UNIVERSITAT DE BARCELONA

Trabajo de Final de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

DETECTOR DE SARCASMO PARA TELEGRAM

Daniel Lopez Roman

Director: ORIOL PUJOL VILA
Realizado en: Departamento de
Matemáticas e Informática

Barcelona, 20 de junio de 2020

ÍNDICE

INTRODUCCIÓN

El sarcasmo es algo que siempre me ha parecido interesante. Las personas nos entendemos al hablar mediante tonos y gestos que añaden más profundidad o sentidos a lo que decimos. El sarcasmo se basa mucho en estos extras añadidos al lenguaje, o esa es mi percepción. Muchas veces puedes saber si alguien esta siendo sarcastico por lo que sabemos de él, o porque sabemos cómo se expresa cuando bromea. Pero. ¿como lo haría una maquina para conocer a todo el mundo de esta misma manera? Si el Bot funciona significa que hay elementos del lenguaje que denotan el sarcasmo más que otros, y esto es muy interesante porque hay gente que no percibe el sarcasmo cuando una máquina sí podría (es algo muy recurrente en televisión y películas, ver máquinas incapaces de hacer o identificar una broma). Además de que me fascina el hecho de que las máquinas adopten comportamientos "humanos" o puedan ser más creativas que yo (me pongo de ejemplo a pesar de que no es muy difícil ser más creativo que yo).

Esto nos lleva al otro punto de la motivación del presente proyecto. Utilizo este proyecto para investigar diversas herramientas, aprender formas interesantes de resolver problemas, y descubrir cuales son las formas de proceder al procesar datos; todo ello basándose en el aprendizaje automático.

El diseño de un "Bot detector de sarcasmo" es el catalizador de todas estas ideas. El objetivo del presente proyecto es la realización de un robot que sea capaz de detectar sarcasmo en tiempo real usando la plataforma de Telegram.
Gran parte de las decisiones de este proyecto han sido tomadas entorno a cuanto se de ellas y mi interés por aprender sobre ellas o ver como ayudan al proyecto.

CONCEPTOS PREVIOS A LA MEMÓRIA

En esta sección se recoge un breve glosario de los términos y conceptos más relevantes que se encontrarán a lo largo de la memoria.

Tf-idf: (*Term frequency – Inverse document frequency*) Medida para expresar la relevancia de una palabra. El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos. Esto quita relevancia a palabras muy comunes en varios documentos.

Entrenar/hacer *fit*: Es el proceso en el que se detectan los patrones de un conjunto de datos. Una vez identificados los patrones, se pueden hacer predicciones con nuevos datos que se incorporen al sistema.

Transformar datos: Es el proceso por el que convertimos unos datos en otros. En el contexto de este proyecto se utiliza para convertir datos convencionales como texto a cosas que nuestro modelo pueda entender, como número o etiquetas.

Set/conjunto de datos: (también llamado *dataset*) Es el conjunto de todos los ejemplos que tenemos (todos los datos). Estos se organizan por muestras o filas, y características o columnas (convencionalmente)

Etiquetas: Tipo de dato capaz de adoptar un número de formas limitado, que representa cada una de sus formas con un identificador diferente.

Regresión logística: Tipo de análisis de regresión utilizado para predecir el resultado de una variable que puede adoptar un número limitado de categorías o formas.

Matriz dispersa: Matriz en la que la mayoría de sus valores son 0.

Lematizar: La lematización es un proceso lingüístico que consiste en, dada una forma flexionada, hallar el lema correspondiente (forma no flexionada).

Emoticonos: Secuencia de caracteres que representa una cara, emoción o acción

Aprendizaje automático: subcampo de las ciencias de la computación e inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que las máquinas aprendan.

API: Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece un portal (bajo los términos del creador) a información, para que software de terceros pueda hacer uso de ella.

Predictor/modelo: Filtro que se ajusta a unos datos (entrenamiento) y es capaz de procesar una respuesta para los nuevos datos en base a lo aprendido en el entrenamiento.

"Accuracy" del modelo: (*exactitud*) Métrica para evaluar modelos de clasificación. La exactitud es la fracción de predicciones que el modelo realizó correctamente del total.

Hardcodear: El hecho de incrustar datos directamente en el código fuente del programa.

Reddit: Sitio web de marcadores sociales, y agregador de noticias donde los usuarios pueden publicar texto, imágenes, vídeos o enlaces publicamente, valorar contenido y interactuar con las diversas publicaciones.

Telegram: Plataforma de mensajería enfocada en la mensajería instantánea, el envío de varios archivos y la comunicación en masa.

Telepot: Librería de python que enmascara las peticiones a la API de Telegram en métodos y clases para facilitar su uso.

Bot: Programa informático que efectúa automáticamente tareas a través de Internet.

Praw: (*Python Reddit API Wrapper*) Librería de python que enmascara las peticiones a la API de Reddit en métodos y clases para facilitar su uso.

Pushshift: API que ofrece acceso a información y contenido de Reddit

Dataframe: Objeto de la librería Pandas de Python; que se comporta como una tabla, con columnas y filas etiquetadas, y con potentes modos y herramientas para indexar, reemplazar o trabajar, con los datos que contiene.

Spacy: Librería de Python. Herramienta para el procesamiento de lenguaje natural.

ANÁLISIS

Por parte del bot no pedimos muchas funcionalidades:

- El usuario debe poder interactuar con él enviando un mensaje.
- La interacción es implícita, no necesita mencionarse al bot. el bot cogerá todos los mensajes que vea sin excepción.
- El bot ha de procesar el mensaje recibido y responder si cree que es sarcasmo o no.
- ha de funcionar tanto en conversaciones privadas como en grupos con varios usuarios

El bot necesitará otra parte para hacer la clasificación, lo llamaremos “predictor”.
Lo que pedimos al predictor és:

- Clasificar datos
- Trabajar con texto
- Poder transformar/operar con diferentes tipos de datos
- Responder rápidamente a nuevos ejemplos

Y el predictor necesitará datos para entrenar y empezar a hacer predicciones. Necesitaremos:

- Una fuente de textos sarcásticos y no sarcásticos (pueden ser dos diferentes)
- Una forma de extraer datos de la fuente
- Extraer la información necesaria de las muestras
- Encontrar una forma eficiente de manejar y guardar los datos, ya que esperamos tener muchos

DISEÑO

Este proyecto está realizado en python 3.
Las estructuras de datos utilizadas a lo largo del proyecto son diccionarios y “Dataframe”s, diccionarios por tener propiedades similares a “Dataframe”, y son fácilmente convertibles a estos (menos en ciertos casos que se mencionan más adelante en los que creamos un método que solucione este conflicto). Estos tipos de datos son empleados por su potencia a la hora de trabajar con diferentes tipos de datos almacenados, y las facilidades que aportan para indexar muestras dependiendo del criterio especificado que puede ser muy diverso. Normalmente si se requiere de algún tipo de indexación potente es cuando se hace la conversión de diccionario a “Dataframe”.

ARQUITECTURA SOFTWARE

El proyecto está estructurado en 3 partes: El bot de telegram, herramientas del predictor y creación de los conjuntos de datos. Esta diferenciación se lleva a cabo no sólo a nivel de esquema, sino también a nivel de estructura; donde se ha priorizado definir los 3 bloques mas que separar por funcionalidad las diferentes partes que componen cada uno.

El componente del “bot de telegram” constituye todo lo necesario para poner en marcha un bot de telegram además de cargar el predictor que utilizará para decidir que es o no es sarcasmo.

El “clasificador” comprende diversas partes que pueden aportar información sobre los datos o el clasificador entrenado, y lo necesario para entrenar o transformar datos para que encajen con lo esperado por el clasificador.

El “preparador de datos” contiene diversas herramientas útiles a la hora de extraer, limpiar y organizar datos, y son utilizadas tanto para construir el set de datos como para preparar los nuevos datos para ajustarlos a lo que espera transformar el clasificador.

EXTRACCIÓN DE DATOS

La decisión sobre dónde extraer los datos me decanta por Reddit porque es una pagina con muchos usuarios distintos, temas de conversación muy diversos, y los comentarios tienen un ámbito más “conversacional” que podría tener algo mas enunciativo o informativo como una página de noticias sarcástica; además del hecho de que en reddit se tienen varias conversaciones distintas por una misma publicación, añadiendo aún más variedad a lo que puedes encontrar. Encima de todo esto está el hecho de que existen herramientas de fácil acceso a esta información vía APIs. Esto ultimo planteaba la opcion de utilizar twitter ya que posee herramientas similares, pero la interacción que genera su formato dista del necesitado. La forma de encontrar el sarcasmo en twitter habría sido buscar cuentas sarcasticas o que publiquen contenido de paginas sarcasticas (ej: la cuenta de un diario de noticias sarcásticas), pero en Reddit funciona de otra manera: en Reddit, los usuarios utilizan un marcador “/s” para denotar que su comentario tenía intención sarcástica, hasta encontrado con este delimitador. Este comportamiento no es nativo de la página sino una evolución de algo introducido por usuarios, y aparece similar a como se marca el final de un “tag” en HTML. Encontrar esto será la forma de encontrar sarcasmo en los comentarios.
Como buscar entre todos los millones de comentarios de reddit llevaría demasiado tiempo (hablamos de una pagina con mas de 1.5 billones de comentarios al año), filtraremos la búsqueda utilizando subreddits de habla hispana, y fiarnos de que todo lo que hay esta en español (con la ocasional palabra o comentario en inglés). Los subreddits utilizados fueron encontrados en un post en el propio reddit donde un usuario pedía una lista de subreddits de habla hispana.
Ya con una fuente de comentarios sarcásticos y con el fin de tener un conjunto de datos equilibrado, se complementan los datos con comentarios no sarcásticos de los mismos usuarios, de esta manera por cada comentario sarcástico tenemos uno no sarcástico de la misma fuente, y introducimos formas de hablar más subjetivas de cada usuario en ambos bandos del análisis; la idea es que cada vez que “sumamos” un usuario y su sarcasmo, “restamos” a ese usuario de la ecuación y nos quedamos con el sarcasmo (en cierto modo lo que aporta el usuario podríamos llamarlo su forma de expresarse).

A simple vista parece que “praw” es la mejor opción para captar los comentarios principalmente por tener una adaptación en forma de librería para python, pero tras comprobar su tiempo de respuesta y el escaso control sobre el filtro de comentarios (inexistente), la decisión es utilizar la API “pushshift” y trabajar con JSONs. Pese a que ambas tienen un límite de comentarios por petición, esta última nos permite hacer búsquedas más concretas.

Con la obtención de datos resuelta queda saber como preprocesar el texto para que sea más digerible por el clasificador y extraer información importante.

La extracción de información ha recaído en el análisis de las expresiones no verbales del texto: emoticonos (una lista de ellos puede ser encontrada en wikipedia, es la utilizada en el proyecto).

Los llamados “emojis” (los dibujitos que vienen por defecto en muchas aplicaciones y son propios iconos) aparecen en telegram, pero no los tratamos porque reddit no da ese tipo de soporte, además de no haber encontrado ninguno en los datos, simplemente los descartamos del texto.

La preparación del texto caerá a cargo de extraer sólo las palabras y lematizarlas, puesto que parece ser un muy buen método de mejorar resultados en el procesamiento de lenguaje natural.

EL BOT

El bot utilizará la plataforma de Telegram, ya que tiene API y Telepot da buen acceso a ella con multitud de ejemplos de uso y es fácil crear nuestro bot a partir de esos ejemplos. Además de esta forma podemos interactuar con el bot fácilmente desde el teléfono o ordenador.

Antes de empezar a definir el bot, hay que conseguir uno. Telegram tiene un bot llamado “@botfather” al que podemos enviar mensajes, que después de facilitarle información para nuestro bot como nombre de usuario y nombre público, nos genera un TOKEN secreto con el que accederemos a nuestro bot y le haremos ejecutar nuestros comandos. Además Botfather nos sirve como menú de opciones para nuestro bot.

El bot es una modificación de un proyecto mío anteriormente realizado para la asignatura TNUI (*Talleres de Nuevos Usos de la Informática*), retocado para que sólo capte mensajes de texto, compruebe su contenido y responda, es algo muy básico y se pueden obtener fácilmente ejemplos casi idénticos con una simple búsqueda en internet de ejemplos para el módulo “telepot” en python.

Tiene dos clases:

- SarcasmBot: se encarga de controlar el bucle en el que espera mensajes.
- SarcasmUser: es donde se captan y envían los mensajes, y contiene la lógica para el procesamiento de estos y su respuesta acorde al resultado.

Nuestro toque en esto es que ese procesamiento consiste en la transformación del texto obtenido a algo que el predictor sepa leer y para lo que pueda procesar una respuesta.

Estas son las únicas clases creadas en el proyecto, todas las demás son importadas

EL CLASIFICADOR

El clasificador tiene dos métodos que lo definen:

- modelFitting: dados los datos de entrenamiento, comprueba que columnas contiene (estas columnas siempre tiene y deben tener el mismo tipo de datos) y los pre-procesa, antes de juntarlos todos para crear una matriz dispersa. Esta matriz es la que se entrega a la regresión Logística para ajustarse. Al final de todo se entregan todos estos componentes ya entrenados al usuario en un diccionario, entre ellos la regresión.
- transformData: dado un predictor del tipo obtenido en el anterior método y una set de datos, utiliza los componentes del predictor para transformar las diferentes columnas de datos y compactarlas en una matriz dispersa en el mismo orden que “modelFitting” que se entrega al usuario, lista para ser entregada a la regresión.

Ninguno de estos métodos hace la predicción en sí, pero para ello se espera que se transformen los datos y se envíen a la regresión mediante el método “predict” que pose.

Esta es una de las instancias en las que una clase podría haber unificado los procesos en un método “fit” y otro “predict” pero me ha parecido bien no crear la clase y que estos métodos funcionen como lo hacen para dar más libertad. Es una decisión que he decidido tomar sobre la estructura del código para no restringirlo.

Además, el clasificador tiene dos métodos que aportan información bastante genérica sobre el conjunto de datos como sus dimensiones (población y características), información sobre las características y su distribución en cuanto a respuesta o categoría (sarcástico/no sarcástico) se refiere; y la información del modelo que viene a ser los tests: exactitud de los tests y matriz de confusión.

Otros métodos auxiliares son los utilizados para guardar y cargar datos mediante la librería “pickle” en absolutamente toda situación debido a lo útiles que son, no solo para guardar el modelo sino para guardar listas y diccionarios ya que los guardan y recuperan tal cual era el objeto en ejecución (para Dataframe esta opción ya viene implementada en la librería).

PREPARACIÓN DE DATOS

Las funciones de los métodos de esta sección son: captar información de reddit, arreglar los datos obtenidos para convertirlos en un Dataframe, y la extracción de las características de los comentarios.

El método más importante de los que hay aquí es “downloadFromUrl” que nos consigue los comentarios haciendo peticiones a la api “pushshift”, recoge el json y extrae los comentarios. Para los datos en este proyecto se piden todos los comentarios de subreddits sugeridos, y todos los de X usuarios. Una vez los tiene comprueba el idioma o si contienen una cadena de caracteres concreta. Todo esto hasta el infinito (todas las peticiones), comprueba un número determinado de comentarios, o sigue hasta que encuentra un cierto número de coincidencias.

Métodos que han sido útiles al desarrollar de los conjuntos de datos son:

- mergeDictionaries que convierte una lista de diccionarios en un solo diccionario.
- trimTaggedString que dado un texto y una palabra, elimina todo texto posterior a la palabra (en reddit el tag de sarcasmo delimita la parte del texto que es sarcástica, de la que no).
- discardRepetition que dado un texto elimina cualquier repetición de la palabra especificada (en este caso útil para suprimir exceso de saltos de línea).
- removeAndCountSub elimina del texto la palabra sugerida y luego cuenta cuantas veces aparece en el texto (utilizado para contar emoticonos).
- Lemmatize consigue los lemas de las palabras de un texto mediante la librería “spacy” y ignora los signos de puntuación.

Además, como a mi entender tiene sentido que estén en este contexto, están los métodos que convierten texto en el equivalente a una fila o muestra del conjunto de datos haciendo uso de los métodos antes mencionados (útiles para procesar los comentarios que recibe el bot, antes de transformarlo con el modelo)

Estructuralmente, la única vez que se rompe la continuidad o encapsulación del código es por el hecho de que en el apartado del clasificador existe la herramienta para guardar objetos complejos en memoria, y esto hace que la parte de preparación del dataset necesite importarla, ya que necesita de esta herramienta para no rehacer o repetir el trabajo realizado y perder tanto tiempo en la parte que más tiempo toma.

CODIFICACIÓN

El método expuesto a continuación es el responsable de la obtención de comentarios de Reddit (página siguiente). Puede que no sea el que tiene los componentes más complejos, pero si la estructura más larga y complicada. Además es el método del que más orgulloso estoy; ya que pese a haber utilizado la estructura de un ejemplo de un usuario de Reddit, las peticiones de este estilo no son algo con lo que haya trabajado en python, y suelo mantenerme alejado de las peticiones a web por ese motivo.
El método que entrena el clasificador puede sonar más interesante por sus herramientas más complejas, pero es muy fácil de entender y su dificultad se hallaba en encontrar la herramienta adecuada y los datos para hacerlas funcionar, más que en su uso.

Parametros:

- Pattern: Entidad para la que se buscarán comentarios (ej: “subreddit”, “author”,...)
- Target: Nombre de la entidad “pattern” para la que buscar comentarios
- Tag: Cadena de caracteres que se buscará en los comentarios
- Tagged: Solo obtendrá comentarios que 'True'= contienen Tag, 'False'= no contienen Tag.
- Limit: Cuantos comentarios queremos recibir ('None' = compruebalos todos)
- Language: lenguaje que buscamos si buscamos de un usuario (Español por defecto)
- Quantity: Numero de comentario que queremos guardar (<0 ignora esta condicion)

```
def downloadFromUrl(pattern, target, tag, tagged, limit=None, language="es" ,quantity=-1):
    print(f"Saving comments from {pattern} {target}")
    maxSeen = 0
    count = 0
    dataset = {'comment':[],
               'author':[],
               'subreddit':[]}

    start_time = datetime.utcnow()
    previous_epoch = int(start_time.timestamp())

    while limit==None or maxSeen < limit:
        new_url = url.format(pattern,target)+str(previous_epoch)
        json = requests.get(new_url, headers={'User-Agent': "Comment downloader"})
        time.sleep(1) # pushshift has a rate limit, if we send requests too fast it will start returning error messages
        try:
            json_data = json.json()
        except ValueError:
            print("Response content is not valid JSON")
            continue
        if 'data' not in json_data:
            break
        objects = json_data['data']
        if len(objects) == 0 or count == quantity:
            break
        for object in objects:
            if count == quantity or (limitI=None and maxSeen == limit):
                break
            #if maxSeen is a number
            if maxSeen%1000==0:
                print(maxSeen)
            maxSeen += 1
            previous_epoch = object['created_utc'] - 1
            try:
                text = object['body']
                textASCII = text.encode(encoding='ascii', errors='ignore').decode()
                if (tag in textASCII) == tagged:
                    if pattern != "author" or detect(textASCII)==language:
                        count += 1
                        if tagged:
                            textASCII = trimTaggedString(textASCII, tag)
                        dataset['comment'].append(textASCII)
                        dataset['author'].append(object['author'])
                        dataset['subreddit'].append(object['subreddit'])
            except Exception as err:
                print(f"Couldn't print comment")
                continue
    print(f"Saved {count} comments out of {maxSeen} checked from {target}")
    classifier.saveModel(default_dataset_path+pattern+"\\"+target, dataset)
    return dataset
```

Inicializacion del diccionario contenedor de la informacion, elementos de control del bucle (maxSeen y count), y elementos que no ayudan a saltarnos el limite de 1000 objetos de respuesta por peticiones (previous_epoch)

"url" es la direccion a la API en formato string, con hueco para especificar nuestra busqueda. Al final concatenamos "previous_epoch". La respuesta que esperamos es un JSON

Extraemos el contenido JSON de la respuesta.
A veces el JSON no se recive correctamente o recivimos un error (suele ser por exceso de peticiones), en este caso volvemos a pedir el mismo JSON (mediante "continue")

1

1.- En caso de que el JSON este vacio (no hay mas elementos que recivir)

1 y 2

2.- En caso que ya hayamos encontrado el numero de datos pedido (count==quantity)

3.- En caso de haber observado el numero de elementos pedido (maxSeen==limit)

2 y 3

Sale del bucle

Herramienta de control para saber cuantos comentarios hemos comprobado hasta ahora

Actualiza "previous_epoch" al ultimo elemento conocido

Comprueba si el "tag" esta en el texto y cumple las especificaciones del usuario.
En caso de cumplirlas lo añade al diccionario

Si no podemos convertir el comentario lo saltamos, y pasamos al siguiente del paquete

Al guardar cada busqueda, no tendremos que realizarla en caso de que algo falle (esta parte ha sido necesaria ya que cada vez que aparecia un fallo inesperado habia que rehacer la busqueda, y ha habido muchos)

PRUEBAS

Las pruebas sobre el predictor se han hecho con diferentes tipos de sets de datos, y se ha comprobado el tiempo de entrenamiento y exactitud del modelo.
En este caso, la diferencia de tiempo es negligible, pero la exactitud tiene pequeños saltos.

Información del contenido de los diferentes archivos:

- **full_raw**: Comentarios sin procesar, primera iteracion.
- **emote_checked**: A partir de “full_raw”, solo comprueba si contiene emoticonos, y cuenta cuántos diferentes aparecen.
- **emote_counted**: A partir de “full_raw”, cuenta el número de emoticonos que aparecen en total por mensaje, los pone en una columna aparte y los elimina del texto.
- **spellchecked**: A partir de “emote_checked”, aplica un corrector ortográfico a todas las palabras desconocidas (que no conozca el corrector).
- **tokenized_emote_counted**: A partir de emote_counted, elimina todo lo que no sean palabras del texto.
- **lemmatized_tokenized_emote_counted**: A partir de tokenized_emote_counted, lematiza todas las palabras que no sean pronombres (mantiene pronombres intactos).

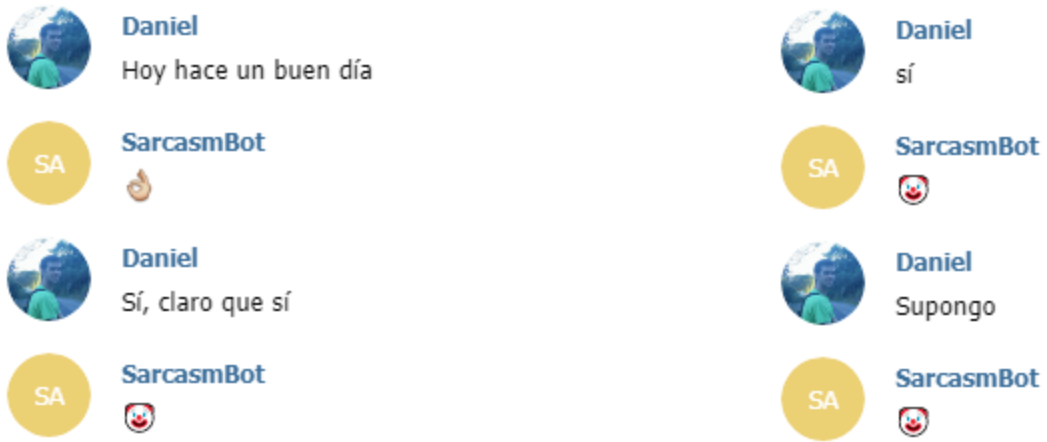
(ordenados en orden ascendente a juzgar por su exactitud)

Nombre del fichero	**Tiempo**	**Exactitud**
Spellchecked	0 m 14 s	0.6980881571959638
Raw	0 m 14 s	0.7007434944237918
Emote_checked	0 m 14 s	0.7022039298990972
Emote_counted	0 m 14 s	0.7033988316516198
Tokenized_emote_counted	0 m 14 s	0.7033988316516198
Lemmatized_tokenized_emote_counted	0 m 13 s	0.703797132235794

La intención de las pruebas era encontrar qué herramientas o capas de preparación son las mejores a la hora de procesar nuestros datos, y deberían verse como diferencias entre él y su predecesor.

En los tests vemos que emote_counted es una mejor técnica para predecir sarcasmo mediante el uso de emoticonos que emote_checked. Otras conclusiones son que tanto la tokenización del texto como lematización de palabras contribuyen muy positivamente tanto en velocidad como en exactitud final, esto es debido a que la normalización del texto facilita la agrupación de contenidos del texto y no fuerza al predictor a guardar diferentes formas del mismo significado.
El único resultado anómalo de las pruebas es el hecho de que corregir el texto dé tan pésimo resultado, pero es seguro por culpa de una mala corrección (elección de corrector) por mi parte, ya que es algo que debería normalizar el texto, y que el predictor encuentre más palabras en común entre pruebas suele ser algo deseado; eso o los fallos ortográficos son detonadores del tipo de mensaje, que es algo posible.

Para las pruebas en telegram he adjuntado un TOKEN que funciona y solo requiere de ejecutar "SarcasmBotMain.py" con conexión a internet. en caso de querer introducir otro bot solo se ha de cambiar el contenido del token con la nueva clave secreta.
El nombre del bot de prueba en telegram es: @ClownDetectorBot.



El ejemplo de la izquierda podemos decir que es deseado, pero también suceden casos como el ejemplo de la derecha donde no debería tener un sentido sarcástico porque apenas se ha dicho nada. Algo observado de esta manera es que el bot considera sarcástica casi cualquier tipo de afirmación.
Estos problemas surgen porque intentamos predecir la intención de mensajes sin un contexto, o la escasez de contextos en los datos a pesar de ser muestras balanceadas; donde puede que el 90% de muestras sarcásticas sean afirmaciones respondiendo a otro usuario, y los no artísticos tener intenciones más diversas.

METODOLOGÍA Y DESARROLLO

Se podría decir que el proyecto ha tenido 3 etapas, que he recorrido en una dirección, y una vez he llegado al final he tenido que volver sobre mis pasos etapa por etapa para afinar y ajustar lo que había cambiado. Esto es debido al aprendizaje que ha supuesto cada desafío, y mejoras que se han ido haciendo a medida que avanzaba el proyecto y aparecían nuevas ideas, herramientas o soluciones.

La primera etapa del desarrollo ha consistido en la creación del bot capaz de interactuar con el usuario de varias formas dependiendo del mensaje que éste escriba, esto ha sido rápido debido a que es un bot simple y es fácil encontrar ejemplos de bots similares.

La segunda etapa consiste en, mediante un set de prueba (en este caso, en inglés), preparar un clasificador y algunas herramientas para hacer el trabajo con más fácil y monitorizar resultados adecuadamente. La técnica utilizada para el preprocesamiento de los datos es únicamente Tf-idf ya que solo tenemos el texto aportado por los usuarios.

La tercera etapa comprende todo lo que se refiere a la captación de la información (en este caso los comentarios de los usuarios, su nombre de usuario y el tema donde los han escrito), y este ha sido con diferencia el más problemático puesto que no había nada de información al respecto y he tenido que buscar formas de conseguirla. Además de que lo poco que se podía encontrar de ayuda ha acabado entorpeciendo el desarrollo. (No hablo de la inexistencia de información sobre cómo recolectar datos y procesarlos, hablo estrictamente en el ámbito de hacerlo en reddit con una serie de especificaciones como serían filtros de texto o idioma)

Esta etapa es una etapa muy costosa debido al procesamiento de ingentes cantidades de datos que necesitábamos cumplieran una serie de demandas (que fueran en español, y que fueran sarcásticos). Además quería tener un conjunto de datos lo mas extenso posible y balanceado, y he acabado con 30 mil comentarios mitad sarcásticos, mitad no sarcásticos, que no son pocos.
Una vez tenía los datos he ido añadiendo capas de pre-procesamiento para preparar los datos y generar diferentes set con el objetivo de tener diferentes ejemplos para comparar resultados, de este modo al final podemos observar eficacia y escoger el que mejor respuesta genere. Los procesos eran: comprobar si hay emoticonos, contar los emoticonos diferentes, extraer los emoticonos del texto, contar todos los emoticonos que aparecen en el texto, aplicar un corrector a todos los textos, eliminar todo carácter no alfanumérico y lematizar toda palabra encontrada.

Una vez terminada esta etapa (que al final acaba siendo fácilmente el 60% del tiempo dedicado si no más) el clasificador queda obsoleto debido a la expansión de los datos a otros campos más allá de la conexión o aparición de palabras (aparecen nuevos campos numéricos y de etiquetas), y por lo tanto volvemos o revisamos la etapa 2. Por ello, aparecen otros componentes del clasificador para procesar dichas cualidades nuevas de los datos, pero seguimos utilizando el mismo tipo de regresión; de esta forma todo funciona igual desde fuera (siempre que los datos estén organizados como esperamos). Aprovechamos el momento para generar los modelos de prueba y comparar los resultados de los tests, que son la exactitud del modelo y una matriz de confusión de los resultados.

Y como último paso se adecua el mensaje recibido por el bot a los datos que espera el nuevo predictor y se genera la respuesta.

POSIBLES MEJORAS

En esta sección se describen posibles mejoras al proyecto:

- Crear un diccionario que relaciona todos los emojis con su respectivo emoticono, para poder tratarlos en el caso de que aparezcan.

- Creación de un fichero (ej: útil) que contenga estos métodos que necesitan distintos archivos. Esto solucionaría tener que incluir un fichero entero que no tiene sentido en el contexto de otro fichero.

- Flexibilidad en el tipo de regresión o clasificador que se quiera utilizar (podríamos querer utilizar un modelo basado en Random Forest por ejemplo). Esto podría haber aumentado el espectro en el que hacer pruebas.

- Comprobar que columnas deben ser tratadas de qué modo en base al tipo de datos que contiene utilizando funciones del Dataframe, asi no ajustamos el conjunto de datos a tener una serie de nombres (esto puede causar problemas a la hora de recuperar objetos dependiendo la manera que se haga, pickle guarda y recupera el mismo objeto, csv no).

- Flexibilidad a la hora de decidir que modelo y datos cargamos para el bot, en vez de “hardcodearlo”. Esto haría más fácil y directo el uso del código por un tercero, no tendría que mirar el código para implementar sus soluciones.

- Aplicar un corrector mejor, o prestar más atención a ese ámbito con el fin de conseguir una mejor corrección. El problema con esto ha sido la incapacidad de encontrar un corrector util (“hunspell” tenia muy buena pinta pero desistí de instalarlo en windows 10)

- El uso de la probabilidad de cada clase en el momento de su predicción podría ayudar a formar perfiles para los usuarios para ayudar a decantar futuros comentarios. (Podría ser algo similar a la tendencia de alguien a ser sarcástico)

- Que el bot responda citando mensajes, ya que en una conversación muy rápida o al recibir varios mensajes muy seguidos, puede que se pierda a quien va respondido qué mensaje.

DISCUSIÓN Y CONCLUSIÓN

En base a los resultados obtenidos, el objetivo de diseñar un modelo detector de sarcasmo ha salido bien puesto que se encuentra bastante por encima del 50%, considero un 70% un buen resultado dado el pequeño tamaño de la población de nuestros datos (hay datasets de millones de comentarios extraídos de Reddit, pero en inglés), pero para solucionar esto hay que invertir mucho más tiempo en buscar textos y quizá sería necesario expandir nuestras fuentes de información.

A nivel del Bot de Telegram, nos lleva tan lejos como el clasificador pueda ya que es algo modesto. Hace lo que al principio del desarrollo queríamos conseguir. Y aunque podría hacer otras cosas más superfluas, el Bot está completado y funcionando.

Lo último que queda valorar es mi experiencia en la realización del proyecto, y pese a las condiciones extremas en las que nos hemos visto recientemente, he hecho un trabajo que considero más que competente, he realizado con éxito muchas tareas en las que me he visto por primera, he aprendido conceptos relacionados con el procesamiento de lenguaje natural y su efecto en los modelos de predicción que desconocía o devaluada, y he sufrido los problemas de intentar armar un dataset desde cero con solo cierta idea de donde mirar. Así que estoy contento con el transcurso y resultado de este proyecto.

En total considero que he hecho un buen trabajo tanto en resultados como en el transcurso de éste.

BIBLIOGRAFÍA

- **https://github.com/aradan1/Sarcasm_Bot** (GitHub del proyecto)
- https://pypi.org/project/langdetect/ (detector de idioma)
- https://www.reddit.com/ (busquedas generales de informacion o soluciones a código, más notablemente la estructura del método captador de comentarios)
- https://es.wikipedia.org/wiki/Wikipedia:Portada (definiciones)
- https://spacy.io/ (libreria spacy, procesamiento de lenguaje natural y matriz sparse)
- https://stackoverflow.com/ (consultas de código generales, normalmente cosas simples que respondían a mi pereza, no porque no conocerlas)
- https://pypi.org/project/pyspellchecker/ (corrector ortográfico)
- https://telepot.readthedocs.io/en/latest/ (API de telegram)
- https://pushshift.io/ (API de Reddit)
- https://scikit-learn.org/stable/ (librería de Python contenedora de la majoria de herramientas de aprendizaje automático usadas)