Universidad Zaragoza
1542

Trabajo Fin de Grado

Prototipado Rápido de Minería de Texto Usando Campos Conceptuales

Fast Prototyping for Text Mining Using Conceptual Fields

Autor

Israel Solanas Navarro

Director

Gregorio de Miguel Casado

Grado en Ingeniería Informática

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2021

Agradecimientos

Agradezco a mis compañeros y mis profesores por todo el apoyo, la ayuda y las enseñanzas recibidas relativas a los estudios de ingeniería informática.

Aprovecho la ocasión para agradecer al profesor y director de este TFG, Gregorio de Miguel Casado, por su disposición y disponibilidad para dirigirme en este trabajo TFG. En especial agradecer las sugerencias y la ayuda recibidas de su parte.

También quiero agradecer a todas las personas que me han amado, ayudado, aconsejado y corregido hasta el día de hoy; por esto y mucho más gracias a todos los hermanos de la iglesia Vida Nueva Zaragoza. En especial gracias a los pastores Luis y Josué Nasarre y a David Owtschinnikow por su gran ayuda e inversión en mí.

Muy especialmente gracias a mi familia, desde mis abuelos a mi hermano Jonatan y, más aún a mis padres, Francisco y Ana, quienes me han ayudado y cubierto en todos los aspectos hasta hoy. ¡Gracias!

Sobre todo ¡muchas gracias! a Dios, sin quien no sería posible que yo hubiera llegado hasta aquí y porque Él merece toda la gratitud. Mi primer amor en la vida es Dios, y mi vida entera le pertenece. ¡Gloria a Dios!

Resumen

Hoy en día la mayor parte de la información reside en textos. Existen multitud de formas de extraer información de ellos, algunas manuales como la lectura o un análisis sintáctico de una frase tradicional y, otras automáticas como la aplicación de ciertos algoritmos de minería de texto.

Las aplicaciones de minería de textos en el ámbito de la informática son inmensas y de diversos tipos, una de ellas es la clasificación de textos que trata de predecir a qué categoría pertenece un determinado texto. Precisamente este tipo de aplicación, la clasificación de textos, es la que se lleva a la práctica en este trabajo de fin de grado.

Por otro lado, un concepto no muy explotado en el ámbito de la informática y empleando el lenguaje castellano es el de campo conceptual que hace referencia a una serie de palabras relacionadas con una idea. Existen diccionarios conceptuales incluso disponibles en la web como Zirano con el que se pueden encontrar de forma sencilla palabras relacionadas con una determinada idea. En este sentido, los campos conceptuales resultan ser una herramienta muy útil para enriquecer la información de un determinado texto.

El trabajo se ha centrado en el desarrollo de un sistema de minería de texto basado en campos conceptuales para lo cual se han empleado las técnicas de Aprendizaje SVM y árboles de decisión. La implementación de este sistema de minería se ha llevado a cabo en un entorno de prototipado rápido que ha permitido llevar a cabo el desarrollo de forma más amena que de la forma tradicional empleando solamente un lenguaje de programación.

En este trabajo se han desarrollado y evaluado varios prototipos de sistemas, uno de los cuales emplea campos conceptuales que obtiene de Zirano relacionados con ciertos sustantivos (sustantivos comunes) presentes en los textos que se emplean para entrenar los clasificadores basados en SVM y árbol de decisión. La fuente de datos de la que se extraen los textos es Twitter de la que se obtienen tweets de diferentes categorías y de extensión razonablemente uniforme, entre 270 y 280 caracteres.

Tras los experimentos realizados, se ha obtenido que el mejor sistema de minería de textos desarrollado con el conjunto de datos que se ha probado ha obtenido un 100% de acierto en la predicción, empleando árbol de decisión, entrenando con un conjunto de 70 tweets y validando con otro conjunto de otros 30 tweets.

Índice

Capítulo 1 - Introducción

La minería de texto es uno de los ámbitos de aplicación de la minería de datos. La información se extrae de documentos textuales que se estima que suponen “más del 80%” [1] de la información almacenada. Estos documentos pueden ser desde comentarios de productos de una tienda electrónica o mensajes cortos de redes sociales hasta colecciones de páginas web o libros completos. La clasificación de textos en categorías es una de las tareas más representativas en este ámbito. Para ello se emplean algoritmos que son capaces de aprender información de textos etiquetados y después predecir la categoría de nuevos textos analizandolos.

Adicionalmente, dentro del contexto de la clasificación de textos en categorías, el manejo de grandes cantidades de información textual puede resultar complejo sin las herramientas adecuadas para analizar, visualizar y probar de forma sencilla e intuitiva el desarrollo de un clasificador de textos. Por ello en este trabajo se aborda la idea de desarrollar un sistema de minería de textos estudiando las posibilidades de prototipado rápido existentes y el uso de características avanzadas derivadas del conocimiento que se dispone del lenguaje castellano.

1.1 Motivación

Una de las motivaciones principales de este trabajo es estudiar las posibilidades de los entornos de prototipado rápido, empleando la programación visual [2] en un proyecto de software como el de este trabajo, es decir una GUI intuitiva y configurable con elementos funcionales que se pueden interconectar para lograr la salida deseada.

Antes de empezar este trabajo de fin de grado ya se ha trabajado con un entorno de prototipado rápido en el contexto de unas prácticas de una asignatura del último curso del grado de ingeniería informática, llamada “Sistemas de ayuda a la toma de decisiones”.

1.2 Objetivos

El objetivo general del proyecto consiste en desarrollar mediante un entorno de prototipado rápido, un sistema de minería de texto basado en campos conceptuales. Los objetivos concretos a desarrollar serían los siguientes:

1. Estudiar qué técnicas y herramientas se utilizan en Minería de Textos reducidos.
2. Estudiar qué herramientas software de prototipado rápido se pueden emplear en el contexto del proyecto.
3. Diseñar un prototipo, una batería de experimentos de prueba y evaluarlo.

A estos anteriores, pueden añadirse algunos sub objetivos más:

1. Definir un conjunto de datos de trabajo de textos etiquetados en categorías.
2. Clasificar textos como noticias o tweets (textos de tamaño reducido).

1.3 Alcance

El trabajo prevé el estudio del prototipado rápido para minería de textos cortos. El trabajo es ambicioso en el sentido de que aborda la inclusión de conocimiento avanzado sobre el lenguaje castellano en la construcción de clasificadores de texto. La experimentación planteada aborda, de forma reducida, este enfoque y se espera poder generalizar los resultados a otro tipo de clasificadores textuales.

1.4 Contenido del documento

Tras este capítulo introductorio, el Capítulo 2 recoge el estudio del estado del arte en cuanto a los objetivos 1 y 2 planteados en la propuesta del trabajo: se estudian los conceptos del procesamiento del lenguaje natural, la minería de textos así como varias de sus técnicas y, también se estudian distintas herramientas software de prototipado rápido.

A continuación, el Capítulo 3 presenta los requisitos funcionales del sistema de minería de textos a desarrollar, algunas de las principales decisiones de diseño tomadas que afectan a la solución propuesta y, también se describe parcialmente la implementación de algunas de las partes fundamentales de apoyo para el trabajo: sistemas que permiten extraer los textos así como parte de los campos conceptuales que se emplearán en los posteriores experimentos y así mismo un sistema que transforma la información de los campos conceptuales de forma que sea usable para los experimentos.

Seguidamente, el Capítulo 4 versa sobre los experimentos realizados y presenta resultados y conclusiones sobre los mismos. Finalmente, el Capítulo 5 presenta las conclusiones del trabajo, así como posible trabajo a futuro.

Adicionalmente, pueden consultarse los Anexos elaborados. El Anexo 1 aporta información sobre dos bases de datos construidas en diferentes SGBD para almacenar información sobre textos. Después, el Anexo 2 aporta información sobre software externo instalado o empleado y, el Anexo 3 amplía información sobre técnicas de Procesamiento de Lenguaje Natural. Finalmente, los Anexos 4 y 5 proporcionan información más detallada sobre la implementación de los sistemas de apoyo a los experimentos y de los propiamente asociados a los experimentos desarrollados, respectivamente.

Capítulo 2 - Estado del arte

En este apartado se presentan algunos conceptos y técnicas relacionados con el contexto del trabajo así como herramientas software que pueden emplearse para su desarrollo.

2.1. Procesamiento de Lenguaje Natural y Minería de Texto

Este apartado recoge algunas ideas básicas sobre Procesamiento de Lenguaje Natural y, concretamente, lo relativo a la Minería de Texto. Además, también se describen algunas de las técnicas y herramientas habitualmente empleadas en estos campos de conocimiento.

2.1.1 Procesamiento del lenguaje natural

Esta disciplina consiste en procesar datos de algún lenguaje humano, como puede ser el castellano o el inglés, de forma que el computador reconoce cómo se habla. Este conocimiento aprendido sobre el lenguaje tiene diversas aplicaciones como: los chatbots que dan conversación a una persona, traducir “voz a texto”, el reconocimiento del sentido positivo o negativo de un texto en el análisis del sentimiento [3].

Al procesar datos textuales en Procesamiento del lenguaje natural o *PLN* es importante tener en cuenta el conocimiento adquirido con los datos previos. Esto es, en “El regalo que le hicieron fue bueno”, la palabra “que” debería asociarse a “El regalo” procesado anteriormente [3].

En PLN se usan 2 familias de modelos de redes neuronales: las recurrentes (RNN) y las de memoria a largo y corto plazo (LSTM), que incluyen técnicas como:

- La red neuronal recurrente (RNN), evolución de las “completamente conectadas de feedforward”.
- Las redes neuronales de memoria a largo y corto plazo (LSTM) son posteriores a las RNN y trabajan con “series de tiempo”.
- Mecanismo de atención, técnicas más rápidas que las RNN que se aprovechan del paralelismo sobre hardware TPU de Google.
- El modelo “Transformador” (o “*Transformer*” en inglés) que codifica “información” relacionada con “una palabra” en un vector.
- BERT es un modelo de Google basado en *Transformer* muy eficaz que representa las características de la entrada.
- Transformer-XL, modelo de 2019 que reutiliza “la información transmitida de segmentos anteriores” y solventa la “fragmentación del contexto”.
- Los transformadores compresivos, que se inspiran en el cerebro humano, comprimen “activaciones ocultas pasadas” y usan la memoria de “largo” y la de “corto plazo”.

Para más detalle sobre las técnicas de PLN mencionadas anteriormente, puede consultarse el Anexo 3.

2.1.2 Minería de texto

La minería de textos es una parte de la minería de datos que trata de extraer información no explícita en textos [1]. Se han obtenido diferentes respuestas sobre las fases que comprende la minería de textos como por ejemplo [1], [4]. A continuación, se explica una posible descripción de las fases:

- Toma de datos y preprocesamiento

  En la que se obtienen los datos y se tratan hasta poderlos usar con algoritmos [1].

- Entrenamiento o clasificación

  Se aplica un algoritmo que extrae conocimiento de los datos.

- Pruebas o evaluación

  Por último, se valida el entrenamiento realizado y se extraen conclusiones.

En cuanto a las aplicaciones existentes en minería de textos caben destacar las siguientes [1]:

- “Extracción de información”

  Se trata de conseguir “información” de los textos, por ejemplo “información semántica” incluso partiendo de un corpus grande.

- “Análisis de sentimientos”

  “El análisis de sentimientos permite agregar etiquetas a los términos dentro de un texto de acuerdo con un sentimiento. Por ejemplo, se pueden usar las categorías de positivo, negativo y neutral. Existen dos métodos para realizar este procedimiento:”

  - “Modelo predictivo”

    Determina la etiqueta asociada a una palabra empleando un algoritmo de “inteligencia artificial”.

  - “Basado en diccionario”

    Consiste en disponer de un diccionario por cada posible etiqueta. En cada diccionario hay una colección con las palabras que lo forman. Así, a una nueva palabra se le asocia la etiqueta del diccionario donde se encuentra.

- “Clasificación de textos”

  Consiste en “agrupar textos de acuerdo con diferentes categorías”. Para ello se suelen emplear técnicas de “Aprendizaje Supervisado”, como los árboles de decisión y el SVM que se explicarán más adelante.

- “Elaboración de resúmenes”

  Se trata de determinar la idea general subyacente en un conjunto de textos. Se puede realizar de dos forma:

  - “Sumarización extractiva”

    Basado en seleccionar ciertas partes de los textos .

  - “Sumarización abstracta”

    Se crea un “texto nuevo” que puede no residir “en los textos originales”.

- “Visualizaciones”

  De los textos se extraen “características” que posteriormente pueden visualizarse de diferentes maneras. Por ejemplo, una de ellas son las nubes de palabras [1] en las que aparecen las “palabras” de “un texto” y cuanta mayor es su “frecuencia” de aparición en este, más grande se visualizan [5].

2.1.3 Técnicas y herramientas

En este apartado se presentan la técnica de la Bolsa de Palabras, distintas técnicas de Aprendizaje en minería de texto, así como técnicas avanzadas en Enriquecimiento de Texto.

2.1.3.1 La técnica de la Bolsa de Palabras

Los textos son datos no estructurados, lo que dificulta su uso con “algoritmos de aprendizaje automático” ya que previamente estos deben transformarse en “vectores de números” [6].

“Un modelo de bolsa de palabras, [...] es una forma de extraer características del texto para utilizarlas en el modelado, por ejemplo con algoritmos de aprendizaje automático” [6].

Esta técnica se refiere a la conocida en inglés como Bag of Words (BoW) que en esencia permite extraer las distintas palabras presentes en un documento (o conjunto de documentos). Se basa en dos ideas clave [6]:

1. “Un vocabulario de palabras conocidas”.

2. “Una medida de la presencia de palabras conocidas”.

En la práctica para aplicar el modelo de bolsa de palabras pueden distinguirse 3 fases [6]:

1. “Recolección de datos”

   En la que se pueden acumular varios documentos que conforman un corpus documental.

2. “Diseño del vocabulario”

   Consiste en obtener las diferentes palabras que aparecen en el corpus documental.

3. “Creación de los vectores de documentos”

   Se crea un vector de tamaño igual al número de palabras en el vocabulario con una componente por palabra del vocabulario. El valor de cada componente puede consistir simplemente en indicar si dicha palabra aparece o no dentro de un documento con un número o un dato booleano. Se obtendrá un vector de características por cada documento en el que se aplique el BoW y, la idea es que este vector sirva “como entrada o salida a un modelo de aprendizaje automático”.

Gestión del vocabulario

Este modelo tiene limitaciones computacionales de espacio en memoria ya que un corpus puede componerse de muchos documentos y estos tener muchas palabras. El máximo número de palabras distintas que podrían aparecer en un documento en castellano es de 100.000 aproximadamente [7].

Hay técnicas que pueden mejorar el rendimiento de este método, como por ejemplo: filtrar algunas palabras que aparecen habitualmente en un texto pero que no aportan mucho significado o *stopwords* en inglés, corregir errores ortográficos, lematizar palabras, etc. [6].

Otro modelo que da buenos resultados por ejemplo en la “clasificación de la documentación” es el de N-gramas (del inglés, N-grams) en la que se forman grupos de N palabras o “gramos” consecutivos [6].

Puntuación de las palabras

Otras técnicas para establecer la puntuación de cada palabra de cada documento dado un vocabulario son [6]:

- “Recuentos. Cuenta el número de veces que aparece cada palabra en un documento”.
- “Frecuencias. [...] frecuencia con la que aparece cada palabra en un documento de entre todas las palabras del documento”.

Hashing de palabras

Una “función hash [...] asigna datos a un conjunto de números de tamaño fijo” [6].

“Podemos utilizar una representación hash de palabras conocidas en nuestro vocabulario“ [6]. Tras clasificar las palabras “de forma determinista [...] en el espacio hash objetivo”, estas se pueden puntuar mediante “puntuación binaria o un recuento” [6].

Al elegir un espacio hash hay un compromiso entre las “colisiones" (según “se adapte al tamaño del vocabulario elegido”) y “la dispersión” [6].

TF-IDF

Para evitar una puntuación excesiva de aquellas palabras que aparecen frecuentemente en un documento, se puede emplear “la frecuencia con la que aparecen en todos los documentos” con el método “Término frecuencia - Frecuencia de documentos inversa, o TF-IDF”, basado en [6]:

- “Término frecuencia”: “frecuencia de la palabra en el documento actual”.
- “Frecuencia de documentos inversa”: puntúa la “rareza de la palabra” a lo largo de los documentos.

2.1.3.2 Técnicas de Aprendizaje en Minería de Texto

Hay dos grupos de algoritmos de minería de texto:

- Supervisados: si se dispone de datos etiquetados con la salida correspondiente que se quiere predecir. Algunos ejemplos de estos algoritmos son [1]:
  - "Naive Bayes"
  - "Redes neuronales"
  - "Naïve Bayes"
  - "Regresión Logística"
- No supervisados: en caso de poseer datos no etiquetados con su salida correspondiente. Que incluye algoritmos como:
  - K-medias (o "K-means" en inglés): divide el conjunto de datos de entrada en "k" grupos [8].
  - LDA [9]
  - Reglas de asociación [10]

Se han buscado artículos que tratan sobre clasificación de textos, al igual que este trabajo, como [11] que decide usar SVM y árboles de decisión para clasificar informes de textos "clínicos". Para clasificación de textos reducidos se ha encontrado otro artículo donde se emplean de nuevo las técnicas SVM y decision tree y alguna más para clasificar tweets [12].

Seguidamente se explican las técnicas de clasificación mediante árboles de decisión y máquinas de vectores de soporte.

Árbol de decisión

"Los árboles de decisión" (*decision tree* en inglés) se aplican en el ámbito del "machine learning" y destacan por ser un algoritmo sencillo de implementar y "fácil" de interpretar [13].

Esta técnica es supervisada y permite resolver problemas tanto de "regresión" (predecir el valor de un número) como de "clasificación" (dado un conjunto finito de clases, inferir a cuál de ellas pertenecen los datos).

"Un árbol de decisión es un modelo predictivo que divide el espacio de los predictores agrupando observaciones con valores similares para la variable respuesta". Las muestras se descomponen en grupos hasta que estos solamente contengan muestras de una de las clases.

La estructura de estos árboles es jerárquica, y se componen de nodos que se deben interpretar desde la raíz hasta las hojas. Hay tres tipos diferentes de nodos:

- "Primer nodo o nodo raíz:" subdivide las muestras según "la variable más importante".
- "Nodos internos o intermedios:" siguen subdividiendo las muestras según los valores de los predictores.
- "Nodos terminales u hojas: se ubican en la parte inferior del esquema" e indican la clase a la que pertenece cierto grupo de muestras.

En la práctica, para crear "un árbol de decisión" se emplea "el algoritmo de Hunt" que trata de separar las muestras de forma "óptima". Para ello subdivide las muestras "en función de una variable" hasta que el nodo solo contenga muestras de una clase (nodo hoja).

Hay varias métricas que pueden ayudar a elegir las variables por las que dividir las muestras, como por ejemplo: el índice Gini, la entropía y el RSS.

- Índice "de Gini" (Figura 1): "mide la probabilidad de no sacar dos registros de la misma clase del nodo". Es preferible un índice bajo que uno alto [13]. Se calcula mediante la siguiente fórmula:

$$GINI(t) = 1 - \sum_{i=1}^{n} (P_i)^2$$

Figura 1: Fórmula del índice de Gini [13].

- “Entropía” (Figura 2): indica el desorden de un sistema, valiendo 0 cuando solo hay muestras de una clase y 1 cuando el número de muestras está distribuido equitativamente entre las posibles clases [13].

$$H = -\sum_{i=1}^{n} P_i * log_2 P_i$$

Figura 2: Fórmula del índice de Entropía [13].

- “Suma residual de cuadrados“ o “RSS” en inglés (Figura 3): Se emplea para regresiones y cuantifica la diferencia “entre los datos reales y los predichos”. Cuanto más pequeño sea el RSS, mejor se ajusta el “modelo a los datos” [13].

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Figura 3: Fórmula del índice de RSS [13].

Máquinas de vectores de soporte (o “SVM”)

“Las Máquinas de Vectores de Soporte ([...] *“SVM”,* del inglés [...]) son un conjunto de algoritmos de aprendizaje supervisado” que permite “elementos” (datos) en categorías [14]. La técnica surge “en la década de 1990”, es eficiente, no requiere excesiva configuración y sigue estando bien considerada [15].

Al principio SVM recibe los datos relacionados con las categorías a clasificar y, después su funcionamiento se divide en dos fases: la de “entrenamiento” en la que “aprende” sobre un conjunto de datos ya etiquetados y, la de predicción en la que asocia categorías típicamente a datos distintos a los de la fase anterior [14]. A continuación, se explica más en detalle cómo funciona esta técnica en cada una de estas fases y posteriormente se hablará sobre las funciones de Kernel, no obstante para profundizar en la técnica se puede acudir a alguna de las fuentes bibliográficas citadas aquí.

Esta técnica resuelve ecuaciones lineales mediante “productos escalares entre vectores” lo cual lo cual es más eficiente computacionalmente frente a la resolución de ecuaciones no lineales.

En caso de tener elementos de dos posibles “clases” (o categorías) en el entrenamiento se calcula el hiperplano, cuya base está formada por “vectores de soporte”, que mejor las divida. Para ello debe ser “máxima” la “distancia entre el objeto más cercano al hiperplano en cada una de las clases”.

En la Figura 4, se muestran datos de dos posibles clases separados por los hiperplanos H1 y H2, ya que H3 no se considera al no separar las clases. No obstante, SVM se queda con H2 porque cumple con estar separado al máximo de los elementos más cercanos de ambas clases.

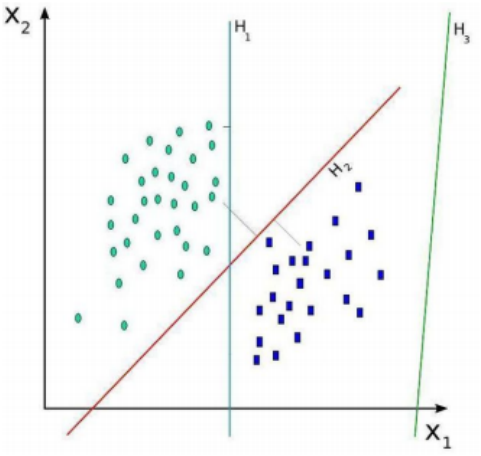

Figura 4: Datos divididos por hiperplanos [14].

Tras calcular “los vectores de soporte”, la fase “de predicción” consiste en determinar “a qué lado del hiperplano se encuentra el nuevo dato a clasificar”. Esto puede resolverse con “productos escalares, calculados por la función Kernel”. La ubicación del dato respecto al “hiperplano” revelará su categoría asociada.

Precisamente un aspecto fundamental de este tipo de técnica son “las funciones de Kernel”. Hay escenarios con datos que tienen “más de dos propiedades” o categorías. Para dividir entonces los datos en categorías ya no basta con una línea recta, ”se utiliza una función de mapeo Φ que asocia a cada elemento del conjunto original un vector de dimensión superior” (Figura 5). Con “el conjunto formado por los mapeos de los datos de entrada”, el problema ya es lineal y resoluble usando SVM.

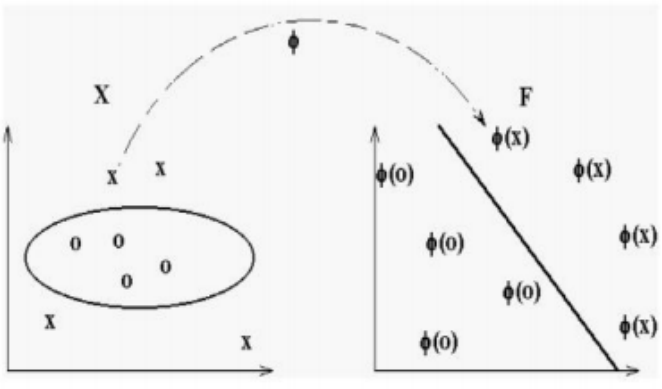

Figura 5: Transformación de un problema no lineal en lineal [14].

Las “funciones de Kernel” tienen “dos vectores” de entrada para los que se calcula su mapeo “Φ” y la salida es el “producto escalar” de estos “mapeos”..

El algoritmo puede diferenciar un par de vectores comparando el producto escalar “de los datos de entrada” y el “de sus mapeos Φ”. Los resultados de aplicar una función Kernel pueden “normalizarse” (rango 0 a 1)].

La mejor función de Kernel para minería de textos se denomina “SubString Kernel”, que valora la similitud de dos cadenas de caracteres. Conforme más largas sean las “subsecuencias” que tienen en común, más positivamente las puntúa pero puntúa negativamente aquellas cuyos caracteres tengan un orden distinto a los de la cadena con la que se compara.

2.1.3.3 Técnicas Avanzadas en Enriquecimiento de Texto

La idea de enriquecer el texto que se utiliza para el entrenamiento de los clasificadores en minería de texto responde a la posibilidad de incorporar metainformación sobre los datos que facilite la introducción de nuevas dimensiones que ayuden a los algoritmos de aprendizaje en la generación de mejores modelos de clasificación. Así, esta técnica no se refiere a aspectos cuantitativos de los datos de entrada, como es el caso del aumento de datos [16], sino de tipo cualitativo.

Dentro de las posibilidades que se encuentran en la literatura científica, en la minería de texto en lengua inglesa, es posible encontrar aproximaciones a esta idea basadas en Tesauros, técnicas de incrustación de palabras, traducción inversa o contextualización del texto [17]. En castellano, este enfoque de “texto aumentado” basado en la idea del diccionario ideológico Luis Casares ya se ha utilizado, de forma muy simplificada, con anterioridad en el análisis de blogs de internet [14].

2.2 Software y Prototipado Rápido

A día de hoy, existen numerosas herramientas software capaces de tratar datos y/o crear aplicaciones, las cuales pueden clasificarse en dos tipos según el tipo de usuario al que están orientadas: usuario común (como *Microsoft Excel*) u orientadas a desarrolladores (IDEs, etc.).

Dentro del marco de este trabajo se ha planteado como objetivo la exploración de herramientas software que permitan llevar a cabo el prototipado rápido para la posible realización de una experimentación sencilla.

Más adelante, se explican algunas herramientas software que pueden ser útiles para implementar funcionalidades más rápida o intuitivamente frente al recurso tradicional del desarrollo específico de programas usando por ejemplo librerías de soporte. Para cada una de ellas se comentan algunos aspectos basados principalmente en información encontrada relacionados con la accesibilidad y usabilidad de las mismas. Sin embargo, no se llega a realizar un test de accesibilidad ni de usabilidad o un estudio profundo de las herramientas con lo que se podría concluir con más precisión qué aspectos de éstas cumplen con ciertas especificaciones de accesibilidad y usabilidad y cuáles no.

A continuación, se va a introducir la idea de prototipo de software.

2.2.1 Prototipo software

En general un prototipo en el ámbito del software se refiere a “documentos, diseños o sistemas que simulan o tienen implementadas partes del sistema final [...]”. “Los prototipos” son “una herramienta muy útil para hacer participar al usuario en el desarrollo y poder evaluar el producto desde las primeras fases del desarrollo” [18].

El uso de prototipos software presenta diversas ventajas como:

- Permite “obtener información sobre un diseño propuesto en la fase inicial de un proyecto”, lo cual puede ayudar tanto a desarrolladores como a usuarios finales o futuros clientes potenciales [18].
- En general, especialmente si dispone de interfaz gráfica de usuario (GUI en inglés), lo hace más intuitivo y fomenta que sea usable y accesible para un abanico más amplio de usuarios que un producto final, especialmente si éste no permite configurarse con flexibilidad según las necesidades del usuario y más todavía en caso de carecer de GUI.
- Puede suponer un ahorro económico y de tiempo frente a cambios continuos en la implementación de un proyecto software tradicional en el que los requisitos cambien con frecuencia [18].
- Actualmente existen numerosos programas para la realización de prototipos (prototipado) versátiles y de licencia gratuita [18].

Sin embargo, el prototipado también presenta desventajas como:

- Coste económico adicional y/o en tiempo si hay que desarrollar posteriormente también un producto software final.
- “Crea falsas” expectativas en ocasiones al no poderse implementar por completo en el producto final lo que se ha prototipado [18].
- En general, no permite llegar al mismo nivel de detalle para desarrollar ciertas funcionalidades que una implementación sobre un lenguaje de programación.
- “Limitado para la corrección de errores” o de difícil solución, al menos para algunos que son dependientes del programa software empleado sobre el que no siempre se tiene acceso total [18].

A continuación, se presentan algunas herramientas software empleadas habitualmente en minería de textos.

2.2.2 Entornos software

En este apartado y en consonancia con el segundo objetivo específico de este trabajo se van a estudiar cinco entornos software que permiten realizar clasificación de texto.

2.2.2.1 KNIME Analytics Platform (KNIME)

Introducción

KNIME es una plataforma basada en Eclipse para ciencias de datos y machine learning [19].

Licencia

Es “software libre” bajo licencia “GPL” [20]. Se puede descargar y usar gratuitamente sobre distintos sistemas operativos: Windows, Linux y Mac OS X.

Accesibilidad

Se puede considerar una herramienta accesible para muchos usuarios ya que uno de sus puntos fuertes es que dispone de una GUI intuitiva basada en arrastrar y soltar (“drag-drop” en inglés) nodos sobre un lienzo (flujo de trabajo). Cada uno de estos nodos desempeña cierta funcionalidad software y puede obtenerse su documentación de forma sencilla en un panel de la GUI al hacer click sobre el nodo. Además, muchos de estos nodos permiten configurar su funcionamiento mediante otra interfaz gráfica de usuario propia, normalmente basada en un formulario consistente en botones, campos de texto y otros elementos. También dispone de un foro para plantear dudas por ejemplo relacionadas con la configuración de cierto nodo.

Usabilidad

En cuanto a sus funcionalidades, KNIME permite realizar multitud de tareas de “minería de datos”. Por ejemplo soporta: conexión y operabilidad con diferentes BBDD (SQL y noSQL), ejecución de algoritmos de machine learning (por ejemplo, decision tree o perceptrón multicapa, entre otros), ejecución de scripts en R, Python o código Java limitado, entre otras funcionalidades. Cabe destacar que permite instalar extensiones software, como por ejemplo el paquete “Textprocessing” [9] que incluye nodos útiles para el tratamiento de textos lo cual puede simplificar tareas en el contexto del proyecto.

2.2.2.2 Orange Data Mining

Introducción

Se trata de una herramienta gráfica fruto de la “Universidad de Ljubljana” que integra funcionalidades para minería de datos [21] y machine learning [22].

Licencia

Tiene una licencia GPL [21] y se puede descargar gratuitamente para diferentes sistemas operativos como Windows y Mac OS X [22].

Accesibilidad

Dispone de interfaz gráfica y de flujos de trabajo (“workflows”, en inglés) [22] prediseñados lo que ayuda positivamente a la accesibilidad de la herramienta al ponerla al alcance de más usuarios que no sean expertos en minería de datos ni programación en general.

Usabilidad

De nuevo la propia interfaz gráfica de usuario, los workflow preconstruidos y, adicionalmente, la documentación que tiene disponible incluso en YouTube contribuyen positivamente a que la herramienta sea usable [22] para un grupo considerable de usuarios.

2.2.2.3 RapidMiner

Introducción

Este programa se ha desarrollado “en Java” [23] y permite realizar análisis y minería de datos.

Licencia

Tiene una licencia AGPL y se puede probar gratis temporalmente o usar mediante una licencia educativa.

Accesibilidad

Dispone de varias distribuciones con GUI incluso una para web, permite crear flujos de trabajo de forma gráfica y cuenta con soporte técnico, lo cual contribuye a favorecer la accesibilidad al permitir que usuarios no expertos en programación puedan usar la herramienta o resolver sus dudas [24]. Según un estudio publicado en [25] en el que se preguntaba sobre “¿Qué software de análisis, macrodatos, ciencia de datos y aprendizaje automático utilizó en los últimos 12 meses para un proyecto real?”, para las respuestas en 2019, *RapidMiner* quedó como el segundo software más empleado tan sólo superado por *Python*. Estos datos también parecen indicar que es una herramienta accesible ya que muchos usuarios la utilizan.

Usabilidad

Aspectos ya comentados para la accesibilidad, como los relacionados con las GUI y el soporte técnico que ofrece contribuyen a que *RapidMiner* sea una herramienta usable en el ámbito de la minería de datos, “la ciencia de datos y el aprendizaje automático" [25].

2.2.2.4 Ludwig

Introducción
Ludwig se trata de un conjunto de herramientas de “aprendizaje profundo” desarrolladas a partir de “TensorFlow”. Con un comando permite entrenar un modelo a partir de su definición en un fichero de configuración de extensión .yaml y los datos de entrada especificados como parámetros [26].

Licencia
Tiene “Licencia Apache 2.0” y es de “código abierto” [27].

Accesibilidad
La característica de que no requiere programación hace que la herramienta sea más accesible para usuarios no programadores.

Usabilidad
Permite entrenar modelos de aprendizaje automático de manera muy rápida y sencilla, con un comando, la configuración del modelo a entrenar y los datos de entrada. Esto fomenta que la herramienta sea usable para este tipo de tareas de aprendizaje automático.

2.2.2.5 Deep Learning Studio

Introducción
Se trata de una herramienta para el aprendizaje profundo ampliamente utilizada que permite entrenar modelos de forma sencilla y mediante interfaz gráfica. Se ha escrito en Python [28].

Licencia
Es una herramienta de “Software propietario” y que puede utilizarse en los sistemas operativos “Windows” y “Ubuntu Linux” y en cloud sobre “Amazon AWS” y “MS Azure” [28]. No obstante, el producto se puede descargar y usar de forma gratuita según se especifica en [29].

Accesibilidad
Su interfaz gráfica de usuario favorece que sea una herramienta accesible para usuarios no expertos en aprendizaje automático.

Usabilidad
De nuevo, la posibilidad de componer un modelo de aprendizaje profundo incluyendo todos estos entornos como se especifica en [29]: “Chainer, H20, Keras, Mxnet, Tensorflow, Caffe, Chainer, Pytorch” mediante una GUI, entrenarlo y obtener resultados, provocan que la herramienta sea más usable comparado con si requiriese programación. De esta manera se simplifica y se requiere menos tiempo para desarrollar un entrenamiento [29].

2.3 Conclusiones

Tras revisar el estado del arte se concluye que las técnicas de minería de texto SVM y árbol de decisión pueden ser adecuadas para clasificar textos de tamaño reducido ya que ambas se emplean se pueden emplear en el ámbito de la clasificación de textos y son algoritmos supervisados como los textos que se quieren clasificar en el proyecto. Además, la técnica SVM arroja unos resultados precisos y su coste computacional no es excesivo y, el árbol de decisión puede ser especialmente útil para entender qué variables (palabras) son más adecuadas para predecir la categoría a la que pertenece cierto texto.

Por su lado, en cuanto a las herramientas software de prototipado rápido estudiadas se puede observar que muchas de ellas comparten características en común (disponen de GUI salvo Ludwig, tienen versión gratuita, etc.) y a priori todas ellas son buenas candidatas para el prototipado rápido de modelos de clasificación de textos.

Capítulo 3 - Propuesta de solución, diseño e implementación

En este apartado se abordan algunas decisiones de diseño tomadas y se explica la parte de la implementación desarrollada que da soporte a los posteriores experimentos en los que se clasifican textos en distintas posibles categorías.

3.1 Requisitos

Los requisitos que debe cumplir un sistema informático se pueden clasificar en dos clases según sean: funcionales o no funcionales. Los requisitos funcionales hacen referencia sobre todo a criterios objetivos que debe cumplir el sistema, como las funcionalidades que este ofrece a un usuario, las entradas y las salidas del sistema. En contraste con los anteriores, los requisitos no funcionales abarcan aspectos más subjetivos y también objetivos pero relacionados con la forma de implementación (lenguaje de programación empleado, tiempo de respuesta para realizar determinada operación, etc.) del sistema.

Los requisitos funcionales que el sistema objetivo de este trabajo debe cumplir son:

1. Recopilar una colección de textos clasificados en categorías.
2. Almacenar los textos y su categoría asociada en un soporte de almacenamiento.
3. Recuperar los textos y su categoría asociada del soporte de almacenamiento donde residan.
4. Obtener datos de texto enriquecido. La idea es usar información de un diccionario conceptual.
5. Clasificar textos según 12 categorías con 2 clasificadores.

3.2 Diseño del Sistema

En este apartado se toma una de las decisiones fundamentales para el trabajo, se decide qué software de prototipado rápido se va a emplear.

Tras el estudio en el apartado 2.2 de varias alternativas software que pueden servir para realizar minería de textos, se decide emplear KNIME como entorno de prototipado rápido. Se escoge este software principalmente porque ya se conoce y se ha adquirido experiencia con él principalmente en la asignatura del grado denominada “Sistemas de ayuda a la toma de decisiones”.

A continuación, se describen algunas decisiones de diseño que afectan al sistema clasificador de textos objeto del proyecto.

3.2.1 Categorías de textos

Se ha elegido un conjunto de 12 categorías de textos. Por un lado, estas categorías se emplearán para extraer un conjunto de textos en el que cada texto tenga una categoría asociada (se presupone que el contenido del texto tratará sobre dicho tema o categoría); estos textos etiquetados en categorías se usarán para entrenar el clasificador de textos. Por otro lado, las categorías se emplearán como las posibles respuestas de la clasificación de los textos de prueba en los experimentos, es decir, dado un texto este se clasificará, empleando minería de texto, de forma que a la salida se obtendrá la categoría o clase a la que corresponda el texto procesado.

La selección de las categorías se ha llevado a cabo tras inspeccionar diferentes palabras o ideas en el diccionario conceptual Zirano del que se hablará en el apartado 3.2.3 y, se han escogido de tres en tres de forma que cada trío se engloba dentro de una categoría más general. Además, dentro de las limitaciones del diccionario Zirano (en algunos casos para encontrar cierta idea que por ejemplo puede tener varias acepciones se requiere buscarla con signos de puntuación o junto a otras palabras) y del tiempo disponible de trabajo, se ha intentado tener en cuenta la métrica del número de ramas de cada árbol de Zirano asociado a cada palabra proporcionada, de forma que las categorías de cada trío no difieran demasiado en número de ramas entre sí.

A continuación, en la Tabla 1, se muestran las 12 categorías de textos elegidas:

Fútbol	Dinero	Ordenador	Viaje
Natación	Tienda	Internet	Automóvil
Atletismo	Préstamo	Televisión	Barco

Cuadro 1: Tabla con las categorías de textos seleccionadas.

Por columnas (en tríos), las categorías presentadas en la Tabla 1 podrían agruparse en categorías más generales: ocio (o deporte), posesión o comercio, información (o tecnología) y desplazamiento, respectivamente.

3.2.2 Corpus de texto

Es necesario obtener datos de texto y su categoría asociada, así como poder almacenarlos y recuperarlos de un soporte de almacenamiento ya que parte del objetivo del trabajo es desarrollar un sistema de minería de texto.

Existen textos que pueden versar sobre distintas categorías y tener distinta extensión. En concreto, se ha decidido desarrollar un sistema de clasificación de textos de longitud relativamente corta.

Para ello se ha optado por obtener los datos de la red social Twitter a través de su API que puede usarse gratuitamente con registro y aprobación previa. Los textos o tweets que proporciona tienen prefijado un límite de longitud en número de caracteres que, en abril de 2021, es de 280 aunque en otro tiempo fue de 140 [30], por lo cual su tamaño es adecuado.

3.2.3 Minería de Texto con Campos Conceptuales

Uno de los objetivos del trabajo es realizar un sistema de minería de texto utilizando metainformación del lenguaje. En correspondencia con la idea de “Texto Aumentado” (Augmented Text) introducida en el apartado 2.1.3.3, se ha decidido explorar la inclusión de palabras clave en el modelo de predicción utilizando la idea de campo conceptual. A continuación, se proporcionan definiciones de interés en este contexto:

- Campo conceptual: en [14] se define como “Conjunto de palabras asociadas a una misma idea o concepto. No es necesario que tengan un origen común, puesto que en lo que se centra un campo conceptual es en el significado de las palabras y no en su forma”.
- Diccionario analógico conceptual: es un conjunto de palabras que permite recuperar palabras más concretas relacionadas con una palabra de entrada o “concepto” [31].

Para el trabajo se ha decido emplear un diccionario analógico conceptual llamado Zirano [32] del que se extraen los campos conceptuales necesarios. El proceso para obtener el campo conceptual de una palabra es el siguiente:

- La palabra se introduce en el buscador que hay presente en [32].
- Se hace click en el botón de “Enviar la consulta”.
- Después se hace click en el botón “Árbol” situado en la parte superior de la pantalla.
- Aparece un árbol conceptual (árbol de Zirano) con la palabra inicial resaltada. Si no es una hoja, habrá un conjunto de palabras hijas relacionadas con esta palabra inicial que conforman el campo conceptual.

3.2.4 Sistema clasificador de textos

El sistema de minería de texto basado en campos conceptuales que se ha planteado como objetivo se decide concretar en un clasificador de textos basado en aprendizaje supervisado, para lo cual es necesario disponer de un conjunto de textos cada uno asociado con su categoría correspondiente.

Antes de realizar la clasificación, tras extraer los textos, puede ser necesario procesarlos para usarlos con los algoritmos de minería de textos a emplear entre otras razones. A continuación, se describen brevemente cuatro posibles fases de preprocesamiento de los datos en KNIME. Según [33], las fases son las siguientes:

1. Lectura de los datos usando tokenización (o Document en KNIME).

   En KNIME un dato *String* que representa un texto puede convertirse en un dato especial llamado *Document* que puede facilitar posteriores procesamientos del mismo. Los documentos se componen de tokens que normalmente equivalen a una palabra del documento. Además, un texto puede tener metainformación como por ejemplo: título, autor, fecha, entre otros.

2. Enriquecimiento de los datos con tags.

   En esta fase se trata de añadir información a los tokens del texto. Esta información añadida se denomina ”tag” y al conjunto de un token con un tag se le llama término (o “term” en inglés) .

   Se pueden añadir distintos tipos de tags como por ejemplo los relacionados con la función gramatical de la token (o POS en inglés; que pueden indicar si una palabra o token es un “verbo”, un “sustantivo”, un adjetivo, etc.), los relacionados con análisis del sentimiento (que por ejemplo mostrarían si el token tiene un sentido positivo o negativo) u otros como los relacionados con las “personas” o la “localización”, etc [34].

3. Limpieza de datos y normalización.

   El objetivo de esta fase es conseguir que sea más pequeño el tamaño del texto o corpus (colección de textos) que se esté tratando y, además, “normalizar los tokens” . Así, se disminuye el tiempo de procesamiento (al emplear un algoritmo de clasificación por ejemplo).

   Algunos métodos que se emplean en la subfase de limpieza son:

   - Stopwords: para eliminar palabras presentes en el texto y que “aportan poco significado” como “a” y “de”, etc. Reduce el número de palabras en el texto y aumenta la frecuencia relativa de las demás palabras que probablemente son más relevantes.
   - Filtrar por “tags”: Consiste en quedarse solo con palabras con ciertos tipos de tags como los de tipo gramatical que sean adjetivos .

   Mientras que en la fase de normalización se emplean las siguientes técnicas como:

   - Derivación (o “stemming” en inglés) : “Proceso en el que, a partir” de “un lexema y un conjunto de morfemas, se obtienen todas las palabras que se pueden formar combinándolos. [...]” [14].
   - Lematización (o “lemmatization” en inglés): “Proceso inverso a la derivación. Dada una palabra, se obtiene su lexema” [14]. De otra manera, consiste en “la retención de palabras legibles por humanos en lugar de términos derivados” .
   - Conversión de letras (“Case conversion” en inglés): Transformar las letras a minúsculas o mayúsculas .

4. Transformación.

   La última fase trata de conseguir una “representación numérica” o de otro tipo a partir de los datos de la fase anterior. Con ello se pueden usar algoritmos para clasificar documentos en categorías, crear clústeres, etc. Hay distintas técnicas para conseguir una representación numérica de un texto [33], como por ejemplo:

   - Bolsa de Palabras (explicada en el apartado 2.1.3.1).
   - Vector de documento, que es una representación numérica y vectorial. Cada texto o documento del corpus tiene su propio vector. Cada vector tiene una columna por término del corpus. El valor de cada una de las componentes representa la presencia o no (1 o 0, respectivamente) del término en el texto asociado al vector.

3.3 Implementación

Principalmente en este apartado se presentan los principales flujos de trabajo realizados para obtener y filtrar los datos necesarios del diccionario analógico conceptual Zirano y, también para obtener los de Twitter de forma que posteriormente se puedan usar en los experimentos de clasificación de textos presentados en el Capítulo 4.

En primer lugar, referente al entorno KNIME empleado para el desarrollo se explican brevemente 2 conceptos fundamentales: los nodos y los flujos de trabajo (o *workflows,* en inglés). Por un lado, los nodos se representan como cajas que pueden tener puertos de entrada de datos, de salida o de ambos tipos. Son bloques funcionales que realizan ciertas operaciones predefinidas con datos aunque normalmente son parcialmente configurables mediante parámetros. Por otro lado, los workflows pueden verse como contenedores donde se pueden depositar nodos, interconectarlos entre sí y donde se pueden realizar anotaciones, etc. Además, son ejecutables y normalmente presentan un conjunto de nodos que interactúan entre sí, realizando cierta tarea con datos.

Cabe mencionar 2 tipos de nodos especiales: los metanodos y los componentes. Estos pueden agrupar dentro de sí varios nodos y permiten configurar por ejemplo su número de puertos de entrada y salida. Precisamente en algunos workflows desarrollados se han empleado metanodos que encapsulan ciertos nodos con lo que se evita tener un workflow con demasiados nodos lo que podría dificultar la legibilidad y además, permiten ser reutilizados lo que puede evitar trabajo innecesario de replicar cada nodo o hacer un copiar y pegar de múltiples nodos. Para más información de los metanodos de este apartado puede consultarse el Anexo 4. Si no se indica lo contrario, cuando se explique el funcionamiento de los nodos de un workflow se supondrá que un nodo recibe en su entrada los datos de la salida del nodo anterior, salvo que se considere un nodo solo con puerto de salida.

Como ya se había comentado anteriormente, el entorno de prototipado rápido empleado para implementar el trabajo ha sido KNIME. En el momento de escribir esta memoria se emplea la versión “KNIME 4.3.3”. A continuación, se destacan algunas de las extensiones de software añadidas y otros recursos externos empleados sobre KNIME en la implementación:

- Nodos Twitter [35].
- Nodo “String Emoji Filter” para filtrar emojis [36].
- Python en KNIME [37].
- R en KNIME [38].
- Nodos Selenium para Web scraping [39].
- Nodo Java Snippet (con descripción: “Switch to popup”) para gestionar una ventana pop up (para poder hacer Web scraping en la ventana del árbol de Zirano con Selenium). Proviene de un workflow [40].
- Extensión “Text Processing” comentada en el apartado 2.2.2.1 [9].
- Treelib, biblioteca para Python para crear y gestionar datos arborescentes [41], [42].

En el Anexo 2 se proporciona más detalle sobre algunas de las extensiones de software instaladas en KNIME y se informa sobre dos de las bibliotecas de Python empleadas.

Adicionalmente, cabe destacar que se ha decidido emplear ficheros CSV como soporte de almacenamiento para la lectura y escritura de datos (por ejemplo, de Twitter y de Zirano) del proyecto frente a la BBDD SQLite construida (para obtener más información relacionada puede consultarse el Anexo 1) debido a que son más sencillos de usar en KNIME y se pueden almacenar varias versiones de los datos en distintos ficheros. No obstante, puede consultarse un workflow implementado para escribir datos en la BBDD en el Anexo 4 (véase A.4.4).

3.3.1 Extracción de textos

En el workflow de la Figura 6 se extraen 1000 tweets de Twitter por categoría, se preprocesan para seleccionar solo aquellos con longitud entre 270 y 280 caracteres y, además, durante la búsqueda de los tweets se filtran mediante operadores de Twitter aquellos con imágenes y links, entre otros elementos. Finalmente de los tweets extraídos y preprocesados, se intentan escribir 120 (10 por categoría, cada texto asociado con una de ellas) en un fichero CSV que se emplea como conjunto de textos en los experimentos.

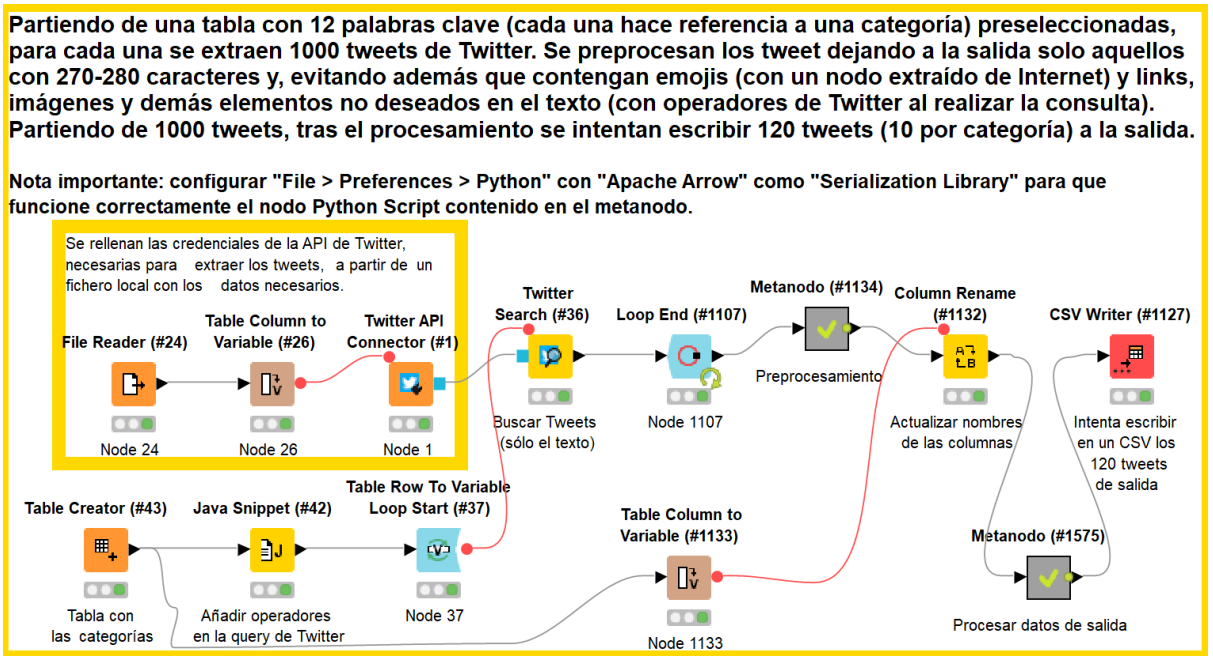

Figura 6: Flujo de trabajo para extraer los datos de Twitter.

3.3.2 Extracción de campos conceptuales

El workflow de la Figura 7 obtiene los campos conceptuales de los árboles de Zirano asociados a palabras clave (categorías en este caso) especificadas en el nodo *Table Creator (#116)* empleando la técnica de *Web scraping* con nodos Selenium.

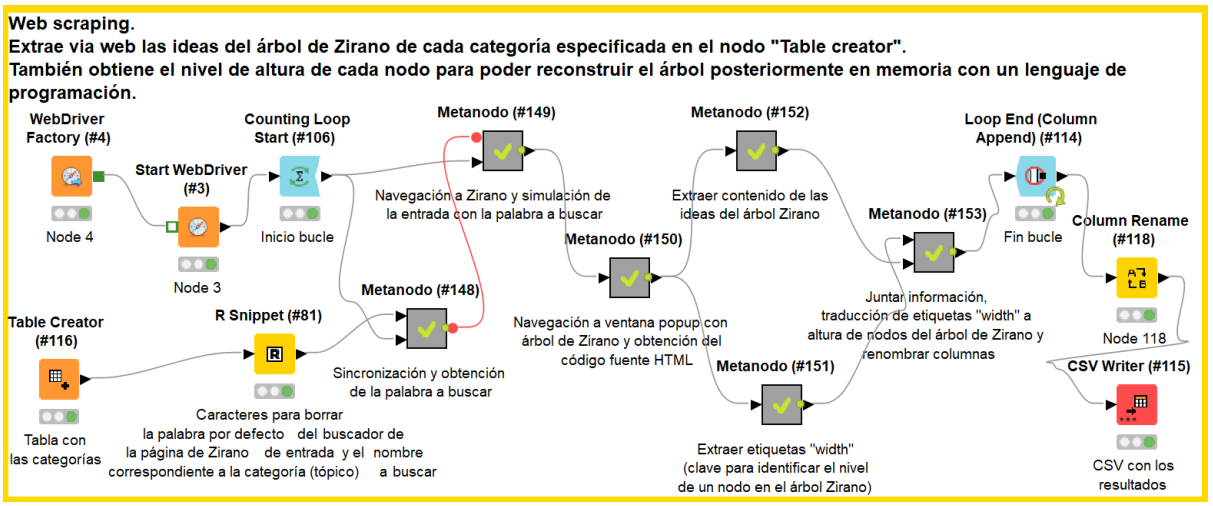

Figura 7: Flujo de trabajo que extrae campos conceptuales de Zirano a partir de nombres de categorías.

3.3.3 Creación y filtrado de árboles

El workflow de la Figura 8 recibe como entrada una colección de ideas de Zirano y una tabla con los nombres de las columnas y a la salida devuelve cada colección de ideas de Zirano preprocesada eliminando ítems con caracteres como '>' al principio, el contenido entre paréntesis de cada ítem, si lo hubiera y, filtra los nodos del árbol de Zirano de manera que se eliminan aquellos (salvo la raíz) cuyo padre no cumple alguna de estas condiciones:

- Tengan el símbolo "mayor que" ('>') al final del ítem.
- O tengan un carácter guión ('-') al final del ítem.
- O bien tengan una aclaración entre paréntesis.

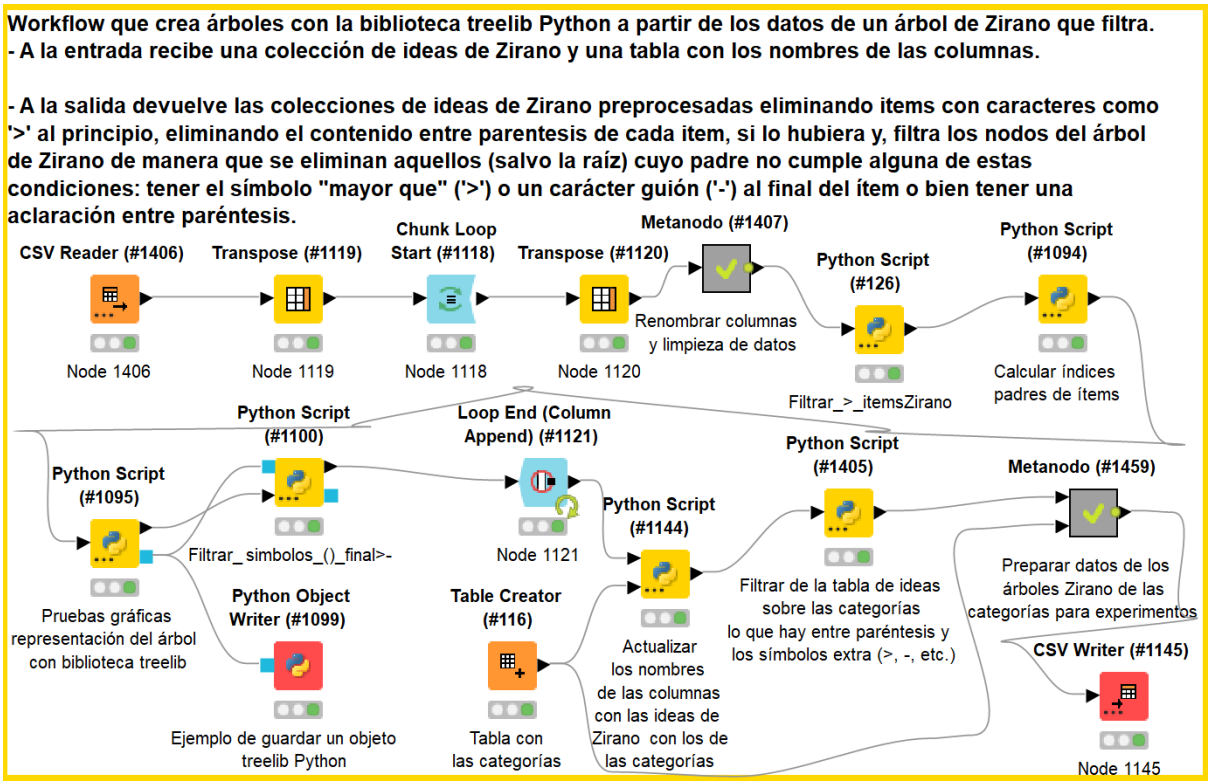

Figura 8: Flujo de trabajo para filtrar información de árboles de Zirano.

Capítulo 4 - Experimentos

En este capítulo se presentan los experimentos que se han implementado para clasificar textos en las diferentes categorías seleccionadas. En concreto se han implementado 6 experimentos distintos. Para cada uno de ellos se presenta el flujo de trabajo asociado (o *workflow* en inglés) y se realiza una descripción. En el Anexo 5, se incluye información más detallada sobre los flujos de trabajo objetos de este apartado. Además, se presentan resultados sobre cada experimento, basados principalmente en métricas como la exactitud (*accuracy),* la precisión (*precision*) y la recuperación (*recall)* y en la matriz de confusión.

Cabe destacar que para todos los experimentos se ha empleado el mismo conjunto de 120 tweets (almacenado en un fichero CSV) extraído de Twitter (empleando un workflow similar al del apartado 3.3.1 de esta memoria) lo que aporta consistencia a los resultados. Así mismo, se han empleado las mismas secuencias de palabras del árbol de Zirano de las categorías (obtenidas con un workflow similar al del apartado 3.3.3 de esta memoria) en los experimentos que las usan.

Además, se destacan algunos workflows externos o partes de ellos que se han empleado en el trabajo (a veces se han adaptado, realizando modificaciones respecto del original):

- El workflow de ejemplo de Knime llamado 02_Document_Classification [44] se ha empleado de alguna manera en todos los experimentos (en algunos se han copiado algunas de sus partes e incluso se han adaptado en algunos casos y en otros sin copiar partes directamente pero se ha tomado como base de inspiración para replicar alguna parte).
- El workflow del experimento 4 está inspirado en uno externo [45].

El entorno en el que se han realizado todos los experimentos presentados es un ordenador portátil cuyas características principales se destacan a continuación:

- Sistema operativo: Windows 10 (64 bits).
- Procesador: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
- Memoria: 16.0 GB (15.8 GB usable)

4.1 Experimento 1

Su objetivo es clasificar tweets en categorías empleando 2 técnicas de minería de textos: SVM y árboles de decisión. Se emplean 120 tweets para la clasificación de textos divididos en dos conjuntos: 70% para entrenar y 30% para test. En la Figura 9 se puede observar el flujo de trabajo asociado a este experimento.

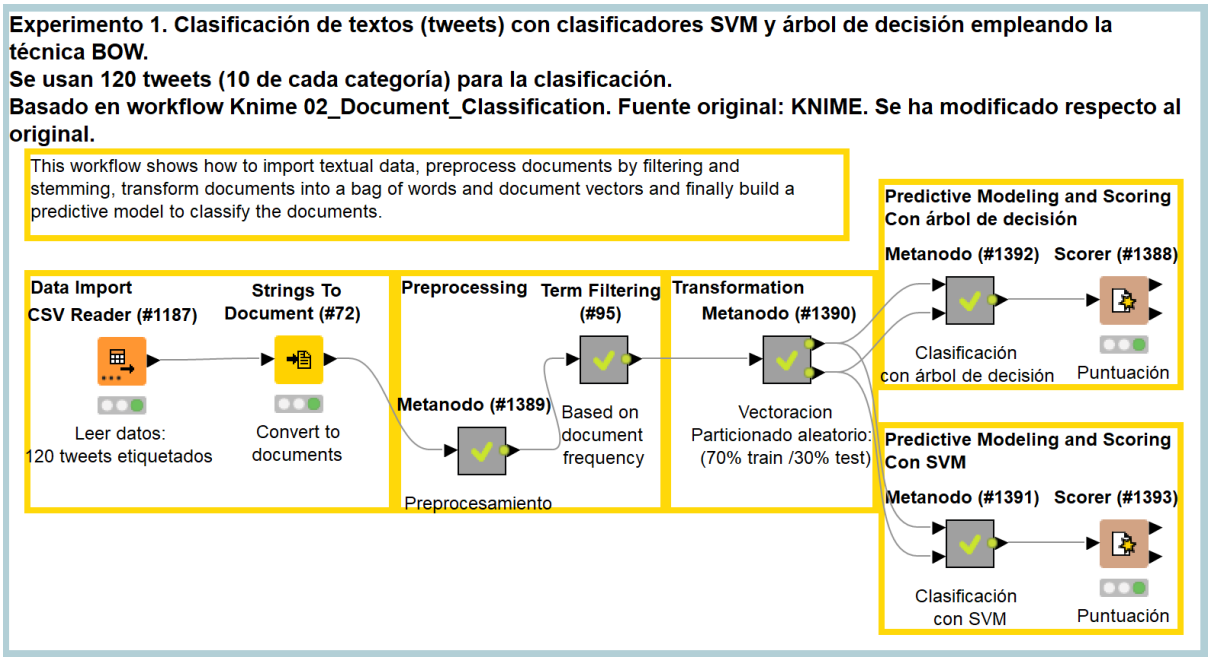

Figura 9: Flujo de trabajo del Experimento 1.

Resultados del experimento con árbol de decisión como clasificador

En la Tabla 3 se muestran los resultados del Experimento 1 empleando árbol de decisión como clasificador. Este clasificador predice todo perfectamente para el conjunto de datos de entrada, es decir, nunca se equivoca en la predicción ya que tanto el *recall* como la *precision* tienen valor 1 (y no hay falsos positivos ni falsos negativos por tanto) para todas las categorías.

Resultados del experimento con SVM como clasificador

En la Tabla 4 se muestran los resultados del Experimento 1 empleando SVM como clasificador. Como se puede observar, los resultados de la clasificación son muy buenos llegando a obtener un 94,4% de *accuracy*, lo que indica que se ha acertado en casi todas las predicciones.

Para la mayoría de las categorías la *precision* y el *recall* tienen valor 1, lo que indica que el clasificador ha funcionado perfectamente en estos casos (ha clasificado todas las muestras bien). El peor caso con el *recall* en el modelo se da en las categorías de Atletismo y Fútbol, en las que sale un valor de 0.8 y precision de 1 lo que indica que a veces identifica incorrectamente lo de su categoría (falso negativo) y, el peor caso de la *precision* se da con las categorías de Barco y Natación porque a veces identifica otras categorías como de la suya propia (falsos positivos).

4.2 Experimento 2

La descripción de este flujo de trabajo es equivalente a la del Experimento 1 excepto que el conjunto de datos con el que se trabaja cambia. Así, solamente difiere en el primer recuadro “Data Import” respecto al workflow del experimento 1. En la Figura 10 se puede observar el flujo de trabajo asociado a este experimento.

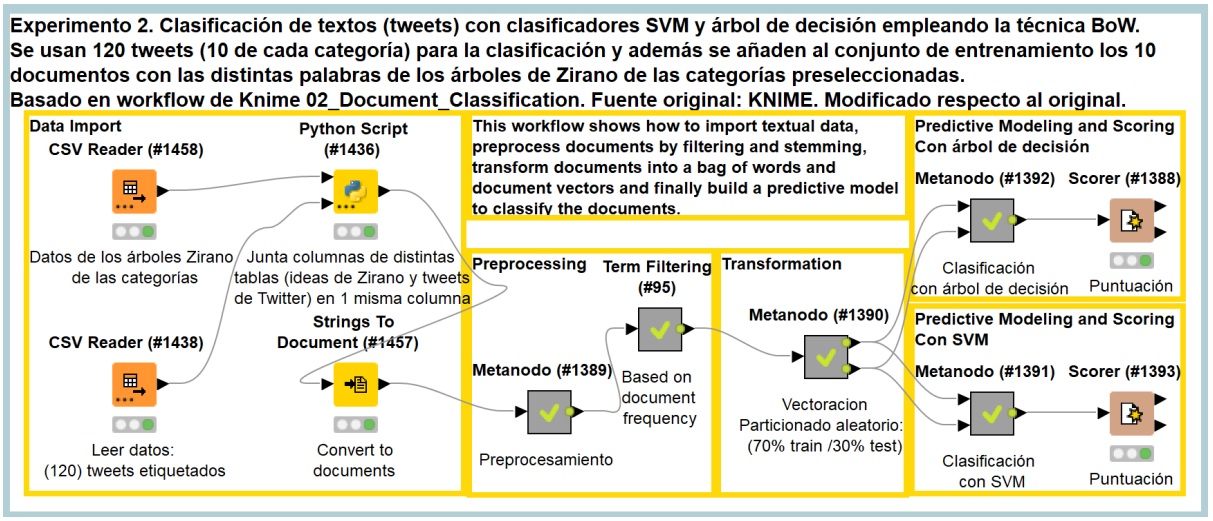

Figura 10: Flujo de trabajo del Experimento 2.

Resultados del experimento con árbol de decisión como clasificador

En la Tabla 5 se muestran los resultados del Experimento 2 empleando árbol de decisión como clasificador. La accuracy es razonablemente elevada, de un 97,5%, lo que indica que el clasificador acierta casi siempre en sus predicciones con los conjuntos de datos de entrada.

El peor caso para el valor del *recall* lo encontramos en la categoría de Televisión, lo que indica que en ocasiones identifica mal las muestras de esa categoría (falsos negativos). Y, en cuanto a la *precision*, el peor valor se obtiene para la categoría de Dinero, con un valor del 80% que aunque es razonablemente alto parece indicar que a veces predice muestras de otras categorías como si fueran de la suya propia (falsos positivos).

Resultados del experimento con SVM como clasificador

En la Tabla 6 se muestran los resultados del Experimento 2 empleando SVM como clasificador. Como se puede observar, los resultados de la clasificación son razonablemente buenos, obteniendo un *accuracy* del 95%, lo que indica que el clasificador ha acertado en la mayoría de las predicciones. De hecho casi todas las categoría las predice perfectamente ya que la *precision* y el *recall* tienen valor 1.

El peor caso con el *recall* en el modelo se da en la categoría de Viaje en la que se obtiene un valor relativamente bajo (66,7%) y *precision* alta (de 1), lo que indica que a veces identifica mal las muestras de esa categoría (falsos negativos) y, el peor en el caso de la *precision* (valor 66,7%) se da con la categoría de Internet, lo que parece indicar que a veces identifica los tweets de otras categorías como si fueran de la suya propia (falsos positivos).

4.3 Experimento 3

El objetivo del tercer experimento también es clasificar textos en categorías empleando dos técnicas de minería de textos: SVM y árboles de decisión. Su principal característica diferenciadora respecto a los experimentos previos reside en sus conjuntos de datos:

- Conjunto de entrenamiento: 12 secuencias de palabras de los árboles de Zirano de las categorías.
- Conjunto de test: formado tanto por 120 tweets como por las mismas 12 secuencias de palabras de los árboles de Zirano de las categorías del conjunto de entrenamiento.

En la Figura 11 se puede observar el flujo de trabajo asociado a este experimento.

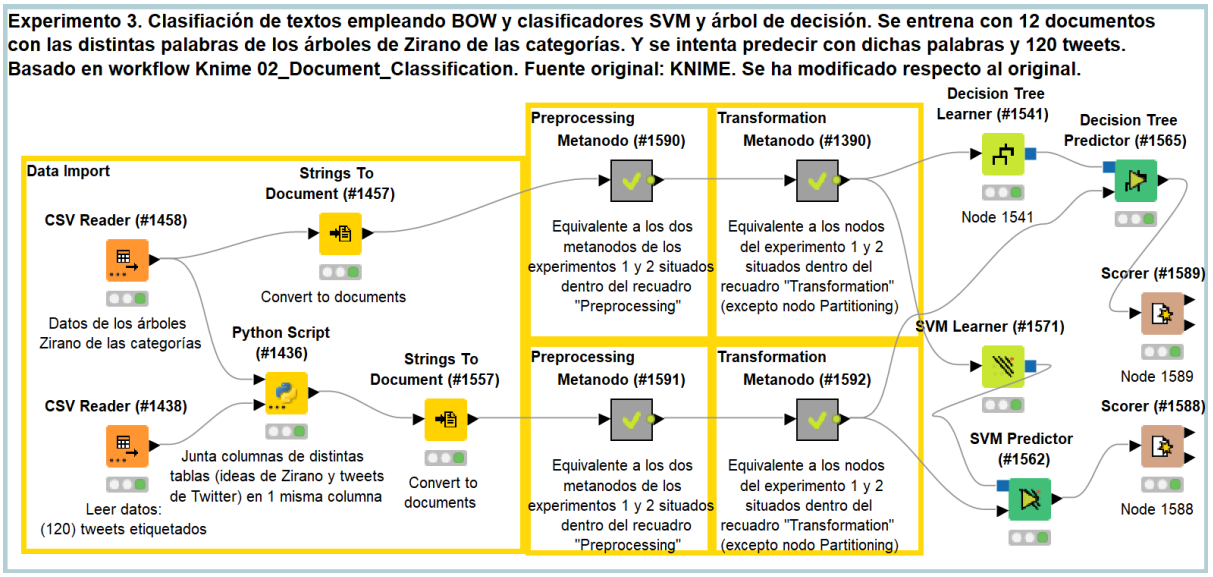

Figura 11: Flujo de trabajo del Experimento 3.

Resultados del experimento con árbol de decisión como clasificador

En la Tabla 7 se muestran los resultados del Experimento 3 empleando árbol de decisión como clasificador. Las predicciones de este clasificador basado en árbol de decisión son claramente negativas dado que su *accuracy* es del 8,30%, lo que indica que casi siempre se equivoca en sus predicciones.

Salvo para la categoría de Atletismo que la predice correctamente (verdadero positivo), en el resto de categorías, el *recall* es de 0 por lo que detecta en muchas ocasiones falsos negativos.

Resultados del experimento con SVM como clasificador

En la Tabla 8 se muestran los resultados del Experimento 3 empleando SVM como clasificador. Los resultados de este clasificador son razonablemente buenos ya que la *accuracy* es de un 93,90%, lo que indica que acierta en casi todas sus predicciones.

Sin embargo, no todas las categorías las predice perfectamente; el peor valor para el *recall* se obtiene para la categoría de Automóvil, con un valor del 54,50%, lo que apunta a que casi en la mitad de las ocasiones identifica mal las muestras de esa categoría (falsos negativos). Para la *precision* el peor valor se obtiene con la categoría de Viaje, valor de 68,80% lo que se ve reflejado en la gran cantidad de falsos positivos (muestras de otras categorías predecidas como de la suya) respecto de verdaderos positivos que se obtiene (5 frente a 11, casi la mitad).

4.4 Experimento 4

En este experimento se implementa (Figura 12) un filtro de diccionario basado en el nodo Dictionary Filter. Se parte de un conjunto de 120 tweets y de 12 secuencias de caracteres formadas por palabras extraídas del árbol de Zirano de cada categoría.

Es un clasificador que predice a qué categoría pertenece cada tweet. Se predice que la categoría de un determinado tweet es aquella con la que más coincidencias tenga (en nº de coincidencias de las palabras del tweet con las que forman la secuencia de palabras del árbol de Zirano asociadas a la categoría). En caso de empate, se escoge la primera de las posibles categorías con las que más coincidencias tenga el tweet.

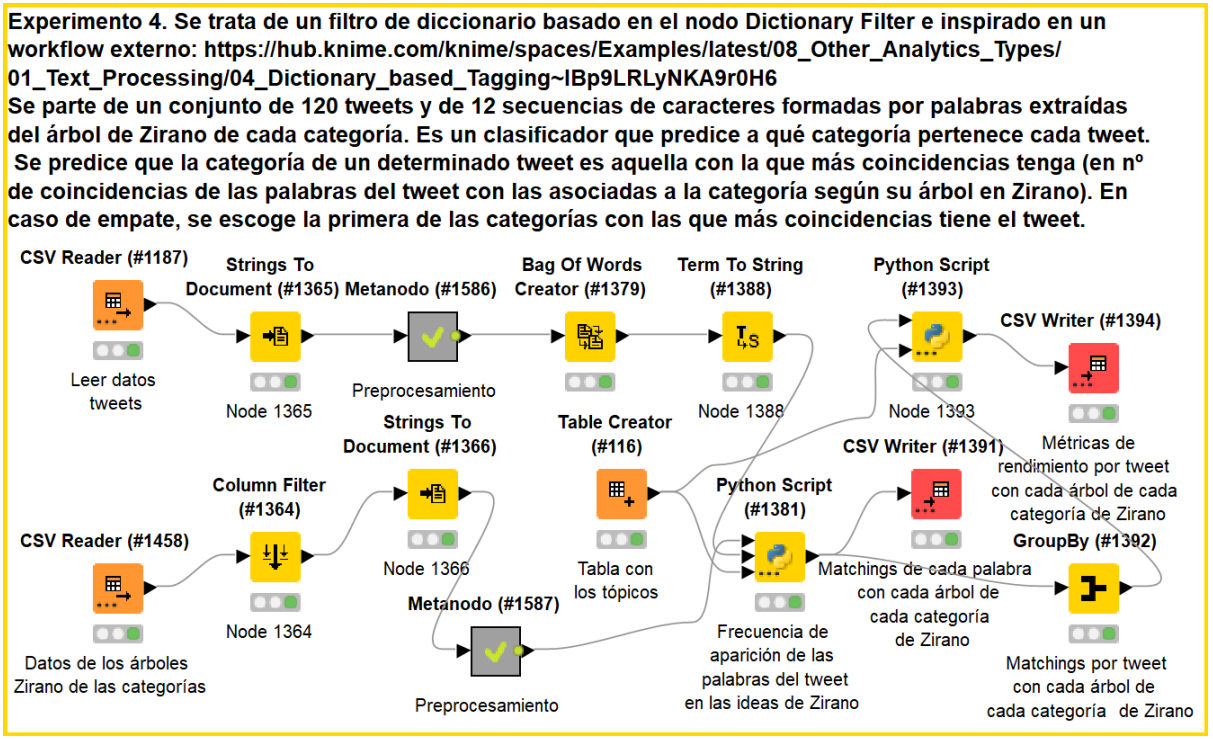

Figura 12: Flujo de trabajo del Experimento 4.

Resultados del clasificador basado en diccionario ideológico

En la Tabla 9 se muestra la matriz de confusión de este experimento y en la Figura 55 se presentan distintas métricas obtenidas con este clasificador, como por ejemplo el *accuracy*.

Los resultados del clasificador son razonablemente buenos ya que se obtiene aproximadamente un 64,16% de *accuracy*, *precision* y *recall*, lo cual aunque indica que se predicen bastantes falsos positivos y falsos negativos es un resultado positivo teniendo en cuenta que el método de clasificación es muy sencillo, basado en un diccionario con palabras estáticas, y los resultados indican que más de la mitad de las veces clasifica correctamente.

4.5 Experimento 5

En este experimento se intentan clasificar en categorías 120 tweets (datos de test) empleando 12 textos basados en las palabras de los árboles de Zirano de las categorías (datos de entrenamiento). En la Figura 13 se puede observar el flujo de trabajo asociado a este experimento.

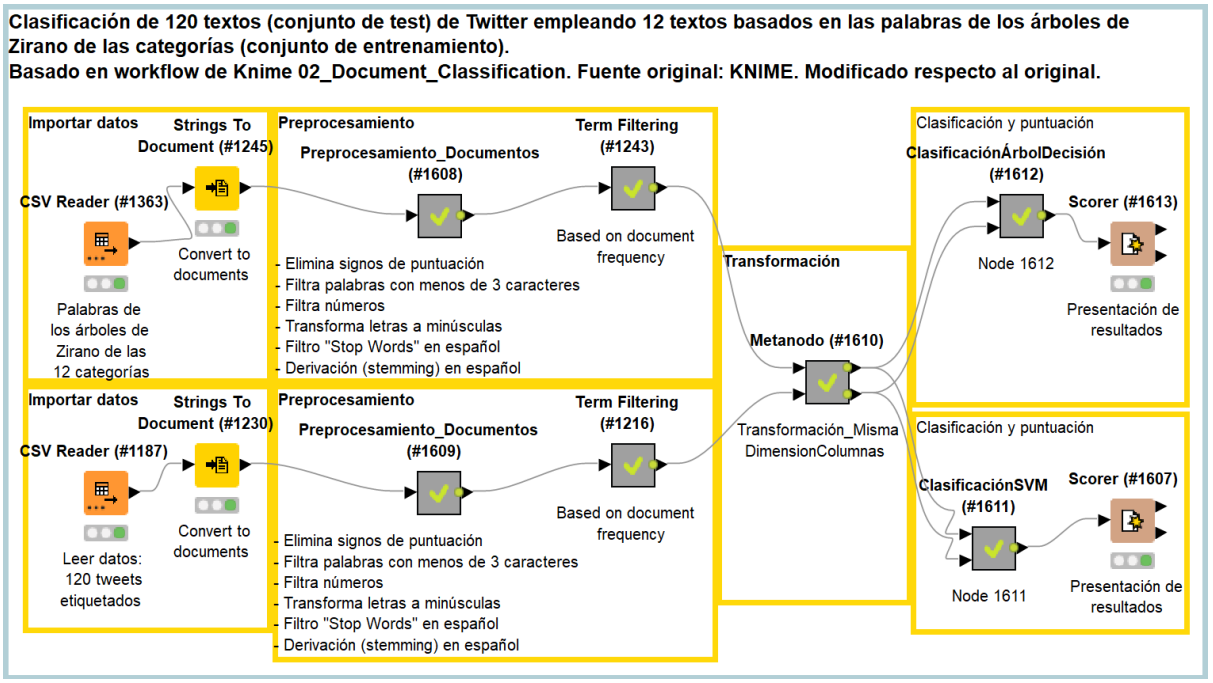

Figura 13: Flujo de trabajo del Experimento 5.

Resultados del experimento con árbol de decisión como clasificador

En la Tabla 10 se muestran los resultados del Experimento 5 empleando árbol de decisión como clasificador. Los resultados con el árbol de decisión son muy bajos, no llegando al 10% de acierto, según el valor de *accuracy* (8,30%). Cabe destacar que las muestras de la categoría de Tienda si que las suele predecir bien como se ve con sus 9 predicciones verdaderas positivas aunque su *precision* es baja y su *recall* elevado, lo cual se ve reflejado en el gran número de falsos positivos que predice. No obstante, mientras que las muestras de categoría Tienda las suele predecir bien, el resto no porque nunca las predice bien o solo una vez en el caso de Atletismo. Esto puede ser debido a que la matemática que subyace en la algoritmia de los árboles de decisión es sencilla y no es capaz de separar claramente tantas categorías con los datos de entrada dados.

Resultados del experimento con SVM como clasificador

En la Tabla 11 se muestran los resultados del Experimento 5 empleando SVM como clasificador. Los resultados obtenidos con SVM son razonablemente buenos, sobre todo teniendo en cuenta que la proporción de datos de entrenamiento frente a los de test es muy baja (en proporción, 1 de cada 10 datos son para entrenamiento). El resultado del *accuracy* es del 87,50% de acierto, lo cual refleja la potencia del uso de campos conceptuales (los de las 12 categorías de Zirano) para predecir las categorías de los tweets. No predice perfectamente ninguna de las categorías pero la mayoría obtienen resultados que llegan al *recall* o al *accuracy* perfectos (valor 1) o están relativamente cerca (valor >= 0,9).

El peor caso para el *recall* con un valor relativamente elevado de *precision*, valores de 0,6 y 0,857, respectivamente, se obtiene con la categoría de Internet, lo que indica que a veces se identifican mal los textos de esa categoría (falsos negativos). En cambio, el peor valor para la *precision*, valor de 0,692, en conjunto con un valor elevado de *recall*, valor de 0,9, se obtiene para la categoría de Préstamo, lo que parece indicar que identifica tweets de otras categorías como si fueran de la suya propia (falsos positivos), como se ve reflejado en los 4 falsos positivos que tiene asociados según la matriz de confusión.

4.6 Experimento 6

Este último experimento trata de clasificar textos enriquecidos con campos conceptuales de Zirano. Se emplean 120 tweets en total: 70% de ellos se enriquecen (empleando ciertos sustantivos de este conjunto de tweets) y se destinan al entrenamiento del modelo y, 30% de los datos originales se usan para test. En la Figura 14 se puede observar el flujo de trabajo asociado a este experimento.

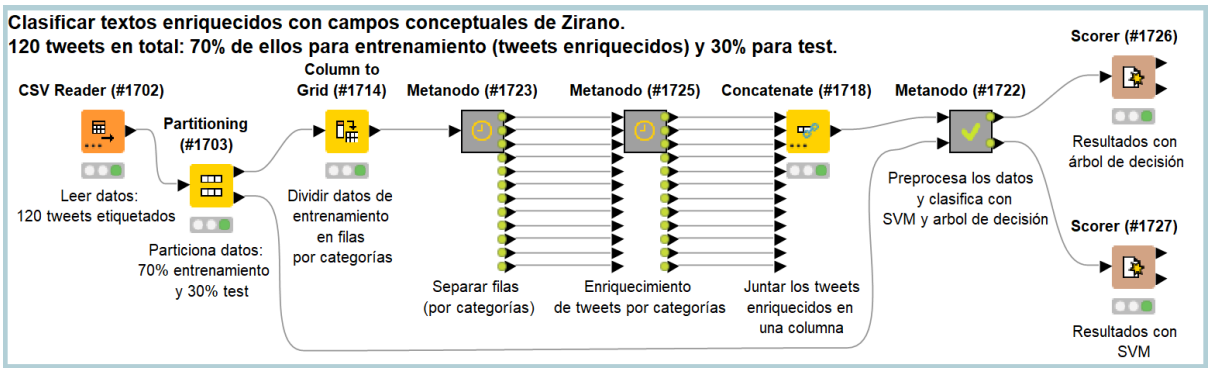

Figura 14: Flujo de trabajo del Experimento 6.

Resultados del experimento con árbol de decisión como clasificador

En la Tabla 12 se muestran los resultados del Experimento 6 empleando árbol de decisión como clasificador. Los resultados con el árbol de decisión son razonablemente bajos, obteniendo un 36,10% de acierto, según el valor del *accuracy*. Cabe destacar que todas las categorías que aparecen en la tabla a partir de Dinero y por debajo, nunca las predice bien. Esto puede ser debido a que la matemática que subyace en la algoritmia de los árboles de decisión es sencilla y no es capaz de separar claramente tantas categorías con los datos de entrada dados.

Resultados del experimento con SVM como clasificador

En la Tabla 13 se muestran los resultados del Experimento 6 empleando SVM como clasificador. Los resultados obtenidos con SVM son razonablemente elevados, pues el *accuracy* da un resultado del 77,80% de acierto en las predicciones. Cabe destacar que predice perfectamente (valores 1 de *recall* y *precision*) las categorías de Automóvil, Ordenador y Barco.

Los peores casos para el *recall* con un valor relativamente elevado de *precision*, valores de 0,667 y 1, respectivamente, se obtienen con las categorías de Natación, Fútbol y Viaje, lo que indica que a veces identifica mal los textos de esas categorías (falsos negativos). En cambio, el peor valor para la *precision*, valor de 0,5, en conjunto con un valor elevado de *recall*, valor de 1, se obtiene para la categoría de Atletismo, lo que parece indicar que identifica datos de otras categorías como si fueran de la suya propia (falsos positivos), como se ve reflejado con los 3 falsos positivos que tiene asociados según la matriz de confusión.

4.7 Conclusiones de los experimentos

Los experimentos realizados son coherentes con los objetivos propuestos en el proyecto. Los resultados obtenidos con la introducción del “texto aumentado” no mejoran el clasificador y permiten sacar las siguientes conclusiones parciales sobre el trabajo:

- KNIME es una plataforma que promete ser instalada en múltiples Sistemas Operativos. No obstante, he detectado que hay diferencias en cuanto a funcionalidad dado que durante la realización del trabajo había workflows que funcionaban sin problemas en Windows pero no en MacOS X.
- El problema de versiones con la plataforma KNIME es de mucho calado debido a la frecuencia de actualizaciones de versión del entorno. Esto puede provocar que se dejen sin actualizar algunos nodos que se requiera seguir usando y acaben como sistemas legados. Es posible que también aparezcan incompatibilidades entre versiones al querer usar un nodo o flujo de trabajo antiguo en una versión nueva.
- Los nodos específicos para el tratamiento de textos dependen de la instalación de paquetes de terceros que no siempre permiten una configuración detallada que permita entender ni parametrizar o especificar su funcionamiento. Esto limita la utilidad del entorno de prototipado rápido.
- Observando los resultados obtenidos en los experimentos realizados, se destacan el mejor y el peor caso en cuanto a los sistemas de minería de texto desarrollados:
  - El mejor sistema de minería de textos desarrollado con el conjunto de datos que se ha probado es el más básico de los que emplean las técnicas de SVM y árbol de decisión: el del experimento 1, obteniendo un 100% de acierto en la predicción (y un 94,4% de acierto con SVM), empleando árbol de decisión (como clasificador y predictor) y entrenando con un conjunto del 70% de los tweets originales y validando con otro conjunto con el 30% de los tweets originales.
  - Por otro lado, los sistemas de minería de textos con los que peores resultados se han obtenido son los de los experimentos 3 y 5, con los que se ha obtenido un porcentaje de acierto (accuracy) del 8,3%, lo que puede deberse a que la base matemática de la algoritmia que emplea la técnica de árbol de decisión es es demasiado sencilla para generar un "separador" de categorías adecuado.
- La idea de introducir campos conceptuales para la ampliación del vocabulario (enriquecimiento de texto) en la detección de categorías de texto no ha mejorado la capacidad de predicción de los clasificadores. En este sentido, creemos que los malos resultados se deben a dos problemas concretos:
  - Los textos de Twitter son demasiado pequeños como para que este enfoque sea eficaz. En un texto con pocas palabras es poco probable en general que haya muchas coincidencias, al menos exactas, con palabras relacionadas con una categoría o tema. A esto puede añadirse que los tweets pueden contener faltas de ortografía con lo que se pierde una posible coincidencia exacta con una palabra relacionada con una categoría.
  - Las limitaciones de KNIME no facilitan el análisis del rendimiento de nodos críticos en el workflow (como el específico de BoW) para entender su implementación concreta y así poder parametrizar su funcionamiento adecuadamente para el caso estudiado.

Capítulo 5 - Conclusiones y trabajo futuro

En este último capítulo se presentan las conclusiones derivadas del trabajo realizado. Así mismo se presentan los tiempos de dedicación al proyecto y algunas ideas en las que se podría trabajar en el futuro para mejorar el trabajo.

5.1 Conclusiones sobre el trabajo

- Se ha aprendido mucho sobre la minería de textos que es un campo muy amplio y que no se había trabajado apenas durante el grado de ingeniería informática. En concreto se ha adquirido un mayor conocimiento de técnicas de preprocesamiento de los datos como el BoW, la asignación de etiquetas POS para enriquecimiento de texto, el “stemmer”, y algunas otras. También se ha aprendido más sobre algunas herramientas de minería de datos que pueden emplearse en minería de textos, como son el SVM, los árboles de decisión e incluso una técnica no supervisada y orientada a la clasificación de textos como es el LDA la cual se ha llegado a probar en KNIME aunque no se recoge en esta memoria.
- Un aspecto que me ha gustado experimentar en el desarrollo en un entorno de prototipado rápido como es KNIME es que a pesar de no requerir escribir código, al menos para ciertas tareas, tiene la capacidad de combinar la programación visual con código de una manera sencilla y soportando además varios lenguajes de programación. Bajo mi punto de vista, esta capacidad hace que este tipo de entornos sean más versátiles en general frente a emplear un solo lenguaje de programación al permitir elegir el lenguaje de programación a usar según la tarea a resolver.
- El propio objetivo general del trabajo sobre desarrollar un sistema de minería de texto (clasificador de texto finalmente) me ha resultado interesante porque el tema de minería de datos, especialmente lo relativo a sus técnicas y, en general, la inteligencia artificial me llaman la atención.
- Sobre emplear campos conceptuales para obtener información adicional sobre el lenguaje no existe mucha documentación al respecto en lenguaje castellano y, combinarlo con la minería de texto ha supuesto un reto además de que la experimentación en ambos temas es nueva para mí. En KNIME hay herramientas de soporte para enriquecimiento de textos como por ejemplo el nodo “Stanford Tagger” que permite insertar etiquetas en las palabras de un documento, pero hay otros nodos que no soportan el idioma castellano.
- Por otro lado, en la práctica especialmente, se ha adquirido experiencia y aprendido más sobre entornos de prototipado rápido al obtener información de algunos de ellos y especialmente desarrollando el trabajo y probando utilidades de KNIME. Aunque esto también tiene su lado negativo en el sentido de que estos entornos no permiten en general llegar a un nivel de detalle tan fino al implementar como el que se puede alcanzar programando en un lenguaje de programación o empleando un IDE, de no ser que integren en la misma plataforma un lenguaje de programación (posiblemente limitados en algunos aspectos) como Python y R en el caso de KNIME, cuyo uso ha facilitado mucho el desarrollo de este trabajo.
- Se ha podido comprobar tras los experimentos 1 y 2 como la aplicación de técnicas muy sencillas de minería de texto, como el BoW y el TF y, su combinación con clasificadores que son ampliamente usados en el ámbito, como el SVM y el árbol de decisión, funcionan muy bien para la clasificación multiclase de textos de tamaño reducido y con un conjunto de datos no muy grande (del orden de 10 textos por categoría a clasificar). De hecho, según los resultados obtenidos en los experimentos, el sistema de minería de textos que mejores resultados obtiene es el más básico de los que emplean las técnicas de SVM y árbol de decisión: el del experimento 1, obteniendo un 100% de acierto en la predicción empleando árbol de decisión.
- El enriquecimiento de texto mediante campos conceptuales probado en la detección de categorías de texto no ha mejorado la capacidad de predicción de los clasificadores. Posiblemente en parte debido a que los textos de Twitter son demasiado cortos (con lo que es menos probable que sus palabras coincidan exactamente con palabras relacionadas con una categoría) y a que pueden contener faltas de ortografía y, también debido las limitaciones de KNIME y del tiempo disponible para el trabajo con lo que no se ha llegado a profundizar en entender la implementación de nodos críticos en los workflow (como el específico de BoW).

5.2 Dedicación

En este apartado se presenta la estimación del tiempo de dedicación al trabajo (Tabla 2) basado principalmente en los tiempos que se han ido registrando durante el trabajo.

Categoría	Tiempo total invertido aproximadamente
Búsqueda de información	33 horas y 30 minutos
Gestión	43 horas y 15 minutos
Diseño, implementación y experimentación	161 horas 30 minutos
Memoria	127 horas y 45 minutos
Reuniones	15 horas
TOTAL	**381 horas**

Cuadro 2: Tabla resumen con la información del tiempo invertido por categorías en el trabajo final de grado.

En total se han invertido 381 horas aproximadamente en el trabajo de fin de grado. Los tiempos se han ido registrando principalmente teniendo en cuenta el tiempo invertido en cada intervalo de tiempo en el que se ha trabajado y, habitualmente de forma aproximada aunque en alguna ocasión se han estimado. Normalmente se ha registrado una descripción del tipo de tarea llevada a cabo asociada a cada marca de tiempo registrada con lo que se ha distribuido el conteo de tiempos en 5 categorías:

- Búsqueda de información: incluye documentación, buscar información para el estado del arte, etc.
- Gestión: que incluye la revisión de normativas del TFG, elaboración de correos con preguntas a secretaría y al director del trabajo, gestión de las referencias y tiempos de dedicación, etc.
- Diseño, implementación y experimentación: en la que se ha registrado el tiempo invertido en decisiones de diseño, en llevar a cabo el diseño (creación de BBDD por ejemplo) y en la implementación de los sistemas y experimentos realizados, así como la documentación de los workflow parcialmente y también la depuración del sistema (código Python).
- Memoria: tiempo invertido en elaborar esta memoria, ligeros cambios en los workflow para extraer las figuras para la memoria (por ejemplo, parte de la documentación y cambios en nodos).
- Reuniones: son las reuniones mantenidas con el director del trabajo.

5.3 Trabajo futuro

- Desarrollar nodos específicos o usar programas externos que permitan parametrizar o implementar tareas críticas del workflow como la creación de la bolsa de palabras o los específicos de Aprendizaje.
- Una vez realizados los experimentos se ha podido apreciar que el uso de campos conceptuales en textos de poca extensión (tweets en este trabajo) no mejora los resultados. Por ello, aplicar las ideas de este trabajo a la clasificación de artículos de periódicos o documentos extensos, a priori, parece una buena opción en la que se podrían obtener mejores resultados en cuanto a la clasificación de textos.
- Probablemente podrían mejorarse los resultados seleccionando las categorías en Zirano de forma que no difieran más de “x” ramas entre sí los árboles de Zirano de las categorías y, además, habría que comprobar que esta diferencia se mantiene en el umbral establecido tras filtrar los datos de los árboles.
- Se podría extender el trabajo realizando análisis de sentimiento de los tweets, empleando campos conceptuales provenientes de otras fuentes y modelos de texto más potentes como *N-gramas*.
- Otra idea es traducir los textos al inglés (automáticamente) y emplear herramientas disponibles para este idioma para tratar los textos y, después, realizar la clasificación. Posteriormente, se podrían volver a traducir los textos a la inversa, de inglés a castellano, y comprobar la coherencia de los resultados.

Índice de figuras

Índice de cuadros

Anexos

Anexo 1: Bases de datos

Este anexo se compone de dos apartados cada uno de los cuales trata sobre un sistema gestor de bases de datos (*SGBD,* en inglés) empleado para montar una base de datos diseñada para el proyecto. Entre los dos apartados se tratan aspectos tales como: la descarga e instalación de software necesario para usar cierto SGBD (al menos parte de él), el diseño del modelo E/R de la base de datos, la creación de las tablas que componen la base de datos, etc.

A.1.1 Base de datos empleada: SQLite

El sistema gestor de bases de datos (o *SGBD)* SQLite se ha llegado a probar para almacenar la información de ciertos documentos (textos -tweets de Twitter- y su clave correspondiente) así como la de las categorías asociadas a los documentos (básicamente los nombres de las posibles categorías consideradas en el proyecto).

Se trata de un SGBD relacional, implementado en “C”, “rápido” y “multiplataforma” [46].

Se ha decidido emplear SQLite en base a la guía del director del trabajo tras problemas encontrados para emplear una BBDD Oracle en KNIME. Además, las BBDD de datos SQLite son ampliamente utilizadas, de hecho, en su página web se dice que “SQLite is the most used database engine in the world” [46].

Para almacenar los datos de los textos a clasificar se ha llegado a construir una SGBD SQLite con 3 tablas:

- Clase: con la información (cadena de caracteres predefinida) de las posibles categorías a las que puede pertenecer un texto determinado.
- Documento: que almacena los textos recopilados.
- Clasificado_como: que es una tabla intermedia que almacena la información de a qué categoría (clave ajena a la tabla “Clase”) pertenece cada uno de los textos (clave ajena a la tabla “Documento”).

Cabe mencionar que la BBDD no se ha llegado emplear para recuperar datos sino que para ello y, en general, como soporte de almacenamiento persistente para gestionar los datos del proyecto, se han empleado principalmente ficheros CSV debido por ejemplo a su mayor sencillez de uso en KNIME y a que se pueden crear varias versiones del mismo. Sin embargo, sí se ha llegado a probar con éxito a introducir datos en la BBDD con un workflow similar al presentado en el Anexo A.4.4.

A continuación, se muestran los detalles de la instalación y creación de la estructura de tablas de la BBDD empleada (SQLite).

Descarga e instalación

Página principal de SQLite: [46]

En cuanto a la descarga, como se trabaja desde Windows 10, se va a descargar un binario precompilado para Windows 10 para mayor simplicidad en la instalación. La versión descargada se llama: *sqlite-tools-win32-x86-3350500.zip* que incluye utilidades para administrar ficheros de la base de datos y los programas como el *sqldiff.exe*, etc.

Una vez descargado se descomprime en la carpeta donde se quiera almacenar los programas de SQLlite. Por ejemplo, en el directorio *C:\Program Files\SQLite* en Windows.

“Configuración” de la “BBDD” para usarla desde la “consola”

En la Figura 15 se muestra parte de la configuración para emplear SQLite desde la consola de comandos.

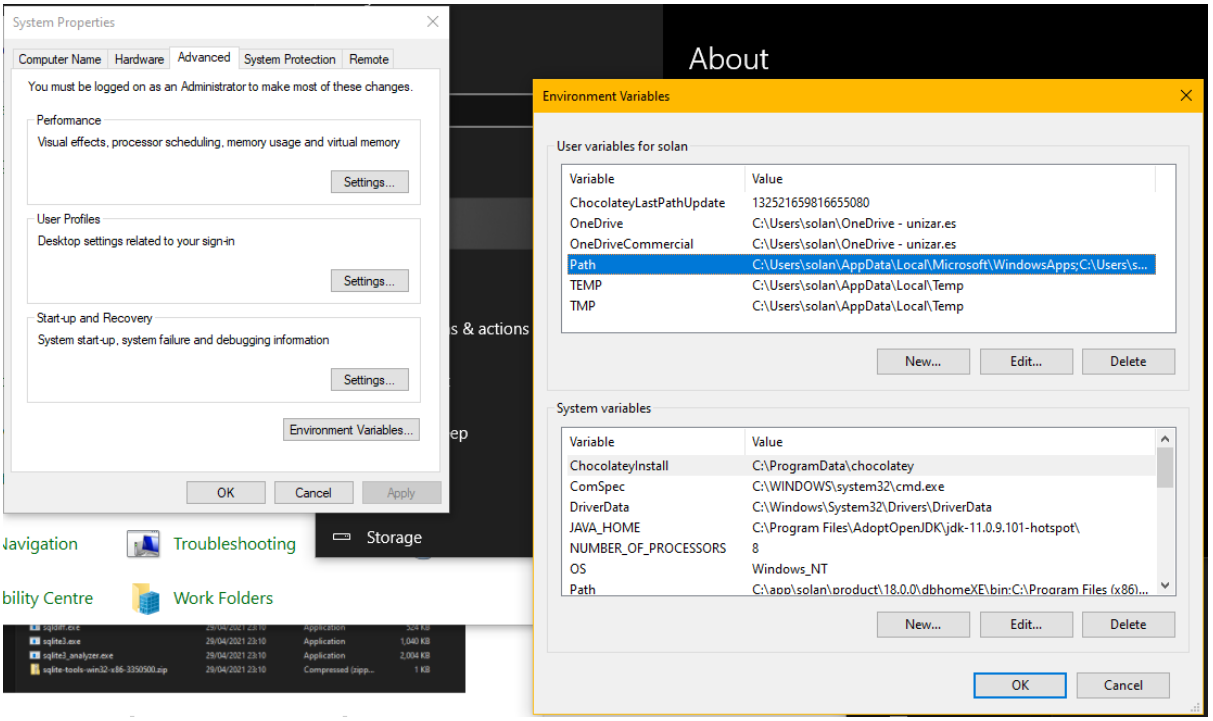

Figura 15: Configuración parcial para usar SQLite desde la consola.

Añadimos la ruta a los programas de SQLite al path: “C:\Program Files\SQLite”. El programa sqlite se puede lanzar desde consola de comandos (cmd en Windows) con el comando sqlite3.

Programa con GUI para visualizar los datos de la BBDD

Se ha descargado el “DB Browser for SQLite - Standard installer for 64-bit Windows”. Enlace a descargas: [47]

Los pasos de la instalación del programa son sencillos (continuar con los típicos botones de “Next” y “Finish” al final, etc.),aunque requiere algunas configuraciones por ejemplo las relativas a los *shorcuts.* Quizás aparezca un mensaje pidiendo aceptar algún permiso o similar, en ese caso podemos contestar afirmativamente para continuar con la instalación. En la Figura 16 se muestra la interfaz gráfica de esta herramienta.

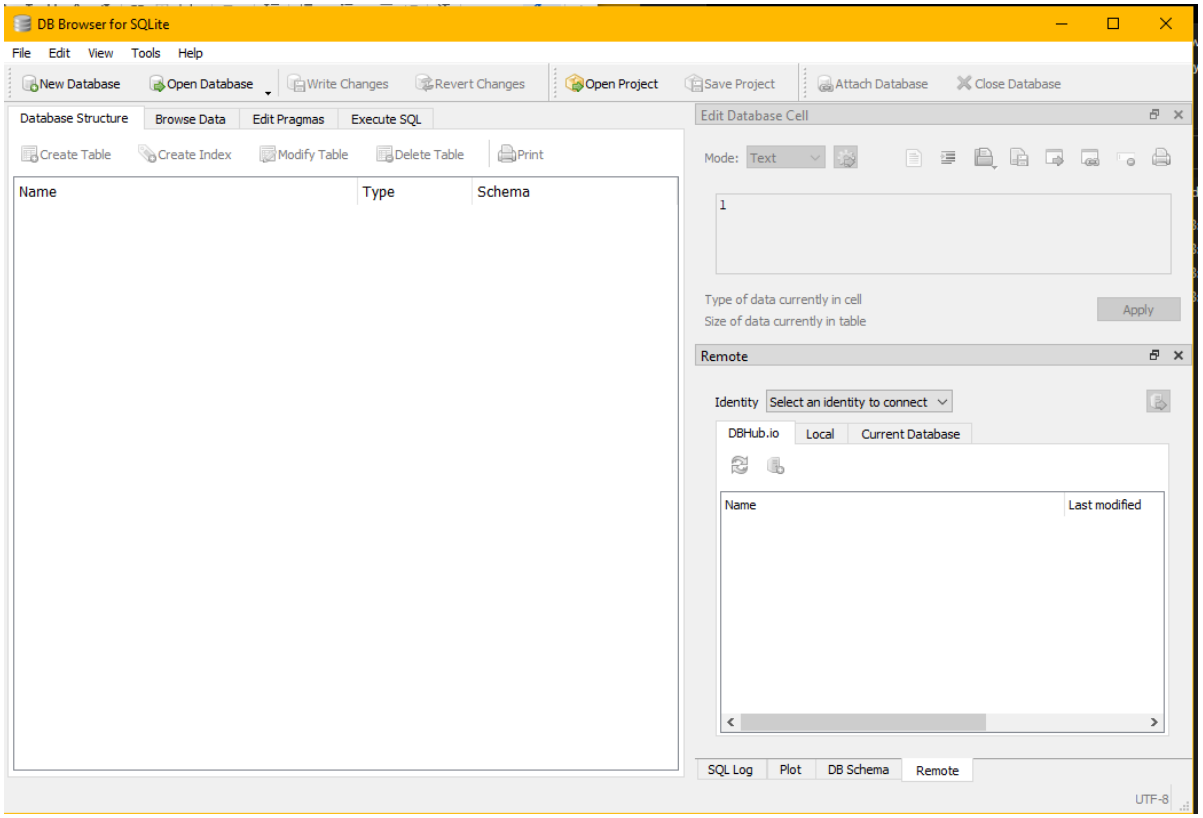

Figura 16: Interfaz gráfica de usuario (“GUI” en inglés) del programa DB Browser para SQLite.

Creación de la BBDD en SQLite

Se ha partido del fichero BBDD_topics.sql con el esquema de la BBDD que se puede ejecutar en un SGBD Oracle pero adaptándolo para que valga para SQLite. Se han omitido algunas restricciones que tenía este fichero SQL original, como la del número máximo de caracteres para el dato “Texto“ de la tabla “Documento” y el dato “Topico” de la tabla “Clase”.

En la Figura 17 se muestran algunas características de la tabla “Documento”:

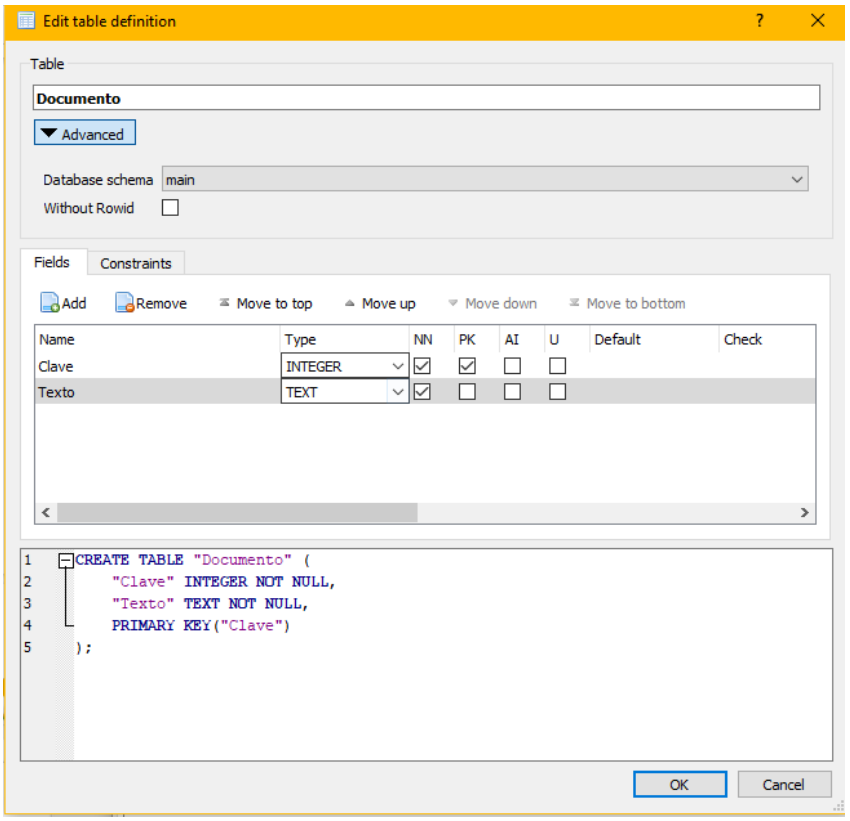

Figura 17: Creación de la tabla “Documento” para SQLite.

En la Figura 18 se muestran algunas características de la tabla “Clase”:

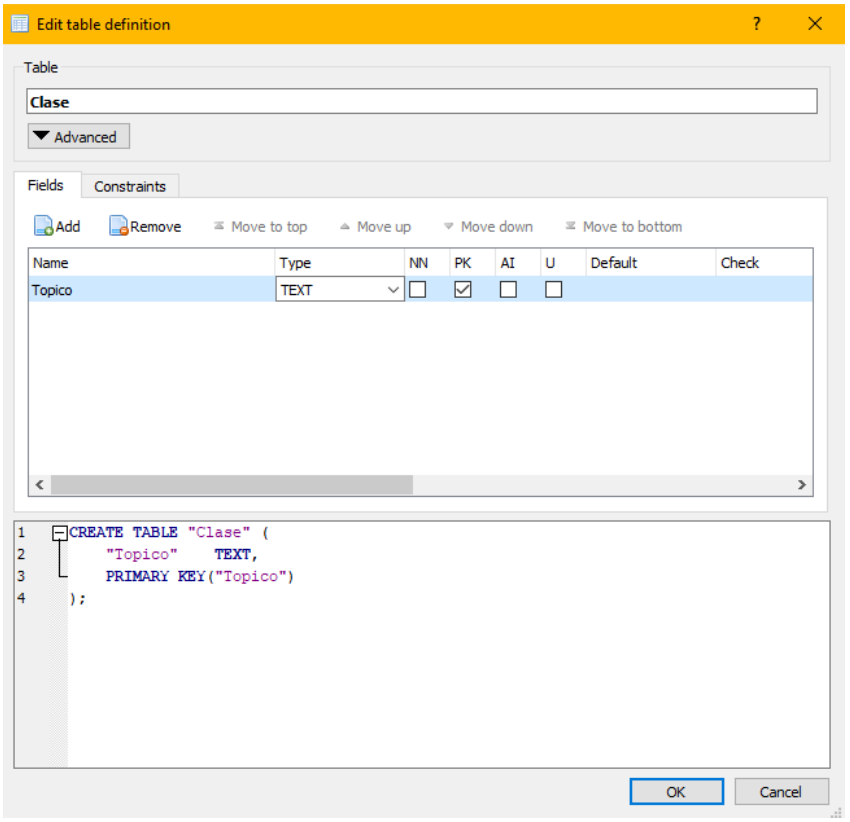

Figura 18: Creación de la tabla “Clase” para SQLite.

En la Figura 19 se muestran algunas características de la tabla “Clasificado_como”:

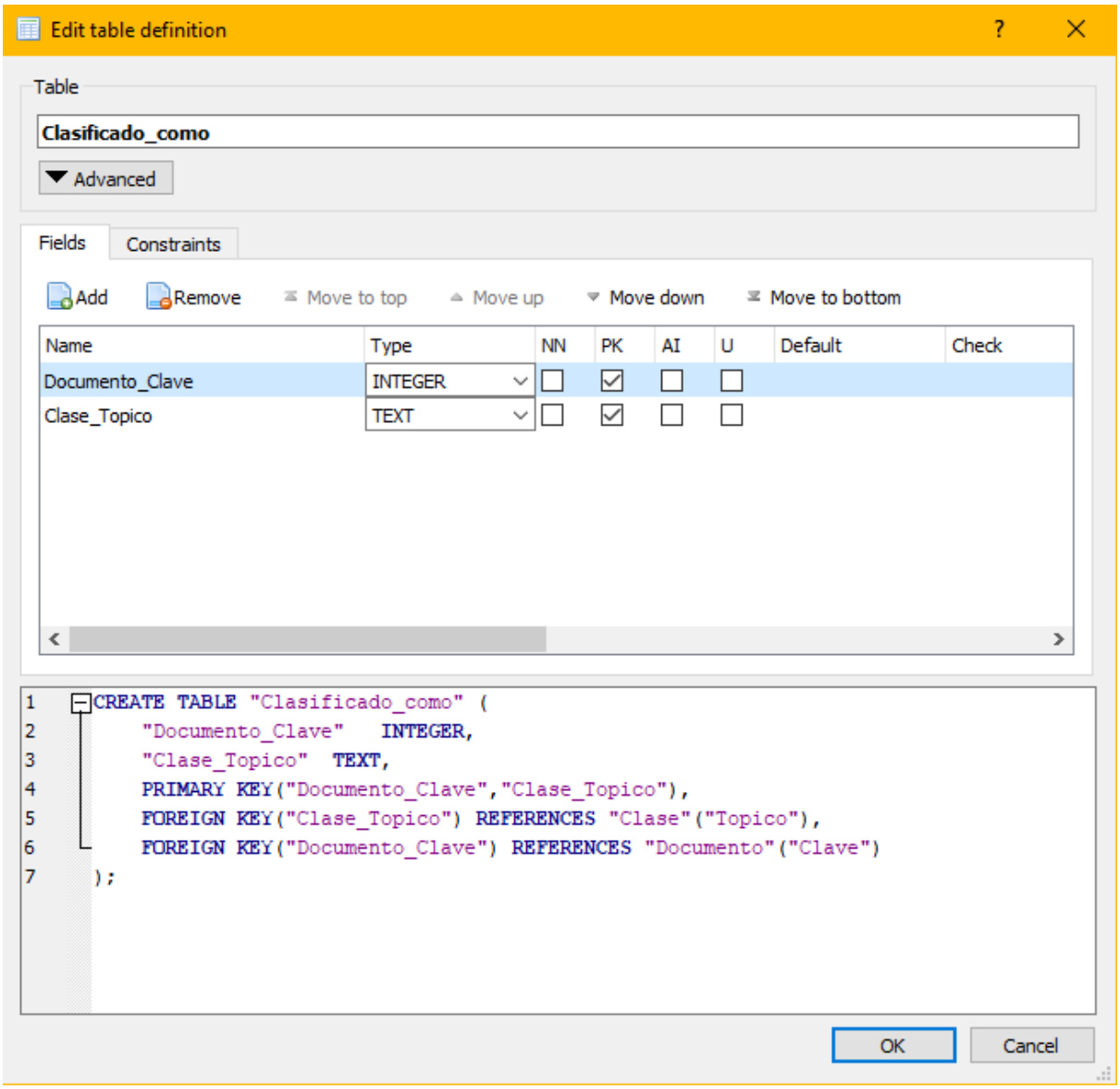

Figura 19: Creación de la tabla “Clasificado_como” para SQLite.

A.1.2 Base de datos: Oracle

La BBDD Oracle que se comenta en este apartado se creó para el trabajo pero finalmente se decidió cambiar a una SQLite (la que se comenta en el apartado anterior).

En la base de datos se pretenden almacenar los textos (su contenido) y las categorías a las que pertenecen cada uno de ellos. Para diseñar el modelo entidad-relación de la base de datos (BBDD) se ha decidido emplear un software llamado *DBDAp*. Este programa permite realizar gráficamente el modelo entidad-relación y posteriormente transformarlo a modelo relacional e incluso pasar de este último a código en formato SQL para distintos sistemas gestores de bases de datos (SGBD). Más Información sobre DBDAp:

- Descarga de la herramienta DBDAp: [48]
- Artículo (con documentación) sobre DBDAp: [49]

Cabe resaltar que la herramienta DBDAp ha sido desarrollada por compañeros universitarios de cursos pasados en Proyectos Final de Carrera que fueron dirigidos por el profesor Sergio Ilarri. Además, esta herramienta se conocía ya por haberla usado en algunas asignaturas del grado dentro de la especialidad de Sistemas de información, como por ejemplo: “Bases de datos 2” y “Almacenes y minería de datos”.

“Modelo E/R” de la base de datos [49]

A continuación, en la Figura 20 se muestra el "modelo E/R" diseñado mediante la herramienta DBDAp consistente en 2 entidades (“Documento” con la información de los textos y “Clase” con las posibles categorías de un texto) relacionadas por medio de la relación “Clasificado_como”:

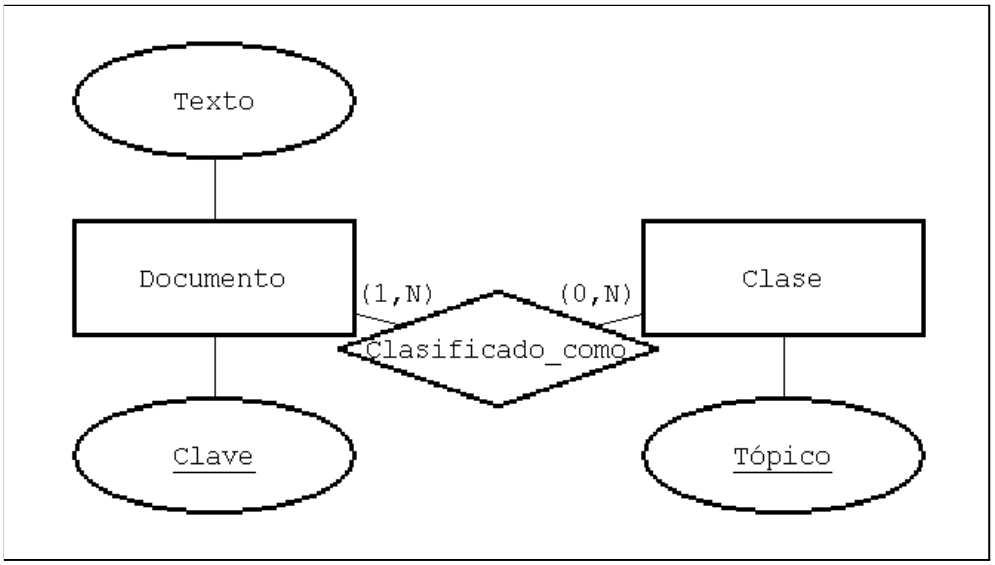

Figura 20: Modelo entidad-relación modelado en DBDAp.

Transformación del “modelo E/R” al modelo relacional [49]

Con la herramienta DBDAp podemos construir un modelo relacional a partir del modelo entidad-relación. De esta forma conseguimos una representación con un nivel más cercano a los SGBD y a partir de una de más alto nivel, como es el modelo entidad-relación. Por último, a partir de este modelo relacional se podrá generar un código SQL que ya sea ejecutable en un determinado SGBD con el que se construya el esquema correspondiente a la base de datos.

En DBDAp podemos transformar un modelo entidad-relación mediante los botones (alternativamente, desde el menú “Herramientas”) correspondientes a las etiquetas “Transformar” o “Transformar paso a paso”, la diferencia es que con el primero obtendremos el modelo relacional directamente y con el segundo se explicara cada transformación que se vaya realizando en el modelo entidad relación hasta obtener cierto modelo relacional.

A continuación, se muestra paso a paso la transformación del modelo entidad-relación diseñado en uno relacional:

1. En la Figura 21 se muestra la pantalla inicial del proceso de transformación del modelo entidad-relación a relacional:

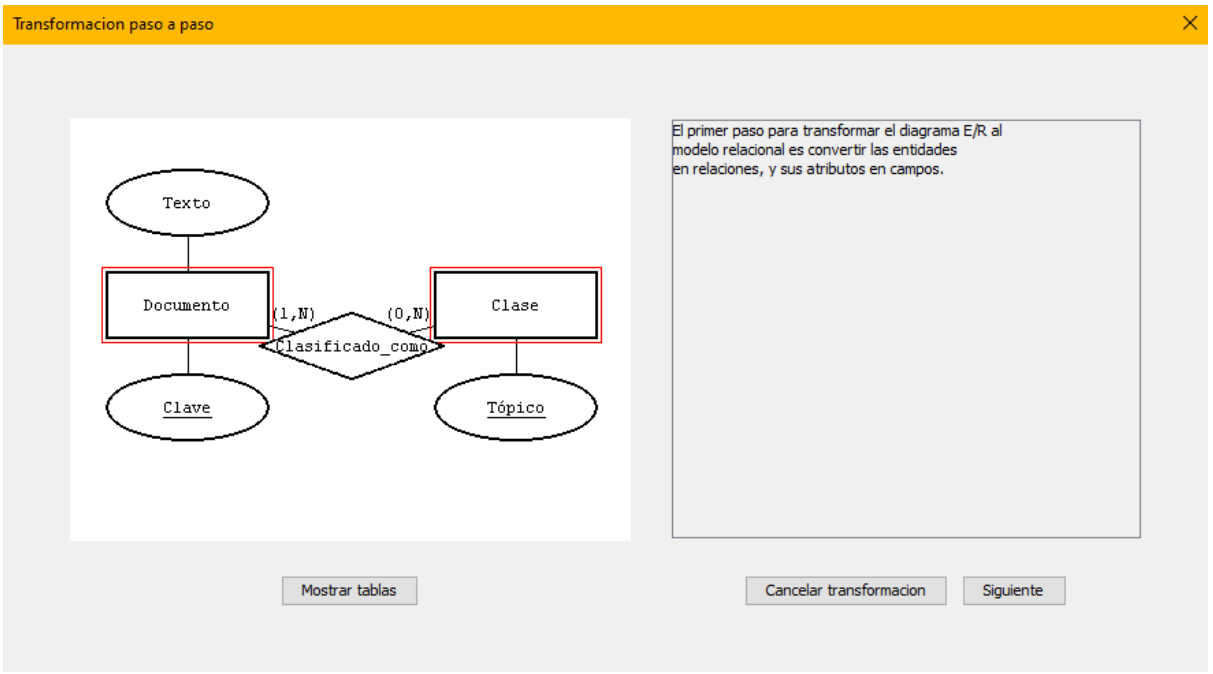

Figura 21: Pantalla inicial del proceso de transformación del modelo entidad-relación a relacional.

2. En la Figura 22 se muestra la pantalla del proceso de transformación del modelo entidad-relación a relacional tras convertir la entidad Documento en una relación:

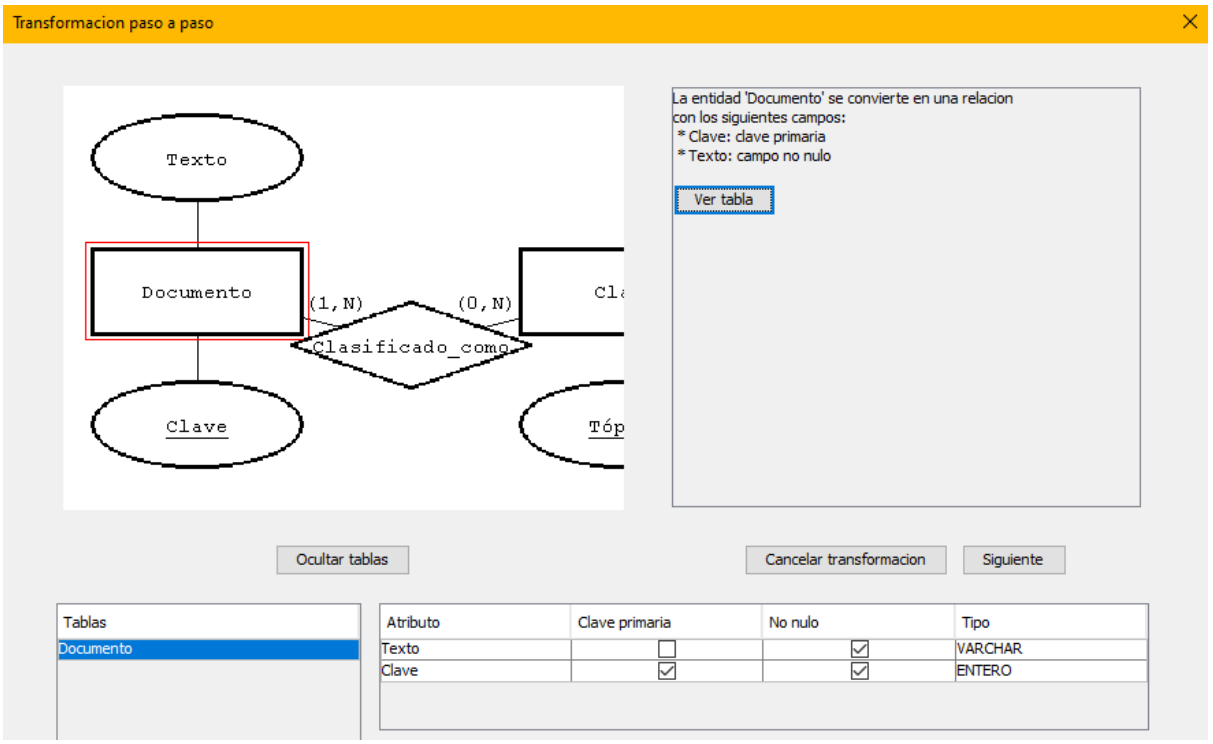

Figura 22: Transformación del modelo entidad-relación a relacional tras convertir la entidad Documento en una relación.

3. En la Figura 23 se muestra la pantalla del proceso de transformación del modelo entidad-relación a relacional tras convertir la entidad Clase en una relación:

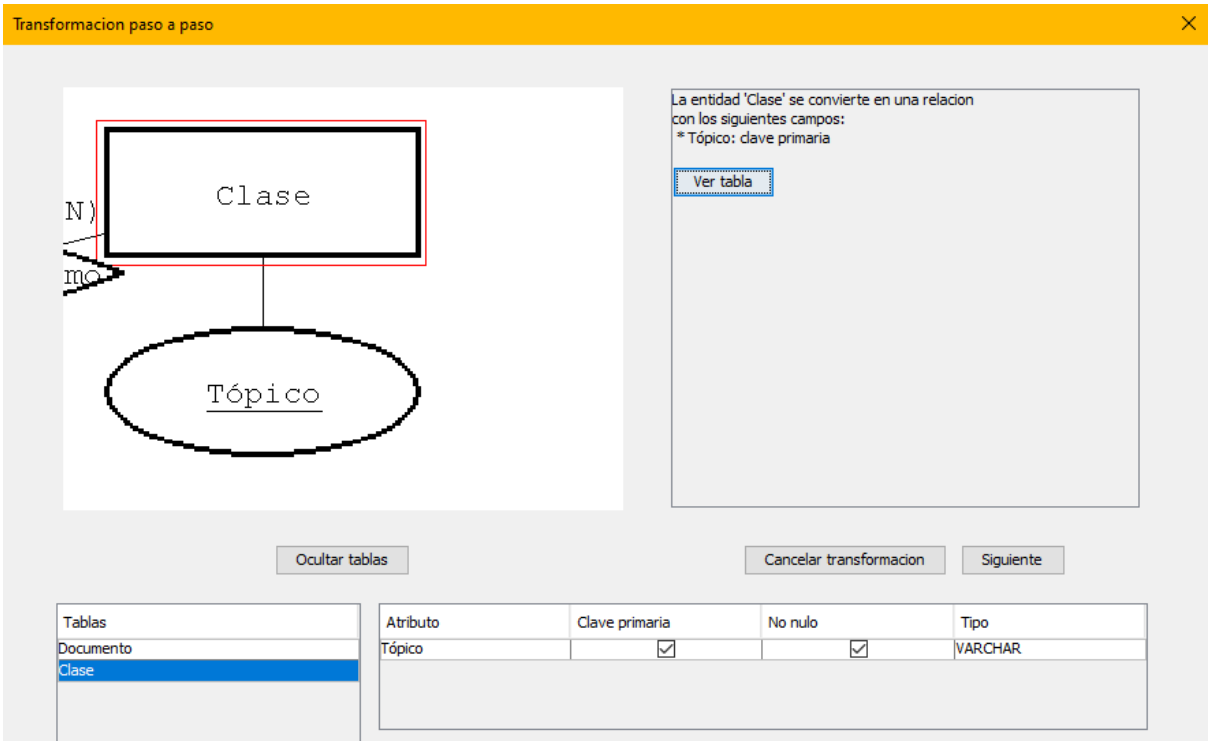

Figura 23: Transformación del modelo entidad-relación a relacional tras convertir la entidad Clase en una relación.

4. En la Figura 24 se muestra la pantalla previa a la conversión de las interrelaciones (en el modelo realizado sólo hay una, “Clasificado_como”) del diagrama E/R dentro del proceso de transformación del modelo entidad-relación a relacional:

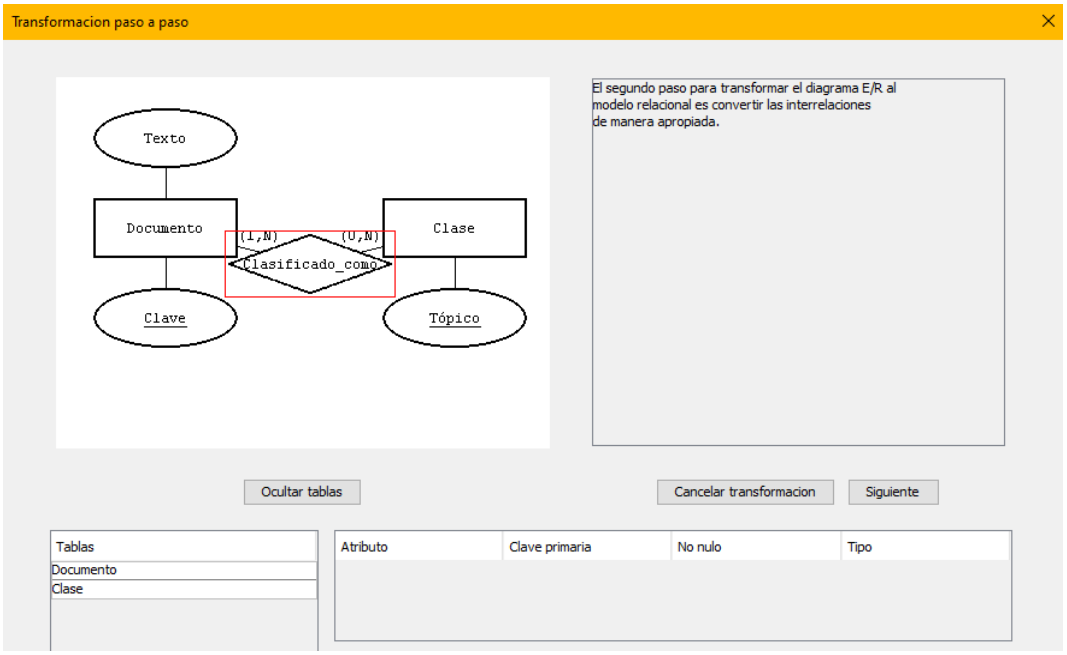

Figura 24: Pantalla previa a la conversión de las interrelaciones del diagrama E/R del proceso de transformación del modelo entidad-relación a relacional.

5. La Figura 25 muestra la pantalla del proceso de transformación del modelo entidad-relación a relacional tras convertir la interrelación “Clasificado_como” en una relación:

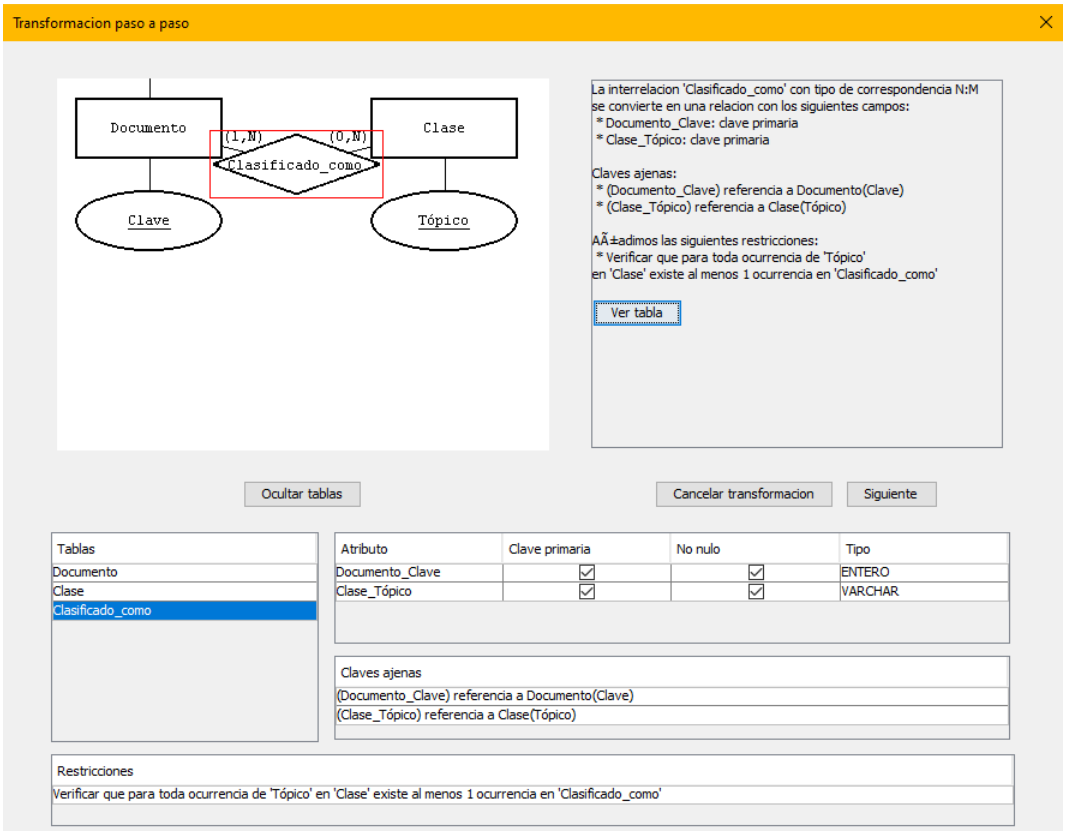

Figura 25: Transformación del modelo entidad-relación a relacional tras convertir la interrelación N:M “Clasificado_como” en una relación.

6. En la Figura 26 se muestra el esquema relacional resultante del proceso de transformación del modelo entidad-relación a relacional:

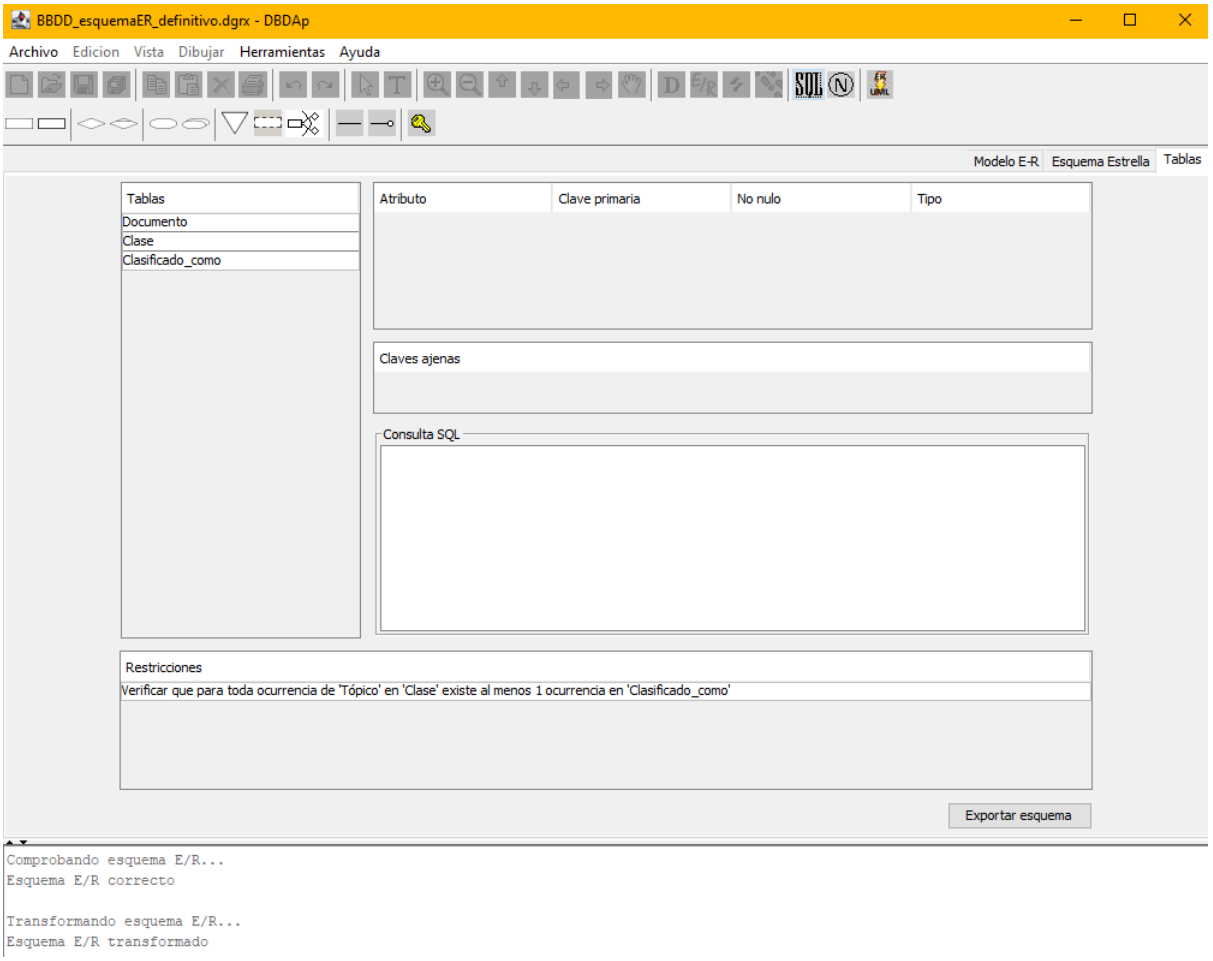

Figura 26: Esquema relacional con las tablas resultantes del proceso de transformación del modelo entidad-relación a relacional.

Obtener código en SQL para el SGBD

Como el SGBD que se usará es Oracle y cada SGBD presenta particularidades en el nombrado de tipos y otras características, se configura la herramienta DBDAp, desde el menú “Herramientas > Configuración”, para que el código final en SQL que genere sea adecuado para un SGBD Oracle.

Instalación del SGBD

Se ha optado por un software de base de datos local y, además, es gratuito. La cantidad de información que se va a gestionar en el trabajo no es muy elevada como para optar por una solución de almacenamiento de datos en la nube (cloud), además se evita así poner a disposición de servicios externos datos que se extraigan de fuentes de información de terceros con los posibles riesgos de privacidad asociados. Y, se estima a priori que la ejecución local de la BBDD ofrecerá un mayor rendimiento que una solución que requiera el uso de conexión a la red para intercambiar datos por los costes asociados a la transmisión de información vía online.

La base de datos a instalar es *Oracle 18c Express Edition*. Algunos de sus pasos de instalación se listan a continuación:

Pasos de la instalación de la BBDD:

1. Descargar archivo.
2. Descomprimirlo.
3. Abrir carpeta descomprimida.

4. Ejecutar instalador setup.exe (está dentro de la carpeta descomprimida, en el ejemplo se llama *OracleXE184_Win64).* Puede aparecer una ventana emergente en la que seleccionamos *"Yes"* o *"Sí"* en castellano.
5. Se puede avanzar por las distintas pantallas del instalador mediante botón "Next >" ("Siguiente >" o similar en castellano) o "Install" según la pantalla, aceptando los términos legales, configurando ruta para instalación, etc. También hay que rellenar en una pantalla los 2 campos con la contraseña elegida para acceder a la BBDD con las cuentas de SYS, SYSTEM y PDBADMIN.
6. Tras la instalación se muestra la última pantalla con información que puede ser interesante para la conexión de la BBDD con el exterior. Por último, hacemos click en "Finish" (o "Finalizar" o similar en castellano).

Herramienta con GUI para visualización y gestión de la base de datos: SQL Developer

Se ha optado por emplear el programa SQL Developer para gestionar la BBDD a crear, al menos, antes de conectarla a KNIME. Este programa permite crear una conexión con el SGBD Oracle instalado y mediante una pantalla tipo formulario permite ver gráficamente los datos almacenados en la BBDD y hacer gestiones o configuraciones de forma más sencilla que vía comandos en una terminal o consola de comandos.

Una vez descargado e instalado el programa (en Windows vía un instalador que no presenta complicaciones resaltables), se puede ejecutar el programa "sqldeveloper.exe". En su pantalla principal, dentro de la sección "Connections" podemos añadir una nueva conexión por ejemplo con el SGBD instalado previamente (Oracle 18c Express Edition) haciendo click en la flecha junto al icono verde de suma y, allí seleccionando con otro click la opción "New database connection…". Alternativamente, se puede establecer conexión con la base de datos al iniciar el programa. En este último caso, en la pestaña "Welcome Page" puede que aparezca una sección llamada "Database Connection" y, en su interior, otra llamada "Databases Detected" donde se mostrarían posibles bases de datos detectadas y seleccionando una de ellas, por ejemplo, "XE" se podría introducir un usuario y una contraseña adecuados para establecer la conexión.

En la Figura 27 se muestra una posible configuración de conexión hacia el SGBD Oracle instalado (estando este en su configuración por defecto tras la instalación):

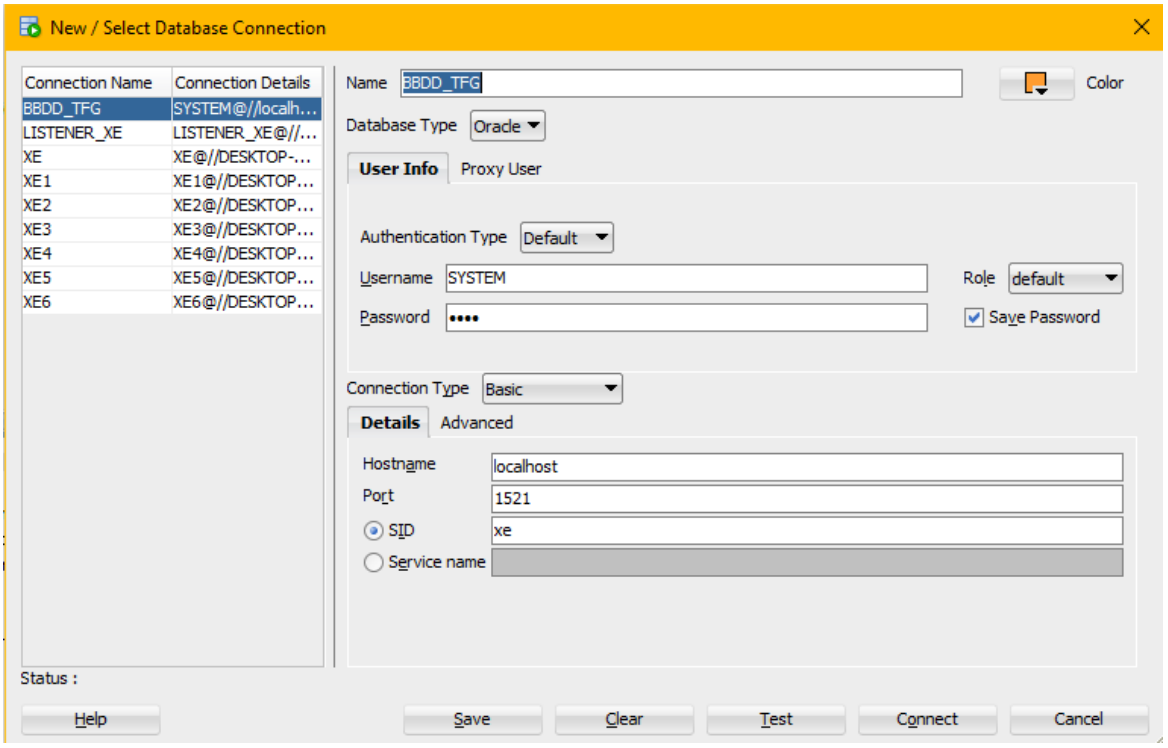

Figura 27: Posible configuración de conexión para BBDD Oracle en SQL Developer.

La contraseña a emplear en esta configuración es la que se ha especificado en una de las pantallas del instalador de Oracle.

Para guardar la configuración se puede hacer click en el botón “Save”. Si queremos comprobar que la conexión es correcta se puede hacer click en el botón “Test” y, si aparece “Status: Success” significa que ha funcionado correctamente. Entonces ya se puede finalizar la configuración haciendo click sobre el botón “Connect”.

Anexo 2: Software externo KNIME

En este anexo se presenta parte del software externo para KNIME empleado durante el transcurso del proyecto.

A.2.1 Extensiones y nodos externos para KNIME

En este apartado se comentan algunas extensiones y nodos externos de KNIME.

A.2.1.1 Emojis

Para filtrar los emojis de los tweets se ha empleado un nodo externo para KNIME [36].

A.2.1.2 Integración de KNIME con R

La extensión principal empleada en este trabajo para usar el lenguaje de programación R con *KNIME es: KNIME Interactive R Statistics Integration* (y alguna otra extensión más relacionada). A continuación, se describen formas que permiten integrar R con KNIME:

1. Una forma de instalar algunas de las extensiones referidas al comienzo de este apartado:

   Se instalan las extensiones que se pueden ver en [50]. Los pasos para la instalación una vez seleccionadas las extensiones que se muestran en la Figura 28 (la versión de las extensiones instaladas posiblemente difiera respecto las mostradas en la Figura 28 aunque coincidiendo en su versión mayor e incluso versión menor, las diferencias se estima que no deberían ser muy notables) son sencillos. Principalmente consisten en usar botón “Next >” para avanzar de pantalla, aceptar condiciones, usar botón “Finish” para finalizar, etc.:

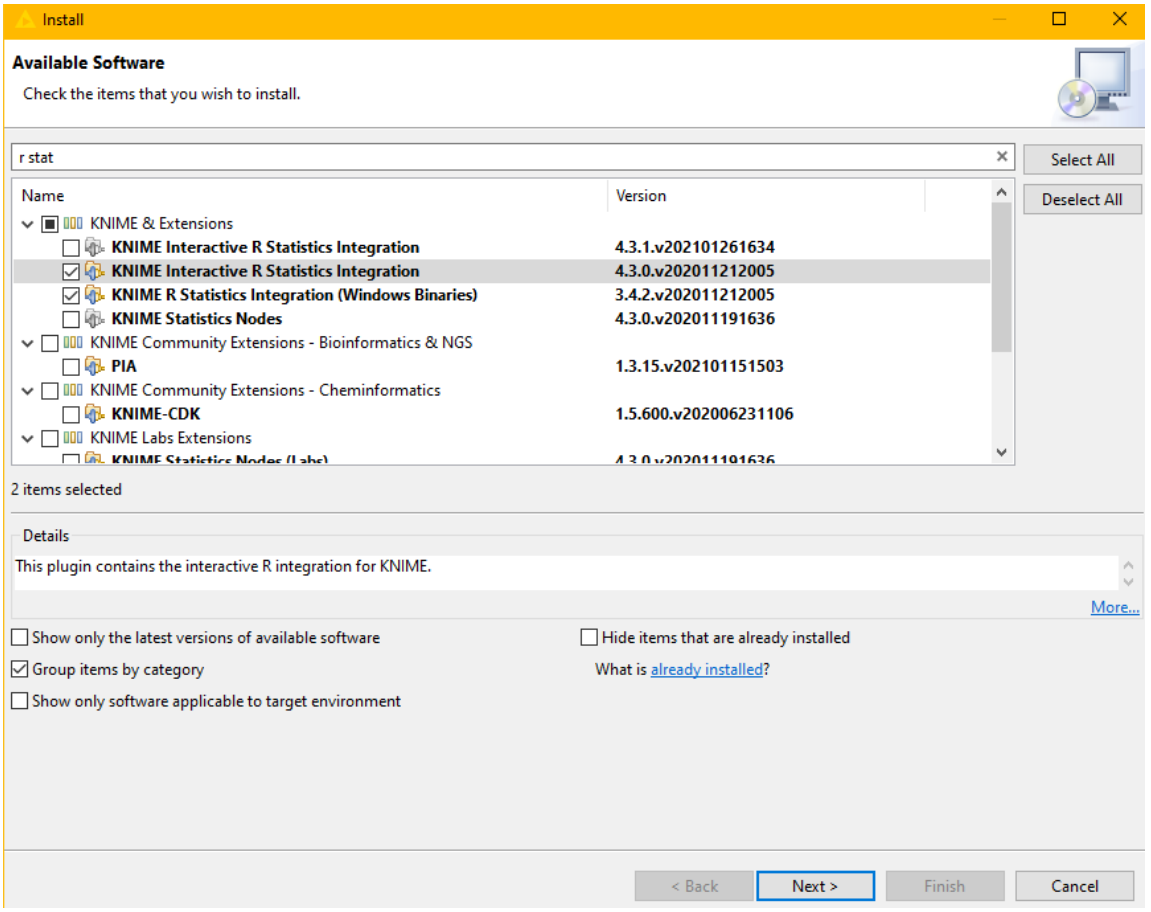

Figura 28: Posible instalación de extensiones de R en KNIME. Software a instalar: “KNIME Interactive R Statistics Integration” y “KNIME R Statistics Integration (Windows Binaries)”.

2. Para instalar un servidor R para KNIME [51]:

   Aunque finalmente se ha instalado gráficamente el paquete *Rserve*, en el programa *RStudio* se ha escrito un script (véase Figura 29) con la siguiente línea:

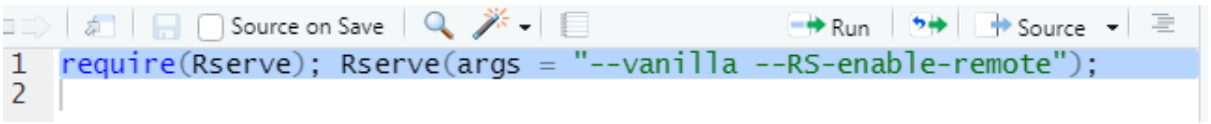

Figura 29: Instalación de *Rserve* mediante script de R.

3. Software adicional instalado:

   También se ha instalado en KNIME la extensión software: “KNIME R Scripting extension” (véase Figura 30). Se han aceptado los términos y después aparece un mensaje preguntando si reiniciamos para aplicar los cambios del software (podemos hacerlo más adelante).

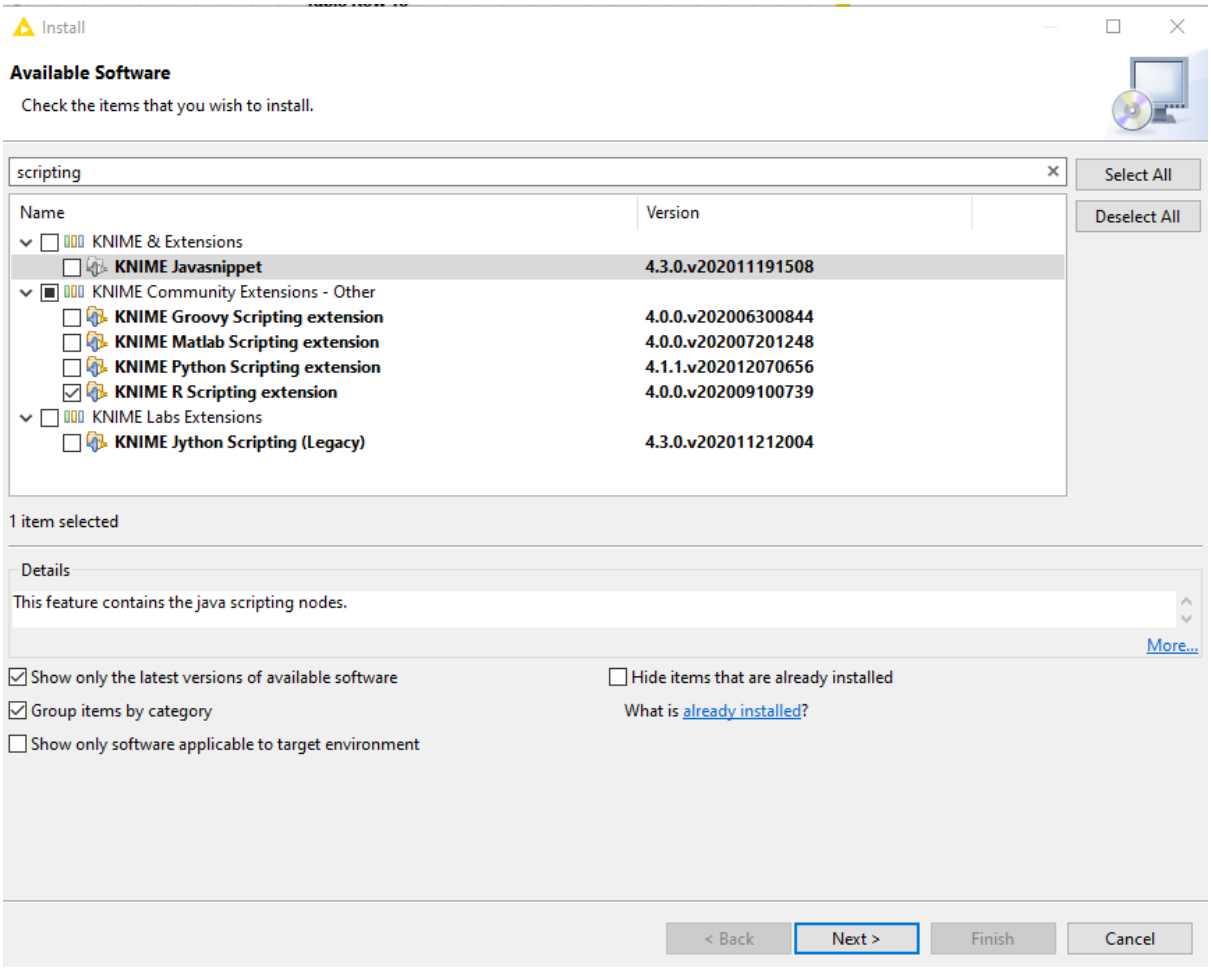

Figura 30: Instalación de extensión de R en KNIME. Software a instalar: “KNIME R Scripting extension”.

A.2.1.3 Python en KNIME

La descarga e instalación de Python en KNIME se ha llevado a cabo empleando la información de la web: [37].

Concretamente se han llevado a cabo los puntos 1, 3 y 4.

Algunos de los pasos (del 1 al 6 aproximadamente) mencionados a continuación se han llevado a cabo a partir de la información disponible en la web [37], la cual presenta una figura animada donde ejemplifica cómo realizar estos pasos.

1. Buscar “python” en esta web: [52]

2. Click en “Extensions”.
3. Seleccionamos (haciendo click) el nodo “KNIME Python Integration”.
4. Arrastramos el icono del nodo a un workflow de KNIME:
5. Click en “Yes” cuando se pregunta si instalar la extensión.
6. Se procede a instalar la extensión software de KNIME llamada “KNIME Python Integration” (Figura 31). Continuamos haciendo click en “Next >” en las siguientes ventanas.

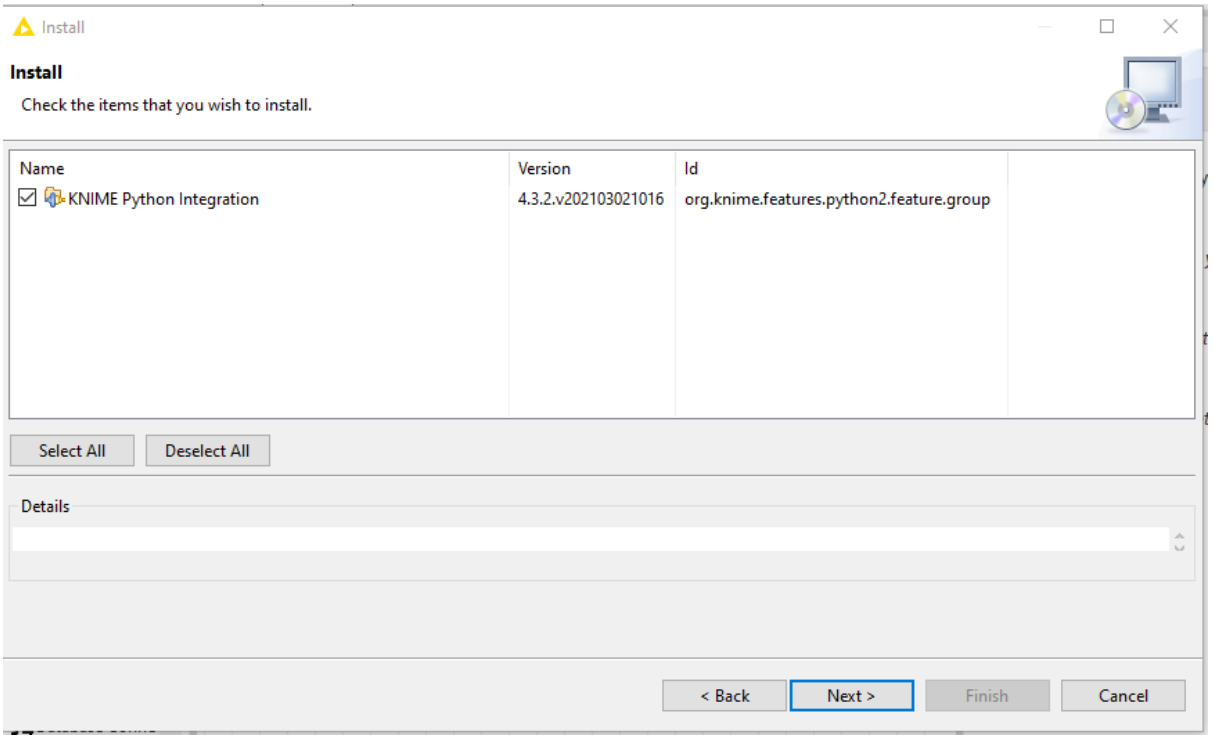

Figura 31: Instalación de extensión de Python en KNIME. Software a instalar: “KNIME Python Integration”.

7. Click en “Finish” tras aceptar los términos.
8. Por último, hacemos click en “Restart Now”.

Configuración del entorno Python en KNIME (Figura 32)

Se requiere también configurar “un entorno Python” con las “dependencias necesarias para la integración KNIME Python”, según se indica en [37].

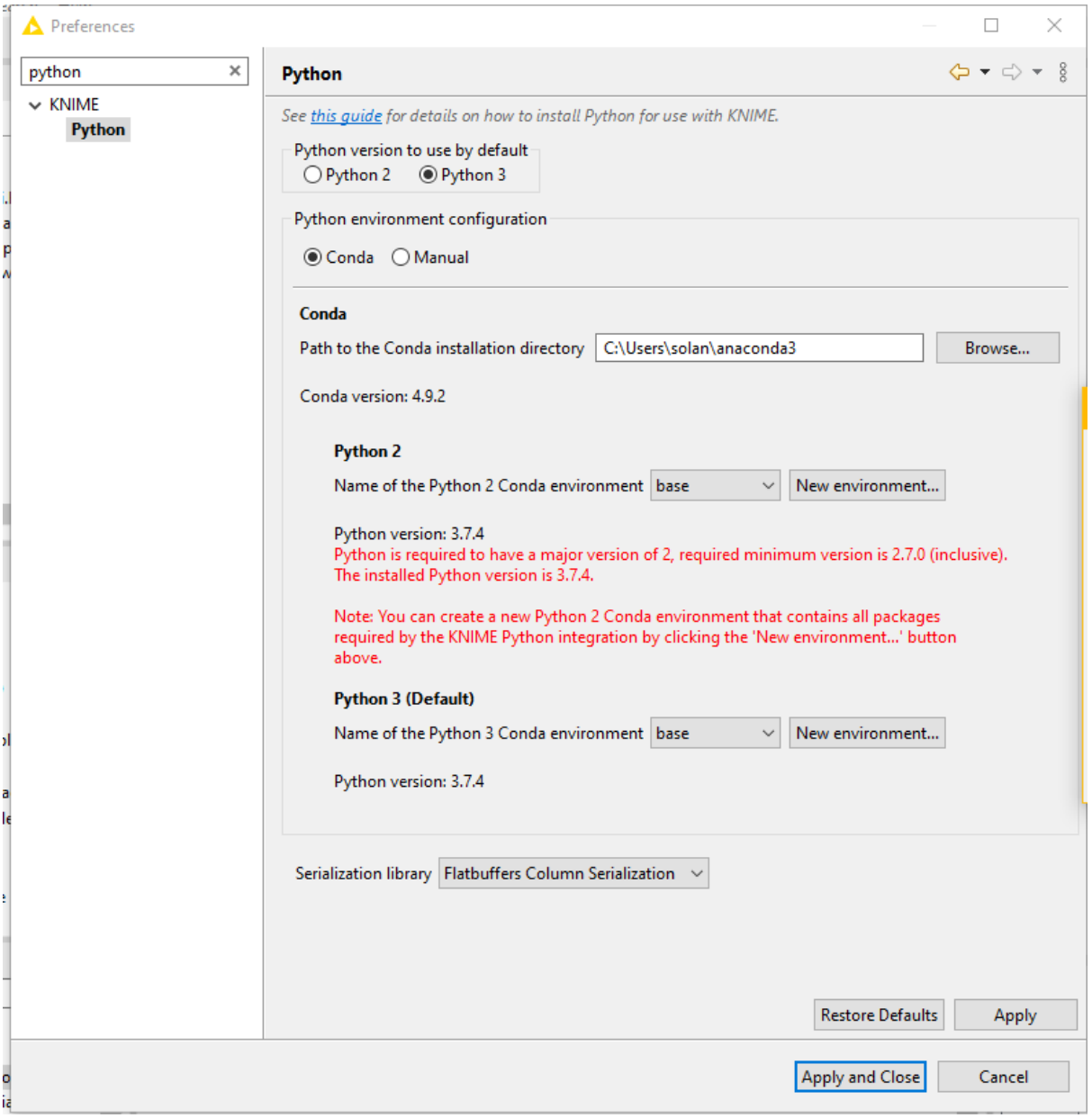

Figura 32: Pantalla principal de configuración de los entornos Python en KNIME.

Para solventar un posible problema que puede aparecer en rojo, como en la Figura 32, hacemos click en el botón “New enviroment…” situado en el lateral derecho y debajo de “Python 2” y creamos un nuevo entorno.

En la web [37], se indica que aparte de esta configuración previa realizada, se necesita tener instalado ya sea “Anaconda” o bien “Miniconda”.

En mi caso ya disponía del programa Anaconda pero he actualizado su versión (y también he realizado una instalación de un Launcher que ofrecía realizar el propio programa mediante una GUI). La actualización de Anaconda se ha llevado a cabo ejecutando un comando en una terminal abierta desde la GUI del navegador de Anaconda:

En la Figura 33 se muestra el navegador de Anaconda en el que se ha seleccionado “Environments” > “base(root)” > “Open Terminal” para obtener una terminal:

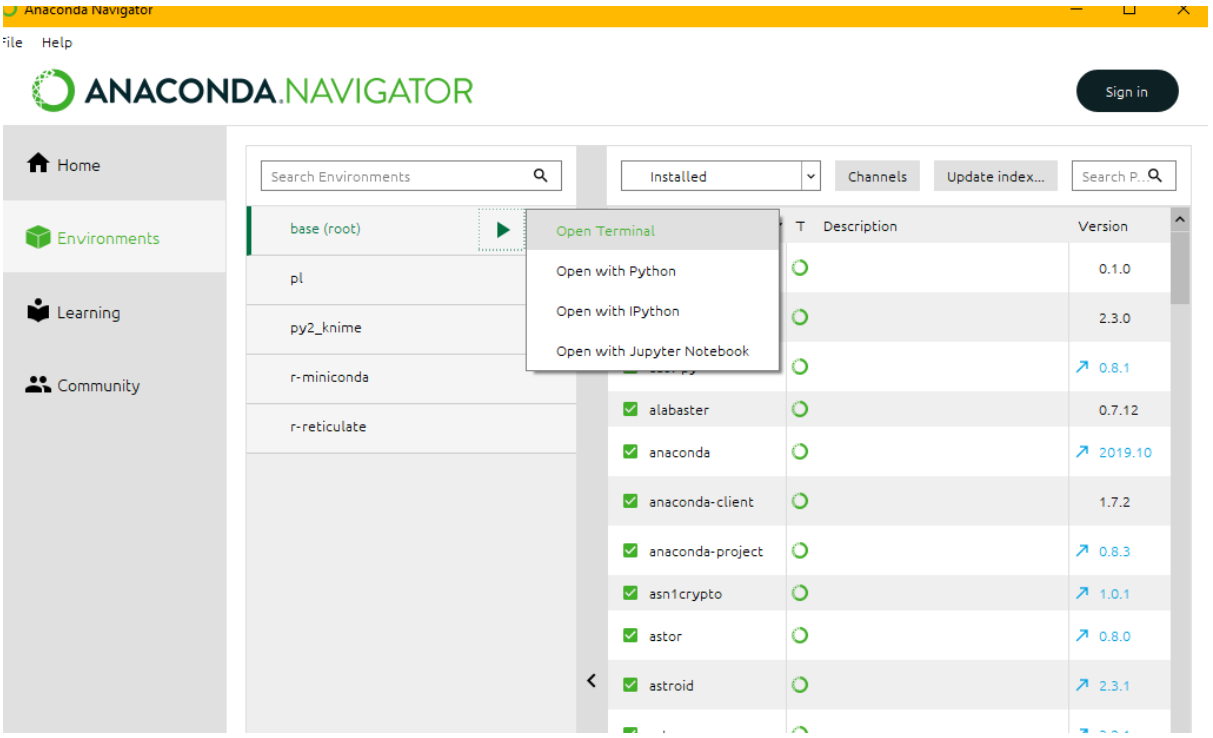

Figura 33: Obtención de una terminal desde el navegador de Anaconda.

El comando ejecutado en la terminal para actualizar Anaconda es “conda update -n base -c defaults conda”.

Finalmente, en la pantalla de configuración de Python en KNIME podemos hacer clic en el botón “Apply” y en “Apply and Close” después para terminar la configuración.

A.2.1.4 Tagged Document Viewer

Se ha instalado la extensión “KNIME JavaScript Views (Labs)” (Figura 34) para disponer de un nodo que permita ver los tags asociados a los términos de un documento en KNIME. La instalación no se incluye pero el proceso es trivial y similar al de otras extensiones.

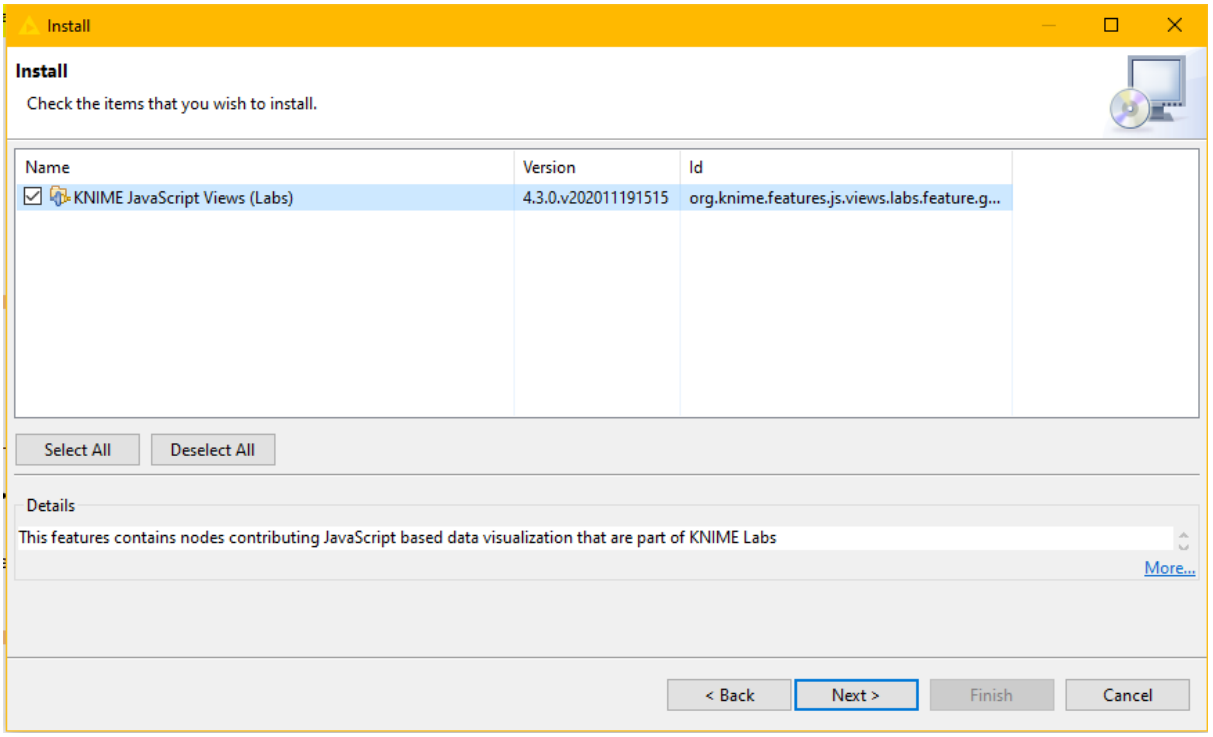

Figura 34: Instalación de extensión de KNIME con la que se obtiene el nodo “Tagged Document Viewer” que permite ver los tags asociados a los términos de un documento en KNIME. Software a instalar: “KNIME JavaScript Views (Labs)”.

A.2.1.5 Configuración de extensión Textprocessing

Como puede observarse en la Figura 35, desde KNIME en “File > Preferences > KNIME > Textprocessing” se ha configurado como tokenizador por defecto para los nodos que lo usan el “StanfordNLP SpanishTokenizer” (el único disponible en castellano).

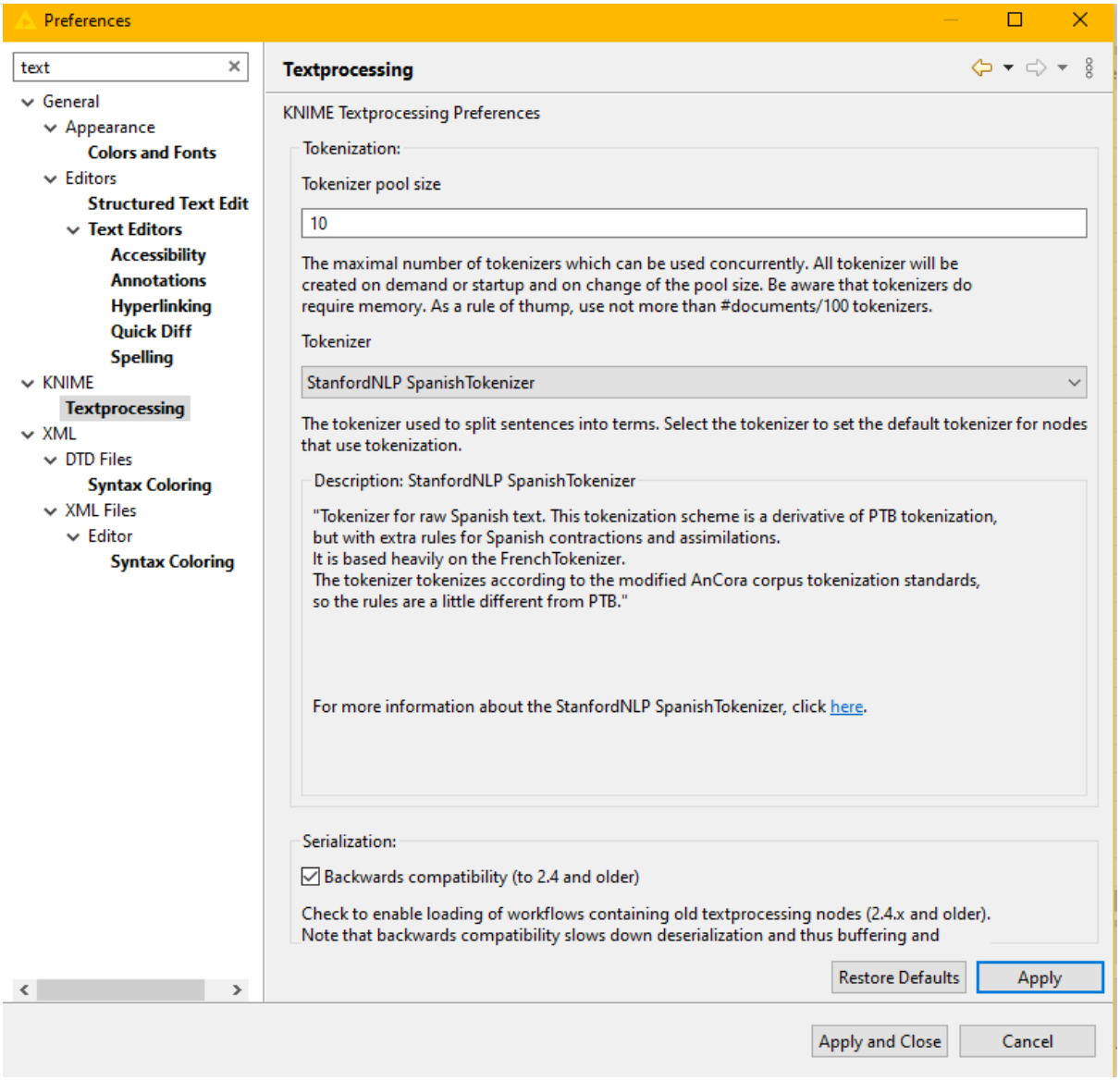

Figura 35: Configuración de extensión *Textprocessing* para usar por defecto el tokenizador “StanfordNLP SpanishTokenizer”.

A.2.1.6 Descarga e instalación de nodos Selenium para Web Scraping

El proceso de descarga e instalación de estos nodos se ha basado en lo que indica la web relativa a estos nodos Selenium [39].

Se ha decidido instalar la siguiente versión de los nodos correspondiente a “KNIME 4.3 software site”, porque en el momento de realizar este proceso se usa la versión de *KNIME 4.3.2*.

En el programa KNIME se ha ido a “File > Preferences” y allí dentro (buscando por palabras o navegando) nos situamos en la sección “Install/Update > Available software sites”.

Allí hacemos click en el botón Add e introducimos un nombre para el paquete que vamos a instalar (por ejemplo, SELENIUM nodes - KNIME 4.3) y añadimos un nombre (en el campo “Name”) y la siguiente URL (en el campo “Location”) para descargar los nodos: https://download.seleniumnodes.com/4.3. Como resultado obtendremos una nueva entrada en la lista de sitios software disponibles. Finalmente hacemos click en el botón “Apply and Close”.

Ahora navegamos a “File > Install KNIME Extensions...”, y buscamos los nodos de Selenium como se muestra en la Figura 36 por ejemplo:

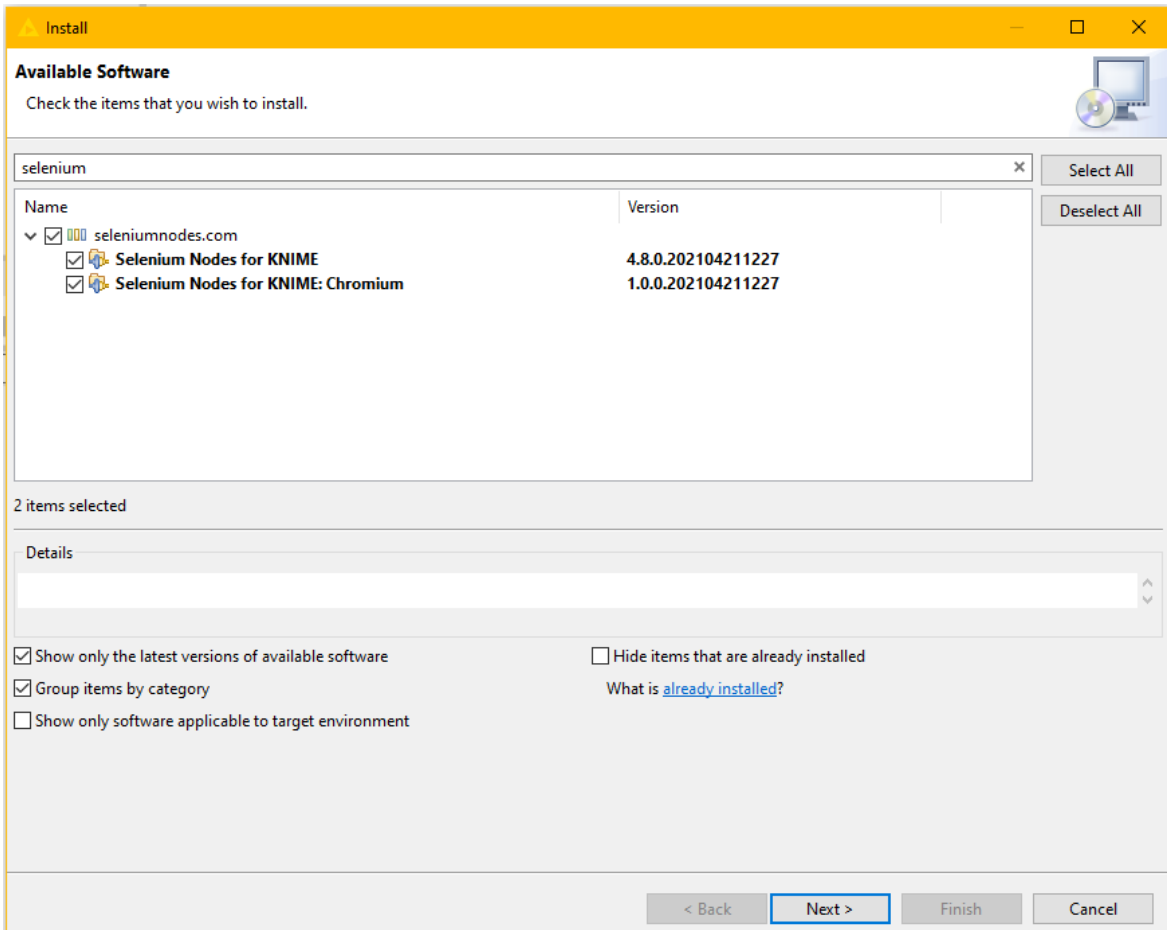

Figura 36: Instalación de extensión de KNIME para *Web scraping*. Software a instalar: “Selenium Nodes for KNIME” y “Selenium Nodes for KNIME: Chromium”.

Los dejamos seleccionados y hacemos click en “Next >” las veces que haga falta a lo largo de la navegación.

Aceptamos los términos que correspondan y avanzamos en la pestaña hasta finalizar la instalación. Si después aparece un mensaje emergente, también lo aceptamos para continuar con la instalación. Finalmente, nos pregunta si reiniciamos KNIME y aceptamos con click en “Restart now”.

Entonces veremos que ya están instalados los nodos de Selenium en el “Node Repository” de KNIME. Aunque hace falta incluir una licencia de Selenium para usarlos, que puede obtenerse gratuitamente según unas condiciones desde [39] y, que tras obtenerla debe incluirse en “KNIME > Selenium”. De esta forma ya se pueden usar los nodos de Selenium por ejemplo para web scraping.

A.2.2 Software externo (bibliotecas) para Python

En este apartado se introducen algunas de las bibliotecas para Python empleadas en la implementación del trabajo con KNIME.

A.2.2.1 Biblioteca treelib

En esta web se indica cómo instalar la biblioteca treelib para Python empleando Conda. Concretamente se ha instalado con el primero de los comandos que aparecen en [42] (el comando es: “install -c conda-forge treelib”). Al instalar esta biblioteca también estará disponible para usarse desde un nodo Python Script en KNIME con el entorno adecuadamente configurado.

Al crear un objeto árbol con la biblioteca *treelib*, se puede visualizar por la salida de la consola una representación gráfica del árbol creado con la función show() de esta biblioteca. Se ha probado a emplear esta función show() sin argumentos (aparte del dato que representa el árbol con el que se llama a la función) en el nodo Python Script de KNIME y no funciona correctamente. Se obtiene un mensaje de error por la consola relacionado con la codificación (el error es similar a: “*charmap codec can’t encode characters in position 7-9: character maps to <undefined> [...]*”). Sin embargo, se ha solucionado mirando una documentación sobre la función show() y empleando el argumento “line_type=’ascii’ ya no se obtiene el error y se puede visualizar correctamente el árbol.

A.2.2.2 Pyarrow

Para poder emplear en KNIME la biblioteca de serialización Apache Arrow se requiere tener instalada la biblioteca *pyarrow* que puede obtenerse con Anaconda como se ejemplifica en [53]. Se puede configurar desde “File > Preferences > KNIME > Python” con Serialization library “Apache Arrow”.

Anexo 3: Profundización en técnicas de PLN

En este anexo se explican con más detalle algunas técnicas basadas en redes neuronales empleadas en PLN.

A.3.1 RNN

La red neuronal recurrente (Figura 37) se basa en las “completamente conectadas de feedforward”. Aplican la misma función a todos los datos de entrada pero el resultado para un dato depende del anterior [3].

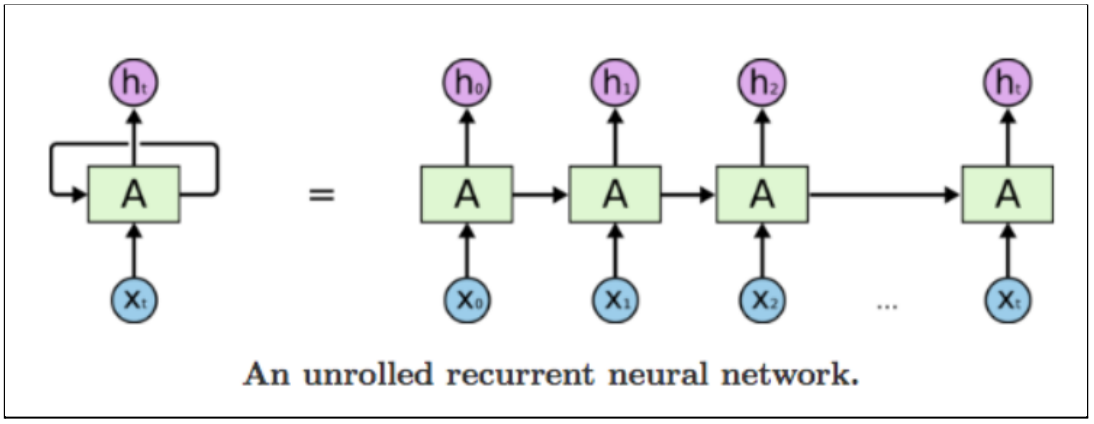

Figura 37: Red neuronal recurrente (*RNN* en inglés) [3].

A.3.2 LSTM

Las redes neuronales de memoria a largo y corto plazo (Figura 38) son una evolución de las recurrentes (RNN). Permiten realizar clasificaciones, procesamientos y predicciones de series de tiempo en las que no se sabe cuánto dura el retraso. Una celda de esta red neuronal tiene tres puertas de entrada “a través de las que se procesa la información”: “la puerta de entrada”, la de olvido y la de salida; también tiene “los datos de la celda oculta y la salida de la celda” [3].

Este tipo de red neuronal no ha llegado a convertirse en la solución definitiva para “generar oraciones coherentes” porque tiene una capacidad de memoria limitada y en consecuencia solamente alcanza a mantener almacenados del orden de “unos pocos cientos de palabras”. Además, entrenarlas requiere un alto coste computacional y no se pueden paralelizar fácilmente [3].

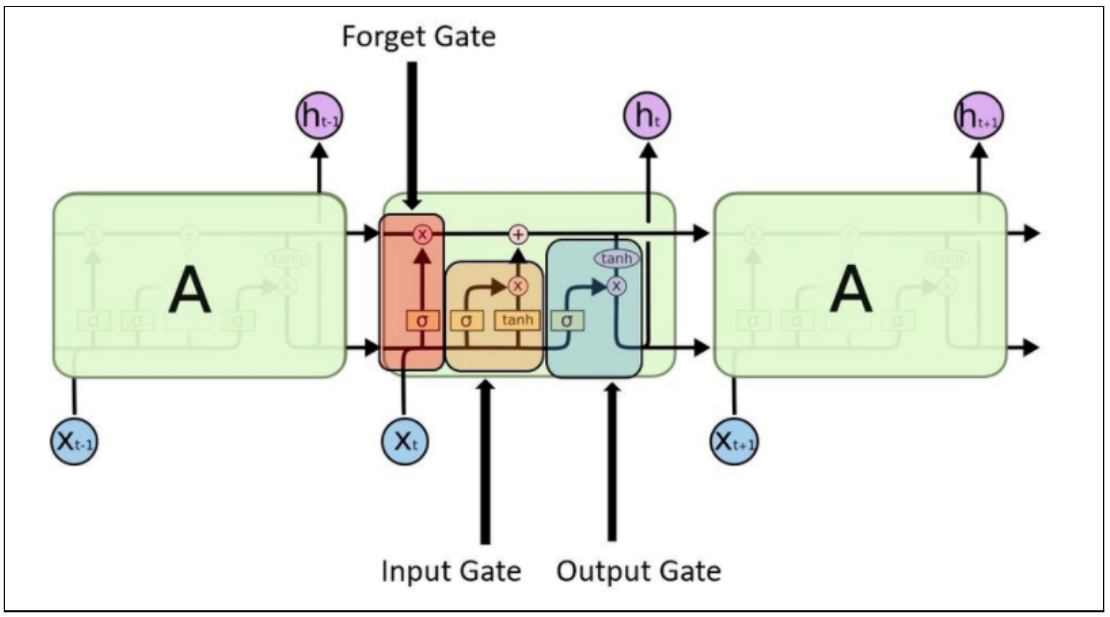

Figura 38: Red neuronal de memoria a largo y corto plazo (*LSTM* en inglés) [3].

A.3.3 Mecanismo de atención

En un documento de “Google Brain y Google Research”, titulado “La atención es todo lo que necesita”, se presentaron técnicas que se aprovechan del paralelismo sobre hardware TPU de Google. Estas presentan mayor rapidez para realizar cálculos en tiempo en comparación con las redes neuronales RNN [3].

El modelo “Transformador” (o “*Transformer*” en inglés) “se basa en un mecanismo de atención”. Emplea un codificador que permite almacenar “información”, relacionada con “una palabra concreta”, sobre “su vector de palabras” [3].

Las salidas las concreta una red que emplea un mecanismo de atención y tiene en cuenta tanto ”palabras contextuales relevantes tanto en la secuencia de entrada como en las salidas predichas hasta ese punto” [3].

A.3.4 BERT

BERT significa “Representaciones de codificador bidireccional de Transformers” (alternativamente llamado “*Bidirectional Encoder Representations from Transformers*” en inglés) se trata de otro modelo que Google presentó en otro artículo de investigación. Es un modelo flexible, capaz de resolver distintos problemas “ajustando el modelo previamente entrenado”. Sus resultados son muy satisfactorios, llegando a superar anteriores marcas en el contexto del PLN como por ejemplo responder 100.000 preguntas de comprensión de lectura empleando como fuente artículos de Wikipedia [3].

Este modelo “utiliza” una “arquitectura Transformer” con la que crea varias “representaciones de la secuencia de entrada [...]”. Cada representación se centra en una característica distinta de la entrada. Originalmente se presentaron 2 modelos [3]:

- BERT base: que consta de “12 capas (bloques transformadores), 12 cabezas de atención y 110 millones de parámetros“ [3].
- BERT grande (o “large” en inglés): con “24 capas, 16 cabezas de atención y 340 millones de parámetros”. Su entrenamiento se basa en un “modelo lingüístico bidireccional profundo (prediciendo una palabra enmascarada en una frase, dadas todas las demás palabras)” [3].

Han surgido adaptaciones de esta arquitectura que funcionan mejor para determinadas aplicaciones, como por ejemplo: ALBERT, RoBerta, TinyBERT, DistilBERT y SpanBERT [3].

A.3.5 Transformer-XL

“En enero de 2019,” se publica un modelo que mejora el de BERT y su arquitectura original “con una arquitectura” denominada “Transformer-XL”. Aparece “en el documento” "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" de “Google Brain y la Universidad Carnegie Mellon” [3].

Esencialmente resuelve dos problemas:

1. Permite reutilizar “los estados ocultos obtenidos en el modelo en segmentos anteriores para servir como una memoria para el segmento actual [...] para construir una conexión recurrente entre los segmentos”.
2. Basado en [54] la fragmentación del contexto se puede definir como la condición en la que un modelo no dispone de suficiente “información contextual [...] para predecir los primeros símbolos” porque se ha elegido deficientemente el contexto (por ejemplo “sin respetar una oración”). Precisamente este problema es resuelto gracias a “la información transmitida de segmentos anteriores” [3].

A.3.6 Transformadores compresivos

Por último, a finales “de 2019” aparecen los transformadores compresivos (o *“Compressive Transformers”* [3] en inglés), también de la mano de Google, que se basan en “memoria de largo alcance” [3].

El cerebro humano guarda parcialmente experiencias del pasado de forma comprimida. “El transformador compresivo [...] de manera similar,” comprime “activaciones ocultas pasadas”, de forma que combinando el uso de la memoria de “largo” y la de “corto” “alcance” funciona mejor que solo empleando la memoria de corto alcance [3].

Desde “2018” han aparecido algunos modelos que han constatado un gran avance en el contexto del PLN y se sigue avanzando. En concreto con “los modelos [...] no supervisados [...] BERT y Transformer-XL” se ha mejorado en tareas como: “la inferencia del lenguaje natural, [...] análisis de sentimientos y respuestas a preguntas” [3].

Anexo 4: Información detallada sobre la implementación

En este anexo se proporciona información más detallada sobre la implementación llevada a cabo de los workflows que sirven de apoyo (para extracción de los textos de Twitter y campos conceptuales, etc.) para los distintos experimentos de minería de texto posteriores.

Cabe resaltar que se pueden ejecutar los distintos workflows que usan algún nodo *Python Script* configurando "Flatbuffers Column Serialization" desde KNIME en "File > Preferences > Python" como la biblioteca de serialización ("Serialization Library", en inglés) a emplear. La excepción a esta biblioteca se da en el workflow de extracción de textos descrito en el apartado 3.3.1, que requiere elegir la biblioteca de serialización “Apache Arrow” para que funcione correctamente.

A.4.1 Extracción de textos

Primeramente, se describe el flujo de trabajo principal asociado al apartado “3.3.1 Extracción de textos” de esta memoria (la Figura 6 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

- El nodo #43 contiene una tabla de datos con las categorías a emplear para clasificar los textos.
- Los datos del nodo #43 pasan al #42 que añade a cada dato (nombre de la categoría) un String con varios operadores de Twitter que se emplearán para filtrar los datos de Twitter. Los operadores están separados por espacios en blanco y consisten en fijar el lenguaje al español para solo recuperar tweets en ese idioma (operador lang:es) además de aplicar filtros como:
  - -filter:retweets: filtra tweets con retweets
  - -filter:mentions: filtra tweets con menciones a usuarios
  - -filter:media: filtra tweets con imágenes o video
  - -filter:images: filtra tweets con imágenes
  - -filter:twimg: filtra tweets con “un enlace pic.twitter.com que representa una o más fotos” [43]
  - -filter:links: filtra tweets con links
  - filter:safe: evita tweets “marcados como potencialmente sensibles” [43]
- El nodo #37 inicia un bucle que realiza una iteración sobre cada fila de datos del nodo anterior (1 vez por categoría de la que se van a buscar tweets).
- Los nodos #24, #26 y #1 consiguen cargar las credenciales de Twitter necesarias para usar su API.
- El nodo #36 busca el contenido del texto de los tweets y se ejecuta una vez por categoría. Como consulta emplea en cada iteración la que recibe del nodo #37.
- El nodo #1107 es un nodo de fin de bucle que además acumula los tweets obtenidos como resultado de las distintas iteraciones quedando todos ellos en 1 columna (también hay otra columna que indica el número de iteración en el que se ha extraído cada tweet).
- Al acabar el bucle, el metanodo #1134 realiza un preprocesamiento de los datos eliminando por ejemplo tweets duplicados y algunos emojis y junta los tweets de la misma categoría en una misma columna, etc.
- El nodo #1133 recibe del #43 los nombres de las categorías y los pasa como valores de variable de flujo..
- El nodo #1132 renombra las columnas según el nombre de la categoría que le corresponde a cada una conforme a los datos que le pasa el nodo #1133.
- El metanodo #1575 realiza operaciones sobre los datos dejando un total de 120 tweets (10 por categoría), todos en una columna y crea otras 2 columnas: una con la categoría a la que pertenece cada tweet y, otra con la fuente de donde se han obtenido los datos, a saber Twitter.

- Finalmente, el nodo #1127 intenta escribir los datos en un fichero CSV (lo escribe si aún no existe un fichero con el nombre especificado en la ruta especificada, si no da un error).

A continuación, se presentan sub workflows contenidos en metanodos del flujo de trabajo principal asociado al apartado “3.3.1 Extracción de textos” de esta memoria.

Metanodo #1134

En la Figura 39 se muestra el contenido del metanodo #1134.

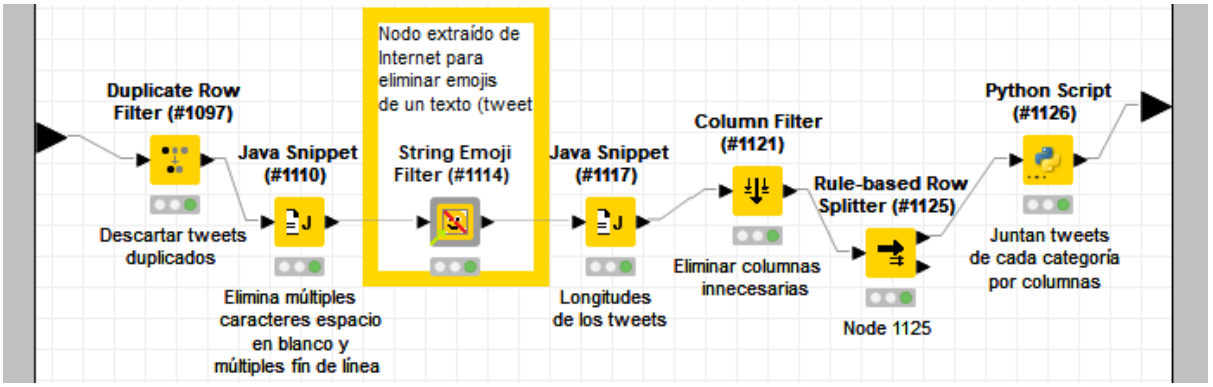

Figura 39: Contenido del metanodo #1134, integrado en el workflow de extracción de textos.

- El nodo #1097 descarta tweets duplicados de los datos que obtiene del nodo anterior (nodo #1107).
- El nodo #1110 elimina múltiples caracteres espacio en blanco y múltiples fín de línea dejando solo uno de cada tipo.
- El siguiente nodo, el #1114, filtra emojis de los tweets (el filtro no es perfecto pero sí que identifica muchos de ellos).
- El nodo #1117 calcula las longitudes de los tweets filtrados.
- El nodo #1121 elimina columnas de datos que no se van a usar y el nodo #1125, filtra filas que no tengan ningún tweet.
- Seguidamente el nodo #1126 junta los tweets de cada categoría por columnas dejando el primero de cada categoría por la primera fila y en subsiguientes filas el resto.

Metanodo #1575

En la Figura 40 se muestra el contenido del metanodo #1575.

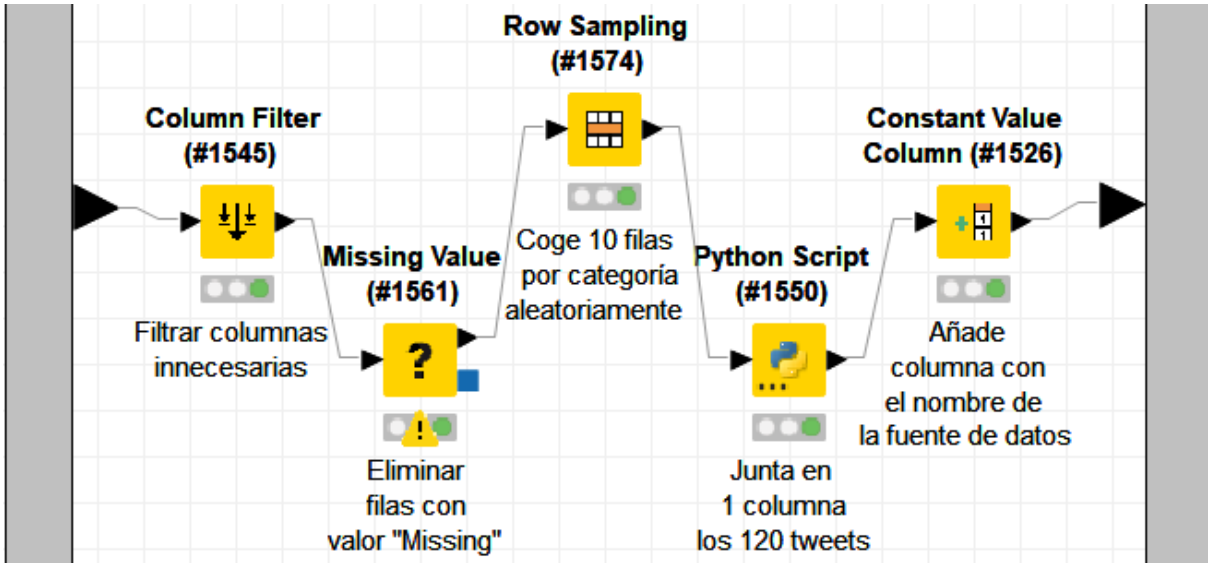

Figura 40: Contenido del metanodo #1175, integrado en el workflow de extracción de textos.

Los siguientes nodos hasta el nodo #1526 realizan operaciones sobre los datos recibidos, dejando un total de 120 tweets (10 de cada categoría) y añadiendo 2 columnas: una con la categoría a la que pertenece cada tweet y, otra indicando la fuente de los datos, que es Twitter.

- El nodo #1545 elimina columnas innecesarias.
- El nodo #1561 elimina filas con valor “Missing” de KNIME.
- Con el nodo #1574 se cogen 10 filas o tweets por categoría de forma aleatoria.
- El nodo #1550 junta en una columna los 120 tweets.
- El nodo #1526 añade una columna con el nombre de la fuente de datos, que es Twitter en este caso.

A.4.2 Extracción de campos conceptuales

Primeramente, se describe el flujo de trabajo principal asociado al apartado “3.3.2 Extracción de campos conceptuales” de esta memoria (la Figura 7 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

A continuación se describe el flujo de trabajo o workflow realizado:

- El nodo Table Creator #116 almacena los nombres de las categorías (palabras clave en general) que se emplearán como parte de la entrada del web scraping.
- El nodo #81 empleando el lenguaje de programación R junta en cada fila de su salida de datos los caracteres necesarios para borrar (representan la tecla de retroceso) la palabra por defecto que aparece en la página de entrada de Zirano empleada para el web scraping junto con el nombre de una categoría.
- Con los nodos #4 y #3 se crea un Web Driver para poder empezar el web scraping en [32] empleando Chrome como navegador.
- El nodo #106 inicia un bucle de 12 iteraciones (una por categoría).
- El metanodo #148 sincroniza la rama superior y la inferior y obtiene uno de los datos construidos (teclas borrado y nombre de categoría) en el nodo #81 en cada iteración.

- El metanodo #149 navega a la página web de Zirano y simula la introducción de la palabra a buscar y el tecleo de un “Enter” para realizar la búsqueda de la palabra introducida.
- Con el metanodo #150 se consigue navegar a la ventana popup con el árbol de Zirano de la categoría (palabra clave) buscada y se obtiene el código fuente HTML de esta página web.
- El metanodo #151 permite extraer ciertas etiquetas "width" de la página del árbol de Zirano buscada. Estas etiquetas son clave para identificar el nivel de un nodo dentro del árbol de Zirano.
- Con el metanodo #152 se extrae el contenido de las ideas del árbol de Zirano de la página buscada.
- El metanodo #153 junta la información de las ideas extraídas del árbol de Zirano y su altura asociada traduciendo las etiquetas "width" a datos (números enteros -de tipo Double- que representan el nivel de altura de cierta idea en el árbol) que identifiquen mejor la altura de cada idea del árbol de Zirano. Además, cambia el nombre de las columnas.
- El nodo #114 acumula un par de columnas para cada una de las 12 iteraciones, esto es, para cada categoría, con los datos resultantes.
- Con el nodo #118 se cambian el nombre de algunas columnas y, finalmente con el nodo #115 se escriben los datos obtenidos en un CSV (si no existe ya un fichero con el nombre especificado en la ruta correspondiente). Los datos obtenidos son cada una de las palabras del árbol Zirano de cada categoría y su nivel de altura correspondiente en su árbol de Zirano representado con números enteros (aunque la columna es de tipo Double).

A continuación, se presentan sub workflows contenidos en metanodos del flujo de trabajo principal asociado al apartado “3.3.2 Extracción de campos conceptuales” de esta memoria.

Metanodo #148

En la Figura 41 se muestra el contenido del metanodo #148.

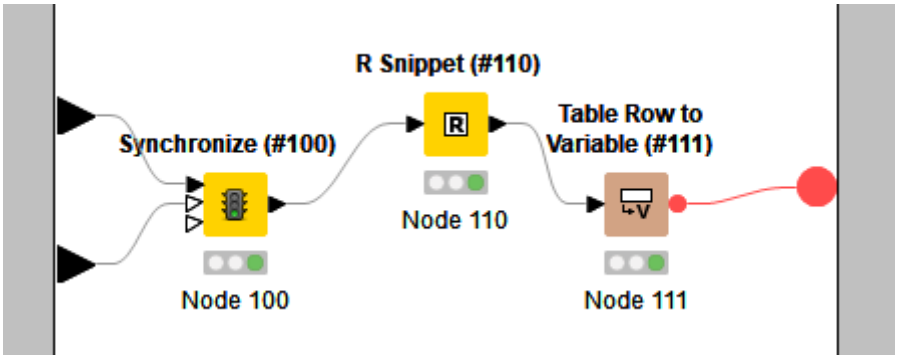

Figura 41: Contenido del metanodo #148, integrado en el workflow de extracción de campos conceptuales.

- El nodo #100 sirve para sincronizar las ramas superior e inferior del workflow principal asociado a este.
- El nodo #110 obtiene los caracteres de borrado junto con el nombre de una categoría construidos en un nodo previo del workflow principal.
- Y el nodo #111 pasa el dato que recibe a una variable de flujo de KNIME.

Metanodo #149

En la Figura 42 se muestra el contenido del metanodo #149.

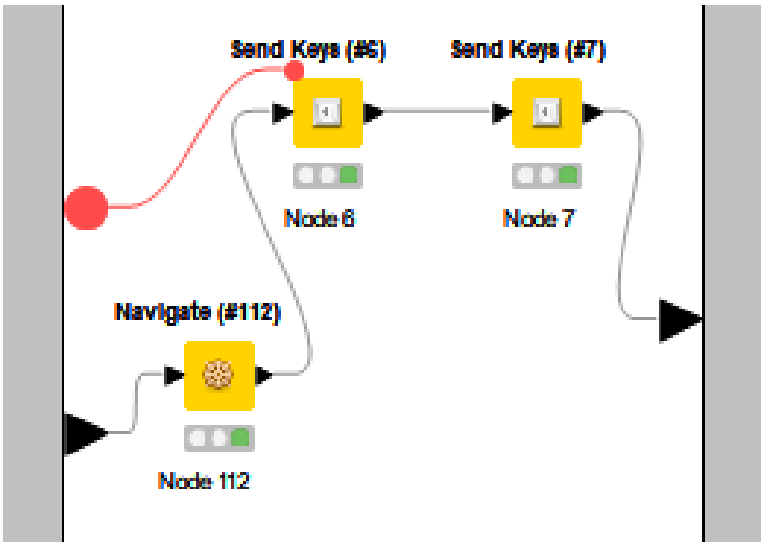

Figura 42: Contenido del metanodo #149, integrado en el workflow de extracción de campos conceptuales.

- Con el nodo #112 se simula la navegación a la página principal de Zirano.
- Es el nodo #6 el que efectivamente simula la presión del conjunto de teclas que recibe en cada iteración sobre el buscador de la página principal de Zirano.
- El nodo #7 simula un Enter con el que se consigue buscar la categoría (o palabra clave) introducida con el nodo anterior.

Metanodo #150

En la Figura 43 se muestra el contenido del metanodo #150.

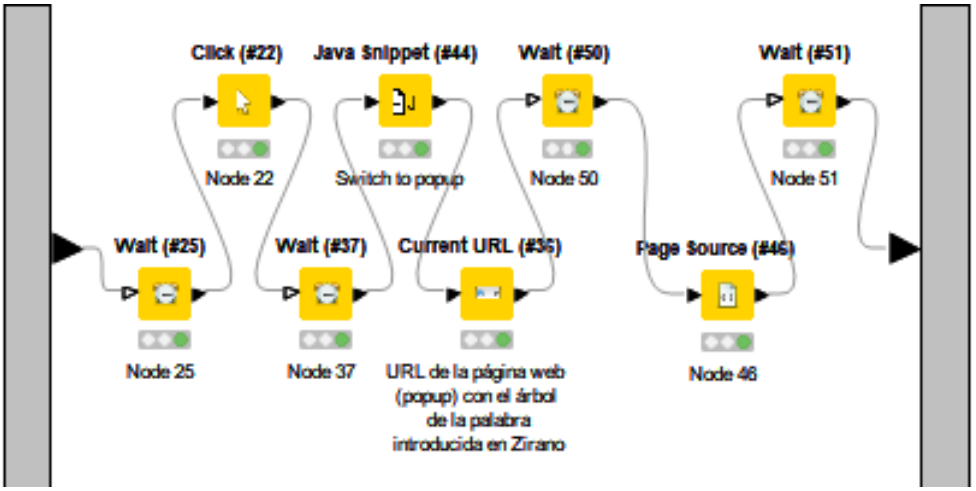

Figura 43: Contenido del metanodo #150, integrado en el workflow de extracción de campos conceptuales.

- Los nodos #25 y #37 simulan esperas antes y después de simular un click con el nodo #22 en el botón de "ARBOL" de la página de Zirano para llegar a abrir la página correspondiente al árbol de Zirano de la palabra introducida.
- Con el siguiente nodo, el #44, se navega a la página del árbol de Zirano (que aparece como ventana popup) y, con el nodo #36 se consigue la URL de dicha página.

- Con los nodos #50, #46 y #51 (ambos incluidos) se espera un tiempo antes y después de extraer el contenido de la página web del árbol de Zirano.

Metanodo #151

En la Figura 44 se muestra el contenido del metanodo #151.

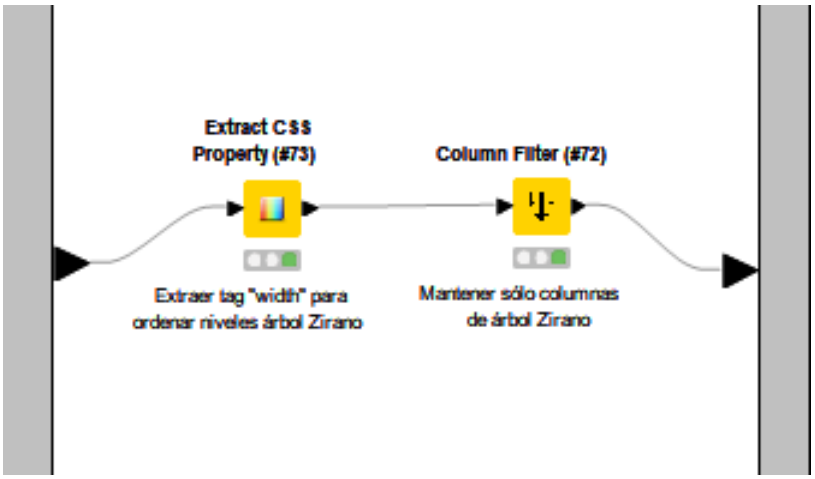

Figura 44: Contenido del metanodo #151, integrado en el workflow de extracción de campos conceptuales.

- Con el nodo #73 se consigue dejar las etiquetas “width” en una columna. Estas etiquetas se seleccionan con “CSS selector”, con la consulta: “body > form > table > tbody > tr > td > table > tbody > tr:nth-child(3) > td > table > tbody > tr:nth-child(3) > td:nth-child(2) > table > tbody > tr > td:nth-child(1)” y cogiendo el valor CSS “width”. Estos valores extraídos se pueden emplear como información que indica el nivel de cada nodo del árbol de Zirano.
- El nodo #72 filtra las columnas menos la que contiene las etiquetas obtenidas.

Metanodo #152

En la Figura 45 se muestra el contenido del metanodo #152.

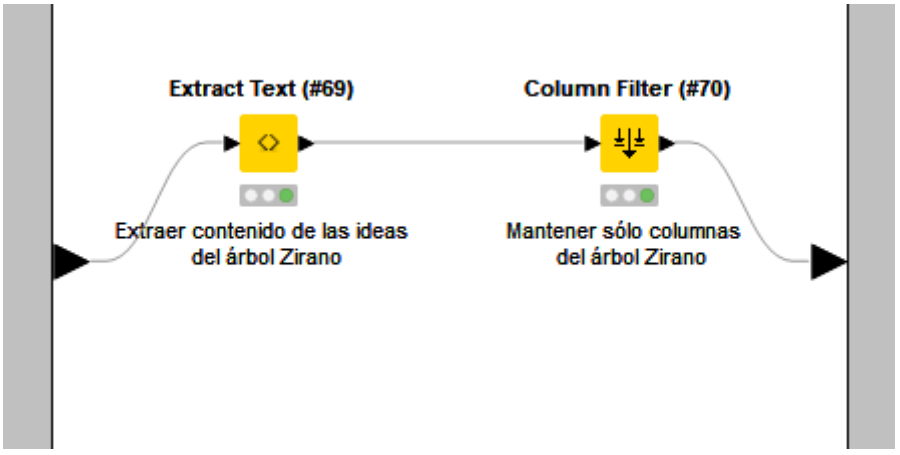

Figura 45: Contenido del metanodo #152, integrado en el workflow de extracción de campos conceptuales.

- El nodo #69 consigue las ideas de los nodos del árbol de Zirano. Las ideas se seleccionan con “CSS selector”, con la consulta: “body > form > table > tbody > tr > td > table > tbody > tr:nth-child(3) > td > table > tbody > tr:nth-child(3) > td:nth-child(2) > table > tbody > tr > td:nth-child(3) > a” y cogiendo la propiedad “textContent”.
- El nodo #70 filtra las columnas menos la que contiene las ideas obtenidas.

Metanodo #153

La Figura 46 muestra el contenido del metanodo #153.

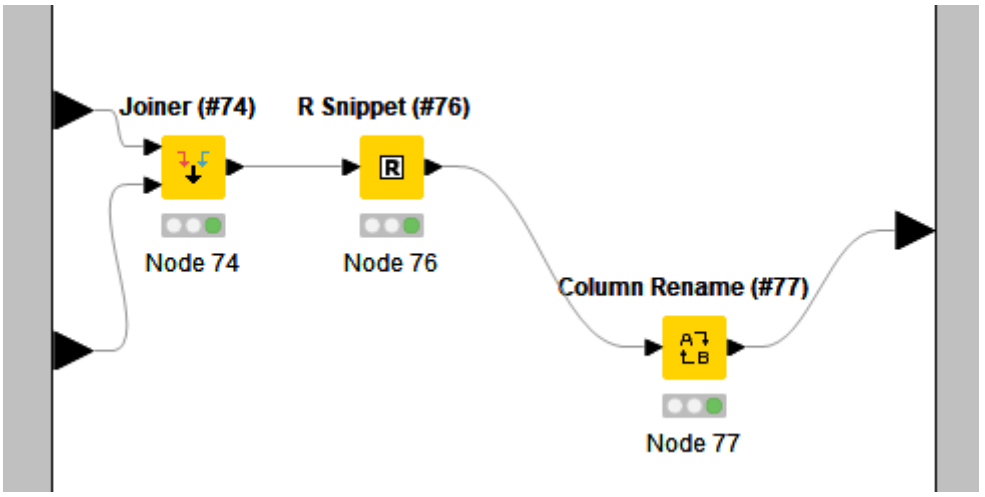

Figura 46: Contenido del metanodo #153, integrado en el workflow de extracción de campos conceptuales.

- El nodo #74 forma una tabla con una columna con las ideas del árbol de Zirano y otra con las etiquetas width obtenidas.
- El nodo #76 traduce la información de las etiquetas “width” en el nivel de altura (número entero de tipo Double) que le corresponde a cada idea del árbol de Zirano.
- El nodo #77 modifica el nombre de las columnas.

A.4.3 Creación y filtrado de árboles

Primeramente, se describe el flujo de trabajo principal asociado al apartado “3.3.3 Creación y filtrado de árboles” de esta memoria (la Figura 8 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

A continuación, se describe el flujo de trabajo realizado:

- El nodo #1406 lee los datos con las ideas de Zirano y su nivel de altura que se han escrito en el CSV del final del workflow principal asociado al apartado 3.3.2 de la memoria.
- El grupo de nodos formado por #1119, #1118 y #1120 reciben los datos del árbol de Zirano e inician un bucle en el que se lleva a cabo una iteración por cada categoría del árbol de Zirano.
- Con el metanodo #1407 se preprocesan y descartan parte de los datos de entrada (se eliminan caracteres invisibles <0xa0> detectados en alguna ocasión y valores que no se necesitan como los valores missing de KNIME similares a los típicos null en lenguajes de programación, etc.).
- El nodo #126 elimina todos los ítems de la tabla que tienen un símbolo '>' al principio.
- Con los nodos #1094 y #1095 se obtiene un listado con números enteros que referencian (indexan) a los padres de cada ítem del árbol (nodo #1094) y, a partir de ello y el resto de los datos del árbol de Zirano (ideas y nivel de altura de cada idea), se reconstruye el árbol en forma de objeto Tree con la biblioteca treelib de Python y se imprime por salida estándar dicho árbol con el nodo #1095.
- El nodo #1099 ejemplifica cómo guardar de forma persistente el objeto treelib construído en el nodo anterior.
- El nodo #1100 tienen como función principal filtrar los nodos del árbol conforme a las tres condiciones listadas al final de la introducción de este apartado 3.3.3.
- Después el nodo #1121 marca el fin del bucle y va recolectando los datos obtenidos de cada iteración.

- El nodo #116 pasa los nombres de las categorías al nodo #1144.
- Cuando acaba el bucle, el nodo #1144 actualiza los nombres de las columnas de datos que recibe del nodo #1121 con el que corresponde a cada categoría.
- Con el nodo #1405 principalmente se filtra de la tabla de ideas sobre las categorías lo que hay entre paréntesis (incluídos los propios símbolos de paréntesis) y los símbolos extra que cumplan esta expresión regular: [^a-zA-Z0-9 áéíóúÁÉÍÓÚñÑü].
- El metanodo #1459 prepara los datos de los árboles Zirano de las categorías para usarlos en los experimentos.
- El nodo #1145 intenta escribir los resultados en un CSV.

A continuación, se presentan sub workflows contenidos en metanodos del flujo de trabajo principal asociado al apartado “3.3.3 Creación y filtrado de árboles” de esta memoria.

Metanodo #1407

En la Figura 47 se muestra el contenido del metanodo #1407.

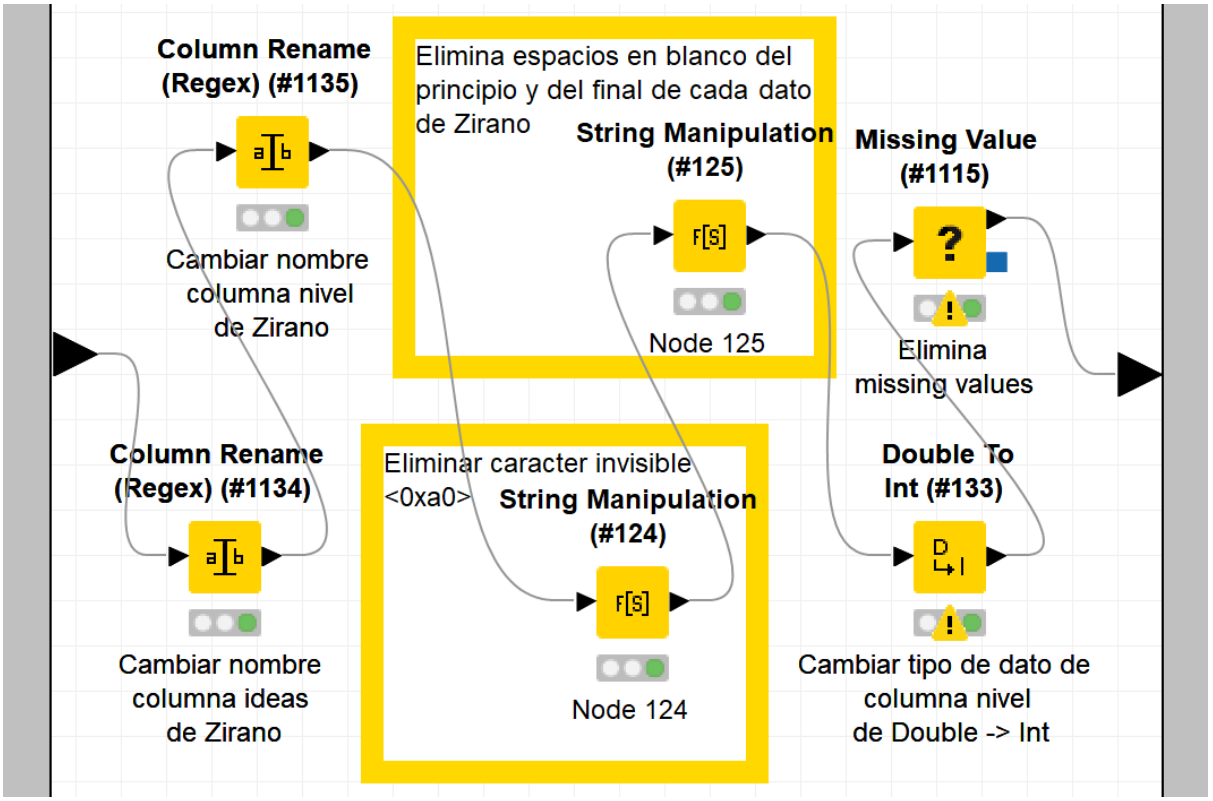

Figura 47: Contenido del metanodo #1407, integrado en el workflow de creación y filtrado de árboles.

- Con estos nodos (del #1134 al #1115) se preprocesan y descartan parte de los datos de entrada (eliminan caracteres invisibles <0xa0> que ha detectado en alguna ocasión, se eliminan valores que no se necesitan (valores missing de KNIME, etc.).
- El nodo #1134 cambia el nombre de la columna de ideas del árbol de Zirano.
- El nodo #1135 cambia el nombre de la columna con los niveles de altura de las ideas del árbol de Zirano.
- El nodo #124 elimina caracteres invisibles <0xa0> detectados en alguna ocasión.
- El nodo #125 elimina espacios en blanco del principio y del final de cada dato de Zirano.
- El nodo #133 cambia el tipo de dato de la columna de nivel de real a entero (Double a Int, en KNIME).
- Y el nodo #1115 elimina valores faltantes o “missing” en KNIME.

Metanodo #1459

En la Figura 48 se muestra el contenido del metanodo #1459.

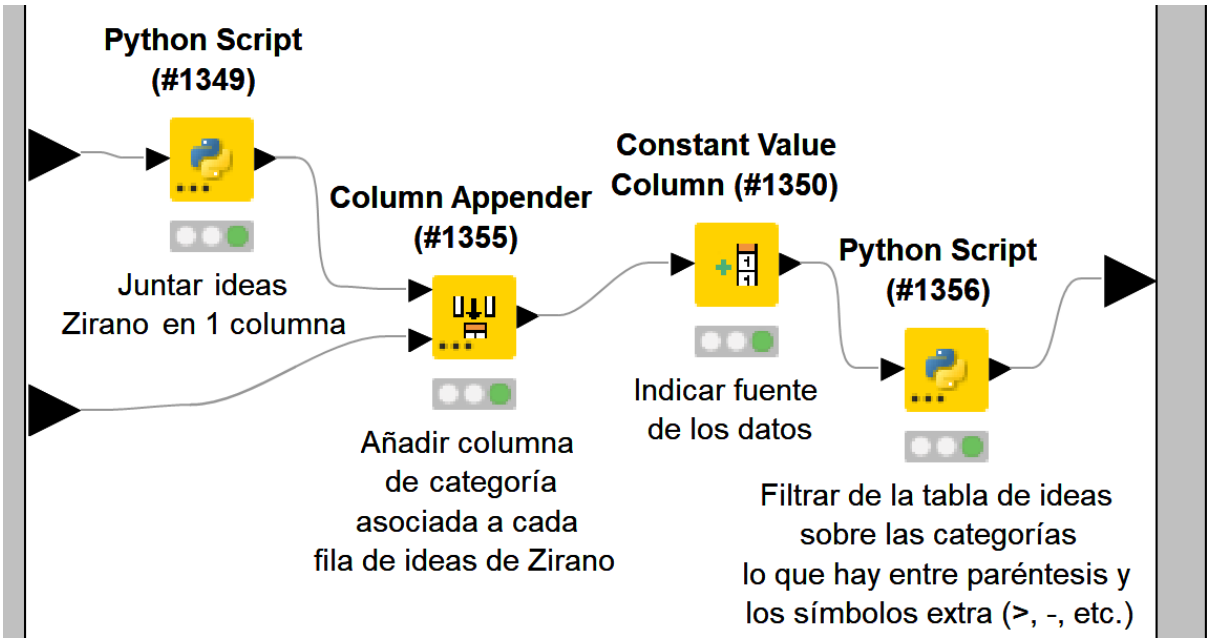

Figura 48: Contenido del metanodo #1459, integrado en el workflow de creación y filtrado de árboles.

- El nodo #1349 junta las ideas de Zirano en una columna.
- El nodo #1355 añade una columna con el nombre de la categoría a la que está asociada cada fila (o secuencia de ideas de Zirano).
- El nodo #1350 añade una columna con un valor constante que representa el nombre de la fuente de datos de la que se han obtenido los datos, que en este caso es Zirano.
- El nodo #1536 filtra de la tabla de ideas sobre las categorías lo que hay entre paréntesis y los símbolos extra que cumplan esta expresión regular: [^a-zA-Z0-9 áéíóúÁÉÍÓÚñÑü].

A.4.4 Almacenar datos en la BBDD

En la Figura 49 se muestra el contenido del flujo de trabajo que ejemplifica cómo almacenar los datos de los textos y su categoría asociada en una BBDD SQLite similar (se probó con éxito en una BBDD con el nombre de la tabla intermedia igual a “clasificado_como” en vez de “Clasificado_como”) a la creada conforme se especifica en el Anexo A.1.1.

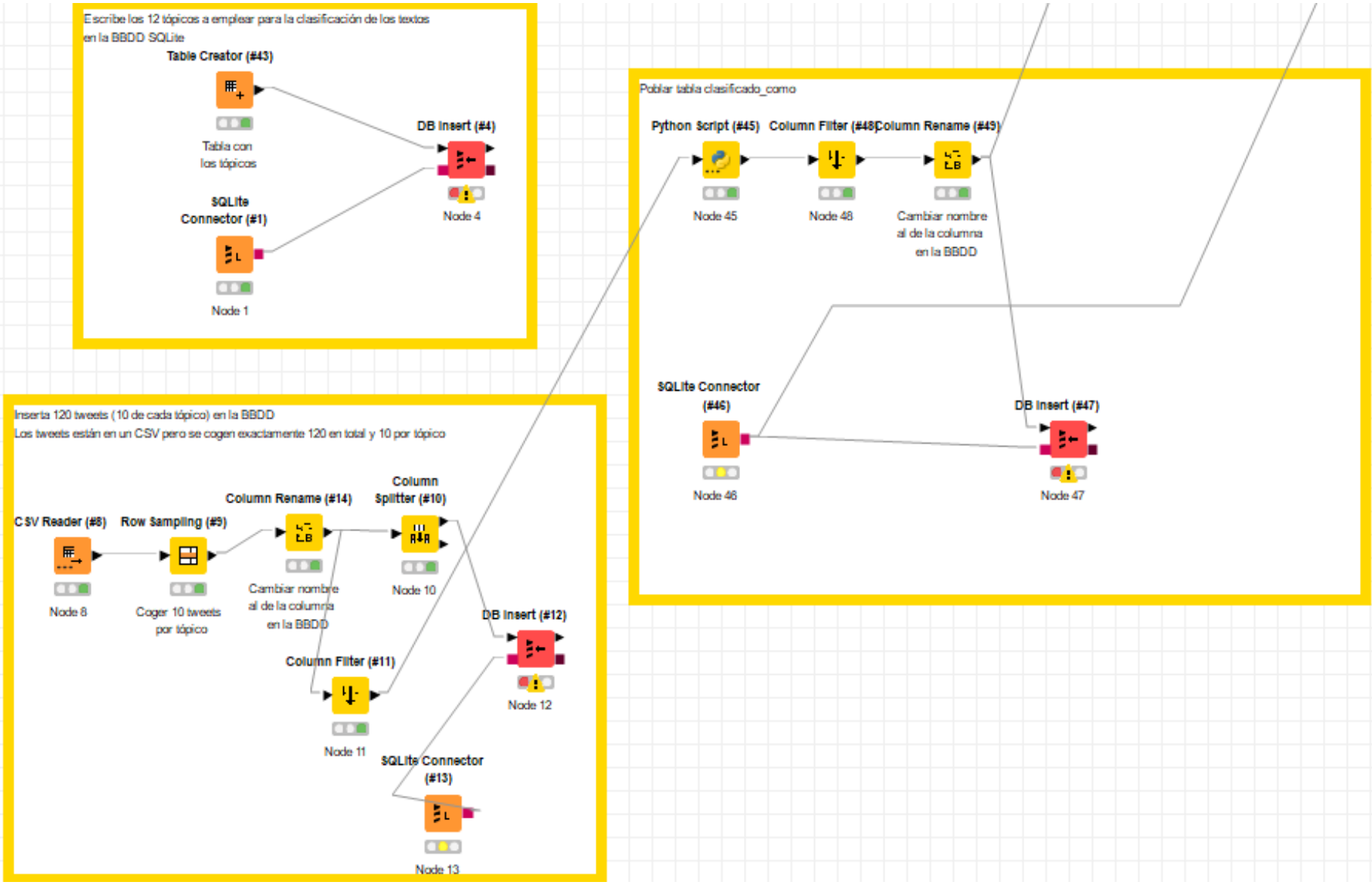

Figura 49: Flujo de trabajo para almacenar datos de los textos en la BBDD SQLite.

Descripción del flujo de trabajo

De modo general todos los nodos SQLite Connector se emplean para conectar con la BBDD y poder usar la conexión en otro nodo para realizar operaciones sobre la BBDD (en el caso de este workflow, operaciones de escritura):

Primeramente se va a describir lo que hacen los nodos del recuadro amarillo de arriba a la izquierda:

- El nodo #43 contiene la información de las 12 categorías que se quieren insertar en la BBDD.
- Con una conexión creada en el nodo #1, se pueden insertar en una base de datos SQLite los datos con el nodo #4 en la tabla que se le especifique.

A continuación, se describen los nodos del recuadro de más abajo para insertar los datos de los tweets en una BBDD SQLite:

- El nodo #8 lee los datos de los tweets de un CSV.
- Con el nodo #9 se cogen 10 tweets por categoría.
- El nodo #14 cambia el nombre de la columna de la que se insertarán los datos en la BBDD para que coincida con la del esquema diseñado.
- El nodo #10 filtra las columnas quedándose con la que se insertarán en la BBDD.
- El nodo #12 permite insertar los datos en la BBDD con la conexión del nodo #13.
- Y, el nodo #11 filtra los datos dejando los que interesan para los nodos del recuadro que falta por comentar.

Por último, en el recuadro de la derecha:

- El nodo #45 genera un índice autoincremental para cada dato (tweet) recibido.
- Los nodos #48 y #49 permiten preparar los datos (nombre de columna adecuado, etc.) que se insertarán en la BBDD para relacionar un tweet con su categoría.
- Y, el nodo #47 permite escribir en la BBDD los datos.

Anexo 5: Información detallada sobre los experimentos

En este anexo se proporciona información más específica acerca de la implementación de los experimentos realizados.

Cabe mencionar que los metanodos de los experimentos 2, 3 y 4 no se detallan a continuación ya que tienen sus metanodos equivalentes o parecidos en el experimento 1.

A.5.1 Experimento 1

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.1 Experimento 1” de esta memoria (la Figura 9 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

- El nodo #1187 lee 120 tweets.
- El siguiente nodo, el #72, convierte a tipo Document los datos leídos en el nodo anterior.
- El metanodo #1389 del recuadro “Preprocessing” realiza tareas de preprocesamiento del texto (quita símbolos de puntuación, filtra palabras con menos de 3 caracteres, aplica stemming, etc.) [44], algunas de las cuales se han comentado en la fase de “Limpieza de datos y normalización” del apartado 3.2.4.
- El metanodo #95 transforma “los documentos [...] en una bolsa de palabras, que se vuelve a filtrar. Sólo los términos que aparecen al menos en el 1% de los documentos [...] se utilizarán como características y no se filtraran” [44]. También aplica TF obteniendo la frecuencia relativa de cada término de cada documento.
- Con el metanodo #1390, principalmente “los documentos se transforman en vectores de documentos” y se dividen los datos, de forma aleatoria, en 2 conjuntos: 70% de ellos para entrenamiento y 30% para test [44].
- El metanodo #1392 entrena un árbol de decisión con los tweets del conjunto de entrenamiento e intenta predecir con los tweets de test. Ídem con el metanodo #1391 pero empleando SVM como clasificador y predictor.
- Los nodos #1388 y #1393, según el clasificador empleado sea árbol de decisión o SVM respectivamente, presentan algunos resultados de la clasificación por ejemplo la matriz de confusión, el accuracy, la *precision* y el *recall* para cada categoría.

Seguidamente, se muestran la Tabla 3 y la Tabla 4 que muestran los resultados del Experimento 1 empleando árbol de decisión y SVM como clasificadores, respectivamente.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	5	0	31	0	1	1	1	1	1	?	?
Barco	2	0	34	0	1	1	1	1	1	?	?
Ordenador	4	0	32	0	1	1	1	1	1	?	?
Fútbol	5	0	31	0	1	1	1	1	1	?	?
Natación	2	0	34	0	1	1	1	1	1	?	?
Televisión	2	0	34	0	1	1	1	1	1	?	?
Tienda	3	0	33	0	1	1	1	1	1	?	?
Dinero	1	0	35	0	1	1	1	1	1	?	?
Préstamo	3	0	33	0	1	1	1	1	1	?	?
Automóvil	4	0	32	0	1	1	1	1	1	?	?
Viaje	2	0	34	0	1	1	1	1	1	?	?
Internet	3	0	33	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	1	1

Cuadro 3: Tabla con los resultados del Experimento 1 con árbol de decisión como clasificador.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	4	0	31	1	0.8	1	0.8	1	0.889	?	?
Barco	2	1	33	0	1	0.667	1	0.971	0.8	?	?
Ordenador	4	0	32	0	1	1	1	1	1	?	?
Fútbol	4	0	31	1	0.8	1	0.8	1	0.889	?	?
Natación	2	1	33	0	1	0.667	1	0.971	0.8	?	?
Televisión	2	0	34	0	1	1	1	1	1	?	?
Tienda	3	0	33	0	1	1	1	1	1	?	?
Dinero	1	0	35	0	1	1	1	1	1	?	?
Préstamo	3	0	33	0	1	1	1	1	1	?	?
Automóvil	4	0	32	0	1	1	1	1	1	?	?
Viaje	2	0	34	0	1	1	1	1	1	?	?
Internet	3	0	33	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	0.944	0.939

Cuadro 4: Tabla con los resultados del Experimento 1 con SVM como clasificador.

A continuación, se presentan sub workflows contenidos en metanodos del workflow asociado al Experimento 1 introducido en el apartado 4.1 de la memoria.

Metanodo #1389

En la Figura 50 se muestra el contenido del metanodo #1389.

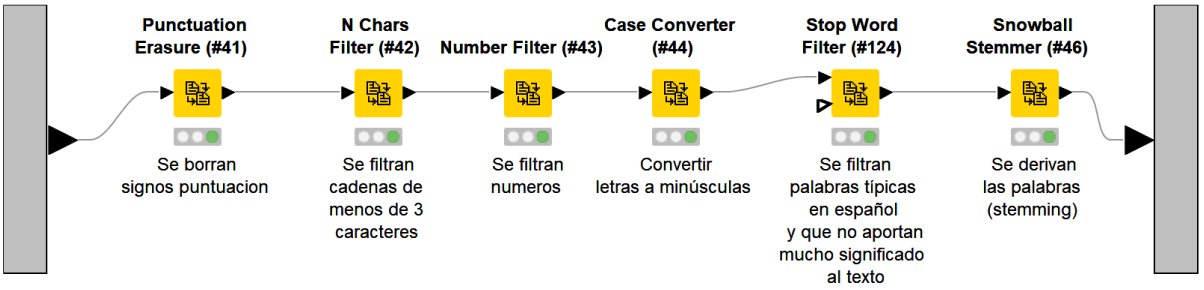

Figura 50: Contenido del metanodo #1389, integrado en el workflow asociado al Experimento 1.

- El nodo #41 borra los signos de puntuación de los documentos que recibe como entrada.
- El nodo #42 filtra los términos que tienen menos de 3 caracteres de longitud.
- El nodo #43 filtra términos que representan números (incluido signos separadores como ‘,’ o ‘.’ y los símbolos ‘+’ y ‘-’).
- Con el nodo #44 todos los términos de los documentos se transforman a letras minúsculas.
- El nodo #124 filtra los términos de los documentos de entrada presentes en cierta lista interna del nodo con palabras del lenguaje español.
- El nodo #46 lematiza los términos de los documentos de entrada dejándolos en su raíz.

Metanodo #95

En la Figura 51 se muestra el contenido del metanodo #95.

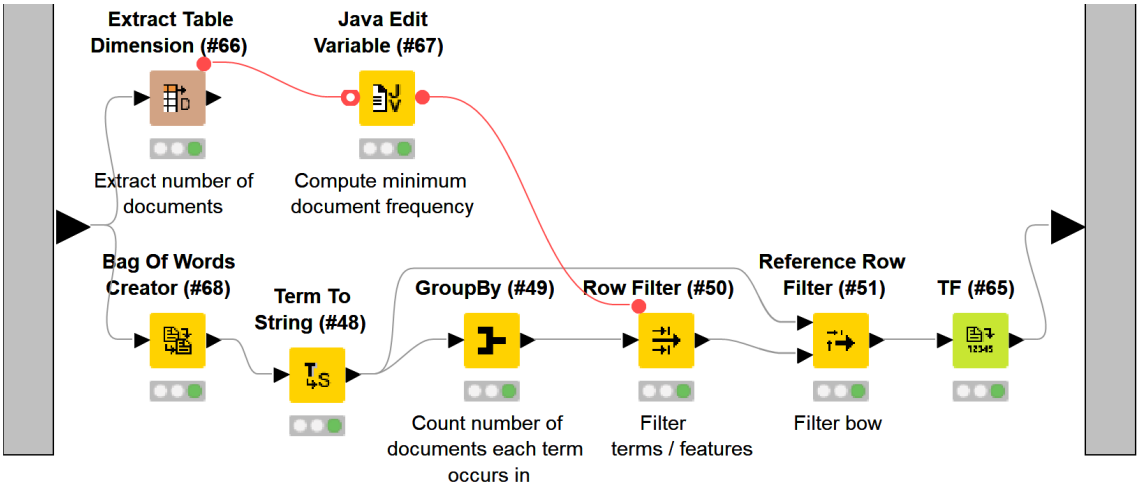

Figura 51: Contenido del metanodo #95, integrado en el workflow asociado al Experimento 1.

- El nodo #66 obtiene la dimensión de la tabla que recibe como entrada (número de filas y columnas que tiene).
- El nodo #67 calcula el mínimo número de documentos en los que tiene que aparecer un determinado término para que no sea filtrado. Se calcula como el número de filas de la tabla de entrada (120 en este caso) dividido para 100, que sale a 1 documento (si una palabra está en al menos 1 documento se tendrá en cuenta).

- El nodo #68 obtiene un modelo BoW con los distintos términos de la columna con documentos que se le ha configurado como entrada.
- El nodo #48 convierte los términos o Term en KNIME a tipo String de KNIME.
- Con el nodo #49 se cuenta el número de documentos en los que aparece cada término.
- El nodo #50 filtra los términos que no aparecen como mínimo en el número de documentos obtenido con el nodo #67 (para este workflow basta con que cada término aparezca en 1 documento para no que no se descarte).
- El nodo #51 incluye las filas de la primera tabla que recibe (la columna con los términos pasados a String del nodo #48) que están en la segunda tabla (la columna con los términos pasados a String del nodo #50).
- El nodo #65 calcula la frecuencia relativa de cada términos de acuerdo a cada documento.

Metanodo #1390

En la Figura 52 se muestra el contenido del metanodo #1390.

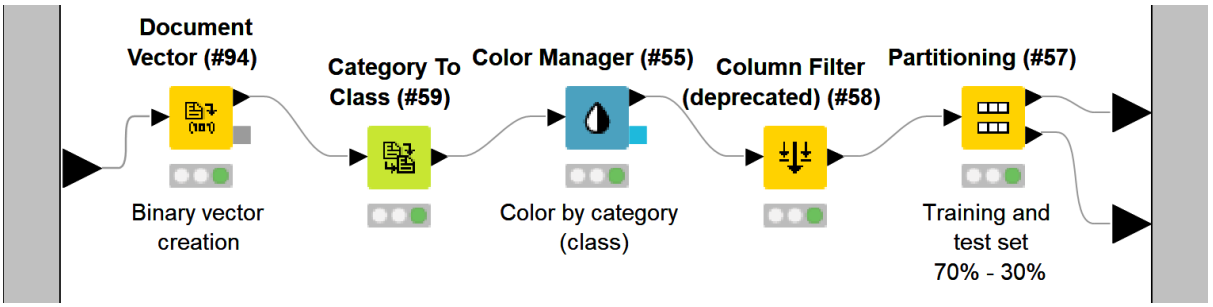

Figura 52: Contenido del metanodo #1390, integrado en el workflow asociado al Experimento 1.

- El nodo #94 transforma los documentos que recibe en vectores de documentos que son una representación numérica de documentos.
- El nodo #59 añade una columna que tiene como valor para cada fila el nombre de la categoría (tipo String) a la que pertenece el documento que aparece en dicha fila.
- El nodo #55 por su parte, asigna un color a una determinada categoría (este color se puede configurar y visualizar su asignación en la tabla de salida del nodo desde KNIME).
- El nodo #58 filtra la columna con los documentos.
- El nodo #57 divide el conjunto de datos en dos subconjuntos: el de entrenamiento con el 70% del total de los datos y el de test con el 30% de los datos.

Metanodo #1392

En la Figura 53 se muestra el contenido del metanodo #1392.

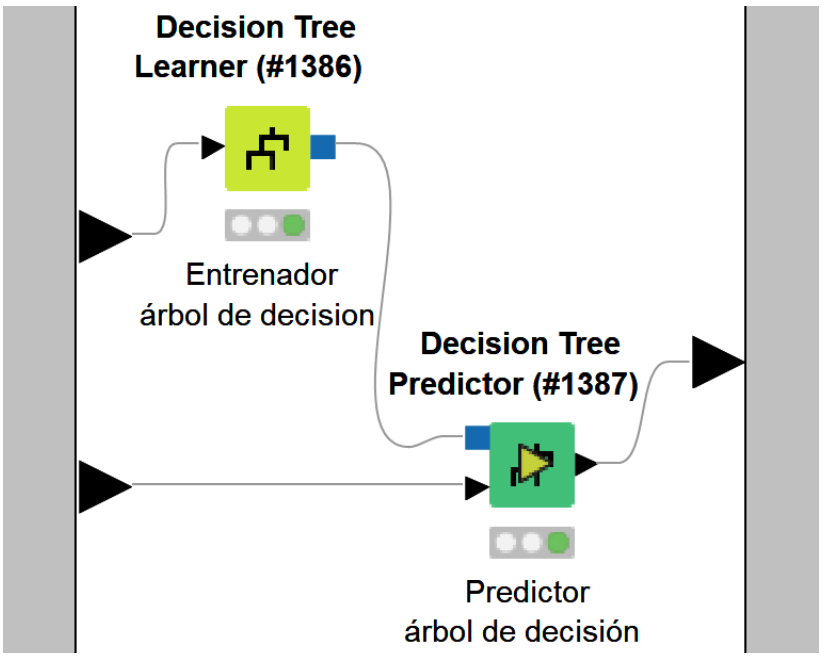

Figura 53: Contenido del metanodo #1391, integrado en el workflow asociado al Experimento 1.

- El nodo #1386 entrena y devuelve a su salida un modelo basado en árbol de decisión con los datos de entrada que recibe.
- El nodo #1387 predice a qué categoría pertenecen los datos de test que tiene como una de sus entradas, empleando el modelo entrenado en el nodo anterior para predecir.

Metanodo #1391

En la Figura 54 se muestra el contenido del metanodo #1391.

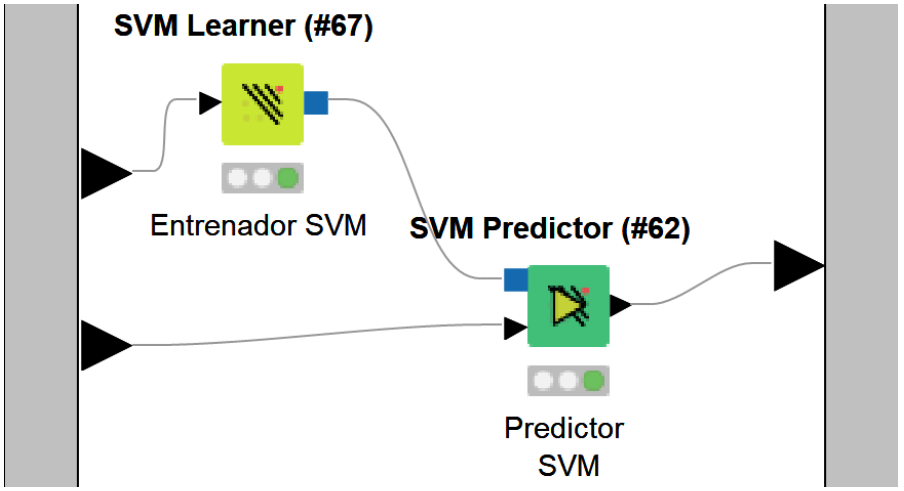

Figura 54: Contenido del metanodo #1391, integrado en el workflow asociado al Experimento 1.

- El nodo #67 entrena y devuelve a su salida un modelo basado en SVM con los datos de entrada que recibe.
- El nodo #62 predice a qué categoría pertenecen los datos de test que tiene como una de sus entradas, empleando el modelo entrenado en el nodo anterior para predecir.

A.5.2 Experimento 2

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.2 Experimento 2” de esta memoria (la Figura 10 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

Este segundo workflow es igual que el del primer experimento y difiere en los tres siguientes puntos:

- El nodo #1438 obtiene los datos de los tweets (120 en total) y el #1458 las 12 secuencias de palabras de los árboles de Zirano de cada categoría.
- Con el nodo #1436 se combinan ambas fuentes de datos en 1 columna (120 + 12 = 132 filas en total).
- El nodo #1457 convierte a tipo documento los datos almacenados en la columna del nodo anterior.
- El resto de nodos equivalen a los del Experimento 1, por lo que su descripción puede consultarse en el apartado 4.1 de esta memoria.

Seguidamente, se muestran la tabla 5 y la tabla 6 que muestran los resultados del Experimento 2 empleando árbol de decisión y SVM como clasificadores, respectivamente.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	2	0	38	0	1	1	1	1	1	?	?
Barco	1	0	39	0	1	1	1	1	1	?	?
Ordenador	4	0	36	0	1	1	1	1	1	?	?
Fútbol	2	0	38	0	1	1	1	1	1	?	?
Natación	3	0	37	0	1	1	1	1	1	?	?
Televisión	3	0	36	1	0.75	1	0.75	1	0.857	?	?
Tienda	4	0	36	0	1	1	1	1	1	?	?
Dinero	4	1	35	0	1	0.8	1	0.972	0.889	?	?
Préstamo	4	0	36	0	1	1	1	1	1	?	?
Automóvil	4	0	36	0	1	1	1	1	1	?	?
Viaje	6	0	34	0	1	1	1	1	1	?	?
Internet	2	0	38	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	0.975	0.972

Cuadro 5: Tabla con los resultados del Experimento 2 con árbol de decisión como clasificador.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	2	0	38	0	1	1	1	1	1	?	?
Barco	1	0	39	0	1	1	1	1	1	?	?
Ordenador	4	0	36	0	1	1	1	1	1	?	?
Fútbol	2	0	38	0	1	1	1	1	1	?	?
Natación	3	0	37	0	1	1	1	1	1	?	?
Televisión	4	0	36	0	1	1	1	1	1	?	?
Tienda	4	0	36	0	1	1	1	1	1	?	?
Dinero	4	0	36	0	1	1	1	1	1	?	?
Préstamo	4	0	36	0	1	1	1	1	1	?	?
Automóvil	4	1	35	0	1	0.8	1	0.972	0.889	?	?
Viaje	4	0	34	2	0.667	1	0.667	1	0.8	?	?
Internet	2	1	37	0	1	0.667	1	0.974	0.8	?	?
Overall	?	?	?	?	?	?	?	?	?	0.95	0.945

Cuadro 6: Tabla con los resultados del Experimento 2 con SVM como clasificador.

A.5.3 Experimento 3

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.3 Experimento 3” de esta memoria (la Figura 11 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

En primer lugar se va a comentar la rama superior del workflow, que tiene por objetivo obtener el modelo resultante de emplear las técnicas mencionadas.

- Para ello se entrena con las 12 secuencias de palabras de los árboles de Zirano de las categorías que son las que lee el primer nodo (#1458).
- Como en otros experimentos, las secuencias de caracteres o textos se convierten a tipo Documento, con el nodo #1457.
- Seguidamente, se realiza el preprocesamiento con el nodo #1590, que tal y como está documentado en dicho nodo, simplemente engloba un conjunto de nodos equivalentes a los que hay dentro del recuadro “Preprocessing” en los experimentos 1 y 2, por lo que si se quiere profundizar en conocer su descripción se remite a revisar la de los metanodos #1389 y #95 del workflow principal del experimento 1 (véase Anexo A.5.1).
- El siguiente nodo, el #1390, es equivalente a los nodos de los experimentos 1 y 2 que están dentro del recuadro “Transformation”, exceptuando el nodo “Partitioning”, por lo que si se quiere profundizar en conocer su descripción se remite a revisar la del metanodo #1390 del workflow principal del experimento 1 (apartado 4.1 de la memoria) pero omitiendo la parte de la división de los datos (nodo Partitioning) que no se incluye en este metanodo.
- Finalmente, se entrenan los modelos (que pasarán a la entrada del predictor correspondiente) de árbol de decisión en el nodo #1541 y el de SVM en el nodo #1571.

En segundo lugar, se explica la rama situada en la parte superior del workflow:

- El nodo #1438 se encarga de leer los datos de un conjunto de tweets (120 tweets en concreto con algunas columnas adicionales como las categorías de cada uno).
- El siguiente nodo, el #1436 de tipo Python Script, junta en una columna las dos fuentes de datos del workflow (los 12 datos con ideas de árboles de Zirano de las categorías y los 120 tweets).
- El nodo #1557 convierte los datos juntados a tipo Document.
- Posteriormente, se realiza el preprocesamiento de los datos en el nodo #1591 de la misma manera que con el #1590 de la primera rama.
- Después, el nodo #1592 se encarga de la transformación de los datos para que puedan emplearse con los nodos predictores, de manera equivalente al nodo #1390 comentado en la primera rama.
- Finalmente se validan los dos modelos obtenidos en la primera rama: con SVM en el nodo #1562 y con árbol de decisión en el nodo #1565 intentando predecir la categoría de los 120 tweets y las 12 secuencias de palabras de los árboles de Zirano y obteniendo resultados con los nodos Scorer respectivos de cada clasificador (nodo #1588 con SVM y #1589 con árbol de decisión).

Seguidamente, se muestran la tabla 7 y la tabla 8 que muestran los resultados del Experimento 3 empleando árbol de decisión y SVM como clasificadores, respectivamente.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	11	121	0	0	1	0.083	1	0	0.154	?	?
Barco	0	0	121	11	0	?	0	1	?	?	?
Ordenador	0	0	121	11	0	?	0	1	?	?	?
Fútbol	0	0	121	11	0	?	0	1	?	?	?
Natación	0	0	121	11	0	?	0	1	?	?	?
Televisión	0	0	121	11	0	?	0	1	?	?	?
Tienda	0	0	121	11	0	?	0	1	?	?	?
Dinero	0	0	121	11	0	?	0	1	?	?	?
Préstamo	0	0	121	11	0	?	0	1	?	?	?
Automóvil	0	0	121	11	0	?	0	1	?	?	?
Viaje	0	0	121	11	0	?	0	1	?	?	?
Internet	0	0	121	11	0	?	0	1	?	?	?
Overall	?	?	?	?	?	?	?	?	?	0.083	0

Cuadro 7: Tabla con los resultados del Experimento 3 con árbol de decisión como clasificador.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	10	0	121	1	0.909	1	0.909	1	0.952	?	?
Barco	10	0	121	1	0.909	1	0.909	1	0.952	?	?
Ordenador	11	0	121	0	1	1	1	1	1	?	?
Fútbol	11	1	120	0	1	0.917	1	0.992	0.957	?	?
Natación	11	0	121	0	1	1	1	1	1	?	?
Televisión	10	0	121	1	0.909	1	0.909	1	0.952	?	?
Tienda	11	1	120	0	1	0.917	1	0.992	0.957	?	?
Dinero	11	1	120	0	1	0.917	1	0.992	0.957	?	?
Préstamo	11	0	121	0	1	1	1	1	1	?	?
Automóvil	6	0	121	5	0.545	1	0.545	1	0.706	?	?
Viaje	11	5	116	0	1	0.688	1	0.959	0.815	?	?
Internet	11	0	121	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	0.939	0.934

Cuadro 8: Tabla con los resultados del Experimento 3 con SVM como clasificador.

A.5.4 Experimento 4

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.4 Experimento 4” de esta memoria (la Figura 12 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

Primeramente se va a describir parte de la rama superior del flujo de trabajo (o *workflow,* del inglés):

- El nodo #1187 una de las columnas de datos que lee es la correspondiente a los tweets (120 en total, 10 por categoría) y pasa su salida al nodo #1365.
- Recibidos los datos, el nodo #1365 convierte los datos a tipo Document de KNIME para poder tratar con él con nodos de la extensión “Text Processing”.
- Los datos de salida del nodo anterior llegan al nodo #1586 que se encarga de preprocesar los datos de manera equivalente al metanodo #1389 situado dentro del recuadro “Preprocessing” del experimento 1.
- El nodo #1379 emplea los documentos preprocesados del nodo anterior para obtener un modelo BoW.
- El siguiente nodo, el #1388, transforma a tipo String los datos de tipo Term que son el listado de palabras (con tags si los hubiese) obtenidos del nodo anterior.

A continuación se describe parte de la rama inferior del workflow:

- Se comienzan leyendo con el nodo #1458 las secuencias de palabras de árboles de Zirano asociadas a las categorías seleccionadas (junto a algunos datos más como la categoría a la que pertenece cada secuencia, etc.).
- El siguiente nodo, el #1364, filtra las columnas de datos innecesarias: las categorías asociadas a cada secuencia de palabras mencionadas del nodo anterior y la relativa a la fuente de la que provienen los datos.
- Los datos de salida del anterior nodo comentado llegan al nodo #1366, un tipo de nodo que como ya se ha mencionado en otras ocasiones, transforma los datos a tipo Document.
- El siguiente nodo, el #1587, preprocesa los datos de forma equivalente al nodo #1586 de la anterior rama descrita y envía sus datos al nodo #1381.

Por último, se explica brevemente la función del resto de nodos de ambas ramas:

- El nodo #116 es una tabla con una columna que tiene como datos los nombres de las posibles categorías seleccionadas para el trabajo.
- La salida del nodo anterior llega al nodo #1381 junto con los datos de los últimos nodos mencionados en las anteriores ramas explicadas diferentes al #1381 y, este nodo principalmente calcula para cada categoría el número de coincidencias de cada palabra del BoW de los tweets preprocesados con cada palabra de la secuencia de palabras del árbol de Zirano de la categoría correspondiente (coincidencias no necesariamente exactas sino que también vale si la palabra del BoW es subcadena de la palabra asociada con cierta categoría).
- La salida de datos del nodo anterior llega al nodo #1391 y, si se dan las condiciones necesarias (que no exista ya el fichero de nombre especificado), los datos se escriben en un fichero CSV con el nombre especificado en la configuración del nodo.
- El nodo #1392 también recibe los datos del nodo #1381 y cuenta el número de coincidencias de las anteriormente mencionadas (matchings, en inglés) pero agrupandolas por tweets.
- Por su parte, el nodo #1393 emplea los datos obtenidos del nodo #1392 para obtener e imprimir por “Standard output” resultados como la matriz de confusión, entre otras métricas.
- Finalmente los datos del nodo #1393 llegan al nodo #1394 y si se dan las circunstancias (no existe un fichero con el nombre especificado en la configuración del nodo todavía) escribe los datos en un fichero CSV con el nombre especificado en la configuración del nodo.

Seguidamente, se presentan la tabla 9 y la figura 55 que muestran algunos resultados del Experimento 4.

MATRIZ DE CONFUSIÓN

```
[[ 7  0  0  0  1  0  0  0  0  0  1  1]
 [ 1  4  0  2  0  0  0  0  0  0  0  3]
 [ 0  0 10  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 10  0  0  0  0  0  0  0  0]
 [ 0  0  1  0  9  0  0  0  0  0  0  0]
 [ 0  0  1  3  0  3  0  0  2  0  1  0]
 [ 2  1  1  1  0  1  3  1  0  0  0  0]
 [ 0  1  0  0  0  0  0  7  2  0  0  0]
 [ 1  0  0  2  0  0  0  0  4  0  3  0]
 [ 0  0  2  2  0  0  0  2  1  3  0  0]
 [ 0  0  1  0  0  0  0  0  1  0  8  0]
 [ 0  0  0  0  0  0  0  0  0  0  1  9]]
```

Cuadro 9: Tabla con la matriz de confusión del Experimento 4.

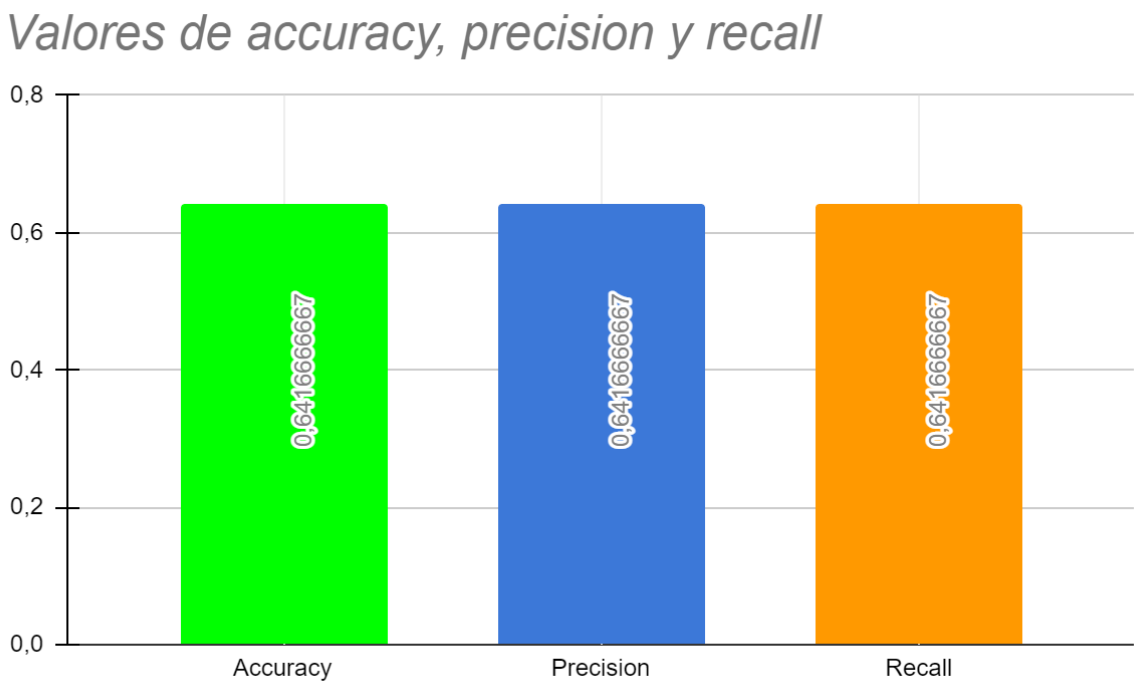

Figura 55: Gráfica con resultados del experimento 4 (valores de *accuracy*, *precision* y *recall* sobre la unidad).

A.5.5 Experimento 5

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.5 Experimento 5” de esta memoria (la Figura 13 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

En el workflow general del experimento 5 hay dos ramas principales que son casi simétricas en procesamiento. Se procede a explicar principalmente la de arriba ya que la de abajo se procesa de la misma manera a diferencia de que los datos de la rama superior son para entrenar (son las 12 secuencias de palabras de los árboles de Zirano de las categorías) y los de la inferior son datos de test (son los 120 tweets) y son distintos entre sí y, además se usan distintos clasificadores en cada rama.

- Con los dos primeros nodos, #1363, #1245, se leen los datos y se convierten a tipo documento de KNIME (Document); cada tarea en un nodo, respectivamente.
- El nodo #1608 preprocesa los datos aplicando técnicas como filtrar palabras con menos de 3 caracteres, stop words en español y derivación (stemming), entre otras.
- El nodo #1243 consigue un modelo de bolsa de palabras a partir de los documentos y deja solo aquellas palabras que aparecen en al menos el 1% de los documentos y aplica TF.
- El nodo #1610 prepara los conjuntos de entrenamiento y test para los clasificadores añadiendo a ambos las columnas correspondientes al otro conjunto de datos que no están presentes en el propio pero con valores 0 en cuanto a presencia de dicha palabra (columna) en sus documentos, es decir, iguala su dimensión de columnas (con valor 0 en las celdas de las filas de las columnas añadidas).
- El metanodo #1612 entrena un árbol de decisión con los 12 documentos con palabras de Zirano de las categorías e intenta predecir con los 120 tweets. Ídem con el nodo #1611 pero empleando SVM para entrenar.
- Finalmente el nodo #1613 presenta algunos resultados de la clasificación con árbol de decisión, por ejemplo la matriz de confusión, el accuracy y, la precision y el recall para cada categoría.

Seguidamente, se muestran la tabla 10 y la tabla 11 que muestran los resultados del Experimento 5 empleando árbol de decisión y SVM como clasificadores, respectivamente.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	1	2	108	9	0.1	0.333	0.1	0.982	0.154	?	?
Fútbol	0	1	109	10	0	0	0	0.991	NaN	?	?
Tienda	9	106	4	1	0.9	0.078	0.9	0.036	0.144	?	?
Dinero	0	1	109	10	0	0	0	0.991	NaN	?	?
Barco	0	0	110	10	0	?	0	1	?	?	?
Ordenador	0	0	110	10	0	?	0	1	?	?	?
Natación	0	0	110	10	0	?	0	1	?	?	?
Televisión	0	0	110	10	0	?	0	1	?	?	?
Préstamo	0	0	110	10	0	?	0	1	?	?	?
Automóvil	0	0	110	10	0	?	0	1	?	?	?
Viaje	0	0	110	10	0	?	0	1	?	?	?
Internet	0	0	110	10	0	?	0	1	?	?	?
Overall	?	?	?	?	?	?	?	?	?	0.083	-0

Cuadro 10: Tabla con los resultados del Experimento 5 con árbol de decisión como clasificador.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	8	1	109	2	0.8	0.889	0.8	0.991	0.842	?	?
Barco	10	1	109	0	1	0.909	1	0.991	0.952	?	?
Ordenador	9	1	109	1	0.9	0.9	0.9	0.991	0.9	?	?
Fútbol	10	1	109	0	1	0.909	1	0.991	0.952	?	?
Natación	9	0	110	1	0.9	1	0.9	1	0.947	?	?
Televisión	10	2	108	0	1	0.833	1	0.982	0.909	?	?
Tienda	9	2	108	1	0.9	0.818	0.9	0.982	0.857	?	?
Dinero	6	0	110	4	0.6	1	0.6	1	0.75	?	?
Préstamo	9	4	106	1	0.9	0.692	0.9	0.964	0.783	?	?
Automóvil	9	0	110	1	0.9	1	0.9	1	0.947	?	?
Viaje	10	2	108	0	1	0.833	1	0.982	0.909	?	?
Internet	6	1	109	4	0.6	0.857	0.6	0.991	0.706	?	?
Overall	?	?	?	?	?	?	?	?	?	0.875	0.864

Cuadro 11: Tabla con los resultados del Experimento 5 con SVM como clasificador.

A continuación, se presentan sub workflows contenidos en metanodos del workflow asociado al Experimento 5 introducido en el apartado 4.5 de la memoria.

Metanodos #1608 y #1609 (Preprocesamiento_Documentos)

En la Figura 56 se muestra el contenido de los metanodos #1608 y #1609.

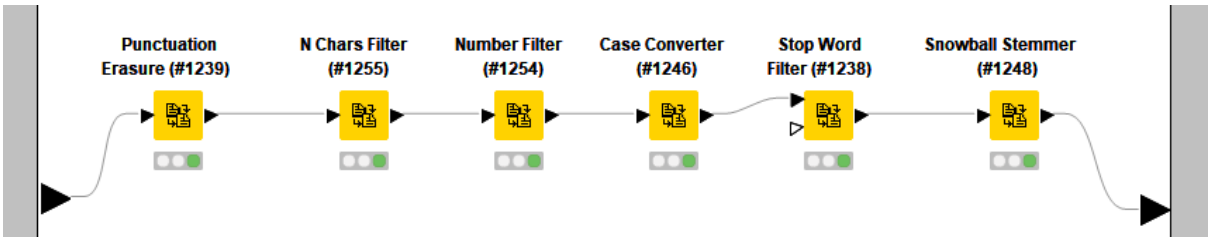

Figura 56: Contenido del metanodos #1608 y #1609, integrados en el workflow asociado al Experimento 5.

Estos metanodos en cuanto a funcionamiento (y configuración) son equivalentes al Metanodo #1389 del experimento 1 y difieren tan sólo en que estos no tienen descripción debajo del nodo a diferencia de los nodos contenidos en el metanodo #1389; por tanto, para obtener más información se remite a ver la descripción de este último metanodo mencionado en el anexo A.5.1.

Metanodos #1243 y #1216 (Term Filtering)

En la Figura 57 se muestra el contenido de los metanodos #1243 y #1216.

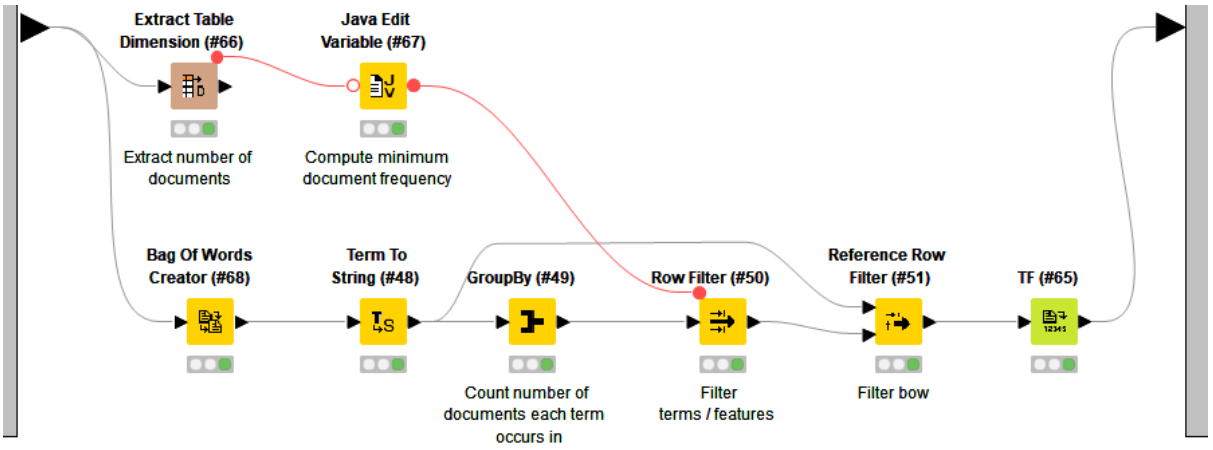

Figura 57: Contenido de los metanodos #1243 y #1216, integrados en el workflow asociado al Experimento 5.

El contenido de estos metanodos es igual que el del metanodo #95 del experimento 1 y difieren tan sólo en el ID del nodo esencialmente; por tanto, para obtener más información se remite a ver la descripción de este último metanodo mencionado en el anexo A.5.1.

Metanodo #1610 (Transformación)

En la Figura 58 se muestra el contenido del metanodo #1610.

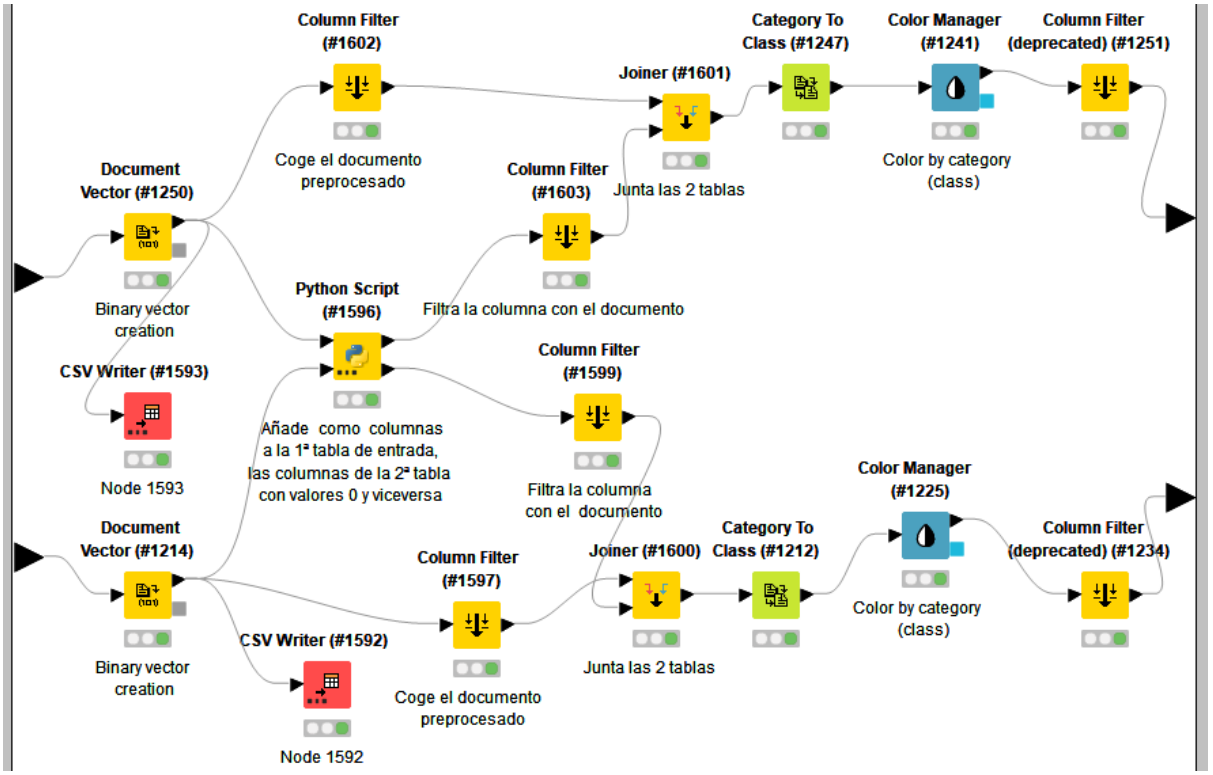

Figura 58: Contenido del metanodo #1610, integrado en el workflow asociado al Experimento 5.

Solo se va a explicar la rama superior de este metanodo ya que el procesamiento llevado a cabo en la inferior es equivalente pero empleando los datos de test (120 tweets) en vez de los de entrenamiento (12 textos basados en las palabras de los árboles de Zirano de las categorías) que se usan en la rama superior.

- El nodo #1250 (o #1214 en la rama inferior) crea un vector de documentos ignorando tags a partir de los documentos que tiene como entrada y empleando la frecuencia relativa de los términos (TF).
- El nodo #1593 intenta escribir en un CSV los datos que recibe del nodo #1250.
- El nodo #1602 (o #1599 en la rama inferior) filtra todas las columnas salvo el documento preprocesado.
- El nodo #1596 añade a la 1ª tabla de entrada aquellas columnas de la 2ª tabla que no están presentes en la 1ª tabla pero con valores 0 en las filas y viceversa.
- El nodo #1603 (o #1599 en la rama inferior) excluye la columna con el documento.
- Con el nodo #1601 (o #1600 en la rama inferior) se juntan, mediante operación de “Inner Join” por el atributo ID de la fila, las tablas de los nodos #1602 (o #1597 en la rama inferior) y #1603 (o #1599 en la rama inferior).
- El nodo #1247 (o #1212 en la rama inferior) añade una columna con el nombre de la categoría asociada a cada fila de la tabla obtenida del nodo anterior.
- El nodo #1241 (o #1225 en la rama inferior) asigna un color a cada categoría.
- Por último, el nodo #1251 (o #1234 en la rama inferior) filtra la columna con el documento preprocesado.

Metanodo #1612 (ClasificaciónÁrbolDecisión)

En la Figura 59 se muestra el contenido del metanodo #1612.

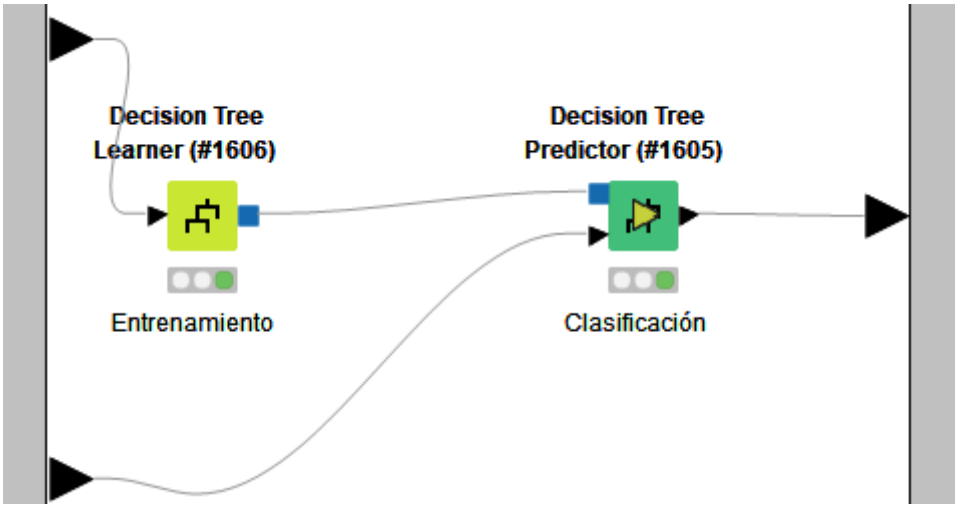

Figura 59: Contenido del metanodo #1612, integrado en el workflow asociado al Experimento 5.

- El nodo #1606 entrena y devuelve a su salida un modelo basado en árbol de decisión con los datos de entrada que recibe.
- El nodo #1605 predice a qué categoría pertenecen los datos de test que tiene como una de sus entradas, empleando el modelo entrenado en el nodo anterior para predecir.

Metanodo #1611 (ClasificaciónSVM)

En la Figura 60 se muestra el contenido del metanodo #1611.

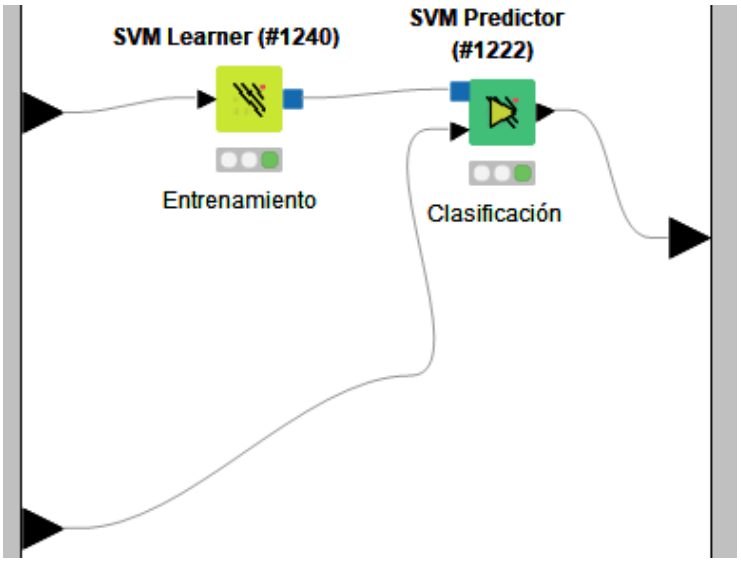

Figura 60: Contenido del metanodo #1611, integrado en el workflow asociado al Experimento 5.

- El nodo #1240 entrena y devuelve a su salida un modelo basado en SVM con los datos de entrada que recibe.
- El nodo #1222 predice a qué categoría pertenecen los datos de test que tiene como una de sus entradas, empleando el modelo entrenado en el nodo anterior para predecir.

A.5.6 Experimento 6

Primeramente, se describe el flujo de trabajo principal asociado al apartado “4.6 Experimento 6” de esta memoria (la Figura 14 muestra el flujo de trabajo correspondiente que se va a describir).

Descripción del flujo de trabajo

- El nodo #1702 lee 120 tweets, su categoría asociada y el nombre de la fuente de datos de la que provienen los tweets, que es “Twitter”.
- El nodo #1703 divide los datos en dos conjuntos: el de entrenamiento con el 70% de los datos y el de test con el 30%.
- Con el nodo #1714 se dejan los tweets de cada categoría y el propio nombre de la categoría por columnas, lo relativo a cada categoría en una misma fila.
- El metanodo #1723 tiene 12 salidas, cada una contiene una de las filas del nodo anterior.
- El metanodo #1725 principalmente enriquece los tweets de cada categoría con campos conceptuales de Zirano. Si se ejecuta el metanodo al completo, se van enriqueciendo las categorías (sus tweets realmente) en paralelo (dentro de cada categoría cada tweet se enriquece secuencialmente).
- El siguiente nodo, el #1718, simplemente junta los tweets enriquecidos en una columna.
- El metanodo #1722, preprocesa los tweets enriquecidos y los emplea para entrenar un modelo con árbol de decisión y otro con SVM que valida con los tweets de test, que no están enriquecidos.
- Los nodos #1726 y #1727 presentan los resultados de la predicción con los dos modelos entrenados en el nodo anterior: árbol de decisión (en nodo #1726) y SVM (en nodo #1727).

Seguidamente, se muestran la tabla 12 y la tabla 13 que muestran los resultados del Experimento 6 empleando árbol de decisión y SVM como clasificadores, respectivamente.

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specificity	F-meas...	Accuracy	Cohen'...
Atletismo	3	1	32	0	1	0.75	1	0.97	0.857	?	?
Natación	1	0	33	2	0.333	1	0.333	1	0.5	?	?
Televisión	3	22	11	0	1	0.12	1	0.333	0.214	?	?
Internet	3	0	33	0	1	1	1	1	1	?	?
Tienda	3	0	33	0	1	1	1	1	1	?	?
Dinero	0	0	33	3	0	?	0	1	?	?	?
Préstamo	0	0	33	3	0	?	0	1	?	?	?
Fútbol	0	0	33	3	0	?	0	1	?	?	?
Automóvil	0	0	33	3	0	?	0	1	?	?	?
Ordenador	0	0	33	3	0	?	0	1	?	?	?
Barco	0	0	33	3	0	?	0	1	?	?	?
Viaje	0	0	33	3	0	?	0	1	?	?	?
Overall	?	?	?	?	?	?	?	?	?	0.361	0.303

Cuadro 12: Tabla con los resultados del Experimento 6 con árbol de decisión como clasificador.

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specificity	D F-meas...	D Accuracy	D Cohen'...
Atletismo	3	3	30	0	1	0.5	1	0.909	0.667	?	?
Natación	2	0	33	1	0.667	1	0.667	1	0.8	?	?
Préstamo	3	2	31	0	1	0.6	1	0.939	0.75	?	?
Fútbol	2	0	33	1	0.667	1	0.667	1	0.8	?	?
Automóvil	3	0	33	0	1	1	1	1	1	?	?
Ordenador	3	0	33	0	1	1	1	1	1	?	?
Televisión	2	1	32	1	0.667	0.667	0.667	0.97	0.667	?	?
Internet	2	1	32	1	0.667	0.667	0.667	0.97	0.667	?	?
Tienda	3	1	32	0	1	0.75	1	0.97	0.857	?	?
Barco	3	0	33	0	1	1	1	1	1	?	?
Viaje	2	0	33	1	0.667	1	0.667	1	0.8	?	?
Dinero	0	0	33	3	0	?	0	1	?	?	?
Overall	?	?	?	?	?	?	?	?	?	0.778	0.758

Cuadro 13: Tabla con los resultados del Experimento 6 con SVM como clasificador.

A continuación, se presentan sub workflows contenidos en metanodos del workflow asociado al Experimento 6 introducido en el apartado 4.6 de la memoria.

Metanodo #1723

En la Figura 61 se muestra el contenido del metanodo #1723.

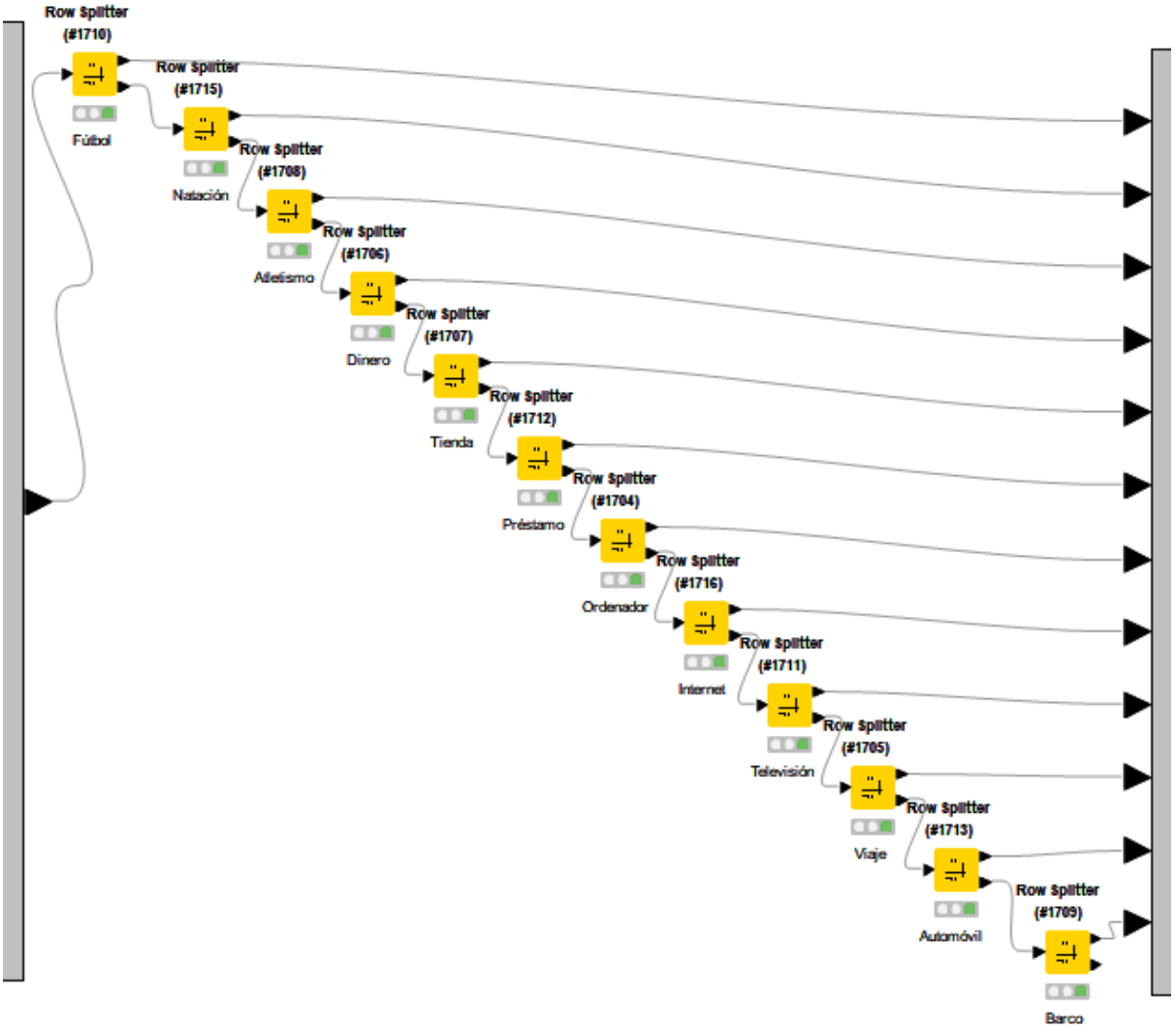

Figura 61: Contenido del metanodo #1723, integrado en el workflow asociado al Experimento 6.

Solo se va a explicar el primer nodo, el #1710, ya que la descripción del resto es igual pero cambiando el nombre de la categoría que se utiliza para su configuración conforme al que aparece en la descripción de abajo de cada nodo.

- El nodo #1710 recibe a la entrada una tabla con varias filas, cada una con varias columnas con tweets de una determinada categoría y una columna con el nombre de la categoría de esos tweets. El nodo se queda con la fila de la tabla de entrada (descarta el resto) que cumple tener como valor en la celda de la columna con nombres de categorías, el nombre de la categoría que el nodo tiene como descripción abajo en la figura, que en este caso es “Fútbol”.

Metanodo #1725

El contenido del metanodo #1725 se muestra, en orden descendente, distribuido entre la Figura 62, la Figura 63 y la Figura 64.

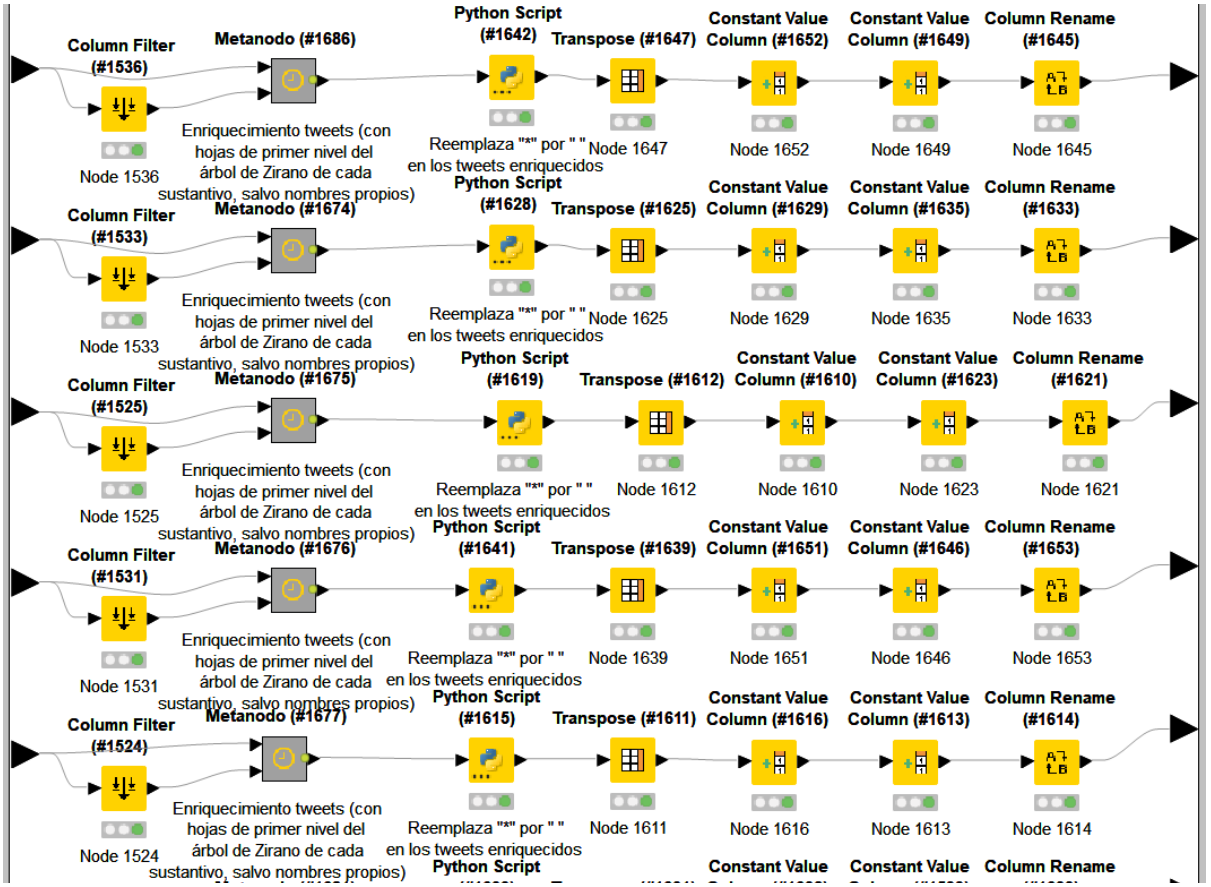

Figura 62: Contenido de las ramas superiores del metanodo #1725, integrado en el workflow asociado al Experimento 6.

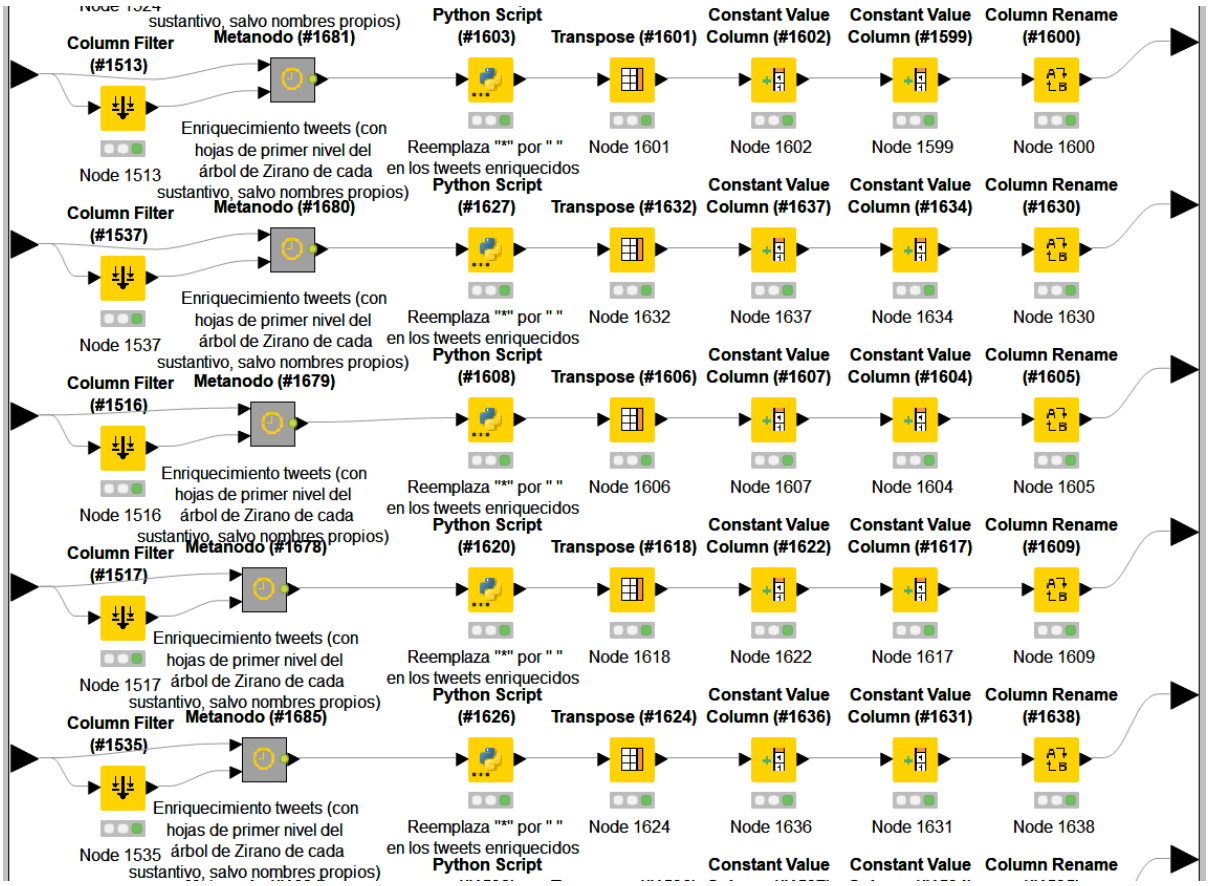

Figura 63: Contenido de las ramas centrales del metanodo #1725, integrado en el workflow asociado al Experimento 6.

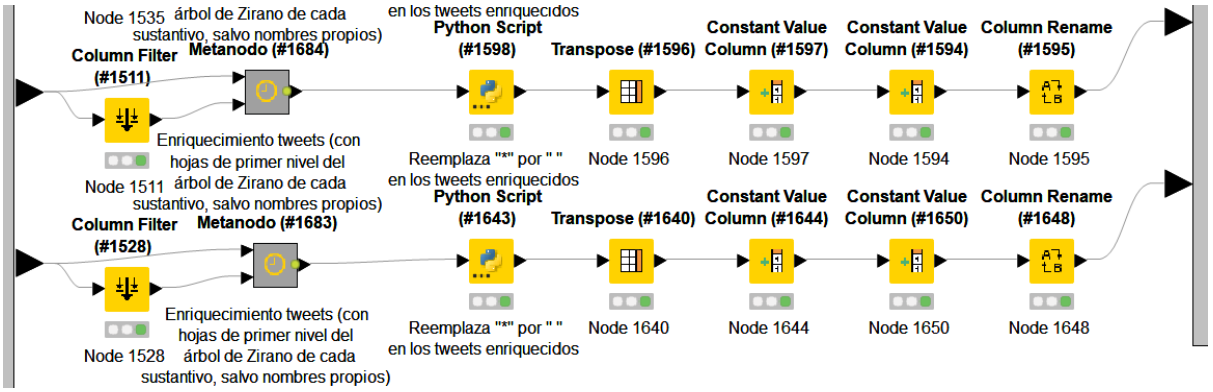

Figura 64: Contenido de las ramas inferiores del metanodo #1725, integrado en el workflow asociado al Experimento 6.

Se procede a explicar solamente la última rama (o fila) del workflow, que se puede ver en la Figura 64, ya que el resto de ramas son equivalentes pero cambiando los datos de los tweets y la categoría a la que están asociados.

- El nodo #1528 filtra todas las columnas salvo la que tiene el nombre de la categoría.
- El metanodo #1683 enriquece con campos conceptuales de Zirano los tweets de la categoría correspondiente al dato de salida del nodo anterior.
- El nodo #1643 reemplaza los caracteres “*” por “ “ de los tweets enriquecidos.
- El nodo #1640 transpone los tweets (su contenido), es decir, transforma las filas a columnas y las columnas a filas, de forma que los tweets ahora quedan en una tabla a razón de uno por fila, todos en una misma columna.

- Con el nodo #1644 se añade una columna con un valor constante en sus celdas que es el nombre de la categoría a la que están asociados los tweets de la rama, en este caso la categoría “Barco”.
- De forma similar al anterior nodo, el #1650 añade una columna con valor constante en sus celdas que viene a representar el nombre de la fuente de datos de la que se obtienen los tweets de la rama que en este caso es “Twitter y Zirano” (los tweets se han obtenido originalmente de Twitter pero se han enriquecido con determinados campos conceptuales de Zirano).
- Por último, el nodo #1648 cambia el nombre de la columna con los tweets estableciendo “Tweet” como su nombre.

Metanodo #1683

En la Figura 65 se muestra el contenido del metanodo #1683.

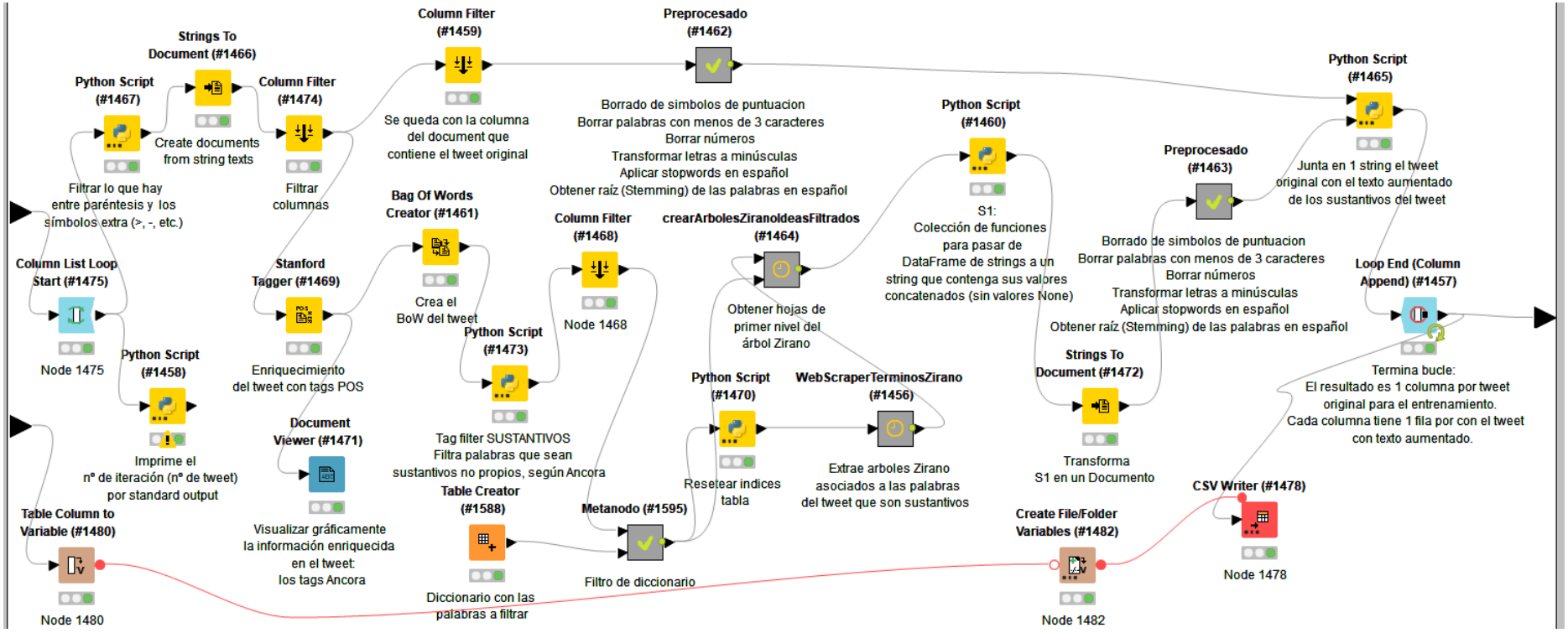

Figura 65: Contenido del metanodo #1683, integrado en el workflow asociado al Experimento 6.

- El nodo #1475 inicia un bucle que itera para cada tweet de entrada (en cada iteración incluye también el nombre de la categoría asociada al tweet).
- El nodo #1458 simplemente imprime en cada iteración por la salida estándar el número de iteración (que puede ser útil para el usuario que ejecuta el workflow para obtener retroalimentación de la iteración por la que se va, ya que la ejecución de este workflow puede ser costosa en tiempo).
- El nodo #1467 filtra del tweet principalmente lo que hay entre paréntesis (incluídos los propios símbolos de paréntesis) y los símbolos extra que cumplan esta expresión regular: [^a-zA-Z0-9 áéíóúÁÉÍÓÚñÑü].
- El nodo #1466 transforma el tweet (que es de tipo String) a tipo Document de KNIME.
- Con el nodo #1474 se descartan algunas posibles columnas innecesarias, manteniendo la del tweet original y la del tweet de tipo Document.

A continuación, de los nodos que faltan por describir, se comentan los situados en la rama superior del workflow:

- El nodo #1459 filtra la columna con el tweet original y mantienen solo la del tweet de tipo Document.
- Este metanodo realiza tareas de preprocesamiento del texto (quita símbolos de puntuación, filtra palabras con menos de 3 caracteres, aplica stemming, etc.).
- El nodo #1465 junta en un String el tweet original con el texto aumentado de los sustantivos (sin incluir nombres propios) del tweet.

Seguidamente se comentan los nodos de la rama situada alrededor del centro del workflow:

- El nodo #1469 enriquece los tweets originales asignándoles tags de tipo POS (Part of Speech) que asigna etiquetas de tipo gramatical a las palabras del documento empleando el modelo de lenguaje llamado “Spanish distsim” que está formado por el conjunto de etiquetas ancora francés español.
- El nodo #1471 permite visualizar de forma gráfica la información con la que se ha enriquecido el tweet, esto es, las etiquetas de ancora nombradas en el nodo anterior.
- El nodo #1461 crea el modelo BoW del documento con el tweet enriquecido con etiquetas POS.
- Con el nodo #1473 se obtienen todas las palabras del documento excepto las que sean sustantivos comunes (no se incluyen nombres propios en la salida).
- El nodo #1468 filtra las columnas de la tabla de entrada dejando tan solo la que contiene los sustantivos obtenidos del nodo anterior.
- El metanodo #1595 implementa un filtro de diccionario que obtiene como salida una tabla como la de entrada recibida del nodo anteriormente comentado (el #1468) en la que se han filtrado aquellas palabras (o filas) que coincidan con alguna de las entradas de la tabla del nodo #1588. Este último nodo, el #1588 es un diccionario que almacena las palabras a filtrar porque se ha comprobado que estas dan problemas posteriormente al intentar realizar el web scraping en Zirano con ellas.
- El nodo #1470 resetea los índices de la tabla con los sustantivos filtrados, comenzando desde la fila 0 hasta la N-ésima.
- El metanodo #1456 extrae los árboles de Zirano (las ideas relacionadas y nivel en el árbol de cada una) asociados a los sustantivos comunes extraídos y filtrados del tweet original.
- El metanodo #1464 obtiene las ideas que son hojas del primer nivel de hojas de cada árbol de Zirano obtenido con el nodo anteriormente comentado (el #1456).
- Con el nodo #1460 se juntan todas las palabras obtenidas con el nodo anterior en una tabla con un solo String (separadas por espacios incluso si una idea del nodo anterior contenía varias palabras, cada palabra está separada por un espacio en blanco).
- El nodo #1472 obtiene un documento (de tipo Document) a partir del String obtenido en el nodo anterior.
- El contenido del metanodo #1463 es igual en cuanto a función que el del metanodo #1462 comentado en la rama previa.

Por último se describen los nodos de la rama inferior:

- El nodo #1480 convierte a variable de flujo de KNIME el nombre de la categoría que recibe como entrada.

- El nodo #1482 crea un fichero, dentro de una carpeta situada en una ruta especificada en la configuración del nodo, con nombre igual al de la categoría recibida como entrada (“Barco” en este caso) y con extensión “.csv”. A la salida devuelve la ruta al fichero de tipo “base_folder” como variable de flujo de KNIME, de forma que puede emplearse en otros nodos para especificar la ruta del fichero.
- El nodo #1457 acumula en diferentes columnas los resultados que obtiene del nodo #1465 en cada iteración.
- Por último, nodo #1478 intenta escribir los datos acumulados del bucle en un CSV en el fichero cuya ruta obtiene del nodo #1482.

Metanodo #1456

En la Figura 66 se muestra el contenido del metanodo #1456.

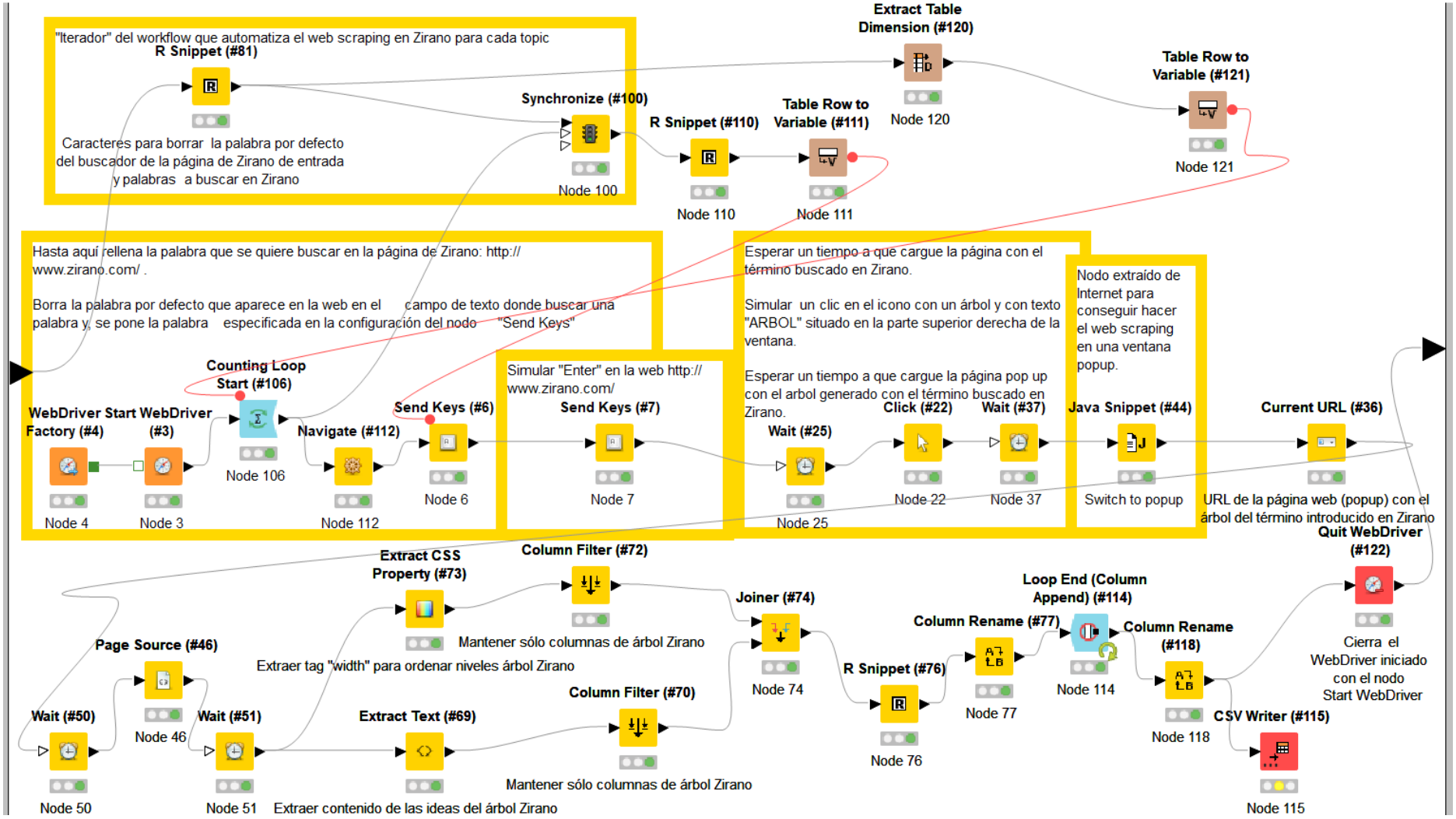

Figura 66: Contenido del metanodo #1456, integrado en el workflow asociado al Experimento 6.

El contenido de este metanodo es igual que el workflow principal del apartado 3.3.2 (Extracción de campos conceptuales) pero si no tuviera metanodos, es decir, con todos los nodos que alberga en cada metanodo expandidos en el workflow principal y difiere además en algunos aspectos:

- Este metanodo #1456 no tiene el nodo #116 que sí tiene el del apartado 3.3.2 de la memoria, pero recibe como entrada del metanodo una tabla que en parte contiene las palabras sustantivos extraídas del tweet y que se seleccionan en el nodo #81 para buscarlas en Zirano posteriormente.
- Los nodos #121 y #122 no están presentes en el workflow del apartado 3.3.2 de la memoria pero sí en este metanodo. Consiguen parametrizar el número de iteraciones del bucle obteniendo la dimensión de la tabla de entrada que será igual al número de filas de la dicha tabla de entrada, lo cual el nodo #122 se lo pasa en forma de variable de flujo al nodo #106 que inicia el bucle de forma que controla el número de iteraciones que se realizan.
- El nodo #122 cierra el WebDriver iniciado con el nodo Start WebDriver (nodo #3).

Metanodo #1462 y #1463

En la Figura 67 se muestra el contenido de los metanodos #1462 y #1463.

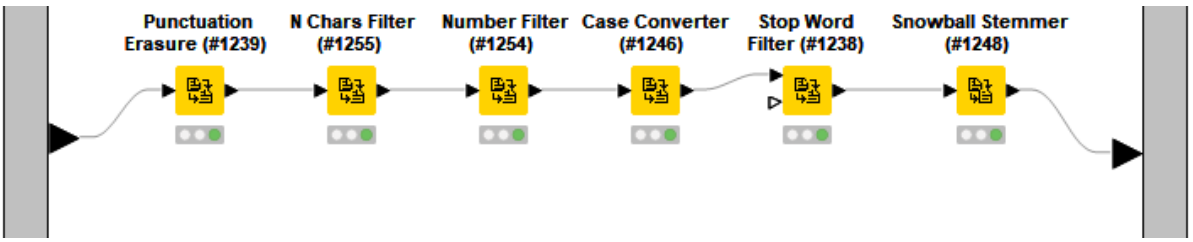

Figura 67: Contenido de los metanodos #1462 y #1463, integrados en el workflow asociado al Experimento 6.

- El contenido del metanodo #1462 y el del #1463 es igual que el de los Metanodos #1608 y #1609 (Preprocesamiento_Documentos) del experimento 5, por lo cual para más información se remite al apartado relativo a estos nodos del anexo A.5.5.

Metanodo #1464

En la Figura 68 se muestra el contenido del metanodo #1464.

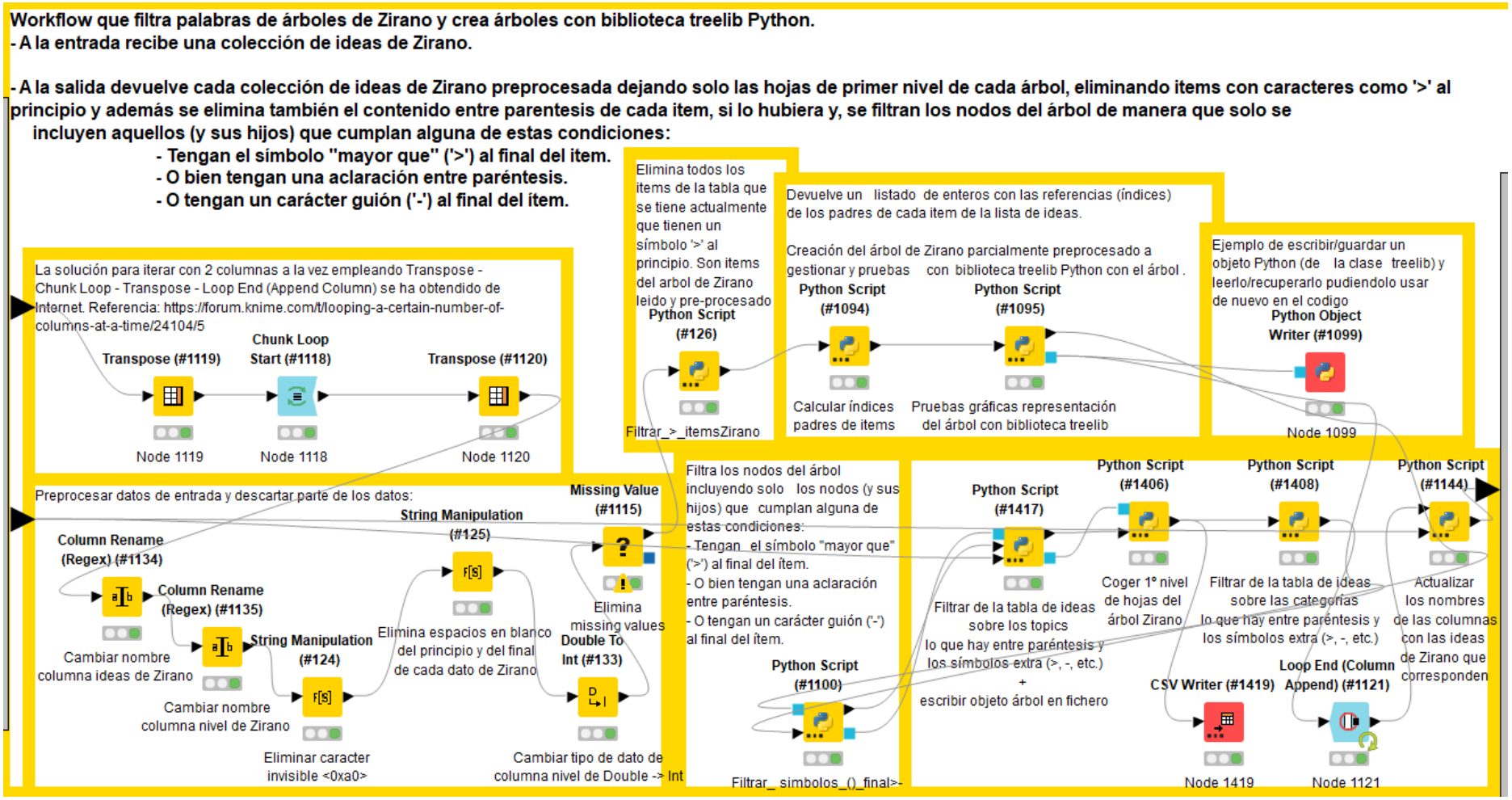

Figura 68: Contenido del metanodo #1464, integrado en el workflow asociado al Experimento 6.

El contenido de este metanodo es igual que el workflow principal del apartado 3.3.3 (Creación y filtrado de árboles) pero si no tuviera metanodos, es decir, con todos los nodos que alberga en cada metanodo expandidos en el workflow principal y, difiere además en algunos aspectos:

- Este metanodo #1464 no tiene el nodo #1406 a diferencia del workflow del apartado 3.3.3 pero una de sus entradas de datos del metanodo es la que recibe como entrada el nodo #1119 con las colecciones de ideas de árboles de Zirano (y el nivel de altura asociado a cada idea) obtenidas en el web scraping del metanodo #1456.
- En este metanodo #1464 la mayoría de los nodos que van después del nodo #1100 en el flujo difieren de los del workflow del apartado 3.3.3 de la memoria, se procede a comentar los propios de este metanodo:
  - El nodo #1417 filtra de la tabla de ideas de árboles de Zirano de las palabras lo que hay entre paréntesis y los símbolos extra (>, -, etc.) y además escribe el objeto árbol (de tipo treelib de Python) en un fichero aparte de mostrar algunos datos por la salida estándar.
  - Con el nodo #1406 se cogen las hojas de primer nivel de los árboles de Zirano construidos en el nodo anterior.
  - El nodo #1416 intenta escribir el resultado (tabla) del nodo anterior en un fichero CSV.
  - El nodo #1408 filtra de la tabla con las ideas hojas de primer nivel obtenidas del nodo anterior lo que hay entre paréntesis y los símbolos extra (>, -, etc.).
  - En este metanodo, el nodo #1121 está en diferente posición respecto al workflow del apartado 3.3.3 de la memoria pero su función es la misma: acumular los resultados que obtiene en cada iteración en columnas.
  - Por último, el nodo #1144 actualiza los nombres de las columnas con las ideas de Zirano, igual que se hacía en el nodo #1144 del apartado 3.3.3 de la memoria aunque se ha empleado en un punto distinto dentro del workflow.

Metanodo #1595

En la Figura 69 se muestra el contenido del metanodo #1595.

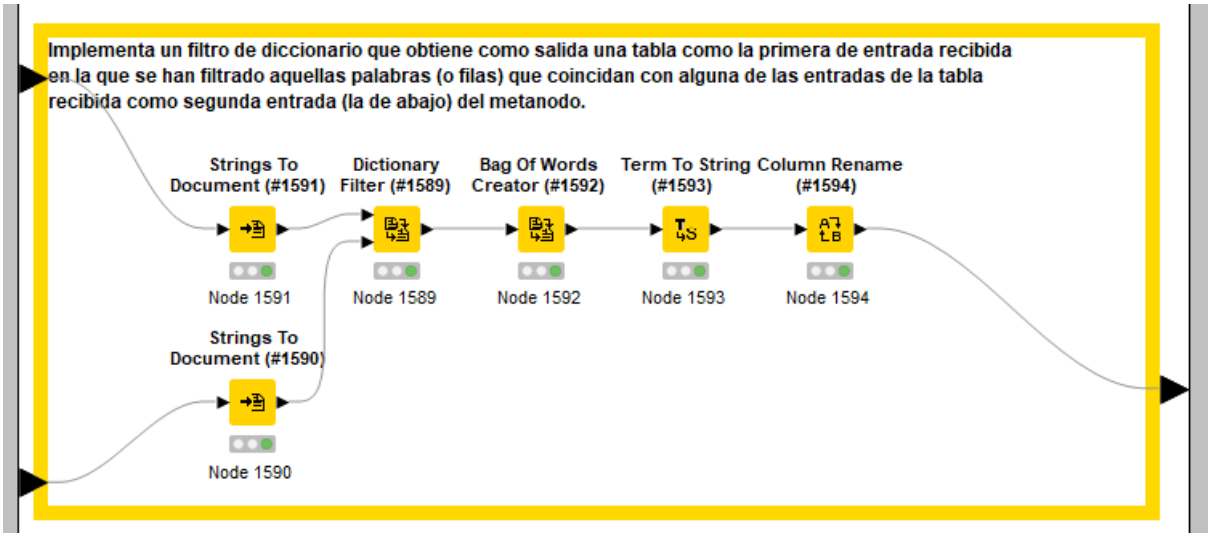

Figura 69: Contenido del metanodo #1595, integrado en el workflow asociado al Experimento 6.

- El nodo #1591 transforma la columna ‘palabra’ de la tabla de entrada (la de arriba) a tipo Document.
- Ídem el nodo #1590 que el nodo #1591 pero con la tabla de entrada de abajo.
- El nodo #1589 filtra las palabras de la tabla (con documentos) de arriba contenidas en la tabla o columna de abajo.
- El nodo #1592 obtiene un modelo BoW a partir del documento preprocesado obtenido del nodo anterior.
- Con el nodo #1593 se transforman los términos obtenidos tras el BoW a tipo String.
- El nodo #1594 renombra con el nombre “palabra” a la columna con los términos de tipo String.

Metanodo #1722

En la Figura 70 se muestra el contenido del metanodo #1722.

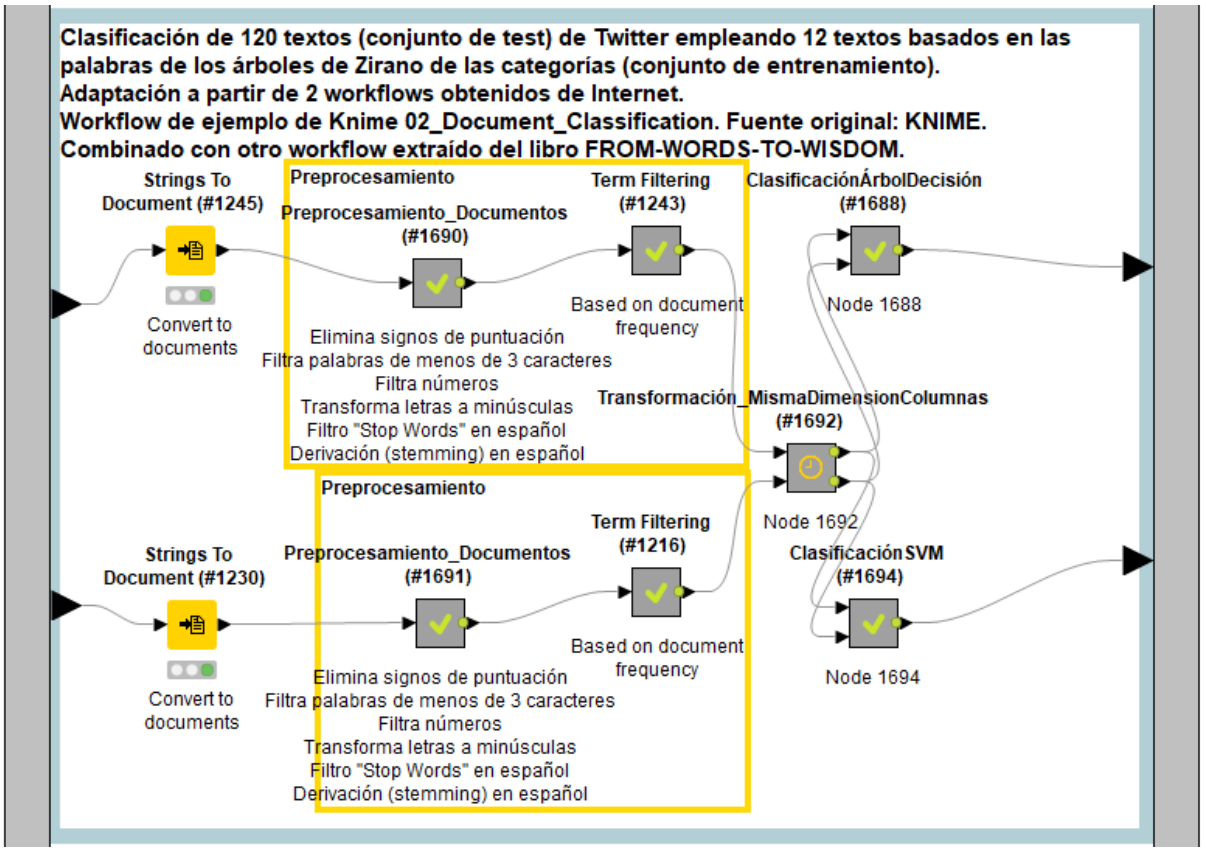

Figura 70: Contenido del metanodo #1722, integrado en el workflow asociado al Experimento 6.

El contenido de este metanodo es equivalente al workflow del experimento 5 de la memoria (véase el apartado 4.5 de la memoria) salvo que difiere en algunos aspectos:

- Aunque los IDs de algunos metanodos de este metanodo no coinciden con el del workflow del experimento 5, son equivalentes en el contenido de sus metanodos las siguientes parejas de cada workflow:
  - El contenido del metanodo #1690 de este workflow y el del #1608 del workflow del experimento 5.
  - Metanodo #1691 de este metanodo y metanodo #1609 del workflow del experimento 5.
  - Metanodo #1688 de este workflow y metanodo #1612 del workflow del experimento 5.
  - Metanodo #1694 de este workflow y metanodo #1611 del workflow del experimento 5.
- No tiene 2 nodos CSV Reader como entrada sino que en vez de ellos el metanodo tiene 2 entradas de las que recibe los datos de entrada a preprocesar y con los que se obtendrá el modelo.
- Este metanodo a diferencia del workflow del experimento 5 de la memoria no tiene los 2 nodos Scorer al final del workflow.

Metanodo #1690 y #1691

En la Figura 71 se muestra el contenido de los metanodos #1690 y #1691.

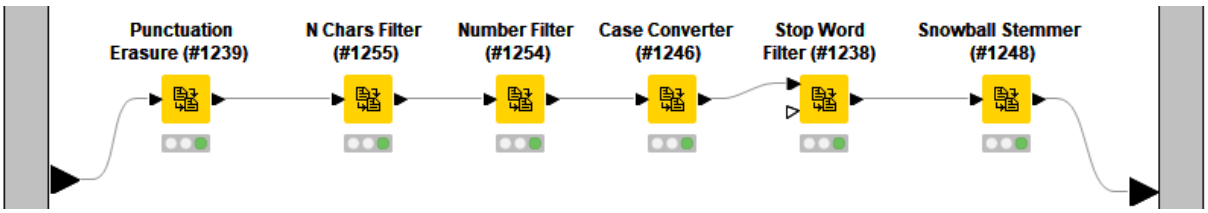

Figura 71: Contenido de los metanodos #1690 y #1691, integrados en el workflow asociado al Experimento 6.

- El contenido de estos metanodos es igual que el de los Metanodos #1608 y #1609 (Preprocesamiento_Documentos) del experimento 5, por lo cual para más información se remite al apartado relativo a estos nodos del anexo A.5.5.

Metanodos #1243 y #1216

En la Figura 72 se muestra el contenido de los metanodos #1243 y #1216.

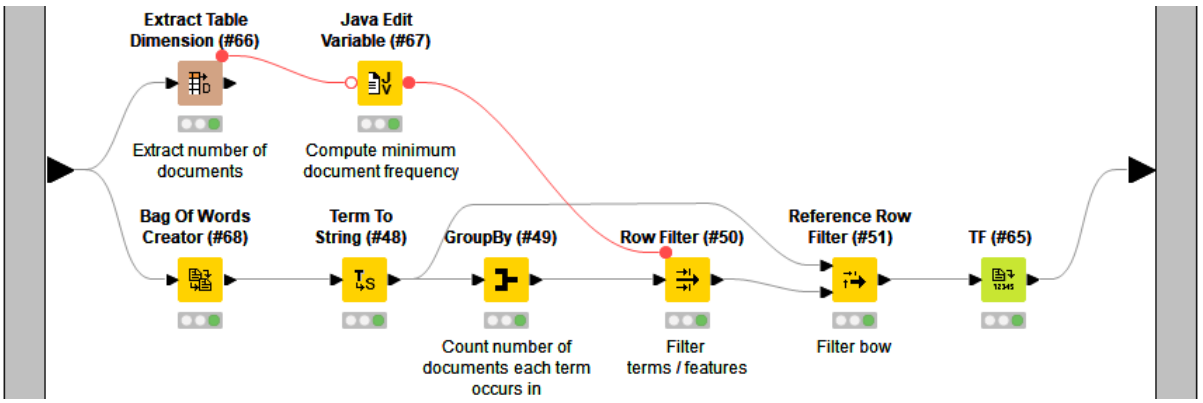

Figura 72: Contenido de los metanodos #1243 y #1216, integrados en el workflow asociado al Experimento 6.

- El contenido de estos metanodos es igual que el del metanodo #95 del experimento 1 y difiere tan solo en el ID del nodo esencialmente; por tanto, para obtener más información se remite a ver la descripción de este último metanodo mencionado en el anexo A.5.1.

Metanodo #1692

En la Figura 73 se muestra el contenido del metanodo #1692.

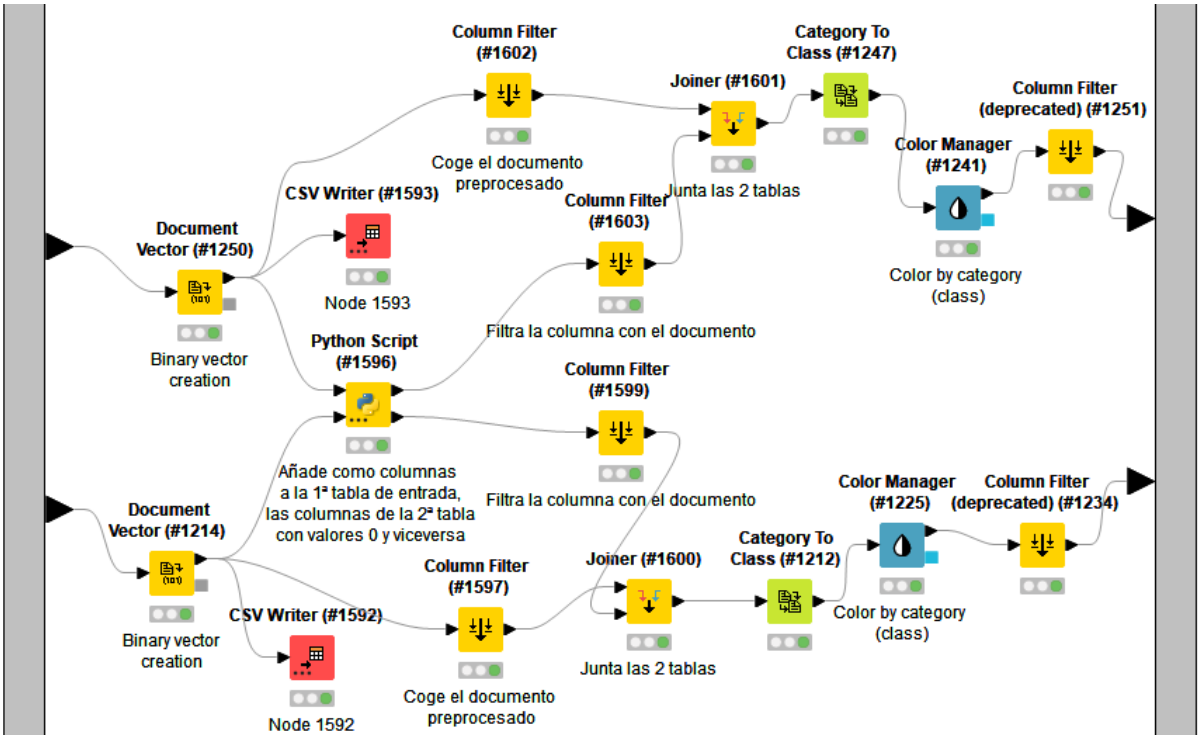

Figura 73: Contenido del metanodo #1692, integrado en el workflow asociado al Experimento 6.

- La descripción de los nodos contenidos en este metanodo es equivalente a la del metanodo #1610 situada en el Anexo A.5.5.

Metanodo #1688

En la Figura 74 se muestra el contenido del metanodo #1688.

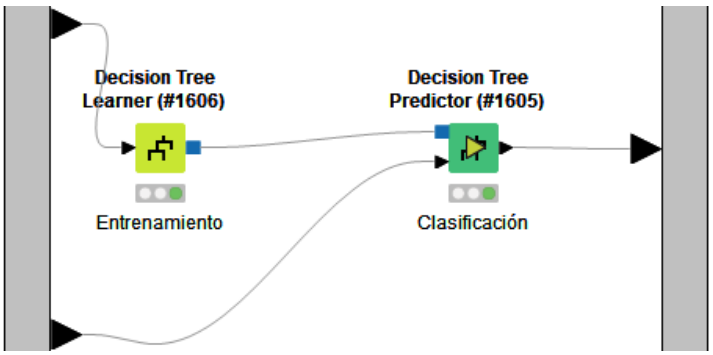

Figura 74: Contenido del metanodo #1688, integrado en el workflow asociado al Experimento 6.

- La descripción de los nodos contenidos en este metanodo es equivalente a la del metanodo #1612 situada en el Anexo A.5.5.

Metanodo #1694

En la Figura 75 se muestra el contenido del metanodo #1694.

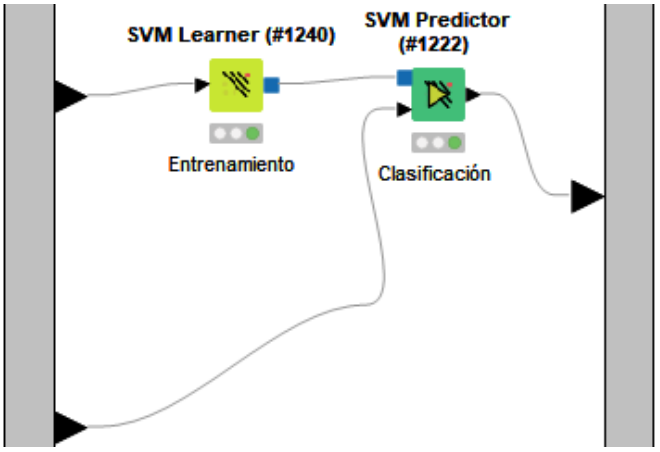

Figura 75: Contenido del metanodo #1694, integrado en el workflow asociado al Experimento 6.

- La descripción de los nodos contenidos en este metanodo es equivalente a la del metanodo #1611 situada en el Anexo A.5.5.

Bibliografía

[1] «Minería de textos», *Wikipedia, la enciclopedia libre*. oct. 17, 2020. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Miner%C3%ADa_de_textos&oldid=130135387

[2] «Programación visual», *Wikipedia, la enciclopedia libre*. abr. 17, 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_visual&oldid=134839454

[3] «The Current State of the Art in Natural Language Processing (NLP)», *ZappyAI*. https://zappy.ai/ai-blogs/the-current-state-of-the-art-in-natural-language-processing-nlp (accedido jul. 01, 2021).

[4] «Metodología de la minería de textos», abr. 06, 2015. https://web.archive.org/web/20150406110300/http://textmining.es/metodolog%C3%ADa%20de%20la%20miner%C3%ADa%20de%20textos.html (accedido jul. 01, 2021).

[5] «Nube de palabras», *Wikipedia, la enciclopedia libre*. jun. 28, 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Nube_de_palabras&oldid=136648147

[6] J. Brownlee, «A Gentle Introduction to the Bag-of-Words Model», *Machine Learning Mastery*, oct. 08, 2017. https://machinelearningmastery.com/gentle-introduction-bag-words-model/ (accedido jul. 01, 2021).

[7] RAE, «@jesugacabrales #RAEconsultas Es imposible saber el número de palabras de una lengua. La última edición del diccionario académico (2014), registraba 93 111 artículos y 195 439 acepciones.», *@RAEinforma*, ene. 23, 2019. https://twitter.com/RAEinforma/status/1088104147612774403 (accedido jul. 01, 2021).

[8] S. Valcheva, «Text Mining Algorithms List: Text Classification Categorization Clustering», *Blog For Data-Driven Business*, dic. 18, 2017. http://www.intellspot.com/text-mining-algorithms/ (accedido jul. 01, 2021).

[9] V. Tursi y R. Silipo, *From Words to Wisdom*. Zurich: KNIME Press, 2021.

[10] «Reglas de asociación (y detección de anomalías) con futbolistas usando R y estadísticas de FIFA», abr. 05, 2020. https://danielredondo.com/posts/20200405_reglas_asociacion/ (accedido jul. 01, 2021).

[11] I. Alsmadi y G. K. Hoon, «Term weighting scheme for short-text classification: Twitter corpuses», *Neural Comput & Applic*, vol. 31, n.º 8, pp. 3819-3831, ago. 2019, doi: 10.1007/s00521-017-3298-8.

[12] E. Sarioglu, H.-A. Choi, y K. Yadav, «Clinical Report Classification Using Natural Language Processing and Topic Modeling», en *2012 11th International Conference on Machine Learning and Applications*, dic. 2012, vol. 2, pp. 204-209. doi: 10.1109/ICMLA.2012.173.

[13] «Qué son los árboles de decisión y para qué sirven», *Máxima Formación*, may 26, 2020. https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/ (accedido jul. 01, 2021).

[14] L. L. Yuste, «SISTEMA ANALIZADOR Y RECOMENDADOR DE BLOGS». Accedido: jul. 01, 2021. [En línea]. Disponible en: https://zaguan.unizar.es/record/6487/files/TAZ-PFC-2011-625.pdf

[15] J. Brownlee, «Support Vector Machines for Machine Learning», *Machine Learning Mastery*, abr. 19, 2016. https://machinelearningmastery.com/support-vector-machines-for-machine-learning/ (accedido jul. 01, 2021).

[16] «Data augmentation», *Wikipedia*. jun. 18, 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Data_augmentation&oldid=1029174861

[17] E. Ma, «Data Augmentation in NLP», *Medium*, jun. 04, 2019. https://towardsdatascience.com/data-augmentation-in-nlp-2801a34dfc28 (accedido jul. 01, 2021).

[18] M. R. Perez, «Prototipado de sistemas interactivos». 2010. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://repositori.udl.cat/bitstream/handle/10459.1/45691/Ramon.pdf?sequence=1

[19] «Consultoría KNIME. Partner oficial en España», *LIS Data Solutions*. https://www.lisdatasolutions.com/knime/ (accedido jul. 01, 2021).

[20] «KNIME», *Wikipedia, la enciclopedia libre*. nov. 23, 2020. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=KNIME&oldid=131159223

[21] «Orange», *Wikipedia, la enciclopedia libre*. sep. 01, 2019. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Orange&oldid=118815531

[22] B. L. Ljubljana University of, «Data Mining». https://orangedatamining.com/ (accedido jul. 01, 2021).

[23] «RapidMiner», *Wikipedia, la enciclopedia libre*. sep. 26, 2019. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=RapidMiner&oldid=119726295

[24] G. Inc, «KNIME vs RapidMiner: Gartner Peer Insights 2021», *Gartner*. https://www.gartner.com/market/data-science-machine-learning-platforms/compare/knime-vs-rapidminer (accedido jul. 01, 2021).

[25] «Python leads the 11 top Data Science, Machine Learning platforms: Trends and Analysis», *KDnuggets*. https://www.kdnuggets.com/python-leads-the-11-top-data-science-machine-learning-platforms-trends-and-analysis.html/2/ (accedido jul. 01, 2021).

[26] «Examples - Ludwig». https://ludwig-ai.github.io/ludwig-docs/examples/#text-classification (accedido jul. 01, 2021).

[27] *ludwig-ai/ludwig*. Ludwig, 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://github.com/ludwig-ai/ludwig

[28] «Deep Learning Studio», *Wikipedia*. jul. 07, 2018. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Deep_Learning_Studio&oldid=849282781

[29] «Deep Learning Studio», *DeepCognition.AI*. https://deepcognition.ai/deep-learning-studio/ (accedido jul. 01, 2021).

[30] «Counting characters». https://developer.twitter.com/en/docs/counting-characters (accedido jul. 01, 2021).

[31] «Diccionario analógico conceptual», *Wikipedia, la enciclopedia libre*. feb. 04, 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Diccionario_anal%C3%B3gico_conceptual&oldid=132950453

[32] «Zirano». http://www.zirano.com/ (accedido jul. 01, 2021).

[33] KNIMETV, *Common Steps in a Text Mining Project*. Accedido: jul. 01, 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=uzw6tNsYAdc&list=RDCMUCRbKmV_XYB7C12SPBokLVHQ&start_radio=1&t=1s&ab_channel=KNIMETV

[34] KNIMETV, *Topic Detection with Text Mining*. Accedido: jul. 01, 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=upAwDcw9ra4&ab_channel=KNIMETV

[35] «KNIME Twitter Connectors», *KNIME Hub*. https://hub.knime.com/knime/extensions/org.knime.features.ext.twitter/latest (accedido jul. 01, 2021).

[36] «String Emoji Filter – takbb», *KNIME Hub*. https://hub.knime.com/takbb/spaces/Public/latest/String%20Emoji%20Filter~jfYhKjzmydqXso38 (accedido jul. 01, 2021).

[37] «How to Set Up the Python Extension», *KNIME*. https://www.knime.com/blog/how-to-setup-the-python-extension (accedido jul. 01, 2021).

[38] «KNIME Interactive R Statistics Integration Installation Guide». https://docs.knime.com/2020-12/r_installation_guide/index.html#_introduction (accedido jul. 01, 2021).

[39] «Selenium Nodes — Download», *Selenium Nodes*. https://seleniumnodes.com (accedido jul. 01, 2021).

[40] «Selenium_Popup_Window [Workflow]», *NodePit*. https://nodepit.com/workflow/com.nodepit.space%2Fqqilihq%2Fpublic%2FSelenium_Popup_Window.knwf (accedido jul. 01, 2021).

[41] «treelib package — treelib 1.5.5 documentation». https://treelib.readthedocs.io/en/latest/treelib.html (accedido jul. 01, 2021).

[42] «Treelib :: Anaconda.org». https://anaconda.org/conda-forge/treelib (accedido jul. 01, 2021).

[43] «Search operators». https://developer.twitter.com/en/docs/twitter-api/v1/rules-and-filtering/search-operators (accedido jul. 18, 2021).

[44] «Document Classification: Model Training and Deployment – kilian.thiel», *KNIME Hub*. https://hub.knime.com/knime/spaces/Examples/latest/08_Other_Analytics_Types/01_Text_Processing/02_Document_Classification~yhjfXaKzovCY83at (accedido jul. 01, 2021).

[45] «Dictionary based tagging – kilian.thiel», *KNIME Hub*. https://hub.knime.com/knime/spaces/Examples/latest/08_Other_Analytics_Types/01_Text_Processing/04_Dictionary_based_Tagging~IBp9LRLyNKA9r0H6 (accedido jul. 01, 2021).

[46] «SQLite Home Page». https://www.sqlite.org/index.html (accedido jul. 01, 2021).

[47] «Downloads - DB Browser for SQLite». https://sqlitebrowser.org/dl/ (accedido jul. 01, 2021).

[48] «DBDAp». http://webdiis.unizar.es/~silarri/DBDAp.html (accedido jul. 12, 2021).

[49] «Ilarri Artigas». http://www.aenui.net/ojs/index.php?journal=revision&page=article&op=view&path%5B%5D=234&path%5B%5D=497 (accedido jul. 01, 2021).

[50] David Sherlock, *Install R for knime: error with install.packages(«Rserve»,,'http://rforge.net/',type='source')*. Accedido: jul. 01, 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=DC95yTgEa8I&ab_channel=DavidSherlock

[51] MPI-CBG, *knime-mpicbg/knime-scripting*. 2021. Accedido: jul. 01, 2021. [En línea]. Disponible en: https://github.com/knime-mpicbg/knime-scripting

[52] «KNIME Hub», *KNIME Hub*. https://hub.knime.com/ (accedido jul. 01, 2021).

[53] «Pyarrow :: Anaconda.org». https://anaconda.org/conda-forge/pyarrow (accedido jul. 01, 2021).

[54] elvis, «A Light Introduction to Transformer-XL», *Medium*, ene. 11, 2019. https://medium.com/dair-ai/a-light-introduction-to-transformer-xl-be5737feb13 (accedido jul. 01, 2021).