
Instituto Politécnico Nacional

Escuela Superior de Cómputo


# Polaridad y sentimientos

Natural Language Processing


*Estudiante:*
Nicolás Sayago Abigail

*Profesora:*
Olga Kolesnicova


Junio 05, 2020

# 1 Resultados

Al principio había decidio usar las reviews de la categoría de música, sin embargo no me gusto mucho porque las personas que escribieron sus opiniones pusieron los títulos de las canciones (muchas en inglés). Así que decidí hacer un anális (experimentos) con 3 categorías.

## 1.1 Coches

Resultados de los n-grams:

```
 Most common 1-grams:
1 .  ('coche', 240)
2 .  ('si', 97)
3 .  ('mas', 50)
4 .  ('motor', 46)
5 .  ('km', 39)
6 .  ('solo', 36)
7 .  ('bien', 36)
8 .  ('dos', 29)
9 .  ('menos', 29)
10 .  ('opel', 29)
11 .  ('aunque', 29)
12 .  ('bastante', 29)
13 .  ('vez', 27)
14 .  ('puertas', 27)
15 .  ('modelo', 27)
16 .  ('gran', 26)
17 .  ('años', 25)
18 .  ('taller', 24)
19 .  ('tres', 24)
20 .  ('hace', 24)
```

```
 Most common 2-grams:
1 .  ('aire acondicionado', 10)
2 .  ('varias veces', 6)
3 .  ('dirección asistida', 6)
4 .  ('años km', 5)
5 .  ('dos años', 5)
6 .  ('gran coche', 5)
7 .  ('llevo opel', 5)
8 .  ('sale caro', 5)
9 .  ('coche mas', 5)
10 .  ('buen coche', 5)
11 .  ('versión blue', 5)
12 .  ('tres puertas', 5)
13 .  ('coche si', 4)
14 .  ('si quieres', 4)
15 .  ('tres años', 4)
16 .  ('año medio', 4)
17 .  ('menos mal', 4)
18 .  ('coche tenia', 4)
19 .  ('servicio oficial', 4)
20 .  ('coche barato', 4)
```

```
 Most common 3-grams:
1 .  ('regulable altura profundidad', 3)
2 .  ('menos tres años', 2)
3 .  ('dos años garantía', 2)
4 .  ('ampliación garantía años', 2)
5 .  ('llevo opel segun', 2)
6 .  ('llevo opel dicen', 2)
7 .  ('compré fiat brava', 2)
8 .  ('fiat brava td', 2)
9 .  ('brava td hace', 2)
10 .  ('td hace año', 2)
11 .  ('hace año medio', 2)
12 .  ('año medio gustaba', 2)
13 .  ('medio gustaba línea', 2)
14 .  ('gustaba línea parecía', 2)
15 .  ('línea parecía amplio', 2)
16 .  ('parecía amplio excelente', 2)
17 .  ('amplio excelente relación', 2)
18 .  ('excelente relación calidadprecio', 2)
19 .  ('relación calidadprecio menos', 2)
20 .  ('calidadprecio menos pensaba', 2)
```

Resultados de Polaridad: Elegí esas palabras porque considero que eran las palabras que describen que busca una persona al comprar un coche.

```
 Polarity of coches
('motor', 'n') 0.1702256097560976
('puerta', 'n') 0.21162387853692205
('acondicionado', 'n') 0.12817777777777778
('caro', 'a') 0.19512698412698412
('barato', 'a') 0.21940264550264552
('dirección', 'n') 0.25457142857142856
('garantía', 'n') 0.04788888888888889
```

## 1.2 Moviles

Resultados de los n-grams:

```
 Most common 1-grams:
1 .  ('movil', 110)
2 .  ('si', 106)
3 .  ('móvil', 94)
4 .  ('nokia', 80)
5 .  ('teléfono', 71)
6 .  ('aunque', 65)
7 .  ('mas', 56)
8 .  ('pantalla', 56)
9 .  ('bien', 55)
10 .  ('bastante', 51)
11 .  ('fotos', 45)
12 .  ('puede', 44)
13 .  ('ser', 43)
14 .  ('telefono', 41)
15 .  ('hace', 39)
16 .  ('batería', 38)
17 .  ('menos', 38)
18 .  ('así', 38)
19 .  ('bueno', 37)
20 .  ('x', 37)
```

```
 Most common 2-grams:
1 .  ('manos libres', 18)
2 .  ('x x', 9)
3 .  ('bastante bien', 8)
4 .  ('memoria interna', 8)
5 .  ('x mm', 8)
6 .  ('sony ericsson', 7)
7 .  ('va bien', 7)
8 .  ('tarjeta memoria', 7)
9 .  ('servicio tecnico', 6)
10 .  ('cámara fotos', 6)
11 .  ('puede ser', 6)
12 .  ('motorola v', 6)
13 .  ('x pixels', 6)
14 .  ('aún así', 6)
15 .  ('marca nokia', 5)
16 .  ('ericsson zi', 5)
17 .  ('tarjeta sim', 5)
18 .  ('radio fm', 5)
19 .  ('interna mb', 5)
20 .  ('gama baja', 5)
```

```
 Most common 3-grams:
1 .  ('x x mm', 7)
2 .  ('memoria interna mb', 5)
3 .  ('según fabricante dice', 4)
4 .  ('fabricante dice dura', 4)
5 .  ('sony ericsson zi', 3)
6 .  ('manos libres integrado', 3)
7 .  ('si compramos libre', 3)
8 .  ('solo durara mes', 2)
9 .  ('cada dos tres', 2)
10 .  ('hola amigos ciao', 2)
11 .  ('tarjeta sim memoria', 2)
12 .  ('sim memoria teléfono', 2)
13 .  ('memoria teléfono así', 2)
14 .  ('mm x x', 2)
15 .  ('posibilidad mandar mensajes', 2)
16 .  ('dura horas conversación', 2)
17 .  ('unas dimensiones x', 2)
18 .  ('dimensiones x x', 2)
19 .  ('bastante bien resolución', 2)
20 .  ('menuda mierda pense', 2)
```

Resultados de Polaridad: Elegí esas palabras porque además de ser las más repetida, considero que es lo que una persona busca al comprar un celular, sin embargo añadí la palabra **nokia** para ver si a las personas les gusta o no la marca.

```
 Polarity of moviles
('nokia', 'n') 0.2694026297085999
('pantalla', 'n') 0.3044697007275132
('batería', 'n') 0.2424207912457913
('memoria', 'n') 0.20141071428571422
('gama', 'n') 0.18767999999999999
('cámara', 'n') 0.26144677419354834
```

## 1.3 Musica

Resultados de los n-grams:

```
 Most common 1-grams:
1 .  ('disco', 175)
2 .  ('tema', 112)
3 .  ('canción', 83)
4 .  ('si', 82)
5 .  ('the', 82)
6 .  ('canciones', 76)
7 .  ('grupo', 63)
8 .  ('voz', 48)
9 .  ('temas', 46)
10 .  ('ser', 43)
11 .  ('rock', 43)
12 .  ('aunque', 43)
13 .  ('mas', 41)
14 .  ('you', 37)
15 .  ('estilo', 35)
16 .  ('bien', 35)
17 .  ('solo', 34)
18 .  ('banda', 34)
19 .  ('discos', 31)
20 .  ('decir', 31)
```

```
 Most common 2-grams:
1 .  ('bon jovi', 10)
2 .  ('do nt', 10)
3 .  ('marilyn manson', 9)
4 .  ('nuevo trabajo', 7)
5 .  ('on the', 7)
6 .  ('smashing pumpkins', 7)
7 .  ('billy corgan', 7)
8 .  ('primera vez', 6)
9 .  ('of the', 6)
10 .  ('daddy yankee', 6)
11 .  ('hip hop', 6)
12 .  ('bonus track', 5)
13 .  ('this left', 5)
14 .  ('hard rock', 5)
15 .  ('puedo decir', 5)
16 .  ('rock and', 5)
17 .  ('and roll', 5)
18 .  ('tokio hotel', 5)
19 .  ('love is', 5)
20 .  ('nt jump', 5)
```

```
 Most common 3-grams:
1 .  ('rock and roll', 5)
2 .  ('do nt jump', 5)
3 .  ('hoy dicho hola', 4)
4 .  ('dicho hola primera', 4)
5 .  ('hola primera vez', 4)
6 .  ('this left feels', 4)
7 .  ('left feels right', 4)
8 .  ('love is dead', 4)
9 .  ('the golden age', 4)
10 .  ('golden age of', 4)
11 .  ('age of grotesque', 4)
12 .  ('born to be', 4)
13 .  ('to be my', 4)
14 .  ('be my baby', 4)
15 .  ('keep the faith', 3)
16 .  ('be there for', 3)
17 .  ('there for you', 3)
18 .  ('push up on', 3)
19 .  ('do nt stop', 3)
20 .  ('nt stop the', 3)
```

Resultados de Polaridad: Como mencione al principio, el principal problema es que había muchas palabras en inglés, sin embargo yo considero que obtener la polaridad de la palabra **rock** nos ayuda a ver qué opinan las personas de ese género musical.

```
 Polarity of musica
('estilo', 'n') 0.24687255291005286
('disco', 'n') 0.2238827947845803
('tema', 'n') 0.20620468864468866
('canción', 'n') 0.1944913419913419
('grupo', 'n') 0.18421629870129866
('voz', 'n') 0.15637747474747477
('rock', 'n') 0.15994572310405647
('guitarra', 'n') 0.2172897406116918
('bateria', 'n') 0.3482857142857143
```

### ✓Código fuente para obtener n-grams

```
import nltk
from operator import itemgetter
import re
import math
from bs4 import BeautifulSoup
from pickle import dump, load
from nltk.corpus import cess_esp
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import stopwords
import glob

################################################
#                          REVIEWS
################################################
def getReviews(path, code):
        names = [f for f in glob.glob(path + "**/*.txt", recursive=True)]
        text = ""
        for name in names:
                f = open(name, encoding = code)
                rev = f.read()
                f.close()
                text = text + rev + " "
                print(name)
        text = text.lower()
        return text

#Parameters: Text
#Return: List of original tokens
def getTokens(text):
        tokens = nltk.word_tokenize(text)
        return tokens

#Parameters: List of tuples of tokens
#Return: List of clean tokens and Tags
def getCleanTokens(tokens):
        clean = []
        for token in tokens:
                t = []
                for char in token:
                        if re.match(r'[a-záéíóúñüA-ZÁÉÍÓÚÜÑ]', char):
                                t.append(char)
                letterToken = ''.join(t)
```

```
                if letterToken != '':
                        clean.append(letterToken)
        return clean
#Parameters: List of clean tokens, language of stopwords
#Return: List of tokens without stopwords
def removeStopwords(tokens, language):
        sw = stopwords.words(language)

        cleanTokens = []
        for tok in tokens:
                if tok not in sw:
                        cleanTokens.append(tok)
        return cleanTokens

##############################################
#                               LEMMAS
##############################################
def getWords(fpath, code):
        f = open(fpath, encoding = code) #Cod: utf-8, latin-1
        text = f.read()
        f.close()

        words = re.sub(" ", " ",  text).split()
        return words

# Return: Dictionary
def createDicLemmas(tokensLemmas):
        lemmas = {}
        j = 0
        for i in range(0, len(tokensLemmas)- 2, 3):
                word = tokensLemmas[i]
                tag = tokensLemmas[i+1]
                val = tokensLemmas[i+2]
                l = word
                lemmas[l] = val
                j = j+1
        return lemmas

def printDictionary(dic, n):
        i = 0
        for j in dic:
                print(j, dic[j])
                i = i + 1
```

```
                if i > n:
                        break
##############################################
#                              UNIGRAMS
##############################################
def flatten_corpus(corpus):
        return ' '.join([document.strip() for document in corpus])

def compute_ngrams(sequence, n):
        return zip(*[sequence[index:] for index in range(n)])

def get_top_ngrams(tokens, ngram_val=1, limit=20):
        # tokens = nltk.word_tokenize(corpus)
        ngrams = compute_ngrams(tokens, ngram_val)
        ngrams_freq_dist = nltk.FreqDist(ngrams)
        sorted_ngrams_fd = sorted(ngrams_freq_dist.items(),
        ↪  key=itemgetter(1), reverse=True)
        sorted_ngrams = sorted_ngrams_fd[0:limit]
        sorted_ngrams = [(' '.join(text), freq) for text, freq in
        ↪  sorted_ngrams]
        return sorted_ngrams

fpathLemmas =
↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Normalize/generate
fpathName = 'generateClean.txt'
code = 'ISO-8859-1'
textLemmas = getWords(fpathLemmas, code)
print(textLemmas[:20])

# Get dictionary of tuples of
lemmas = {}
lemmas = createDicLemmas(textLemmas)
print("dictionary of lemmas:")
lemmas["abaláncenosla"] # Check the first
lemmas["zutano"]                    # Check the last
lemmas["acercarnos"]
printDictionary(lemmas, 10)


kind = 'moviles'
path =
↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Practice/26/corpus
reviews = getReviews(path + kind, code)
tokens = getTokens(reviews) #  24565
```

```
print(len(tokens))

cleanTokens = getCleanTokens(tokens) # 20951
print(len(cleanTokens))

tok = removeStopwords(cleanTokens, 'spanish') # 11463
print(len(tok))

n = 1
aux = get_top_ngrams(tok, n, 20)
#### UNIGRAMS
print('\n Most common ' + str(n) +'-grams:')
for i in range(len(aux)):
        print(i+1, ". ", aux[i])

n = 2
aux = get_top_ngrams(tok, n, 20)
#### UNIGRAMS
print('\n Most common ' + str(n) +'-grams:')
for i in range(len(aux)):
        print(i+1, ". ", aux[i])

n = 3
aux = get_top_ngrams(tok, n, 20)
#### UNIGRAMS
print('\n Most common ' + str(n) +'-grams:')
for i in range(len(aux)):
        print(i+1, ". ", aux[i])
```

### ✓Código fuente para obtener polaridad

```
import nltk
import re
import math
from bs4 import BeautifulSoup
from pickle import dump, load
from nltk.corpus import cess_esp
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import stopwords

##############################################
#                          GETTING DICTIONARY
##############################################
```

```
def getElement(s):
        word = ''
        j = s.find('">')
        j = j + 3
        while(s[j] != '<'):
                word += s[j]
                j = j + 1
        word = word[:len(word)-1]
        return word

# Return a string
def getPOS(s):
        j = s.find('pos=')
        j = j + 5
        tag = ''
        while(s[j] != '"'):
                tag = tag + s[j]
                j = j + 1
        return tag

# Return a number that represent polarity
def getPolarity(s):
        j = s.find('pol=')
        j = j + 5
        pol = ''
        while(s[j] != '"'):
                pol = pol + s[j]
                j = j + 1
        pol = float(pol)
        return pol

# Return dictionary of pairs with polarity
def getDictionary(corpusRoot, code):
        dictionary = {}
        f = open(corpusRoot, encoding = code)
        text = f.readlines()
        f.close()

        for i in range(0, len(text)):
                word = getElement(text[i])
                POS = getPOS(text[i])
                polarity = getPolarity(text[i])
                pair = (word, POS)
                dictionary[pair] = polarity
```

```
        return dictionary

#################################################################
#                                          NORMALIZE TEXT
#################################################################
def getReviews(path):
        names = [f for f in glob.glob(path + "**/*.txt", recursive=True)]
        text = ""
        for name in names:
                f = open(name, encoding = 'ISO-8859-1')
                rev = f.read()
                f.close()
                text = text + rev + " "
        return text

#Parameters: Text
#Return: List of original tokens
def getTokens(text):
        tokens = nltk.word_tokenize(text)
        return tokens

#Parameters: List of normalize tokens
#Return: Set, vocabulary
def getVocabulary(tokens):
        s = set()
        for l in tokens:
                for element in l:
                        s.add(element)
        vocabulary = sorted(s)
        return vocabulary

#Parameters: List of tuples of tokens
#Return: List of clean tokens and Tags
def getCleanTokensTags(tokens):
        clean = []
        for token in tokens:
                t = []
                for char in token[0]:
                        if re.match(r'[a-záéíóúñüA-ZÁÉÍÓÚÜÑ]', char):
                                t.append(char)
                letterToken = ''.join(t)

                if len(token[1]) > 0:
                        tag = token[1]
```

```
                        tag = tag[0].lower()

                if letterToken != '':
                        letterToken = letterToken.lower()
                        l = (letterToken, tag)
                        clean.append(l)
        return clean

#Parameters: List of clean tokens, language of stopwords
#Return: List of tokens without stopwords
def removeStopwords(tokens, language):
        sw = stopwords.words(language)

        cleanTokens = []
        for tok in tokens:
                l = ()
                if tok[0] not in sw:
                        l = (tok[0], tok[1])
                        cleanTokens.append(l)
        return cleanTokens

def getWords(fpath, code):
        f = open(fpath, encoding = code) #Cod: utf-8, latin-1
        text = f.read()
        f.close()

        words = re.sub(" ", " ",  text).split()
        return words
#################################################################
#                                                    TAGGING
#################################################################
def make_and_save_combined_tagger(fname):
    default_tagger = nltk.DefaultTagger('v')
    patterns = [ (r'.*o$', 'n'),    # noun masculine singular
                        (r'.*os$', 'n'),  # noun masculine plural
                  (r'.*a$', 'n'),   # noun feminine singular
                  (r'.*as$', 'n')   # noun feminine singular
                ]
    regexp_tagger = nltk.RegexpTagger(patterns, backoff=default_tagger)
    cess_tagged_sents = cess_esp.tagged_sents()
    combined_tagger = nltk.UnigramTagger(cess_tagged_sents,
     ↪  backoff=regexp_tagger)

    output = open(fname, 'wb')
```

```
    dump(combined_tagger, output, -1)
    output.close()

def tag(fname, text):
    input = open(fname, 'rb')
    default_tagger = load(input)
    input.close()

    s_tagged = default_tagger.tag(text)
    return s_tagged

#####################################################################
#                                                          LEMMAS
#####################################################################
def getWord(word):
        cleanWord = ''
        for char in word:
                if char != '#':
                        cleanWord += char
        return cleanWord

def getTag(word):
        c = 'v'
        if len(word) > 0:
                c = word[0]
        return c.lower()

# Return: Dictionary
def createDicLemmas(tokensLemmas):
        lemmas = {}
        j = 0
        for i in range(0, len(tokensLemmas)- 2, 3):
                word = tokensLemmas[i]
                tag = tokensLemmas[i+1]
                val = tokensLemmas[i+2]
                l = (word, tag[0].lower())
                lemmas[l] = val
                j = j+1
        return lemmas

def lemmatizeText(tokens, lemmas):
        text = []
        for token in tokens:
                lemma = token[0]
```

```
                if token in lemmas:
                        lemma = lemmas[token]
                aux = (lemma, token[1])
                text.append(aux)
        return text

#############################################################
#                                                 CREATE FILE
#############################################################
def printContext(context):
        for i in range(0, len(context)):
                aux = ''
                for j in range(0, len(context[i])):
                        aux += context[i][j] + " "
                print(aux)

def printDictionary(dic, n):
        i = 0
        for j in dic:
                print(j, dic[j])
                i = i + 1
                if i > n:
                        break

def makePKL(fname, aux):
        output = open(fname, 'wb')
        dump(aux, output, -1)
        output.close()

def getPKL(fname):
        input = open(fname, 'rb')
        aux = load(input)
        input.close()
        return aux

def get_sentences(fname):
    text_string = getReviews(fname)
    sent_tokenizer=nltk.data.load('nltk:tokenizers/punkt/english.pickle')
    sentences = sent_tokenizer.tokenize(text_string)
    return sentences

############################
#                      POLARITY
############################
```

```
def getPolaritySen(dictionary, sentence):
        pol = 0
        cont = 0
        for word in sentence:
                if word in dictionary:
                        pol = pol + dictionary[word]
                        cont = cont + 1
        if cont:
                pol = pol / cont
        return pol

def getDictionaryPolarity(tokens, dictionary):
        dic = {}
        for i in range(0, len(tokens)):
                dic[i] = getPolaritySen(dictionary, tokens[i])
        return dic

def findWords(words, tokens):
        dic_words = {}
        for word in words:
                dic_words[word] = set()

        for i in range(0, len(tokens)):
                for j in range(0, len(tokens[i])):
                        if tokens[i][j] in dic_words:
                                dic_words[tokens[i][j]].add(i)
        return dic_words

def calculateTotalPolarity(list_words, words, polarity):
        ans = {}
        for word in list_words:
                total = 0
                cont = 0
                for i in words[word]:
                        total = total + polarity[i]
                        cont = cont + 1
                if cont:
                        total = total / cont
                ans[word] = total
        return ans

################################################
################################################
################################################
```

```
# Get Tokens by Generate.txt to create dictionary of lemmas
fpathLemmas =
 ↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Normalize/generate
code = 'ISO-8859-1'
textLemmas = getWords(fpathLemmas, code)
lemmas = {}
lemmas = createDicLemmas(textLemmas)
# Get dictionary of polarity
fpath =
 ↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Practice/25/ML-Sen
dictionary = getDictionary(fpath, 'utf-8') # Dictionary of tuples with
 ↪  polarity

# Read file of corpus and get Sentences
kind = 'moviles'
words = [('nokia', 'n'), ('pantalla', 'n'), ('batería', 'n'), ('memoria',
 ↪  'n'), ('gama', 'n'), ('cámara', 'n')]

path =
 ↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Practice/26/corpus
sentences = get_sentences(path+kind) # 755
# Tagging
fcombinedTagger =
 ↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/pkl/combined_tagge
#make_and_save_combined_tagger(fcombinedTagger)
sentencesTag = []
for i in range(0, len(sentences)):
        aux = getTokens(sentences[i])
        auxTag = tag(fcombinedTagger, aux)
        sentencesTag.append(auxTag)

sentencesCleanTokens = []
for i in range(0, len(sentencesTag)):
        aux = getCleanTokensTags(sentencesTag[i])
        sentencesCleanTokens.append(aux)

# Remove Stopwords
language = 'spanish'
sentencesClean = []
for i in range(0, len(sentencesCleanTokens)):
        aux = removeStopwords(sentencesCleanTokens[i], language)
        sentencesClean.append(aux)
```

```
# Lemmatize text
tokens = []
for i in range(0, len(sentencesClean)):
        aux = lemmatizeText(sentencesClean[i], lemmas)
        tokens.append(aux)

sentences_polarity = getDictionaryPolarity(tokens, dictionary)
polarity_words = findWords(words, tokens)
result = calculateTotalPolarity(words, polarity_words, sentences_polarity)

print("\n\n Polarity of " + kind)
printDictionary(result, 10)
```