Especialización en Ciencia de Datos

TRABAJO FINAL INTEGRADOR

Detección del Grado de Satisfacción del Empleado en la Organización

Alumno: Lic. Maximiliano Bordón

Tutor: Dra. Leticia Irene Gómez

Ciudad de Buenos Aires, Junio 2022

Índice

1. Introducción

Esta obra fue realizada con la intención de desarrollar un prototipo que detecte y clasifique opiniones procedentes de la fuerza de trabajo a partir de revisiones (reviews), con el objeto de permitirle al área de recursos humanos obtener un grado de satisfacción del trabajador y aplicar medidas que logren fidelizar al empleado de una manera efectiva. Cabe destacar que la solución propuesta se encontrará adaptada a las necesidades particulares de la empresa en donde se relevó la problemática.

El trabajo comienza describiendo el estado del arte de la cuestión. En dicha sección se hará alusión a algunos de los estudios realizados sobre esta temática. Luego, prosigue definiendo la problemática, considerando además la justificación del estudio, en donde se pondrá hincapié en las características que hacen que esta obra difiera de las demás relevadas.

Concluida la etapa de la justificación, nos remitiremos a detallar cual es el alcance del estudio, considerando las limitaciones derivadas de la disponibilidad de los recursos financieros, humanos y tecnológicos que se utilizarán para la consecución del proyecto.

Finalmente, establecido el alcance, el objetivo general y los objetivos específicos, se presentará el desarrollo de una prueba de concepto, como medida de solución al problema.

2. Estado del Arte / Contexto

En la actualidad, los empleados son uno de los mayores capitales en las organizaciones, ya que su bienestar se ve reflejado directamente en el desempeño global de la empresa, según plantean Piersanti, Brandetti y Failla (2017): “Human Resources are one of the most important assets in modern organizations. Their capability of facing employees’ needs is critical in order to have an effective and efficient company, where people are the center of all business processes”, (p.1).

Actualmente, se utilizan encuestas con el objeto de determinar el grado de satisfacción de los colaboradores, factor que determina su compromiso (employee engagement). El problema que se presenta, a menudo, es que las mismas suelen responder a un diseño sumamente genérico, lo cual deriva en una limitación para poder comprender las necesidades planteadas por los trabajadores. Sin embargo, el advenimiento del Big Data posibilita establecer modelos que permiten identificar y clasificar la percepción que tienen la fuerza de trabajo sobre la organización, haciendo posible para el área de recursos humanos abordar problemáticas tales como: altos índices de rotación y/o desmotivación, utilizando un enfoque proactivo orientado a los datos para tal fin.

Lo mencionado con antelación es posible por medio de la minería de opiniones (text mining) y el análisis de sentimientos (sentiment analysis). Es aquí cuando el texto libre perteneciente a una revisión adquiere preponderancia y se vuelve una piedra angular que contribuye a la realización del diagnóstico. Considerando las referencias bibliográficas, para dicho sondeo se tienen en cuenta redes sociales tales como Glassdoor/Lovemondays, Indeedo, Twitter, valiéndose eventualmente de emails como complemento para el análisis.

Cabe destacar que, a efectos de realizar la captura de datos, se utilizan APIs (Application Programming Interfaces) de desarrollador (como es el caso de Twitter) o bien webscraping. En líneas generales, al considerar redes sociales, si bien se cuenta con un rating o puntaje vinculado a cada
revisión (review), no se considera como un elemento definitorio ya que al tratarse únicamente de un número que engloba varios aspectos subyacentes, impide establecer una granularidad adecuada para reflejar la percepción de los trabajadores de manera efectiva, según ponen de manifiesto en (Luo, Zhou, Shon, 2016), afirmando lo siguiente: “We believe that ratings alone from Glassdoor is too generic and fail to reflect the granularity of employee satisfaction. Thus, we are interested in finding employee satisfaction dimensions from textual reviews”,(p.3).

Los aspectos a que se hace mención están vinculados, en líneas generales, a balance vida personal/trabajo, salario, oportunidades de crecimiento, trabajo en equipo, comunicación, respeto.

Este enfoque permite desglosar las opiniones en categorías, lo cual se logra a partir de una descomposición en clústeres o grupos. Finalmente se le asigna sentimientos a cada una de las categorías.

Aún más, encontramos que en una de las investigaciones relevadas (Luo, Zhou y Shon, 2016) se propone manifestar la relación entre la satisfacción del empleado y la performance de la compañía, considerando que categorías/valores inciden en esta correlación. Los autores manifiestan textualmente “We extract anonymous employee reviews for textual analysis to reveal the relation between employee satisfaction and company performance. Using categories from corporate value studies, our analysis not only provide a “bird’s eye view,” but alsoprovide specific aspects of employee satisfaction are responsible for driving these correlations”,(p.1). En esta obra también llegan a determinarse cuales categorías son las que tienen un impacto negativo.

En otro de los trabajos encontrados se toma en consideración aquellos factores que repercuten negativamente para permitirle a recursos humanos llevar a cabo acciones que reviertan la situación, tal como indican Piersanti, Brandetti y Failla (2017): “With the present study, we aim to categorize employee satisfaction in a more detailed and automatic way, identifying common trends among employees and clustering them into groups that share similar problems. The goal is to help HR-BPs in having an overall view of their resources’ mood and make effective adjustments in critical situations”, (p.2).

Este mismo enfoque tambien se mantiene en otro de los trabajos considerados como referencia. En (Costa & Veloso, 2015) se afirma: “In this paper we propose approaches that capture and analyze employee feedback, and then use them to understand factors that impact employee retention,satisfaction and productivity”, (p.1).

Todo lo anteriormente expuesto sienta las bases que constituirán el hilo conductor del presente trabajo.

3. Definición del Problema

La problemática que nos atañe consiste en la falta de sistematización, dentro de una organización, para llevar a cabo la detección y clasificación de las opiniones de los empleados incorporando parámetros propios de la empresa.

De lo relevado en la Sección 2, se deriva que los medios utilizados por las organizaciones, tales como una plataforma online de encuestas tercerizado o cartas de salida, no resultan efectivos. En el primero de los casos, responden a un diseño muy genérico que no permite identificar de manera adecuada los aspectos a fortalecer. Por otra parte, las mismas se disponen en periodos espaciados de tiempo, lo cual inhibe la creación del hábito, por parte de colaboradores, de aportar opiniones constructivas a través de un canal formal, con el objeto de mejorar el clima laboral. Lo mencionado con antelación deriva en el hecho que las encuestas son completadas de forma eventual. Respecto a las cartas de salida, se expresan sólo cuando el empleado renuncia, es decir representa una instancia tardía para tomar un curso de acción que revierta esta situación. empresa.

4. Justificación del Estudio

Este trabajo se lleva a cabo con el fin de incorporar una herramienta mejor adaptada a las particularidades de la empresa, con el objeto de determinar el grado de satisfacción de la fuerza de trabajo. Se trata de una propuesta que tiene por finalidad cubrir una necesidad puntual de la organización.

La novedad de nustra propuesta radica exclusivamente en la parametrización de determinados selectores, tales como las redes sociales laborales, que se incluirán como fuente de información, sumando además las diferentes categorías que intervendrán en la clasificación, tales como balance vida personal/laboral, salario, oportunidades de crecimiento, comunicación, respeto, trabajo en equipo. Debe considerarse que adicionalmente se incluirán las revisiones procedentes de encuestas propias de la empresa.

Según el relevamiento efectuado en otras obras solo se considera por estudio un conjunto fijo de parámetros, tal como un conjunto acotado de redes sociales para la extracción de información, complementándose eventualmente con emails, sin considerar otras fuentes adicionales que pueden utilizarse de manera combinada como se ponderan en la presente obra.

5. Alcances del Trabajo y Limitaciones

Los usuarios directos de la herramienta qye se presenta en este trabajo, serán el área de recursos humanos de la empresa, ya que entre sus tareas se busca poder afianzar la relación con el trabajador, sondeando el clima laboral, a los efectos de evitar una alta tasa de rotación y fomentar adicionalmente el interés de posibles candidatos que apliquen para la organización.

La herramienta permitirá al área de recursos humanos poder conocer el grado de satisfacción del empleado considerando diferentes categorías tales como: balance vida personal/laboral, salario, crecimiento, comunicación, respeto, trabajo en equipo e identificar oportunidades de mejora.

Cabe destacar que la aplicación será un prototipo, el cual será montado en un entorno de trabajo local (notebook). El mismo contará con las siguientes características técnicas: 16 GB de RAM, procesador Intel 5, sistema operativo Windows y 1 TB de disco rígido. Estos features tendrán repercusión directa sobre la performance del aplicativo durante las etapas de extracción de información, en donde debemos considerar la cantidad posible de hilos de ser disparados en paralelo para recopilar información, carga, para la cual debemos tomar en consideración la capacidad de almacenaje destinada a las revisiones y procesamiento, el cual estará condicionado por la cantidad de ítems a computar. Por las razones mencionadas con antelación, la carga de trabajo destinada deberá ser coherente con las limitaciones de la plataforma.

Uno de los inputs requeridos para el modelo depende de una aplicación externa a este trabajo, más específicamente una plataforma online de encuestas, actualmente en desarrollo por parte de la compañía en la cual se utilizará nuestro prototipo. Esta plataforma almacenará revisiones en una base de datos que será utilizada como entrada por nuestro aplicativo. Si por determinados factores, tales como retrasos en la planificación u otros motivos bloqueantes no se llegara a contar con este input, la ingesta de información quedará limitada a la utilización de las redes sociales mencionadas con anterioridad en el presente documento.

Además de los resultados arrojados en el informe correspondiente a este trabajo, existirá una instancia en donde la aplicación será testeada por el área de recursos humanos. Es necesario mencionar que este sector requerirá una sesión de capacitación para probar debidamente la herramienta.

Durante el desarrollo del proyecto no se contará con asistencia financiera por parte de la compañía. Se trata de una propuesta de valor generada internamente que dará lugar a un prototipo y en función de los resultados arrojados por el estudio y el testeo efectuado por recursos humanos, la organización podrá aceptar o rechazar la propuesta.

La información disponible en este trabajo será de dominio público, no siendo requerida ninguna restricción en lo que atañe a la confidencialidad de la obra.

6. Objetivos

6.1. Objetivo General

El objetivo del presente trabajo es construir un prototipo, utilizando algoritmos de machine learning, para la detección y clasificación de opiniones de los empleados con respecto a la empresa.

6.2. Objetivos Específicos

- Construir una herramienta de software que permita capturar las revisiones de redes sociales a través APIs o por medio de WebScraping, teniendo en consideración un relevamiento previo de los sitios web a consumir.

- Consolidar un dataset con las nuevas encuestas de la empresa y revisiones de redes sociales laborales.

- Crear un modelo que detecte opiniones y las clasifique bajo categorías tales como: balance vida personal/ trabajo, salario, oportunidades de crecimiento, comunicación, respeto, trabajo en equipo.

- Realizar la validación del modelo generado en el punto anterior.

- Generar reportes de satisfacción del empleado respecto a la empresa, considerando categorías mencionadas en el tercer objetivo.

- Construir una interfaz web que le permita al operador efectuar las siguientes funciones: parametrización de selectors, activación/desactivación del crawler desarrollado, activación/desactivación del módulo destinado al análisis de sentimientos y la clasificación de las categorías, generación de reportes sobre grado de satisfacción del empleado..

- Capacitar al área de recursos humanos para la utilización del prototipo.

7. Metodología Aplicada

Para la implementación del prototipo, se empleará SCRUM (Agile) como metodología de trabajo, ya que permite dividir el proyecto en diferentes fases identificando objetivos y factores bloqueantes para cada una de las mismas. Dichas etapas estarán en consonancia con los objetivos específicos mencionados en la Sección 6.2.

7.1. Modelo Conceptual

El modelo propuesto incorpora parámetros propios de la empresa, para detectar y clasificar opiniones de los empleados efectuando análisis de sentimientos.

Se consideran las siguientes variables cualitativas, cuya relación es del tipo causa-efecto:

- **Opinión:** desde el punto de vista nominal, se refiere a la percepción que tiene una persona sobre una serie de tópicos asociados a la empresa/proyecto, la cual es expresada en texto que tiene presencia en la encuesta o revisión. Si consideramos la definición operacional será medida en: cantidad de palabras total, cantidad de palabras en mayúsculas, número de signos de exclamación y/o interrogación, emoticones, ratings, etc. Se trata de la variable independiente y es clasificatoria.
- **Categoría:** desde el punto de vista nominal, constituye el tema o tópico sobre el cual versa la opinión y se descubre a partir de la misma. Se trata de una variable dependiente de la opinión y es clasificatoria. Desde el punto de vista operacional se consideran las siguientes posibilidades**:** balance vida personal/trabajo, salario, oportunidades decrecimiento, comunicación, respeto, trabajo en equipo**.** Siempre tendrá asociado un sentimiento.
- **Sentimiento:** desde el punto de vista nominal, es el estado que se calcula a partir de la opinión y se asocia a una categoría. En su definición operacional podrá oscilar entre los siguientes valores: positiva, negativa y neutral. Esta variable depende de la opinión y la categoría siendo clasificatoria.

Inicialmente, habrá una etapa de extracción de datos desde la web, para luego aplicar transformaciones del texto recopilado y proceder con la carga en una base de datos NoSQL.

También existirá un módulo de análisis de sentimientos, que tomará como input las revisiones analizadas, y en una instancia posterior se clasificarán las mismas bajo las categorías de los valores corporativos, lo que determinará como resultado el grado de satisfacción del empleado en la organización.

Con respecto a la variable sentimiento**,** si bien las librerías disponibles ofrecen un manejo de tipo numérico, nosotros enmascararemos este valor bajo los siguientes valores: positivo, negativo y neutro**.**

Finalmente, para completar la prueba de concepto, desarrollaremos una interfaz web que permita efectuar al operador las siguientes funciones:

a) Parametrización de selectores que serán utilizado como input del modelo tales como sitios web en donde se efectuará la recolección de información, repositorio de opiniones procedentes de la nueva herramienta de la empresa y categorías utilizadas en la clasificación.

b) Activación/desactivación del crawler desarrollado en el inciso uno (uno) destinado a la fase de extracción de la información (reviews).

c) Activación/desactivación del módulo destinado al análisis de sentimientos y la clasificación de las categorías mencionadas en el inciso tres (3).

d) Generación de reportes a demanda en donde se verifique el grado de satisfacción del empleado considerada las categorías seleccionadas.

En la Figura 1, se puede ver visualmente el esquema de los solución propuesta.

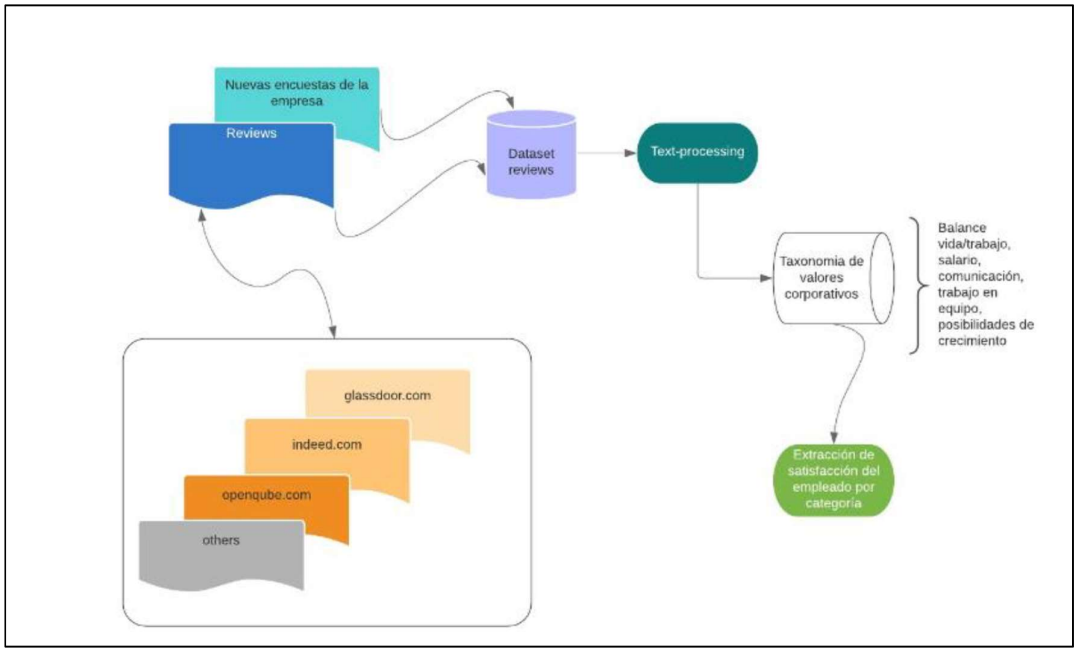


Fig. 1 - Diagrama de la Solución – Basado en la obra de Luo, Zhou y Shon, 2016

7.2. Algoritmos y Herramientas a Utilizar

Para el desarrollo de la prueba de concepto, se utilizará lenguaje de programación Python.

Para la etapa de extracción de información desde la web, se utilizará WebScraping, una técnica que simulan la navegación de un humano. Para ello se aplicarán las librerías PySpider, BeautifulSoup, Mechanical Soup, Scrapy y Selenium.

A los efectos de poder almacenar las reviews se optará por una base de datos NoSQL Mongo DB, lo cual nos proporcionará flexibilidad en cuanto al tratamiento de los datos, permitiendo además una mayor escalabilidad de la solución.

Con el objeto de efectuar el análisis de sentimientos se utilizará librerías de Python tales como NLTK (Natural Language Toolkit) y TextBlob.

Para llevar a cabo la clasificación bajo las categorías de los valores corporativos, se utilizará el algoritmo K-Means clustering.

Dicho algoritmo es delo tipo de clasificación no supervisada, el cual se utiliza cuando se tienen datos no etiquetados, es decir datos sin categorías o grupos definidos. El objetivo que se persigue, con su aplicación, es poder agrupar los datos presentes, a partir de sus características, considerando un número de grupos o clústeres, los cuales se encuentran representados por la variable K.

El algoritmo cuenta con las siguientes fases: La primera, de inicialización consiste en el establecimiento de K centroides una vez definida la variable K, luego cada uno de los objetos intervinientes es asignado a su centroide más cercano (considerando la distancia cuadrada euclidiana) y finalmente se lleva a cabo la actualización de los centroides, es decir se actualiza la posición del mismo en cada clúster o grupo al considerar como nuevo centroide la posición promedio de los objetos pertenecientes a esa agrupación.

La principal ventaja de dicho algoritmo es que puede aplicarse de manera fácil y rápida, ya que es sumamente versátil refiriéndonos a los tipos de agrupación que pueden considerarse. La principal desventaja radica es que es necesario definir el número de clústeres o grupos (es decir la variable la K) a priori; esta definición puede afectar considerablemente a los resultados y, adicionalmente, esta técnica es muy susceptible a *outliers.*

8. Desarrollo del Modelo

8.1. Fuentes de Información

Las fuentes de datos a utilizar se basan en dos websites: **indeed.com** y **openqube.com**. La información recolectada son las opiniones volcadas por las personas en las páginas utilizando **web-scraping**.

A modo ilustrativo, en las Figuras 2 y 3 se adjuntan capturas de pantallas de los mencionados websites.

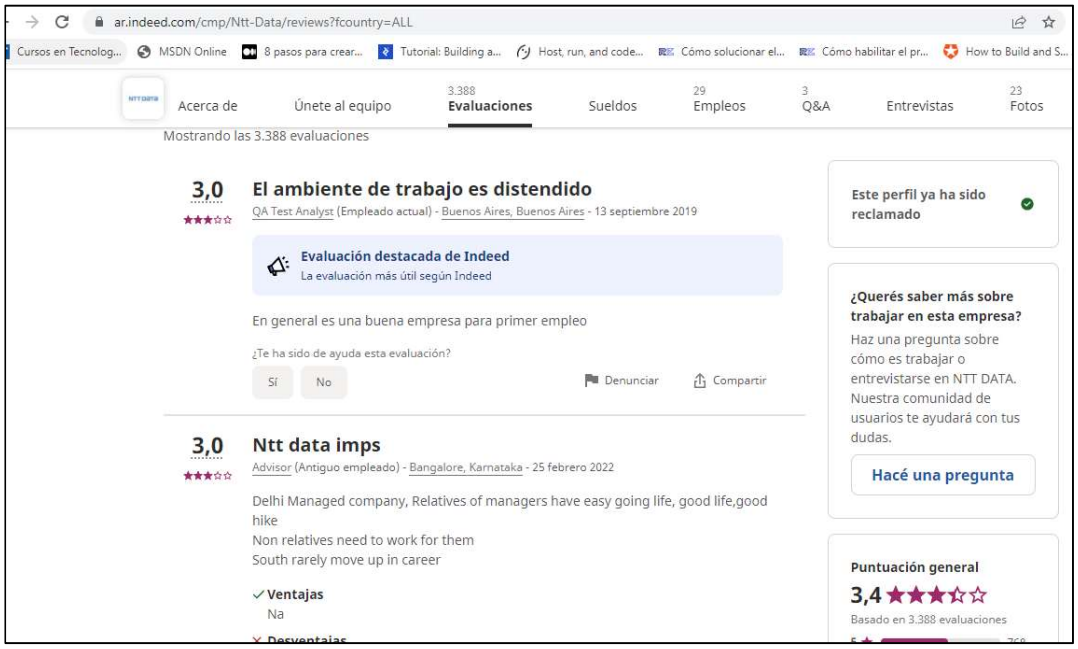


Fig. 2 – Captura de pantalla de opiniones en Indeed.com

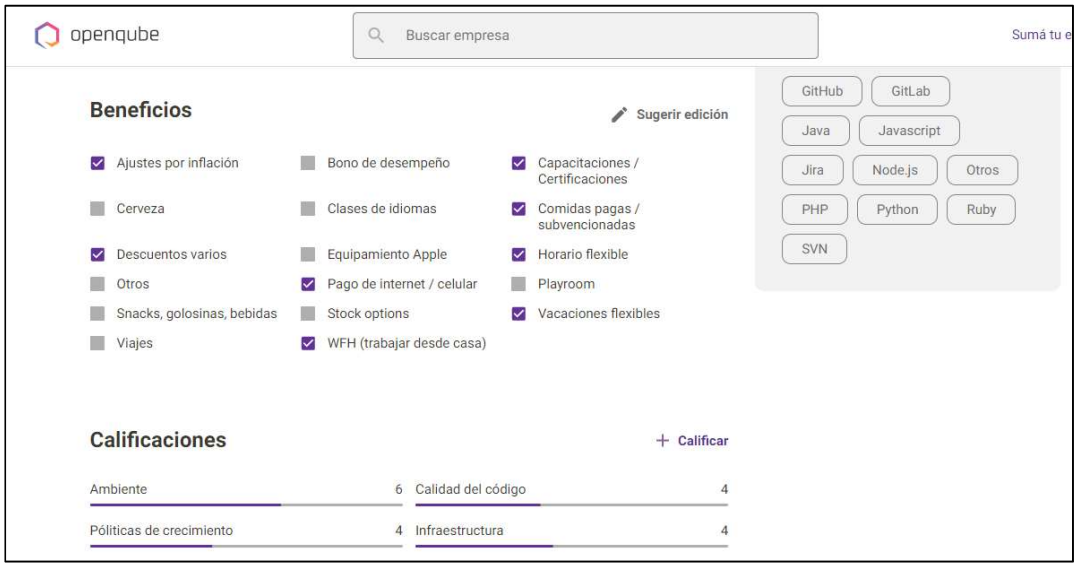


Fig. 3 – Captura de pantalla de opiniones en OpenQube.com

8.2. Proceso de Transformación Aplicado

Como se mencionó en la Sección 8.1., para la etapa de extracción, se consideraron dos fuentes de información, **indeed.com** y **openqube.com**. Se utilizó webscraping para llevar a cabo la recolección de las opiniones. Debido a que estos dos sitios cuentan con diseños completamente distintos tuvo que diseñarse una clase de recolección distinta para cada uno de ellos.

En las Secciones 12.3. y 12.4 del Anexo, puede consultarse el código fuente Phyton empleado para la recolección de opiniones de cada uno de estos dos sitios, a través de los scripts **indeedscraper.py** y **openqubescraper.py.**

Es importante mencionar que, a los efectos de evitar el bloqueo por parte de los sitios web, se añade una pausa en la búsqueda, al momento de realizar la extracción de la información, y adicionalmente se procede a utilizar un proxy distinto cada vez, distribuidos en diferentes regiones del planeta. Esto se logra por medio de la articulación

de dos clases llamadas llamadas **ProxyPool** y **ProxyScraper** en donde la primera disponibiliza un pool de proxies que carga llamando a la segunda.

Como se mencionó anteriormente, la clase ProxyScraper recupera los proxies invocando a la página *https://free-proxy-list.net/* y devolviendo una lista a ProxyPool.

En las Secciones 12.5 y 12.6 del Anexo puede consultarse el código fuente de las clases **ProxyPool** y **ProxyScraper**, respectivamente.

A medida que cada extractor se va ejecutando, se procede con la invocación de una API destinada a la traducción del comentario original al idioma Inglés, antes de proceder a su almacenamiento en la base de datos mongo.db.. En la Sección 12.7 del Anexo se encentra a el código fuente de la clase **TranslatorApi.**

Cabe descatar que los datos de configuración se encuentran disponibles en el archivo **apitranslatorconfig.json**, como se muestra a continuación:

```
{
 "url":"https://translate.yandex.net/api/v1.5/tr.json/translate",
 "key":"trnsl.1.1.20211011T195920Z.ab5b9d2b3c270ca6.8e61ebc044a725338f0f9
c5d6bb0402850799ac2",
 "language":"en"
}
```

Finalmente la información traducida es almacenada en base de datos MongoDb, utilizando como servicio de acceso a datos la clase ComentarioDB. El código fuente de dicha capa se puede consultar en la Sección 12.8 del Anexo.

Para llevar a cabo la administración de las bases de datos creadas en Mongo, se utiliza **MongoCompasDb**, así como también las respectivas colecciones, que en nuestro caso particular es una sola intitulada “comentarios”.

En la Figura 4 se puede consultar una vista de la base de datos con los documentos cargados.

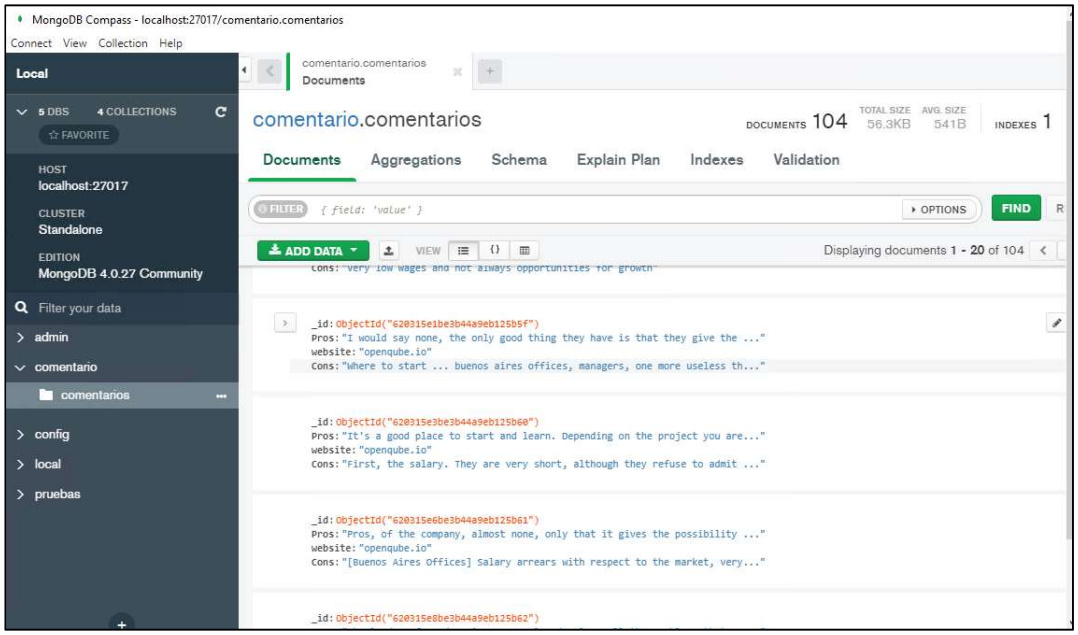


Fig. 4 – Vista de la base de datos con documentos cargados

8.3. Análisis de Sentimientos

Para el análisis de sentimientos se desarrolló el módulo llamado **sentimentanalyzer.py**, el cual utiliza la librería **ntlk** para llevar a cabo esta tarea. El texto provisto como input se encuentra en inglés, ya que ha sido transformado antes de almacenarse en la base de datos, tal como se indicó en la Sección 8.2. Se ha elegido este idioma por resultar sumamente conveniente para este tipo de labores de análisis.

Se exponen dos funciones: la primera de ellas, denominada **tokenize,** tiene por finalidad descomponer un párrafo en múltiples oraciones, mientras que la segunda, denominada **analyze**, es la rutina que efectúa el análisis de sentimientos propiamente dicho, devolviendo un diccionario compuesto de cuatro valores**: neg (negativo), neu (neutral), pos (positivo)** y **compound (compuesto)**. Cabe destacar que de los devueltos el coeficiente preponderante es compound (cuanto más cercano a 1 sea más positivo será el comentario, si por el contrario este valor estuviera desplazado a los números inferiores a cero la valoración será negativa).

A modo ilustrativo se hizo una prueba con el siguiente texto: ***"The environment is so stressful. The salary is good".*** Fue la rutina tokenize, la que procedió a la descomposición del texto en dos sentencias simples, al detectar un delimitador punto, en este caso.

La salida de la ejecución fue la siguiente:

1. *{'neg': 0.515, 'neu': 0.485, 'pos': 0.0, 'compound': -0.6418}*
2. *{'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}*

donde la primera ocurrencia alude a **"The environment is so stressful".** En este punto podemos observar que efectivamente el compound es negativo, por tanto el significado atribuido a la frase también es negativo. La segunda ocurrencia observada refiere a "**The salary is good",** el compound arrojado es positivo, por tanto la valoración de la frase es positiva también.

Cabe aclarar que, el "main" no se ejecutará si es que se procede a llamar a este módulo desde otro componente haciendo referencia a sus rutinas directamente, que es la forma de ejecución que actualmente tiene lugar.

Para la fase de clasificación, más específicamente para conocer la valoración del comentario asociado a un tópico en particular se utilizará el valor compound.

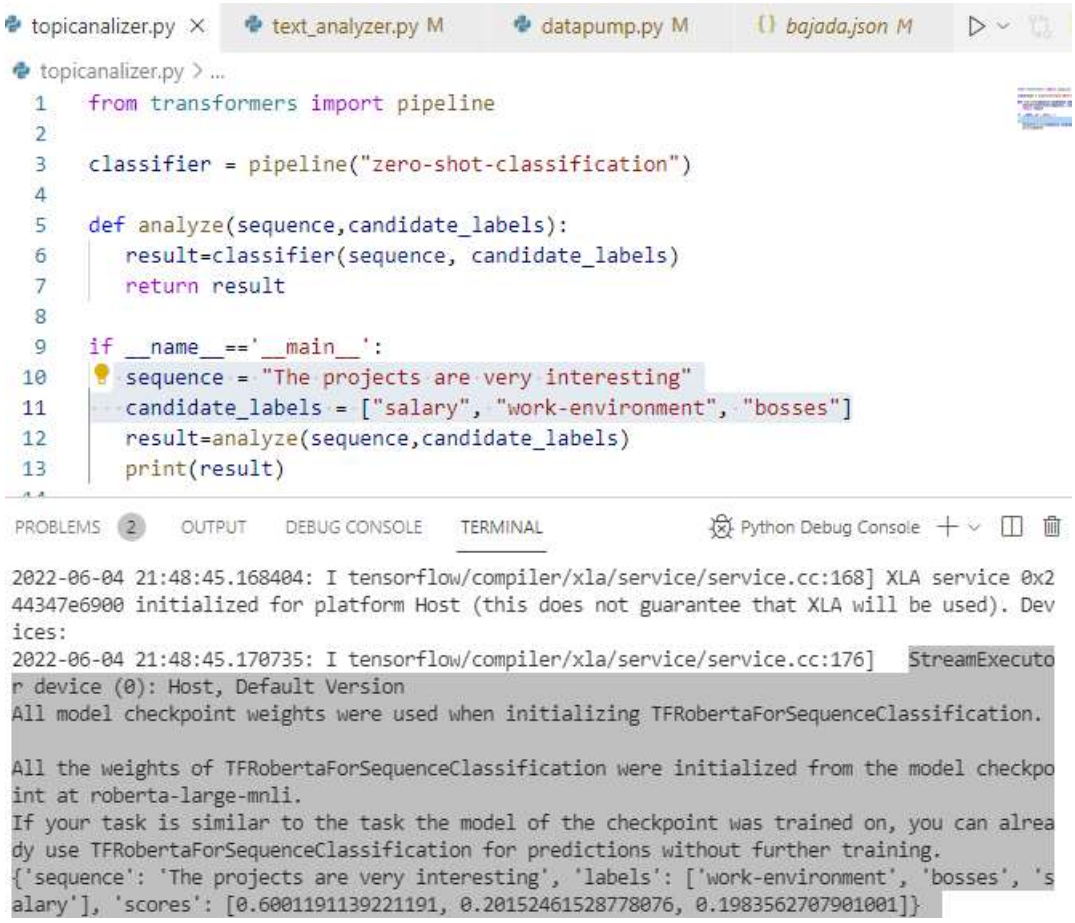
En la Sección 12.9 del Anexo se puede consultar el código fuente de este módulo.

8.4. Clasificación de las Revisiones

En esta etapa intervienen varios scripts. Uno de ellos es ***topicanalizer.py*** (ver Sección 12.13 del Anexo) que tiene por finalidad recuperar los comentarios almacenados en la base de datos mongo y, acorde a criterios guía cargados en el sistema (por ejemplo: salary, work-environment, bosses, etc), clasifica el comentario en proceso, es decir detectar a qué se refiere, por medio de **Zero-Shot Classification.** Ésta última es una técnica que permite asociar una etiqueta (label)a una pieza de texto,

a través de un modelo pre-entrenado llamado **ROBERTA** (a Robustly Optimized BERT[1] Pretraining Approach). En este caso, las etiquetas son los criterios guía.

A fines didácticos, se incluyen tres ejemplos de ejecución. A continuación, mostramos el primero de ellos:



El input proporcionado en topicanalizer.py fue el siguiente:

sequence = "The projects are very interesting"

candidate_labels = ["salary", "work-environment", "bosses"]

Con dicho input, se arrojaron los siguientes resultados de ejecución:

[11] Bidirectional Encoder Representations from Transformers

{'sequence': 'The projects are very interesting', 'labels': ['work-environment', 'bosses', 'salary'], 'scores': [0.6001191139221191, 0.20152461528778076, 0.1983562707901001]}

Esto quiere decir que la sentencia “The projects are very interesting” se encuentran asociado al label o criterio “work-environment”, si consideramos que la ponderación máxima asignada es 0.6001191139221191, mientras que para “bosses” es de 0.20152461528778076 y a “salary” le corresponde una ponderación de 0.1983562707901001.

Si tomamos como referencia otro ejemplo y modificamos la sentencia de entrada a “The managers are wonderful people”, conservando los criterios guía o labels anteriormente mencionados el resultado es el siguiente:

```
1   from transformers import pipeline
2
3   classifier = pipeline("zero-shot-classification")
4
5   def analyze(sequence,candidate_labels):
6       result=classifier(sequence, candidate_labels)
7       return result
8
9   if __name__=='__main__':
10      sequence = "The managers are wonderful people"
11      candidate_labels = ["salary", "work-environment", "bosses"]
12      result=analyze(sequence,candidate_labels)
13      print(result)
```

```
ROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL        Python Debug Console

022-06-04 23:58:38.690718: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your CPU sup
orts instructions that this TensorFlow binary was not compiled to use: AVX2
022-06-04 23:58:38.698302: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x1
709a1b9a0 initialized for platform Host (this does not guarantee that XLA will be used). Dev
ces:
022-06-04 23:58:38.700607: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecuto
 device (0): Host, Default Version
ll model checkpoint weights were used when initializing TFRobertaForSequenceClassification.

ll the weights of TFRobertaForSequenceClassification were initialized from the model checkpo
nt at roberta-large-mnli.
f your task is similar to the task the model of the checkpoint was trained on, you can alrea
y use TFRobertaForSequenceClassification for predictions without further training.
'sequence': 'The managers are wonderful people', 'labels': ['bosses', 'work-environment', 's
lary'], 'scores': [0.7696594595909119, 0.21921265125274658, 0.011127856560051441]}
5 C:\Users\Administrator\tpfinal\tpfinal>
```

Aquí puede observarse que la sentencia de entrada se esta refiriendo a “bosses” debido a la valoración asignada es de 0.7696594595909119 para este label, la cual es superior en contraposición a las restantes como ser “work-environment” cuyo valor es de 0.21921265125274658, mientras que “salary” queda rezagado con una ponderación de 0.011127856560051441.

Como tercer ejemplo, se incluye una ejecución donde la sentencia utilizada como input es modificada a “pay is way under market”, mientras se conservan los criterios o labels anteriormente indicados.

```python
from transformers import pipeline

classifier = pipeline("zero-shot-classification")

def analyze(sequence,candidate_labels):
    result=classifier(sequence, candidate_labels)
    return result

if __name__=='__main__':
    sequence = "pay is way under market"
    candidate_labels = ["salary", "work-environment", "bosses"]
    result=analyze(sequence,candidate_labels)
    print(result)
```

```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  Python Debug Console

ices:
2022-06-05 01:35:34.910527: I tensorflow/compiler/xla/service/service.cc:176]  StreamExecuto
r device (0): Host, Default Version
All model checkpoint weights were used when initializing TFRobertaForSequenceClassification.

All the weights of TFRobertaForSequenceClassification were initialized from the model checkpo
int at roberta-large-mnli.
If your task is similar to the task the model of the checkpoint was trained on, you can alrea
dy use TFRobertaForSequenceClassification for predictions without further training.
{'sequence': 'pay is way under market', 'labels': ['salary', 'work-environment', 'bosses'], '
scores': [0.8605127334594727, 0.11105599999427795, 0.028431233018636703]}
PS C:\Users\Administrator\tpfinal\tpfinal>
```

En este último ejemplo, podemos observar que la sentencia hace referencia al salario o “salary” ya que la ponderación obtenida para este label es la máxima: 0.8605127334594727, mientras que para “work-environment” se obtuvo un

0.11105599999427795, quedando en último lugar la etiqueta “bosses” con un valor de 0.028431233018636703.

Luego de detectar a qué refiere cada frase, se le asigna un valor positivo o negativo, dependiendo del *compound* capturado en el análisis de sentimiento, esta labor es realizada por el script **sentiment_analyzer.py** (Sección 12.9 del Anexo). Los dos scripts anteriores son invocados por **text_analyzer.py** (Sección 12.14 del Anexo). Una salida del script que combina la detección de tópicos con la asignación de sentimientos, si se toma como referencia la sentencia de entrada anterior, es la siguiente:

{'topic': 'salary', 'feeling': -0.1027, 'phrase': 'pay is way under market'}

En este output puede verse que el tópico detectado en función de la frase “pay is way under market” es salary o salario y la connotación que posee en referencia a los sentimientos es negativa ya que en la entrada denominada como *feeling* figura un compound con valor inferior a cero.

Otro ejemplo combinado, pero de connotación positiva, es el siguiente (como puede observarse el campo feeling donde figura el valor del compound es superior a cero)

{'topic': 'work-environment', 'feeling': 0.4585, 'phrase': "I still can't find any negative points."}

Finalmente, un script llamado **datapump.py** (Sección 12.15 del Anexo) es el que llama a text_analizer.py y genera un archivo json de output. Motramos a continuación un ejemplo:


```
{"topic": [0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 1, 1, 1, 3, 1, 1, 3, 1, 0, 1, 3,
2, 2, 3, 1, 1, 1, 2, 2, 2, 3], "opinion": [0.705, -0.5409, -0.4215,
0.8363, 0.6124, 0.0, -0.2446, 0.4215, 0.6808, 0.0, -0.0754, -0.6249, -
0.5423, 0.0, -0.3314, -0.25, 0.0, -0.2755, 0.0, -0.1511, -0.4019, -
0.7096, -0.3182, 0.0772, 0.3818, -0.3182, -0.3182, -0.6696, 0.0, 0.0, -
0.5256]}
```


Aquí, *topic* contendrá la lista de tópicos numerados del 0 en adelante, mientras que en la lista *opinion* se almacenarán los compounds recuperados del análisis de

sentimientos previamente realizado. Este archivo será utilizado como input por **cluster.py** (Sección 12.17 de Anexo), que tendrá por finalidad llevar a cabo la categorización de tópicos por opinión y generación del gráfico resultante.

Como se mencionó con anterioridad, para poder aplicar la clusterización es necesario definir la variable K. En nuestro caso para efectuar tal definición se aplicaron las siguientes rutinas dentro del módulo:

- Se agrupan en un diccionario las opiniones por tópico por medio de la rutina **groupbytopicsandopinion**

```
def groupbytopicsandopinion(topiclist,opinions):
      results={}
      for topic, opinion in zip(topiclist,opinions):
            if not topic in results:
                  results[topic]=[opinion]
            else:
                  results[topic].append(opinion)

      return results
```

- Tomando como entrada el diccionario generado en este punto, se genera uno nuevo contabilizando los tipos de opiniones por tópico. Se contabilizan positivas y negativas, por medio de la rutina **calculate_summarized** en función del compound calculado en etapas previas del análisis.

```
def calculate_summarized(results):
       summarized={}
       for topic in results:
         lista=results[topic]
         summarized[topic]=0
         neg_count = len(list(filter(lambda x: (x < 0), lista)))
         pos_count = len(list(filter(lambda x: (x >= 0), lista)))
         if neg_count>0:
              summarized[topic]+=1
         if pos_count>0:
              summarized[topic]+=1
       return summarized
```

- Finalmente se procede con la sumatoria de los tipos de opinion por tópico dando lugar al cálculo de K.

```
def calculate_k(topiclist, opinions):
      results=groupbytopicsandopinion(topiclist,opinions)
      summarized=calculate_summarized(results)
      k=0
      for topic in summarized:
           k+=summarized[topic]
      return k
```

- A partir de K y del input antes mencionado se procede con la clusterización.

```
def make_cluster(Data):
       df = DataFrame(Data,columns=['topic','opinion'])
       kmeans =
KMeans(n_clusters=calculate_k(Data['topic'],Data['opinion'])).fit(df)
       centroids = kmeans.cluster_centers_
       print(centroids)
       print(kmeans.labels_)
       colores=['red','green','blue','cyan','yellow','violet',
'orange','black']
       asignar=[]
       for row in kmeans.labels_:
            asignar.append(colores[row])


       plt.scatter(df['topic'], df['opinion'], marker="o",s=100,
alpha=0.5,c=asignar)
       plt.scatter(centroids[:, 0], centroids[:, 1],marker = "*",
alpha = 0.9,s=300,c='red',label='Centroides')
       plt.title('Clúster de  tópicos')
       plt.xlabel('Tópicos')
       plt.ylabel('Opiniones')
       plt.legend()
       plt.savefig(output)
```

8.5. Herramienta de Asistencia al Usuario

Se desarrolló una aplicación, utilizando el framework **Django de Python,** que permite la administración (**CRUD – Create, Read, Update, Delete**) de usuarios, los orígenes de datos utilizados como fuentes de información como *websites* y los criterios para el análisis de las opiniones recolectadas.

A modo ilustrativo, en la Figuras 5 a 11 se muestran capturas de pantalla con la configuración de las diferentes opciones que corresponden al usuario **administrador.**

Fig. 5 – Pantalla de administración de usuarios

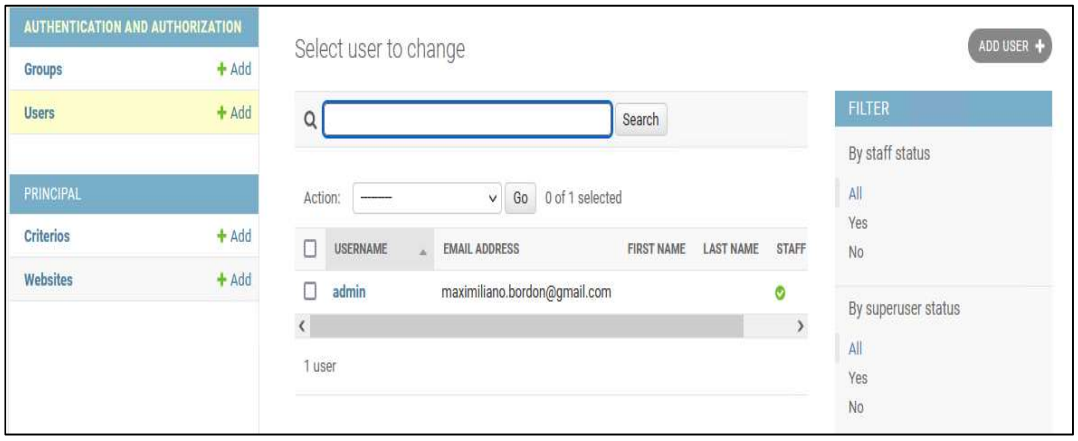


Fig. 6 – Pantalla de selección de usuarios para edición

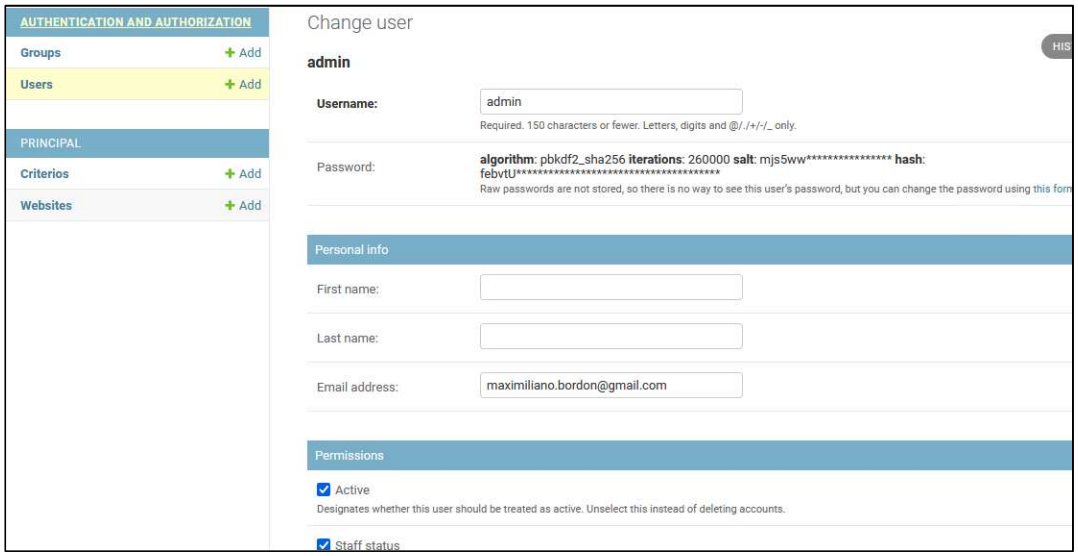


Fig. 7 – Pantalla de modificación de usuarios

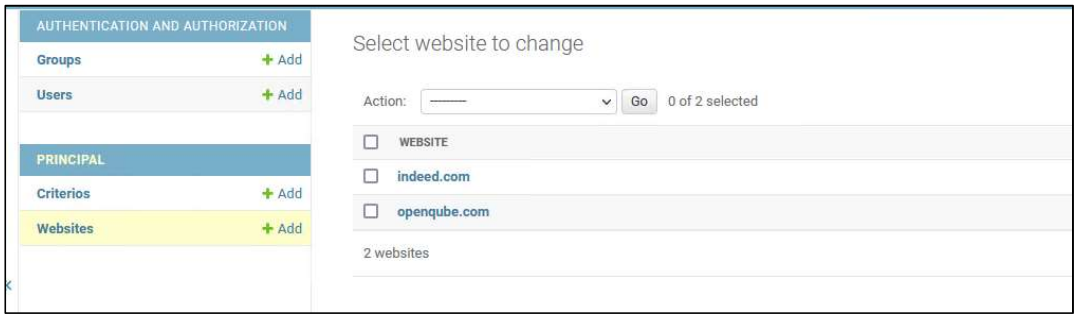


Fig. 8 – Pantalla de selección de orígenes de datos para edición

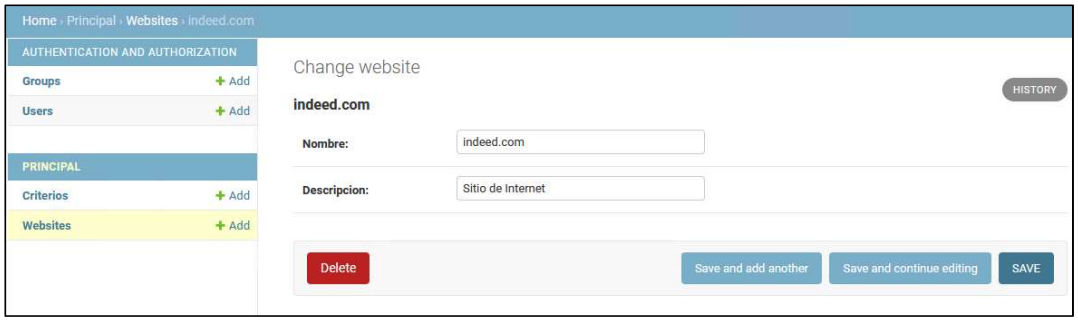


Fig. 9 – Pantalla de modificación de orígenes de datos (websites)

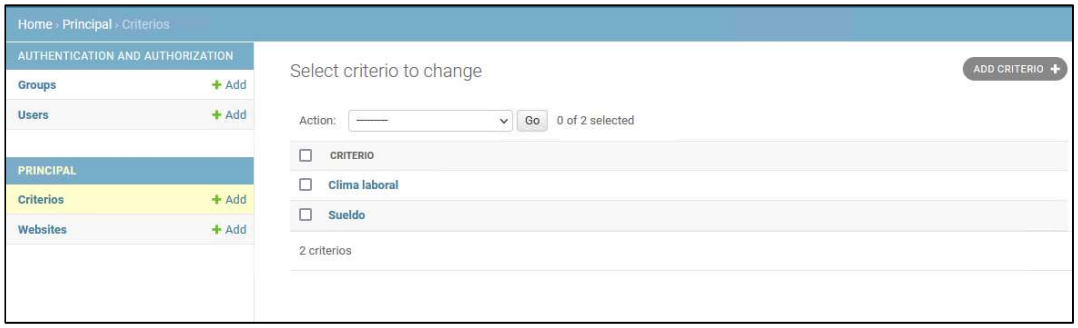


Fig. 10 – Pantalla de selección de criterios para edición

Fig. 11 – Pantalla de modificación de criterios

Por otro lado, los usuarios regulares cuentan con un panel que permite ejecutar las labores correspondientes a las etapas de ETL (extracción, transformación y carga de datos) desde un panel web, según se ilustra en las Figuras 12 a 14.

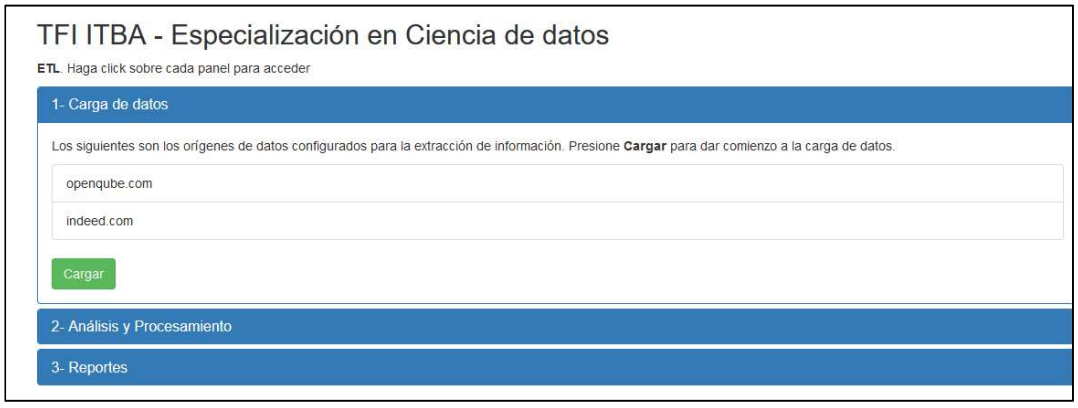


Fig. 12 – Pantalla de carga de dato para usuarios regulares

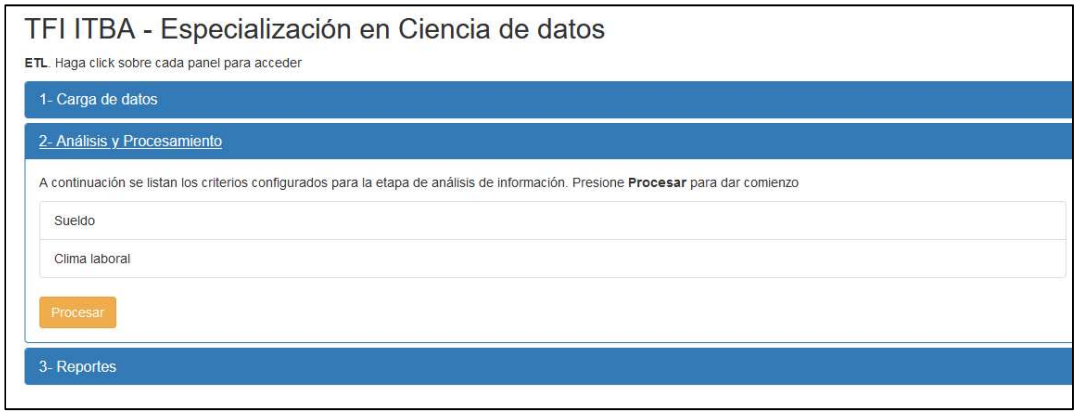


Fig. 13 – Pantalla de análisis para usuarios regulares

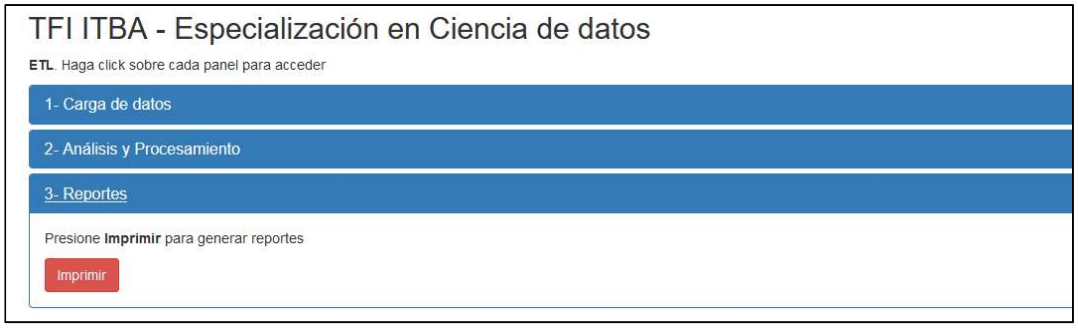


Fig. 14 – Pantalla de reportes de resultados para usuarios regulares

Además de Python Django el sitio emplea **Bootstrap 3.4.1**, según puede evidenciarse en la siguiente captura de código de **index.html:**

```
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></s
cript>
```

En las Secciones 12.10 a 12.12 del Anexo, se pueden consultar más detalles.

9. Resultados Experimentales

Para la ejecución de los casos experimentales, se ha modificado el input que utiliza el módulo cluster.py y eventualmente se han acotado en algunos casos los criterios guía a los efectos de ilustrar mejor al lector.

Como se mencionó con anterioridad el valor de K se calcula de forma automática en función de la cantidad de tópicos distintos detectados en el dataset conjuntamente con el tipo de valoración asociada, positiva o negativa.

Los labels o criterios guía cargados en nuestro sistema son los siguientes, los cuales reciben numeración de 0 a N. En esta primera ejecución tenemos lo siguiente:

- salary (0)
- work-environment (1)
- bosses (2)

Caso 1

Ejecutamos data_pump.py con 13 registros, y obtuvimos una salida como la siguiente:


```
{"topic": [0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 1, 1, 1], "opinion": [0.705, -0.5409, -0.4215, 0.8363, 0.6124, 0.0, -0.2446, 0.4215, 0.6808, 0.0, -0.0754, -0.6249, -0.5423]}
```


Podemos observar que tenemos tres tópicos: salary (0), work-environment (1) y bosses (2). El gráfico resultante se puede ver en la Figura 15. En el eje *x* se tienen los tópicos, mientras que sobre el eje *y* estan las valoraciones de las opiniones capturadas.

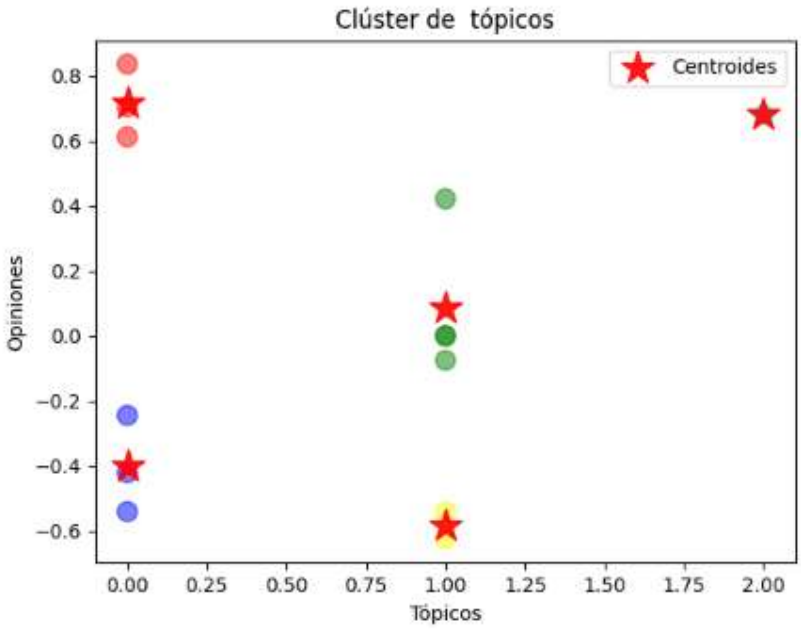


Fig. 15 – Tópicos y opiniones para el caso 1

Para el valor 0 (salary) en el eje x tenemos dos grupos: Los que opinan mal acerca del salario, representados en color azul y los que opinan bien, representados en color rojo.

Para el valor 1 (work-environment) en el eje x tenemos dos grupos: Los que opinan mal acerca de work-environment, representados en amarillo, y las opiniones neutrales a buenas, representadas en verde.

Finalmente, para el valor 2 (bosses) en el eje x tenemos un solo grupo: Los que opinan bien, el dataset en este caso cuenta con una sola opinión positiva en este caso.

Las estrellas rojas en todos los casos representan los centroides. Para este ejemplo se armaron 5 grupos o clústeres.

Caso 2

Para el segundo caso, se seleccionaron aquellas opiniones con valoración negativa en lo que refiere a work-space (1).

El input empleado para este caso fue el siguiente:


```
{"topic": [1, 1, 1, 1, 1, 1, 1], "opinion": [-0.0754,-0.6249,-0.5423,-
0.3314,-0.25,-0.2755,-0.3182]}
```


El gráfico resultante se puede consultar en la Figura 16.

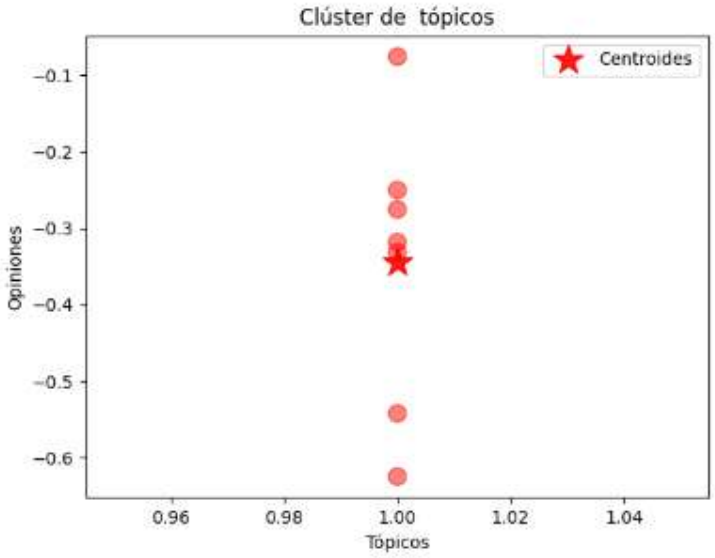


Fig. 16 – Tópicos y opiniones para el caso 2

Como puede observarse, se generó solamente un grupo correspondiente al tópico workspace con valoración negativa, ya que todos los valores del eje *y* son inferiores a cero. Las estrellas rojas en todos los casos representan los centroides.

10. Conclusiones y Trabajo a Futuro

En la actualidad, uno de los mayores capitales de una organización está dado por sus empleados. Para poder detectar su grado de satisfacción, que se refleja directamente en el desempeño global de la empresa, hay muchos métodos novedosos, que se desprenden del advenimiento de la era Big Data, y que no necesariamente están representados por una encuesta de satisfacción tradicional.

La minería de opiniones (text mining) y el análisis de sentimientos (sentiment analysis), aplicados a un reporte libre, a opiniones en redes sociales, o a emails, adquieren relevancia en la realización de un diagnóstico.

En este trabajo, hemos evidenciado que la utilización de análisis de sentimientos (por medio de la librería ntlk de Python), en conjunto con la técnica One Shot Classification que toma como base el modelo pre-entrenado ROBERTA, constituyen una solución viable para problemas que implican la categorización de opiniones por valoración, ya que por medio de etiquetas o criterios guía, puede identificarse a que tópico en específico hace referencia una sentencia y si la opinión asociada es positiva o negativa.

Un paso más allá, sería utilizar otros modelos basados en BERT (Bidirectional Encoder Representations from Transformers) y comparar resultados con los expuestos en este trabajo.

Adicionalmente, también buscaremos añadir a la solución planteada en este trabajo, APIs de terceros para la extracción de información de sitios y aplicaciones, con el objeto de enriquecer los resultados obtenidos.

Como mejora importante a futuro, proponemos trabajar en optimizar la captura de información, paralelizando la recopilación realizada por web-scraping, a los efectos de colectar mayor cantidad de datos en menor tiempo.

11. Referencias

Bagnato, J. (2020). *Aprende Machine Learning en Español: Teoría + Práctica Python.* ISBN 8409258161. ISBN-13 978-8409258161

Costa, A. & Veloso, A. (2015). *Employee Analytics through Sentiment Analysis*. Brazilian Symposium on Databases. Petrópolis-RJ-Brazil. doi: 10.13140/RG.2.1.1623.3688.

Dabirian, A., Kietzmann, J. & Diba, H. (2016). *A Great Place to Work!? Understanding Crowdsourced Employer Branding*. Business Horizons, 60(2), pp. 197–205. doi: 10.1016/j.bushor.2016.11.005.

Luo, N., Zhou, Y. y Shon, J. (2016). *Employee Satisfaction and Corporate Performance: Mining Employee Reviews on Glassdoor.com*. Thirty Seventh International Conference on Information Systems, Dublin. Recuperado de https://core.ac.uk/download/pdf/301370386.pdf.

Piersanti, M., Brandetti, G. & Failla, P. (2017). *Automatic Evaluation of Employee Satisfaction*. Italian Conference on Computational Linguistics, Roma, Italia. Recuperado de http://ceur-ws.org/Vol-2006/paper003.pdf.

Shami, N., Muller, M., Pal, A., Masli, M. & Geyer, W. (2015). *Inferring Employee Engagement from Social Media.* Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems-CHI 2015. Seoul,Korea. doi:10.1145/2702123.2702445.

12. Anexos

El código fuente del trabajo se encuentra subido a https://github.com/mabordon/tpfinal en el branch **develop.**

También queda indicado en esta sección el código fuente utilizado por la aplicación desarrollada.

12.1. Módulo ltbatools.py

Se trata de una librería que provee funciones auxiliares como registro en logs y conexiones a diferentes apis utilizadas en el proceso.

```
import json
import sys
import logging
import os.path
from singleton import Service

services={"api_translator":"apitranslatorconfig.json","comentarios_db":"d
bconfig.json","firefox":"firefox.json"}

class PropertyHook(metaclass=Service):
            def __init__(self,type):
                        self.type=type
            def __load_properties(self,inifile):
                 with open(inifile,'r') as
file:
                      self.__dict__=dict((json.load(file)).items())

            @classmethod
            def get_instance(cls,servicetype):
                      _instance=None
                      if servicetype in services:
                            _instance=PropertyHook(servicetype)
                            _instance.__load_properties(services[servic
etype])
                      return _instance

def extract_digits(input):
      number=''.join(filter(str.isdigit, input))
      if number.isdigit():
           return int(number)
      else:
```

```
            return 0

def get_db_property_hook():
         return PropertyHook.get_instance("comentarios_db")

def get_api_translator_property_hook():
        return PropertyHook.get_instance("api_translator")

def get_firefox_driver_hook():
         return PropertyHook.get_instance("firefox")

def get_itba_logger(logname,screen=False):

          def find_handler(handlername,isfile=False):
              if isfile:
                 items=list(filter(lambda
x:os.path.basename(x.stream.name)==handlername,l.handlers))

              else:
                 items=list(filter(lambda
x:x.stream.name==handlername,l.handlers))

              return len(items)>0

          logging.basicConfig(level=logging.DEBUG,handlers=[])
          l= logging.getLogger(logname)
          formatter=logging.Formatter('%(asctime)s - %(name)s -
%(levelname)s - %(message)s')

          if not find_handler("{0}.log".format(logname),isfile=True):

              file_handler =
logging.FileHandler(filename="logs/{0}.log".format(logname))
              file_handler.setFormatter(formatter)
              l.handlers.append(file_handler)

          if screen and not find_handler("<stdout>"):

               stdout_handler = logging.StreamHandler(sys.stdout)
               stdout_handler.setFormatter(formatter)
               l.handlers.append(stdout_handler)

          return logging.getLogger(logname)
```

12.2. Clase Singleton.py

Permite disponibilizar los diferentes servicios como única instancia en memoria (patrón Singleton)

```
class Singleton(type):
                __cls__={}
                def __call__(cls,*args,**kwargs):

                                if cls not in Singleton.__cls__:
                                                Singleton.__cls__[cls]=type.__call
__(cls,*args,**kwargs)
                                return Singleton.__cls__[cls]

class Service(type):
                __cls__={}
                def __call__(cls,*args,**kwargs):
                                service_type="{0}_{1}".format(cls.__name__,args
[0])
                                if service_type not in Service.__cls__:
                                                Service.__cls__[service_type]=type
.__call__(cls,*args,**kwargs)
                                return Service.__cls__[service_type]
```

12.3. Script Indeedscraper.py

Aplica webscraping sobre la página Indeed recolectando opiniones de los usuarios en la base de datos mongo.db previa traducción al idioma inglés.

```
from selenium import webdriver
import time
import json
from selenium.webdriver.firefox.options import Options
opts = Options()
opts.headless = True
assert opts.headless
from itbatools import get_firefox_driver_hook,extract_digits
from singleton import Singleton
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.proxy import Proxy, ProxyType
from proxyservice import ProxyPool
from translatorapi import TranslatorApi
from comentarioDB import ComentarioDB
```

```
__website__="ar.indeed.com"
__url__="https://{0}/cmp/Ntt-
Data/reviews?fcountry=ALL".format(__website__)
__sleeptime__=2
__max_score__=5
__page_size__=20

translator=TranslatorApi.get_instance()
dbComentarios=ComentarioDB.getDataBase()

class IndeedScraper(metaclass=Singleton):
      def __init__(self,proxy):
            self.driver=webdriver.Firefox(executable_path=
get_firefox_driver_hook().executable_path,options=opts, proxy=proxy)
            self.baseurl =__url__
            self.driver.get(self.baseurl)

      @classmethod
      def get_instance(cls,proxy):
            return IndeedScraper(proxy)

      def save_ratings_summary(self):
                titles=self.driver.find_elements_by_class_name("css-
7bbylr")
                ratings=self.driver.find_elements_by_class_name("css-
1vmx0e0")
                company_ratings={}
                for title, rating in zip(titles,ratings):
                     company_ratings[title.text]=rating.text
                company_ratings["website"]=__website__
                company_ratings["maxscore"]=__max_score__
                print(company_ratings)
                dbComentarios.insert_comentario(company_ratings)

      def print_comments(self):
                     reviews_counter=self.get_reviews_counter()
                     counter=__page_size__
                     times=(reviews_counter //__page_size__)
                     print("La cantidad de iteraciones
sera",times)
                     for page in range(0,times):
                             self.add_comments()
                             time.sleep(__sleeptime__)

                             url=self.baseurl+"&start={0}".format(counter
)
```

```
                        print(url)
                        self.driver.get(url)
                        counter+=__page_size__

    def get_reviews_counter(self):
        #Recupera la cantidad de reviews de la pagina
        result=0
        div=self.driver.find_element_by_class_name("css-r5p2ca")
        if (div):
            result=extract_digits(div.find_element_by_tag_name("span")
.text)
        return result
    def add_comments(self):
          comments= self.driver.find_elements_by_class_name("css-
e6s05i")
          for comment in comments:
                  page_comment={}
                  item_title=comment.find_elements_by_class_name("css-
i1omlj")
                  title=""
                  if len(item_title)>0:
                          title=item_title[0].find_element_by_class_n
ame("css-82l4gy").text
                          page_comment["title"]=json.loads((translato
r.translate(title)).text)["text"][0]
                  item_body=comment.find_elements_by_class_name("css-
rr5fiy")
                  if len(item_body)>0:
                        elem=item_body[0].find_elements_by_class_name(
"css-1cxc9zk")
                        page_comment["body"]=""
                        if len(elem)>0:
                            paragraphs=elem[0].find_elements_by_xpath
("span/span")
                            for p in paragraphs:
                                page_comment["body"]+=p.text
                        else:
                             elem=item_body[0].find_elements_by_class
_name("css-qodkr")
                             if
len(elem)>0:
                                  paragraphs=elem[0].find_elements_b
y_xpath("span/span")
                                  for p in paragraphs:
                                      page_comment["body"]+=p.text
```

```
                            if "body" in
page_comment.keys():
                                page_comment["body"]=json.loads((translat
or.translate(page_comment["body"])).text)["text"][0]
                        if len(item_body)>1:
                                procons_title=item_body[1].find_elements_b
y_tag_name("h2")
                                div=item_body[1].find_elements_by_class_na
me("css-1z0411s")

                                if (len(div)>0):
                                    pro=div[0].find_element_by_xpath("sp
an/span").text
                                    page_comment["Pros"]=pro

                                if
(len(div)>1):
                                    cons=div[1].find_element_by_xpath("sp
an/span").text
                                    page_comment["Cons"]=cons
                                else:
                                    div=item_body[1].find_elements_by_cla
ss_name("css-1jysqrt")
                                    if len(div)>0:
                                        pro=div[0].find_element_by_xpath
("span/span").text
                                        page_comment["Pros"]=pro

                                    if
(len(div)>1):
                                        cons=div[1].find_element_by_xpath
("span/span").text
                                        page_comment["Cons"]=cons

                                page_comment["website"] =__website__
                                if "Pros" in page_comment.keys():
                                    page_comment["Pros"]=json.loads((trans
lator.translate(page_comment["Pros"])).text)["text"][0]

                                if "Cons" in page_comment.keys():
                                    page_comment["Cons"]=json.loads((trans
lator.translate(page_comment["Cons"])).text)["text"][0]

                                dbComentarios.insert_comentario(page_comme
nt)
def process_indeed():
```

```
        pool = ProxyPool.get_instance()
        pool.refresh()
        myProxy = pool.get_next()
        proxy = Proxy({
                 'proxyType': ProxyType.MANUAL,
                 'httpProxy': myProxy,
                 'ftpProxy': myProxy,
                 'sslProxy': myProxy,
                 'noProxy': ''
                 })

        instance=IndeedScraper.get_instance(proxy)
        instance.save_ratings_summary()
        dbComentarios.delete_many({"website":__website__})
        instance.print_comments()


if __name__=='__main__':
         process_indeed()
```

12.4. Script openqubescraper.py

Aplica webscraping sobre la página openqube.com, recolectando opiniones de los usuarios en la base de datos mongo.db previa traducción al idioma inglés

```
from selenium import webdriver
import time
import json
from selenium.webdriver.firefox.options import Options
opts = Options()
opts.headless = True
assert opts.headless
from itbatools import get_firefox_driver_hook
from singleton import Singleton
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.proxy import Proxy, ProxyType
from proxyservice import ProxyPool
from itbatools import extract_digits
from translatorapi import TranslatorApi
from comentarioDB import ComentarioDB
__website__="openqube.io"
__url__="https://{0}/company/everis/".format(__website__)
__sleeptime__=2
__records__=2
```

```
__max_score__=10

translator=TranslatorApi.get_instance()
dbComentarios=ComentarioDB.getDataBase()

class OpenQubeScraper(metaclass=Singleton):
      def __init__(self,proxy):
            self.driver= webdriver.Firefox(executable_path=
get_firefox_driver_hook().executable_path,options=opts, proxy=proxy)
            self.baseurl =__url__
            self.driver.get(self.baseurl)
      @classmethod
      def get_instance(cls,proxy):
            return OpenQubeScraper(proxy)
      def save_rating_summary(self):
          ratings={}
          company_ratings=self.get_rating_summary()
          rating_item=json.loads(translator.translate(json.dumps(company_
ratings)).text)["text"][0]
          ratings["website"]="openqube.io"
          ratings["rating"]=rating_item
          dbComentarios.insert_comentario(ratings)
      def get_rating_summary(self):
                          #Recuperar los puntajes del
sitio
                          sections=self.driver.find_elements_by_class_name
("company__item-rating")
                          company_ratings={}
                          company_ratings["website"]=__website__
                          company_ratings["maxscore"]=__max_score__
                          for section in sections:
                                  aspect=section.find_elements_by_class_n
ame("company__item-ratingtxt")[0].text
                                  rating=section.find_elements_by_class_n
ame("company__item-ratingnum")[0].text
                                  company_ratings[aspect]=rating
                          return company_ratings
      def add_comments(self):
            lista=
self.driver.find_elements_by_class_name("reviewlist__items")
            items=self.driver.find_elements_by_class_name("reviewlist__t
itle")
            __counter__=(extract_digits(items[0].find_element_by_tag_nam
e("span").text)//__records__)+1
            print(__counter__)
            for scroll_index in range(0,__counter__):
```

```
                    button=self.driver.find_element_by_class_name("viewmo
re__button")
                    time.sleep(__sleeptime__)
                    self.driver.execute_script("arguments[0].click();",
button)
             articles=
self.driver.find_elements_by_class_name("review")
             for article in articles:
                     textos=
article.find_elements_by_class_name("review__text")
                    comment={}
                    for texto in textos:
                       procons=texto.find_element_by_tag_name("strong")
                       item= texto.find_element_by_tag_name("span")
                       if len(item.text)>0:
                          translated_comment=json.loads((translator.tran
slate(item.text)).text)["text"][0]
                          comment[procons.text]= translated_comment
                          comment["website"]
=__website__
                    dbComentarios.insert_comentario(comment)


      def get_comments(self):
             comments=[]
             lista=
self.driver.find_elements_by_class_name("reviewlist__items")
             items=self.driver.find_elements_by_class_name("reviewlist__t
itle")
             __counter__=(extract_digits(items[0].find_element_by_tag_nam
e("span").text)//__records__)+1
             print(__counter__)
             for scroll_index in range(0,__counter__):
                    button=self.driver.find_element_by_class_name("viewmo
re__button")
                    time.sleep(__sleeptime__)
                    self.driver.execute_script("arguments[0].click();",
button)
             articles= self.driver.find_elements_by_class_name("review")
             for article in articles:
                     textos=
article.find_elements_by_class_name("review__text")
                    comment={}
                    for texto in textos:
                       procons=texto.find_element_by_tag_name("strong")
                       item= texto.find_element_by_tag_name("span")
```

```
                    if len(item.text)>0:
                        comment[procons.text]=item.text
                comments.append(comment)
        return comments

def process_open_qub():
        pool = ProxyPool.get_instance()
        pool.refresh()
        myProxy = pool.get_next()
        proxy = Proxy({
                 'proxyType': ProxyType.MANUAL,
                 'httpProxy': myProxy,
                 'ftpProxy': myProxy,
                 'sslProxy': myProxy,
                 'noProxy': ''
                 })
        instance=OpenQubeScraper.get_instance(proxy)
        print(instance.get_rating_summary())
        print("Borrando")
        dbComentarios.delete_many({"website":__website__})
        print("Fin borrado")
        instance.save_rating_summary()
        print("Guardando comentarios")
        instance.add_comments()


def print_comentarios():
       pool = ProxyPool.get_instance()
       pool.refresh()
       myProxy = pool.get_next()
       proxy = Proxy({
                 'proxyType': ProxyType.MANUAL,
                'httpProxy': myProxy,
                'ftpProxy': myProxy,
                 'sslProxy': myProxy,
                 'noProxy': ''
                 })
       instance=OpenQubeScraper.get_instance(proxy)
       print(instance.get_rating_summary())
       print("Imprimiendo comentarios....")
       for items in instance.get_comments():
            print(items)

if __name__=='__main__':
         print_comentarios()
```

12.5. Script proxyserver.py

La clase ProxyPool provee al proceso de relector una serie de proxies para realizar la colecta de información.

```
import requests
from itertools import cycle
from proxyscraper import ProxyScraper
from singleton import Singleton

class ProxyPool(metaclass=Singleton):
      def __init__(self):
               self.proxyscraper=ProxyScraper.get_instance()
               self.proxies_pool=cycle(self.proxyscraper.get_proxy_list()
)
      def get_next(self):
                available_proxy=next(self.proxies_pool)
                return available_proxy
      def refresh(self):
            self.proxies_pool=cycle(self.proxyscraper.refresh_server_list
())
      @classmethod
      def get_instance(cls):
            return ProxyPool()



if __name__=='__main__':
         _instance=ProxyPool()
         proxy=_instance.get_instance().get_next()
         print(proxy)
```

12.6. Script proxyscraper.py

Recupera de https://free-proxy-list.net/ los proxies que la aplicación puede utilizar. Es utilizada por la clase ProxyPool

```
import requests
from itertools import cycle
from singleton import Singleton
from lxml.html import fromstring

class ProxyScraper(metaclass=Singleton):
             def __init__(self):
                    self.url = 'https://free-proxy-list.net/'
```

```
                self.proxies=[]
            def refresh_server_list(self):
                self.proxies=self.start_scraping()
                return self.proxies
            def start_scraping(self):
                response = requests.get(self.url)
                parser = fromstring(response.text)
                proxies =[]
                for attribute in parser.xpath('//tbody/tr'):
                    if
(attribute.xpath('.//td[7][contains(text(),"yes")]')):
                        proxy =
":".join([attribute.xpath('.//td[1]/text()')[0],
attribute.xpath('.//td[2]/text()')[0]])
                        proxies.append(proxy)

                return proxies

            @classmethod
            def get_instance(cls):
               return ProxyScraper()
            def get_proxy_list(self):
                self.refresh_server_list()
                return self.proxies
if  __name__=='__main__':
       _instance=ProxyScraper.get_instance()
       print(_instance.get_proxy_list())
```

12.7. Script translator.py

Se encarga de realizar las traducciones de los comentarios obtenidos antes de proceder con el almacenaje de los mismos en la base de datos mongo.db. Detrás de escena se utiliza la api **https://translate.yandex.net/api/v1.5/tr.json/translate**

```
import requests
import json
from itbatools import get_api_translator_property_hook
from singleton import Singleton
class TranslatorApi(metaclass=Singleton):
      def __init__(self):
            self._propertyhook=get_api_translator_property_hook()
      def translate(self,texto):
```

```
                response = requests.request("GET",
self._propertyhook.url,

                                                    params={"lang":self._p
ropertyhook.language,
                                                            "key":self._pr
opertyhook.key,
                                                            "text":texto
})

                return response
      @classmethod
      def get_instance(cls):
            return TranslatorApi()

if __name__=='__main__':
         t=TranslatorApi.get_instance()
         mensaje="{'mensaje': Ambiente di lavoro ottimo'}"
         resultado=t.translate(mensaje)
         mensaje=json.loads(resultado.text)
         print(mensaje)
```

12.8. Script comentarioDb.py

Expone la clase homónima, la cual se conecta contra la base datos mongoDb para llevar a cabo la inserción de comentarios.

```
from pymongo import MongoClient
from itbatools import get_db_property_hook
from singleton import Singleton

class ComentarioDB(metaclass=Singleton):
    def __init__(self,_dbconfigurator):
           self._mongoClient = MongoClient(_dbconfigurator.host,
                                           _dbconfigurator.port)
           self._db=self._mongoClient.comentario
           self._comentarios=self._db.comentarios
    def insert_comentario(self,comentario):
        self._comentarios.insert_one(comentario)
    def delete_comentario(self,comentario):
        self._comentarios.delete_one(comentario)
    def delete_many(self,criteria):
        self._comentarios.delete_many(criteria)
    def printrecords(self):
            for comentario in self._comentarios.find({}):
```

```
                print(comentario)
    def test_add(self):
            comentario={"body":"Hello"}
            self.insert_comentario(comentario)
    def test_delete(self):
            comentario={"body":"Hello"}
            self.delete_comentario(comentario)
    @classmethod
    def getDataBase(cls):
       _dbconfigurator=get_db_property_hook()
       return ComentarioDB(_dbconfigurator)



if __name__=='__main__':
        db=ComentarioDB.getDataBase()
        db.test_delete()
        db.test_add()
        db.printrecords()
```

12.9. Script sentimentanalyzer.py

Tiene por finalidad llevar a cabo el análisis de sentimientos, dado un texto determinado que se proporciona para esta finalidad.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk import sentiment
from nltk import word_tokenize
import nltk
import nltk

def tokenize(paragraph):
    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    sentences = tokenizer.tokenize(paragraph)
    return sentences

def get_compound(scores):
      return scores["compound"]

def analyze(sentence):

   _analyzer = SentimentIntensityAnalyzer()
   scores = _analyzer.polarity_scores(sentence)
   return scores
```

```
if __name__=='__main__':
      sentences=tokenize("The environment is so stressful. The salary is
good")
      for sentence in sentences:
            print(analyze(sentence))
```

12.10. Archivo admin.py

Este archivo es utilizado para registrar aquellas clases en la consola administrativa que provee Django para realizar operaciones de alta, baja, modificación:

```
from django.contrib import admin
from .models import Criterio,Website
# Register your models here.

admin.site.register(Criterio)
admin.site.register(Website)
```

12.11. Archivo models.py

En este archivo se incluyen dos clases, utilizadas por parte de la aplicación que manejan los usuarios.

```
class Criterio(models.Model):

    id=models.AutoField(primary_key=True)
    nombre=models.CharField(max_length=200)

    def __str__(self):
         return self.nombre

class Website(models.Model):
      id=models.AutoField(primary_key=True)
      nombre=models.CharField(max_length=200)
      descripcion=models.CharField(max_length=200)
      def __str__(self):
           return self.nombre
```

12.12. Archivo apps.py

En este archivo se le indica al framework de Django el nombre de la aplicación utilizada. En nuestro caso se denomina **aplicaciones.principal**

```
from django.apps import AppConfig

class PrincipalConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'aplicaciones.principal'
```

12.13. Módulo topicanalizer.py

Es invocado por text_analyzer.py y tiene por finalidad determinar a qué label o etiqueta informada corresponde, utilizado zero-shot-classification.

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")

def analyze(sequence,candidate_labels):
   result=classifier(sequence, candidate_labels)
   return result

if __name__=='__main__':
   sequence = "The projects are very interesting"
   candidate_labels = ["salary", "work-environment", "bosses"]
   result=analyze(sequence,candidate_labels)
   print(result)
```

12.14. Módulo text_analyzer.py

Es invocado por data_pump.py con la finalidad de poder obtener a partir de una opinión determinada a que tópico o label hace referencia y el análisis de sentimiento correspondiente, el resultado se devuelve en el diccionario llamado results. Se invoca a topic_analizer.py y a sentimientanalyzer.py para recuperar la información antes mencionada.

```
from unittest import result
import topicanalizer
import sentimentanalyzer

def analyze(phrase, candidates):
```

```
      sentences=sentimentanalyzer.tokenize(phrase)
      results={}
      for sentence in sentences:
            feelings_results=sentimentanalyzer.analyze(sentence)
            topic_result=topicanalizer.analyze(sentence,candidates)
            detected_topic=topic_result["labels"][0]
            compound=sentimentanalyzer.get_compound(feelings_results)

            results["topic"]=detected_topic
            results["feeling"] = compound
            results["phrase"]=sentence
      return results
if __name__=='__main__':
       phrase = "The managers are wonderful. The chances are good"
       candidate_labels = ["salary", "work-environment",
"bosses","progress"]
       analyze(phrase,candidate_labels)
```

12.15. Módulo data_pump.py

Recupera los comentarios de la base de datos y por medio de la invocación del módulo text_analyzer.py puede recuperar a que tópico refiere cada opinión almacenada en la base de datos mongo db y adicionalmente obtener a partir del análisis de sentimientos el compound por cada opinion emitida. Se genera un archivo de salida llamado './clusterprocessor/input/bajada.json' para que sea tomado como input por parte del módulo processor.py.

```
from comentarioDB import ComentarioDB
import text_analyzer
import json


def process(canditate_labels):
        db=ComentarioDB.getDataBase()
        corpus=None
        topiclist=[]
        opinionlist=[]
        rowindex=0
        for comment in db.get_all_records():
              if "title" in comment.keys():
                    title=comment["title"]
              if "body" in comment.keys():
                      corpus=comment["body"]
              if "Pros" in comment.keys():
                      corpus=comment["Pros"]
```

```
                if "Cons" in comment.keys():
                        corpus=comment["Cons"]
                if corpus:
                        results_cons=text_analyzer.analyze(corpus,candidat
e_labels)
                        print(results_cons)
                        indice=candidate_labels.index(results_cons['topic'
])
                        topiclist.append(indice)
                        opinionlist.append(results_cons['feeling'])

                        rowindex+=1
        dict_object={"topic":topiclist,"opinion":opinionlist}
        json_string=json.dumps(dict_object)
        print(json_string)
        jsonFile = open("./clusterprocessor/input/bajada1.json", "w")
        jsonFile.write(json_string)
        jsonFile.close()

if __name__=='__main__':
         candidate_labels = ["salary", "work-environment",
"bosses","facilities"]
         process(candidate_labels)
```

12.16. Módulo processor.py

Invoca a cluster.py el cual tiene por finalidad realizar la clusterización.

```
from cluster import process

if __name__=='__main__':
       process()
```

12.17. Módulo cluster.py

Tiene por finalidad ejecutar el cluster. Toma como input el archivo generado en la ruta './clusterprocessor/input/bajada.json' y se almacena el gráfico generado en './clusterprocessor/output/cluster.png'. Es invocado desde el archivo processor.py.

```
import matplotlib
import numpy as np
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import json

input='./clusterprocessor/input/bajada.json'
output='./clusterprocessor/output/cluster.png'

def groupbytopicsandopinion(topiclist,opinions):
      results={}
      for topic, opinion in zip(topiclist,opinions):
            if not topic in results:
                  results[topic]=[opinion]
            else:
                  results[topic].append(opinion)
      print("Imprimiendo resultados")
      print(results)
      return results

def calculate_summarized(results):
       summarized={}
       for topic in results:
         lista=results[topic]
         print("Imprimiendo la lista")
         print(topic,lista)
         summarized[topic]=0
         neg_count = len(list(filter(lambda x: (x < 0), lista)))
         pos_count = len(list(filter(lambda x: (x >= 0), lista)))
         if neg_count>0:
              summarized[topic]+=1
         if pos_count>0:
              summarized[topic]+=1
         print("Estadisticas=>",topic,summarized[topic])
       return summarized

#Devolvemos la cantidad de clusters
def calculate_k(topiclist, opinions):
      results= groupbytopicsandopinion(topiclist,opinions)
```

```
        summarized=calculate_summarized(results)
        k=0
        for topic in summarized:
             k+=summarized[topic]
        print(k)
        return k


def make_cluster(Data):
        df = DataFrame(Data,columns=['topic','opinion'])
        kmeans =
KMeans(n_clusters=calculate_k(Data['topic'],Data['opinion'])).fit(df)
        centroids = kmeans.cluster_centers_
        print(centroids)
        print(kmeans.labels_)
        colores=['red','green','blue','cyan','yellow','violet','orange','b
lack']
        asignar=[]
        for row in kmeans.labels_:
             asignar.append(colores[row])
        plt.scatter(df['topic'], df['opinion'], marker="o",s=100,
alpha=0.5,c=asignar)
        plt.scatter(centroids[:, 0], centroids[:, 1],marker = "*", alpha =
0.9,s=300,c='red',label='Centroides')
        plt.title('Clúster de  tópicos')
        plt.xlabel('Tópicos')
        plt.ylabel('Opiniones')
        plt.legend()
        plt.savefig(output)
def process():
          data = open(input,)
          data = json.load(data)
          make_cluster(data)
          print("Done")


if __name__=='__main__':
                process()
```

12.18. Librerías python para la aplicación

A continuación se listan las librerías empleadas para la aplicación. Esto implica desde la captura de información por medio del webscraping, la aplicación Django y la generación de la bajada de información realizada por data_pump.py. Se generó por medio de conda un entorno virtual llamado tpfinal conteniendo las siguientes librerías dicho listado se obtiene ejecutando la siguiente linea de comando desde conda prompt, en el entorno virtual mencionado: **pip list --format=freeze**

absl-py==1.0.0
asgiref==3.4.1
astunparse==1.6.3
atomicwrites==1.4.0
attrs==21.2.0
blinker==1.4
brotlipy==0.7.0
cachetools==4.2.4
certifi==2022.5.18.1
cffi==1.15.0
charset-normalizer==2.0.4
click==8.0.4
colorama==0.4.4
cryptography==3.4.8
Django==3.2.5
filelock==3.7.0
gast==0.3.3
google-auth==1.35.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.46.1
h5py==2.10.0
idna==3.3
importlib-metadata==4.11.3
iniconfig==1.1.1
joblib==1.1.0
Keras-Preprocessing==1.1.2
lxml==4.7.1
Markdown==3.3.7
mkl-fft==1.3.1
mkl-random==1.2.2
mkl-service==2.4.0
more-itertools==8.12.0
nltk==3.7
numpy==1.22.4
oauthlib==3.1.1

opt-einsum==3.3.0
packaging==21.3
pandas==1.4.2
pip==21.2.2
pluggy==0.13.1
protobuf==3.20.1
psycopg2==2.8.6
py==1.10.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.21
PyJWT==2.1.0
pymongo==3.12.0
pyOpenSSL==21.0.0
pyparsing==3.0.4
PySocks==1.7.1
pytest==6.2.4
pytest-base-url==1.4.2
pytest-html==3.1.1
pytest-metadata==1.11.0
pytest-selenium==2.0.1
pytest-variables==1.9.0
python-dateutil==2.8.2
pytz==2021.3
regex==2022.1.18
requests==2.26.0
requests-file==1.5.1
requests-ftp==0.3.1
requests-oauthlib==1.3.0
requests-toolbelt==0.9.1
rsa==4.8
sacremoses==0.0.53
scikit-learn==1.0.2
scipy==1.4.1
selenium==3.141.0
sentencepiece==0.1.96
setuptools==62.3.2
six==1.16.0
sqlparse==0.4.1
tenacity==6.3.1
tensorboard==2.2.2
tensorboard-plugin-wit==1.8.1
tensorflow==2.2.0
tensorflow-estimator==2.2.0
termcolor==1.1.0

threadpoolctl==2.2.0
tokenizers==0.8.1rc2
toml==0.10.2
tqdm==4.63.0
transformers==3.1.0
typing-extensions==3.10.0.2
urllib3==1.26.7
Werkzeug==2.1.2
wheel==0.37.0
win-inet-pton==1.1.0
wincertstore==0.2
wrapt==1.14.1
zipp==3.8.0

12.19. Librerías python para el proceso de cluster

Para la ejecución del clúster como proceso, se generó por medio de **conda** un entorno virtual llamado tpfinal conteniendo las siguientes librerías dicho listado se obtiene ejecutando la siguiente linea de comando desde conda prompt, en el entorno virtual mencionado: **pip list --format=freeze**

aniso8601==9.0.1
Bottleneck==1.3.4
certifi==2022.5.18.1
click==8.0.4
colorama==0.4.4
cycler==0.11.0
Flask==2.0.3
Flask-JSON==0.3.4
Flask-RESTful==0.3.8
fonttools==4.25.0
itsdangerous==2.0.1
Jinja2==3.0.3
joblib==1.1.0
kiwisolver==1.3.2
MarkupSafe==2.0.1
matplotlib==3.5.2
mkl-fft==1.3.1
mkl-random==1.2.2
mkl-service==2.4.0
msvc-runtime==14.29.30133
munkres==1.1.4
numexpr==2.8.1
numpy==1.22.4

packaging==21.3
pandas==1.4.2
Pillow==8.0.0
pip==21.2.4
pyparsing==3.0.4
python-dateutil==2.8.2
pytz==2021.3
scikit-learn==1.1.1
scipy==1.8.1
setuptools==61.2.0
sip==4.19.13
six==1.16.0
threadpoolctl==3.1.0
tornado==6.1
Werkzeug==2.0.3
wheel==0.37.1
wincertstore==0.2