Organización de Datos 75.06/95.58
Trabajo Práctico Nº 2

DatUs

Apellido y Nombre	Padrón	Correo electrónico
Boada Ignacio	95212	ignacio.boada@outlook.com
Goñi Mauro	87646	maurogoni@gmail.com
Perez Cristian	95536	cfperez@fi.uba.ar
Perez Machado Axel	101127	axelmpm@gmail.com

`Github:`
`https://github.com/axelmpm/TPs_DATOS_1C_2020`

Indice

1 Introducción **5**
1.1 Resumen . . . 5
1.2 Herramientas . . . 5

2 Preprocesamiento y Feature Engineering **6**
2.1 Resumen . . . 6
2.2 Overview . . . 6
2.3 Primer conjunto de features . . . 7
2.3.1 Location . . . 9
2.4 Segundo conjunto de features . . . 10
2.4.1 Hipotesis . . . 11
2.4.2 Hashtags . . . 11
2.4.3 Menciones . . . 15
2.4.4 Links . . . 18
2.4.5 HTML . . . 22
2.4.6 Emojis . . . 22
2.4.7 Mails . . . 23
2.4.8 Fechas y Horas . . . 24
2.4.9 Simbolos raros . . . 27
2.4.10 Patrones numericos varios . . . 31
2.4.11 Dinero . . . 31
2.4.12 Numeros . . . 32
2.4.13 Signos de Puntuacion que no son ? ni ! (esos los tratamos luego) . . . 35
2.4.14 Signos gramaticales no tan usados . . . 35
2.4.15 Signos de ? y ! . . . 37
2.4.16 Inpeccion . . . 38
2.4.17 Tratamiento de Lugares, Nombres y Acronimos . . . 40
2.4.18 Resultados . . . 45
2.4.19 Evaluando que tan bien se limpio la data . . . 46

3 KNN **47**
3.1 Resumen . . . 47
3.2 Motivacion . . . 47
3.3 Feature Engineering . . . 47
3.4 Desarrollo . . . 47
3.5 Resultados . . . 47

4 SVM **49**
4.1 Resumen . . . 49
4.2 Motivacion . . . 49
4.3 Preprocesamiento . . . 49

4.4 Feature Engineering . . . 50
4.5 Desarrollo . . . 50
4.6 Resultados . . . 50

5 NaiveBayes **51**
5.1 Resumen . . . 51
5.2 Motivacion . . . 51
5.3 Preprocesamiento . . . 51
5.4 Feature Engineering . . . 51
5.5 Desarrollo . . . 51
5.6 Resultados . . . 51

6 Pipeline **52**
6.1 Resumen . . . 52
6.2 Motivacion . . . 52
6.3 Feature Engineering . . . 52
6.4 Desarrollo . . . 52
6.5 Resultados . . . 53

7 Logistic Regression **54**
7.1 Resumen . . . 54
7.2 Motivacion . . . 54
7.3 Preprocesamiento . . . 54
7.4 Feature Engineering . . . 54
7.5 Desarrollo . . . 54
7.6 Resultados . . . 55

8 Context **56**
8.1 Resumen . . . 56
8.2 Motivacion y Preprocesamiento . . . 56
8.3 Feature Engineering . . . 56
8.4 Desarrollo . . . 57
8.4.1 Supuestos . . . 57
8.4.2 Desarrollo continuacion . . . 58
8.4.3 Tuneo de hiperparametros . . . 59
8.5 Resultados . . . 59

9 Random Forest **60**
9.1 Resumen . . . 60
9.2 Motivacion . . . 60
9.3 Preprocesamiento . . . 61
9.4 Feature Engineering . . . 61
9.5 Desarrollo . . . 61

9.6 Resultados 64

10 Decision Tree **65**
10.1 Resumen 65
10.2 Motivacion 65
10.3 Preprocesamiento 65
10.4 Feature Engineering 65
10.5 Desarrollo 65
10.6 Resultados 68

11 CatBoost **69**
11.1 Resumen 69
11.2 Motivacion 69
11.3 Feature Engineering 69
11.4 Desarrollo 69
11.5 Resultados 69

12 XGBoost **70**
12.1 Resumen 70
12.2 Motivacion 70
12.3 Feature Engineering 70
12.4 Desarrollo 70
12.5 Resultados 71

13 T5 **72**
13.1 Resumen y Motivacion 72
13.2 Preprocesamiento 72
13.3 Feature Engineering 72
13.4 Desarrollo 73
13.5 Resultados 73
13.6 Conclusiones 75

14 BERT **76**
14.1 Resumen 76
14.2 Motivacion 76
14.3 Preprocesamiento 76
14.3.1 Encoding 76
14.3.2 Balanceo y Desbalanceo de los datasets 78
14.3.3 Estratificacion por keywords 79
14.4 Desarrollo 82
14.4.1 Red neuronal "tradicional" 82
14.4.2 Fiteo de BERT 83
14.5 Resultados 84

14.6 Conclusion 85

15 Cosas que no llegamos a probar **86**
15.1 HDBScan 86
15.2 SVD 86
15.3 Boostings de XGboosts 86
15.4 Ensambles 86

16 Conclusión **87**
16.1 Conclusiones del TP 87

17 Bibliografía y Referencias **88**

1 Introducción

1.1 Resumen

Todo nuestros esfuerzo podria dividirse en cuatro grandes categorias:

- limpieza de sets de datos y feature engineering
- Probar algoritmos varios para ver como andan
- Probar con mayor atencion XGBoosts
- Probar con mayor atencion redes neuronales

Se tomaron multiples recaudos sobre los sets de datos a usar en cada caso especial del algoritmo usado.

En todos los casos se asume la hipotesis basica de que el set de train y de test comparten la misma naturaleza y que cualquier cosa que se aprenda de train podra generalizarse sobre test.

1.2 Herramientas

Se trabajo sobre notebooks en Python sobre jupyter, google colabs y kernels de kaggle.

El informe y todo el repo puede encontrarse en el link de mas abajo y este informe fue hecho con Overleaf en latex.

El código se encuentra disponible en:
`https://github.com/axelmpm/TPs_DATOS_1C_2020`

2 Preprocesamiento y Feature Engineering

2.1 Resumen

En esta seccion se expandera sobre todo el trabajo de preprocesamiento y extraccion de features que nos parecieron que podrian ser de significancia para los modelos a usar.

2.2 Overview

Dado que el TP se desarrollo en forma iterativa/incremental, este proceso de preprocesamiento y extraccion de features de realizo en dos pasadas (en realidad fueron un poco mas y de forma mas difusa).

Esto resulto en multiples versiones y archivos de los datasets a usar:

- FE 1 test.csv
- FE 1 train.csv
- to predict feature merged.csv
- train feature merged.csv
- test feature merged.csv
- train features 1
- test features 1
- T5 fe 1.csv
- sentiment.csv
- clean train.csv
- clean test.csv

A estos se les dieron diferentes grados de atencion dependiendo del algoritmo a usar (datos estructurados o no estructurados) y dependiendo de los resultados que fueron dando los algoritmos con los mismos. Otras versiones de estos archivos existieron pero fueron pisadas por versiones mejoradas.

2.3 Primer conjunto de features

Este conjunto de features se trabajo para extraer propiedades del texto mas que sobre la limpieza del mismo. Estos features tenian una orientacion hacia los algoritmos que toman datos esctructurados para funcionar, como los arboles.

Las propiedades que componian tales sets eran:

- keyword
- location
- text
- complete text: Concatenacion de keyword, location y text
- target
- continent: Continente del que venia le tweet
- country: Pais del que venia le tweet
- city: Ciudad del que venia le tweet
- hasNum: Si poseia numeros
- hasLink: Si poseia links
- url count: cantidad de links
- hasUser: Si hacia una mencion a algun usario
- mention count: cantidad de menciones
- hasHashtag: Si tenia hashtag
- hashtag count: cantidad de hashtags
- exclamacion: Cuantos signos de exclamacion usaba
- interrogacion. Cuantos signos de interrogacion usaba
- mayusculas: Cuantos mayusculas figuraban usaba
- longitud: Longitud del tweet en caracteres
- word count: cantidad de palabras
- unique word count: cantidad de palabras unicas

- mean word length: tamaño promedio en palabras
- negativity: Valor obtenido gracias a el output de la clase Context
- sigmoidic negativity: Negatividad mapeada a un intervalo de [0;1] por la clase Context
- neg to label: sigmoidic negativity pero mapeada a 0,1 a travez de un escalon unitario.
- sentiment: Un valor que asignaba una red neuronal transormer de la libreria Hugging Face para analisis de sentimiento
- 2-gramas y 3-gramas: Se veia con one hot encoding y posteriormente con binary encoding que 2-grama o 3-grama y en que medida poseia el tweet en cuestion
- posesion o no de ciertas keywords: Se probaron varias keywords que se pensaron relevantes
- foco en ciertas localizaciones: se hizo foco en ciertas localizaciones que nos parecieron relevantes.

Gran mayoria de los features aca presentados (como hasLink, hasUser, hasHashtag, hasNum, excalamacion, interrogacion, etc) son explicados en mayor profundidad en las siguientes secciones, otros como (complete text, mayusculas, longitud y las corespondientes a la negatividad, etc) se consideran obvias o ya explicadas. La siguiente seccion tocara el tema de las features relacionadas a la localizacion.

2.3.1 Location

Notamos que mucha gente pone de todo en el campo location, pero que habia cierta informacion util y latente. Por ejemplo, sabiendo que los tweets son en su grandisima mayoria en ingles, se podria pensar que continentes como Norte America o Europa podrian tener mas relevancia en estos tweets. El tema es que la gente que por ejemplo pone como location 'Andorra' o 'Luxemburgo' al no ser paises muy conocidos quizas los algoritmos no terminan de entender que su localizacion es de importancia (Europa en el ejemplo).

Motivados por este problema se nos ocurrio desarrollar una serie de funciones que, en combinacion con la libreria geonamescache, extraen del campo location el continente, pais y ciudad.

A modo clarificador si se tuviese por ejemplo 'Buenos Aires' como campo location, nosotros extraeriamos 'America Sur','Argentina','Buenos Aires'.

Es de esta forma que conseguimos mas features latentes en la informacion existente en el set de datos. Cabe decir que nuestra implementacion y sus resultados asociados dejan bastante que desear pero no nos queriamos perder la oportunidad de intentarlo.

2.4 Segundo conjunto de features

En esta seccion se expande sobre el analisis de features y distribuciones latentes en el set de datos a trabajar.

Esta segunda ronda de procesamiento se realiza mas avanzada en la cursada con la mentalidad de facilitarle la tarea de aprendizaje a las redes neuronales que estamos entrenando en ese momento.

El razonamiento es el siguiente. Las redes neuronales (en particular los transformers que usamos que ya estan preeentrenados) requieren que los datos esten codificados en forma numerica. Esta codificacion se realiza a travez de tokenizadores fuertemente entrenados y especializados. Los mismos realizan lo que se conoce como un Embedding sobre los vectores que representan tokens.

Dado que estos embeddings ya estan aprendidos nuestra intencion es limpiar lo mas que se pueda todo el corupus de texto para que las redes puedan entender el texto y codificarlo sin tener que realizar un embedding nuevo sobre palabras que considera nueva.

Cosas que por ejemplo queriamos evitar eran el caso de links. Links hay muchisimos y son todos distintos. El tokenizador de las redes pre entrenadas interpretarian estos links como palabras nuevas a aprender con un significado que decifrar. Esto es tentador porque en principio es cierto que los links tengan alguna importancia semantica en el contexto de un tweet, pero por otro lado no queremos que la red se tome el trabado de aprender cada uno de los links, mas teniendo en cuenta que la red vera a cada link como una palabra nueva y distinta. Es por esta razon que cosas como link, emoticones, etc se mapearon a tags.

Como dato cursioso queriamos decir que esta seccion fue comoda de hacer gracias a que nos dedicamos a aprender sobre expresiones regulares en python. El uso de las mismas nos allano enormemente el camino para el parseo e identificacion de anomalias en el corpus.

2.4.1 Hipotesis

Todo lo que se va a hacer ahora es sobre el set de train SOLAMENTE porque sino estariamos haciendo las cosas mal.

Lo que limpiamos debe basarse solo en lo que va a ver el modelo y no en consideraciones del set de test o de lo contrario no vamos a poder medir bien el overfit.

De todas formas todo lo que se hace aca (analizando el set de train) asume que puede generalizarse al set de test. Es decir, asumimos que las distribuciones encontradas sobre el set de train son las mismas sobre el set de test.

2.4.2 Hashtags

```
hashtag_pattern = re.compile('#\S*')
hashtags = pd.Series(re.findall(hashtag_pattern,corpus))
hashtags
```

```
]: 0        #earthquake
1         #wildfires
2            #Alaska
3         #wildfires
4         #RockyFire
             ...
3436         #iPhone
3437       #breaking
3438             #LA
3439      #WorldNews
3440     #stormchase
Length: 3441, dtype: object
```

Figure 1

```
print("Canitdad de hashtags en el corpus: ",len(hashtags))
print("Canitdad de hashtags unicos en el corpus: ",len(set(hashtags)))
counts = hashtags.value_counts()
```

```
Canitdad de hashtags en el corpus:  3441
Canitdad de hashtags unicos en el corpus:  2179
```

Figure 2

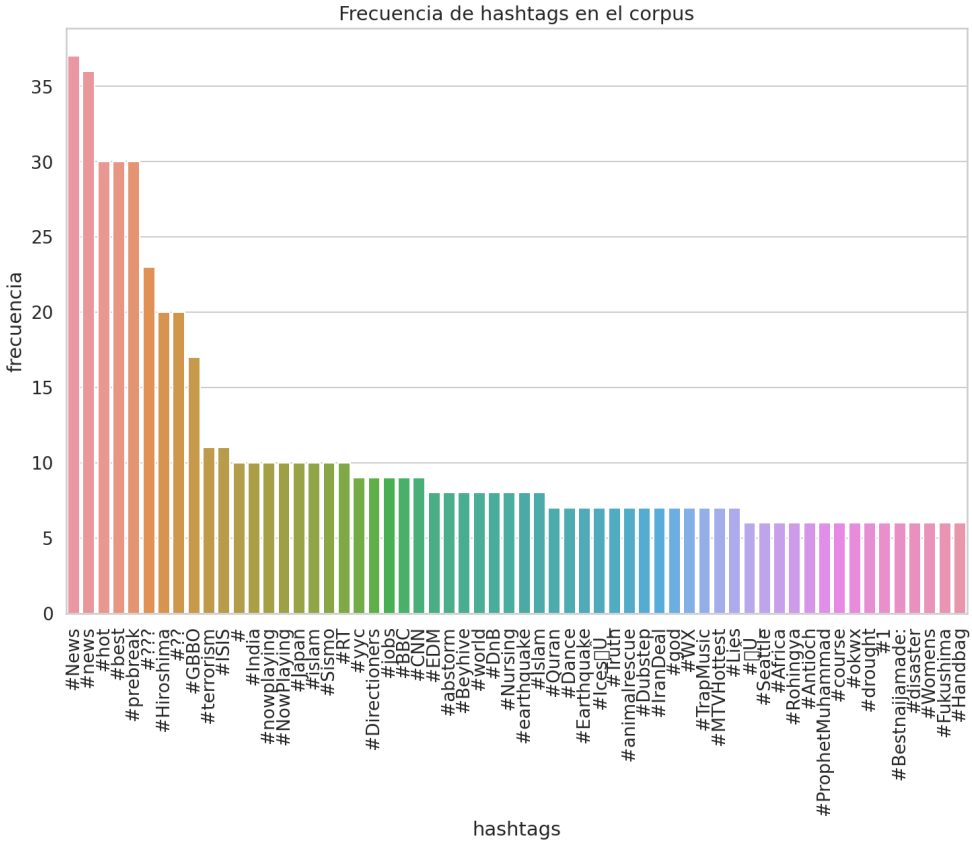

Figure 3

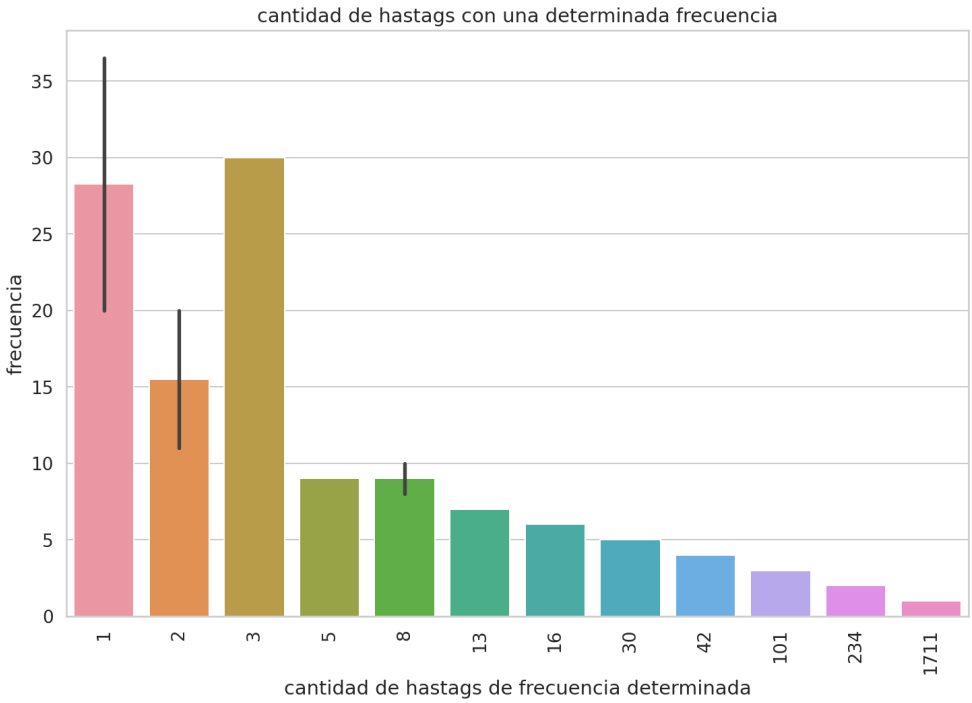

Figure 4

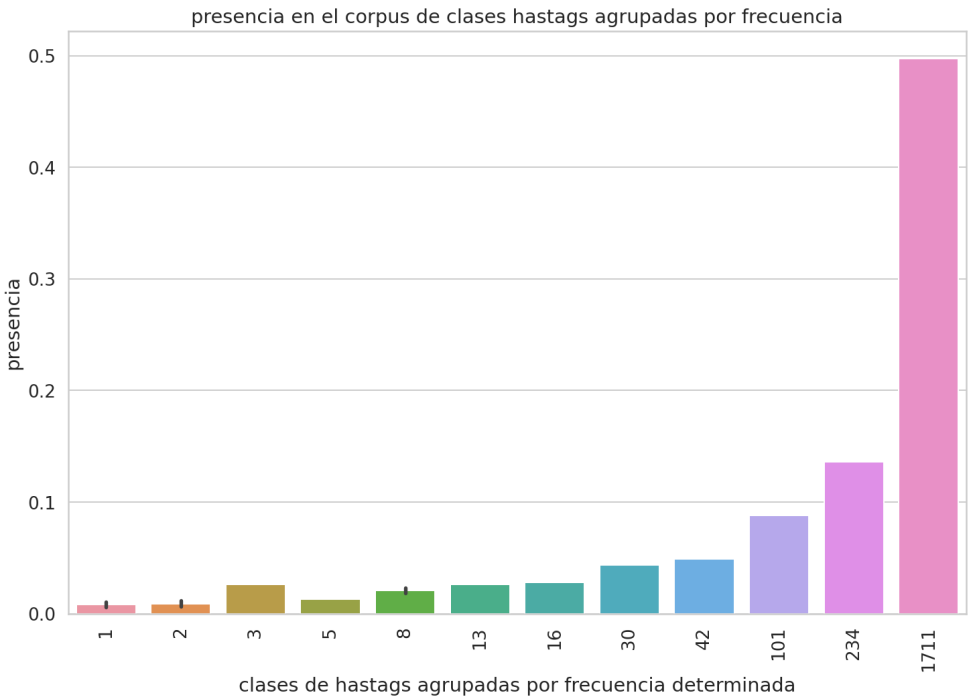

Figure 5

Vemos que algunos hashtags se usan bastante y quizas tengan importancia para predecir el label por lo que queremos quedarnoslos y hacer que nuestro modelo los aprenda.

Sin embargo hemos visto que son muchisimos y no tenemos tantos tweets como para que el modelo pudiese aprenderlos todos. Ademas, segun vemos en el grafico anterior, las clases de tweets agrupadas por probabilidad de ocurrencia siguen una power law. Esto quiere decir que la mayoria tienen poca precencia u ocurrencia, mientras que unos pocos son los que mas se usan.

Vamos a buscar a partir de que threshold de agrupamiento de tweets el porcentage de los desechados es menor a 50 porciento.

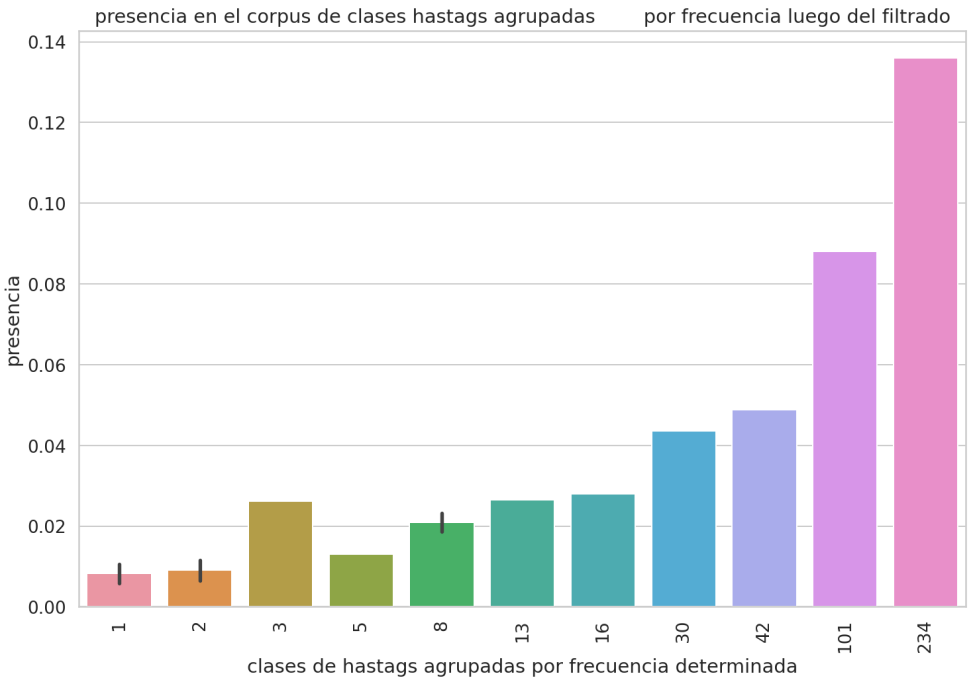

Figure 6

```
Cantidad de hashtags que quedan:  1730.0
Cantidad de hashtags que que habia:  3441
Porcentaje filtrado por baja ocurrencia: 49.7%
```

Figure 7

Ahora que seleccionamos que hashtags filtrar vamos a proceder a crear la funcion que filtra del dataset aquellos hashtags que seleccionamos para filtrar.

En nuestro caso filtrar simplemente significa asignarle el tag HASHTAG. De este modo el 49.7 porciento de las ocurrencias de hashtags tendran ese tag y el modelo que usemos sera mas capaz de aprender de estos.

2.4.3 Menciones

La operatoria sera analoga a la de los hashtags asi que de aca en mas se procedera a hacerlo de forma menos explicita.

```
mentions_pattern = re.compile('@\S*(?!\.com)')
mentions = pd.Series(re.findall(mentions_pattern,corpus))
```

Figure 8

```
print("Canitdad de menciones en el corpus: ",len(mentions))
print("Canitdad de menciones unicos en el corpus: ",len(set(mentions)))
counts = mentions.value_counts()
counts
```

```
Canitdad de menciones en el corpus:  2784
Canitdad de menciones unicos en el corpus:  2365
```

Figure 9

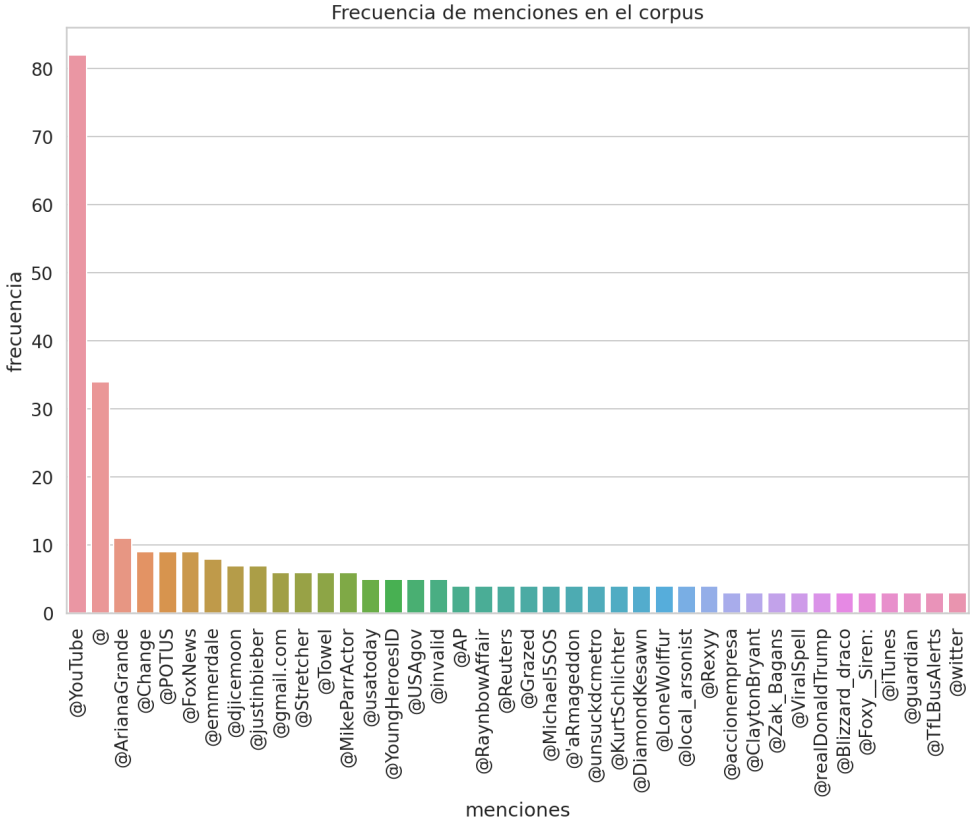

Figure 10

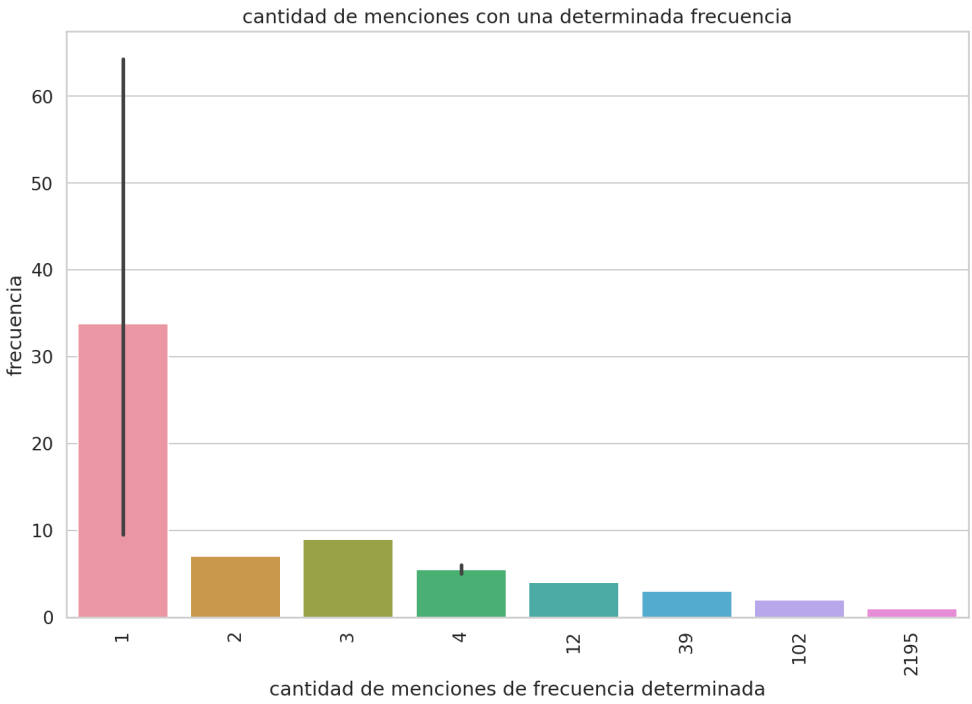

Figure 11

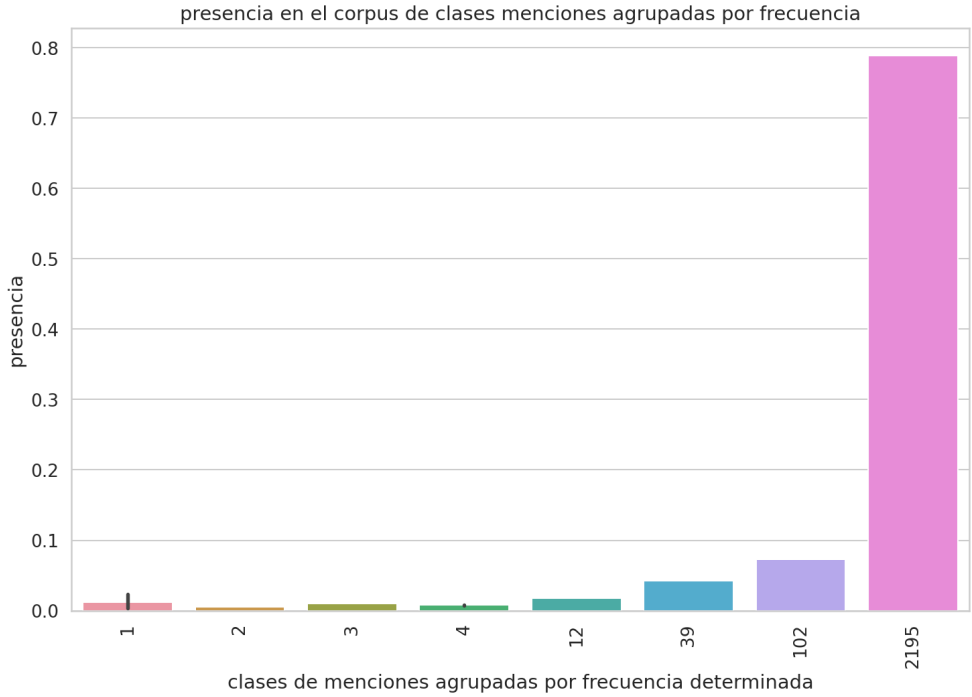

Figure 12

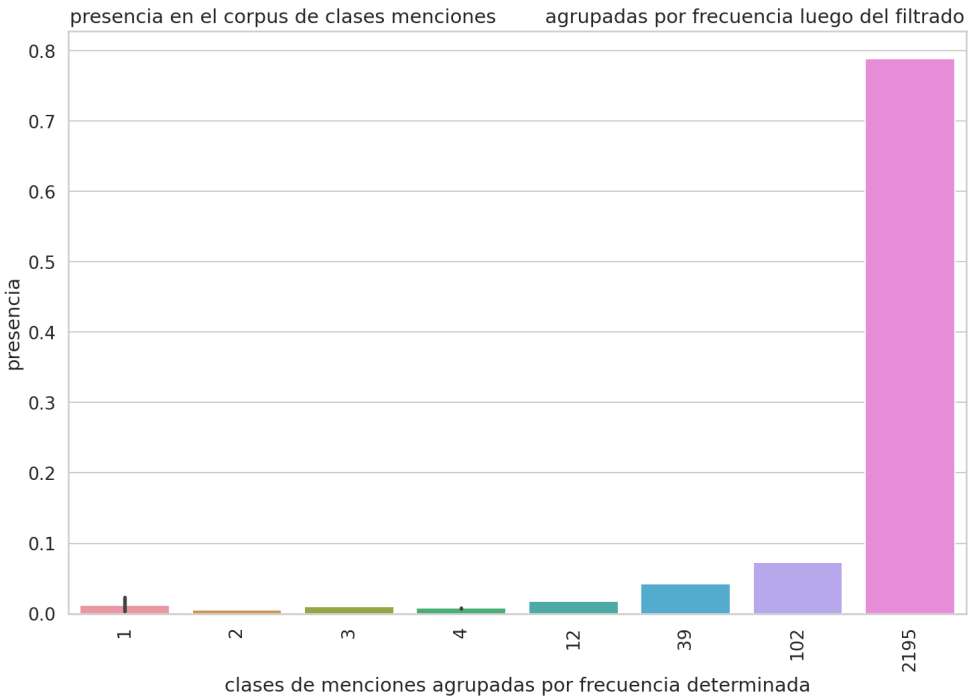

Figure 13

Aca pasa algo especial. Las demas clases sumadas no superan a la clase de las menciones con baja ocurrencia.

Sabiendo que la clase de mayor presencia son menciones de ocurrencia 1, vamos a proceder a eliminar esa clase por mas que clase esta constituya el 79 porciento de las menciones porque esto no es mas que ruido inecesario.

```
Cantidad de menciones que quedan:  589
Cantidad de menciones que que habia:  2784
Porcentaje filtrado por baja ocurrencia: 78.8%
```

Figure 14

2.4.4 Links

```
links_pattern = re.compile('https?://\S+|www\.\S+')
links = pd.Series(re.findall(links_pattern,corpus))
links
```

```
5]
0       http://t.co/lHYXEOHY6C
1       http://t.co/YAo1e0xngw
2       http://t.co/2nndBGwyEi
3       http://t.co/qqsmshaJ3N
4       http://t.co/3ImaomknnA
                 ...
4726    http://t.co/3SICroAaNz
4727    http://t.co/I27Oa0HISp
4728    http://t.co/STfMbbZFB5
4729    http://t.co/zDtoyd8EbJ
4730    http://t.co/YmY4rSkQ3d
Length: 4731, dtype: object
```

Figure 15

```
print("Canitdad de links en el corpus: ",len(links))
print("Canitdad de links unicos en el corpus: ",len(set(links)))
counts = links.value_counts()
counts
```

```
Canitdad de links en el corpus:  4731
Canitdad de links unicos en el corpus:  4606
```

Figure 16

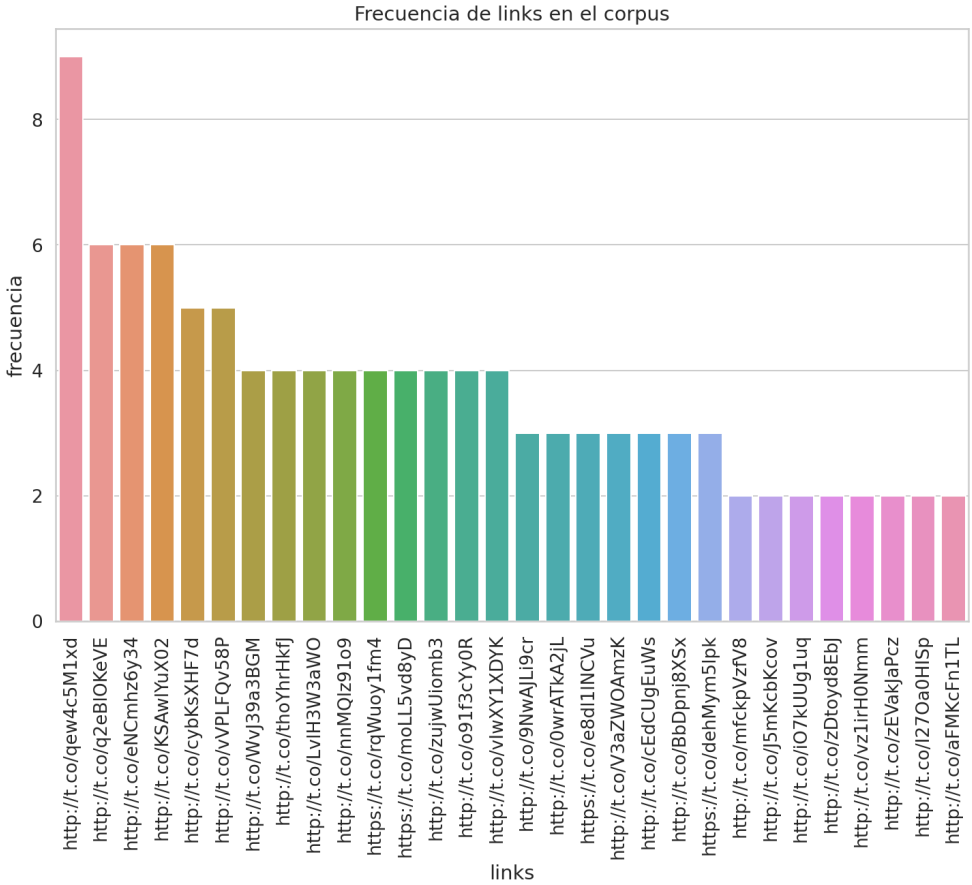

Figure 17

Aparentemente hay algunos links que se repiten asi que vamos a tener una consideracion tambien por los links mas repetidos.

De todas formas si resulta que la repeticion no es tan marcada es posible que terminemos mapeando todos los links a LINK por las razones antes decriptas en los hashtags y menciones

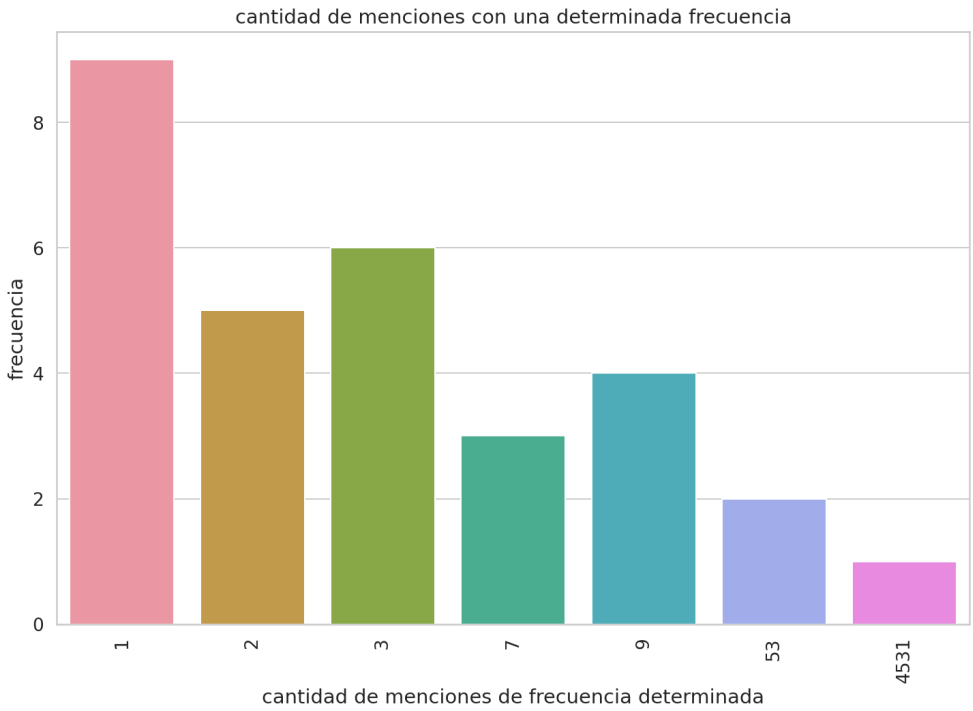

Figure 18

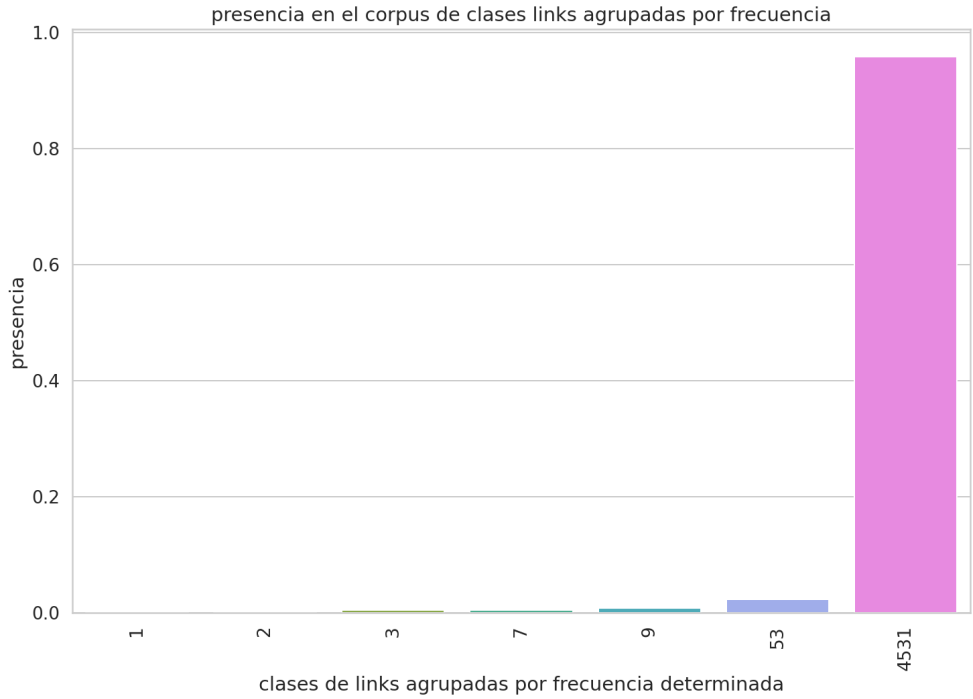

Figure 19

Aca esta clarisimo que la grandisima mayoria de links solo se repiten una vez en todo el corpus por lo que de entrada vamos a descartar esa clase

```
Cantidad de links que quedan:  200.0
Cantidad de links que que habia:  4731
Porcentaje filtrado por baja ocurrencia: 95.8%
```

Figure 20

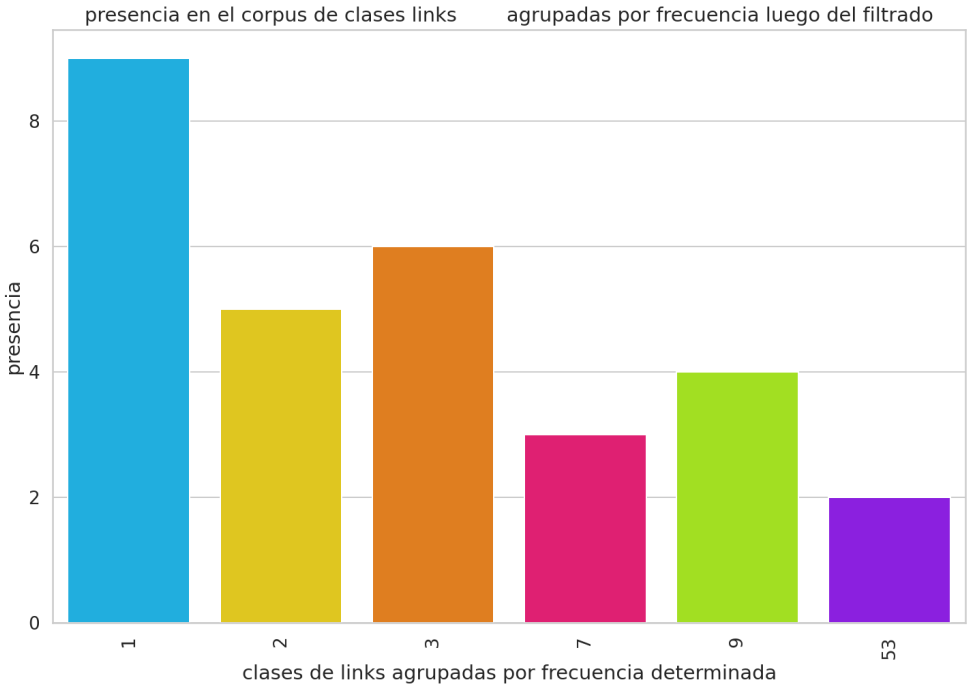

Figure 21

Habiendo filtrado los links que solo aparecen una vez vemos que los restantes ya de por si son pocos y ademas siguen apareciendo poco (los mas frecuentes solo son 1 y aparece solo 9 veces!)

Esto nos lleva a mapear TODOS los links a un tag LINK

2.4.5 HTML

```
html_pattern = re.compile('<.*?>')
htmls = pd.Series(re.findall(html_pattern,corpus))
htmls
```

Figure 22

No hay. Mejor, menos trabajo

2.4.6 Emojis

```
emoji_pattern = re.compile("["
                            u"\U0001F600-\U0001F64F"
                            u"\U0001F300-\U0001F5FF"
                            u"\U0001F680-\U0001F6FF"
                            u"\U0001F1E0-\U0001F1FF"
                            u"\U00002702-\U000027B0"
                            u"\U000024C2-\U0001F251"
                            "]+", flags=re.UNICODE)
emojis = pd.Series(re.findall(emoji_pattern,corpus))
emojis
```

Figure 23

Tampoco hay. Genial

2.4.7 Mails

```
mails_pattern = re.compile('\S+@\S+.com')
mails = pd.Series(re.findall(mails_pattern,corpus))
mails
```

```
0           heccfidmss@gmail.com
1       'fuckface@wineisdumb.com
2     everydaynigerian@gmail.com
3    contactSimpleNews@gmail.com
4          Ashxjonespr@gmail.com
5     zboyer@washingtontimes.com
6    TonyJ@Centralizedhockey.com
7            cigarknub@gmail.com
8          Lovethterry@gmail.com
dtype: object
```

Figure 24

Son realmente pocos y no se reptien asi que mapeamos todos a MAIL.

2.4.8 Fechas y Horas

```
common_times_names = '|'.join([
    'Jan','Feb','May','Apr','March', 'Jun','July','Agugust', 'Sept','Nov','December',
    'Mond','Thurs', 'Wednes','Tues','Frid','Satur','Sunday',
    'Year','Month','Day','Week'
])
times_pattern_1 = re.compile('^.*('+common_times_names+').*$',
                          flags = re.IGNORECASE|re.M)
times = pd.Series(re.findall(times_pattern_1,corpus))

times_pattern_2 = re.compile('\d{2,4}[:/-]\d{2}[:/-]?\d{2,4}?')
times = pd.concat([times,pd.Series(re.findall(times_pattern_2,corpus))])

times_pattern_3 = re.compile('20\d{2}|19\d{2}')
times = pd.concat([times,pd.Series(re.findall(times_pattern_3,corpus))])

times_pattern_4 = re.compile('(?:1|2)\d[:]\d{2}')
times = pd.concat([times,pd.Series(re.findall(times_pattern_4,corpus))])

times
```

Figure 25

```
print("Canitdad de tiempos en el corpus: ",len(times))
print("Canitdad de tiempos unicos en el corpus: ",len(set(times)))
counts = times.value_counts()
counts
```

```
Canitdad de tiempos en el corpus:  746
Canitdad de tiempos unicos en el corpus:  170
```

Figure 26

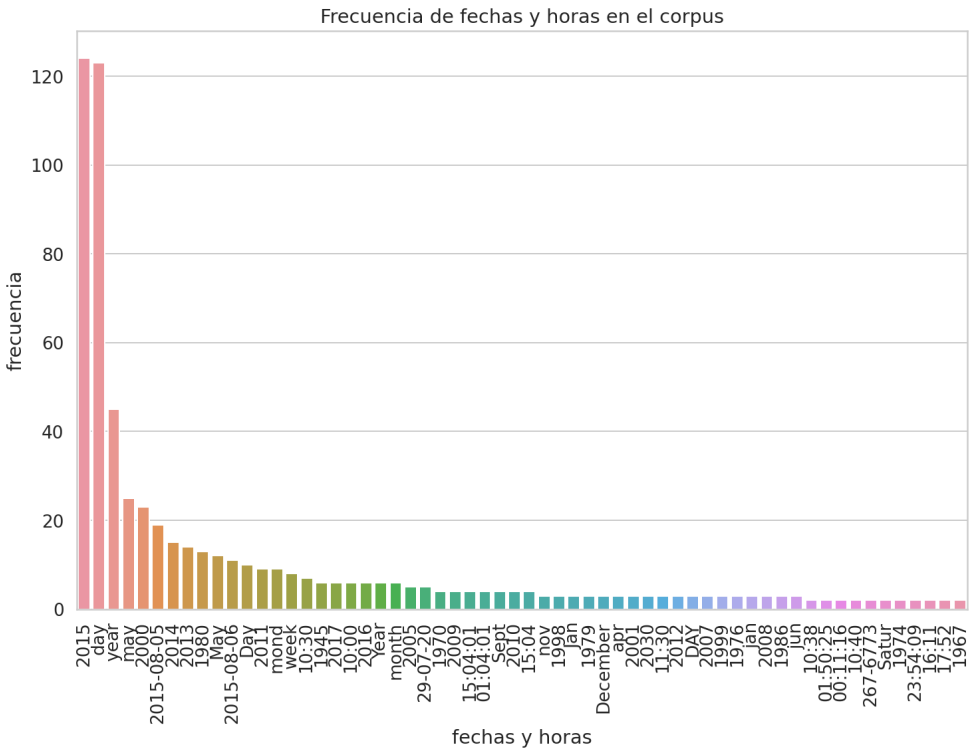

Figure 27

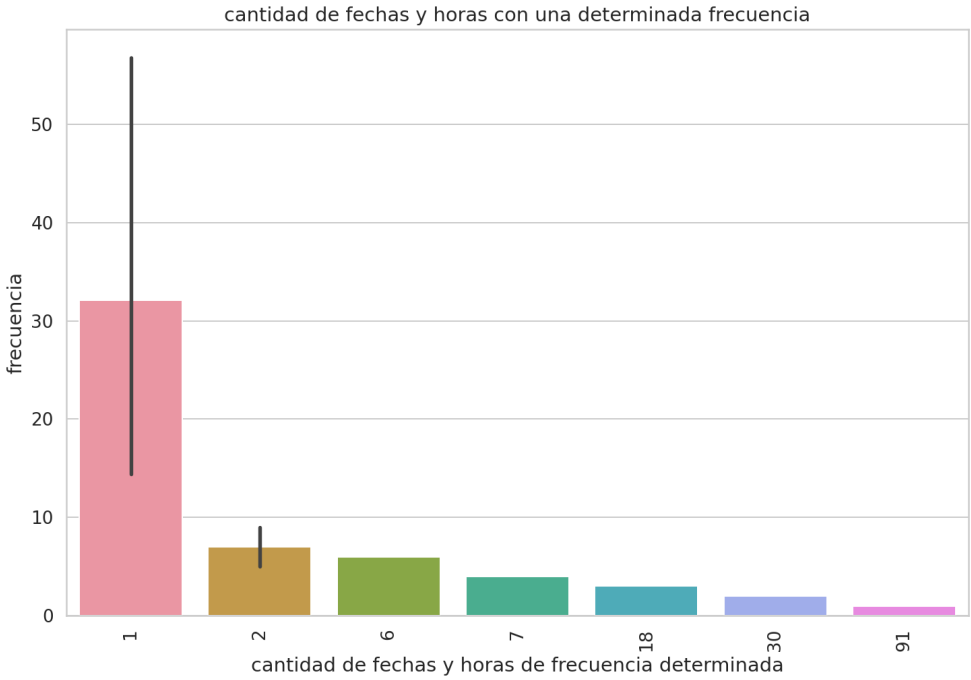

Figure 28

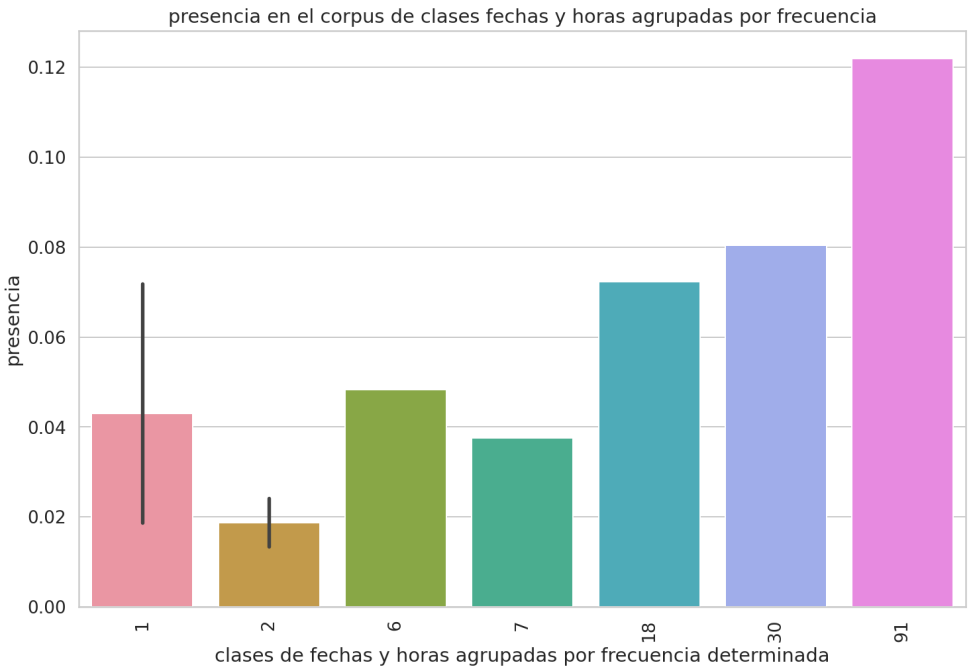

Figure 29

Estos que se muestran a continuacion son los tiempos que no se van a filtrar por su elevada ocurrencia

```
array(['2015', 'day', 'year', 'may', '2000', '2015-08-05', '2014'],
      dtype=object)
```

Figure 30

Vemos que la gran mayoria de tiempos (y en particular fechas) no tienen mucha frecuencia.

Esto es un gran alivio porque de haber tenido mucha frecuencia esto hubiese significado que estabamos contando con un set de datos ordenado temporalmente. Cosa que complejiza el problema.

Con esto descartamos esta hipotesis o al menos descartamos que podamos hacer algo al respecto.

Llama la atencion que 2015 sea un valor frecuente. Quizas el dataset sea de ese año

2.4.9 Simbolos raros

Vamos a ocuparnos de quitar esos simbolos raros.

```
weird_pattern = re.compile('[\x00-\x08\x0b\x0c\x0e-\x1f\x7f-\xff]')
weirds = pd.Series(re.findall(weird_pattern,corpus))
' '.join(weirds)
```

```
'\x89 Û Ò \x89 Û Ó \x89 Û Ó å Ê \x89 Û \x89 Û \x89 Û \x89 Û \x89 Û \x89 Û \x89 Û \x8
\x89 Û Ï \x89 Û \x9d \x89 Û ª \x89 Û Ò \x89 Û \x89 Û ÷ \x89 Û ª \x89 Û Ï \x89 Û \x9d \x89 Û
Û \x89 Û \x89 Û \x89 Û \x89 Û \x89 Û \x89 Û ª \x89 Û Ï \x89 Û \x9d \x89 Û Ò \x89 Û \
89 Û Ê \x89 Û Ê \x89 Û ÷ \x89 Û ª \x89 Û ÷ \x89 Û ª \x89 Û ÷ \x89 Û ª \x89 Û \x89 Û ª \x89
Û \x9d \x89 Û Ï \x89 Û \x9d Ì £ \x89 Û ÷ \x89 Û ª \x89 Û Ì Ñ å Ê \x89 Û \x89 Û \x89 Û \x89
Û \x89 Û Ì ¬ Ì ç Ì £ \x89 Û \x89 Û ª \x89 Û ÷ \x89 Û ª \x89 Û \x89 Û Ó Ì Ï \x89 Û ª \x89 Û
x89 Û Ì ü \x89 Û \x89 Û \x89 Û å \x89 Û \x89 Û \x89 Û Ì Ï å Ê \x89 ã ç \x89 Û Ï \x89 Û \x9d
89 Û \x9d \x89 Û ª \x89 Û ª \x89 Û \x89 Û ÷ \x89 Û ª \x89 Û \x89 Û Ï \x89 Û \x9d Ì \x89 Û \
ç \x89 Û \x89 Û \x89 Û ª \x89 Û Ó \x89 Û ª \x89 Û \x89 Û Ó Ì ¬ \x89 Û ª \x89 Û ª \x89 Û ç \
\x89 Û Ï \x89 Û \x9d \x89 Û \x89 Û Ï \x89 Û \x9d \x89 Û Ï \x89 Û \x9d \x89 Û \x89 Û
```

Figure 31

Investigando un poco aparentemente este es un "simbolo de escape" del teclado. LINK ACA.

Dicho de otra forma. No es nada con un significado semantico para nuestro tweet por lo que dependiendo de su distribucion y presencia en el corpus podemos o mapearlo todo a WEIRD o eliminarlo.

```
print("Canitdad de raros en el corpus: ",len(weirds))
print("Canitdad de raros unicos en el corpus: ",len(set(weirds)))
counts = weirds.value_counts()
counts
```

```
Canitdad de raros en el corpus:  2598
Canitdad de raros unicos en el corpus:  37
```

Figure 32

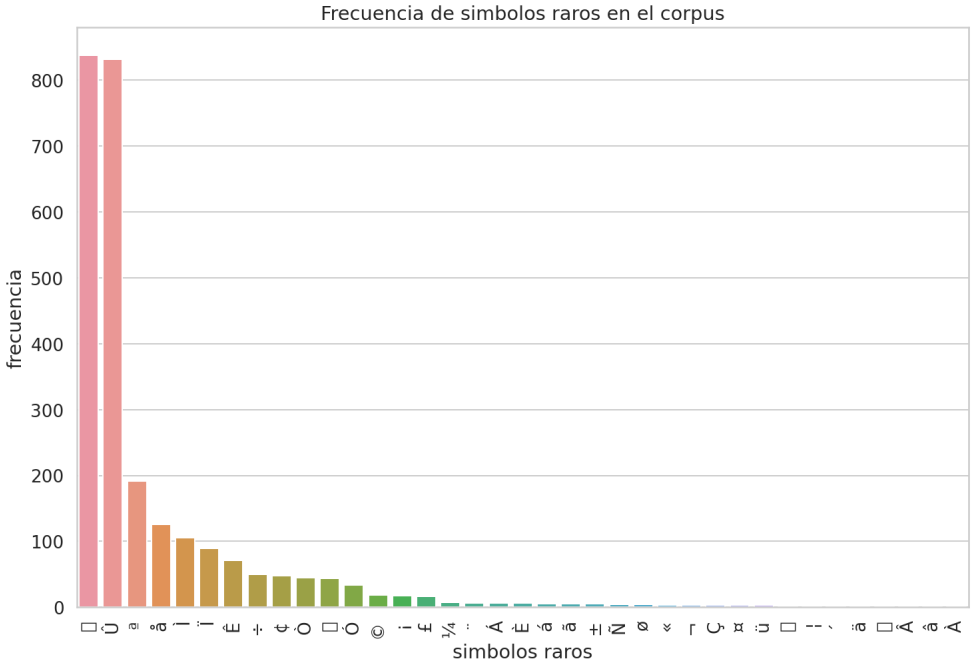

Figure 33

Vemos que realmente no son tan pocos como para tirarlos.

Deberiamos ser lo mas agnosticos posible resepecto de que es semanticamente util para la data y mantener estos simbolos.

De todas formas la decision ahora pasa por mantener mapear todos a un tag o solo algunos.

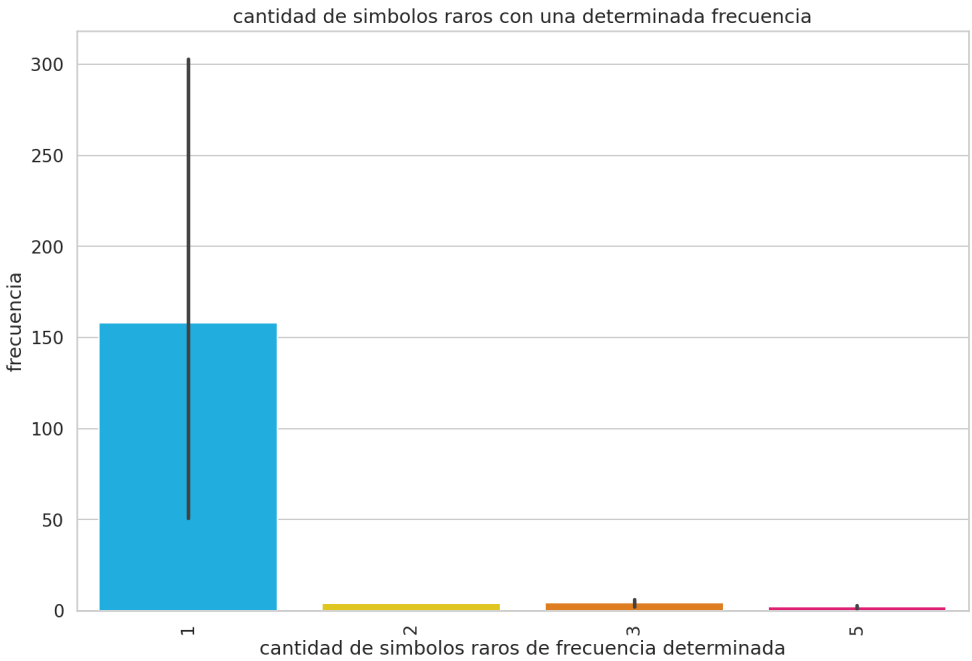

Figure 34

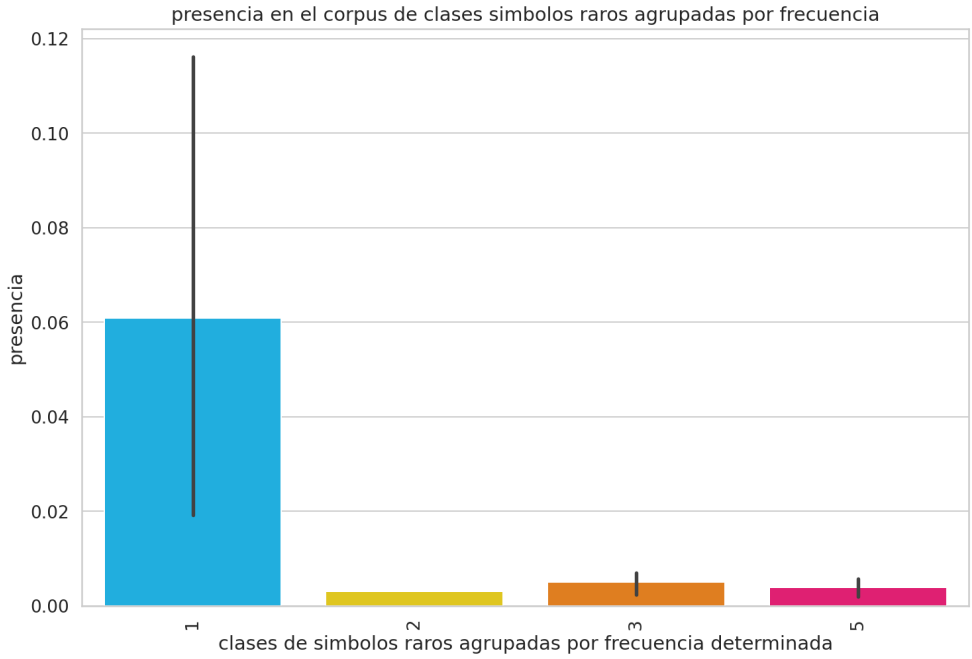

Figure 35

Es interesante ver que esta distribucion es parecida a la de palabras/simbolos relacionados con el tiempo y las fechas o la hora.

De haberlos desechado quizas hubiesemos estado tirando algo tan importante como el tiempo mismo!!!

```
Cantidad de raros que quedan:  1670.0
Cantidad de raros que que habia:  2598
Porcentaje filtrado por baja ocurrencia: 35.699999999999996%
```

Figure 36

hay algunos simbolos de monedas pero tienen muy poca frecuencia y ademas no estamos seguros de que realmente el codigo se decodifique asi o sea el python el que los lee asi.

Viendo que solo hay 37 simbolos unicos y viendo que no parecen significar nada concreto e importante, decidimos mapear TODOS a un tag WEIRD

2.4.10 Patrones numericos varios

```
phone_pattern = re.compile('(\d+[-]\d+)+')
phones = pd.Series(re.findall(phone_pattern,corpus))
list(phones)
```

```
['29-07',
 '29-07',
 '29-07',
 '29-07',
 '29-07',
 '9-11',
 '9-11',
 '1998-99',
 '1-6',
 '88-92'
```

Figure 37

Buscando telefonos nos encontramos con mas formatos numericos extraños (y algunas fechas que ya habiamos encontrado en la seccion anterior). Mapeamos todo esto a NUMERIC PATTERN (no son tantos)

2.4.11 Dinero

```
money_pattern = re.compile('\d*[$€£¥]\d*.?\d+',re.M)
#Si, estamos sesgando un poco no poniendo todas las monedas
#no se puede todo en la vida
money = pd.Series(re.findall(money_pattern,corpus))
money
```

```
0        $15
1        $17
2    £279.00
3     $16.99
4    $238.00
5     $98.00
6       $1.5
7         $2
```

Figure 38

Esta categoria tambien es chica. Todo se mapea a MONEY

2.4.12 Numeros

```
percentage_pattern = re.compile('\d*\s*[%]\s*\d*')
numbers = pd.Series(re.findall(percentage_pattern,corpus))

number_pattern = re.compile('[-]?\d+\.?\d*')
numbers = pd.concat([numbers,pd.Series(re.findall(number_pattern,corpus))])

numbers
```

```
0        %20
1        %20
2        %20
3        %20
4        %20
        ...
11857     04
11858      5
11859      8
11860      4
```

Figure 39

```
Canitdad de numeros en el corpus:  13108
Canitdad de numeros unicos en el corpus:  922
```

Figure 40

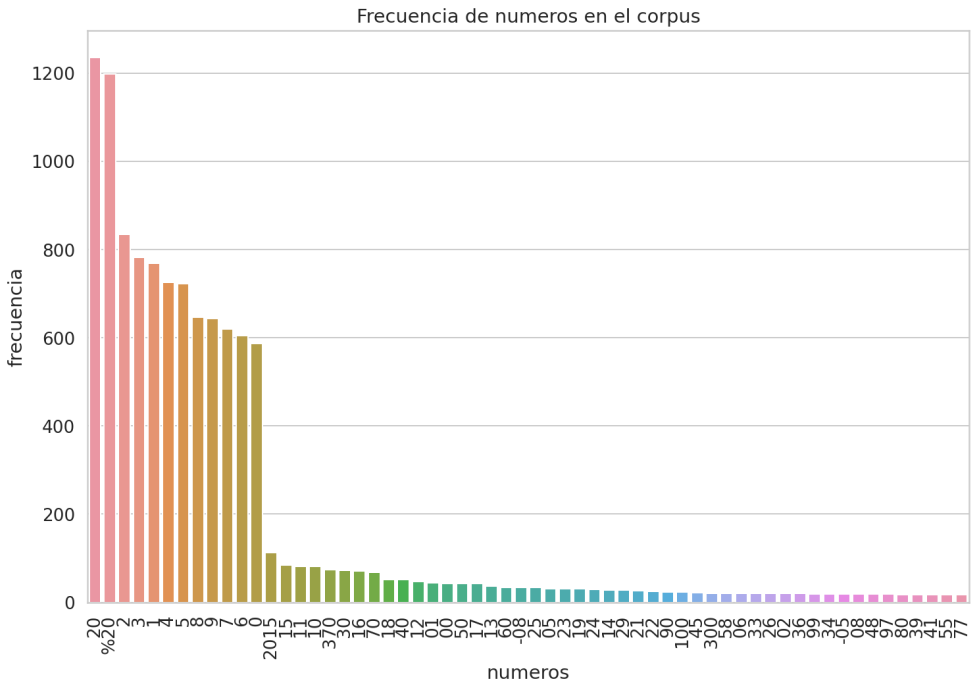

Figure 41

Hay cierta tendencia por los numeros bajos

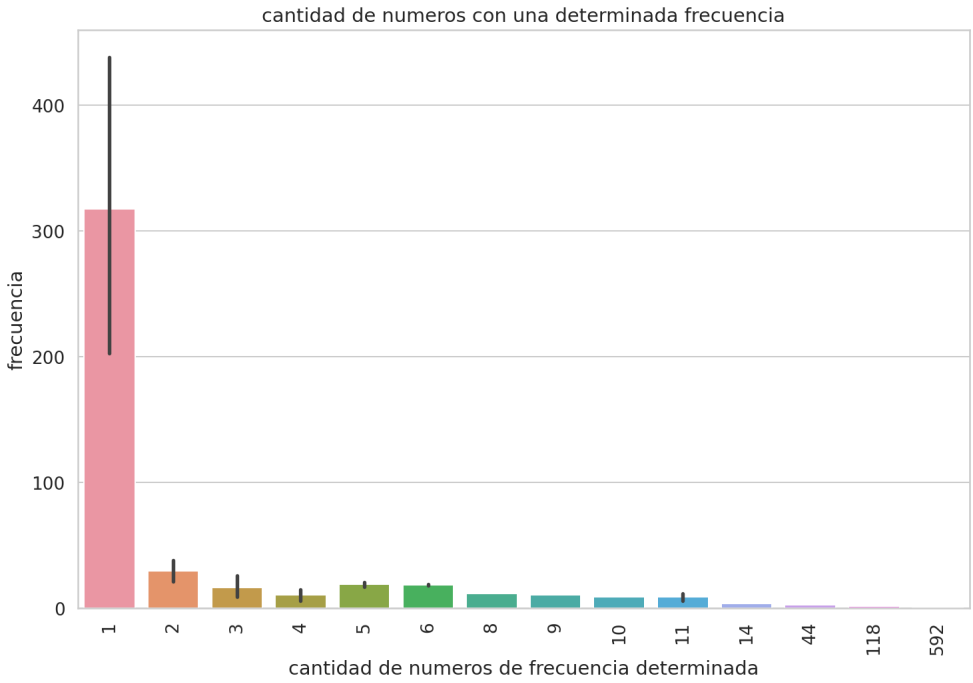

Figure 42

No confundir este grafico. El eje x no son numeros que pone la gente. Son cantidades de numeros que aparecen tantas veces.

Por ejemplo:

La primera barra arrancando desde la izquierda (la que dice 1 en x y 600 aprox en y) significa que hay 1 numero (no sabemos cual) que aparece alrededor de 600 veces.

La barra que tiene 2 en x y aprox 50 en y significa que hay 2 numeros que aparecen aprox 50 veces y asi...

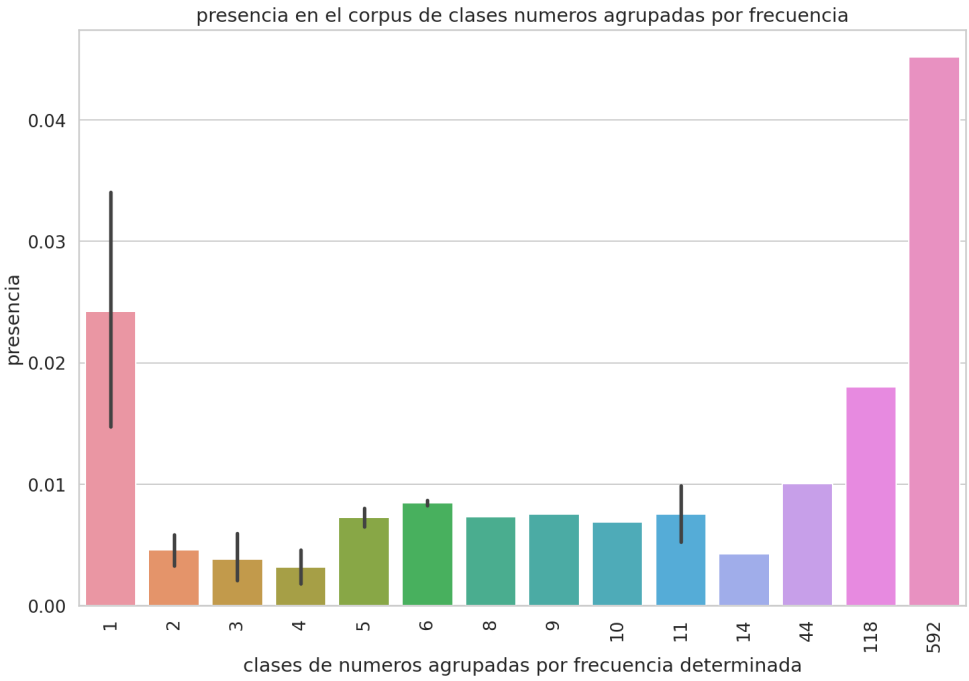

Figure 43

Vemos otra distribucion similar a la del tiempo y fechas y a la de caracteres raros

```
Cantidad de numeros que quedan:  6912
Cantidad de numeros que que habia:  13108
Porcentaje filtrado por baja ocurrencia: 47.3%
```

Figure 44

2.4.13 Signos de Puntuacion que no son ? ni ! (esos los tratamos luego)

```
grammaticals_pattern = re.compile('[.,:;]')
grammaticals = pd.Series(re.findall(grammaticals_pattern,corpus))
grammaticals.value_counts()
```

```
.    12162
:     6966
,     1961
;      595
dtype: int64
```

Figure 45

No los tocamos porque BERT los usa

2.4.14 Signos gramaticales no tan usados

```
punctuation_pattern = re.compile('&amp|[-_\\(\)\[\]\{\}|+=/*&]|%20|<|>')
punctuations = pd.Series(re.findall(punctuation_pattern,corpus))
punctuations.value_counts()
```

```
/       14801
-        1934
%20      1198
_         896
)         419
(         390
&amp      342
|         222
*         189
&         164
[         137
]         136
+          89
=          53
}           6
{           5
>           1
```

Figure 46

```
print("Canitdad de puntuaciones en el corpus: ",len(punctuations))
print("Canitdad de puntuaciones unicos en el corpus: ",len(set(punctuations)))
counts = punctuations.value_counts()
counts
```

```
Canitdad de puntuaciones en el corpus:  20982
Canitdad de puntuaciones unicos en el corpus:  17
```

Figure 47

Las puntuaciones estan ahi y son bastantes.

Para el caso del uso de BERT no vamos a removerlas ya que posee embeddings para estas.

LINK ACA

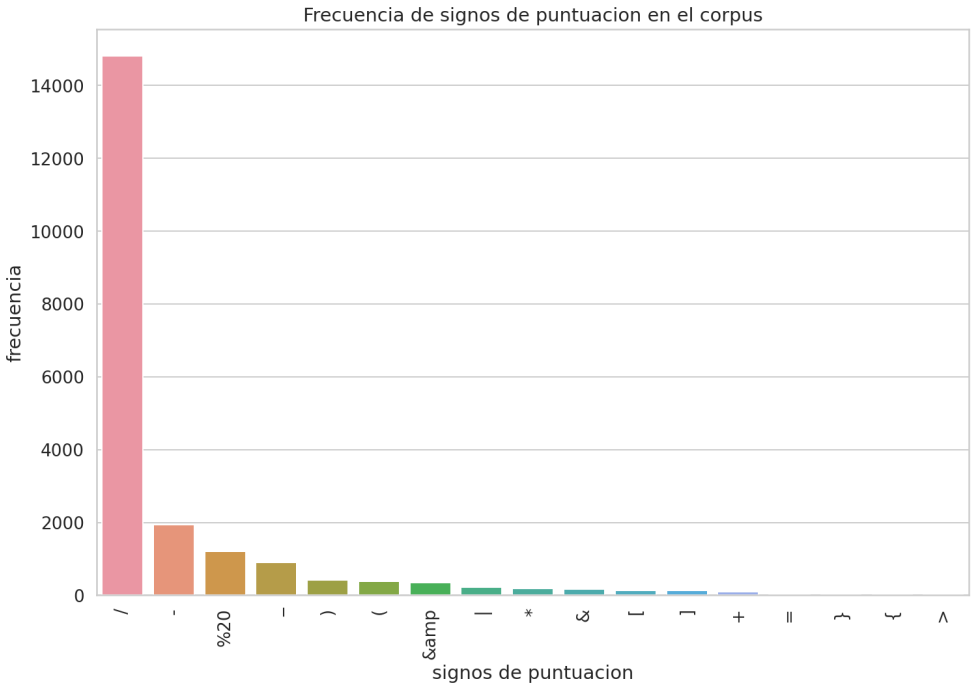

Figure 48

```
Cantidad de puntuaciones que quedan:  14801.0
Cantidad de puntuaciones que que habia:  20982
Porcentaje filtrado por baja ocurrencia: 29.5%
```

Figure 49

2.4.15 Signos de ? y !

Queremos que cosas como !!! se mapeen a ! ! ! para que el tokenizador identique que hay 3 '!' y no 1 '!!!'

```
excl_interr_pattern = re.compile('(?:[a-zA-Z0-9]+|\s)([!?]+)') #no tomar el hashtag '#???'
excl_interr = pd.Series(re.findall(excl_interr_pattern,corpus))
excl_interr
```

```
0            ?
1       !!!!!!
2            !
3            !
4            ?
         ...
2268         !
2269         !
2270         ?
2271        ??
2272        ??
Length: 2273, dtype: object
```

Figure 50

2.4.16 Inpeccion

Antes de aplicar los filtros al corpus inspeccionamos primero el campo de keywords solo y luego el de location solo.

```
keywords_corpus = ' '.join(data.keyword)
keywords_corpus
```

```
ageddon armageddon armageddon armageddon armageddon armageddon armageddon armageddon armaged
mageddon army army army army army army army army army army army army army army army arm
army army army army army arson arson arson arson arson arson arson arson arson arson arson a
son arson arson arson arson arson arson arson arson arson arson arsonist arsonist arsonist a
arsonist arsonist arsonist arsonist arsonist arsonist arsonist arsonist arsonist arsonist ar
arsonist arsonist arsonist arsonist arsonist arsonist arsonist attack attack attack attack a
attack attack attack attack attack attack attack attack attack attack attack attack attack a
attack attacked attacked attacked attacked attacked attacked attacked attacked attacked atta
tacked attacked attacked attacked attacked attacked attacked attacked attacked attacked atta
tacked attacked avalanche avalanche avalanche avalanche avalanche avalanche avalanche avalan
he avalanche avalanche avalanche avalanche avalanche avalanche avalanche avalanche avalanche
avalanche battle battle battle battle battle battle battle battle battle battle battle battl
tle battle battle battle battle battle bioterror bioterror bioterror bioterror bioterror bio
error bioterror bioterror bioterror bioterror bioterror bioterror bioterror bioterror bioter
or bioterror bioterror bioterror bioterror bioterror bioterror bioterror bioterror bioterror
```

Figure 51

Una inspeccion revela que el campo keyword esta realmente muy limpio

```
location_corpus = ' '.join(data.location)
location_corpus
```

```
"                                                          Birmingham Est.
Wide!!   Paranaque City Live On Webcam   milky way       GREENSBORO,NORTH CAROL
eim Abuja USA South Africa Sao Paulo, Brazil hollywoodland  Edmonton, Alberta -
ry, AB Birmingham San Francisco CLVLND Nashville, TN Santa Clara, CA UK St. Lou
Mateo County, CA North Carolina Njoro, Kenya   Your Sister's Bedroom   Arlingto
H Wilmington, NC   New Hanover County, NC   global Alberta | Sask. | Montana Ch
Kenya Instagram - @heyimginog  304 Switzerland 304 US 304 Instagram - @heyimgin
itzerland Switzerland Oshawa, Canada Baker City Oregon     United States 304 30
21 Pennsylvania Salt Lake City, Utah Palo Alto, CA   Spain but Opa-Locka, FL Ja
l News and Views   Indonesia y(our) boyfriends legs  New Mexico, USA Somewhere
Canada   Thrissur Havenford India 92   Israel Fashion Heaven. IG: TMId_ San Fra
LA West Wales   Happily Married with 2 kids  Cambridge, MA Arizona  Mumbai   Am
North Carolina, USA   Karachi Happily Married with 2 kids  Happily Married with
OUTUBE CHANNEL. Lexington   USA Hannover, Germany     Playa Davidson, NC Higher
8+? Greater Manchester, UK Boston     The Canopy Kingdom   USA     the own zone
Views From The Six \x89ÛÒ   University of Toronto   Swaping Around   London Alb
```

Figure 52

En location la gente se pone mas creativa....

Aplicamos los filtros que desarrollamos hasta ahora

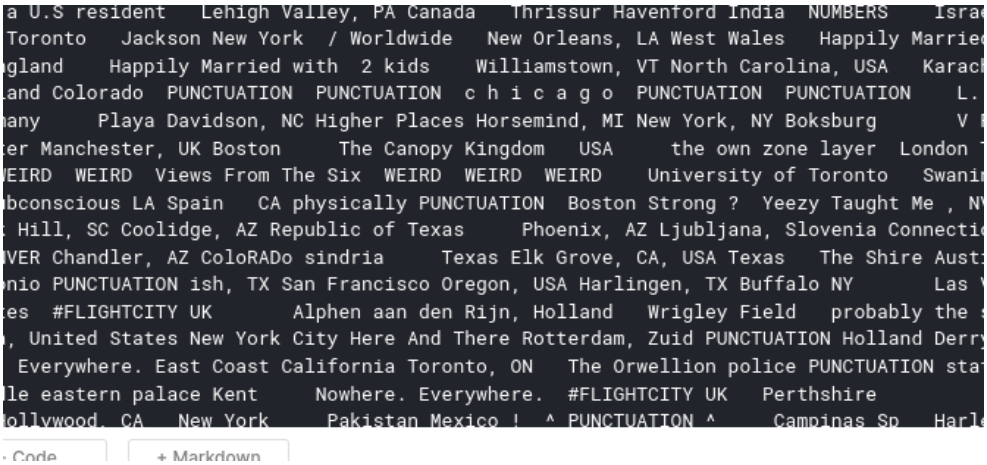

Figure 53

Realmente el texto queda mas limpio

2.4.17 Tratamiento de Lugares, Nombres y Acronimos

Ahora se va a analizar que "lugares" son mas frecuentes, cuantos hay y que filtrar.

Decimos "lugar" porque la gente pone de todo y tranquilamente podriamos detectar 'The Greatest Country Ever' como una location.

No tenemos forma de validar si eso es un pais o no sin tener un diccionario (que tenemos, pero no anda bien).

Estos casos no nos importan. Si es un pais real o si la gente se refiere a el como pseudonimo entonces va a tener mucha frecuencia. Esto es lo que vamos a usar.

Lo mismo hacemos con nombres y acronimos.

```
name_pattern = re.compile('(([\w*\.?]*[A-Z]+[\w*\.?]*\s+[\w*\.?]*[A-Z]+[\w*\.?]*|[\w*\.?]*[A-Z]+[\w*\.?]*)+)')
names = pd.Series([n[0] for n in re.findall(name_pattern,corpus)])
```

Figure 54

```
print("Canitdad de nombres en el corpus: ",len(names))
print("Canitdad de nombres unicos en el corpus: ",len(set(names)))
counts = names.value_counts()
counts
```

```
Canitdad de nombres en el corpus:  35383
Canitdad de nombres unicos en el corpus:  19226
```

Figure 55

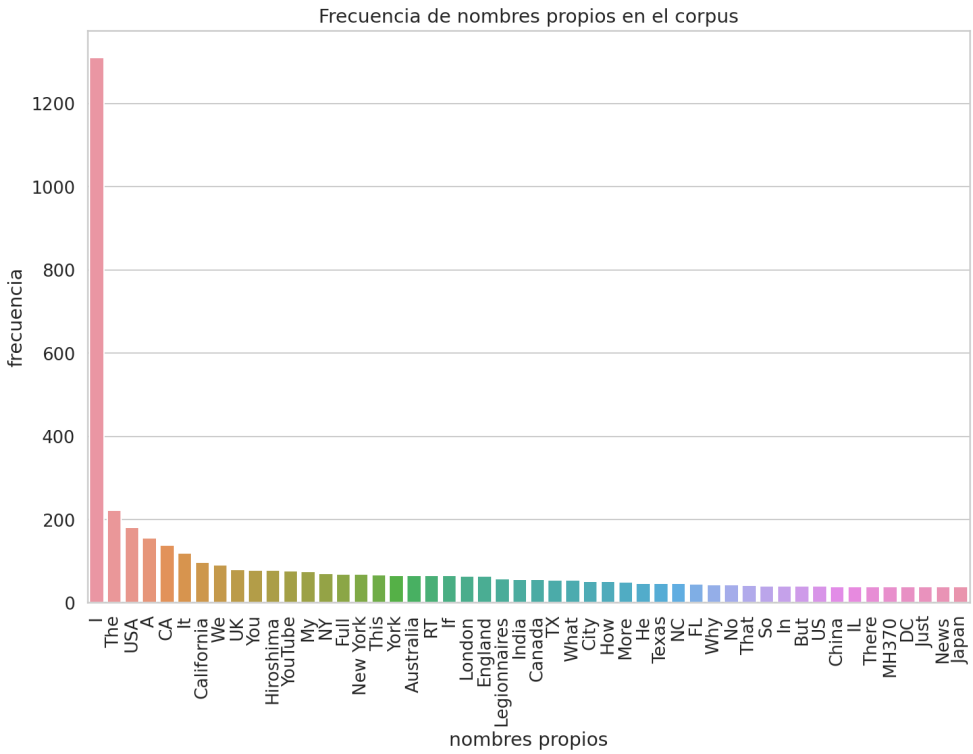

Figure 56

Vemos muchas stopwords aca.

Queremos dejar en claro que estaria MAL quitar las stopwords si pretendemos darle este texto a BERT debido a que BERT aprendio a manejarlas y funciona mejor con ellas.

Dicho esto vamos a quitar las stopwords SOLO de es esta seccion para que el filtrado de nombres sea justo.

Es interesante como dato de color ver que 'I' es mas frecuente que 'The' y que cualquier otro nombre. Mucha gente hablando de uno.

```python
names = pd.Series([n for n in list(names.values) if not n.lower() in stops])
print("Canitdad de nombres en el corpus: ",len(names))
print("Canitdad de nombres unicos en el corpus: ",len(set(names)))
counts = names.value_counts()
counts
```

```
Canitdad de nombres en el corpus:  31662
Canitdad de nombres unicos en el corpus:  19064
```

```
USA            181
CA             139
```

Figure 57

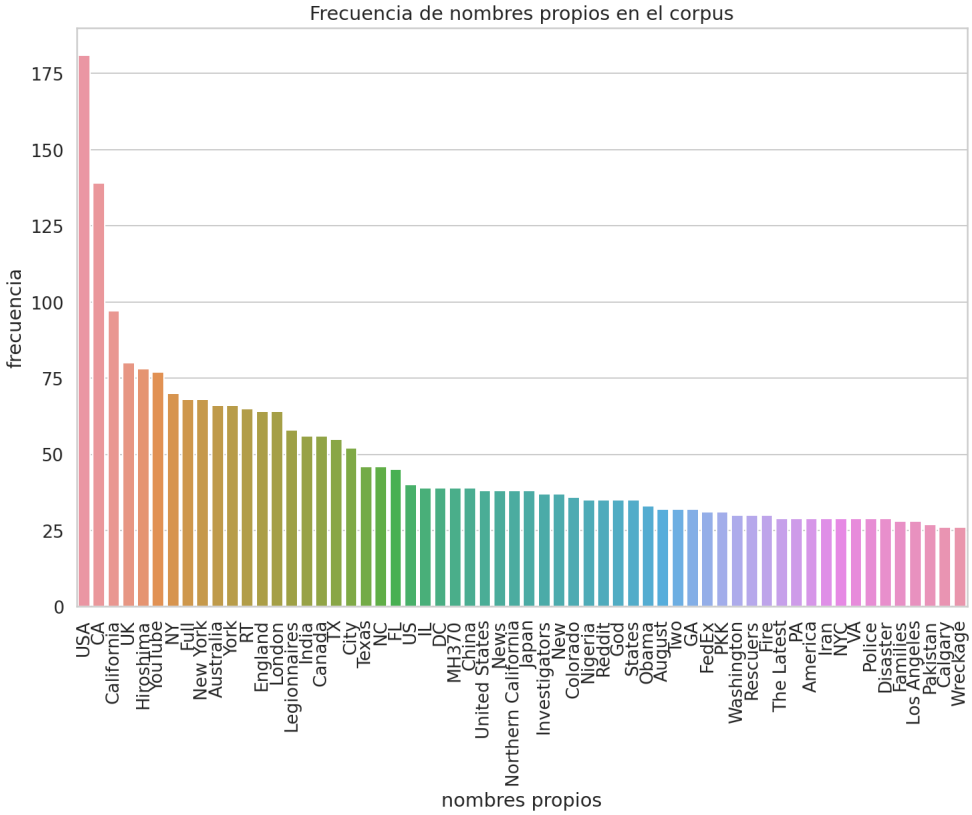

Figure 58

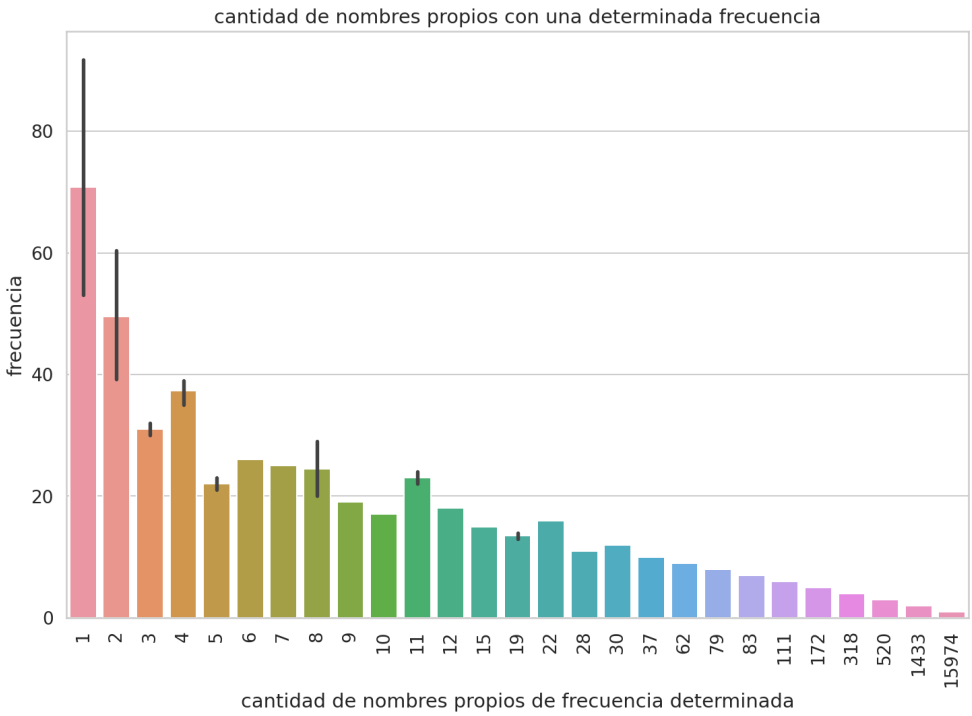

Figure 59

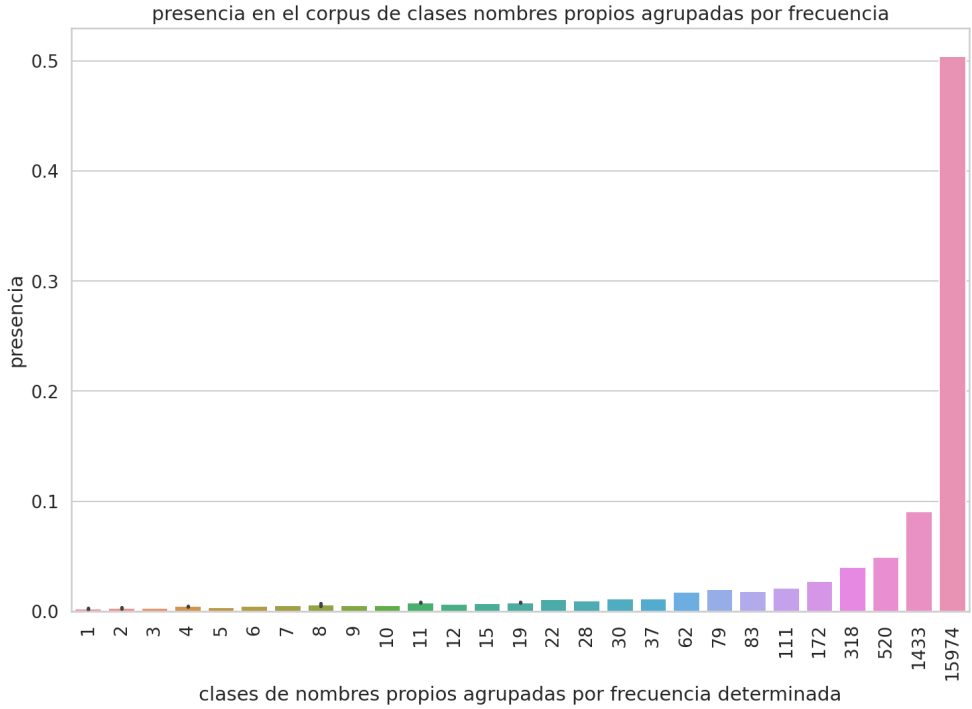

Figure 60

Otra cosa interesante. De los nombres que se usan hay una power law. Hay mas nombres que se usan poco (por mucho) que nombres que se usan mucho.

Aca podemos pasa algo interesante. La clase de los nombres que solo figuran una vez tiene una presencia del 50.4

Esto siginifica que ni todas las otras sumadas la superan. Como no queremos quedarnos con todos los nombres filtramos a mano esta ultima clase.

```
Cantidad de nombres que quedan:  15688.0
Cantidad de nombres que que habia:  31662
Porcentaje filtrado por baja ocurrencia: 50.5%
```

Figure 61

2.4.18 Resultados

```
tags = ' '.join([HASHTAG, MENTION, LINK, HTML, EMOJI, MAIL,
        TIME, WEIRD, NUMERIC_PATTERN, MONEY, NUMBERS,
        PUNCTUATION, NAME])

tokens = pd.DataFrame({
    'content':one_grams,
})
tokens['isTag'] = tokens.content.apply(lambda x: x in tags)

print(f"Porcentaje de tags en corpus: {round(tokens.isTag.sum()/len(tokens),3)*100}%")
```

```
Porcentaje de tags en corpus: 19.6%
```

Figure 62

Un quinto de todos los tokens del corpus es un tag nuestro.

Esto podria ser malo si hicimos mal el trabajo de tageo ya que estariamos tirando a la basura informacion util.

Ahora, si hicimos bien el trabajo de tageo entonces esto es muy bueno porque significa que al menos un quinto del set de datos era ruido que eliminamos.

Terminamos la tarea de limpieza. Lo que generamos a ahora es un train y test que BERT y cualquier red neuronal sera capaz de aprovechar al maximo (en particular las preentrenadas)

Para los arboles de decision, que manejan data estructurada, (como XGBoost), la cosa de valor producida es este notebook mismo. El analisis aca expuesto es capaz de extraer algunos features bastante utiles.

Posteriormente se busco generalizar todo este analisis hecho para que pueda extenderse al set de test sin 'overfitear la limpieza de datos' (sin usar patrones nativos del set de train para filtrar el de test)

Por ejemplo, si en el analisis de hashtags aplicasemos exactamente el mismo filtro que en train estariamos diciendo que test no debe contemplar los hashtags menos frecuentes de train.

El problema con esto es que los hashtags menos frecuentes de train, al ser los menos frecuentes, es probable que figuren poco tambien en el de test o que no figuren en absoluto en el de test. Esto resulta en un filtro inutil.

Siguiendo con el ejemplo, lo que hariamos seria tomar como hashtags a filtrar los menos frecuentes pero de test. A esto nos referimos con generalizar el analisis.

2.4.19 Evaluando que tan bien se limpio la data

Cortesia de https://www.kaggle.com/gunesevitan/nlp-with-disaster-tweets-eda-cleaning-and-bert https://www.kaggle.com/gunesevitan/nlp-with-disaster-tweets-eda-cleaning-and-bert

Se utiliza una conocida y potente herramienta de embeding fuertemente entrenada para constatar que nivel de covertura "reconoce" glove de nuestro dataset antes y despues de limpiarlo.

La idea es que si glove reconoce mucho de el, probablemente el encoder de BERT tambien y eso deberia mejorar su performance. (esto aplica a los modelos preentrenados)

```
GloVe Embeddings cover 51.61% of vocabulary and 82.15% of text in Dirty Training Set
GloVe Embeddings cover 56.65% of vocabulary and 81.40% of text in Dirty Test Set

GloVe Embeddings cover 79.22% of vocabulary and 94.35% of text in Cleaned Training Set
GloVe Embeddings cover 79.44% of vocabulary and 93.53% of text in Cleaned Test Set

GloVe Embeddings cover 79.21% of vocabulary and 93.48% of text in Cleaned not tagged Training Set
GloVe Embeddings cover 79.43% of vocabulary and 92.62% of text in Cleaned not tagged Test Set
```

Figure 63

Se puede ver que la limpieza es muy buena. Segun glove, es mejor la limpieza con los tags pero eso es porque los tags estan compuestos por palabras de lenguaje natural. Al ser este el caso podria estar introduciendo un bias.

Lo rescatable es que sin las palabras mapeadas la evaluacion es similar y mejor que con el texto sucio.

3 KNN

3.1 Resumen

Uno de los algoritmos utilizados fue K-Nearest-Neighbors. El algoritmo es bastante simple y consiste básicamente en, para cada uno de los tweets que queremos predecir su target, buscar los k vecinos más cercanos, siendo k un hiper-parámetro que se pasa cuando se instancia el Classifier. A partir de encontrar estos k vecinos más cercanos, evalúa cuántos vecinos pertenecen cerca del tweets al que queremos predecir el target, dando un valor cercano a estos.

3.2 Motivacion

Este es uno de los algoritmos más simples de Machine Learning, se decidió probarlo ya que suele arrojar resultados decentes sin tener mucha información. Fue nuestro primer algoritmo a usar, y se uso para poner una vara o piso a superar. Es decir de aca para arriba.
Ademas se lo uso para probar algunos features que se nos ocurrian. Nos quedamos sólo con featuess numéricos y encodeamos algunas de las categóricas, realizamos pocos features en este momento, ya que fue de los primeros algoritmos que utilizamos

3.3 Feature Engineering

Con este algoritmo se fueron probando features que se nos fue ocurriendo, al ser el primero.

Por ejemplo. la longitud del tweets, la cantidad de palabras, el promedio de la cantidad de palabras, si tiene cierto keyword (ya que en el TP1 se analizo que ciertos keyword son propensos a ser de un target o del otro).

La mayoria fueron categorias numericas, para facilitar y volver a usar el set en otro algoritmos mejores en el futuro.

3.4 Desarrollo

Se utilizó tres distancias distintas las cuales se obtuvieron resultados bastantes distintos. Utilizamos las distancias euclidiana, manhattan y chebyshev, cada una con K vecinos distintos.

3.5 Resultados

El procesamiento fue relativamente rapido y el mejor resultado en el set de validacion fue con distancia manhattan y k 28. Con un f1 score de 0.73.

Aunque un error al principio fue de no colocar un random state en el split del set de train y validacion, al no colocarlo cada vez que se hacia el split los tweets se separaban al azar y cada vez que ejecutaba el algoritmo daba distinto resultado. Al elegir un random state (87 o 42 segun se pudo averiguar es comun) el resultado se volvio constante.

Figure 64

4 SVM

4.1 Resumen

Las máquinas de vectores de soporte (SVM) son modelos de aprendizaje supervisado que pueden ser formulado para resolver problemas de áreas como clasificación, regresión, entre otros. Las SVM se diseñaron inicialmente para manejar problemas de clasificación con dos clases bajo el supuesto de que estas clases podrían separarse linealmente, como este caso es inusual en problemas reales, se modificó para lidiar con clases no linealmente separables agregando una función de kernel y un factor de penalización. Su característica principal es que los hiperparámetros definen el conjunto de funciones de decisión disponibles [3]. Desde esta dependencia, podemos observar que obtener los hiperparámetros óptimos conduce a la mejor función de decisión de un Modelo SVM. El problema del ajuste de hiperparámetros consiste en encontrar un conjunto de valores que maximizar algunos criterios de rendimiento, como el índice de precisión y la proporción de vectores de soporte en el caso de la clasificación de vectores de soporte (SVC).

4.2 Motivacion

Ya que nos encontramos con un problema complejo, nos parecio una herramienta interesante para probar como funcionaba ya que la misma puede lidiar con espaciones no lineales mediante la utilizacion de los distintos kernels.

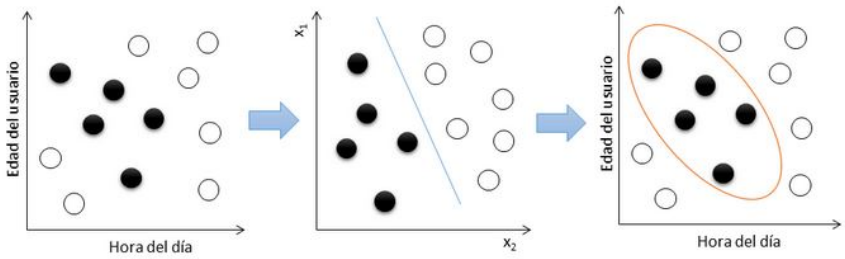

Figure 65

De esta manera con una serie de datos de prueba tendremos caracterizada nuestra clasificación.

4.3 Preprocesamiento

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas adicionando que se escalo los datos para que los mismos enten dentro del mismo rango.

4.4 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas.

4.5 Desarrollo

En pos de encontrar el mejor modelo. Se trabajó en el desarrollo del mismo y optimizacion de los hiper-párametros y selección del kernel. Se utilizó grid-seach para la determinación del parametro C y Gamma con los distintos kernels.

```
def svc_param_selection(X, y, nfolds):
    Cs = [0.001, 0.01, 0.1, 1, 10]
    gammas = [0.001, 0.01, 0.1, 1]
    param_grid = {'C': Cs, 'gamma' : gammas}
    grid_search = GridSearchCV(SVC(kernel='poly'), param_grid, cv=nfolds,scoring='f1')
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_
svc_param_selection(x_train, y_train,5)
{'C': 0.1, 'gamma': 1}
```

Figure 66

Siendo la metodología utilizada en en la figura precedente la seguida y se obtuvo que el mejor modelo, el cual quedó definido por los siguiente hiper-parámetros: kernel='linear',C=258,gamma=

4.6 Resultados

El modelo nos brindo el score: 0.66932 por lo que no se continuó con este algoritmo.

5 NaiveBayes

5.1 Resumen

Los clasificadores ingenuos de Bayes son una colección de algoritmos de clasificación basados en el Teorema de Bayes. No es un algoritmo único, sino una familia de algoritmos donde todos comparten un principio común, es decir, cada par de características que se clasifican es independiente entre sí.

5.2 Motivacion

Es un algoritmo que clasifica de manera individual y ademas es relativamente mas rapidos que los demas. El modelo Naive Bayes es fácil de construir y particularmente útil para conjuntos de datos muy grandes. Junto con la simplicidad, se sabe que Naive Bayes supera incluso a los métodos de clasificación altamente sofisticados.

5.3 Preprocesamiento

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas adicionando que se escalo los datos para que los mismos enten dentro del mismo rango.

5.4 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas.

5.5 Desarrollo

El mismo se utilizó en modo standar, a fin de tener un baseline para otros algoritmos mas complejos. Por lo que no se realizó tunning sobre sus parametros.

5.6 Resultados

Se ubtuvo como resultado el score: 0.69506

6 Pipeline

6.1 Resumen

Pipeline no es un algoritmo en si, es solo un modo de automatizar limpieza de datos, extracción de las características, encodeos de datos, seleccion de features, normalización de las características, reducción de la dimensionalidad, etc. Lo que nos permitiría aumentar considerablemente la productividad.
La automatización del procesado de datos se puede conseguir mediante el uso de pipelines en Scikit-learn. Las cuales se construyen con la clase Pipeline. Permitiendo crear flujos de trabajos en los que se pueden aplicar transformaciones y un estimador a un conjunto de datos.

6.2 Motivacion

Pipeline es solo una noción abstracta, no es un algoritmo ml existente. A menudo, en las tareas de ML, debe realizar una secuencia de diferentes transformaciones (encontrar un conjunto de características, generar nuevas características, seleccionar solo algunas buenas características) del conjunto de datos sin procesar antes de aplicar el estimador final.
El estimador final puede se cualquier algoritmo, y por su simpleza de elegir el estimador final se lo uso en este TP.

6.3 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes-mencionadas.

6.4 Desarrollo

Con pipeline se realizaron dos variantes:

- Solo se utilizo el texto para predecir
- Se utilizo el texto y datos de localizacion, keyword y demas.

En ambos se utilizo countVectorizer(), para contar las palabras y tfidf para el texto.
Y como Clasificadores se utilizaron:

- MultinomialNB
- LogisticRegression
- LgisticRegressionCV
- Perceptron

- XGBoost Classifier

Tambien se usaron LabelEncoder y TargetEncoder con las keywords y BinaryEncoder con los datos preprocesados del continente, ciudad y pais, ademas del keyword, en la variante con locaclizaciones.
Luego para ciertos features que ya venian preprocesados(longitud, count word,etc) se le aplico una transformacion que no hace nada, lo deja pasar, ya que no quiero aplicarle nada a ese feature ya que quiero usarlo asi.

Como en pipeline se debe especificar que transformacion se le aplica a cada columna y no a todas las columnas le aplico la misma transformacion, lo usamos en combinacion con ColumnTransformer para seleccionar cada columna y que transformacion le quiero pasar (al texto se les aplico las antes mencionada)

6.5 Resultados

A continuacion se mostraran los puntajes de solo la primer variante (solo texto), ya que basandome en ella nos dedicamos, en la segunda variante, a usar ese estimador.

- MultinomialNB : 0.7909
- LogisticRegression : 0.7912
- LogisticRegressionCV : 0.7916
- Perceptron : 0.75543
- XGBoost Classifier : 0.7839

Y por ultimo, los resultados de la segunda variante fueron(con LogisticRegressionCV):

- 0.79926 sin tunning
- 0.79987 con tunning

7 Logistic Regression

7.1 Resumen

La Regresión Logística describe y estima la relación entre una variable binaria dependiente y las variables independientes. Logistic Regression lleva el nombre de la función utilizada en el núcleo del método, la función logística es también llamada función Sigmoide. Esta función es una curva en forma de S que puede tomar cualquier número de valor real y asignar a un valor entre 0 y 1.

7.2 Motivacion

Logistic Regression es uno de los algoritmos de Machine Learning más simples y más utilizados para la clasificación. Su simplidad para implementarlo, sumado a que puede ser usado como punto de partida para cualquier otro algoritmo hace de este un metodo interesante para probar.

7.3 Preprocesamiento

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas adicionando que se escalo los datos para que los mismos enten dentro del mismo rango.

7.4 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas.

7.5 Desarrollo

Para el desarrolo del algoritmo se destino la mayor parte del tiempo a:

- Comprender el funcionamiento del algoritmo.
- Buscar librerias e implementaciones a fin de ver ejemplos de uso y funcionamiento para adaptarlos a nuestra necesidad.
- Optimización de los hiper-parámetros

En la busqueda de mejorar lo mas posible el modelo predictivo, nos embarcamos en la busqueda y mejora de los hiper-parámetros. Se realizó la optimización de los siguientes hiper-parámetros:

- Penalidad (penalty).
- Intensidad de la penalización (C).

- Solucionadores (solver)

En los tres casos de la busqueda los hiperparametros, se usó grid search con cross-validation y 5 folds. De esta manera se obtuvo la mejor combinación de los parametros a la cual se la continúo mejorando por la misma vía hasta llegar al siguiente modelo.

```
logreg = LogisticRegression(C=2.3357214690901213, class_weight=None, dual=False,
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                    max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                    random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                    warm_start=False)
logreg.fit(x_train, y_train)

evaluate(logreg, x_train,y_train, x_test, y_test)
```

```
Entrenamiento: 96.1464%
Testeo: 95.8508%.
F1 Score:  0.9585084033613446
F1 Score default:  0.9502831969792322
```

Figure 67

Se obtuvieron los mejores hiper-parametros y estos fueron: Best Penalty: l1 , Best C: 2.3357214690901213 , Best solver: warn

7.6 Resultados

Se obtuvo como mejor resultado: 0.71467

8 Context

8.1 Resumen

En esta seccion se discute la motivacion, metodologia y resultados sobre la adaptacion de la clase Context como algoritmo de machine learning y sobre el uso de la negativad que esta genera como predictor del target.

8.2 Motivacion y Preprocesamiento

En el TP1 habiamos desarrollado una clase Context que a grandes rasgos realizada los siguientes pasos:

- Se tokenizaba y mapeaba las palabra a su significado (en forma rudimentaria)
- Se recolectaba pares de tokens por tweet y su frecuencia global a todo el dataset. Esta informacion se almacenaba en lo que llamamos en la clase contextualMap.
- Se calculaba una negatividad intrinseca para cada token, donde esta se definia como P(desastre / token)
- Para cada token en un tweet se calculaba la negatividad contextual, siendo una medida de la negativdad que resultaba de figurar dicho token con otros tokens negativos en el mismo tweet. Algo asi como una negatividad transferida. Como se calcula esto esta bien explicado en el informe del TP1 pero a grandes rasgos tiene que ver con esas frecuencias de pares de palabras calculadas en el segundo item mencionado.
- Sumando negatividad intrinseca y contextual para cada token en un tweet y luego sumando cada token calculabamos la negatividad de un tweet. Este valor resultaba monotono creciente con la probabilidad de ser desastre.

8.3 Feature Engineering

Dado que queremos ver que tan bien funciona la negatividad para predecir el target utilizamos a la negatividad como unico feature, por lo que no hubo feature engineering involucrado.

8.4 Desarrollo

Hasta ahora lo que hacia Context es asignar a cada tweet una negatividad y en el proceso recolectaba informacion acerca de la relacion entre los tokens y como estos a su vez se relacionan con los desastres. Para utilizarlo como predictor necesitabamos algunos ajustes y algunos supuestos.

8.4.1 Supuestos

Context en su version mas basica requeria del target del tweet para obtener la informacion necesaria para calcular la negatividad. Si queremos usar a Context como predictor necesitamos cambiar esto. Lo que se va a hacer es asumir que la negatividad intrinseca de un token cualquiera que figure tanto en el set de train (con el que se entrena a Context) como en el de test es invariante. Pensamos que esto es un buen supuesto porque esto es lo que significa que la negatividad sea intrinseca. Si el token figura en el set de test pero no en el de train lamentablemente no queda otra que decir que su negatividad intrinseca es 0. Esto podria significar un problema para el algoritmo.

En cuanto a la negativdad contextual casi no hay cambios. Para su calculo se toma en cuenta el contextualMap obtenido del set de train pero tambien ahora el obtenido de procesar el set de test.

Con estos dos valores de negatividad calculados ahora podemos calcular un valor de negatividad razonable sobre un dataset que no tiene labels.

8.4.2 Desarrollo continuacion

La idea es mapear este valor de negatividad a un intervalo de 0 a 1 de forma tal que el 0.5 corresponda al valor de negatividad n tal que a partir de el para arriba el tweet pueda se clasificado como desastre con suficiente "confianza".

La forma en la que vamos a encontrar ese valor de n es a travez de probar diferentes valores para n y evaluarlos con el score F1. El que mejor de es el que usaremos para el mapeo.

El mapeo se realizara a travez de una sigmoidea que se sabe que sirve para tareas de clasificacion.

Concretamente la forma en la que se va a mapear la negatividad a una probabilidad de desastre es asi:

```
def mapNegToLabel(x,a,b):
    """
    RECIBE: Una negatividad x y unos parametros a y b
    DEVUELVE: 1 o 0, que intentaria ser la prediccion del target en base a x
    """

    s = a * x + b
    sigm = 1/(1 + np.exp(-s))
    return 1 if sigm >= 0.5 else 0
```

Figure 68

El n del que se habla antes es aca lo que seria el -b ya que este representa un corrimiento horizontal de la funcion sigmoidea.

Se empieza por buscar el b optimo con un a = 1 porque se intuye que el b es mas determinante que a. Luego se busca linealmente a para ajustar el resultado. Por esta razon se recurre a un gridsearch artesanal.

8.4.3 Tuneo de hiperparametros

Al realizar este gridsearch del que se hablo los valores a los que se arriban para a y b son de:

- a = 1
- b = -17.34
- f1 score obtenido con train: 0.6828

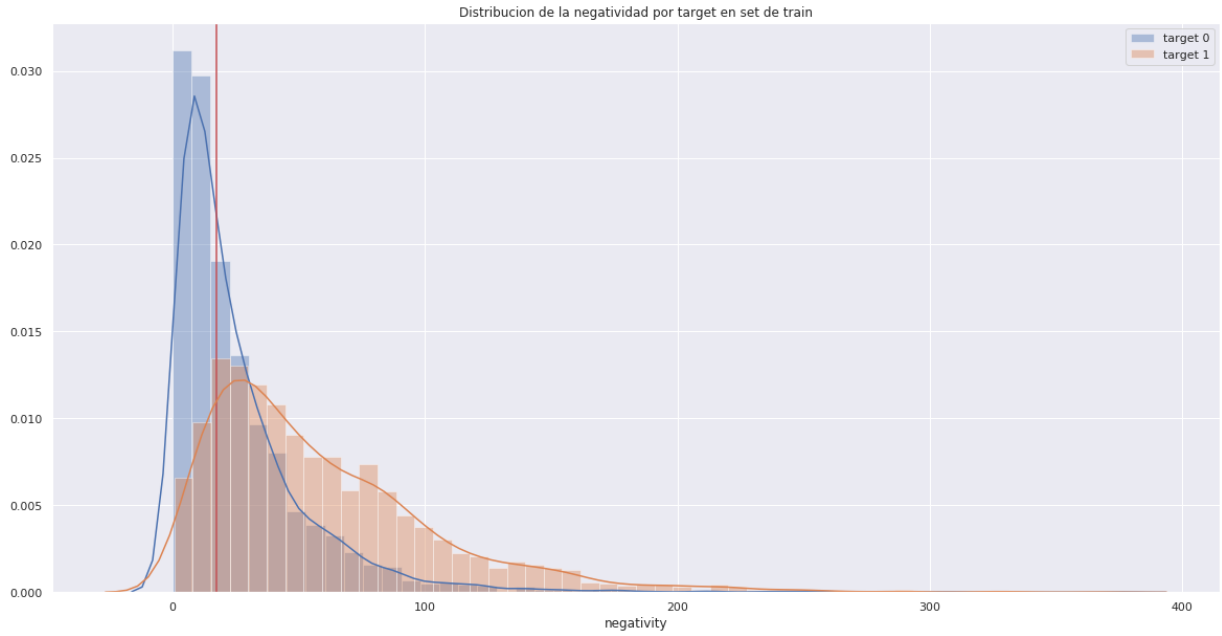

Figure 69: La linea roja vertical que se ve en el grafico es el valor de n de negatividad optimo a partir del cual se considera que cualquier tweet de negatividad mayor a el es un desastre.

8.5 Resultados

Es interesante notar que en el TP1 se observo que el mejor valor para separar desastre de no desastre parecia andar alrededor de 20 y que aca se vea que es 17.34 aprox.

El resultado obtenido en el submit fue de 0.60220. Nada impresionante pero por lo menos es mas que 0.5 por lo que la negatividad no es un feature completamente ortogonal a la idea de desastre.

9 Random Forest

9.1 Resumen

A este algoritmose lo puede describirse como un conjunto de árboles de decisión y es uno de los métodos de Ensemble Learning (aprendizaje conjunto). La idea detrás de los métodos de Ensamble es combinar los modelos más débiles para construir un modelo con menor error de generalización, menos susceptible al overfitting. El funcionamiento del algoritmo se puede resumir esquemáticamente en cuatro pasos:

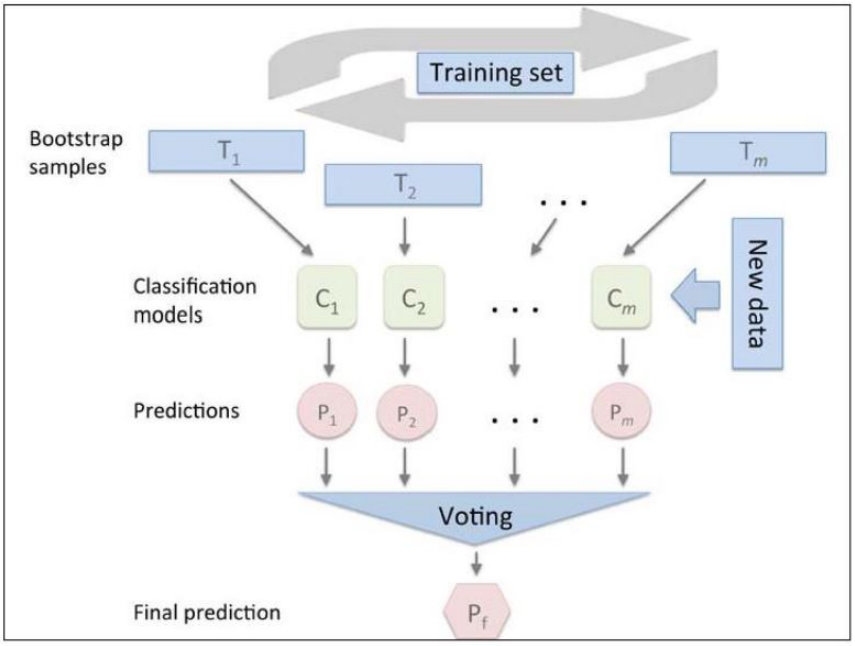

Figure 70

- Se toma una muestra aleatoria con reemplazo de tamaño N elegido aleatoriamente.
- Se genera un árbol de decisión con la muestra anterior.
- Se repiten los pasos anteriores k veces.
- Se asigna una predicción promediada teniendoen cuenta la predicción de cada árbol.

9.2 Motivacion

Este algoritmo fue uno de los que primeros decidimos en utilizar, el mismo fue de nuestro interes, ya que en las clases nos comentaron que es uno de los que permite ver la importancia de los features creados y ademas es uno de los mas utilizados en Machine Learning.

9.3 Preprocesamiento

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas adicionando que se escalo los datos para que los mismos enten dentro del mismo rango.

9.4 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas.

9.5 Desarrollo

Se decidio realizar el tuneo de los hiper-parámetros utilizando optimización bayesiana. Esta modalidad de busqueda consiste en crear un modelo probabilístico en el que el valor de la función objetivo es la métrica de validación del modelo. Con esta estrategia, se consigue que la búsqueda se vaya redirigiendo en cada iteración hacia las regiones de mayor interés, ya que en grid search o random search no tiene en cuenta y se evaluan los parametros sin esta inteligencia, como se ve en la siguiente imagen.

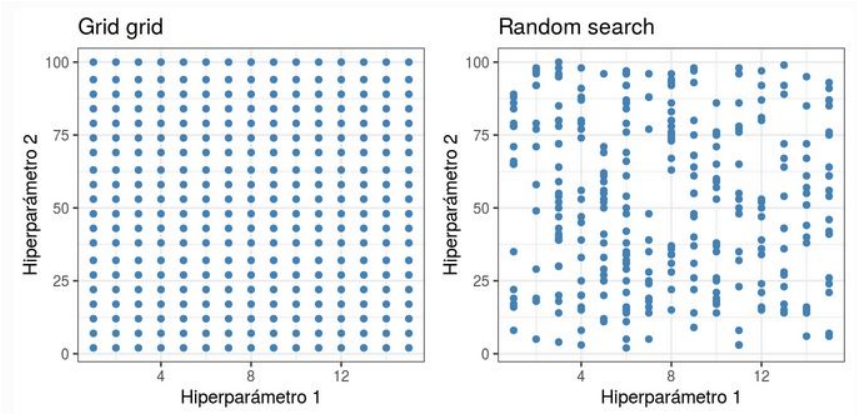

Figure 71

Mientras que si utilizamos optimizacion bayesiana la busqueda del hiper-parámetro se vería de la siguiente manera, como se aprecian en la gráfica siguiente. Que los parametros testeados van como convergiendo hacia el mejor valor sin necesidad de testear valores que ya sabemos que no nos serán utiles.

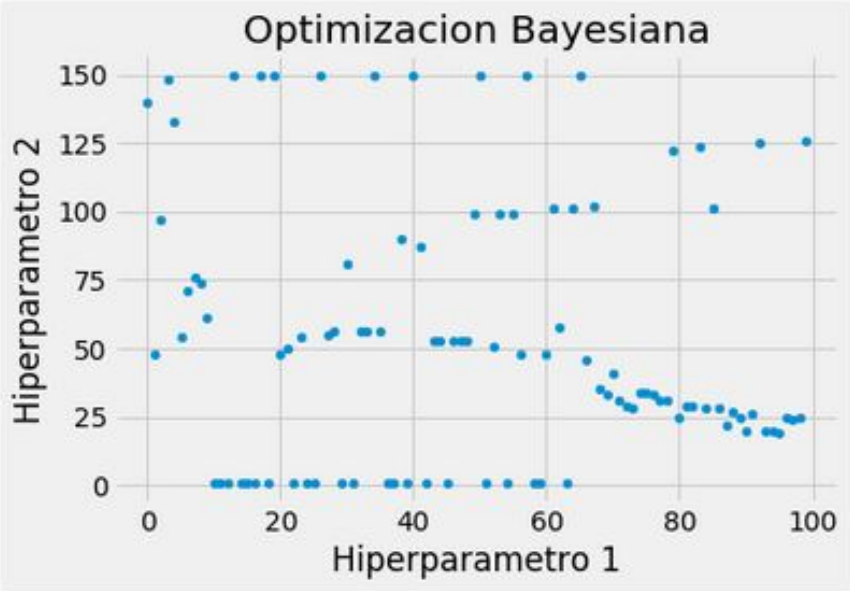

Figure 72

El objetivo final es reducir el número de combinaciones de hiperparámetros con las que se evalúa el modelo.

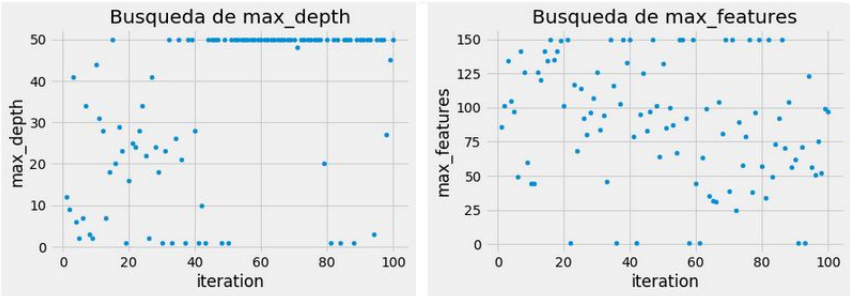

Figure 73

Por lo que este metodo es ideal ya que con menor tiempo nos permite ir eligiendo únicamente los mejores candidatos. Esto significa que, la ventaja frente a las otras estrategias mencionadas, se maximiza cuando el espacio de búsqueda es muy amplio o la evaluación del modelo es muy lenta. Se realizaron la busqueda obteniendo las graficas precedentes,asi como tambien, siguientes gráficas.

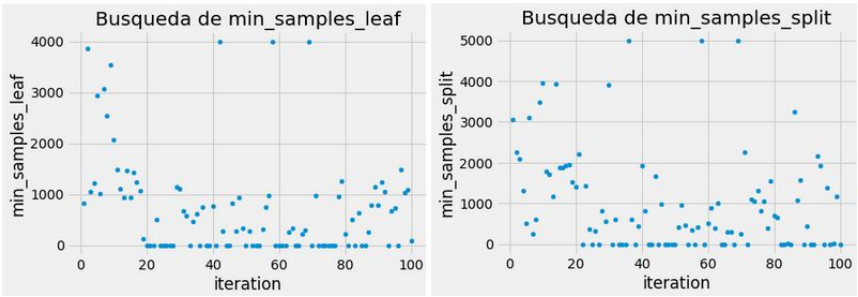

Figure 74

Un arbol por ejemplo se ve con la siguiente estructura:

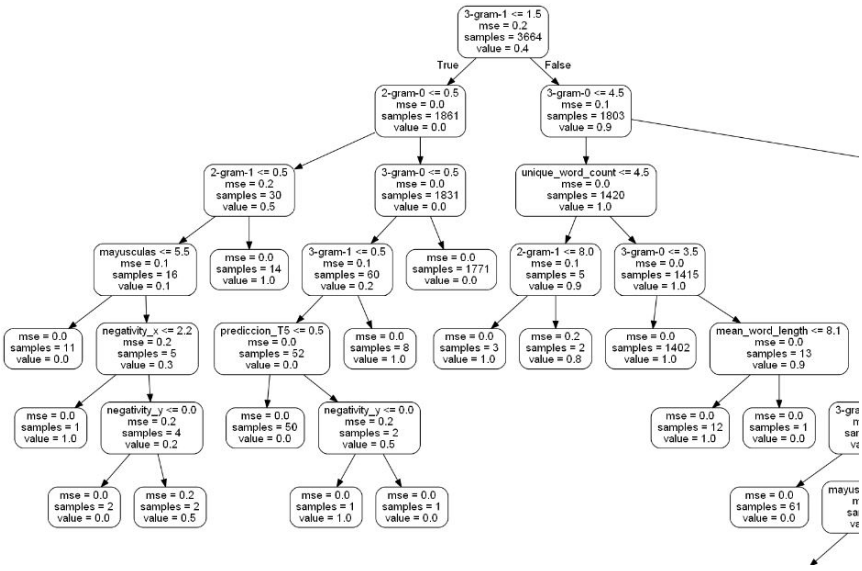

Figure 75

Si vemos el un arbol completo, este, tendria la siguiente estructura (figura siguiente) pero dada la cantidad de información que contiene es muy dificil poder analizarlo completo en papel.

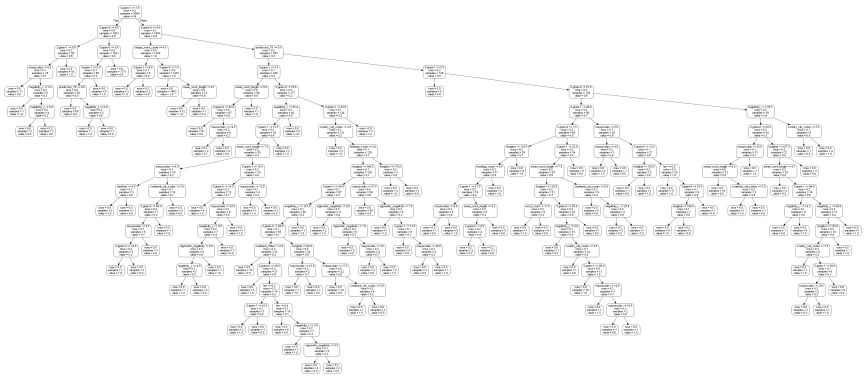

Figure 76

9.6 Resultados

Intentamos con varios submits, pero vimos que otros algoritmos tomaban amplia ventaja sobre este por lo cual decidimos incrementar los esfuerzo en estos. Se obtuvo un score de: 0.71805

10 Decision Tree

10.1 Resumen

El árbol de decisión es un tipo de algoritmo de aprendizaje supervisado, mayormente usado en situaciones en las que se desea clasificar. Este algoritmo admite variables de entrada y salida categóricas y continuas. Con la información que tiene al ir procesando y conocer la clases, el algoritmo va realizando distintos cortes (splits) en nuestro set de datos dividiendo los mismos en conjuntos mas significativos para el feature que se esta procesasando.

10.2 Motivacion

Se inicio con este algoritmo como una primera aproximación, por su simplicidad a la hora de su uso, para tener un marco de referencia para cuando se esté trabanado con Random Forest. Las principales ventajas de este método fueron:

- Su sencilles, tanto en el uso como en la información que proporciona.
- El tipo de dato no es un inconveniente, ya que al implementarlo, puede manejar tanto variables categóricas como numéricas.
- Es un método no paramétrico, por lo cual no genera soposiciones sobre la distribucion espacial o sobre la estructura del clasificador.

10.3 Preprocesamiento

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas adicionando que se escalo los datos para que los mismos enten dentro del mismo rango.

10.4 Feature Engineering

Se trabajo un set de datos ya preprocesado, en el mismo se realizaron las acciones antes mencionadas.

10.5 Desarrollo

Para el desarrolo del algoritmo se destino la mayor parte del tiempo a:

- Comprender el funcionamiento del algoritmo.
- Buscar librerias e implementaciones a fin de ver ejemplos de uso y funcionamiento para adaptarlos a nuestra necesidad.
- Optimización de los hiper-parámetros

Se realizó la optimización de los siguientes hiper-parámetros:

- Profundidad del Arbol (max depth).
- Porcentaje de la muestra en hojas (min samples split).
- Muestra minima por nodo (min samples leaf)

En los tres casos se comenzó la busqueda en un amplio rango y luego se fue reninando la misma.
En el caso de Profundidad del Arbol (max depth). Se probaron valores entre 1 y 10.

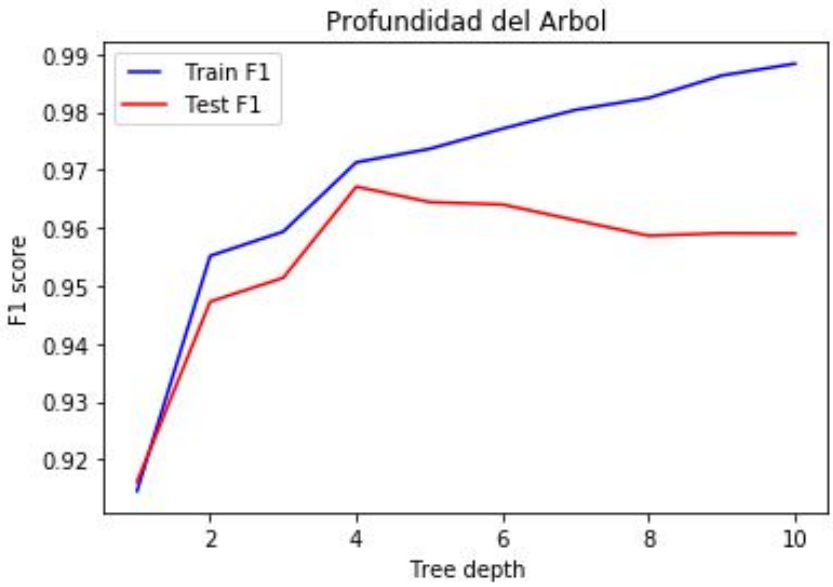

Figure 77

Se obtuvo el mejor valor en max depth=4.

Luego se realizó la busqueda de Porcentaje de la muestra en hojas (min samples split). Probando varios valores entre 0 y 1.

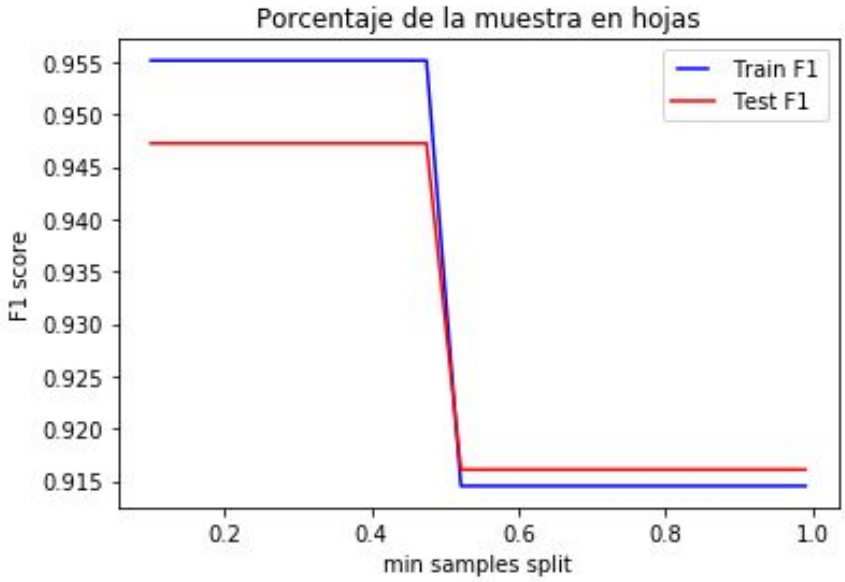

Figure 78

Se encontro el mejor valor para min samples split=0.49210526

Para la busqueda de muestra minima por nodo (min samples leaf) se procedio de la misma manera que las ocasiones anteriores buscando el valor adecuado.

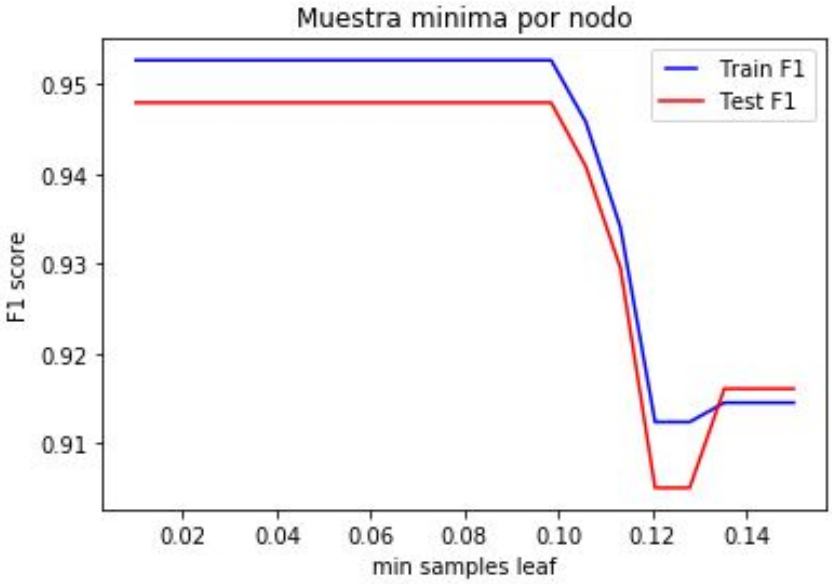

Figure 79

Se encontro que el mejor valor para min samples leaf=0.09842105

10.6 Resultados

Luego del realizar el tuneo de los hiper-parámetros se obtuvo el score: 0.70671.
Siendo una primera aproximación.

11 CatBoost

11.1 Resumen

El nombre CatBoost proviene de la unión de los términos “Category” y “Boosting”. “Category” hace referencia al hecho de que la librería funciona perfectamente con múltiples categorías de datos, como audio, texto e imagen, incluidos datos históricos. Por otro lado, “Boosting” hace referencia al hecho de que el algoritmo se basa en la técnica en “Gradient boosting” (potenciación del gradiente).
Es una mejora del bossting que puede manejar datos categoricos.

11.2 Motivacion

Quisimos probar Catboost por por los siguientes puntos fuertes que tiene:

- Produce resultados excelentes con conjunto de datos relativamente más pequeños que otros métodos de aprendizaje automático.
- Trabaja automáticamente con características categóricas. Siendo posible usar tanto datos numéricos como categóricos con CatBoost sin realizar ningún procesado previo. CatBoost convierte valores categóricos en números utilizando diversas estadísticas sobre combinaciones de características categóricas y combinaciones de características categóricas y numéricas.
- Requiere configurar pocos hiperparámetros evitando la posibilidad de que aparezca sobreajuste. Obteniendo de esta manera modelos más generalizados.

11.3 Feature Engineering

Se trabajo un set de datos ya preprocesado con los featues de XGBoost.

11.4 Desarrollo

Con este algoritmo no se realizo mucha incapie, se lo probo una o dos veces y al ver que su score no era mejor que los anteriores probados se lo dejo de lado.

11.5 Resultados

Los dos submits subidoas a kaggle fueron con 1600 iterations, 6 depth y dio un score de 0.737.
El otro fue con 2000 iterations y 8 depth, y con unos featues mas colocados (asumo que serian de si aparece tal palabra en el texto) y dio un score de 0.740.

12 XGBoost

12.1 Resumen

Este es un algoritmo de Boosting que genera un árbol para cada iteración, y es considerado como uno de los mejores algoritmos y para nosotros resultó el mejor, hasta que usamos las redes neuronales.
Con XGBoost se estaban obteniendo muy buenos resultados, pero la duración del entrenamiento eran más largas que con KNN, cada entrenamiento del XGB duraban aproximadamente entre 10 segundos a una 200 segundos dependiendo de la cantidad de parámetros utilizados.

12.2 Motivacion

Como ya se menciono, la motivacion de usar este aalgoritmo fue porque es generalmente el de mejor o uno de los mejores resultado en la mayoria de los casos de uso.

12.3 Feature Engineering

En este algoritmo se usaron los mismos features que en los de knn, se agregaron unos mas como la probabilidad de ser 0 o 1 dependiendo de las palabras que se tenian en el tweets o un (cuasi) puntaje , mientras en puntaje es mas alto, mas probabibilidad de ser 0 o 1.
Tambien se agregaron de si aparecen ciertas palabras que parecian de tweets de target 1, o si tenia cierto keyword se le daba un porcentaje de ser 1 o 0 segun los datos historicos del conjunto entero de train.
Ademas, en un principio se uso la negatividad o el context (cosas del TP1), pero no daban muy buen resultado y se los quito. En ciertos momentos se llegaron hasta 600 features (columnas), luego se los fue bajando hasta tener unos 200 maso menos y luego a unos 50.

12.4 Desarrollo

Se comenzó utilizando los valores de estimadores bajos, ya que a medida de que se agrandaban la cantidad de estimadores, se hacia mas larga la espera del entrenamiento del mismo. Este árbol dentro de todo chico, se tuneo para ver cual era la mejor aproximación de hiperparametros para poder replicarlo en el grande, este proceso se realizó mediante un Grid Search para poder mejorar su precisión.

Tambien se utilizo como features n-gramas de palabras de longitus 1(1 palabra), 2 o 3 con target 0 o 1. Se lo uso bastante, pero aunque mejoraba el score un poco, tambien overfiteaba demasiado al set de train y validacion, y se procedio a quitarlo como feature.

A continuacion un grafico del Feature importance en XGBoost (set con 50 featues masomensos, solo aparecen los importantes)

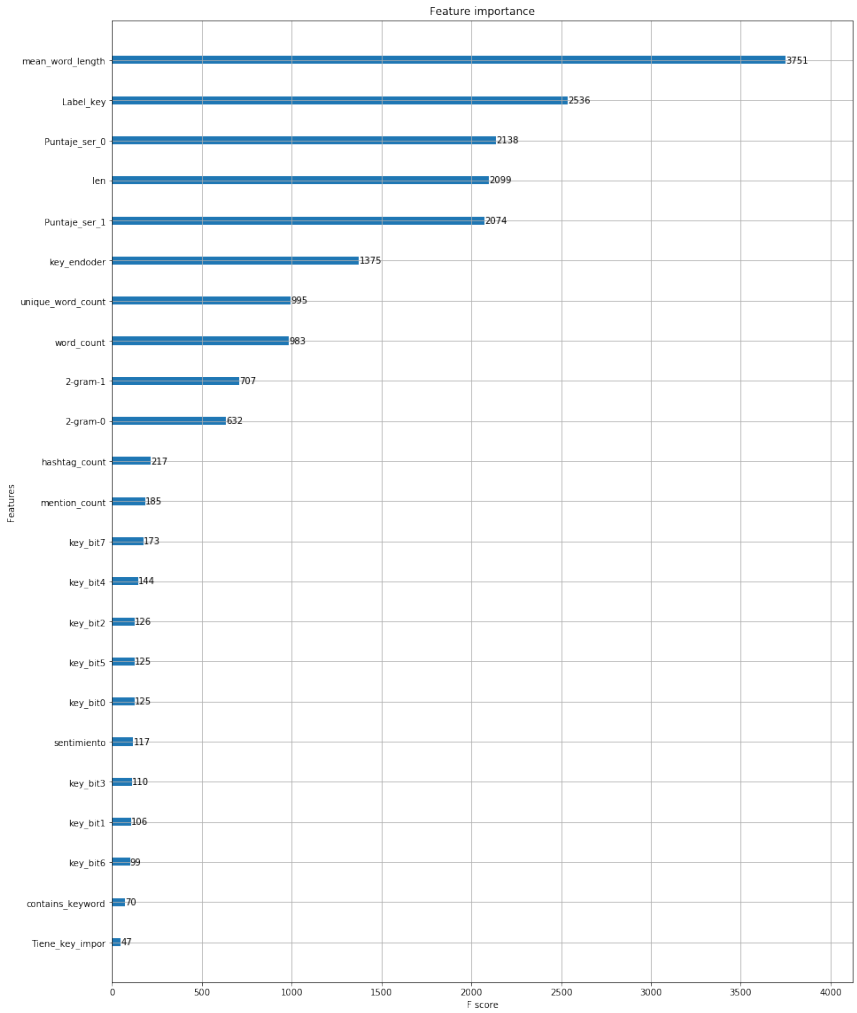

Figure 80: Feature Importance en XGBoost

12.5 Resultados

El mejor submit, fue uno con 200 features masomenos(la mayoria si esta o no tal palabra o keywords) y fue de 0.7998 de F1 score.

13 T5

13.1 Resumen y Motivacion

A muy grandes rasgos en el mundo del deep learning tenemos diferentes "niveles" (si se quiere) de sofisticacion respecto a las arquitecturas de redes neuronales para NLP.

Estos "niveles" se podrian enumerar en orden de la siguiente manera:

- redes neuronales basicas basadas puramente en la idea del Perceptron
- redes neuronales convolucionales.
- redes neuronales recurrentes.
- LSTM (long short term memory)
- redes neuronales transformer

El modelo que se va a probar en esta seccion es uno de estos transformers llamado T5 y constituye el repertorio del estado del arte en redes neuronales para NLP.

Una de las muchas ventajas de usar estos modelos es que existen versiones open source fuertemente preentrenadas sobre muchisimos datos de buena calidad. Lo que se hace en esta seccion es hacer el llamado fine tunning sobre el set de datos de tweets.

13.2 Preprocesamiento

En el caso de T5, y dado que se uso la libreria aportada por Hugging Face, no se realizo mayor preprocesamiento sobre el set de datos mas que mapear los Nones en keywords y location a un string vacio y contatenar keywords, location y text en un campo complete text.

Se procedio luego a utilizar un tokenizador preentrenado que venia con la libreria utilizada para encodear este texto de forma adecuada.

Si se tuvo el cuidado de entrenar a la red con un set de datos balanceado. En otros notebooks incluso se hace un poco de hincapie en elegir la longitud maxima adecuada a usar para el encodeo ya que los transformers trabajan con inputs de longitud fija y mas rapido entrenan cuando mas pequeña es dicha longitud.

13.3 Feature Engineering

Practicamente no hubo. Lo que tiene de interesante las redes neuronales (a diferencia de los arboles de decision) es que pueden recibir texto plano (obviamente encodeado de la forma correcta) y con su "magia" extraer la informacion util.

13.4 Desarrollo

Realmente no hay mucho para destacar. Sinceramente la mayoria del esfuerzo fue dirigido a:

- investigar que NN usar
- entender como funcionan
- buscar ejemplos concretos de codigo
- adaptar el codigo a el caso particular de la materia
- debugear constantemente (y volver a empezar en muchos casos)

Lo que se tiene en el notebook es el resultado de este proceso.

No se realizo tuneo de hiperparametros ni mucho menos cross validation. La razon es porque entrenar estas redes es imposible en forma local y realmente muy costoso en GPU (y la misma esta limitada). La otra razon es que se conocen ya ciertos valores estandar que han dado buenos resultados. Se fue por ese camino.

13.5 Resultados

Con T5 se realizaron 5 submits de resultados:

- T5 submit 1.csv : 0.82224
- T5 submit 2.csv : 0.74961
- T5 submit 3.csv : 0.79865
- submission.csv : 0.79098
- T5 submit 5.csv : 0.62090

los hiperparametros usados para los submits fueron:

submit 1:

```
model_name_or_path='t5-base',
tokenizer_name_or_path='t5-base',
max_seq_length=512,
learning_rate=3e-4,
weight_decay=0.0,
adam_epsilon=1e-8,
warmup_steps=0,
train_batch_size=8,
eval_batch_size=8,
num_train_epochs=2,
gradient_accumulation_steps=16,
```

submit 2:

```
model_name_or_path='t5-base', tokenizer_name_or_path='t5-base',
    max_seq_length=512,
learning_rate=1e-4,
weight_decay=0.01,
adam_epsilon=1e-8,
warmup_steps=1,
train_batch_size=8,
eval_batch_size=8,
num_train_epochs=2,  gradient_accumulation_steps=25,
```

submit 3:

```
model_name_or_path='t5-base', tokenizer_name_or_path='t5-base',
    max_seq_length=max(train.complete_text.apply(len)),
learning_rate=4e-4,
weight_decay=0.00,
adam_epsilon=1e-8,
warmup_steps=3,
train_batch_size=8,
eval_batch_size=8,
num_train_epochs=1,  gradient_accumulation_steps=40,
```

submit 5:

```
model_name_or_path='t5-base',
tokenizer_name_or_path='t5-base',
max_seq_length= max(train.complete_text.apply(len)), learning_rate=2e-5,
weight_decay=0.00,
adam_epsilon=1e-8,
warmup_steps=1,
train_batch_size=8,
eval_batch_size=8,
num_train_epochs=2,
gradient_accumulation_steps=15,

con data limpia
```

No pudimos encontrar los hiperparametros usados en el cuarto submit. Por suerte no fue de los mejores.

Mas alla de esto no hay tanto mas para mostrar. Hay muchas cosas de este modelo que son mas bien de caja negra.

Para no ser repetitivos hay algunas cosas de las que se hablaron aca que se muestran graficamente en la seccion de BERT mas adelante pero la idea es la misma.

13.6 Conclusiones

Sabemos que los transformers en general tienen unos scores de entre 0.85 y 0.90. Ademas sabemos que tienen un buen transfered learning. Que no se haya podido expresar esto aca claramente se debe a que hubo algo que nos falto contemplar. Ese algo podria ser los hiperparametros, la arquitectura especifica usada, la funcion de loss, el optimizador, etc.

14 BERT

14.1 Resumen

Se explicara ahora el uso que se le dio a la red neuronal transformer del momento para NLP (sin contar a la bestia de GPT-3). Ademas se mostrara todo el camino y diferentes estrategias que se probaron. Esta seccion es medianamente extensa debido a que aca se concentro bastante de nuestro esfuerzo.

14.2 Motivacion

Mucho de lo que hay para decir acerca de BERT ya fue dicho en la seccion de T5. BERT es otra de estas redes enormes pre entrenadas que constituyen el estado del arte. Esta (y una inspeccion sobre los notebooks de kaggle) fue suficiente motivacion para usarlo.

14.3 Preprocesamiento

En esta linea se siguieron dos caminos:

- Jugar con el desbalanceo/balanceo de los data sets
- Estratificar por los keywords

14.3.1 Encoding

Sobre estas dos estrategias se aplico un encoding provisto por la libreria de BERT usada

```
https://tfhub.dev/tensorflow/bert_en_uncased_L-24_H-1024_A-16/2"
```

Como el tokenizer/encoder de BERT requiere tener texto de longitud fija se tuvo que elegir que longitud usar. El padding no fue un problema.

El problema que si nos hizo analizar un poco la situacion es la siguiente: Si tomamos como longitud fija la longitud del tweet mas largo vamos a tener bastante padding en todos los tweets.

Esto es malo por dos razones. Una porque cuando mas largo es el encodeo mas tarda el algoritmo en aprender dado que hay mas neuronas y mas pesos que ajustar. La segunda razon es que al haber mucho padding, este puede funcionar como un ruido que en promedio atenua la señal (el significado, el patron, etc) que el modelo quiere captar.

Por esta razon analizamos la opcion de truncar los tweets en un valor tal que la perdida de characters y de palabras no sea tanta.

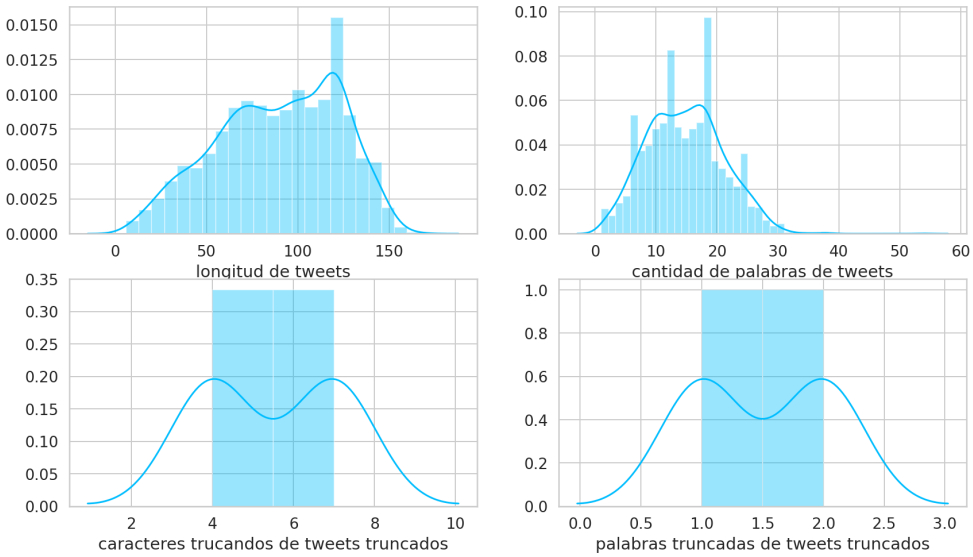

Figure 81

```
porcentaje de tweets que quedan cortados: 0.03% con MAX_LEN = 160
porcentaje de palabras que se pierden al truncar: 0.09% con MAX_LEN = 160
```

Figure 82

Nos decidimos por la longitud maxima de 160 caracteres.

(Como otro dato a contar, usamos BERT uncased por lo que pasamos todas las palabras a minuscula)

14.3.2 Balanceo y Desbalanceo de los datasets

Se arranco de la forma mas tradicional optando por no sesgar al algoritmo y decidimos subsamplear la clase mayoritaria.

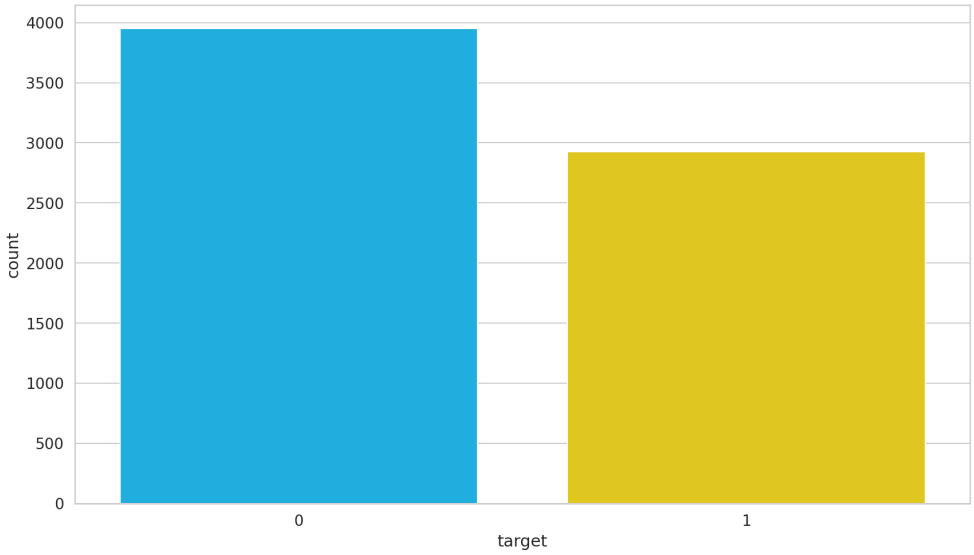

Figure 83

Se puede ver que esta es la formada por los ceros. Lo que se hizo entonces fue tomar una cantidad de ceros tal que sea la misma cantidad que unos para el train, pero de forma tal que en el set de validacion queden ceros y unos para probar.

Con esta estrategia se probaron dos submits (el 1 y el 2 de BERT).

El resultado era bueno, pero parecia haber un problema con la clase del cero. Por alguna razon habia menos precision en el cero y menos recall en el uno. Parecia como si al modelo le costara entender el cero. Pensamos que quizas esto se debia a que el cero era "mas dificil" de entender.

Esto nos motivo a probar clases desbalanceadas. Primero con el cero y luego a favor del uno. Los resultados hicieron que se overfiteara el set de validacion y empeoraron el de test. Esto se puede ver en los submits 3 y 4 de BERT donde tambien se empieza a usar K fold CV pero con folds random.

14.3.3 Estratificacion por keywords

Tal como se dice aca https://www.kaggle.com/gunesevitan/nlp-with-disaster-tweets-eda-cleaning-and-bert parece que la data se agrupa por keywords de forma bastante pareja. Nos pusimos a analizar esto.

```
print(f"keywords: {len(' '.join(data.keyword).split())}")
print(f"keywords unicas: {len(set(' '.join(data.keyword).split()))}")
```

```
keywords: 7552
keywords unicas: 221
```

```
data.keyword.apply(lambda x: len(x.split())).value_counts()
```

```
1    7552
0      61
Name: keyword, dtype: int64
```

Figure 84

Hay 7552 tweets con una sola keyword y 61 sin ninguna. Esto es conveniente porque entonces podemos usar un groupby para agrupar los tipos de tweets por keyword.

```
tweet_types = data.groupby('keyword').target.count()
tweet_types
```

```
keyword
                          61
ablaze                    36
accident                  35
aftershock                34
airplane%20accident       35
                          ..
wounded                   37
wounds                    33
wreck                     37
wreckage                  39
wrecked                   39
Name: target, Length: 222, dtype: int64
```

Figure 85

```
tweet_types.describe()
```

```
count    222.000000
mean      34.292793
std        5.103441
min        9.000000
25%       33.000000
50%       35.000000
75%       37.000000
max       61.000000
Name: target, dtype: float64
```

Figure 86

Cada clase de tweets esta bastante balanceada.

```
tweet_types_1 = data[data.target == 1].groupby('keyword').target.count()
tweet_types_0 = data[data.target == 0].groupby('keyword').target.count()
```

+ Code + Text

```
tweet_types_1.describe()
```

```
count    221.000000
mean      14.800905
std        9.820026
min        1.000000
25%        6.000000
50%       13.000000
75%       23.000000
max       42.000000
Name: target, dtype: float64
```

```
tweet_types_0.describe()
```

```
count    219.000000
mean      19.826484
std        9.465875
min        1.000000
25%       12.000000
50%       20.000000
75%       28.000000
max       40.000000
Name: target, dtype: float64
```

Figure 87

La distribucion de las clases dividiendolas por target tambien es bastante parecida entre ellas y en su homologo de target.

Todo esto nos habilita para seguir con esta intencion de hacer cross validation con k folds donde cada fold podria ser una agrupacion de estas clases.

La idea es que cada BERT aportaria una mirada distinta. Se estaria aplicando un Mayority Vote de BERTs ponderado por el F1 score.

Claramente no vamos a usar un BERT por cada keyword. La razon es doble. No queremos entrenar 221 BERTs porque es una locura y aun si estuviesemos lo suficientemente locos como para hacerlo, la cantidad de tweets en cada una de esas 221 clases es muy chica como para poder fine tunear un modelo tan grande y complejo como lo es BERT.

```
[ ]   set(to_predict.groupby('keyword').id.count().index)\
      .difference(
          set(data.groupby('keyword').id.count().index))
```

```
set()
```

Figure 88

Las categorias de tweets para el set de train y test son exactamente las mismas.

Dado que vamos a hacer k fold, pero viendo que hay ciertos temas comunes entre keywords. Por ejempl tenemos cosas como 'demolished', 'demolish' o 'wound' e 'injury' que son en el fondo lo mismo.

Como no queremos mezclar temas, los agrupamos. Esto generaliza bien al set de test porque comparten exactamente las mismas keywords.

Se realizaron otros dos submits (el 5 y 6 de BERT). El 5 fue nuestro mejor resultado. El 6 deja mucho que desear. En el probamos lo mismo que en el 5 pero cambiando el threshold para el cual considerabamos a una probabilidad de ser 1. Es decir deciamos que un tweet era 1 si su probabilidad era mayor a 0.15 en vez del tradicional 0.5 que veniamos usando. Esto daba bien en el set de validacion pero mal en el de test (otra vez)

14.4 Desarrollo

En esta seccion se da una vista de pajaro de las cosas que se hicieron de cara a la construccion de los modelos y de como se los entreno.

Decimos 'vista de pajaro' porque estuvimos literalmente dias probando variantes y fine tuneando hiperparametros.

Hicimos solo 7 submits con BERT pero muchas muchas mas fueron las corridas que le dimos a BERT.

Menciones honorables sobre cosas que probamos incluyen:

- optimizadores Adam y SGD
- diferentes funciones de loss como binary cross entropy y categorical cross entropy
- diferentes valores de learning rate
- agregarle algunas capas densas antes de la salida de la red
- Cambiar la funcion de activacion de salida entre sigmoide y softmax
- probamos con diferentes proporciones de set de validacion y train
- diferentes tamaños de batch (por temas de memoria y para que el algoritmo no sea tan estocastico por la estabilidad en el aprendizaje)

14.4.1 Red neuronal "tradicional"

Antes de seguir con BERT tambien queremos mencionar que probamos usar el enfoque mas simple de armar una arquitectura de red profunda mas comun con la API provista por Keras.

Probamos diferentes combinaciones de capas:

- Capas densas (obviamente)
- Conv-1D
- Capas de embedding (estas anduvieron bastante bien)
- Capas LSTM (que tienen mucho sentido en el contexto NLP)

Nos fue mejor con BERT.

14.4.2 Fiteo de BERT

Al principio se probo con simplemente entrenar a BERT con train y validar con el set de validacion. Esto no anduvo mal, pero vimos en kaggle la opcion de usar K fold cross validation y quisimos probarlo.

En rigor lo que hicimos no es realmente Cross validation pero si es un K fold.

Partimos el set de train y val en K partes y generamos K modelos BERT. Para cada uno de esos K modelos realizamos P epochs entrenandolos con su fold de train correspondiente (de aca que decimos que en rigor no es cross validation ya que no estamos entrenando con todo train) y validandolo con su fold de val correspondiente.

Antes de pasar al siguiente modelo con el siguiente par de folds, le dabamos una epoch al modelo de todo train. La intencion de esto es para que cada modelo se especialise en su fold pero tenga una nocion global de todo el dataset y asi evitar un poco el overfit.

Al final de la etapa de entrenamiento (y tal como se comento en otras secciones), se predecia usando majority vote ponderando por el score que habia tenido el modelo durante el entrenamiento.

Este tipo de entrenamiento se realizo tomando tanto folds aleatorias como folds estratificadas por agrupaciones semanticas de keywords.

14.5 Resultados

Con T5 se realizaron 5 submits de resultados:

- BERT keras 1.csv : 0.83113
- BERT keras 2.csv : 0.80845
- BERT keras 3.csv : 0.79098
- BERT keras 4.csv : 0.81887
- BERT keras 5.csv : 0.83420 (mejor score nuestro)
- BERT keras 6.csv : 0.74103
- BERT keras 7.csv : 0.80447

los hiperparametros usados para los submits fueron: informe y todo el repo puede encontrarse en el link de mas abajo y este informe fue hecho con Overleaf en latex.≠
submit 1:
Se usa el texto sin limpiar y sin K fold.

submit 2:
texto limpio y lr = 0.8 e**-6

submit 3:
con kfold K = 5, batch = 4, epochs = 4, 4 modelos con mayority vote

submit 4:
Desbalancea el set de datos al de su distribucion de targets original. Sigue usando Kfold 2 epochs con subset de fold y una epoch con todo el set. Batch = 2. 4 modelos y mayority vote. Analiza y mejora el truncamiento de los twets para no perder tantas palabras. Elimina los tags y los remplaza por white spaces.

submit 5:
Agrupa keywords por temas en tres folds. aumenta la relacion de 0 con respecto a 1 (0.3 valsize en 1 y 0.1 val size en 0). trabaja solo con el texto sin location ni keyword. batch = 4, epochs = 2 con 3 folds y cada una ve al final todo el set completo.

submit 6:
Mismo que el intento 5 pero con un threshold de mapeo de probabilidad a partir del 0.15 es 1 y si no es 0.

submit 7:
Entrena otros 3 BERTs mas bajo las mismas condiciones que el intento 6 (son 3 modelos

+ el del intento 6 con un peso de 0.8).

14.6 Conclusion

BERT es genial y si bien seguro es la forma correcta de modelar el problema, creemos que lo agotamos todo lo que podiamos. Quizas podriamos probar tocando hiperparametros como factores de regularizacion o usando el parametro momentum en el SGD o agregando batch normalization para mejorar el descenso de gradiente.

Fuera de estas potenciales mejoras no seria mala idea considerar seguir por el camino de una red neuronal construida desde cero. Esto hace que sea mas simple y mas flexible de cambiar que un modelo de caja negra como es BERT.

15 Cosas que no llegamos a probar

15.1 HDBScan

Quisimos usar HDBScan como algoritmo de clustering para los tweets porque nos parecio de las clases que era un buen algoritmo de clustering para usar en comparacion con las demas opciones.

La idea era generar clusters y poner a cada tweet su numero de cluster como feature para usar luego en un XGBoost por ejemplo.

15.2 SVD

En la misma linea que lo dicho sobre HDBScan tambien se quiso hacer una SVD sobre los tweets para usar como feature las coordenadas de los tweets en funcion de la base vectorial de conceptos latentes en el corpus.

15.3 Boostings de XGboosts

Se nos ocurrio para hacer un boosting de XGBoost. La idea era predecir con un XGBoost y esa prediccion guardarla como feature y predecir denuevo y producit otra columna y predecir ahora con las dos columnas y generar una tercera y asi.....

15.4 Ensambles

Nos quedamos tambien cortos de tiempo para usar mas ensambles. Es cierto que XGBoost es un boosting y que Random Forest es un majority vote, pero nos falto hacer ensambles entre por ejemplo BERT y XGBoost.

16 Conclusión

16.1 Conclusiones del TP

Fue muy interesante y enriquecedor. Se aprendio muchisimo en el trayecto de querer mejorar las posiciones en el leadeboard.

El problema presentado por la competencia nos parecio que ya es un problema semi resuelto en el mundo del machine learning y que parece que el mejor modelo esta en los transformers (aunque es cierto que estas ultimas semanas salio un modelo aun mejor para NLP).

Aun asi, por la complejidad implicita de las redes neuronales, la alternativa de usar otros algoritmos como XGBoost es una buena alternativa.

17 Bibliografía y Referencias

- Organización de Datos. Apunte del Curso. October 19, 2018v2.0
- Pandas Cookbook: Recipes for Scientific Computing, Time Series Analysis and Data Visualization Using Python Book by Theodore Petrou v1.0.1