

Examen Final Inteligencia Artificial



ALFONSO VILLALOBOS
M.I.C NORMANDO ALI ZUBIA HERNÁNDEZ

JUEVES 30 DE MAYO DEL 2019

ÍNDICE

- Estrategia de recolección de comentarios
- Etapa de análisis y pre-procesamiento de datos
- Etapa de entrenamiento y creación de algoritmo para análisis de sentimientos
- Uso de algoritmo
- Sección de resultados
- Conclusiones y trabajo futuro

Estrategia de recolección de comentarios

Para esta parte se utilizó para los primeros comentarios, la recolección a mano, con la que se buscó por el nombre específico de amlo, así como por # relacionados con el mismo gobierno de Andrés Manuel.

En el paso de obtener los tweets de nuestra fase de presentación de datos, debido a que se debía juntar más metadata y no contaba con el tiempo necesario decidí buscar alguna alternativa y descubrí que twitter tiene un api con la que se puede extraer de una manera más rápida los datos que requería. Debido a que me equivoqué con la aplicación para obtener el api key, se tardaron mucho en darme el aprobé y Andrea encontró una app donde te daba todos los datos que requeríamos llamada TAGS. Esta aplicación nos permitía agregar las frases que quisiéramos poner la cantidad de datos que querías, filtro de seguidores sobre usuarios, por si querías puros tweets de famosos, y nos regresaba muchísima información.

El id del tweet, nombre de usuario y id del mismo, geolocalización del tweet y localización del usuario, si se hizo respondiendo algún tweet, el estado de la respuesta, fecha de creación tanto en utc como en la que el usuario la creó en su timezone, lenguaje del usuario, seguidores y amigos del usuario, lo que nos permitirá que a la hora de que entrenemos nuestro modelo, obtengamos mucha más información.

Etapas de análisis y pre-procesamiento de datos.

<https://boards.las.leagueoflegends.com/es/c/charlas-generales/mmkIV1LU-lista-de-malas-palabras-en-lol-latinoamerica>

En esta parte primero cargamos nuestro df que juntamos, le quitamos el id debido a que como pegamos todos los comentarios era irrelevante y cambiamos todos los datos de positivo y negativo por un tiny int que si era positivo ponía 1 y negativo 0, después procedimos a remover las malas palabras.

Para esto busqué una lista de malas palabras en internet y me encontré en una página, las palabras baneadas del servidor de lol Latinoamérica las puse en un txt, las cargué abriendo el archivo, poniéndolo en un string, luego pasar todo a lowercase y hacerle split para poderlo cargar en un objeto tipo set() y procesarlo como las stopwords.

```
'''
Aqui se carga nuestro txt con las
malas palabras y se pone en un
objeto set como las stopwords
'''
with open('badwords.txt', 'r') as file:
    malas_palabras = (file.read().replace('\n', '')).lower().split(',')
    badwords = set(malas_palabras)
```

Ya una vez teniendo listos los datos necesarios para comenzar el pre-procesamiento pasamos a hacerlo, primero aplicamos una lambda a todos los tweets donde aplicamos lowercase y luego a tokenizar, luego agarramos el df tokenizado y pasamos a utilizar nuestros objetos set para quitar las stopwords y malas palabras.

```
def remove_badword_stopwords(log):  
    """  
    Funcion encargada de remover  
    las malas palabras  
    """  
    log_remove = []  
    for word in log:  
        if word in badwords:  
            log_remove.append('MALAPALABRA')  
        elif word not in stop_words:  
            log_remove.append(word)  
    return log_remove  
  
tweets_df['log'] = tweets_df.apply(lambda tweets_df: nltk.word_tokenize(tweets_df['log'].lower()), axis=1)  
tweets_df['log'] = tweets_df['log'].apply(lambda log: remove_badword_stopwords(log))
```

Etapas de entrenamiento y creación de algoritmo para análisis de sentimientos

Para poder generar nuestro modelo. Primero vectorizamos el df, este paso es necesario para que se transforme nuestro arreglo log por una matriz de tokens y lo pusimos en nuestros features, después agarramos nuestra clase objetivo y extraímos los valores para, sacar nuestras matrices de testeo y entrenamiento.

```
"""  
vectorize df  
"""  
vectorize = CountVectorizer(lowercase=False, tokenizer=(lambda arg: arg), preprocessor=None)  
data_features = vectorize.fit_transform(tweets_df['log'])  
# print(vectorize.vocabulary_)  
  
"""  
Train test split  
"""  
data_labels = tweets_df['positivo_negativo'].values  
  
X_train, X_test, Y_train, Y_test = train_test_split(data_features, data_labels, test_size=0.3)  
  
"""  
Using MultinomialNB model  
"""  
clf = MultinomialNB()  
clf.fit(X_train, Y_train)  
  
score = clf.score(X_test, Y_test)  
print(score)
```

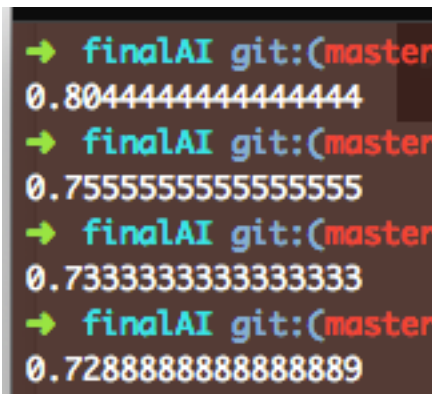
En esta parte ya teníamos nuestro modelo y comenzamos con los scores, el primero nos dio 76, después de varias iteraciones el modelo variaba entre 69 a 76, así que decidí pre-procesar mas los datos, agregando stemming y quitar los números.

```
stemmer = SnowballStemmer('spanish')

def preprocesing(log):
    """
    Funcion encargada de remover
    las malas palabras
    """
    log_remove = []
    for word in log:
        if word in badwords:
            log_remove.append('MALAPALABRA')
        elif word not in stop_words:
            # log_remove.append(stemmer.stem(word))
            log_remove.append(word)
    return log_remove

tweets_df['log'] = tweets_df.apply(lambda tweets_df: nltk.word_tokenize(tweets_df['log'].lower()), axis=1)
tweets_df['log'] = tweets_df['log'].apply(lambda log: preprocesing(log))
tweets_df['log'] = tweets_df['log'].apply(lambda log: [word for word in log if not word.isdigit()])
```

Nuestro modelo mejoro al inicio para ir buscando una manera de mejorar el score aumentamos el tamaño del testeo de 0.3 a 0.5 que bajo mucho el output a 60 con 4 no mejoro y en 2 también bajo del promedio que teníamos así que lo dejamos en 3.



```
→ finalAI git:(master) 0.8044444444444444
→ finalAI git:(master) 0.7555555555555555
→ finalAI git:(master) 0.7333333333333333
→ finalAI git:(master) 0.7288888888888889
```

Luego buscamos tratamos mejorarlo quitando el stemming lo que nos dejo el mismo resultado, para que la informacion fuera mas compacta decidimos dejarlo, aun sabiendo que no mejoraba el resultado.

Viendo el vocabulario encontré un error que tiene el steemer, este en realidad no deja la palabra base si no que corta la palabra a la menor posible, esto significa que en lugar de poner posibilidad pone posib y posible también lo

corta a posib lo que te deja con errores a la hora de errores ortográficos generando palabras nuevas en lugar de colocarlas en la palabra base.

Uso de algoritmo

Para seleccionar nuestro algoritmo, después de una investigación nos dimos cuenta que el algoritmo mas utilizado para procesamiento de lenguaje natural es el de naive bayes debido a que la librería de sklearn es la que utilizamos en clase procedi a leer la documentación de opciones que tenia la misma librería sobre modelos que utilizan este algoritmo.

```

→ finalAI git:(master) 0.7155555555555555
→ finalAI git:(master) 0.7155555555555555
→ finalAI git:(master) 0.6933333333333334
→ finalAI git:(master) 0.72
→ finalAI git:(master) 0.6711111111111111

```

Primero tenemos el gauseano que tuvimos que pasar nuestro vector a un arreglo para que se pudiera procesar, nos daba un score entre 72 y 65 por lo que nos dimos cuenta que no era el mas optimo para nuestro problema, el multinominal aunque se utiliza específicamente para tener mas de 2 objetivos funcionaba bastante mejor que el gauseano, con un scores entre 70 a 80, el siguiente que se probó fue el de complementNB que es lo mismo que el multinominal pero funciona mejor cuando el df no esta balanceado debido a que el nuestro lo esta no se utilizó aunque se probó y tuvo resultados muy similares al multinominal.

Para el final se probó el de Bernoulli, este me llamo mucho la atención porque en la documentación comentaba que el de multinominal tenia problemas con features nuevos, que era un poco preocupante a la hora de predecir el nuevo df que extrajimos de tags. Tambien hablaban de que era mejor con documentos mas cortos y por el limite de caracteres de los tweets pensé que seria el mejor pero al final funciono igual que el multinominal, en ciertos casos pero, busque aumentar el tamaño de datos de testeo que lo ayudo un poco pero aun asi no quise arriesgarme con el debido a que no veía una mejora considerable me daba un score de entre 65 y 75.

```

→ finalAI git:(master) 0.7533333333333333
→ finalAI git:(master) 0.7233333333333334
→ finalAI git:(master) 0.6766666666666666
→ finalAI git:(master) 0.6586666666666666
→ finalAI git:(master) 0.6977777777777778
→ finalAI git:(master)

```

Al final se utilizó tamaño de testeo de 3 y el multinominal, aunque se tomo en cuenta utilizar tanto el multinominal como el de Bernoulli con diferentes splits, para ver la mejora, pero no vimos el caso de complicarlo mas.

```

→ finalAI git:(master) 0.7733333333333333
→ finalAI git:(master) 0.7377777777777778
→ finalAI git:(master) 0.7155555555555555
→ finalAI git:(master) 0.7333333333333333
→ finalAI git:(master) 0.7155555555555555
→ finalAI git:(master) 0.7955555555555556
→ finalAI git:(master)

```

La primera imagen es el gauseano que nos dio escores bajos y muy variados luego el bernolli que no supero la expectativa que genero.

Aquí al final esta el multinominal que fue con el que se inicio, que es muy estable, no baja del .71 y sube hasta 80 en precisión en siertos casos.

Sección de resultados

Aquí se tuvo que procesar los datos que se iban a predecir de la misma manera que se procesaron los de testeo y entrenamiento, batallé con la parte de `fit_transform`, no recordaba que hacía y ya leyendo la documentación me di cuenta que hace un vocabulario y regresa un `term_document`, que es el que se utiliza para entrenar a nuestro modelo, pero existe una opción que solo para los datos que le mandas a una matriz que era lo que en realidad necesitaba.

Después de esto utilice el modelo para predecir, los tweets del otro df que tiene mas metadata, borrando valores que no utilizaríamos en el procesamiento de resultados. Quitamos los que eran únicos, tenían demasiados valores faltantes, o algunos que no le podíamos sacar información valiosa.

```
tweets_df_moreinfo = tweets_df_moreinfo.drop(['geo_coordinates', 'id_str', 'profile_image_url', 'from_user', \
                                              'in_reply_to_user_id_str', 'in_reply_to_screen_name', 'in_reply_to_status_id_str', \
                                              'source', 'user_friends_count', 'entities_str'], axis=1)

tweets_df_moreinfo['user_lang'] = tweets_df_moreinfo['user_lang'].apply(lambda row: row if row == 'es' else None)

tweets_df_moreinfo = tweets_df_moreinfo.dropna(subset=['log', 'user_lang'])

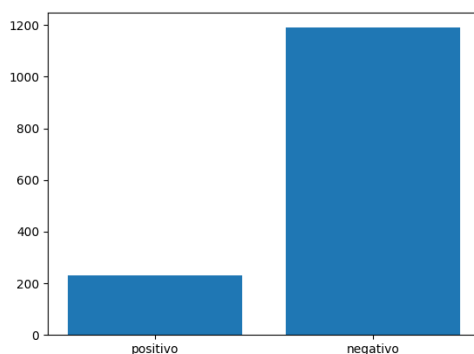
print(tweets_df_moreinfo.head(20), '\n')
print(tweets_df_moreinfo.describe(), '\n')
print(tweets_df_moreinfo.apply(lambda x: x.isnull().any()), '\n')
print(pd.DataFrame({'percent_missing': tweets_df_moreinfo.isnull().sum() * 100 / len(tweets_df_moreinfo)}), '\n')
print(pd.DataFrame({'percent_unique': tweets_df_moreinfo.apply(lambda log: log.unique().size/log.size*100)}), '\n')

tweets_df_moreinfo['log_proces'] = tweets_df_moreinfo.apply(lambda tweets_df_moreinfo: nltk.word_tokenize(tweets_df_moreinfo['log'].lower()), axis=1)
tweets_df_moreinfo['log_proces'] = tweets_df_moreinfo['log_proces'].apply(lambda log: preprocessing(log))
tweets_df_moreinfo['log_proces'] = tweets_df_moreinfo['log_proces'].apply(lambda log: [word for word in log if not word.isdigit()])

'''
Se vectoriza para poder
procesarlo
'''
data_features_2 = vectorize.transform(tweets_df_moreinfo['log_proces'])
# print(vectorize_2.vocabulary_)
'''
Se predice y se guarda la informacion
en un csv para despues obtener mas
informacion de ese df
'''

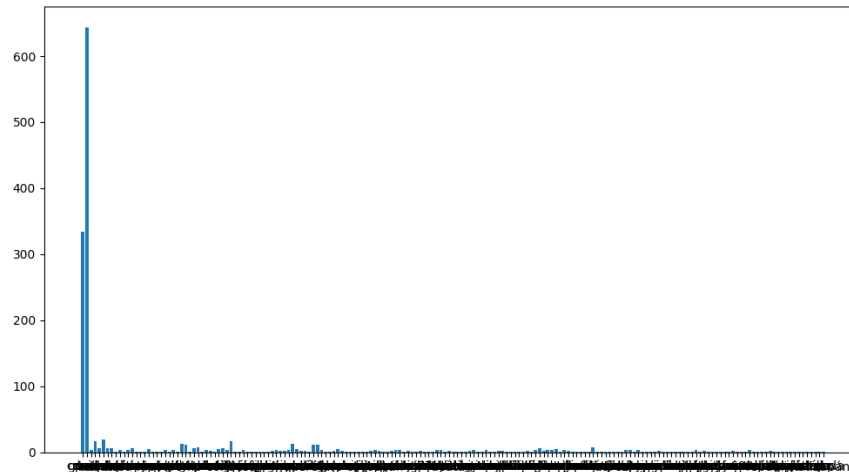
# print(X_train, '\n')
# print(data_features_2, '\n')
predictions = clf.predict(data_features_2)
tweets_df_moreinfo = tweets_df_moreinfo.drop(['user_lang', 'log_proces'], axis=1)
tweets_df_moreinfo['positivo_negativo'] = predictions
# print(data_frame2.head(20), '\n')
tweets_df_moreinfo.to_csv(path_or_buf='newdf.csv')
```

Después de esto hicimos otro script para generar las graficas de resultados, la primera grafica



que intente hice fue la de hacer una relación entre `user_location` y positivo negativo, a la hora de ver los valores únicos de la localidad, me di cuenta que hay valores que se repiten pero que se escriben diferente así que tuvimos que cambiar, tratar de hacerlos lo menos único posible, entre eso, saque la cantidad de positivos y negativos.

Con esto se me ocurrió sumar todos los valores de las zonas únicas que existían, dándole un sort obtuve los siguientes resultados. Lo que me permitió saber que el dato de user_location



tenia demasiados datos vacíos y o outliers lo que nos dejaba con información inútil.

Para el siguiente y ultimo paso procedí a investigar sobre los comentarios negativos y positivos en base al tiempo sobre el gobierno de amlo, por lo que pase a ver las fechas en utc, este valor estaba en un formato que con hacer Split por espacio separábamos la fecha en un arreglo ['dia de la semana', 'mes', 'dia numero', 'time zone', 'año']

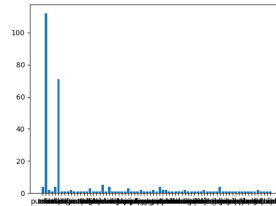
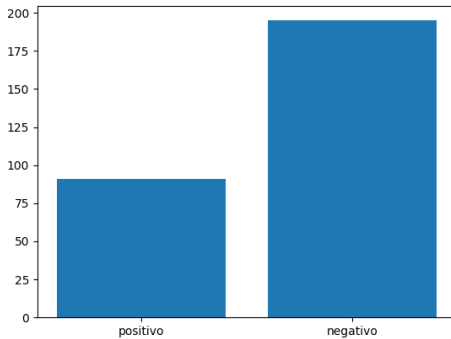
```
tweets_df.created_at = tweets_df.created_at.apply(lambda date: date.split(' '))
tweets_df['year'] = tweets_df.created_at.apply(lambda date: date[-1])
tweets_df = tweets_df.drop(['created_at'], axis=1)

# print(tweets_df['year'].head(20))
print(pd.DataFrame({'percent_unique': tweets_df.apply(lambda log: log.unique().size/log.size*100)}), '\n')
print(tweets_df.year.unique(), '\n')
```

Aplicando estos códigos nos dimos cuenta de que el año era siempre 2019 luego pasamos a ver por mes y al igual que la localidad ver que tantos outliers tenia nuestro con esto nos dimos cuenta que todo nuestro dataframe era del mismo día y lo único que variaba eran las horas

valores de las zonas únicas que me permitió saber que el dato	percent_unique
log	52.076003
time	76.917664
user_followers_count	48.838846
user_location	30.049261
positivo_negativo	0.140746
mes	0.070373
['25']	

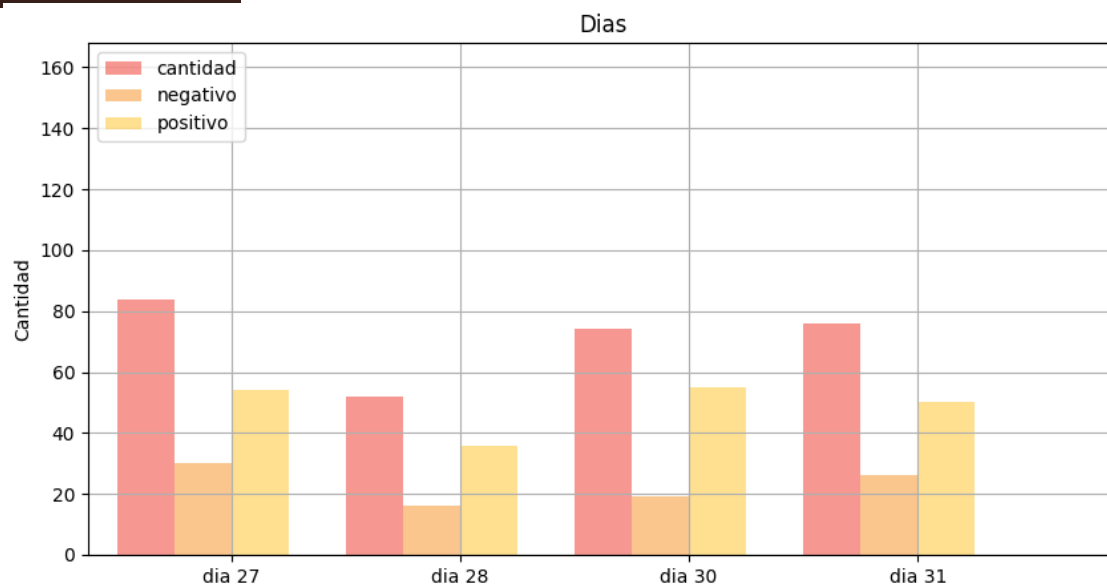
Por esto procedí a predecir un nuevo df de 365 datos positivo negativo salió, 91 positivo y 195 negativos, después volvimos a hacer el intento con la localización sacando la información exacta que tiraron todas las ciudades y haciendo su respectiva grafica.



```
{
  "dia 27": {
    "cantidad": 84,
    "negativo": 54,
    "positivo": 30
  },
  "dia 28": {
    "cantidad": 52,
    "negativo": 36,
    "positivo": 16
  },
  "dia 30": {
    "cantidad": 74,
    "negativo": 55,
    "positivo": 19
  },
  "dia 31": {
    "cantidad": 76,
    "negativo": 50,
    "positivo": 26
  }
}
```

Nos dio resultados similares al anterior con muchísimos outliers dando pura información inutil, después debido a que generamos este df para tener mas variación de fechas procedimos a generar graficas de los días que tuvimos, fueron 4 27, 28, 30 y 31.

lo que nos deja con un data frame bastante variado, despues se genero un json, con los datos positivos y negativos de cada dia contando tambien la totalidad de valore, al final se graficaron los resultados agrupados por dias.



Conclusiones y trabajo futuro

Este proyecto genera información muy valiosa si se consigue un buen dataframe, debido a que puedes sacar y vender información para estrategias en campañas políticas. Si se procesa adecuadamente los datos de las ubicaciones, puedes saber en que estados, el político tiene mas pucha, donde tiene que trabajar mas, que sectores, están involucrados en las redes sociales, comentarios de otros países sobre el mismo.

También funciona para saber que opina la gente con influencia sobre el mismo político. Se podría ampliar muchísimo el alcance de este si se extrae información de comentarios en Facebook, debido a que mas gente lo utiliza.

Yo opino que es una herramienta que tiene que pulirse muchísimo, pero se puede vender bastante cara si se obtiene un resultado usable para algún usuario final, donde el mismo usuario por medio de una web app intuitiva pueda buscar palabras clave y obtener estadísticas sobre su campaña.