The aim of this notebook is to make use of the word2vec model to find similar songs

WORD2VEC - Exploracion con un Corpus de canciones en Español usando tokenizer y stopwords para crear el corpus

Esto va a ser menos exacto que los anteriores porque hay menos palabras. Pero es interesante verlo.

In [34]:
```
import pandas as pd
import numpy as np
import gensim.models.word2vec as w2v
import multiprocessing
import os
import re
import pprint
import sklearn.manifold
import matplotlib.pyplot as plt
```

Cargamos las el quijote

In [46]:
```
f = open("data/quijote.txt", "r", encoding="utf8")
content =f.read()
```

Limpiamos los caractes raros y quitamos acentos

In [68]:
```
from unidecode import unidecode
clean_file = []
for k in content.split("\n"):
    k = unidecode(k) #quitamos los acentos
    clean_file.append(re.sub(r"[^a-zA-Z0-9]+", ' ', k)) #limpiamos las lineas de caracteres raros

for line in clean_file:  #quitamos las lineas vacias
    if line == '':
        clean_file.remove(line)
```

To train the word2vec model, we first need to build its vocabulary. To do that, I iterated over each song and added it to an array that can later be fed to the model.

Usamos tokenizer y stopwords para tener un mejor vocabulario

In [98]:
```
import nltk
from nltk.corpus import stopwords

text_corpus = []
for line in clean_file:
    tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+') #para dividir por words y quitar puntuacion
    lower_case = line.lower()
    tokens_sin_puntuacion = tokenizer.tokenize(lower_case)
    tokens = [i for i in tokens_sin_puntuacion if (len(i)>1) and i not in stopwords.words('spanish')] #quitamos stop

    text_corpus.append(tokens)
```

In [100]:

```
# Dimensionality of the resulting word vectors.
#more dimensions, more computationally expensive to train
#but also more accurate
#more dimensions = more generalized
num_features = 50
# Minimum word count threshold.
min_word_count = 1

# Number of threads to run in parallel.
#more workers, faster we train
num_workers = multiprocessing.cpu_count()

# Context window length.
context_size = 7


downsampling = 1e-1

# Seed for the RNG, to make the results reproducible.
#random number generator
#deterministic, good for debugging
seed = 1

songs2vec = w2v.Word2Vec(
    sg=1,
    seed=seed,
    workers=num_workers,
    size=num_features,
    min_count=min_word_count,
    window=context_size,
    sample=downsampling
)

songs2vec.build_vocab(text_corpus)
print (len(text_corpus))
```

```
32339
```

Entrenamiento:

In [101]:
```
import time
start_time = time.time()



songs2vec.train(text_corpus, total_examples=songs2vec.corpus_count, epochs=2)

if not os.path.exists("trained"):
    os.makedirs("trained")

songs2vec.save(os.path.join("trained", "songs2vectors.w2v"))

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 0.6159968376159668 seconds ---
```

In [102]:
```
songs2vec = w2v.Word2Vec.load(os.path.join("trained", "songs2vectors.w2v"))
```

Let's explore our model

Find similar words

In [103]:
```
songs2vec.wv.most_similar("quijote")
```

Out[103]:
```
[('sancho', 0.957434356212616),
 ('asi', 0.9564037322998047),
 ('cura', 0.9529216289520264),
 ('dila', 0.9478768110275269),
 ('pues', 0.9369694590568542),
 ('duque', 0.9366923570632935),
 ('voz', 0.9357013702392578),
 ('senora', 0.9349565505981445),
 ('duquesa', 0.9339219331741333),
 ('barbero', 0.9335750341415405)]
```

In [105]:
```
songs2vec.wv.most_similar("sancho")
```

Out[105]:
```
[('asi', 0.9668811559677124),
 ('quijote', 0.9574342966079712),
 ('cura', 0.9552509784698486),
 ('pues', 0.945313036441803),
 ('decir', 0.9427224397659302),
 ('senora', 0.9385666847229004),
 ('si', 0.9359698295593262),
 ('verdad', 0.9331722259521484),
 ('habia', 0.9328069686889648),
 ('bien', 0.9322810769081116)]
```

In [106]:
```
songs2vec.wv.most_similar("caballo")
```

Out[106]:
```
[('pie', 0.9992693066596985),
 ('palabra', 0.999201774597168),
 ('mesmo', 0.9991865754127502),
 ('primo', 0.9990447759628296),
 ('cuenta', 0.998988687992096),
 ('nunca', 0.9989795088768005),
 ('lugar', 0.9989739656448364),
 ('iba', 0.9988583326339722),
 ('camino', 0.9988481998443604),
 ('ama', 0.9988390207290649)]
```

In [107]:
```
songs2vec.wv.most_similar("camino")
```

Out[107]:
```
[('mismo', 0.9994622468948364),
 ('mucha', 0.9994367957115173),
 ('canonigo', 0.9994223713874817),
 ('primero', 0.9993078708648682),
 ('fin', 0.9992329478263855),
 ('entender', 0.9992038607597351),
 ('nunca', 0.9991317391395569),
 ('libro', 0.9990227222442627),
 ('tenia', 0.9989996552467346),
 ('contento', 0.9989832043647766)]
```

In [110]:
```
songs2vec.wv.most_similar("libro")
```

Out[110]:
```
[('dijese', 0.9994610548019409),
 ('aventura', 0.9994574189186096),
 ('mucha', 0.9993853569030762),
 ('entender', 0.9993103742599487),
 ('mismo', 0.9992714524269104),
 ('duena', 0.9992218613624573),
 ('desa', 0.9992120265960693),
 ('nombre', 0.9991418123245239),
 ('adonde', 0.9991415739059448),
 ('oyo', 0.9991327524185181)]
```

Words out of context

In [111]:
```
songs2vec.wv.doesnt_match("sancho quijote aventura".split())
```

```
C:\Users\jhern\Anaconda3\lib\site-packages\gensim\models\keyedvectors.py:877: FutureWarning: arrays to stack must b
e passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is depre
cated as of NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

Out[111]:
```
'aventura'
```

In [117]:
```
songs2vec.wv.doesnt_match("camino senora senor".split())
```

Out[117]:
```
'camino'
```

In [120]:
```
songs2vec.wv.doesnt_match("comida bebida árbol".split()) #se equivoca
```

Out[120]:
```
'bebida'
```

In [121]:
```
songs2vec.most_similar(positive=['sancho', 'panza'], negative=['quijote'])
#queen
```

```
C:\Users\jhern\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_si
milar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[121]:
```
[('vuesa', 0.9589300751686096),
 ('mio', 0.95602947473526),
 ('digo', 0.956020176410675),
 ('senor', 0.9536997079849243),
 ('decir', 0.9526917934417725),
 ('respondio', 0.9521059989929199),
 ('dijo', 0.9520894885063171),
 ('bien', 0.9513685703277588),
 ('dice', 0.9496554136276245),
 ('dios', 0.9490358829498291)]
```

Semantic distance between words

In [122]:
```
def nearest_similarity_cosmul(start1, end1, end2):
    similarities = songs2vec.wv.most_similar_cosmul(
        positive=[end2, start1],
        negative=[end1]
    )
    start2 = similarities[0][0]
    print("{0} es a {1}, lo que {2} es a {3}".format(start1, end1, start2, end2))
```

In [124]:
```
nearest_similarity_cosmul("sancho", "panza", "don") #bien
```

```
sancho es a panza, lo que quijote es a don
```

In [125]:
```
nearest_similarity_cosmul("libro", "historia", "camino") #mas o menos
```

```
libro es a historia, lo que llegando es a camino
```

In [127]:
```
nearest_similarity_cosmul("verso", "poesia", "cancion") #mas o menos
```

```
verso es a poesia, lo que ciencias es a cancion
```

In [0]:
```

```