow, pairing the models together for evaluation.

First, we delete temporary training data to free up RAM.

```
model_dbow.delete_temporary_training_data(keep_doctags_vectors=True,
keep_inference=True)
model_dmm.delete_temporary_training_data(keep_doctags_vectors=True,
keep_inference=True)
```

Concatenate two models.

```
from gensim.test.test_doc2vec import ConcatenatedDoc2Vec
new_model = ConcatenatedDoc2Vec([model_dbow, model_dmm])
```

Building feature vectors.

```
def get_vectors(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words, steps=20)) for doc in sents])
    return targets, regressors
```

Train the Logistic Regression

```
y_train, X_train = get_vectors(new_model, train_tagged)
y_test, X_test = get_vectors(new_model, test_tagged)

logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

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

Testing accuracy 0.6778572623828648

Testing F1 score: 0.664561533967402

The result improved by 1%.

For this article, I used training set to train doc2vec, however, in Gensim's tutorial, the whole data set was used for training, I tried that approach, using the whole data set to train doc2vec classifier for our consumer complaint classification, I was able to achieve 70% accuracy. You can find that notebook here, it is a little different approach.

The Jupyter notebook for the above analysis can be found on Github. I look forward to hearing any questions.