Luis Alfredo Jaramillo Uday

*Sistemas Informáticos y Computación*

*Universidad Técnica Particular de Loja*

*Loja – Ecuador*

[*lajaramillo5@utpl.edu.ec*](mailto:lajaramillo5@utpl.edu.ec)

Un modelo Doc2Vec

*Abstract*— En el siguiente document veremos plasmados los Diseños del modelo Doc2Vec que es una extencion de Word2Vec los cuales son modelos latentes para predecir clasificaciones de elementos de usuario. Se usara 3 modelos para comparar un Corpus de medicina.

# **INTRODUCCIÓN**

E

N el presente trabajo abordaremos una metodología, para abordar el problema en cuestión, que se trata del análisis semántico de un corpus de medicina el cual esta en español, para ello se usuara el uso y el entrenamiento de tres modelos los cuales nos va ayudar a representar de mejor manera la representación que deseamos.

Se deberá hacer un análisis minucioso sobre todo el corpus el cual nos va a servir para ayudar en futuras consultar o emplear estas técnicas en otros entornos.

# **desarrollo**

Para el presente proyecto, debemos conocer las siguientes herramientas y definiciones:

## **Doc2vec**

E

Hay que saber que Doc2vec es una extensión de Word2vec que es ideal para codificar documentos completos en lugar de solo palabras individuales.

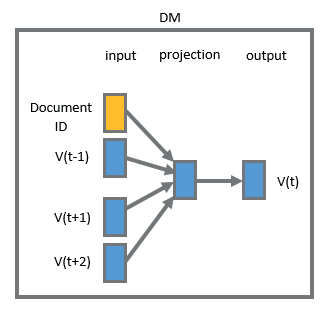
Los vectores Doc2Vec representan el tema o el significado general de un documento. El nombre del documento, como el nombre del archivo o la ID del archivo, será la entrada, y la ventana deslizante de las palabras del documento serán consideradas como la salida.

* 1. **Modelos**

Al igual que word2vec, docevec cuenta con dos modelos que son muy similares o variantes de los modelos CBOW y SKIP GRAM

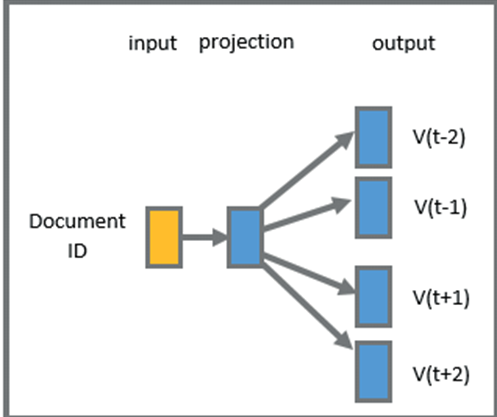
• modelo distribuido-memoria

Este es similar al CBOW, este modelo lo que intenta es definir una palabra, a partir de sus palabras de contexto que se encuentran a su alrededor y con la adición de un id de párrafo.



• El modelo de bolsa de palabras distribuida

basado en el modelo de omisión de gramo de Word2vec, con una excepción en lugar de usar la palabra de destino como entrada, toma la identificación del documento como entrada e intenta predecir palabras muestreadas al azar del documento



El nombre del archivo de entrada no es muy importante. El modelo Doc2Vec puede predecir las palabras del documento en función de su nombre de archivo, el modelo Doc2Vec sabe qué palabras van juntas en un documento.

Doc2Vec utiliza las similitudes de palabras aprendidas durante el entrenamiento para construir un vector que prediga las palabras en el nuevo documento.

Una vez que obtengamos ese nuevo vector de documento, podemos comparar este vector de documento con otros vectores de documentos y encontrar qué vectores de documentos del pasado son los más similares.

* 1. ***Anaconda Navigator (Anaconda 3)***

Anaconda es una suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la ciencia de datos con Python

* 1. **Jupyter**

Es un IDE web queexiste para desarrollar software de código abierto

Jupyter Lab: Es un entorno de desarrollo interactivo basado en la web para portátiles, código y datos Jupyter

o Cuaderno de Jupyter: Es una aplicación web de código abierto que le permite crear y compartir documentos que contienen código, adicionalmente permite la: limpieza y transformación de datos, aprendizaje automático y mucho más.

* 1. **Gensim**

Gensim es una biblioteca de Python para modelado de temas, indexación de documentos y recuperación de similitudes con grandes corpus. El público objetivo es la comunidad de procesamiento del lenguaje natural (PNL) y recuperación de información (IR).

* 1. **Corpus**

Base de datos con documentos clínicos en español.

# **Trabajos relacionados sobre el tema**

A continuación, se presenta los diferentes proyectos de aplicación o recolección de datos a través de la herramienta de doc2vec, como ya se definió anteriormente la herramienta de doc2vec es un tipo de aplicación que se usa para el análisis de textos o corpus

* 1. **An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation**

Recientemente, Le y Mikolov (2014) propusieron doc2vec como una extensión de word2vec (Mikolov et al., 2013a) para aprender incrustaciones a nivel de documento. A pesar de los resultados prometedores en el documento original, otros han luchado por reproducir esos resultados. Este artículo presenta una evaluación empírica rigurosa de doc2vec en dos tareas. Comparamos doc2vec con dos líneas de base y dos metodologías de incrustación de documentos de última generación. Descubrimos que doc2vec funciona de manera sólida cuando se usan modelos entrenados en grandes cuerpos externos, y se puede mejorar aún más mediante el uso de incrustaciones de palabras pre-entrenadas. También proporcionamos recomendaciones sobre configuraciones de hiperparámetros para aplicaciones de propósito general, y lanzamos código fuente para inducir incrustaciones de documentos utilizando nuestros modelos doc2vec capacitados.

* 1. **Author Profiling with Doc2vec Neural Network-Based Document Embeddings**

Para determinar la demografía del autor de los textos en las redes sociales como Twitter, blogs y reseñas, utilizamos las incrustaciones de documentos doc2vec para entrenar un clasificador de regresión logística. Experimentamos con la identificación de la edad y el género en el autor del PAN que perfila corpus 2014-2016 en condiciones de género único y cruzado. Mostramos que, bajo ciertas configuraciones, las características basadas en la red neuronal superan a las características tradicionales cuando se usa el mismo clasificador. Nuestro método supera el estado actual del arte en algunos entornos, aunque los resultados actuales del estado del arte en esas tareas han sido bastante débiles.

* 1. **Comparing Semantic Models for Evaluating Automatic Document Summarization**

El enfoque principal de este trabajo es el examen del modelado semántico en el contexto del resumen automático de documentos y su evaluación. El área principal de nuestra investigación es el resumen extractivo, más específicamente, el resumen de opinión contrastante. Y como sucede con todas las tareas de resumen, la evaluación de su desempeño es un problema desafiante por sí solo. Hoy en día, la técnica de evaluación más utilizada es ROUGE (Estudio orientado al recuerdo para la evaluación de Gisting). Incluye medidas (como el recuento de n-gramos superpuestos o secuencias de palabras) para determinar automáticamente la calidad de los resúmenes al compararlos con resúmenes ideales hechos por el hombre. Sin embargo, estas medidas no tienen en cuenta la semántica de las palabras y, por lo tanto, los sinónimos no se tratan como iguales. Exploramos este problema experimentando con varios modelos de lenguaje, examinando su rendimiento en la tarea de calcular la similitud de documentos. En particular, elegimos cuatro modelos semánticos (LSA, LDA, Word2Vec y Doc2Vec) y un modelo basado en frecuencia (TfIdf), para extraer las características del documento. Luego, los experimentos se realizaron en nuestro conjunto de datos personalizado y los resultados de cada modelo se compararon con los valores de similitud evaluados por los anotadores humanos. También comparamos estos valores con los puntajes ROUGE y observamos las correlaciones entre ellos. El objetivo de nuestros experimentos es encontrar un modelo que pueda imitar mejor una estimación humana de la similitud del documento.

# **EXPERIMENTACIÓN**

De acuerdo a la investigación realizada sobre el uso de tecnologías para mejorar el uso de conocimientos de los estudiantes universitarios hemos identificado aspectos importantes acerca de la lo útil que es para los estudiantes y docentes tener herramientas que le sean utilis para llevar acabo una interaccion a distancia. Por ello en la investigación se ha identificado algunos datos que refleja la situación que presenta el tema de estas tecnologías digitales.

1. **Comparacion de texto xon grupo de textos.**

La primera experimentación se trata de entrenar un modelo que compare una carpeta llena de varios archivos de txt con un solo txt y ver la similitud del txt con los txt que contiene la carpeta.

1. **Explicación del experimento**

Para comenzar con el entrenamiento debemos importar los paquetes que necesitaremos para realizar el exxperimento

import gensim.models as gsm

from os import listdir

from os.path import isfile, join

from gensim.models.doc2vec import TaggedDocument

from collections import OrderedDict

Seguidamente procedemos a entrenar nuestro modelo introduciendo en el, la carpeta enla cual contenemos nuestros txt.

#ruta a los archivos del corpus de entrada

train\_corpus="./doc"

Procedemos a dar el respectivo etiquetado a losdiferentes txt que hemos ingresado en la carpeta

# etiquetar los archivos de texto

class DocIterator(object):

def \_\_init\_\_(self, doc\_list, labels\_list):

self.labels\_list = labels\_list

self.doc\_list = doc\_list

def \_\_iter\_\_(self):

for idx, doc in enumerate(self.doc\_list):

yield TaggedDocument(words=doc.split(), tags=[self.labels\_list[idx]])

docLabels = [f for f in listdir(train\_corpus) if f.endswith('.txt')]

print(docLabels)

data = []

for doc in docLabels:

data.append(open(join(train\_corpus, doc), 'r').read())

it = DocIterator(data, docLabels)

#train doc2vec model

model = gsm.Doc2Vec(vector\_size=300, window=10, min\_count=1, workers=11,alpha=0.025, min\_alpha=0.025) # use fixed learning rate

model.build\_vocab(it)

model.train(it, total\_examples=len(doc), epochs=20)

model.save("paper.model")

print("model is saved")

Con esto también conseguimos crear nuestro modelo y guardarlo.

Ahora debemos llamar el modelo que acabamos de crear y guardar para poderlo utilizar y ver que funciona correctamente.

Ahora procedemos a realizar la comparación entre un txt escogido por nosotros, utilizando el modelo que hemos entrenado.

import gensim

import os

import collections

import smart\_open

import random

from gensim import models,corpora,similarities

#ruta de acceso a los archivos de prueba

test\_paper="./textos/S0210-48062009000300013-4.txt"

new\_test = open(join(test\_paper), 'r').read().split()

#print(new\_test)

inferred\_docvec = m.infer\_vector(new\_test)

m.docvecs.most\_similar([inferred\_docvec], topn=3)

#print('%s:\n %s' % (model, m.docvecs.most\_similar(positive=[inferred\_docvec], topn=5)))

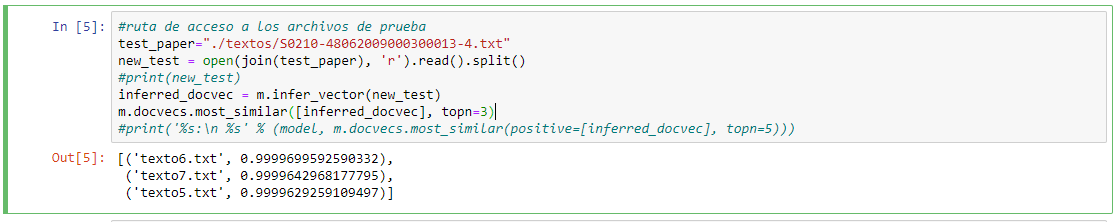
# cargando el modelo

model="paper.model"

m=gsm.Doc2Vec.load(model)

print("model is loaded")

1. **Evaluación de resultados**

Una vez realizado todo el procedimiento procedemos a ver los resultados obtenido

1. **Conclusiones de la Experimentación**

Las pruebas que hemos procedido hacer nos ha dado un resultado exitoso, obteniendo lo experado por el experimento.

model = gensim.models.doc2vec.Doc2Vec(vector\_size=50, min\_count=2, epochs=55)

model.build\_vocab(train\_corpus)

%time model.train(train\_corpus, total\_examples=model.corpus\_count, epochs=model.epochs)

# model.infer\_vector(['only', 'you', 'can', 'prevent', 'forest', 'fires'])

# **Comparacion de un archivo csv con un txt**

Ahora procederemos hacer la comparación de un csv que contiene el corpus completo con el que estamos trabajando, con un txt escogido por nosotros.

1. **Creacion de modelo**

Para esto debemos hacer la importación de los paquetes que necesitaremos para hacer el entrenamiento del modelo.

Ahora procedemos hacer el ingreso de los archivos necesarios

# Establecer nombres de archivo para trenes y datos de prueba

test\_data\_dir = '{}'.format(os.sep).join([gensim.\_\_path\_\_[0], 'test', 'test\_data'])

lee\_train\_file = 'hola.csv'

lee\_test\_file = './textos/S0004-06142006000500002-4.txt'

Procedemos hacer el entrenamiento de nuestro modelo

def read\_corpus(fname, tokens\_only=False):

with smart\_open.smart\_open(fname, encoding="iso-8859-1") as f:

for i, line in enumerate(f):

if tokens\_only:

yield gensim.utils.simple\_preprocess(line)

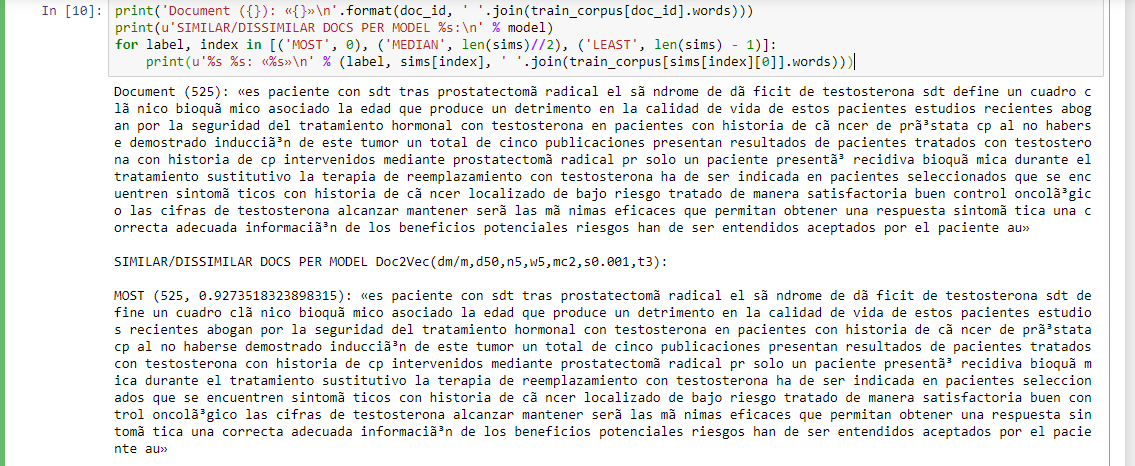
else:

# For training data, add tags

yield gensim.models.doc2vec.TaggedDocument(gensim.utils.simple\_preprocess(line), [i])

train\_corpus = list(read\_corpus(lee\_train\_file))

test\_corpus = list(read\_corpus(lee\_test\_file, tokens\_only=True))



ranks = []

second\_ranks = []

for doc\_id in range(len(train\_corpus)):

# infer new vectors for each document of the training corpus

inferred\_vector = model.infer\_vector(train\_corpus[doc\_id].words)

# Caculate simialrity of inferred vector with documents in the model

# sims is array containing result arranged in descending order.

# eg [(0, 0.9274195432662964), (48, 0.8267428278923035)] where first element of tuple is doc\_id and second is simialrity score

sims = model.docvecs.most\_similar([inferred\_vector], topn=len(model.docvecs))

# return the rank of the current document from most similar document list

# we are assuming the most similar document is itself and thus rank should be at o index

rank = [docid for docid, sim in sims].index(doc\_id)

ranks.append(rank)

second\_ranks.append(sims[1])

# cuente cómo se clasifica cada documento con respecto al corpus de capacitación

# la comprobación de un vector inferido contra un vector de entrenamiento es una especie de "comprobación de cordura"

# en cuanto a si el modelo se comporta de una manera útil y consistente, aunque no es un valor real de "precisión".

collections.Counter(ranks)

Despues del entrenamiento procedemos hacer la comparcion esta vez compararemos la similitud de texto y no de vectores.

print('Document ({}): «{}»\n'.format(doc\_id, ' '.join(train\_corpus[doc\_id].words)))

print(u'SIMILAR/DISSIMILAR DOCS PER MODEL %s:\n' % model)

for label, index in [('MOST', 0), ('MEDIAN', len(sims)//2), ('LEAST', len(sims) - 1)]:

print(u'%s %s: «%s»\n' % (label, sims[index], ' '.join(train\_corpus[sims[index][0]].words)))

**2. Evaluacion de resultados**

Una vez realziada todo el procedimiento de entrenamiento procedemos a reflejar los resultado obtenidos.

**3. Conclusiones de la Experimentación**

Las pruebas que hemos procedido hacer nos a servido de gran ayuda para saber que nuestro modelo se ha entrenado con facilidad, y nos ha dado los resultados esperados.

# **ANALISIS DE corpus completo**

Para el ultimo experimento entrenaremos esta vez un vocabulario que nos servirá para comparar las palabrs mas revelantes que contenga nuestro corpus de medicina

**1. Entenamiento de modelo**

Para esto procederemos hacer el ingreso de los paquetes necesarios para el entrenamiento del modelo

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

from gensim.models import Doc2Vec

from collections import namedtuple

import gensim.utils

from langdetect import detect

import re

import string

Seguido de eso procedemos a ingresar nuestro csv el cual contiene nuestro corpus.

imdbdata=pd.read\_csv('datos.csv')

data = np.array(imdbdata.titulo)

titles=np.array(imdbdata.descripcion)

imdbdata.head()

Ahora procederemos a entrenar nuestro modelo para eso haremos uso de una similitud de entrenamiento, que se usa para el vocabulario de sentimientos.

model.train(alldocs, total\_examples=model.corpus\_count, epochs=100, start\_alpha=0.01, end\_alpha=0.01)

model.save("model\_prueba2")

#Doc2Vec.load("model")

SentimentDocument = namedtuple('SentimentDocument', 'words tags title original\_number')

n=0

alldocs = [] # Will hold all docs in original order

regex = re.compile('[%s]' % re.escape(string.punctuation)) #to remove punctuation

for line\_no, line in enumerate(data):

if (type(line)==str):

if (len(line)>150):

if (detect(line) == 'es') :

line = regex.sub('', line)

tokens = gensim.utils.to\_unicode(line).lower().split()

words = tokens[0:]

tags = [n]

title = titles[line\_no]

alldocs.append(SentimentDocument(words, tags, title, line\_no))

n=n+1

l = []

for doc in alldocs:

l.append(len(doc.words))

print('Number of Documents : ', len(alldocs))

print('Mean length of documents : ', np.mean(l))

plt.figure (figsize=(20, 6))

plt.bar(range(0, len(l)), l)

plt.xlabel('Documents')

plt.ylabel('Length of the description')

index = 0

doc = alldocs[index]

print(doc, '\n')

print(data[doc.original\_number])

# PV-DM

model = Doc2Vec(dm=1, vector\_size=300,window=10,hs=0,min\_count=10,dbow\_words=1,sample=1e-5)

# build the vocabulary

model.build\_vocab(alldocs)

# Entrenar El Moelo Con El Vocabulario, Procedemos Hacer Las Distintas Comparaciones, y cuales palabras tienen relacion con ellas.

model.wv.most\_similar\_cosmul(positive = ["cáncer"])

model.wv.most\_similar\_cosmul(positive = ["mama"])

Después compararemos en que partes del corpus aparece una palabra determinada.

**2. Resultados Del Experimento**

# Closest document to the word "love"

tokens = "tumor"

new\_vector = model.infer\_vector(tokens.split() ,alpha=0.001 ,steps = 5)

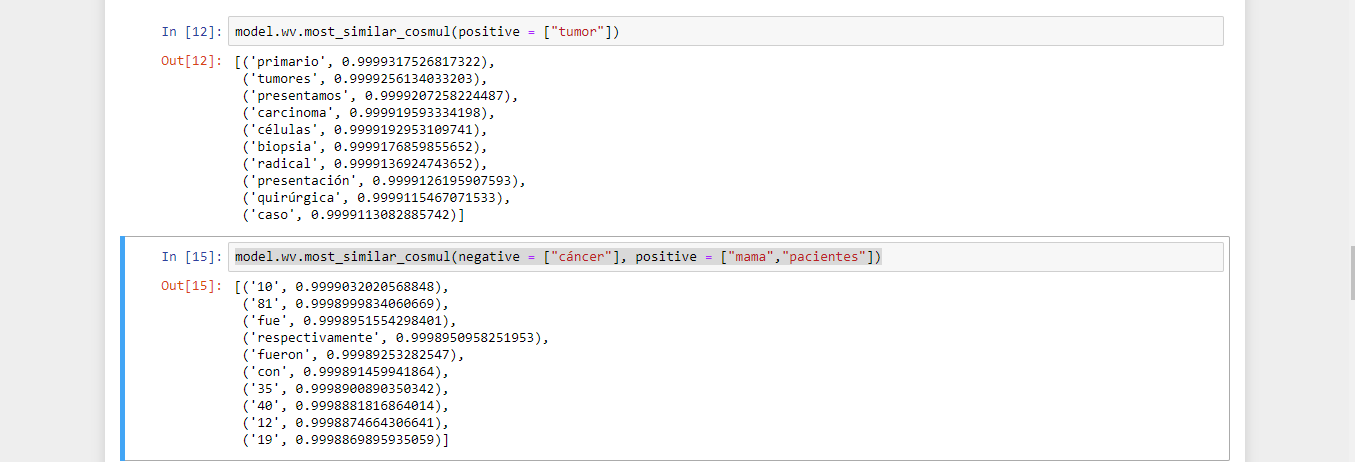
sims = model.docvecs.most\_similar([new\_vector], topn=model.docvecs.count) # get \*all\* similar documents

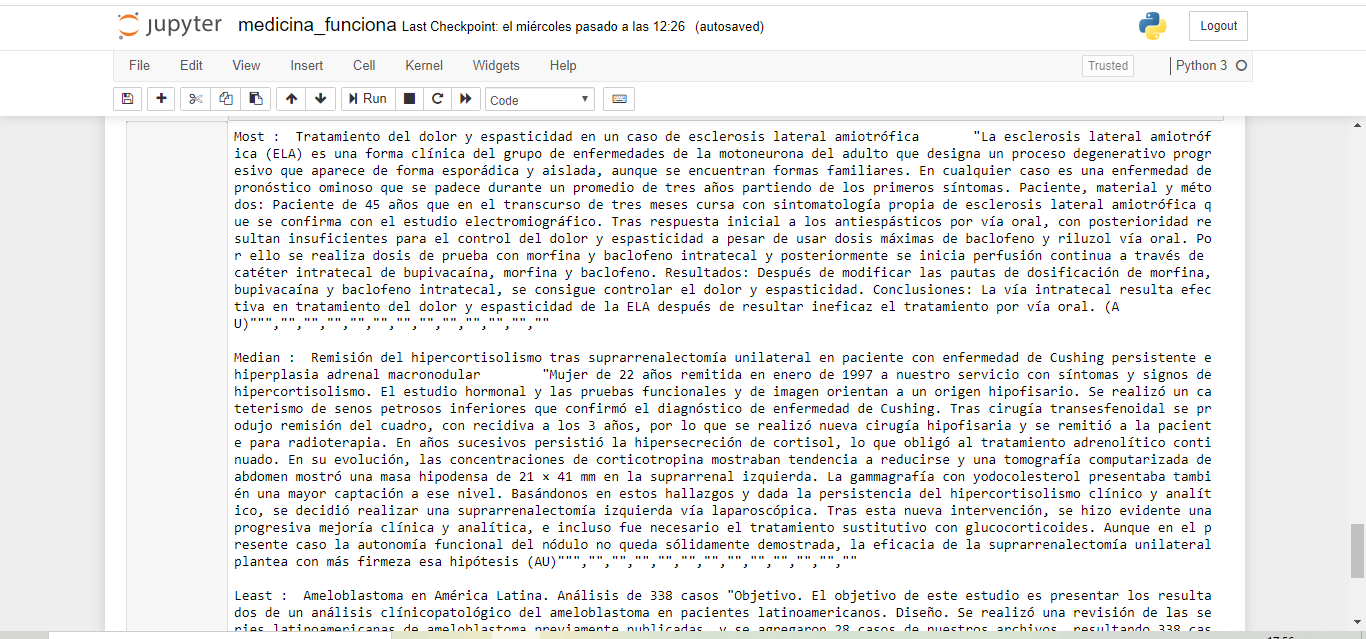
print("Most : " , data[alldocs[sims[0][0]].original\_number], "\n")

print("Median : " , data[alldocs[sims[170][0]].original\_number], "\n")

print("Least : " , data[alldocs[sims[-1][0]].original\_number])

Despues de los diferentes métodos que hemos usado en nuestro modelo procedemos a mostrar los resultado.





1. **CONCLUSIONES**

En general, los resultados encontrados nos confirman que el uso de la inteligencia artificial para el reconocimiento de texto es muy bueno

La dificultad que se ha presentado es sobre el courpus dado que debemos presentar un corpus con una estructura adecuada,

Alguna implementación podida hacer dado que no eran compatibles con la versión de gensim utilizada.

Por lo general hacido una gran experiencia de implementación y aprendizaje

**Referencias**

1. N. Manish, «medium,» 05 06 2019. [En línea]. Available: https://medium.com/towards-artificial-intelligence/an-intuitive-introduction-of-document-vector-doc2vec-42c6205ca5a2. [Último acceso: 02 11 2019].
2. <https://arxiv.org/abs/1607.05368>
3. <https://link.springer.com/chapter/10.1007/978-3-319-62428-0_9>
4. <https://dl.acm.org/doi/10.1145/2993148.2993203>
5. <https://link.springer.com/chapter/10.1007/978-3-319-24033-6_29>