



Universidad Complutense de Madrid

Facultad de Informática

Trabajo de Fin de Grado

*Minería de Procesos aplicada al estudio de
Wikis*

Ignacio García Sánchez-Migallón

Director:

Javier Arroyo Gallardo

Madrid, Mayo, 2019

Probando formato

Resumen

La escritura colaborativa siempre ha sido uno de los pilares de internet. La colaboración desinteresada entre los diferentes usuarios ha hecho posible la existencia de comunidades cuyo único propósito es la difusión del conocimiento: las Wikis. Estas comunidades han sido ampliamente estudiadas, sin embargo, los procesos inherentes a su propio funcionamiento y estructura no son del todo conocidos. En este proyecto proponemos y aplicamos una serie de técnicas conocidas como minería de procesos para descubrir y analizar estos procesos existentes en la labor de la escritura colaborativa haciendo uso de la Wikipedia Española como referencia. Comenzamos haciendo uso de la taxonomía de intenciones tras cada revisión compuesta de 13 categorías desarrollada por Diyi Yang et al. Haciendo uso de su dataset ya etiquetado desarrollamos nuestro propio modelo predictivo alcanzando un valor de F1 micro de 0.66. Con este modelo generamos un corpus compuesto de diferentes *artículos destacados*. Aplicaremos una serie de técnicas de minería social y de procesos gracias a las cuales descubrimos la estructura colaborativa de los usuarios, los procesos seguidos por los artículos así como los procesos seguidos por los propios usuarios en sus sesiones de edición. Los resultados muestran que la colaboración a veces es organizada y sigue una cierta estructura. Además, observamos explícitamente determinados patrones en el comportamiento de los usuarios que verifican los hallazgos obtenidos en otros estudios.

Palabras clave: Minería de procesos, Minería social, Aprendizaje automático, ProM, Petri net, Wikia, Wiki, Colaboración, Proceso, Intención, Edición

Abstract

Collaborative writing have always been one of the pillars of the internet. Selfless collaboration between the different users made possible the existence of communities whose only purpose is the diffusion of knowledge: the Wikis. This communities have been widely studied. However, the inner processes inherent to their activity are not fully known. In this project we propose and use a set of techniques known as Process Mining to discover and analyze the inherent processes in the task of collaborative writing in knowledge-based communities using the Spanish Wikipedia. The starting point is the 13-category taxonomy of semantic intentions behind the revisions of an article developed by Diyi Yang et al. Using their labeled article edits we create our own predictive model achieving a F1-Score micro of 0.66. Using this model a labeled corpus is composed out of different *featured articles*. With this corpus different techniques of Social mining and process mining are applied. Specifically we discover the collaboration structure of the users, the processes followed by the article and also the processes followed by the users in their edit sessions. The results show that the collaboration is sometimes organized and follow a certain structure. Furthermore we explicitly observe behavioral patterns in the editors that verify the findings of existing studies on this topic.

Keywords: Process mining, Social mining, Machine learning, ProM, Petri Net, Wikia, Wiki, Collaboration, Process, Intention, Edition

Índice general

Índice general	v
Índice de figuras	vii
Índice de cuadros	1
1. Introducción	2
1.1. Motivación	2
1.2. Objetivos	2
1.3. Metodología	3
1.4. Estructura	5
2. Introduction	6
2.1. Motivation	6
2.2. Objectives	6
2.3. Methodology	6
2.4. Structure	9
3. Tecnologías	10
4. ¿Qué es la Minería de Procesos?	13
4.1. Punto de comienzo: el formato XES	15
4.2. Descubriendo los procesos	16
4.3. Redes de Petri	17
4.4. ProM tools	18
4.5. De la minería de procesos a la minería social	19
5. Desarrollo previo	22
5.1. Descarga de datos	22
5.2. Extracción de información	22
5.3. Obtención de características	23
5.4. Cambio de formato	23
5.5. Generación y análisis de modelos predictivos	24
5.5.1. Datos iniciales	24
5.5.2. Métricas de evaluación	27
5.5.3. Clasificación binaria	27
5.5.4. Clasificación multilabel	39

ÍNDICE GENERAL

5.5.5. Conclusiones	46
6. Minería de Procesos	49
6.1. Obtención de datos	50
6.2. Transformación a XES	53
6.3. Análisis exploratorio de los datos	53
6.4. Análisis a nivel artículo	58
6.4.1. Análisis de la red de Petri descompuesta 1	61
6.4.2. Análisis de la red de Petri descompuesta 2	62
6.4.3. Análisis del conjunto de mini redes de Petri descompuestas 3	63
6.5. Análisis a nivel editor	71
6.5.1. Editores de actividad baja	71
6.5.2. Editores de actividad intermedia	75
6.5.3. Editores de actividad alta	80
7. Minería Social	86
7.1. Obtención de datos	88
7.2. Handover of Work	89
7.2.1. Artículo Tierra	89
7.2.2. Artículo Ácido desoxirribonucleico	91
7.3. Subcontracting	92
7.3.1. Artículo Tierra	92
7.3.2. Artículo Ácido desoxirribonucleico	94
8. Conclusiones	96
8.1. Conclusiones Minería de Procesos a nivel artículo	96
8.2. Conclusiones Minería de Procesos a nivel editor	96
8.3. Conclusiones Minería Social	97
8.4. Conclusiones globales	98
9. Conclusions	100
9.1. Process mining applied to the article: conclusions	100
9.2. Process mining applied to the editor: conclusions	100
9.3. Social mining conclusions	101
9.4. Global conclusions	102
10. Future Work	104
11. Código	105
Appendix	106
A. Scripts	107
A.1. wiki_dump_downloader.py	107
A.2. revision_id_extractor.py	113
A.3. corpus_filter.py	117

Índice de figuras

4.1. Estructura de la minería de procesos (fuente: [5])	13
4.2. Ejemplo de log de eventos	14
4.3. Diagrama de la estructura del formato XES (fuente: http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf)	15
4.4. Ejemplo de evento en XML	16
4.5. Descubrimiento de los procesos dentro de un log de eventos	16
4.6. Ejemplo de una red de petri	17
4.7. Interfaz de ProM tools 6.8	18
4.8. Visor de grafos de ProM tools 6.8	19
4.9. Estructura de la minería social en ProM	20
5.1. Distribución de intenciones en el conjunto de datos	25
5.2. Revisiones con múltiples intenciones	25
5.3. Gráfica de apariciones de revisiones con múltiples intenciones	26
5.4. Distribución de las intenciones en porcentajes	26
5.5. Proporción de positivos y negativos por intención	28
5.6. Resultados por intención del DummyClassifier binario	29
5.7. Matriz de confusión del clasificador Dummy binario	30
5.8. Resultados por intención de default Random Forest binario	31
5.9. Matriz de confusión de default Random Forest binario	32
5.10. Resultados por intención de default SVC	33
5.11. Matriz de confusión de SVC	34
5.12. Resultados Random Forest binario tras Over-Sampling	35
5.13. Matriz de confusión tras realizar over-sampling en el Random Forest binario	36
5.14. Resultados con Feature Engineering y Over-Sampling en Random Forest binario	36
5.15. Matriz de confusión con Feature Engineering y Over-Sampling del Random Forest binario	37
5.16. Resultados del ajuste de hiperparámetros del Random Forest binario	38
5.17. Resultados del ajuste de hiperparámetros del Random Forest binario	38
5.18. Matriz de confusión de Dummy Classifier multilabel	41
5.19. Matriz de confusión de default Random Forest multilabel	42
5.20. Matriz de confusión de default KNN	43
5.21. Resultados de los clasificadores de base multilabel	43
5.22. Matriz de confusión de Random Forest con Feature Engineering multilabel	44
5.23. Comparación de resultados de base con RF y Feature Engineering multilabel	45
5.24. Matriz de confusión de Random Forest con parámetros ajustados multilabel	46

ÍNDICE DE FIGURAS

5.25. Matriz de confusión de MLKNN con parámetros ajustados	47
5.26. Resultados de modelos multilabel finales	47
5.27. Gráfica con resultados de modelos finales	48
6.1. Proceso seguido para la descarga y preparación de los datos para realizar minería de procesos	50
6.2. Esquema del filtrado del corpus para el análisis a nivel editor	52
6.3. Representación de los atributos del formato CSV tras la conversión XES	53
6.4. Corpus en formato XES	54
6.5. Revisiones a lo largo del tiempo por artículo	55
6.6. Intenciones a lo largo del tiempo en el corpus	56
6.7. Representación de las distintas intenciones	57
6.8. Número de revisiones por editor	58
6.9. Número de editores según cantidad de ediciones realizadas	58
6.10. Petri net obtenida con Minero Heurístico	60
6.11. Petri net del proceso de edición tras su descomposición 1	61
6.12. Petri net del proceso de edición tras su descomposición 2	64
6.13. Redes de petri del proceso de edición tras su descomposición 3	65
6.14. Inicio de la red de petri del proceso de edición tras su descomposición 1	66
6.15. Sección intermedia de la red de petri del proceso de edición tras su descomposición 1	67
6.16. Final de la red de petri del proceso de edición tras su descomposición 1	68
6.17. Inicio de la red de petri del proceso de edición tras su descomposición 2	69
6.18. Final de la red de petri del proceso de edición tras su descomposición 2	70
6.19. 1º Petri net del proceso seguido por los editores de baja actividad	72
6.20. 2º Petri net del proceso seguido por los editores de baja actividad	72
6.21. 3º Petri net del proceso seguido por los editores de actividad baja	73
6.22. 4º Petri net del proceso seguido por los editores de actividad baja	74
6.23. Sección inferior de la 4º Petri net del proceso seguido por los editores de actividad baja	75
6.24. Sección superior de la 4º Petri net del proceso seguido por los editores de actividad baja	76
6.25. 5º Petri net del proceso seguido por los editores de actividad baja	76
6.26. 1º Petri net del proceso seguido por los editores de actividad intermedia	77
6.27. 2º Petri net del proceso seguido por los editores de actividad intermedia	78
6.28. 3º Petri net del proceso seguido por los editores de actividad intermedia	79
6.29. 4º Petri net del proceso seguido por los editores de actividad intermedia	80
6.30. 1º Petri net del proceso seguido por los editores de actividad alta	83
6.31. 2º Petri net del proceso seguido por los editores de actividad alta	84
6.32. 3º Petri net del proceso seguido por los editores de actividad alta	85
7.1. Estructura de la minería social en ProM	87
7.2. Proceso a seguir para la obtención de los datos usados en la minería social	88
7.3. Grafo handover of work del artículo Tierra	89
7.4. Zoom grupo central del grafo handover of work del artículo Tierra	90
7.5. Grafo handover of work del artículo Ácido desoxirribonucleico	91

ÍNDICE DE FIGURAS

7.6. Zoom grupo central del grafo handover of work del artículo Ácido desoxiribonucleico	92
7.7. Grafo subcontracting del artículo Tierra	93
7.8. Zoom grupo central del grafo subcontracting del artículo Tierra	93
7.9. Grafo subcontracting del artículo Ácido desoxirribonucleico	94
7.10. Zoom grupo central del grafo Subcontracting del artículo Ácido desoxirribonucleico	95

Índice de cuadros

1.1. Intenciones según la taxonomía usada y su descripción (fuente: [21])	4
2.1. Taxonomy of edit intentions in Wikipedia revisions (Source: [21])	8
5.1. Vistazo general de los datos	24
5.2. Resultados micro-averaged de los 3 clasificadores binarios de base	32
5.3. Resultados micro averaged de over-sampling en el Random Forest binario . .	34
5.4. Resultados micro averaged de over-sampling y feature engineering en el Ran- dom Forest binario	35
5.5. Resultados de los diferentes Random Forest binarios creados	39
5.6. Resultados del dummyClassifier multilabel	40
5.7. Resultados del default Random Forest multilabel	40
5.8. Resultados de default MLKNN	41
5.9. Resultos micro-averaged de los modelos finales ajustados de cada enfoque . .	47

Capítulo 1

Introducción

La colaboración es uno de los pilares de internet. Desde su inicio, su objetivo era poder eliminar las barreras existentes en la comunicación. Así, en cuestión de unas decadas, el volumen de datos existente en la red ha aumentado exponencialmente llegando a influir en numerosos aspectos vidas diarias. Uno de sus logros más destacables es la facilidad de la difusión de la información, haciendola más accesible y abundante que nunca. Esto ha dado lugar a la creación de las Wikis: páginas de conocimiento basadas en la escritura colaborativa. A día de hoy existen miles de Wikis diferentes pero de entre todas destaca Wikipedia, la enciclopedia libre más grande del mundo con 5.853.387 artículos, 36.262.835 editores y más de 18 mil millones de visitas anuales poniendo el conocimiento a disponibilidad del mundo entero.

1.1. Motivación

A pesar de que las Wikis son usadas por una cantidad enorme de usuarios cada día y han sido ampliamente estudiadas (especialmente Wikipedia) los procesos inherentes a su propio funcionamiento no son tan conocidos ni estudiados. ¿Existe algún proceso determinado que siguen los artículos durante su desarrollo?. ¿Qué proceso, en caso de existir uno, siguen los usuarios durante su historial de ediciones?. ¿De qué manera colaboran entre sí los usuarios?

Para poder responder a estas preguntas, se va a aplicar una metodología que aporta un nuevo punto de vista: La Minería de Procesos.

1.2. Objetivos

Los objetivos de este proyecto son por tanto, tratar de responder a las preguntas previamente formuladas. Mediante la minería de procesos, se tratará de descubrir los procesos internos seguidos por un conjunto seleccionado de *artículos destacados*. Los *artículos destacados* son aquellos que han sido catalogados como referente de calidad, es decir, de 'los mejores artículos de Wikipedia'. En la Wikipedia Española, esto supone sólo un total de 1127, el 0.07 % del total. Además se descubrirán también los procesos seguidos por los propios usuarios en su comportamiento habitual de edición y las estructuras de colaboración que puedan formarse entre los mismos.

1.3. Metodología

Una de las ventajas que ofrecen las Wikis, es la existencia de un historial de revisiones sufridas por cada artículo desde su creación hasta la actualidad. Partiendo de la base del uso de la Wikipedia Española, haremos uso de esta función de las Wikis para obtener el historial de ediciones de un conjunto de artículos seleccionados a mano de entre el conjunto de los *artículos destacados*.

La metodología a seguir para lograr responder a los objetivos propuestos se basará en la investigación realizada por Diyi Yang et al. donde desarrollan una taxonomía de intenciones semánticas tras cada revisión de un artículo basada en 13 intenciones. Así, haremos uso de su conjunto de revisiones etiquetadas con sus respectivas intenciones para crear un modelo predictivo que pueda determinar automáticamente las intenciones de un conjunto de revisiones cualquiera. En este caso, el objetivo es etiquetar un corpus de 8 *artículos destacados* aleatorios de Wikipedia de distintas temática. Con estas revisiones etiquetadas por intención semántica podemos hacer uso de la minería de procesos para vislumbrar el proceso que siguen los propios artículos en base a esta taxonomía así como el seguido por los propios usuarios en sus sesiones individuales de edición. Además, se usará minería social para determinar las relaciones entre los propios usuarios y lograr una perspectiva más amplia que nos permitirá responder a las preguntas anteriormente planteadas.

CAPÍTULO 1. INTRODUCCIÓN

Intención	Descripción
Clarification	Especificar o explicar un hecho existente o un significado mediante un ejemplo o discusión sin añadir información nueva
Copy editing	Editar oraciones, mejorar gramática o sintaxis.
Counter-Vandalism	Eliminar vandalismo
Disambiguation	Actualizar el enlace de una página desambiguada a una específica
Elaboration	Extiende/Añade una cantidad substancial de contenido nuevo. Añade hechos o nuevos datos importantes.
Fact update	Actualiza datos, fechas, puntuaciones, estado, etc... basándose en nueva información disponible
Point of view	Re-escribir utilizando un tono neutral propio de enciclopedias. Elimina valoraciones de carácter personal
Process	Comenzar/Continuar un flujo de trabajo en la Wiki como marcar un artículo con noticias references a su limpieza, borrado, etc.
Refactoring	Re-estructurar el artículo. Mover y re-escribir contenido sin cambiar el significado del mismo
Simplification	Reduce la complejidad del artículo; puede eliminar contenido.
Vandalism	Intenta deliberadamente dañar el artículo
Verification	Añade/modifica referencias/citaciones; elimina texto que no pueda ser verificado
Wikification	Cuestiones de formato para cumplir con las guías de estilo
Other	Ninguna de las anteriores

Cuadro 1.1: Intenciones según la taxonomía usada y su descripción (fuente: [21])

1.4. Estructura

La estructura del documento es la siguiente:

1. Introducción: La introducción plantea el problema inicial, la motivación para resolverlo y el proceso seguido para ello. Este capítulo además se encuentra tanto en inglés como en español.
2. Tecnologías: Este capítulo se centra en la explicación de las diferentes tecnologías usadas a lo largo del proyecto.
3. ¿Qué es la Minería de procesos?: Explica en detalle en qué consiste la minería de procesos y qué utilidad tiene. Además se explica en qué consiste la minería social y su relación con la minería de procesos. Es decir, detalla el fundamento teórico del proyecto.
4. Desarrollo: Su objetivo es explicar todo el proceso seguido desde la descarga de los datos iniciales hasta la obtención de un conjunto de datos listo para ser analizado mediante la minería de procesos. Incluyendo por tanto la elaboración y uso de un modelo predictivo basando en la taxonomía de Diyi Yang et al.
5. Minería de procesos: Explica en detalle todo el trabajo realizado con técnicas de minería de procesos desde dos perspectivas diferentes: usuario y artículo para obtener información acerca del proceso seguido por los artículos en su evolución y el seguido por los editores en sus sesiones de edición.
6. Minería social: En este capítulo comentamos en detalle los resultados de aplicar un análisis de minería social para descubrir las posibles colaboraciones entre los editores de Wikipedia usando un variado conjunto de datos y diferentes métricas.
7. Conclusiones: Clara y breve descripción de todos los hallazgos hechos a lo largo del proyecto con la aplicación de la minería social y de procesos.
8. Future Work: Capítulo que habla del trabajo que aún no ha sido realizado, los motivos de esto y posibles mejores que se puedan añadir.
9. Código: Qué scripts han sido utilizados durante el proyecto así como dónde se encuentran localizados y su autoría.

Capítulo 2

Introduction

Collaboration is one of the pillars of the Internet. Since the beginning of the internet its objective was to eliminate the existing communication barriers. In just a couple of decades the volume of data existing on the net has grown exponentially reaching a point of direct influence on our daily lives. One of Internet's biggest achievement is the diffusion capacity making knowledge more available and free than ever before. The peak of knowledge divulgation comes from the hand of the Wikis: encyclopedias based on collaborative writing. There are thousands of different Wikis, however, the most relevant one is Wikipedia, the biggest free encyclopedia in the work with 5.853.387 of articles, 36.262.835 editors and more than 18 billions of annual visits.

2.1. Motivation

Even though Wikis are used by a huge amount of people daily and have been deeply studied the processes inherent to its own structure and functioning are not fully known. Is there any particular process followed by the articles during its lifespan? Do users follow specific process of editing? How do they collaborate with each other?

To answer those questions a methodology that brings a breeze of fresh air in this topic will be used: Process Mining.

2.2. Objectives

The goal of this project is answering the questions suggested above. Through Process Mining we will try to discover the inner processes followed by a randomly selected corpus of *featured articles*. *Featured articles* are the top quality articles in Wikipedia. In the Spanish Wikipedia *featured articles* amount to 0.07 % of the total of articles: only 1127 articles belong to this category. Also the processes followed by the uses will be discovered as well as the collaboration structures that may happen naturally within the editors these communities.

2.3. Methodology

One of the advantages of studying a Wiki is the existence of a history of revisions for each article. Using the Spanish's Wikipedia as a reference we will make use of this function

to obtain the history of edits of a corpus of hand-picked *featured articles*. These articles are known for its high quality, presumably, the highest achievable in Wikipedia.

The methodology followed to answer the questions is based on the research done by Diyi Yang et al. She and her team developed a 13-category taxonomy of semantic intentions behind each revision. Using their own data set of labeled revisions we will create a predictive model able to automatically assign the intention behind each revision of our corpus with a micro averaged f1-Score of 0.66. Our corpus is composed by 8 random articles of different themes out of all the existing *featured articles* available at the Spanish's Wikipedia. With this labeled revisions we are able to make use of the process mining techniques to discover the work flow followed by the articles as well as the processes (if existent) of the users in their edit sessions. Furthermore, social mining will be used to unveil the collaboration structures formed in the community to have a bigger, more complete perspective that will allow us to meet our goals.

Intention	Description
Clarification	Specify or explain an existing fact or meaning by example or discussion without adding new information.
Copy editing	Rephrase; improve grammar, spelling, tone, or punctuation.
Counter-Vandalism	Revert or otherwise; remove vandalism.
Disambiguation	Relink from a disambiguation page to a specific page.
Elaboration	Extend/add substantive new content; insert a fact or new meaningful assertion.
Fact update	Update numbers, dates, scores, episodes, status, etc. based on newly available information.
Point of view	Rewrite using encyclopedic, neutral tone; remove bias; apply due weight.
Process	Start/continue a wiki process workflow such as tagging an article with cleanup, merge or deletion notices.
Refactoring	Restructure the article; move and rewrite content, without changing the meaning of it.
Simplification	Reduce the complexity or breadth of discussion; may remove information.
Vandalism	Deliberately attempt to damage the article.
Verification	Add/modify references/citations; remove unverified text.
Wikification	Format text to meet style guidelines, e.g. add links or remove them where necessary.
Other	None of the above.

Cuadro 2.1: Taxonomy of edit intentions in Wikipedia revisions (Source: [21])

2.4. Structure

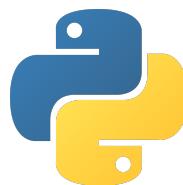
The structure of this document is as follows:

- Introduction: The introduction states the initial problem, motivation to solve it and the approach used for solving it.
- Technologies: This chapter focus on the explanation of the different technologies used throughout the project.
- What is process mining?: This chapter is centered around an explanation of what is process mining and how it does work. Also, its relation with social mining will be explained along with its introduction. In short, this chapter will lay out all the theoretical foundation of this project.
- Development: Focus on all the work to get a corpus ready for Process Mining, from the download of the initial data to the elaboration of the model to predict the intentions based on the 13-category taxonomy.
- Process mining: Explains in detail all the work done with the process mining techniques regarding this project from two perspectives: editor and article.
- Social mining: This chapter describe the results of analyzing social mining to unveil the inherent collaborations between the different editors using a rich set of data and different social metrics.
- Conclusions: Clear and brief explanation of all the finding done throughout this project regarding the process mining task.
- Future Work: Work that hasn't been done yet as well as possible improvements for the actual work flow.
- Code: Chapter explaining the different code used as well as its location and authorship.

Capítulo 3

Tecnologías

A continuación, se establecen las tecnologías usadas en este proyecto así como la motivación tras su elección.



Python ha sido escogido como lenguaje de programación para los diferentes scripts necesarios a lo largo del proyecto por su facilidad de uso y la enorme cantidad de librerías y recursos existentes para manejo de datos así como para técnicas de minería de datos.

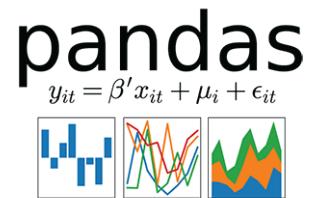


Anaconda ha sido escogido para servir de base para hacer uso de Python y manejar las diferentes librerías por su simplicidad y facilidad a la hora de gestionar las librerías y versiones de python así como sus múltiples utilidades como la opción de hacer uso de Jupyter.



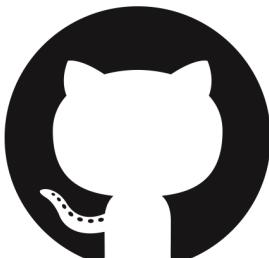
Para realizar tareas aprendizaje automático se ha utilizado Jupyter en conjunto con Anaconda. Los Jupyter Notebook dan lugar a un entorno de programación donde además se puede añadir texto generando un código interactivo ejecutable sección a sección aumentando mucho la legibilidad del código y favoreciendo la posibilidad de realizar un análisis al mismo tiempo que se programa.

Por otro lado, se ha hecho uso en los diferentes scripts del proyecto (11) de las librerías



matplotlib para las visualizaciones, pandas para el manejo de datos así como Numpy y Scikit-learn para aplicar machine learning. Estas 4 librerías, dan un vuelco a python y hacen del machine learning y el tratamiento de datos una tarea fácil.

- Matplotlib permite generar gráficas de alta calidad con tan sólo unas líneas de código, simplificando en gran parte la tarea.
- Numpy se trata de un paquete fundamental de python para la computación científica como es este proyecto. Aporta un tipo de arrays N-dimensionales con muchas funcionalidades además de contar con muchas funciones de álgebra lineal útiles entre otras cosas.
- Pandas es una librería centrada en el manejo de datos. Se encuentra construida sobre Numpy y Matplotlib y hace de las tareas de visualización y manipulación de datos una tarea fácil.
- Scikit-learn es la librería por excelencia para aplicar algoritmos de machine learning. Aporta herramientas simples y eficientes para aplicaciones de data mining y es open-source, desarrollada sobre NumPy, SciPy y matplotlib.



Para almacenar y organizar el código en un servicio externo se ha hecho uso de Github. Github aporta facilidades para realizar una gestión de versiones y mantenimiento del código además de salvaguarda en caso de necesitar revertir cambios.



ProM tools es la herramienta líder en minería de procesos. Es un framework que permite una gran variedad de técnicas de minería de procesos implementada en Java y disponible en www.processmining.org. Se ha utilizado en su versión 6.8 pues su target de audiencia es la investigación garantizando su estabilidad y ausencia de futuros cambios de modo que resultados ya publicados no se vean afectados por cambios en la herramienta.

Aunque no se menciona a lo largo del proyecto pues solo ha sido utilizado para guardar los datos, se ha hecho uso de una base de datos SQL de SQLite por su ligereza y facilidad de uso a la hora de importar archivos y realizar consultas rápidas.

Capítulo 4

¿Qué es la Minería de Procesos?

La minería de procesos se compone por un conjunto de técnicas de minería de datos que permiten extraer información de logs de eventos para descubrir, monitorizar y mejorar procesos[5]. Esta información puede ser analizada para tomar forma de decisiones de negocio o estratégicas para optimizar los procesos o simplemente para conocer el propio funcionamiento de los mismos. Sin embargo esto genera la siguiente pregunta ¿En qué consiste la minería de datos?

La minería de datos consiste en un conjunto de aprendizaje automático y estadística cuyo objetivo es el descubrimiento de patrones en grandes cantidades de datos[8]. Por lo que en este caso, la minería de procesos se basa en los mismos principios pero aplicado a un nivel organizativo mayor: el proceso en sí mismo.

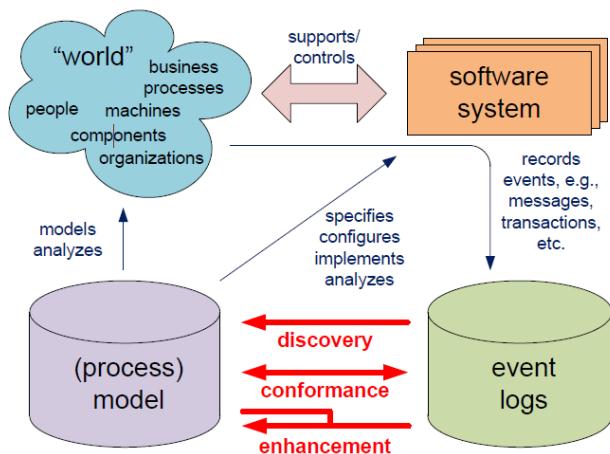


Figura 4.1: Estructura de la minería de procesos (fuente: [5])

Así, la minería de procesos tiene multitud de aplicaciones en diferentes áreas, pero principalmente sirve para unir la brecha entre la minería de datos y el Business Intelligence[5]. Algunos ejemplos de empresas que han aplicado la minería de procesos con notable éxito son

CAPÍTULO 4. ¿QUÉ ES LA MINERÍA DE PROCESOS?

Walmart o Vodafone tal y como expone Michal Rosik en [13]. Walmart aplicó la minería de procesos para descubrir y hallar ineeficiencias en el proceso de compra, hallando que el checkout no era lo eficiente que debería lo que les llevó a poder hallar estrategias para reducir el tiempo que los usuarios perdían en el proceso. Vodafone por otro lado ha conseguido aumentar el número de procesos que funcionan satisfactoriamente sin intervención humana en un 20% en tan solo dos años mediante el uso de minería de procesos según expone el artículo de Rosik.

Los requisitos de cara a poder realizar minería de procesos son pocos, sólo necesitamos un log de eventos. Un log de eventos no es más que un fichero de texto donde almacenamos información proveniente de bases de datos, transacciones... Es decir, son colecciones de secuencias de eventos.

	page_id	page_title	page_ns	revision_id	timestamp	contributor_id	org:resource	bytes	intentionality
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	3825	Leche	0	7778	2002-11-20T0...	1	AstroNomo	466	wikification
2	3825	Leche	0	7883	2002-11-20T0...	1	AstroNomo	497	
3	3825	Leche	0	22507	2002-11-24T0...	7	Maveric149	509	wikification
4	3825	Leche	0	38407	2003-08-09T1...	5	Andre Engels	574	copy-editing
5	3825	Leche	0	42101	2003-10-06T0...	2121	Moriell	592	wikification

Figura 4.2: Ejemplo de log de eventos

Un ejemplo de un conjunto de datos que podría ser considerado un log de eventos puede de ser observado en la figura 4.2. Como vemos se compone de diversos atributos habituales en ficheros de datos: id, timestamp, título... Sin embargo, para poder aplicar técnicas de minería de procesos es necesario un formato específico para almacenar los datos: el formato XES.

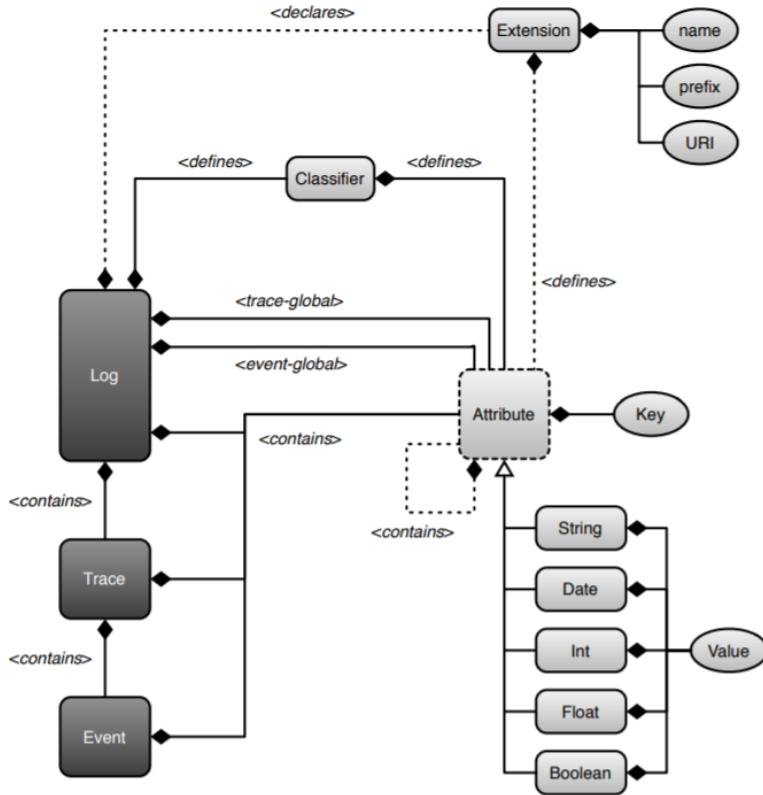


Figura 4.3: Diagrama de la estructura del formato XES (fuente: http://www.xes-standard.org/_media/xes_xes_standard_proposal.pdf)

4.1. Punto de comienzo: el formato XES

XES es el formato estándar de la IEEE Task Force en minería de procesos[5]. La estructura de estos archivos puede verse en el esquema 4.3. La base del formato XES y de los logs de eventos es asumir que es posible guardar secuencialmente eventos de modo que cada evento se refiera a una actividad y esté asociado a un caso particular (traza)[5]. Además, los logs de eventos pueden guardar otra información extra en forma de atributos como el actor del evento iniciando o finalizando la actividad, la marca de tiempo o timestamp o datos relativos al evento.

- Las trazas representan cada caso del proceso a analizar. Por ejemplo, en un log de eventos compuesto por las revisiones realizadas en diferentes artículos de Wikipedia, cada traza podría ser representada por cada artículo.
- Los eventos reflejan cada acción/actividad dentro de cada caso del proceso (traza). Continuando con el ejemplo anterior, los eventos estarían representados por cada revisión realizada a cada artículo. Dentro de un log de eventos, el atributo que determina cada evento toma el nombre de concept:name.
- De entre los posibles atributos que puedan tener una traza o evento es importante la existencia de una marca de tiempo o timestamp para cada evento. En el caso del ejemplo

anterior, el timestamp sería la hora a la que se realizó una revisión. En un log de eventos el atributo que representa el timestamp es llamado time:timestamp.

- Otro de los posibles atributos existentes en cada traza o evento es el actor que realiza el evento. Siguiendo la misma linea que anteriormente, esto estaría representado por el nombre o id del editor que realiza una revisión. En un log de eventos esto es denominado org:resource.

```

<event>
  <int key="page_id" value="45307"/>
  <string key="org:resource" value="|JorgeGG|"/>
  <string key="concept:name" value="wikification,elaboration"/>
  <int key="bytes" value="1323"/>
  <string key="lifecycle:transition" value="complete"/>
  <date key="time:timestamp" value="2004-05-08T01:01:44.000+02:00"/>
  <int key="page_ns" value="0"/>
  <int key="revision_id" value="145370"/>
  <string key="contributor_id" value="2906"/>
</event>
```

Figura 4.4: Ejemplo de evento en XML

La figura 4.4 representa en XML el aspecto que tendría un evento dentro de un log de eventos. Como vemos, nos encontramos con org:resource, concept:name y time:timestamp además de otros atributos adicionales. Lifecycle:transition representa si el evento está comenzando o finalizando, en el caso del ejemplo, finalizando pues indica que ha sido completado.

4.2. Descubriendo los procesos



Figura 4.5: Descubrimiento de los procesos dentro de un log de eventos

La principal técnica y la más utilizada dentro de la minería procesos es el descubrimiento. El descubrimiento consiste en la generación de un modelo que represente los procesos existentes dentro del log de eventos. Las técnicas de descubrimiento parten de un log de eventos para generar un modelo sin ninguna información a priori[5]. Estas técnicas se basan en diferentes algoritmos de minería que hacen uso de estadística y aprendizaje automático para buscar estos patrones existentes en los datos que representan un proceso.

Dentro de la multitud de algoritmos de minería existentes dentro de la minería de procesos destacan:

- Alpha Miner: Fue el primer algoritmo de minería de procesos en desarrollarse y como tal tiene ciertos problemas. Examina las relaciones entre los diferentes eventos generando un modelo donde cada transición representa una tarea observada.

- Minero heurístico: Se trata de una mejora respecto al algoritmo Alpha Miner. El minero heurístico se centra en el flujo de control considerando solo el orden de los eventos dentro de cada evento[19]. Por lo cual un log de eventos con timestamp es necesario de cada a hacer uso de este algoritmo. Además, tiene en cuenta las frecuencias de aparición pudiendo filtrar comportamiento infrecuente y permite saltarse las actividades individuales. Básicamente genera una matriz de directly-follow y la analiza teniendo en cuenta la anterior información.
- Minero inductivo: El minero inductivo garantiza la generación de modelos 'sound' que traducido de modo literal implica modelos 'buenos'. 'Soundness' es una característica que implica que todo el comportamiento observado en el log de eventos puede ser reproducido por el modelo generado. Para esto, el algoritmo genera un arbol que representa el proceso, lo cual logra diviendo en log del modo más optimo posible hasta generar el árbol/es. Sin embargo, esto puede dar lugar a modelos difíciles de interpretar. Al tratar de generar modelos que representen todo el comportamiento observable en el log de eventos como mínimo, el minero inductivo puede recurrir a modelos en forma de flor. Un modelo en forma de flor es aquel que permite cualquier tipo de comportamiento en base a un tipo dado de actividades.

Estos algoritmos generan una salida que representa los procesos existentes en el log de eventos mediante una red de petri.

4.3. Redes de Petri

Las redes de petri son un modelo abstracto y formal de mostrar un flujo de información[11]. Permiten mostrar el flujo que sigue un proceso de principio a fin representando así los procesos descubiertos en un log de eventos.

Definición de una red de petri: Una red de petri es una tupla $N = (P, T, F)$ donde P es el conjunto de lugares, T de transiciones, $P \cap T = \emptyset$ y $F \subseteq (PxT) \cup (TxP)$ la relación del flujo[2]

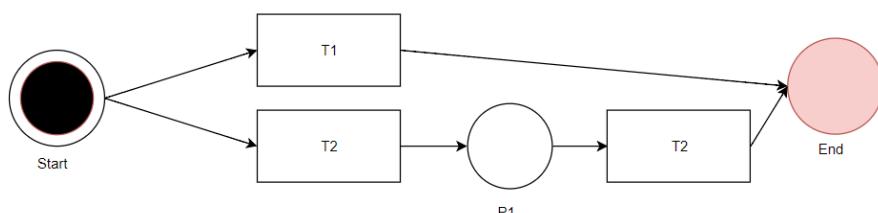


Figura 4.6: Ejemplo de una red de petri

Con esto en cuenta, en la imagen 4.6 podemos ver una red de petri (P, T, F) donde $P = \{Start, C1, end\}$, $T = \{T1, T2, T3\}$ y $F = \{(Start, T1), (Start, T2), (T2, C1), (C1, T3), (T3, End), (T1, end)\}$. Así, la red comienza en el lugar Start y finaliza en End pasando por las diferentes transiciones. El flujo podría ser o bien de Start a T1 y de ahí a End o bien de Start a T2 y hasta llegar

CAPÍTULO 4. ¿QUÉ ES LA MINERÍA DE PROCESOS?

a End para finalizar el proceso. Esto sería traducido en que el proceso representado por la red de la figura escenifica dos posibles caminos, o bien el proceso sigue el camino de la acción determinada en la transición T1 o bien sigue el camino compuesto por las transiciones T2 y T3 pero en ningún caso ambos caminos simultáneamente.

No obstante, las redes de petri generadas no suelen representar el posible proceso existente al 100 % a pesar de que el minero inductivo tiene esto como objetivo. Es por esto que existe una métrica llamada fitness la cual mide en qué medida el log de eventos puede ser reproducido correctamente en el modelo. Un valor de 0 implicaría que la red generada no representa en absoluto el log de eventos y de 1, que lo hace de modo perfecto.

4.4. ProM tools

Para aplicar estas técnicas de minería de procesos se hace uso de la herramienta ProM tools. ProM es la herramienta líder en minería de procesos. Se trata de un framework que permite una alta variedad de técnicas de minería de procesos implementado en Java y disponible en www.processmining.org.[9] La versión 6.8 ha sido la versión escogida en lugar de la última pues el target de esta versión es la investigación al garantizar que no se realizarán cambios que puedan afectar a los resultados que ya hayan sido publicados u obtenidos.[12].

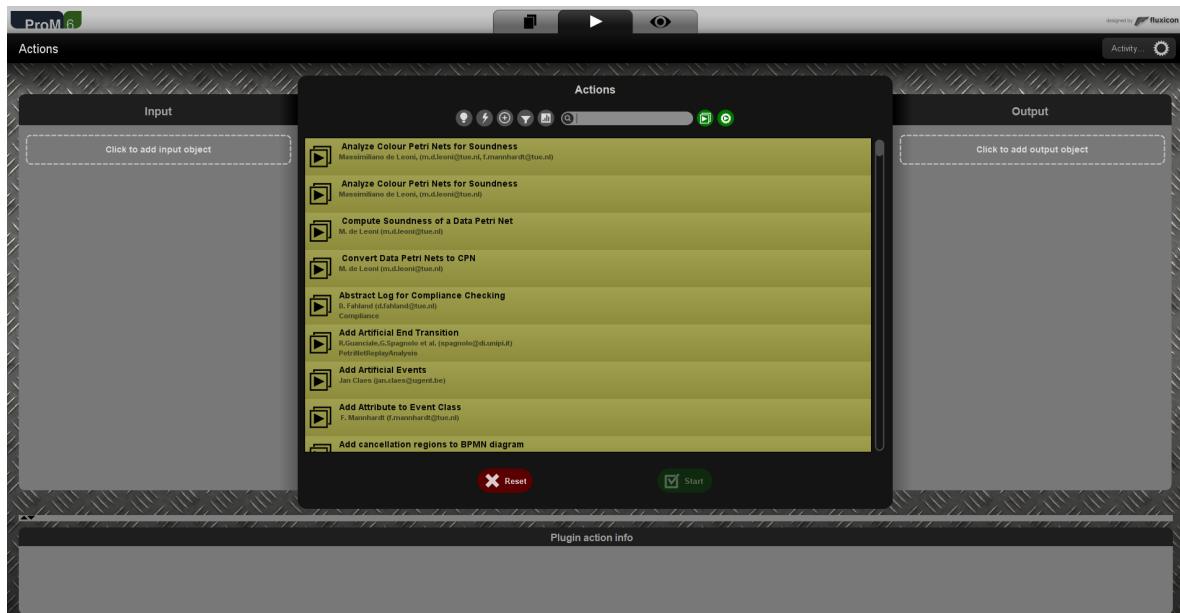


Figura 4.7: Interfaz de ProM tools 6.8

ProM es un programa de sencillo manejo con una interfaz bastante básica, tal y como vemos en la imagen 4.7 nos encontramos con una pestaña izquierda donde seleccionamos los datos de entrada, una pestaña central donde seleccionamos la técnica o algoritmo que queremos aplicar y la pestaña derecha donde aparecerán los ficheros de salida tras aplicar la técnica/algoritmo seleccionado. Nos permite tanto importar como exportar ficheros para poder utilizarlos en otro momento y cuenta con un visor integrado para poder ver gráficamente las redes de petri generadas. Sin embargo este visor cuenta con limitaciones como limitadas

opciones de organización y en el caso de la visualización de grafos no poder regular el tamaño y grosor de los nodos y aristas respectivamente manualmente.

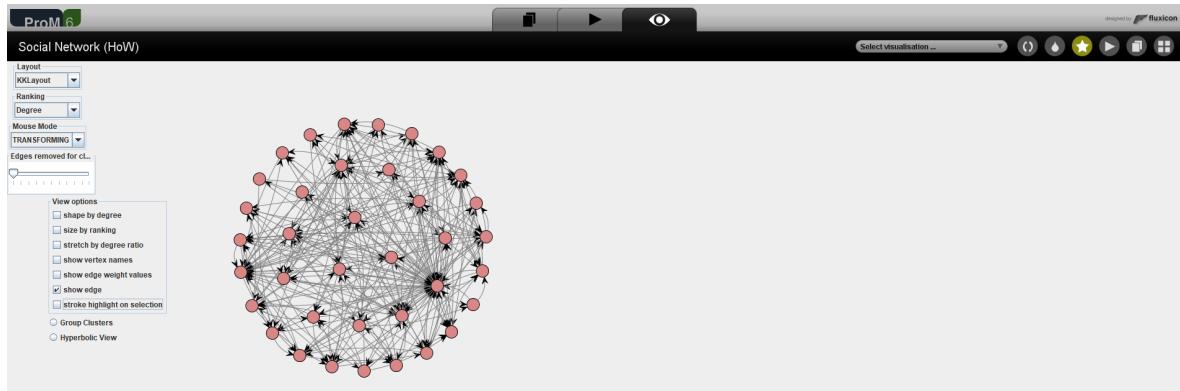


Figura 4.8: Visor de grafos de ProM tools 6.8

No obstante, técnicas de minería de procesos no es lo único que ofrece ProM, también ofrece técnicas de minería social.

4.5. De la minería de procesos a la minería social

Para complementar un análisis realizado mediante minería de procesos para descubrir los procesos existentes dentro de un log de eventos, se pueden aplicar técnicas de minería social para descubrir también las relaciones existentes entre los diferentes resources del log.

Las técnicas de minería social hacen uso de técnicas de sociometría y de análisis de redes sociales[4]. La utilidad que tienen son la posibilidad de observar y conocer las posibles relaciones existentes entre los diferentes actores (org:resource) que aparecen a lo largo de un log de eventos. Estos algoritmos son implementados en ProM mediante la librería basada en las métricas establecidas por Wil M.P. van der Aalst et al en [3]. Representan las relaciones existentes en forma de grafo donde cada resource es un nodo y cada arista una relación, dependiendo el peso de la intensidad de esta relación y hacen uso de los mismos ficheros de datos que la minería de procesos: log de eventos.

Se pueden hacer uso de diferentes métricas para obtener las relaciones entre los diferentes autores en un log:

1. Handover of Work: Se define como Handover Of Work como el traspaso de trabajo de un individuo i a un individuo j si hay dos actividades subsecuentes entre ellos dentro de un log de eventos.[3] Es decir, si después de editar i edita j se establece una relación entre ellos. Estos individuos i y j son representados mediante nodos y su conexión mediante aristas, variando el peso en función de lo fuerte que sea la relación entre ellos.
2. Subcontracting: Subcontracting cuenta el número de veces que un individuo j ejecuta una actividad entre dos actividades ejecutadas por el individuo i[3]. Es decir, si i edita, j edita i vuelve a editar, se establece una relación de i a j. Así, siendo i y j representado

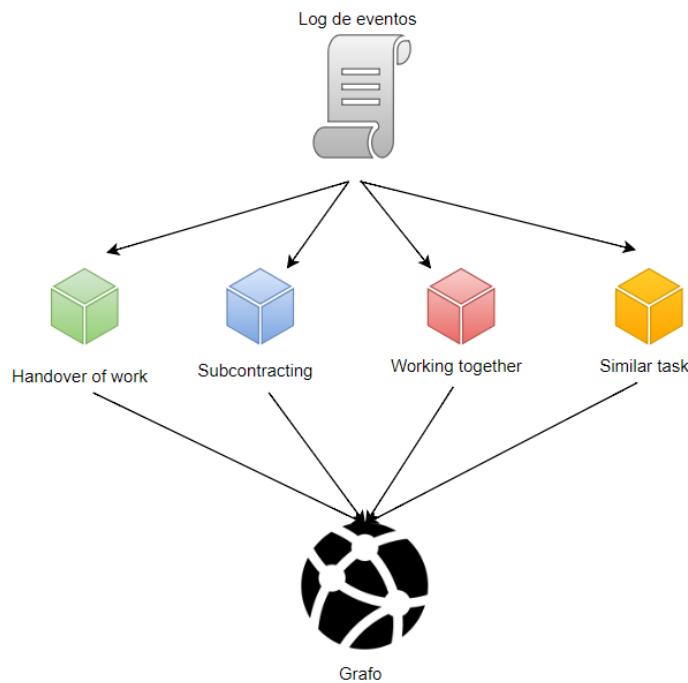


Figura 4.9: Estructura de la minería social en ProM

como nodos su relación se establece mediante una arista común cuyo peso dependerá de las veces que suceda la relación.

3. Working-Together: Working Together simplemente tiene en cuenta la frecuencia con la cual dos resources realizan actividades en el mismo caso (traza)[3] dentro del log de eventos. De este modo, dos resources que trabajen en el mismo caso estarán relacionados y el peso de su arista dependerá de la frecuencia con la que suceda siendo representados por nodos.
4. Similar Task: Similar Task se centra en el tipo de actividades que realizan los diferentes resources. Así, se establecerá una relación entre resources que estén realizando un tipo similar de actividades, el peso de su arista al representar los resources mediante nodos dependerá de la similitud de las tareas realizadas. La similitud entre las tareas realizadas por los diferentes resources se puede calcular mediante 4 métricas diferentes:
 - Distancia Euclidea: mide la distancia en el espacio entre 2 puntos.
 - Pearson Correlation Coefficient: es una métrica de correlación lineal utilizada frecuentemente para encontrar relaciones entre casos.
 - Jaccard Similarity Coefficient: compara los diferentes grupos posibles y mide que miembros son compartidos entre los grupos y que miembros no.
 - Hamming distance por otro lado, comprueba el número mínimo de sustituciones requerido para transformar un string en otro.

Con toda esta información la base teórica necesaria para aplicar la minería social y de procesos es suficiente. De esta manera en los próximos capítulos se definirá y aplicará la

CAPÍTULO 4. ¿QUÉ ES LA MINERÍA DE PROCESOS?

minería social y de procesos contextualizandola dentro de los objetivos de este proyecto.

Capítulo 5

Desarrollo previo

Durante este capítulo, se detallará el proceso seguido desde la descarga del historial de revisiones de un artículo en Wikipedia hasta la generación de los datos a analizar mediante la minería de procesos. Así, el proceso consta de un número reducido de pasos donde se descargan los datos, se extrae la información de los mismos y se finaliza con la predicción las intenciones de cada revisión disponible.

5.1. Descarga de datos

El primer paso de todos es la descarga de los datos iniciales. Estos datos, pueden ser obtenidos de Wikipedia ya sea en forma de un artículo específico o un conjunto de artículos en un archivo de texto plano. Para esto, se hace uso del script `wiki_dump_downloader.py` (11). Este script ha sido desarrollado en base a https://phabricator.wikimedia.org/diffusion/PWBC/browse/master/scripts/maintenance/download_dump.py. La diferencia principal con el código en el cual el script está basado es la eliminación de las dependencias con la librería PyWiki además de la adición de numerosas utilidades como la selección del idioma de la Wiki escogida, la posibilidad de pasar por parámetro una lista de artículos a descargar y la unión de las diferentes partes descargadas (pues los artículos son descargados separados en diferentes fragmentos de menor tamaño) en una sola.

A priori, este script sirve para descargar el historial de revisiones de un artículo específico o de una lista de artículos en una Wikipedia de un lenguaje específico (e.j Wikipedia española). El archivo descargado tendrá formato XML y será un archivo de elevado peso. Es por esto, que se necesita convertir el formato del archivo en otro formato más facil de manejar, lo que conduce al siguiente paso de este proceso: la extracción de información de estos archivos XML obtenidos.

5.2. Extracción de información

Para extraer información del XML descargado se utiliza el parser desarrollado por Abel Serrano Juste akronix5@gmail.com `wiki_dump_parser.py` (11) como parte de un conjunto de scripts para Wikipedia. Este script recibe como entrada el conjunto histórico de revisiones de un artículo en formato XML y lo convierte en un CSV legible con información útil. Esta

información está compuesta por el id y título del artículo, timestamp, id y nombre del editor así como el conjunto de bytes afectados en la edición.

5.3. Obtención de características

Llegado a este punto, el objetivo es obtener las diferentes características de cada revisión en comparación con la revisión previa pues esto posibilita obtener información al respecto de lo sucedido durante la misma. Para ello, hacemos uso de la información disponible en la investigación realizada por Aaron Halfaker y Diyi Yang[21]. En la investigación, donde se intenta crear una taxonomía de intenciones existentes tras cada revisión, hacen uso de los resultados obtenidos mediante el uso del diff online de la API de Wikipedia a través de un conjunto de scripts realizados por ellos mismos (11). Sin embargo, ha habido que realizar modificaciones para conseguir adaptar su flujo de trabajo a este proyecto. En primer lugar, ha habido que actualizar librerías deprecadas y adaptar los parámetros de entrada para poder añadir lenguaje de la Wikipedia utilizada.

Gracias a esto, es posible hacer un diff entre las diferentes revisiones generando un fichero de salida que cuenta con 207 atributos. Estos 207 atributos se encuentran estructurados en 3 grupos. El primero se pone de atributos asociados al propio editor del artículo, el segundo se compone de 16 atributos en base al comentario escrito por el editor respecto a su revisión y el tercer grupo consta de los restantes 198 atributos y consiste de detalles del diff de Wikipedia[21]. La entrada de este conjunto de scripts requiere de un conjunto de parejas de id's de revisión e intenciones asociadas. Dado que no se cuenta con esas intenciones asociadas en nuevos artículos que se acaben de descargar, se asigna siempre un placeholder de 0 en la intención en este punto.

Así, una vez se tiene el fichero resultado del parser previamente mencionado¹, se utiliza el script llamado `revision_id_extractor.py` (11) cuya función es extraer el id de revisión de cada revisión de los artículos seleccionados generando un nuevo archivo compuesto por el id de revisión y una etiqueta de 0 asignado automáticamente como intención. Con este archivo obtenido tras hacer uso del script de extracción se llama al conjunto de scripts para generar los 207 atributos en base a las diferencias entre revisiones. Es importante mencionar que debido a que el diff de Wikipedia se realiza online en sus propios servidores, este proceso lleva un elevado número de horas para un conjunto grande de revisiones, siendo una limitación real de cara a analizar comunidades muy grandes.

5.4. Cambio de formato

Debido a que el output del diff utilizado tiene formato arff, es necesario realizar un cambio de formato a CSV por motivos de simplicidad de cara a futuras secciones. Para ello se hace uso del script `arffToCsv.py` (11). Se trata de un script ligero y simple que cumple con su función. El siguiente y último paso es la elaboración de un modelo predictivo.

¹`wiki_dump_parser.py` (11)

feats_0	feats_1	feat_2	feats_3	feats_4	feats_5	...	other	wikification	vandalism	simplification	elaboration	verifiability	process	clarification	disambiguation	point-of-view
741692138	0.0	0.0	-1.0	0.0	555.0	...	1	0	0	0	0	0	0	0	0	0
710764506	0.0	0.0	-1.0	0.0	3.0	...	1	0	0	0	0	0	0	0	0	0
711588802	0.0	0.0	-1.0	0.0	9.0	...	0	0	0	0	0	0	0	0	0	0
709526386	0.0	0.0	-1.0	0.0	326.0	...	0	0	0	0	0	0	0	0	0	0
713098731	0.0	0.0	-1.0	0.0	190.0	...	0	0	0	0	0	0	0	0	0	0

Cuadro 5.1: Vistazo general de los datos

5.5. Generación y análisis de modelos predictivos

El objetivo principal de esta subsección es explicar el proceso seguido en la generación de un modelo predictivo para asignar una o varias intenciones a cada revisión realizada al artículo / Wiki objeto del estudio en base a la taxonomía creada por Diyi Yang Y Aaron Halfaker[21]. Para esto, se cuenta con un conjunto de revisiones elaborado para la citada investigación que será utilizado como entrenamiento a la hora de determinar el mejor modelo posible. En la propia investigación realizada por ellos así como en su GitHub referenciado anteriormente, tienen también disponible un modelo predictivo. Sin embargo, debido a la dificultad para hacerlo funcionar y sus resultados no excesivamente buenos, se ha decidido tratar de realizar un modelo que mejore los resultados existentes en cuanto a predicción de intenciones basadas en esta taxonomía.

Con el objetivo de obtener un modelo predictivo que sea capaz de generar los resultados más precisos posibles se entrenarán diferentes modelos basados en variados algoritmos de clasificación tales como el Random Forest, Support Vector Machine o Nearest Neighbors. Además, se aplicarán diferentes técnicas para optimizar los datos tales como Feature Engineering, over-sampling o la normalización de los datos. Una vez creado el mejor modelo predictor posible, este es exportado de cara a automatizar el proceso para futuros artículos / Wikis que se quieran analizar.

5.5.1. Datos iniciales

Inicialmente contamos con el dataset generado durante el estudio realizado por Diyi Yang y Aaron Halfaker que cuenta con 7170 revisiones de artículos de Wikipedia con intenciones asignadas a mano[21].

El conjunto de revisiones, como se puede observar en la tabla 5.1 está formado por 208 atributos y 14 etiquetas binarias diferentes, una por cada intención.

Los atributos destacan por su difícil interpretabilidad. Son de tipo escalar y con nombres que no aportan ninguna información a priori. Sin embargo, se encuentran estructuradas en 3 grupos. El I grupo se compone de atributos asociados con el editor; el grupo II tiene 16 atributos asociados con el comentario escrito por el editor sobre la revisión y finalmente el grupo III consta de 198 atributos basados en el diff entre revisiones[21]. Además, para facilidad en el futuro uso de las predicciones generadas, el id de revisión ha sido añadido como feats_0. Además todos los valores de los atributos son correctos, no hay NULL ni espacios en blanco.

Una vez que se conoce el formato de los datos, el próximo paso es realizar un análisis exploratorio de los mismos.

En la figura 5.1 podemos ver la distribución de las intenciones en el dataset. Se observa

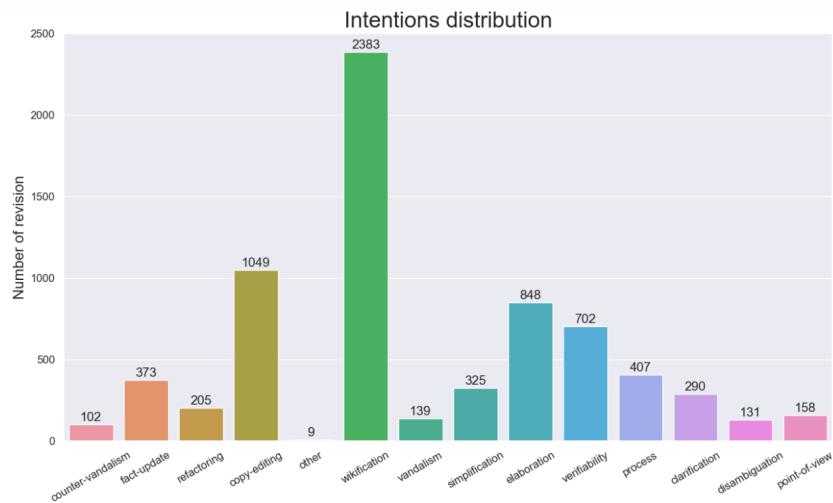


Figura 5.1: Distribución de intenciones en el conjunto de datos

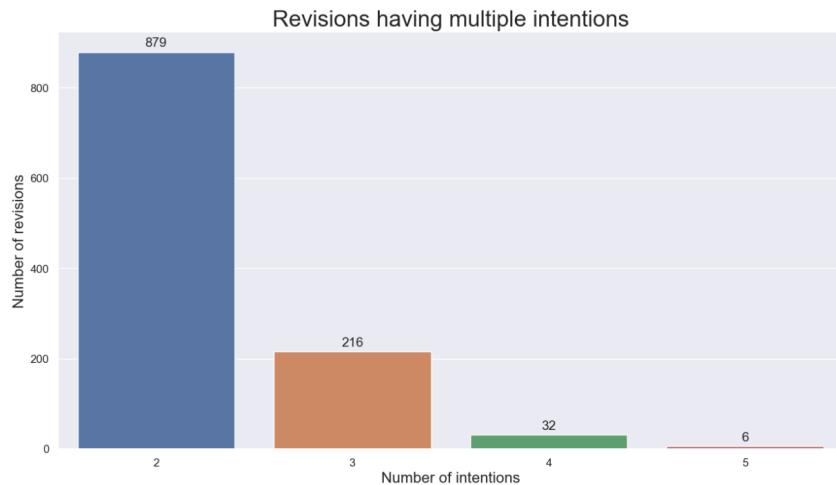


Figura 5.2: Revisiones con múltiples intenciones

que no hay un gran balance entre las diferentes intenciones. Más específicamente, en la figura 5.4 podemos observar estos mismos datos en porcentajes. Intenciones como Other, counter vandalism, disambiguation, point of view o vandalism se encuentran en proporciones inferiores al 3 % mientras que Wikification aparece en el 41 % de las revisiones. El caso más serio se da en la intencion Other. El objetivo de esta intencion es determinar que en esa revision se ha hecho algo diferente a todo lo demás, sin embargo dada su frecuencia de aparición podría considerarse incluso un outlier en si mismo y no ser considerado durante la predicción.

Por otro lado, en lo que a revisiones con multiples intenciones se refiere nos encontramos con que la figura 5.2 muestra una cantidad relativamente alta de ediciones con más de 1 intención. Sin embargo, según aumenta el número de intenciones por revisión, decrece exponencialmente la frecuencia con que sucede. En la figura 5.3 podemos ver como sólo un 0,1 % de las revisiones tienen 5 intenciones, alcanzando un 15 % revisiones con 2 intenciones. Así,

el 80 % de todas las revisiones del dataset solo poseen una intencionalidad.

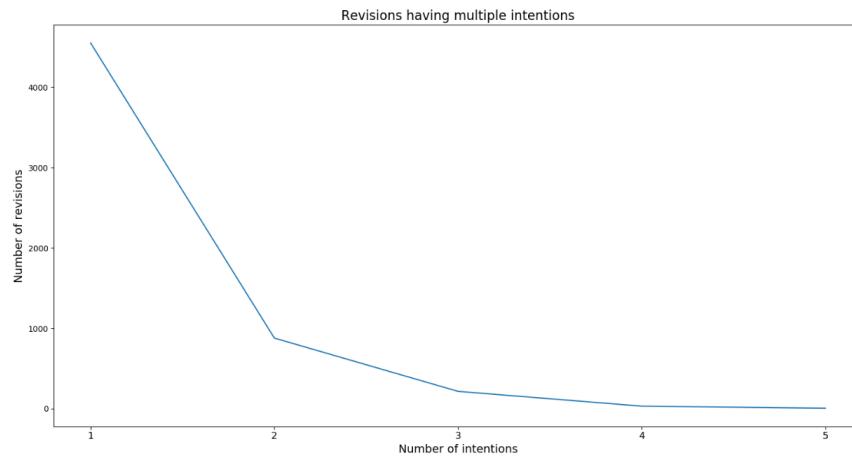


Figura 5.3: Gráfica de apariciones de revisiones con múltiples intenciones

	intention	count	% of appearance
0	counter-vandalism	102	1.792619
1	fact-update	373	6.555360
2	refactoring	205	3.602812
3	copy-editing	1049	18.435652
4	other	9	0.158172
5	wikification	2383	41.880492
6	vandalism	139	2.442882
7	simplification	325	5.711775
8	elaboration	948	14.903339
9	verifiability	702	12.337434
10	process	407	7.152900
11	clarification	290	5.096661
12	disambiguation	131	2.302285
13	point-of-view	158	2.776801

Figura 5.4: Distribución de las intenciones en porcentajes

En base a estos datos obtenidos, de cara a obtener el modelo predictor más optimo posible, se aplicarán 2 metodologías diferentes.

1. clasificación multilabel: Un solo target cuya label se compone de las diferentes intenciones posibles.
2. clasificación binaria: Cada intención es predecida por un modelo diferente, así, solo debe determinar si esa intención se encuentra en la revisión o no. En total se desarrollarían 14 modelos personalizados, uno para cada posible intención y se agruparían obteniendo un resultado final igual al obtenido mediante clasificación multilabel.

La motivación principal es tratar de ver como se comportan los modelos en cada metodología ante la multitud de atributos diferentes y la falta de balance. Así, en la clasificación binaria la falta de balance genera un desafío. Sin embargo al poder realizar un modelo a medida por intención se obtiene un grado mayor de flexibilidad. Por otro lado la clasificación

multilabel aunque más robusta ante la falta de balance lo limitado en tamaño del dataset generará un desafío en su precisión a la hora de catalogar una cantidad de etiquetas tan alta (14).

Esencialmente, el objetivo es realizar una clasificación donde se obtengan las posibles diferentes intenciones que suceden naturalmente en cada revisión. No obstante la diferencia en cada enfoque radica en la creación de un modelo a medida para cada intención o utilizar uno general que prediga todas las intenciones de una sola vez. De este modo, la idea de la clasificación binaria es agrupar la salida de cada modelo individual por intención de modo que el resultado final es el mismo con ambos enfoques, lo que varía es el proceso interno seguido hasta que este es conseguido.

Debido a que son enfoques diferentes, es importante fijar unas métricas de referencia que sirvan para poder establecer una comparación justa entre ambas metodologías.

5.5.2. Métricas de evaluación

De cara a hacer uso de las mismas métricas y garantizar su comparabilidad entre los diferentes enfoques, se han fijado unas métricas de base:

- Precisión: Representa el número de ejemplos positivos asignados correctamente clasificados dividido por el total de ejemplos positivos asignados clasificados con ese valor[17].

$$P = \frac{TP}{TP+FP}$$
- Recall: Representa el número de ejemplos positivos correctamente clasificados dividido entre el número de positivos en el conjunto de datos[17].

$$R = \frac{TP}{TP+FN}$$
- F1: Se trata de una combinación de Precisión y Recall.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$
- Matriz de confusión: Sirve para describir el rendimiento de un modelo predictivo mostrando visualmente los True Positive, True Negative, False Positive y False negative obtenidos.

Sin embargo, a priori estos datos no son comparables entre los dos enfoques. El motivo es que mientras que en el enfoque multilabel se obtienen los valores calculados mediante micro averaging (calculando la media teniendo en cuenta las proporciones de cada intención) entre todas las intenciones predecidas, en el enfoque binario se obtienen métricas independientes para cada modelo. Es por esto que se calcularán los valores medios entre todas las intenciones en el enfoque binario mediante micro averaging a partir de los valores obtenidos en su matriz de confusión. De este modo se obtienen valores directamente comparables con el enfoque multilabel y podemos realizar una comparación justa entre modelos binarios agrupados y el modelo multilabel.

5.5.3. Clasificación binaria

La clasificación binaria es aquella en la que la etiqueta toma dos valores posibles: 1 o 0. En este caso una etiqueta marcada con un 1 implicaría que esa revisión tiene esa intención. Así, existe la necesidad de generar 14 modelos diferentes, uno por intención y agrupar sus resultados de modo que se obtenga el conjunto de posibles intenciones que suceden en cada revisión.

Como consideraciones iniciales, se ha eliminado la intención 'Other' pues no es necesaria al no aportar información extra y se ha utilizado un random_state = 64 en todos los algoritmos para facilitar la reproducibilidad.

En lugar de hacer uso de conjuntos de datos diferentes para test y para train prefijados, se hace uso de cross-validation siguiendo la estrategia de stratifiedKFolds con k = 4 mediante GridSearch. Después, se copia el mejor modelo obtenido por grid search y se re-entrena haciendo uso de cross_val_predict de nuevo haciendo uso de cross validation con k = 4 para obtener la matriz de confusión y poder calcular los valores micro-averaged del conjunto de las intenciones.

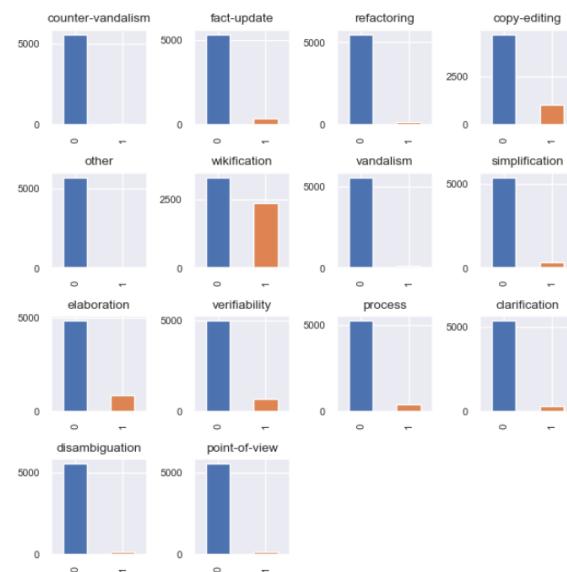


Figura 5.5: Proporcion de positivos y negativos por intencion

Así, el conjunto de datos ha de ser dividido por el número de intenciones diferentes. En este caso se generan 14 targets diferentes, uno por cada intención. De esta manera quedaría la proporción de positivos y negativos en cada intención observada en la figura 5.5

En base a toda la información obtenida, el proceso a la hora de determinar el modelo siguiente ha sido el siguiente:

1. Clasificador dummy para establecer métricas de base: mediante un clasificador que se base en reglas simples prefijadas, obtenemos unas métricas que establecen un rendimiento mínimo a superar por el resto de clasificadores.
2. Random Forest: Se aplicarán algoritmos de random forest que hace uso de n árboles de decisión para generar una predicción, reaccionando positivamente ante la falta de balance.
3. SVM con Kernel Lineal: Se aplicará un SVM con Kernel Lineal para tratar de ver si su funcionamiento es óptimo para el problema a solucionar.

4. Over-Sampling: Como forma de solucionar la falta de balance entre las diferentes intenciones entran las técnicas de over-sampling cuyo objetivo es equilibrar la frecuencia de las mismas.
5. Feature engineering: El objetivo aquí es aplicar técnicas de feature engineering, cuya función es la optimización del conjunto de datos, para determinar si generan un efecto positivo en los resultados de la clasificación.
6. Ajuste de hiperparámetros: Una vez todos los resultados de los puntos anteriores han sido estudiados se ajustarán los hiperparámetros del modelo seleccionado como mejor en busca de incrementar sus valores de precisión, recall y F1 aún más.

Clasificador Dummy

	Intention	Precision	Recall	F1
0	counter-vandalism	0.010864	0.009995	0.010411
1	fact-update	0.034535	0.029457	0.031793
2	refactoring	0.041667	0.039027	0.040303
3	copy-editing	0.185020	0.160165	0.171698
4	wikification	0.413157	0.395302	0.404032
5	vandalism	0.007577	0.007144	0.007354
6	simplification	0.052085	0.046147	0.048936
7	elaboration	0.129831	0.110846	0.119590
8	verifiability	0.147573	0.121075	0.133017
9	process	0.078488	0.066346	0.071908
10	clarification	0.034611	0.031055	0.032736
11	disambiguation	0.000000	0.000000	0.000000
12	point-of-view	0.013506	0.012814	0.013151

Figura 5.6: Resultados por intención del DummyClassifier binario

El motivo de la creación de un modelo con un clasificador es de establecer unos mínimos valores de rendimiento para los demás clasificadores. Así, el clasificador seleccionado es el DummyClassifier de Scikit-learn, el cual hace predicciones basándose en reglas simples[15]. En este caso la estrategia que sigue es llamada 'Estratificación' que predice basándose en la proporción de las labels. Los resultados obtenidos son los observables en la tabla de la imagen 5.6.

En general son valores muy pobres. Las intenciones con mayor falta de balance son las más perjudicadas, como se puede ver en los valores de precision y recall de Point of View, Disambiguation o Counter-vandalism. Observando la matriz de confusión se ve como efectivamente la matriz principal se encuentra compuesta de valores muy reducidos, explicando los valores encontrados en la tabla. Por otro lado, se observa que las intenciones están siendo clasificadas erróneamente con counter-vandalism indiscriminadamente.

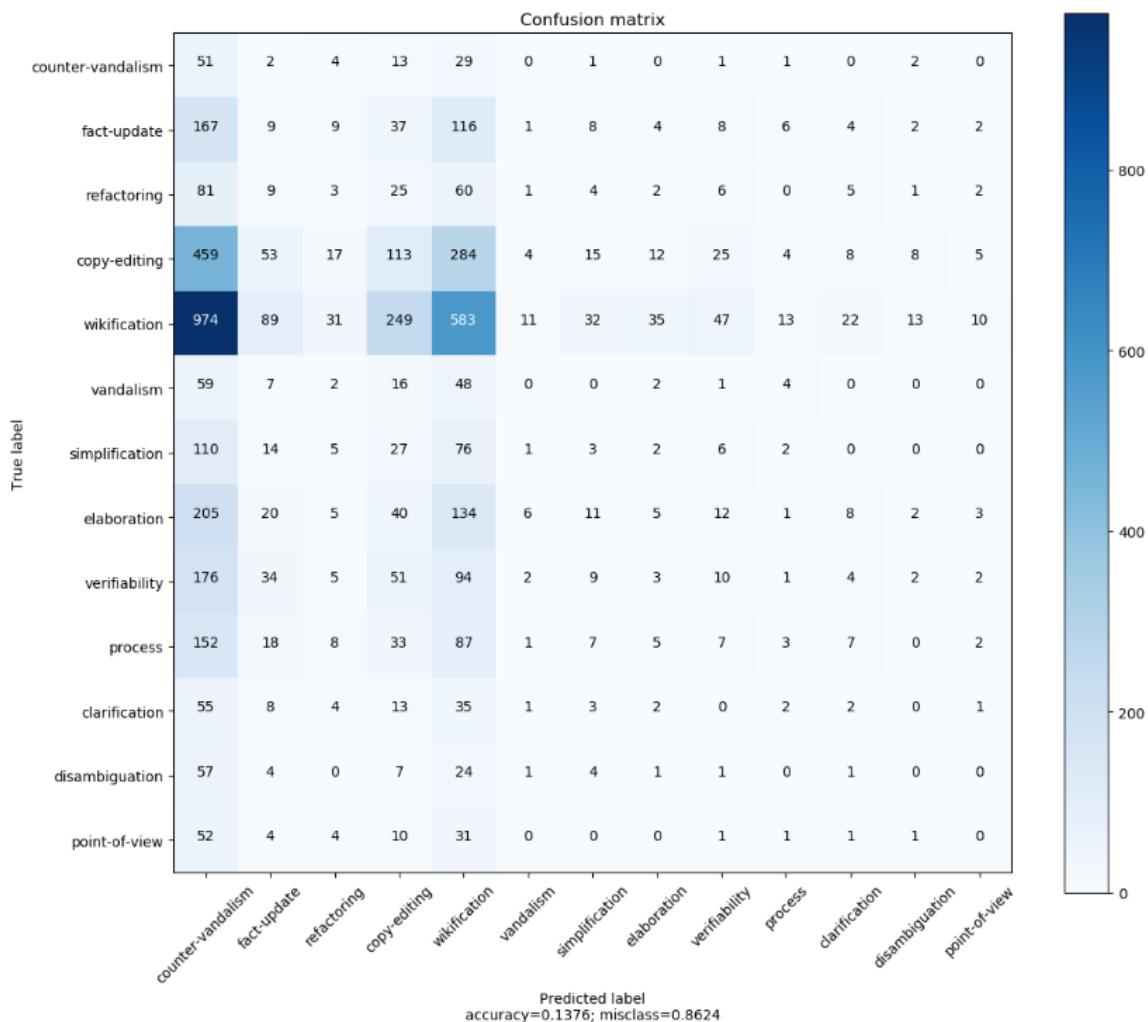


Figura 5.7: Matriz de confusión del clasificador Dummy binario

Random Forest

Random Forest es un algoritmo Ensemble que utiliza n árboles de decisión diferentes para generar una predicción[1]. Por este motivo, pueden reaccionar positivamente ante la falta de balance y son robustos ante el overfitting. En este caso, el objetivo del random forest es generar una predicción de 1 o 0. Se ha entrenado usando los parámetros por defecto y random_state = 64 para poder reproducir los resultados. Los resultados por intención se pueden observar en la figura 5.9.

En general se observa que los resultados son mejores que los obtenidos por el Dummy Classifier. Los valores de precisión son altos, sin embargo el recall es generalmente bajo con excepciones. Se puede ver claramente como aquellas clases más desequilibradas tiene valores de recall muy bajos. Esto indica que el clasificador está asignando más negativos de los que debería, mientras que hay un valor muy bajo de falsos positivos. En líneas generales es un modelo con una capacidad predictora débil. Observando su matriz de confusión vemos

	Intention	Precision	Recall	F1
0	counter-vandalism	0.870866	0.254259	0.390340
1	fact-update	0.759645	0.257209	0.369556
2	refactoring	0.608652	0.205287	0.301969
3	copy-editing	0.733808	0.394650	0.512596
4	wikification	0.772576	0.624407	0.689090
5	vandalism	0.749375	0.288019	0.412580
6	simplification	0.643821	0.335890	0.398222
7	elaboration	0.736693	0.544813	0.626013
8	verifiability	0.772896	0.639595	0.698970
9	process	0.846705	0.319359	0.463621
10	clarification	0.667632	0.055305	0.093515
11	disambiguation	0.784200	0.260628	0.365335
12	point-of-view	1.000000	0.582364	0.719088

Figura 5.8: Resultados por intención de default Random Forest binario

como su diagonal principal es muy diferente a la del clasificador binario pues en este caso el porcentaje de misclasificación es de el 51 %. Se observa que intenciones como clarification o refactoring son clasificadas erróneamente casi en su totalidad. Además, y aunque en menor medida con respecto al Dummy Classifier, aún se clasifican equivocadamente muchas revisiones como counter-vandalism.

SVM Linear

Se ha utilizado un algoritmo de Linear Support Vector Classification (LinearSVC) en lugar de una SVM con Kernel Lineal al uso por su mayor flexibilidad. SVC clasifica principalmente mediante la construcción de hiperplanos de modo que estos planos separen las diferentes clases, en este caso 1 o 0[16]. Se ha aplicado StandardScaler para escalar los datos de modo que sus valores se encuentren más uniformemente repartidos. Además se ha entrenado con los valores por defecto y random_state = 64 por motivos de reproducibilidad con resultados que pueden consultarse en la tabla de la imagen 5.10.

A simple vista puede verse que el clasificador es similar al random forest: altos valores de precision en ciertas intenciones pero con valores de recall generalmente bajos. En su matriz de confusión se ve que efectivamente la diagonal principal es ligeramente más débil que la del random forest, con una misclasificación del 66 % y un mayor porcentaje de confusión con counter-vandalism. Por otro lado se ve como el modelo es incapaz de predecir intenciones como clarification, point-of-view o process

En líneas generales ninguno de los tres clasificadores ha tenido un gran rendimiento pues el mejor porcentaje de misclasificación obtenido es ligeramente superior al 50 %. Es decir, sólo 1 de cada dos intenciones es predecida adecuadamente. Mientras que son precisos, sus bajos valores de recall los convierten en clasificadores no fiables pues asignan un gran número

CAPÍTULO 5. DESARROLLO PREVIO

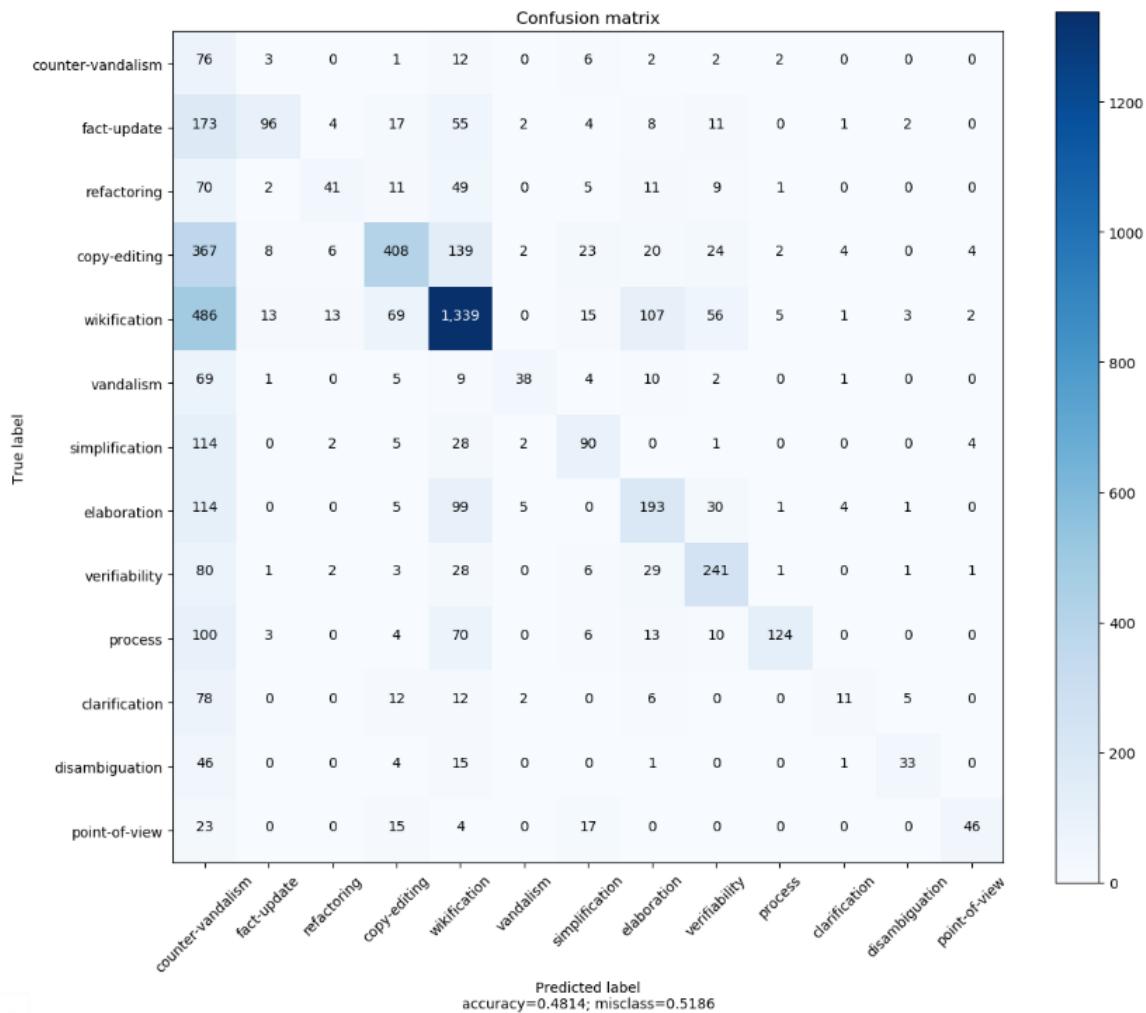


Figura 5.9: Matriz de confusión de default Random Forest binario

Algorithm	Precision	Recall	F1 micro
Dummy classifier	0.213536	0.191648	0.202001
Random Forest	0.752936	0.477784	0.584602
SVC	0.648503	0.371485	0.472376

Cuadro 5.2: Resultados micro-averaged de los 3 clasificadores binarios de base

de Falsos Negativos. En la tabla 5.2 vemos los valores micro-averaged para cada clasificador. Claramente, el que mejor rendimiento ha tenido es el Random Forest, superando al Dummy-Classifier y al SVC en todas las métricas.

No obstante, los valores de recall del random forest son alarmantemente bajos. Sin embargo, y debido a que en comparación tiene el mejor rendimiento como hemos podido ver en su matriz de confusión, será el algoritmo utilizado a partir de ahora en la clasificación binaria en búsqueda de su optimización, pues por ejemplo el SVC falló en su totalidad la clasificación

	Intention	Precision	Recall	F1
0	counter-vandalism	0.586231	0.441529	0.495271
1	fact-update	0.527069	0.198248	0.280371
2	refactoring	0.487456	0.258820	0.335521
3	copy-editing	0.582443	0.246944	0.328946
4	wikification	0.740611	0.542971	0.612757
5	vandalism	0.664106	0.286994	0.380450
6	simplification	0.489223	0.178650	0.248467
7	elaboration	0.742780	0.380903	0.501048
8	verifiability	0.731040	0.594170	0.649355
9	process	0.383056	0.081122	0.133490
10	clarification	0.234822	0.051925	0.069583
11	disambiguation	0.416424	0.169703	0.205362
12	point-of-view	0.127472	0.057350	0.066970

Figura 5.10: Resultados por intencion de default SVC

de las intenciones con clarification, point of view o process.

Llegados a este punto está claro que el imbalance en los datos está generando un gran perjuicio, dificultando mucho la tarea de predicción. Por ello, se va a aplicar Over-Sampling en búsqueda de mejores resultados en el Random Forest.

Over-Sampling

Para solventar la falta de equilibrio, hay numerosas técnicas de sampling. Estas técnicas se dividen en Over-Sampling y Under-Sampling. Mientras que en over-sampling se aumenta la frecuencia de aparición de las clases minoritarias, en under-sampling se reduce la proporción de las labels mayoritarias. En este caso debido al tamaño de los datos, se ha decidido hacer uso de una técnica de Over-Sampling llamada ADASYN.

ADASYN hace uso de una distribución con pesos para cada clase en minoría que resulta más difícil de predecir, así, genera ejemplos de esa clase en minoría mejorando los resultados en la clasificación reduciendo el bias.[10]

Para hacer uso de ADASYN se implementa el over-sampling en un pipeline proporcionado por la librería imblearn ya que transforma solo los datos del train set durante el proceso de cross validation, dando lugar a que el test set esté libre de muestras creadas con over-sampling. Procedemos a la creación de un Random Forest con parámetros default para ver si sus resultados se han mejorado con respecto a los resultados obtenidos previo al balanceo. Los resultados obtenidos por el Random Forest pueden verse en la tabla de la figura 5.12.

Vemos como sus valores de precisión se han reducido, pero al mismo tiempo Recall ha aumentado así como F1. La bajada de precision en sí no es positiva, sin embargo la proporción

CAPÍTULO 5. DESARROLLO PREVIO

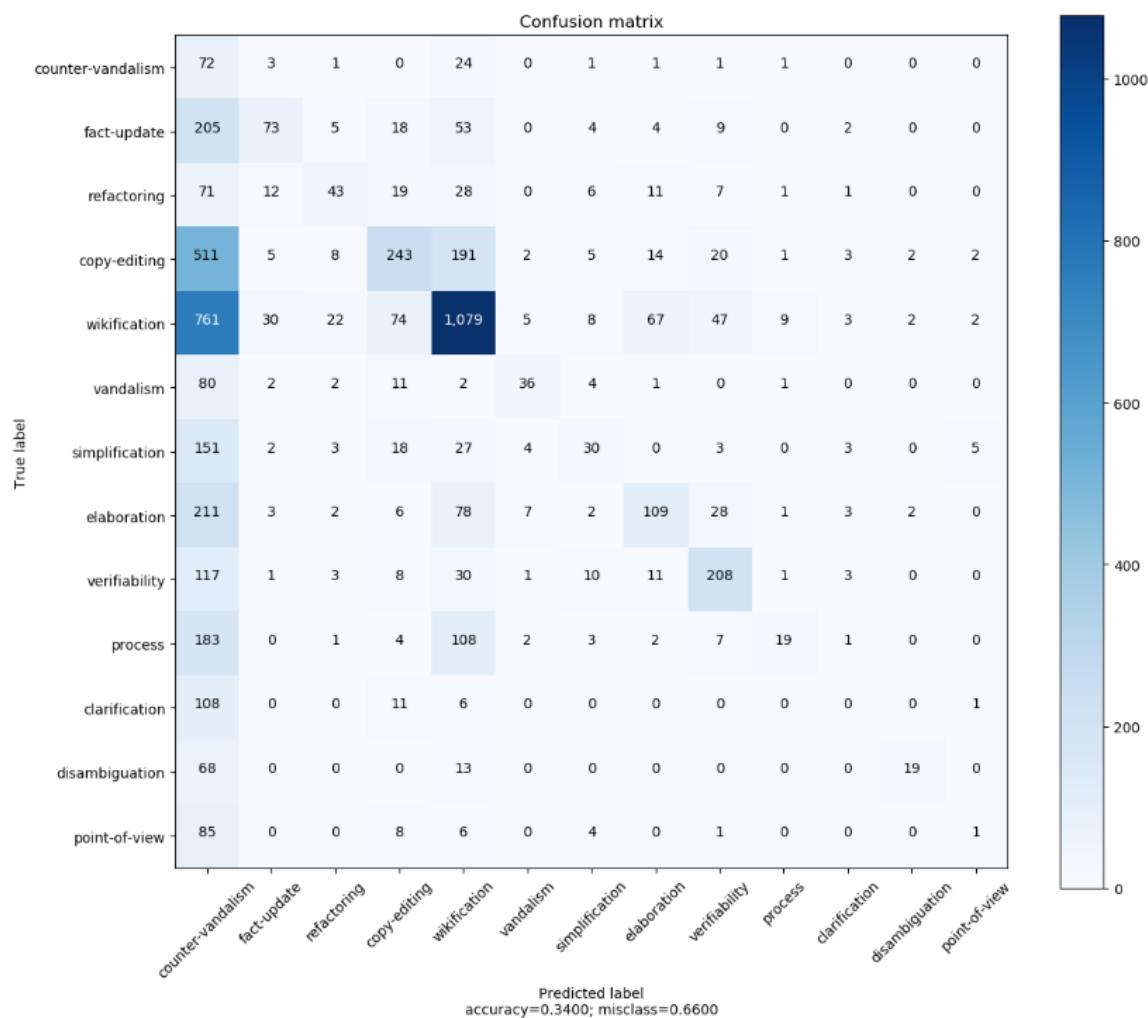


Figura 5.11: Matriz de confusión de SVC

en la que ha bajado en contraposición con la subida obtenida en recall hace que el trade-off sea positivo. Esto es debido a que ahora que los dataset de entrenamiento se encuentran balanceados el clasificador asigna más positivos mientras que anteriormente asignada 0 con mayor facilidad. Esto, también puede verse en la matriz de confusión, cuya diagonal principal se refuerza ligeramente, obteniendo valores más altos en todas las intenciones y reduciendo el porcentaje de misclasificación al 47 %. Además, se observa que cada vez hay menos confusiones con counter-vandalism y se observa una clara mejoría en aquellas intenciones que tenían una clara falta de balance como point of view.

	Precision	Recall	F1 micro
	0.634887	0.58802	0.610556

Cuadro 5.3: Resultados micro averaged de over-sampling en el Random Forest binario

En definitiva, el Over-sampling ha tenido un efecto positivo por lo que se procederá a intentar mejorar los resultados obtenidos haciendo Feature Engineering y ajustando los hi-

	Intention	Precision	Recall	F1
0	counter-vandalism	0.777242	0.420784	0.532608
1	fact-update	0.536928	0.466285	0.490948
2	refactoring	0.484232	0.361475	0.406565
3	copy-editing	0.621315	0.534808	0.573636
4	wikification	0.758610	0.655882	0.702033
5	vandalism	0.564130	0.453153	0.493376
6	simplification	0.509271	0.416387	0.383081
7	elaboration	0.651862	0.696927	0.673615
8	verifiability	0.733903	0.716508	0.723814
9	process	0.602732	0.462150	0.516976
10	clarification	0.546754	0.290138	0.320929
11	disambiguation	0.699689	0.352709	0.412033
12	point-of-view	1.000000	0.684718	0.805000

Figura 5.12: Resultados Random Forest binario tras Over-Sampling

perparámetros del Random Forest.

Feature Engineering

Feature engineering es el proceso por el cual se crean o eliminan atributos que hacen que los algoritmos predictores funcionen. En este caso, el tipo de feature engineering en el que nos centraremos utiliza un enfoque Model-based. El funcionamiento es simple, en base a un threshold de importancia dado, el propio algoritmo clasificador (Random Forest en este caso) decide la importancia de cada atributo y descarta todos aquellos con un nivel de importancia menor. El threshold más óptimo se obtiene mediante gridSearch de entre los valores $0.75 * \text{mean}$, mean , $1.25 * \text{mean}$ and $1.5 * \text{mean}$ siendo mean el valor medio de importancia de los atributos del dataset según el clasificador. El Random Forest con hiperparámetros por defecto los resultados obtenidos pueden verse en la tabla de la imagen 5.15.

Precision	Recall	F1 micro
0.646687	0.603768	0.624491

Cuadro 5.4: Resultados micro averaged de over-sampling y feature engineering en el Random Forest binario

Los valores de precisión, recall y f1 son ligeramente más altos que antes de feature-engineering y esto además se observa en la matriz de confusión, muy similar a la matriz de confusión previa a aplicar feature engineering pero con un porcentaje de misclasificación un 1% menor y menor número de FP.

CAPÍTULO 5. DESARROLLO PREVIO

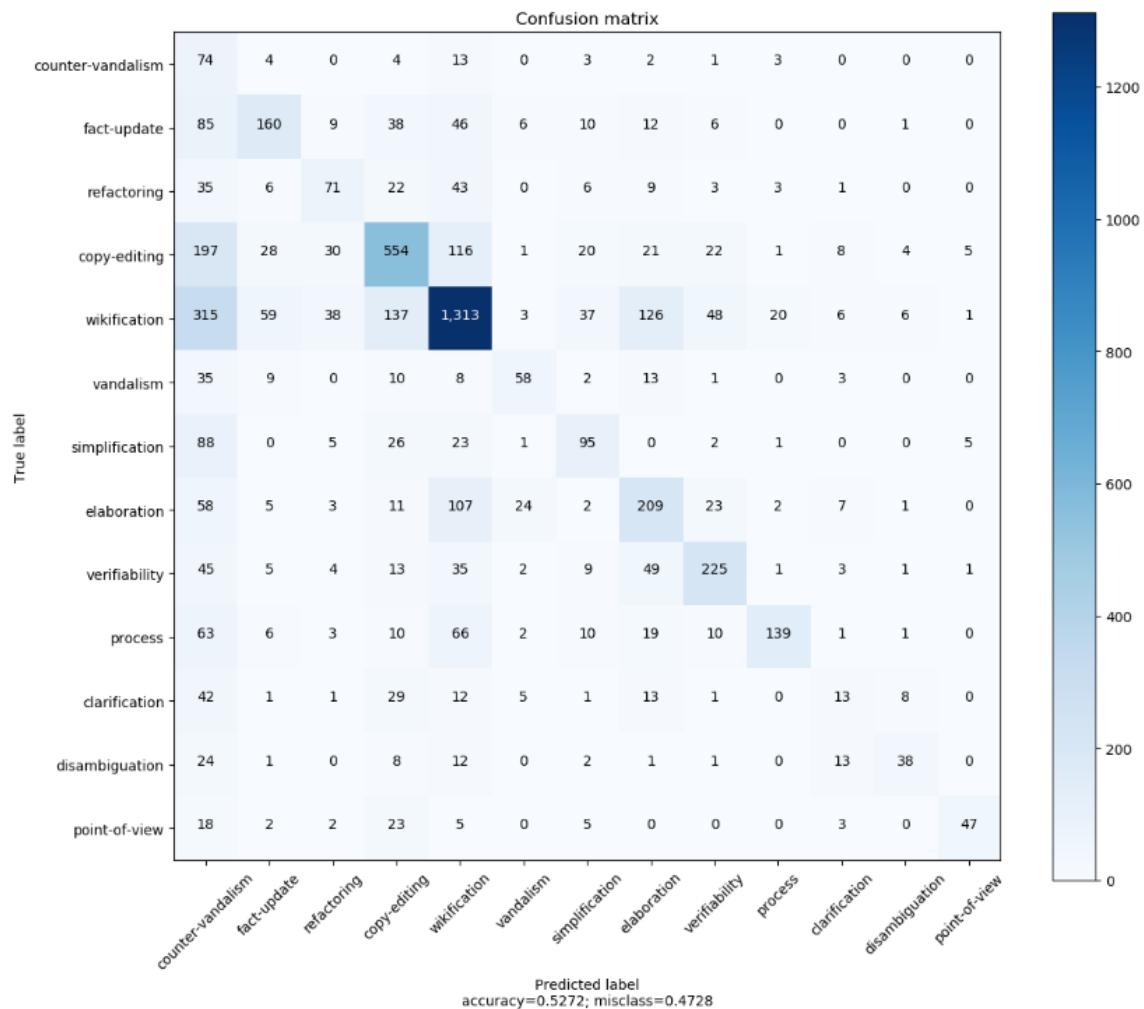


Figura 5.13: Matriz de confusión tras realizar over-sampling en el Random Forest binario

	Intention	Precision	Recall	F1	Feature Engineering	Threshold
0	counter-vandalism	0.778699	0.646185	0.689049		mean
1	fact-update	0.514346	0.487699	0.493058		1.25*mean
2	refactoring	0.468619	0.351677	0.398886		0*mean
3	copy-editing	0.641270	0.563405	0.598175		0*mean
4	wikification	0.759129	0.663853	0.707014		0*mean
5	vandalism	0.645099	0.496018	0.542738		0*mean
6	simplification	0.485749	0.422459	0.398862		0*mean
7	elaboration	0.651809	0.670993	0.660862		0*mean
8	verifiability	0.726087	0.737927	0.728638		0.75*mean
9	process	0.573872	0.474249	0.517251		mean
10	clarification	0.612841	0.390420	0.386251		mean
11	disambiguation	0.712669	0.421134	0.458514		mean
12	point-of-view	1.000000	0.993745	0.996833		mean

Figura 5.14: Resultados con Feature Engineering y Over-Sampling en Random Forest binario

Tuning del modelo final

Finalmente, se ha decidido seguir haciendo uso de feature engineering y over-sampling pues toda mejora, aunque pequeña, es bienvenida.

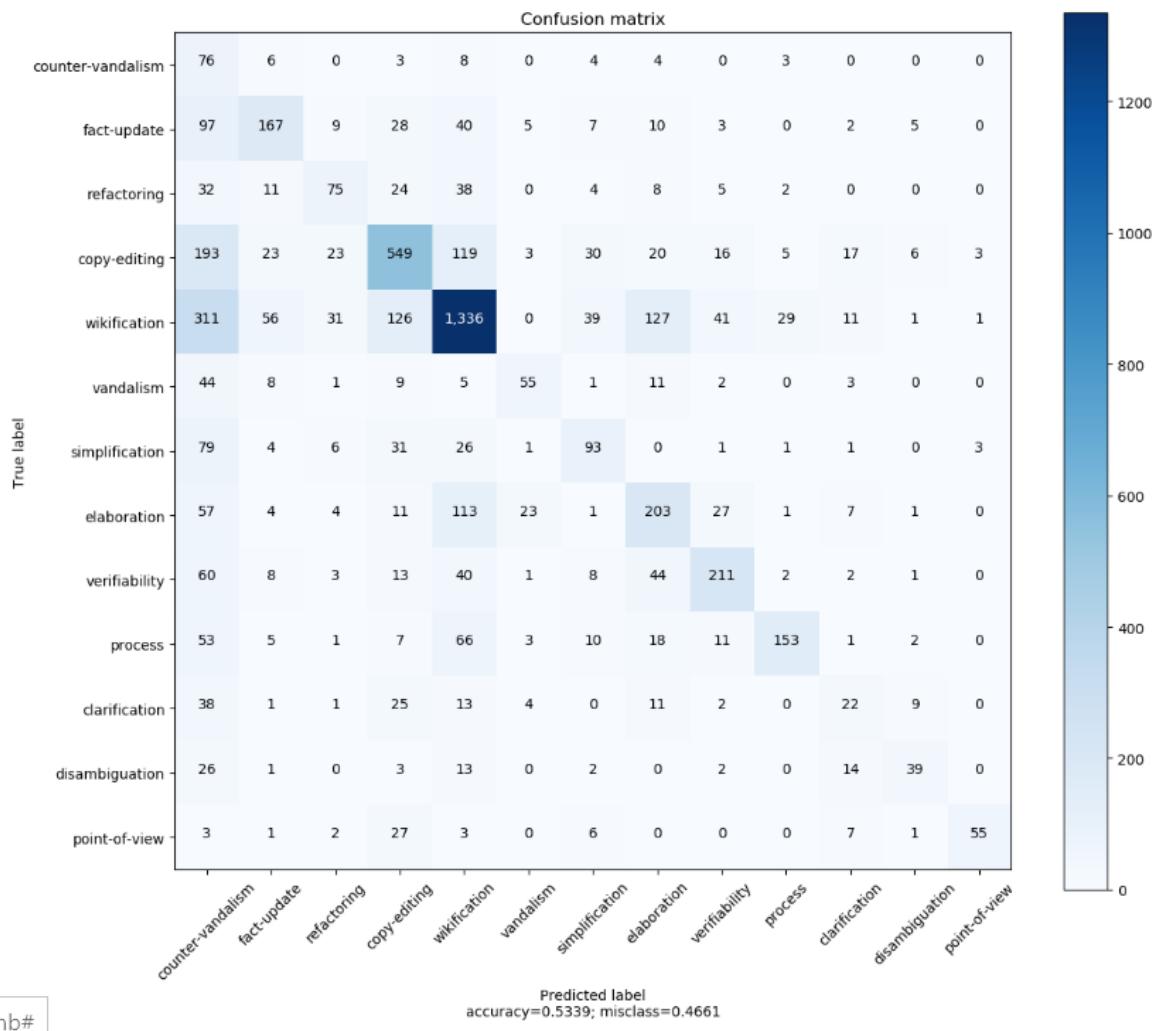


Figura 5.15: Matriz de confusión con Feature Engineering y Over-Sampling del Random Forest binario

Para el entrenamiento de el modelo final haciendo uso de Random Forest, se hace uso de GridSearchcv. Con GridSearchcv se itera entre los valores [1,2,4,8,16,32,48,64,80,96,128,160,192,256,385,512] para el parámetro del Random Forest n_estimators, que determina el número de arboles de decisión que se generan y de nuevo entre 0.75*mean, mean, 1.25*mean and 1.5*mean de cara al threshold de feature engineering.

Así, los resultados obtenidos son los observables en la tabla de la figura 5.17. Se aprecia una mejora en los resultados a simple vista, obteniendo valores relativamente altos de todas las métricas. Observando la matriz de confusión vemos como su diagonal principal muestra valores intermedios, pero visiblemente más altos que anteriormente. Claramente algunas intenciones son clasificadas con mucho más éxito que otras, sin embargo, el porcentaje de misclasificación es del 41 %, el mejor de entre los obtenidos hasta ahora aunque no obstante es un porcentaje alto de error.

CAPÍTULO 5. DESARROLLO PREVIO

	Intention	Precision	Recall	F1	FE threshold	N_estimators
0	counter-vandalism	0.793396	0.675785	0.707020	mean	385
1	fact-update	0.561341	0.538596	0.539395	mean	256
2	refactoring	0.553980	0.434899	0.478802	0*mean	256
3	copy-editing	0.664299	0.619651	0.639648	0*mean	385
4	wikification	0.766797	0.749041	0.756438	0.75*mean	192
5	vandalism	0.665165	0.517245	0.560203	0*mean	192
6	simplification	0.616544	0.481190	0.426262	0*mean	128
7	elaboration	0.658665	0.748815	0.700688	0*mean	192
8	verifiability	0.732362	0.786393	0.756328	mean	192
9	process	0.675325	0.491521	0.566737	0*mean	192
10	clarification	0.565297	0.476739	0.417546	1.5*mean	512
11	disambiguation	0.808815	0.429174	0.515872	0*mean	128
12	point-of-view	1.000000	1.000000	1.000000	mean	16

Figura 5.16: Resultados del ajuste de hiperparámetros del Random Forest binario

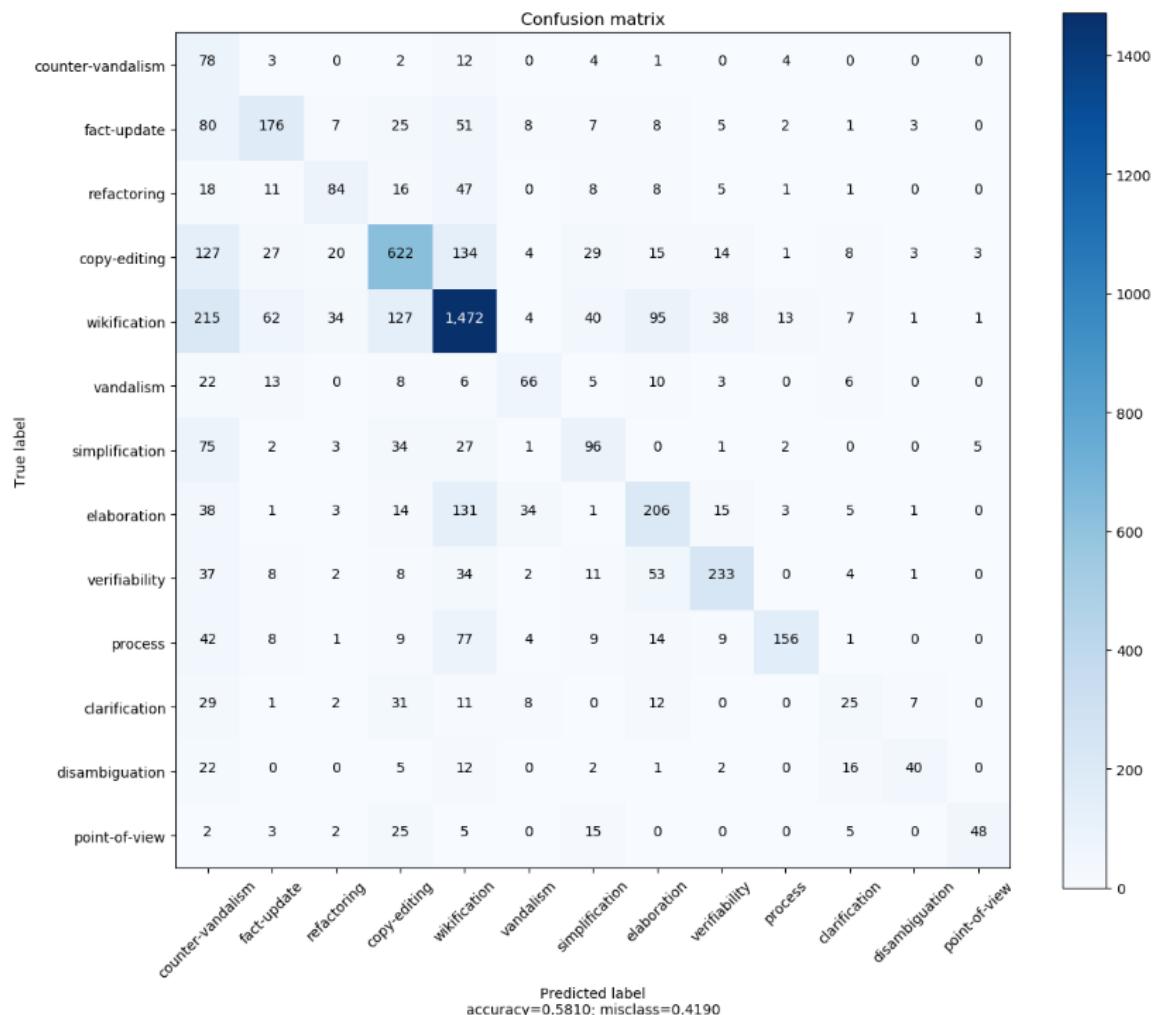


Figura 5.17: Resultados del ajuste de hiperparámetros del Random Forest binario

Algorithm	Precision	Recall	F1 micro
Random Forest	0.752936	0.477784	0.584602
RF+OS	0.634887	0.58802	0.610556
RF+OS+FE	0.646687	0.603768	0.624491
RF Ajustado	0.670533	0.658465	0.664444

Cuadro 5.5: Resultados de los diferentes Random Forest binarios creados

En la tabla 5.5 se ve en orden todos los modelos de random forest generados hasta el momento. Los valores, aunque modestos, se ve como mejoran progresivamente obteniendo tras el ajuste de los hiperparámetros de feature engineering y random forest y over-sampling como obtenemos finalmente un valor de 0.66 F1 micro, notablemente más alto que los obtenidos hasta ahora[21]

5.5.4. Clasificación multilabel

La clasificación multilabel es aquella en la que el target puede tomar la forma de diferentes labels. En este caso, cada intención es un label posible y no son excluyentes entre sí, es decir, una revisión puede tener varias intenciones al mismo tiempo, sin limitaciones.

No se ha eliminado la intención 'Other' y se ha utilizado un random_state = 64 en todos los algoritmos para facilitar la reproducibilidad.

Un poco de pre-procesamiento es necesario para adecuar el target al formato requerido por los algoritmos de clasificación multilabel. Para ello se define una función llamada Creating-Multilabel que unifica todas las intenciones en una sola columna. Después, se utiliza el MultiLabelBinarizer default para generar una 'sparse matrix' con el target. Además se ha usado StandardScaler en el dataset de cara a ser usado en el algoritmo KNN pues normalizando los valores de los atributos se mejora la eficiencia del algoritmo.

Los algoritmos han sido entrenados y evaluados mediante cross validation con la estrategia StratifiedKFold con k = 4 gracias al uso de GridSearchCV.

En base a toda la información obtenida, el proceso a la hora de determinar el modelo siguiente ha sido el siguiente:

1. Clasificador dummy para establecer métricas de base: mediante un clasificador que se basa en reglas simples prefijadas, obtenemos unas métricas que establecen un rendimiento mínimo a superar por el resto de clasificadores.
2. Random Forest: Se aplicarán algoritmos de random forest que hace uso de n árboles de decisión para generar una predicción, reaccionando positivamente ante la falta de balance.
3. Multilabel K Nearest Neighbors: Versión desarrollada para la clasificación multilabel del algoritmo Nearest Neighbors para generar predicciones.
4. Feature engineering: El objetivo aquí es aplicar técnicas de feature engineering, cuya función es la optimización del conjunto de datos, para determinar si generan un efecto positivo en los resultados de la clasificación.

5. Ajuste de los parámetros del modelo final: Una vez todas los resultados de los puntos anteriores han sido estudiados se ajustarán los hiperparámetros de los modelos seleccionados como más óptimos en busca de incrementar sus valores de precisión, recall y F1 aún más.

Clasificador Dummy

Precision	Recall	F1 micro	F1 macro	Accuracy
0.229347	0.217741	0.216844	0.087811	0.067042

Cuadro 5.6: Resultados del dummyClassifier multilabel

El motivo de la creación de un modelo con un clasificador de prueba es de establecer unos mínimos valores de rendimiento para los demás clasificadores. Así, el clasificador seleccionado es el DummyClassifier de Scikit-learn, el cual hace predicciones basándose en reglas simples. En este caso la estrategia que sigue es llamada 'Estratificación' que predice basándose en la proporción de las labels. Los resultados obtenidos son los observables en la tabla 5.6.

En general son valores muy bajos en todas las métricas. Con una precisión de 0,06, es decir, solo un 6 % de las clasificaciones son correctas. Esto, concuerda con lo que observamos en su matriz de confusión donde la diagonal principal es prácticamente inexistente.

Random Forest

Precision	Recall	F1 micro	F1 macro	Accuracy
0.746308	0.441838	0.554825	0.309699	0.391167

Cuadro 5.7: Resultados del default Random Forest multilabel

De nuevo, el Random Forest ha sido escogido como algoritmo clasificador. Sus buenos resultados en la clasificación binaria y el hecho de que sean naturalmente capaces de realizar tareas de clasificación multilabel lo convierten en un candidato a considerar.

En este caso el Random Forest utilizado tiene los hiperparámetros por defecto con Random_state = 64 para facilitar la reproducibilidad. Los resultados obtenidos en el test de pruebas, observables en la tabla 5.7 muestran una accuracy del 39 % y valores bajos de recall, 0,44. El rendimiento es menor del esperado sin embargo aún tiene capacidad de mejora. Para obtener más información acerca de la clasificación realizada, se ha generado su matriz de confusión.

Se puede ver como hay grandes diferencias en la cantidad de aciertos según la intención, en este caso y a diferencia de la clasificación binaria, no se observa que las intenciones en minoría tengan peores resultados de media que aquellas en mayor proporción como Wikification. Curiosamente todas las intenciones son a menudo clasificadas erróneamente como Counter-Vandalism. Por ejemplo, Refactoring es catalogada más veces como counter-vandalism que como refactoring, lo cual también sucede en intenciones como clarification,

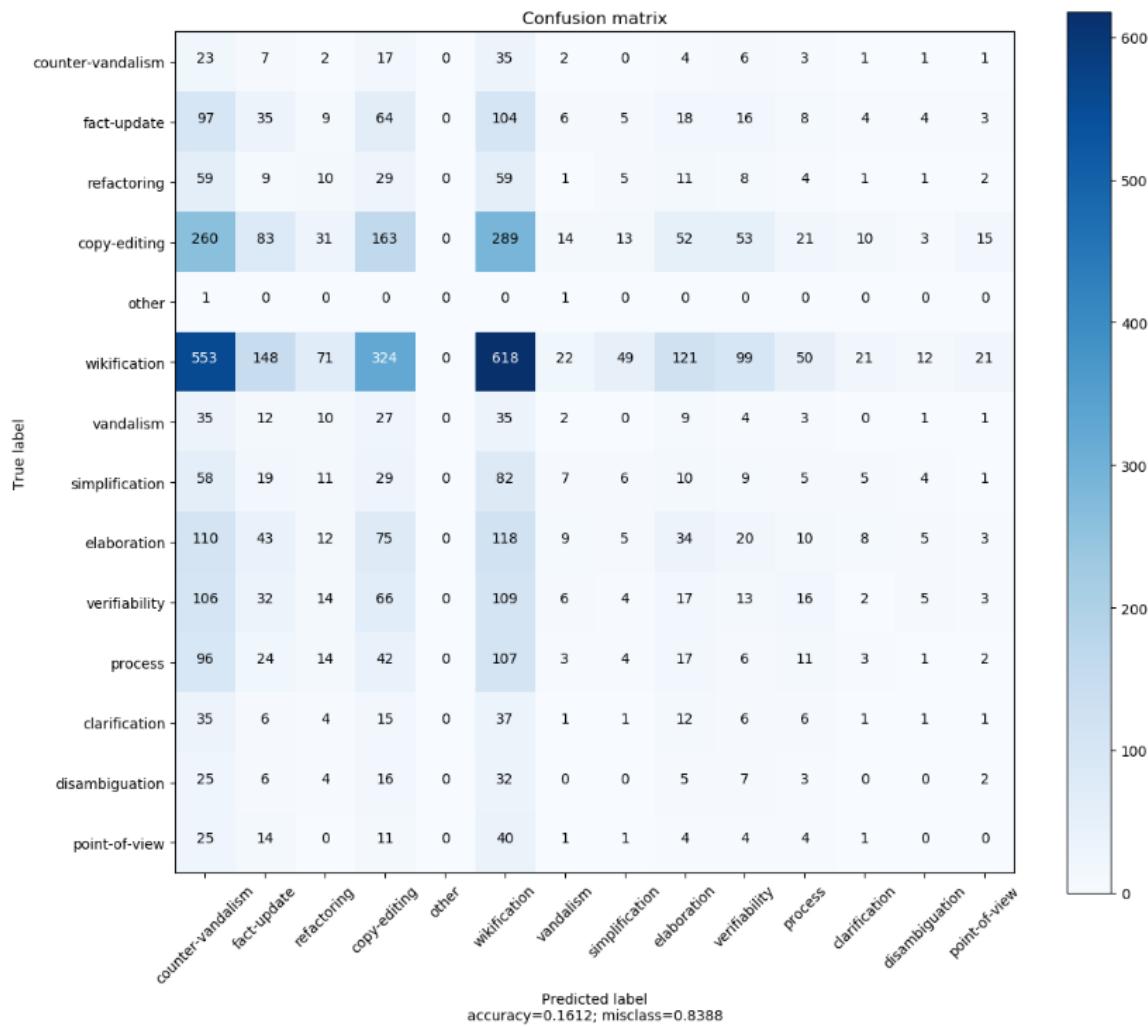


Figura 5.18: Matriz de confusión de Dummy Classifier multilabel

disambiguation o point of view. Esto puede ser debido a que los motivos por los cuales una intención es catalogada como Counter-Vandalism son difusos, por ejemplo, que haya revertido respecto a la edición anterior. En resumen, el 54 % de las labels han sido asignadas de modo erróneo, es decir, 1 de cada 2 predicciones, es incorrecta.

MLKNN

Precision	Recall	F1 micro	F1 macro	Accuracy
0.614699	0.416713	0.497729	0.289816	0.340665

Cuadro 5.8: Resultados de default MLKNN

MLkNN es una adaptación para clasificación multilabel del algoritmo K-NearestNeighbors. Encuentra los ejemplos más cercanos a la intención del target y utiliza inferencia Bayesiana

CAPÍTULO 5. DESARROLLO PREVIO

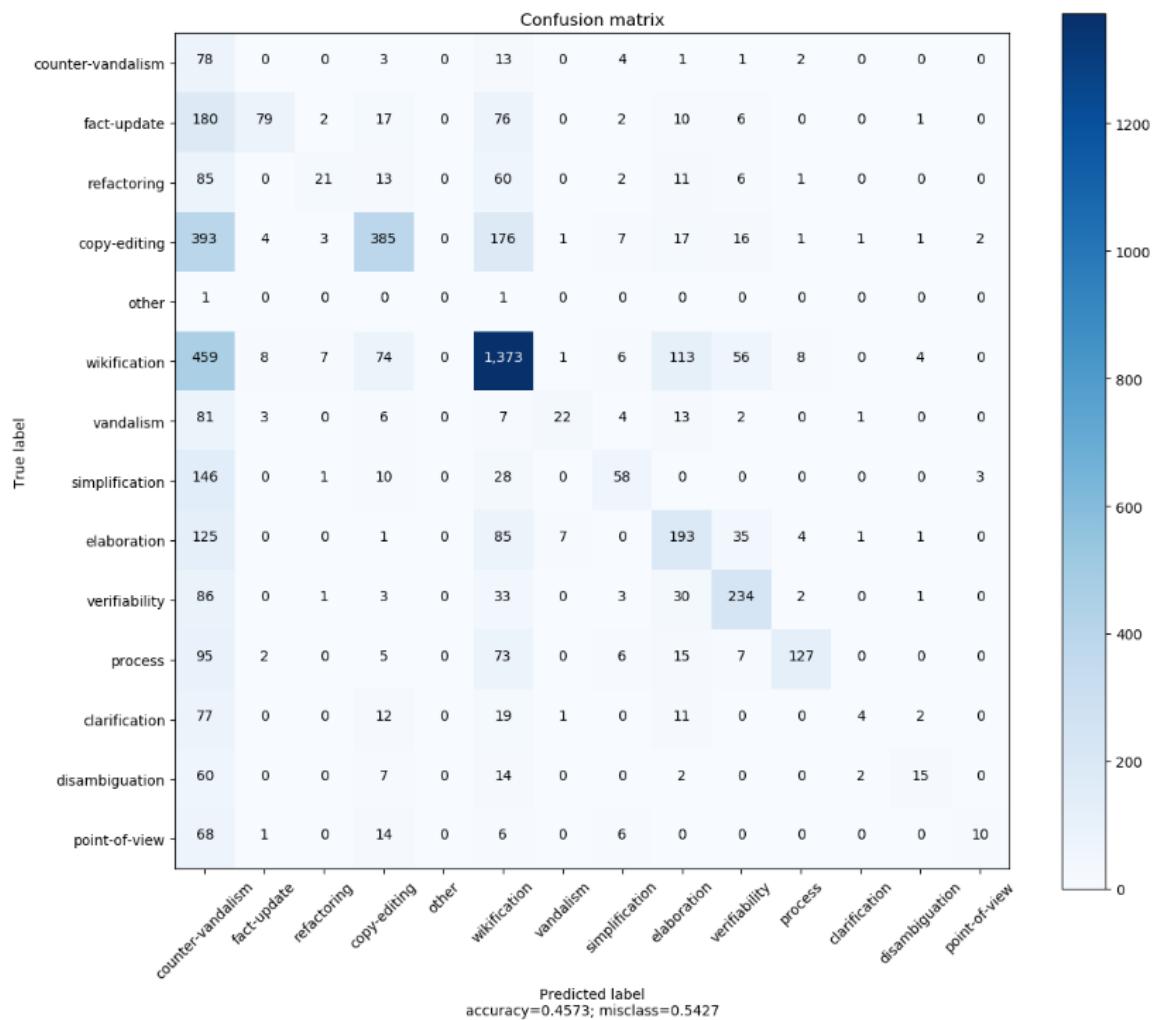


Figura 5.19: Matriz de confusión de default Random Forest multilabel

para seleccionar las labels asignadas.

El algoritmo ha sido creado utilizando todos los parámetros por defecto, sus resultados se pueden ver en la figura 5.8. Son unos resultados similares al random forest, sin embargo la precision y el recall son ligeramente menores. Sin embargo para entender mejor estos resultados, es necesario consultar la matriz de confusión.

La diagonal principal es bastante débil, con una misclasificación de 0,59. El algoritmo, al igual que el Random Forest no está siendo capaz de clasificar las intenciones y en su lugar, como se puede ver de nuevo, está asignando la intención de Counter-Vandalism a todo.

En la tabla 5.21 se pueden ver los resultados de los tres clasificadores entrenados hasta el momento. Random Forest tiene el mejor rendimiento a pesar de que su rendimiento es bajo. Los resultados de MLKNN aunque peores, superan a los del DummyClassifier y se

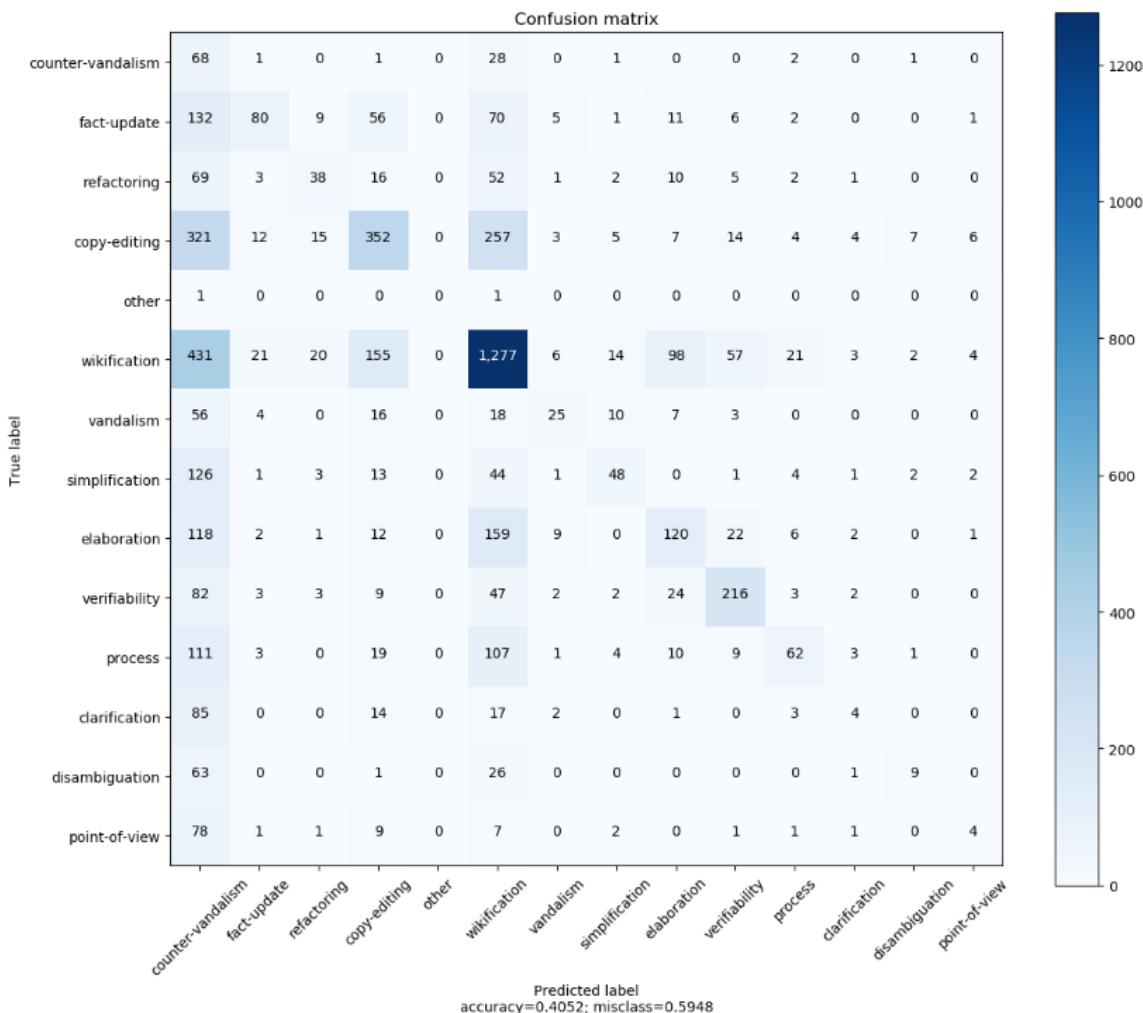


Figura 5.20: Matriz de confusión de default KNN

	Dummy classifier	Random Forest classifier	ML KNN
Precision	0.229347	0.746308	0.614699
Recall	0.217741	0.441838	0.416713
F1 micro	0.216844	0.554825	0.497729
F1 macro	0.087811	0.309699	0.289816
Accuracy	0.067042	0.391167	0.340665

Figura 5.21: Resultados de los clasificadores de base multilabel

ajustarán sus hiperparámetros en búsqueda de un incremento de su rendimiento así como con el Random Forest. Pero antes de realizar un ajuste de los hiperparámetros, se aplicará Feature Engineering en búsqueda de optimizar más el dataset para facilitar la tarea de clasificación y con suerte, poder solucionar el gran porcentaje de clasificaciones incorrectas en el caso del Random Forest.

Feature Engineering

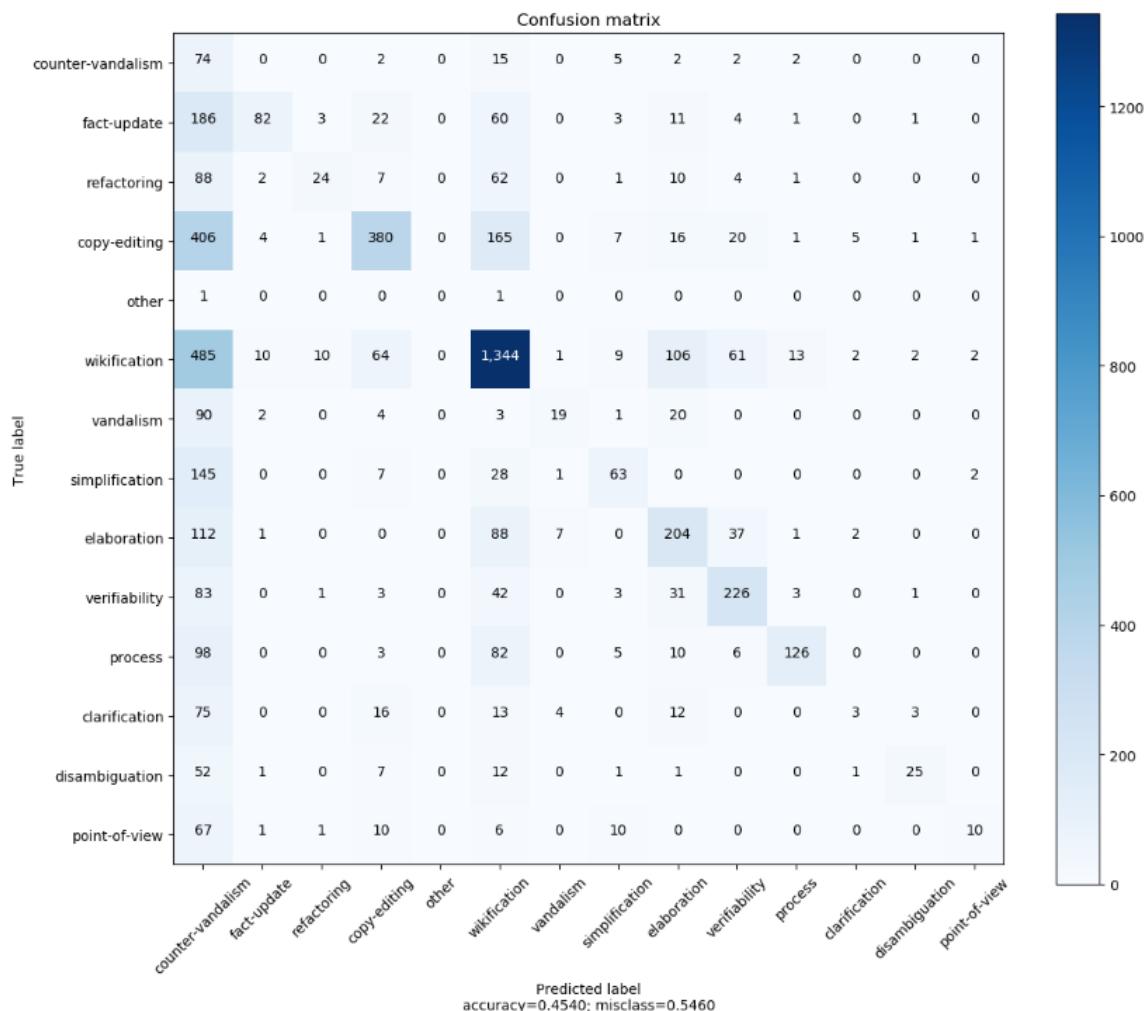


Figura 5.22: Matriz de confusión de Random Forest con Feature Engineering multilabel

El tipo de Feature Engineering a aplicar es el mismo que el de la clasificación binaria, siguiendo un enfoque basado en el modelo. Por motivos técnicos sólo se ha aplicado al Random Forest pues el algoritmo de MLKNN no tiene los atributos necesarios como para poder valorar la importancia de cada feature del dataset.

Así, los resultados obtenidos son a priori casi iguales que los obtenidos previamente. En la tabla 5.23 podemos ver los valores obtenidos en las diferentes métricas en comparación con los demás modelos entrenados previo a Feature Engineering.

Observando su matriz de confusión vemos que la situación es similar, con muchas intenciones clasificadas erróneamente como counter-vandalism e incapaz de predecir clarification prácticamente en su totalidad. De hecho, muchas intenciones son catalogadas como counter-vandalism más veces que ellas mismas.

	Dummy classifier	Random Forest classifier	ML KNN	RF with Feature Engineering
Precision	0.229347	0.746308	0.614699	0.735617
Recall	0.217741	0.441838	0.416713	0.442496
F1 micro	0.216844	0.554825	0.497729	0.553229
F1 macro	0.087811	0.309699	0.289816	0.314779
Accuracy	0.067042	0.391167	0.340665	0.386944

Figura 5.23: Comparación de resultados de base con RF y Feature Engineering multilabel

Tuning modelo final

Para el ajuste final de los hiperparámetros de los modelos se han probado dos técnicas: GridSearchCV y Algoritmos genéticos. La implementación del algoritmo genético se ha realizado mediante EvolutionaryAlgorithmSearchCV que permite realizar Stratified KFold cross-validation. Sin embargo, GridSearchCV realizaba la tarea en menor tiempo por lo que fue seleccionado en su lugar.

A la hora de realizar el ajuste del Random Forest, se ha creado un Pipeline con el clasificador y feature engineering que es utilizado por GridSearch para buscar la mejor combinación posible de hiperparámetros. En este caso el hiperparámetro a optimizar es el número de estimadores del bosque, es decir, el número de áboles. Este valor varía entre 1 y 512 en potencias de dos. Tras el ajuste, el parámetro más óptimo de threshold en feature engineering de 0.75*mean y un número de estimadores igual a 32.

Por otro lado, para el MLKNN, no se ha hecho uso de un pipeline y los parámetros a ajustar son K y S, K representando el número de vecinos y S el factor de ajuste. El rango de K varía entre 1 y 30 mientras que el de S entre 0.0001 y 100 en potencias de 10. Se ha realizado la misma técnica de cross-validation que con el Random Forest. Tras el ajuste los parámetros más óptimos han sido k = 4 y s = 0.0001

Los resultados obtenidos por ambos clasificadores se encuentran en la tabla de la figura 5.26. Claramente, el Random Forest ha obtenido los mejores resultados con una precision de casi 0,8 aunque con un recall bajo de 0,47. Su Accuracy se ha incrementado ligeramente hasta 0,41 pero son resultados demasiado bajos. MLKNN aunque también ha mejorado sus resultados en métricas como recall o Accuracy, sigue teniendo peor rendimiento que el Random Forest y un rendimiento muy bajo en líneas generales.

Observando las matrices de confusión de los clasificadores se puede ver como en el caso del Random Forest la situación es muy similar a la previa al ajuste de los hiperparámetros, con una diagonal principal con valores ligeramente más altos y de ahí el incremento en Accuracy pero no obstante todavía sigue clasificando las revisiones erróneamente como counter-vandalism y es prácticamente incapaz de predecir intenciones como point of view o clarification. El caso del MLKNN se ve que el ajuste tampoco ha solucionado el problema existente con las clasificaciones erróneas de counter-vandalism. Además, sigue haciendo clasificaciones erróneas

CAPÍTULO 5. DESARROLLO PREVIO

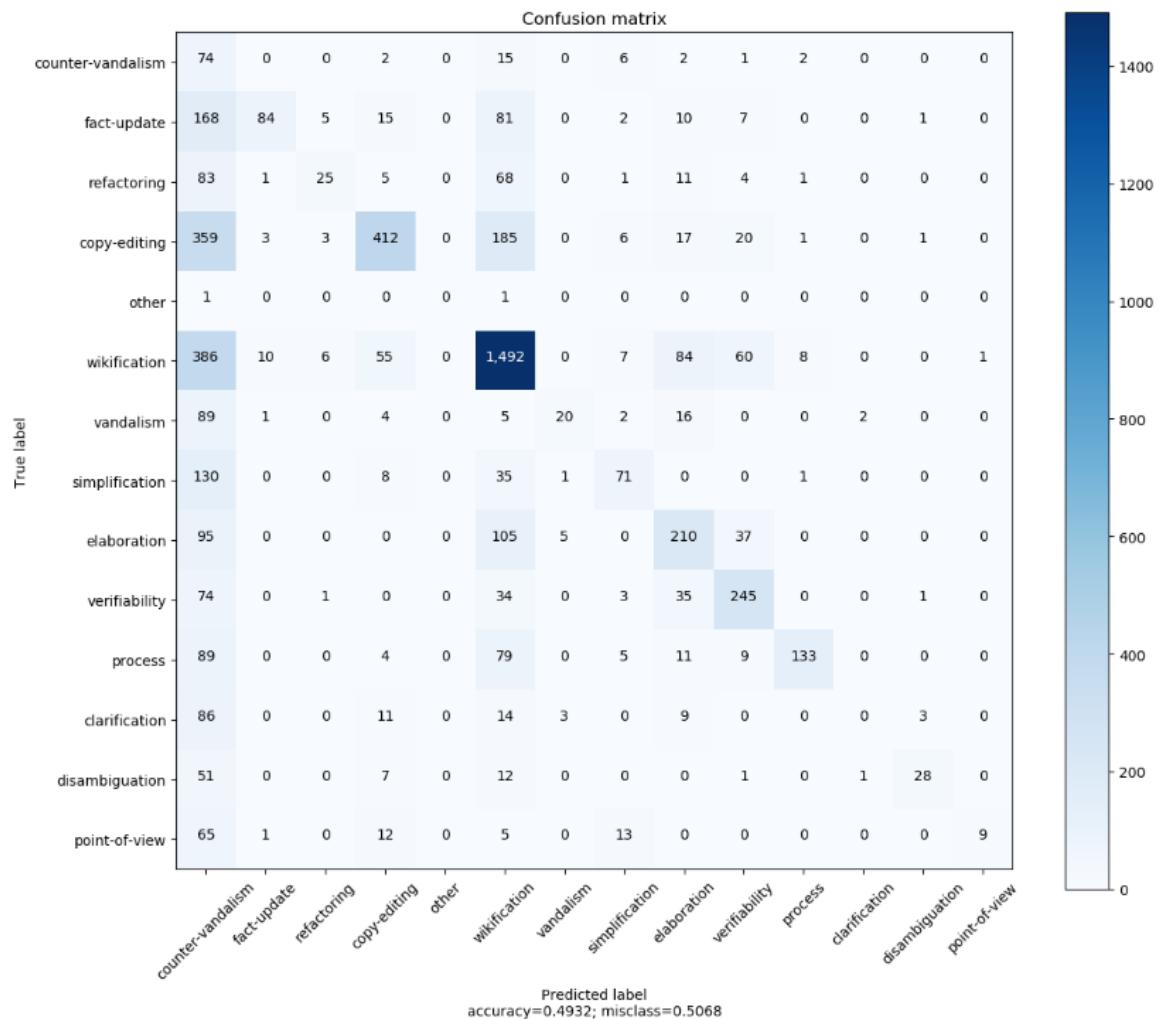


Figura 5.24: Matriz de confusión de Random Forest con parámetros ajustados multilabel

en general, de modo muy similar al Random Forest en intenciones como clarification o point of view con un porcentaje de misclasificación total del 54 %.

5.5.5. Conclusiones

En definitiva está claro que los clasificadores tienen problemas prediciendo las diferentes intenciones que pueden existir detrás de una revisión. A pesar de utilizar dos enfoques diferentes y distintos algoritmos, los resultados no son sorprendentes. Sin embargo, no siempre es posible obtener clasificaciones perfectas y es por esto que el tipo de datos que se esté prediciendo es muy importante.

El rendimiento de estos clasificadores aún puede ser incrementado. Con conocimiento del dominio suficiente, se podría ajustar el dataset existente para reducir ambigüedades entre las diferentes intenciones o eliminar por completo atributos que no sean necesarios para determinadas intenciones. Sin embargo, eso se escapa al objetivo de este análisis.

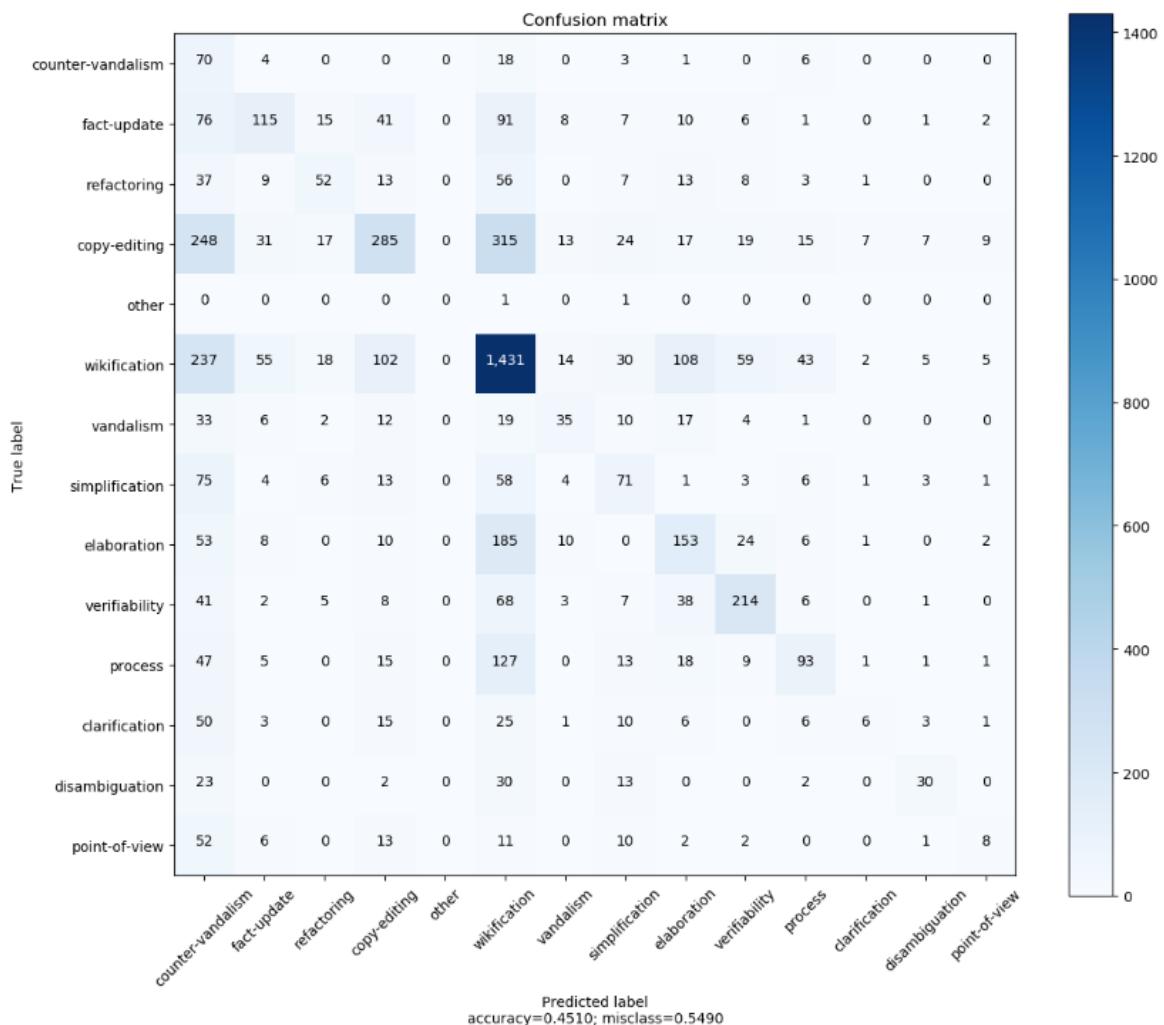


Figura 5.25: Matriz de confusión de MLKNN con parámetros ajustados

	Dummy classifier	Random Forest classifier	ML KNN	RF with Feature Engineering	Tuned Random Forest with Feature Engineering	Tuned MLKNN
Precision	0.229347	0.746308	0.614699	0.735617	0.760734	0.511497
Recall	0.217741	0.441838	0.416713	0.442496	0.474013	0.491035
F1 micro	0.216844	0.554825	0.497729	0.553229	0.584657	0.490279
F1 macro	0.087811	0.309699	0.289816	0.314779	0.337217	0.307753
Accuracy	0.067042	0.391167	0.340665	0.386944	0.415274	0.379025

Figura 5.26: Resultados de modelos multilabel finales

Algorithm	Precision	Recall	F1 micro
Random Forest Ajustado binario	0.670533	0.658465	0.664444
Random Forest Ajustado multilabel	0.760734	0.474013	0.584657

Cuadro 5.9: Resultados micro-averaged de los modelos finales ajustados de cada enfoque

La clasificación binaria ha tenido un rendimiento mayor que la clasificación multilabel aunque en ambos casos el algoritmo con mejores resultados haya sido el mismo. Mientras que el RF binario ha obtenido un valor de F1 micro de 0.664444 y una precision y un recall de 0,67 y 0,65 respectivamente, el RF multilabel tiene un valor de F1 micro de 0.584657 y valores de precision y recall de 0,77 y 0,47 respectivamente. Por tanto, el enfoque seleccionado para hacer futuras clasificaciones de revisiones será el conjunto de clasificadores de RF utilizado en la clasificación binaria. Además, observando las matrices de confusión de los random forest de ambos enfoques vemos claramente que el conjunto de árboles del enfoque binario sí es capaz de predecir intenciones como point of view o clarification con cierto éxito mientras que el multilabel no. Está claro que el imbalance del dataset juega un papel negativo en el enfoque de un sólo modelo multilabel mientras que en el enfoque binario al poder realizar over-sampling a nivel individual el problema es solventado en cierta parte.

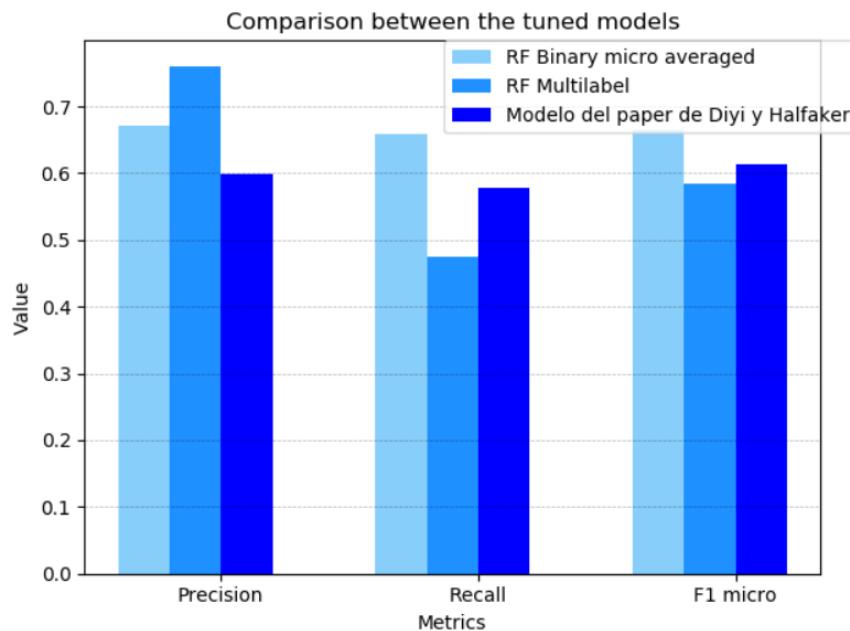


Figura 5.27: Gráfica con resultados de modelos finales

En definitiva y como se puede ver claramente en la figura 5.27, el random forest que utiliza el enfoque de clasificación binaria supera a los demás claramente en recall y f1 micro, solo estando ligeramente por debajo en precisión frente al random forest multilabel. Por otro lado, el objetivo de esta subsección se ha logrado pues se ha conseguido obtener mejores resultados de precisión, recall y f1 micro que aquellos obtenidos en [21], mostrados en la leyenda bajo 'Modelo del paper de Diyi y Halfaker' con valores de 0,613 en F1 micro, 0,578 en recall, 0,599 en precisión y un valor de accuracy del 54 %.

Capítulo 6

Minería de Procesos

Tal y como se explica en el capítulo 4 la minería de procesos se compone por un conjunto de técnicas de minería que permiten extraer información de logs de eventos para descubrir, monitorizar y mejorar procesos[5]. Esta información puede ser analizada para tomar forma de decisiones de negocio o estratégicas para optimizar los procesos o simplemente para conocer el propio funcionamiento de los mismos. Para ello se hace uso de logs de eventos. Un log de eventos se compone de información proveniente de bases de datos, transacciones... y pueden considerarse una colección de secuencias de eventos ya que se asume que es posible guardar secuencialmente eventos de modo que cada evento denote una actividad (evento) y esté asociado a un caso particular (traza).

Sin embargo, sus aplicaciones para incrementar la eficiencia de un proceso no es lo que lo hace interesante de cara al estudio objeto de este proyecto, si no el propio descubrimiento de los procesos inherentes a la edición y evolución de los *artículos destacados* en comunidades de conocimiento colaborativo abiertas como la Wikipedia española. De nuevo, *artículos destacados* son aquellos que han sido catalogados como referente de calidad, es decir, de 'los mejores artículos de Wikipedia'. En la Wikipedia Española, esto supone sólo un total de 1127, el 0.07 % del total.

Así el objetivo de usar la minería de procesos aplicada a las revisiones de los artículos en comunidades de conocimiento colaborativo es descubrir y analizar el flujo de trabajo que siguen los propios artículos desde su creación hasta la fecha actual en caso de que este exista así como los posibles procesos existentes dentro del comportamiento de los propios editores.

La herramienta escogida para aplicar las técnicas de minería de procesos es ProM tools 8.6 tal y como ha sido comentado previamente durante el capítulo 3 y 4.

El proceso a seguir será el siguiente:

1. Obtención de datos: En este apartado se explicará el proceso a seguir para obtener el log de eventos de cara a aplicar las técnicas de minería de procesos. Se determinará de manera clara cuales son los log de eventos que serán utilizados durante las demás secciones del capítulo.
2. Transformación a XES: Se centra en la conversión del log de eventos al formato estándar

de la minería de procesos: XES determinando de manera clara su estructura y utilidad en las futuras secciones.

3. Análisis exploratorio de los datos: Antes de aplicar las técnicas de minería se estudiará el log de eventos mediante la realización de diferentes gráficas para poder analizar visualmente el conjunto de datos de partida y así obtener información acerca de la organización y estructura del log.
4. Análisis a nivel artículo: Se aplicará la minería de procesos de cara a descubrir los procesos existentes dentro de la evolución de los artículos de Wikipedia.
5. Análisis a nivel editor: Esencialmente se hará lo mismo que en el análisis a nivel de artículo pero cambiando el foco de estudio. En lugar de descubrir los procesos inherentes a la evolución de los artículos estudiaremos aquellos seguidos por los propios editores en sus sesiones de trabajo.

6.1. Obtención de datos

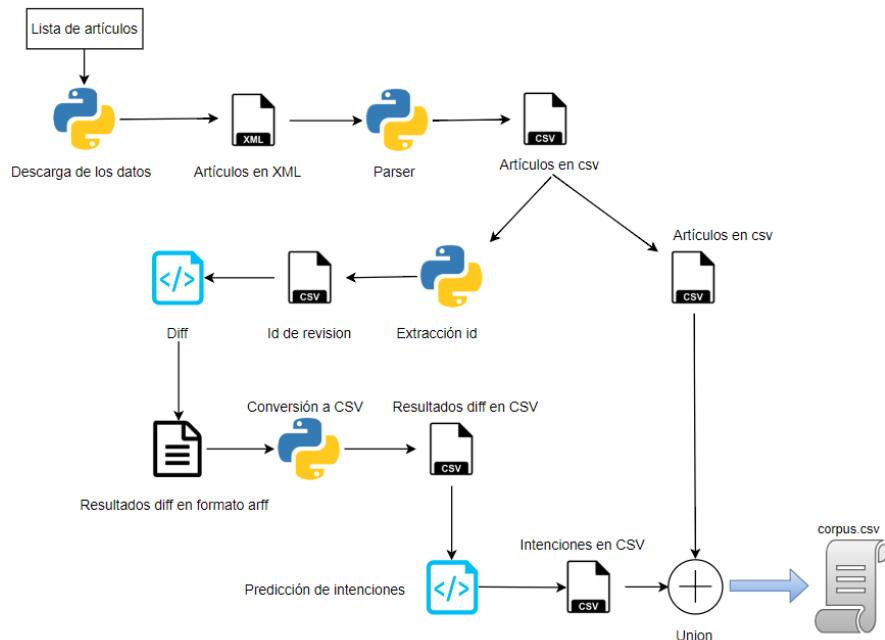


Figura 6.1: Proceso seguido para la descarga y preparación de los datos para realizar minería de procesos

Durante el capítulo 5 se ha explicado el proceso seguido desde la descarga del historial de revisiones un artículo de Wikipedia hasta la predicción de las intenciones semánticas tras cada revisión según la taxonomía de Diyi Yang et al[21]. Además de retomar su explicación aplicado a resolver el objetivo de esta sección, en la figura 6.1 podemos observar un esquema

de la estructura del proceso a seguir.

1. El proceso comienza con la descarga de un historial de revisiones de un artículo en Wikipedia, realizado mediante el script de descarga¹. Esto, genera un archivo en formato XML como el de la figura [insertar ejemplo de XML]. En este caso, se han seleccionado los siguientes *artículos destacados* de diferentes categorías aleatoriamente de entre todos los disponibles en Wikipedia:

- Leche
- Odín
- Homer Simpson
- Bifaz
- Angkor Wat
- Airbus A380
- Ácido desoxirribonucleico
- Tierra

El motivo de la selección de artículos de diferentes categorías es para tratar de descubrir procesos a nivel general dentro de la escritura colaborativa. Variedad de temas da lugar a variedad de editores y por tanto de estilos.

2. Una vez contamos con los XML de cada historial de revisiones de artículos descargado hacemos uso del parser proporcionado por Abel Serrano Juste² que los transforma en archivos CSV dotados de id y título de artículo y revisión, timestamp e id y nombre del editor además del conjunto de bytes afectados en la revisión.
3. Con estos archivos CSV que almacenan la información de los artículos seleccionados extraemos cada id de cada revisión³ y ejecutamos el conjunto de scripts realizados por Diyi Yang et al⁴ que realiza un diff de cada revisión sucesiva en artículo generando un archivo en formato arff por artículo compuesto de 207 atributos y el id de revisión. Estos atributos hacen referencia a las diferencias encontradas en cada comparación y son utilizados para la posterior predicción de las intenciones tras cada revisión.
4. Para predecir las intenciones en base a los archivos arff generados por el diff, primero, debemos cambiar su formato a CSV de nuevo lo cual realizamos mediante el uso de un script de conversión⁵.

¹wiki_dump_downloader.py (11)

²wiki_dump_parser.py (11)

³Haciendo uso del script revision_id_extractor.py (11)

⁴Más en 11 y https://github.com/diyiy/Wiki_Semantic_Intention

⁵arffToCsv.py (11)

5. Llegados a este punto, hacemos uso del modelo seleccionado en la sección 4.5 del capítulo 5 Desarrollo para predecir las intenciones tras cada revisión del conjunto de CSV dotados de los 207 atributos obtenidos en el diff⁶. Estas intenciones, al mismo tiempo que son predecidas, son añadidas a los archivos CSV resultado de hacer uso del parser de Abel durante el punto 2 de esta lista.

Se cuenta por lo tanto con 8 CSV, uno por cada artículo seleccionado que contienen id y título de artículo y revisión, timestamp e id y nombre del editor además del conjunto de bytes afectados en la revisión y la intencionalidad tras la misma. El último paso es unir todos los archivos en uno solo denominado **corpus.csv** y será el fichero base de todo este capítulo.

Para realizar el análisis a nivel artículo se hará uso de este mismo **corpus.csv** en su totalidad pues el objetivo es descubrir los procesos ocultos tras la elaboración de artículos en Wikipedia por lo que un filtrado podría alterar los resultados.

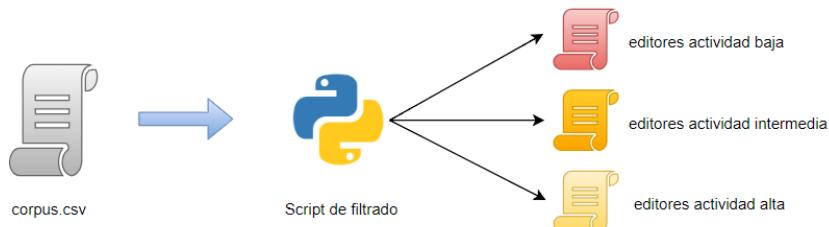


Figura 6.2: Esquema del filtrado del corpus para el análisis a nivel editor

En cambio, en la sección análisis a nivel de autor se van a generar 3 ficheros diferentes filtrando por cantidad de revisiones por revisor. Como observamos en el esquema 6.2 mediante un script de filtrado⁷ generaremos 3 archivos siguiendo la siguiente pauta:

- Editores de actividad baja: engloba solo revisiones realizadas por autores con menos de 5 revisiones.
- Editores de actividad intermedia: contiene exclusivamente revisiones hechas por autores que han realizado entre 5 y 50 revisiones.
- Editores de actividad alta: Compuesto solo por aquellos con más de 50 revisiones.

De este modo a lo largo de las sucesivas secciones estos archivos serán referidos de la misma manera: fichero de revisiones de editores con actividad baja/intermedia/alta.

El motivo de este filtrado es el siguiente: durante la sección análisis a nivel de autor el foco está puesto en el editor como individuo, por lo que para analizar un grupo específico de editores es necesario poder filtrar aquel comportamiento que no nos interesa pues solo añade ruido al análisis. Así se puede determinar si en función del número de ediciones existen

⁶generate_predictions.py (11)

⁷corpus_filter.py (11)

comportamientos diferentes.

Las características del corpus y sus ficheros derivados dan lugar a que puedan ser considerados log de eventos pues reunen todas las características tal y como se detalla en el capítulo 4 ¿Qué es la minería de procesos? Sin embargo, aún se encuentran en formato CSV por lo que para ser utilizados en la minería de procesos aún deben ser convertidos al formato estándar: XES.

6.2. Transformación a XES

De cara a realizar minería de procesos con ProM es necesario el uso del formato estándar del IEEE para los logs de eventos[20]. La conversión CSV a XES se realiza mediante la herramienta de conversión proporcionada por el propio ProM.

El formato XES se encontraba compuesto por eventos que refieren una actividad asociadas a un caso particular (una traza). Además cada evento puede contar con atributos como timestamp indicando el tiempo en el que sucedió dicho evento o org:resource que representa el actor que ejecutó tal evento. En este caso, se ha realizado la conversión al formato XES de todos los ficheros obtenidos en la sección anterior siguiendo el esquema de la figura 6.3.

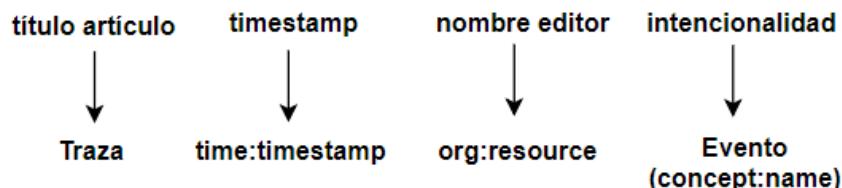


Figura 6.3: Representación de los atributos del formato CSV tras la conversión XES

De esta manera nos encontramos con que hemos determinado cada traza como cada artículo y cada evento como cada intención, de modo que cada revisión está determinada por esta. Además, hemos seleccionado el nombre del editor como actor ejecutor de cada evento, es decir org:resource. El timestamp se ha utilizado para determinar la hora a la que se realizó la edición.

Así, contamos ya con el log de eventos **corpus** así como los log de eventos generados a partir del mismo en función de las revisiones de los editores en formato XES.

6.3. Análisis exploratorio de los datos

Antes de comenzar a aplicar la minería de procesos, se va a realizar un análisis exploratorio del log de eventos **corpus** en formato XES para obtener una mayor información del mismo con la esperanza de que esto pueda traer beneficios a la hora de realizar un análisis de los resultados obtenidos mediante la minería.

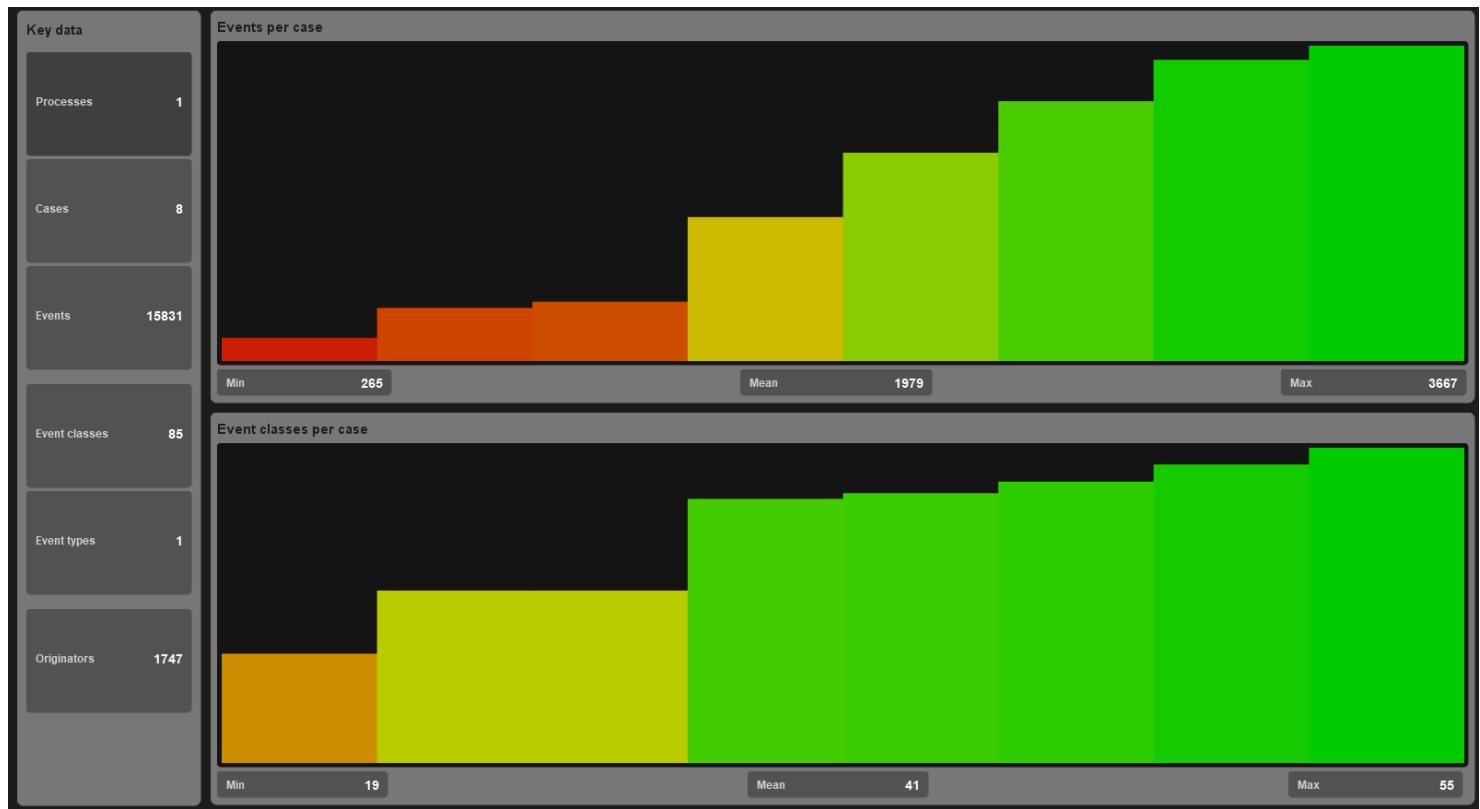


Figura 6.4: Corpus en formato XES

En la imagen 6.4 observamos la representación del archivo XES del corpus en ProM. El log de eventos corpus había sido dividido en casos (trazas) en función de cada artículo y se había constituido cada revisión como un evento determinado por cada intención. La figura puede descomponerse en tres secciones:

- Margen izquierdo: contiene datos claves del log de eventos:
 1. Processes: Indica el número de procesos existentes, en este caso solo 1 pues solo hacemos uso de un corpus.
 2. Cases: Número de casos, dado que hemos separado cada caso por cada artículo, tenemos 8 casos.
 3. Events: Número de eventos, en este caso revisiones, determinadas por su intención tal y como hemos determinado al transformar a formato XES. Contamos con 15831 revisiones.
 4. Event classes: Clases de eventos diferentes. Al venir determinados por su intención, esto indica el tipo de intencionalidades diferentes observadas. En este caso se observan 85 tipos de intencionalidad diferentes. Teniendo en cuenta que la taxonomía de intenciones cuenta con 13 intenciones que pueden combinarse entre si mismas, el valor máximo de esta casilla sería:

$$\sum_{n=1}^{13} \binom{13}{n}$$

5. Event types: Hace referencia a los tipos de evento. En este caso sólo hemos determinado cada evento (revisión) por su intencionalidad por lo que el valor de esta casilla es 1. Si se determinasen, por ejemplo, mediante intencionalidad y org:resource su valor sería de 2.
 6. Originators: Representa en este caso la cantidad de resources diferentes, es decir, la cantidad de editores. Es decir, contamos con 1747 editores diferentes repartidos a lo largo de 8 artículos.
- Gráfica superior: Aquí observamos un gráfico de barras que representa el número de eventos por caso. Esto es, el número de revisiones por artículo. Claramente hay mucha disparidad entre los 8 artículos, con el más pequeño de todo compuesto solo de 265 revisiones y el mayor con 3667. No obstante, todos los artículos pertenecen a la categoría de *artículo destacado* lo que podría denotar que el número de revisiones no es un indicador fiable de la posible calidad de un artículo.
 - Gráfica inferior: De nuevo se trata de un gráfico de barras que representa la cantidad de clases de eventos en cada artículo. Así, cada barra de las 8 existentes muestra el número de intenciones diferentes tras el conjunto de revisiones en su respectivo artículo. Aquí también se encuentran disparidades, con el artículo con menor variedad de intencionalidades registrando sólo 19 diferentes en contraposición con el artículo con mayor variedad: 55.

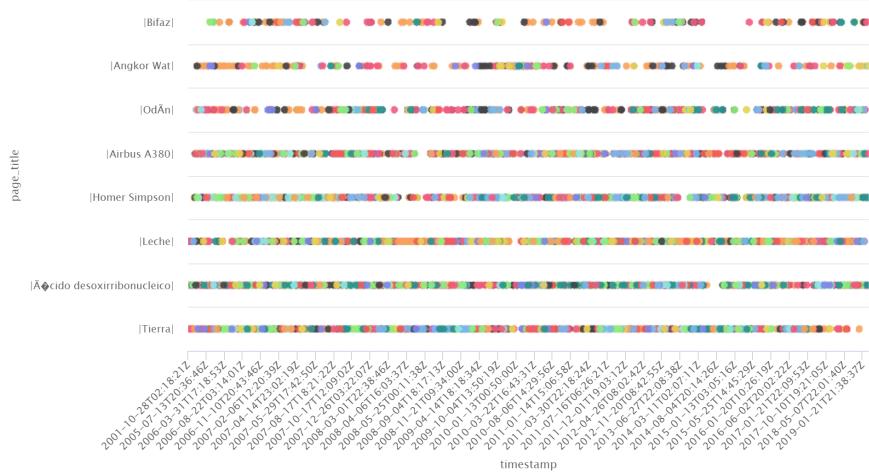


Figura 6.5: Revisiones a lo largo del tiempo por artículo

En la gráfica 6.5 podemos ver las revisiones de los artículos a lo largo de su existencia. Todos los artículos fueron creados antes de 2006, sin embargo, los más antiguos datan de 2001. Se puede observar como todos los artículos han sido editados sin descanso hasta el día de hoy pues se observan muy pocas interrupciones en general. El artículo Bifaz además de ser el último en crearse, es el que más 'parones' ha tenido en su historial de revisiones. Estos dos factores, son los que hacen que sea el artículo con menos revisiones del corpus: 265. A pesar de ello y al igual que los demás, se trata de un *artículo destacado*, por lo que podríamos extraer que la calidad del artículo no es directamente proporcional a la cantidad de revisiones que tiene, aunque para confirmarlo habría que hacer uso de un corpus de mayor extensión. Además,

CAPÍTULO 6. MINERÍA DE PROCESOS

se puede ver en el eje temporal de la gráfica que hay un salto temporal de 2001 a 2005. Esto, es debido a que las fechas son mostradas proporcionalmente en función del número de revisiones realizado entre cada período de tiempo representado en el eje. Teniendo en cuenta que la Wikipedia se fundó en 2001 no resulta sorprendente que entre 2001 y 2005 haya la misma cantidad de revisiones que en períodos de tiempo mucho más reducidos a partir de 2005 pues se observa en los datos que solo hay 247 revisiones antes de ese año a lo largo de todo el corpus.

Por otro lado, en la figura 6.6 se puede ver la evolución de las intenciones del conjunto del corpus en el tiempo. Intenciones como Wikification, elaboration, copy-editing y vandalism evolucionan rápidamente de modo lineal frente a todas las demás intenciones con menos frecuencia de aparición. Además, en los últimos años se observa como la frecuencia de aparición de fact-update cambia y empieza a crecer a mayor velocidad. De este modo las intenciones con mayor frecuencia como elaboration o copy-editing o wikification son asociadas con alta calidad de los artículos[21] lo cual concuerda con el estatus de *artículo destacado* de los artículos que conforman el corpus.

Otras intenciones como re-factoring y point of view se observa que empiezan a aparecer pasado un tiempo, indicando que se trata de una intención propia artículos con cierto recorrido.

El vandalismo en Wikipedia, suele caracterizarse por su persistencia en el tiempo[14]. De este modo, también en nuestro caso el vandalismo es persistente en el tiempo, indicando quizás que la madurez de un artículo no tiene relación con el nivel de vandalismo que recibe.

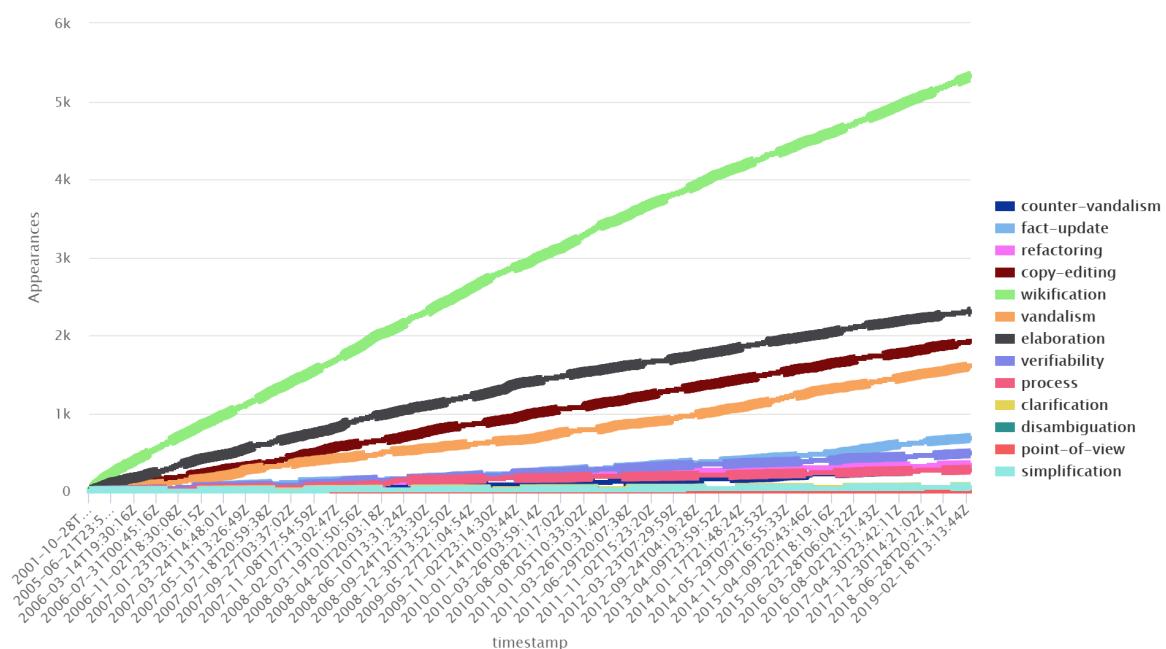


Figura 6.6: Intenciones a lo largo del tiempo en el corpus

En cuanto a los eventos (revisiones), hemos visto que hay 85 clases diferentes, para ver su

representación en el corpus, se ha creado una gráfica tipo WordCloud que puede observarse en la figura 6.7.

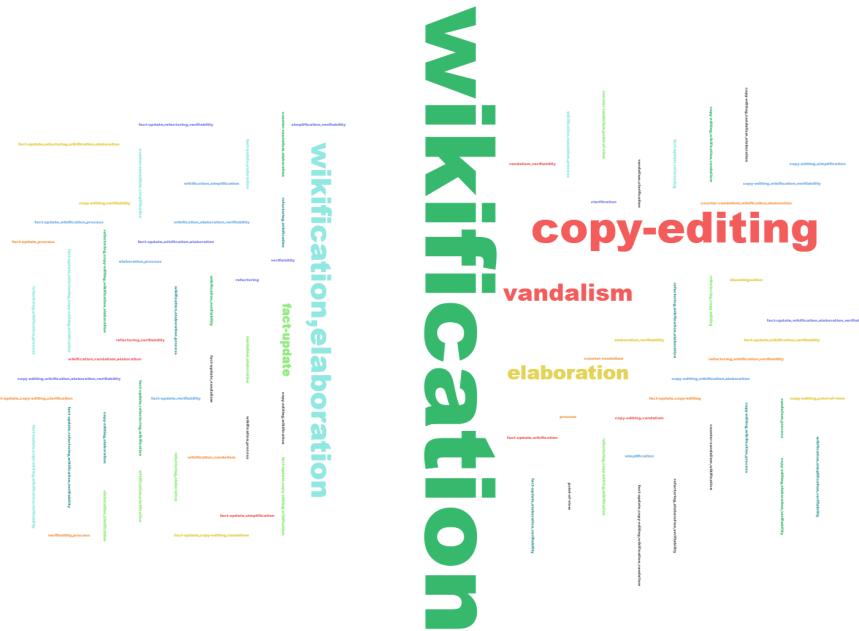


Figura 6.7: Representación de las distintas intenciones

Entre las diferentes intenciones posibles en una revisión, la que tiene mayor frecuencia de aparición es sin duda wikification, seguido de copy-editing, wikification+elaboration, elaboration y vandalism. La frecuencia de aparición de Wikification es considerablemente más alta que la de cualquier otra intención: 3569 apariciones en 15831 revisiones sin tener en cuenta aquellas revisiones en los que aparece combinada con otras, además, aparece constantemente desde el inicio de un artículo y durante toda su evolución. Esto, podría indicar un problema en sí mismo con las herramientas de Wikipedia a la hora de escribir y dar formato al artículo. No parece especialmente óptimo que constantemente se tenga que estar editando el formato de los artículos a pesar de que según avanza la calidad de un artículo, aumenta la importancia de determinadas intenciones, como wikification y con ello su frecuencia[21].

Por último, están los editores. A lo largo de todo el corpus, se encuentran 1747 editores diferentes. Sin embargo, solo 869 han realizado más de 1 edición y 380 más de 5, así, pocos autores realizan grandes números de ediciones, es decir, la mayoría de editores son editores 'casuales' o de baja actividad. En gran parte las revisiones han sido realizadas por usuarios anónimos, representando aproximadamente un tercio del total de revisiones. Los usuarios individuales representan una fracción pequeña del total de revisiones a nivel individual, sin embargo, existen grandes diferencias entre ellos. Así, la mayoría de editores son usuarios que han realizado un número reducido de revisiones.

En resumen, nos encontramos con un event log compuesto de 8 artículos y 15831 revisiones formado por 85 combinaciones diferentes de intenciones, siendo los artículos diferentes entre

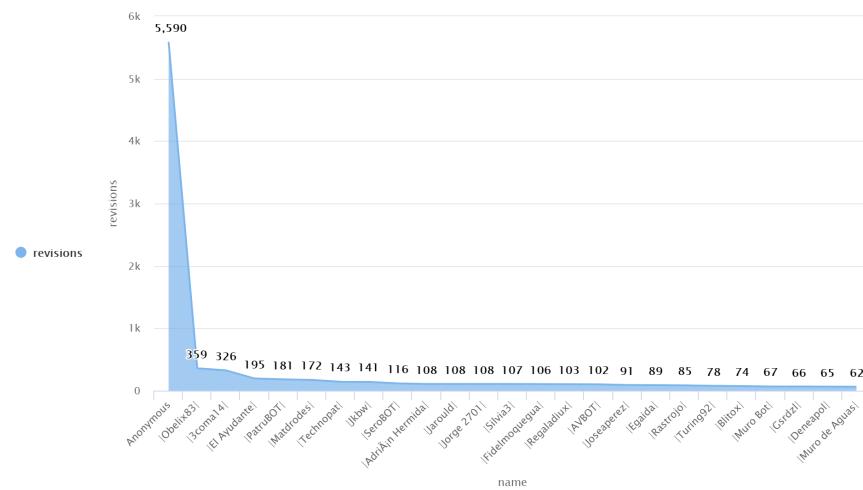


Figura 6.8: Número de revisiones por editor

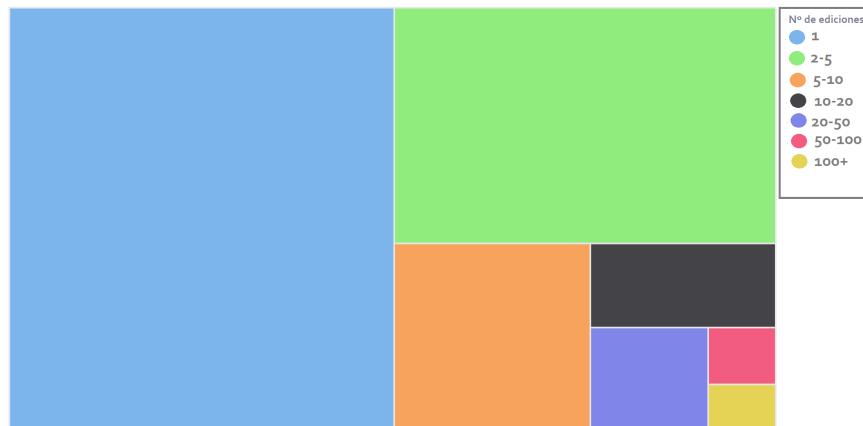


Figura 6.9: Número de editores según cantidad de ediciones realizadas

ellos con la calidad y la antiguedad como nexo de unión. Por otro lado, se cuenta con 1747 editores diferentes dando lugar a una gran variedad de datos para la aplicación de minería social a nivel de editor. Ahora que se conoce el **corpus** y los datos que lo componen, se procede a aplicar la minería de procesos. Para esto, se utilizarán los enfoques previamente introducidos: a nivel artículo y a nivel editor.

6.4. Análisis a nivel artículo

El objetivo de esta sección es realizar un análisis mediante la utilización de técnicas de minería de procesos para intentar estudiar los procesos inherentes a la propia evolución de los artículos en la Wikipedia española.

Así, partimos del mismo archivo XES (**corpus**) que ha sido analizado en la sección previa.

El primer paso de todos es escoger el algoritmo minero adecuado para nuestro tipo de log de eventos, que en este caso será el Minero Heurístico. De entre las opciones disponibles en ProM, se ha hecho uso de el minero heurístico ya que tal como se describe en la sección 5.2 de este documento es un algoritmo práctico que puede utilizarse para explicar el comportamiento principal registrado en un log de eventos además de lidiar muy bien con el posible ruido de los datos[19]. Además, ha sido utilizado con éxito en ámbitos como la escritura colaborativa[18] y en la educación[7], similares en esencia al nuestro, especialmente la escritura colaborativa ya que es precisamente en lo que se basan las comunidades de conocimiento colaborativas como las Wikis.

Aplicando el minero heurístico con los parámetros por defecto al **corpus**, obtenemos una red heurística que es transformada en red de petri automáticamente por ProM, con un fitness de 0.7996. Este valor, representa en una escala de 0 a 1 la porción de los eventos observados en el log de eventos que puede ser reproducida en la red. De este modo, se trata de un valor razonablemente alto para el contexto en el que nos encontramos. Sin embargo, es una red de petri muy compleja como para analizar a ojo (6.10).

Para poder simplificar esta red de petri de modo que se pueda extraer información de ella de modo visual, se aplica una técnica de aprendizaje no supervisado: clustering. El clustering consiste en agrupar un conjunto de objetos de modo que cada grupo esté compuesto por objetos similares. En este caso, el objetivo de realizar clustering en el **corpus** es el de agrupar las revisiones por similitud en base a los procesos que se observen. Esto es realizado de modo trivial mediante la herramienta incluida en ProM 'Discover clusters'.

Una vez generados que se ha aplicado 'Discover clusters' al **corpus**, se utiliza tanto el conjunto de clusters obtenidos como el propio **corpus** y se aplica 'Discover using Decomposition' que hace uso del minero heurístico teniendo en cuenta la agrupación realizada generando una versión descompuesta en diferentes partes de la red de petri generada previamente (6.10).

De esta manera, hemos logrado simplificar la red de petri anterior en varias redes diferentes de menor tamaño y mayor legibilidad. Así, contamos con (i) la red de petri general (6.10), (ii) la red de petri descompuesta 1 (6.11), (iii) la red de petri descompuesta 2 (6.12) y (iv) el conjunto de mini redes de petri descompuestas 3 (6.13)

Claramente puede verse que aún siendo más simples, siguen siendo complejas de analizar para el ojo humano tanto la red 1 (ii) como la red 2 (iii). Esto, no es sorprendente puesto que el proceso que siguen los artículos durante su evaluación es esencialmente anárquico en cuanto a la frecuencia o la cantidad de usuarios que colaboran, dando lugar a procesos potencialmente diferentes entre artículos. No obstante, a pesar de ser un corpus de 8 artículos diferentes compuesto de muchos editores diferentes y con un recorrido muy largo en el tiempo, la red es razonablemente comprensible. Se observa que los procesos están llenos de bucles, por lo tanto son naturalmente y de modo inevitable iterativos, no hay una sola cadena de ediciones a seguir si no muchas posibilidades diferentes. Esencialmente podría traducirse como que no existe un proceso unificado y real que se siga en Wikipedia, lo cual es esperado pues surge del conjunto de trabajo de muchos usuarios potencialmente sin colaboración explícita entre ellos. Sin embargo, observamos diferencias entre las redes simplificadas.

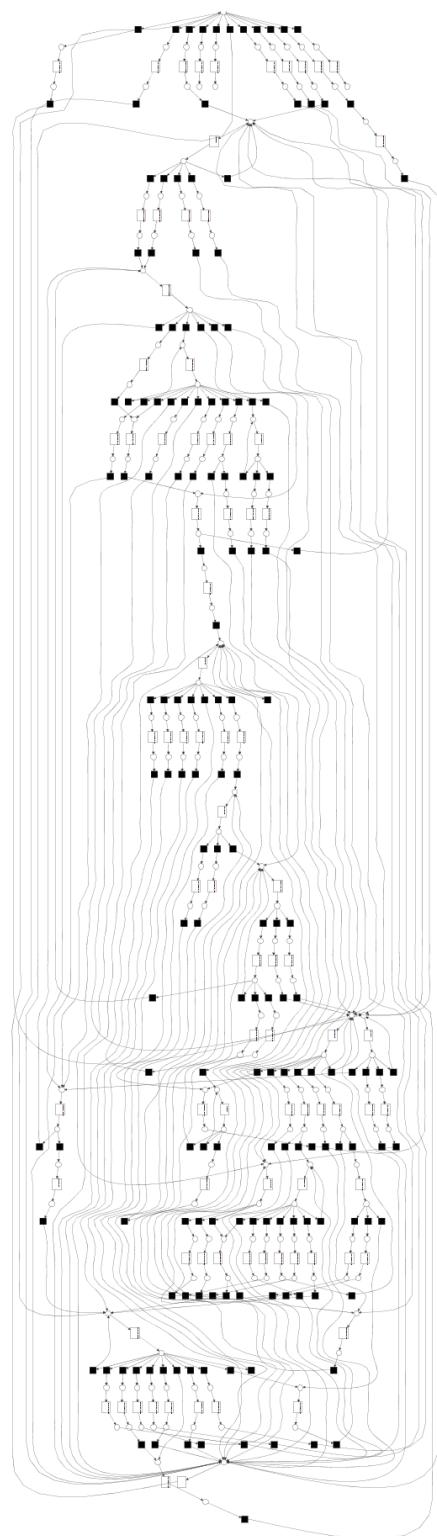


Figura 6.10: Petri net obtenida con Minero Heurístico

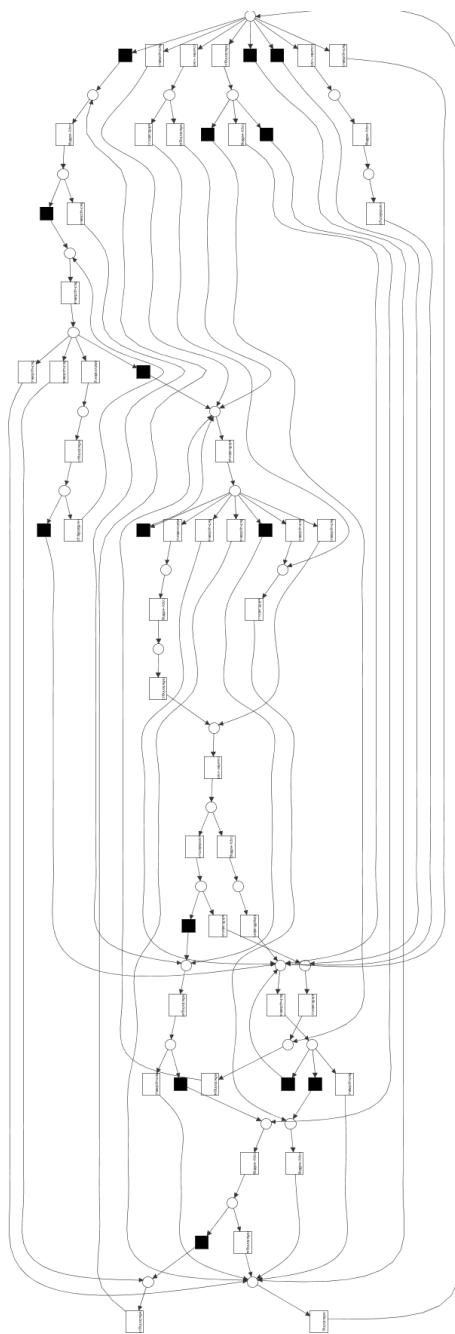


Figura 6.11: Petri net del proceso de edición tras su descomposición 1

6.4.1. Análisis de la red de Petri descompuesta 1

Como vemos en la red de petri de la imagen 6.11 a grandes rasgos se observa un proceso con mucha interconexión y de gran complejidad. Para facilitar su análisis se ha dividido la red

en tres fragmentos para poder observarlos más en detalle. Esto está además motivado en parte por que el inicio tiene interes especial pues ver si hay un inicio específico dentro del proceso podría ayudar a identificarlo al mismo tiempo que el final aporta utilidad para descubrir si existen puntos finales en el proceso o es abierto y por tanto potencialmente 'infinitamente' iterativo.

El inicio de la red de petri 1 se puede ver en la figura 6.14. A simple vista resalta la cantidad de intenciones complejas que se ven en el proceso, siendo en todos casos intenciones combinadas. A priori se observa como estos posibles inicios del proceso van marcados por intenciones como refactoring+copy-editing, fact-update+refactoring+verifiability, fact-update+refactoring+wikification+elaboration...entre otros. Estas intenciones observadas tendrían sentido en etapas tempranas de un artículo donde aún está todo por hacer, lo cual podría explicar que se encadenen muchas revisiones con intenciones combinadas complejas. Durante el paper 'Turbulent Stability of Emergent Roles' ([6]) Daxenberger et al. determinan que durante las etapas tempranas de desarrollo de un artículo, el 60% de sus editores toman el rol de 'All round contributor' lo que significa que son usuarios todo terreno que hace un poco de todo. Esto sin embargo se ve reforzado por los resultados observados en esta sección de la red de petri donde las intenciones a menudo contienen elementos de todo tipo como refactoring+copy-editing+fact-update+wikification.

En la figura 6.15 vemos la siguiente parte de la red de petri descompuesta 1. Más concretamente su sección intermedia. Las intenciones se van simplificando poco a poco, empezando a ser las combinaciones de 3 o más intenciones menos frecuentes. Esto es esperado pues según aumenta en antiguedad el artículo se reduce el porcentaje de revisores que representan 'All round-contributor' en pos de roles más específicos como copy-editors o layout-shapers o 'quick and dirty editors'[6]. Esto puede observarse en intenciones como refactoring+elaboration o simplification+verifiability o incluso vandalism+verifiability que puede estar relacionado con aquellos bajo el rol 'quick and dirty editors' ya que a veces sus rápidas ediciones son confundidas por vandalismo[6]. Sin embargo aún se siguen dando intenciones complejas pues se observan conexiones que vuelven al inicio de la red.

La figura 6.16 muestra el último tramo de la red de petri descompuesta 1. No aporta mucha información adicional respecto a la sección intermedia pues se observa un comportamiento similar en cuanto a las intenciones que se ven. Sin embargo vemos que no existe un punto final en el proceso, tras llegar al final la naturaleza iterativa de proceso de re-edición continuo de Wikipedia hace que exista la posibilidad de volver a otros puntos del proceso. Es decir, es un proceso abierto.

En definitiva aunque no podemos extraer demasiada información mas allá de observar ciertas similitudes con otros estudios ya realizados previamente al respecto de los roles de usuario y cómo se edita a lo largo de la vida útil de un artículo en Wikipedia pues de nuevo la natureza anárquica de la escritura colaborativa dificulta la existencia de un proceso claro y unificado.

6.4.2. Análisis de la red de Petri descompuesta 2

Como vemos en la red de petri de la imagen 6.12 y al igual que la red de Petri descompuesta 1 a grandes rasgos se observa un proceso con mucha interconexión y de gran complejidad.

Para facilitar su análisis se han seleccionado el inicio y el final de la red para poder observarlos más en detalle. Esto está además motivado en parte por que el inicio tiene interés especial pues ver si hay un inicio específico dentro del proceso podría ayudar a identificarlo al mismo tiempo que el final aporta utilidad para descubrir si existen puntos finales en el proceso o es abierto y por tanto potencialmente 'infinitamente' iterativo.

Como se ve en el inicio de la red en la figura 6.17 el inicio puede ir determinado por copy-editing+elaboration+verifiability o wikification+vandalism. Otra posibilidad es comenzar con Disambiguation sin embargo esto debe ser puesto en contexto. Dado que como se observa que hay caminos que vuelven al nodo inicial desde puntos más avanzados de la red, resulta normal asumir que esta intención no se da de modo inicial, si no en iteraciones futuras. En caso de comenzar realizando wikification+vandalism el flujo prosigue con copy-editing seguido de o bien de un flujo iterativo de fact-update o refactoring+wikification+elaboration o wikification+process. Es curioso ver las grandes diferencias existentes con el inicio de la otra red obtenida fruto de la descomposición (6.11). Aquí se observan intenciones más específicas fruto de editores más especializadas.

Por otro lado, observando el final de la red vemos que el patrón es el mismo, intenciones de mayor simplicidad al no estar combinadas, tareas de contra-vandalismo seguidas de fact-update o wikification, sin embargo el número de caminos que se observan es muy grande haciendo sumando cierta incertidumbre al proceso real que se pueda seguir.

6.4.3. Análisis del conjunto de mini redes de Petri descompuestas 3

En la figura 6.13 vemos un conjunto de redes formadas por un lugar de inicio y final y una sola transición compuesta por intenciones combinadas. Esto, no determina ningún comportamiento específico si no que es el resultado de la técnica de minería utilizada intentando de nuevo reproducir todo el comportamiento observado. Estas intenciones no han conseguido ser agrupadas dentro de ninguno de los anteriores procesos descubiertos por lo que se han convertido en redes en sí mismas para poder reproducir esa sección específica del log de eventos.

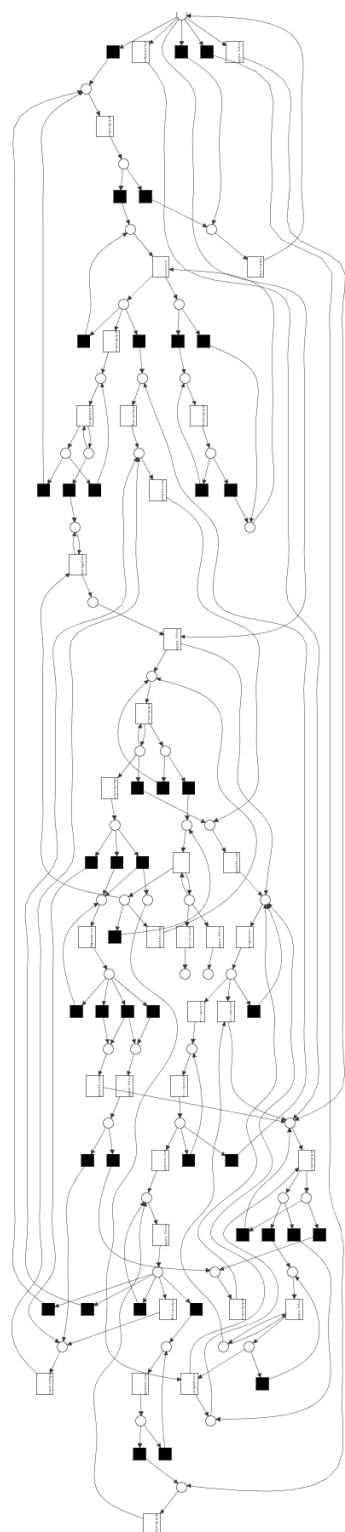


Figura 6.12: Petri net del proceso de edición tras su descomposición 2

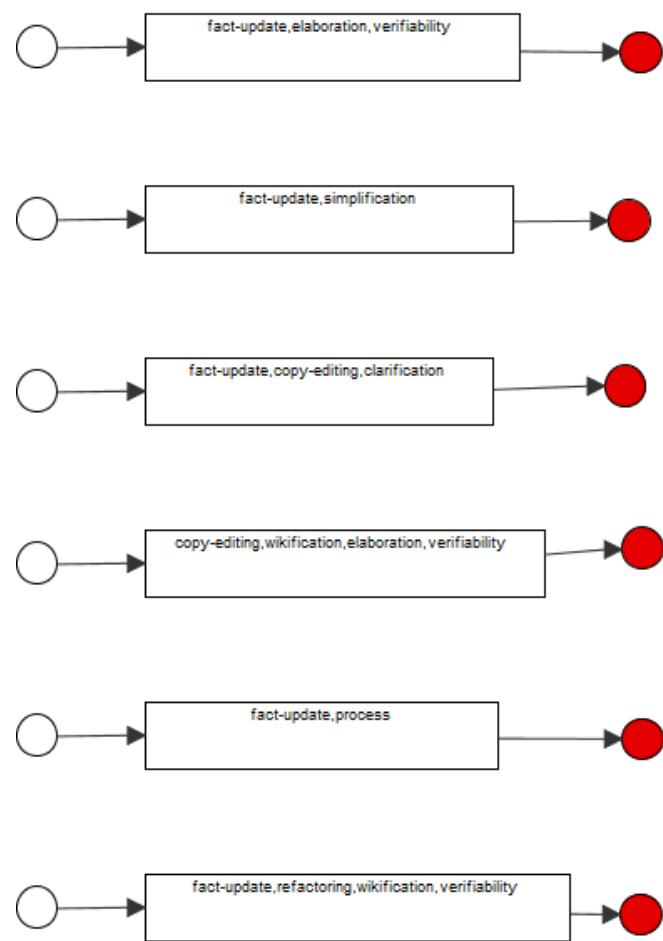


Figura 6.13: Redes de petri del proceso de edición tras su descomposición 3

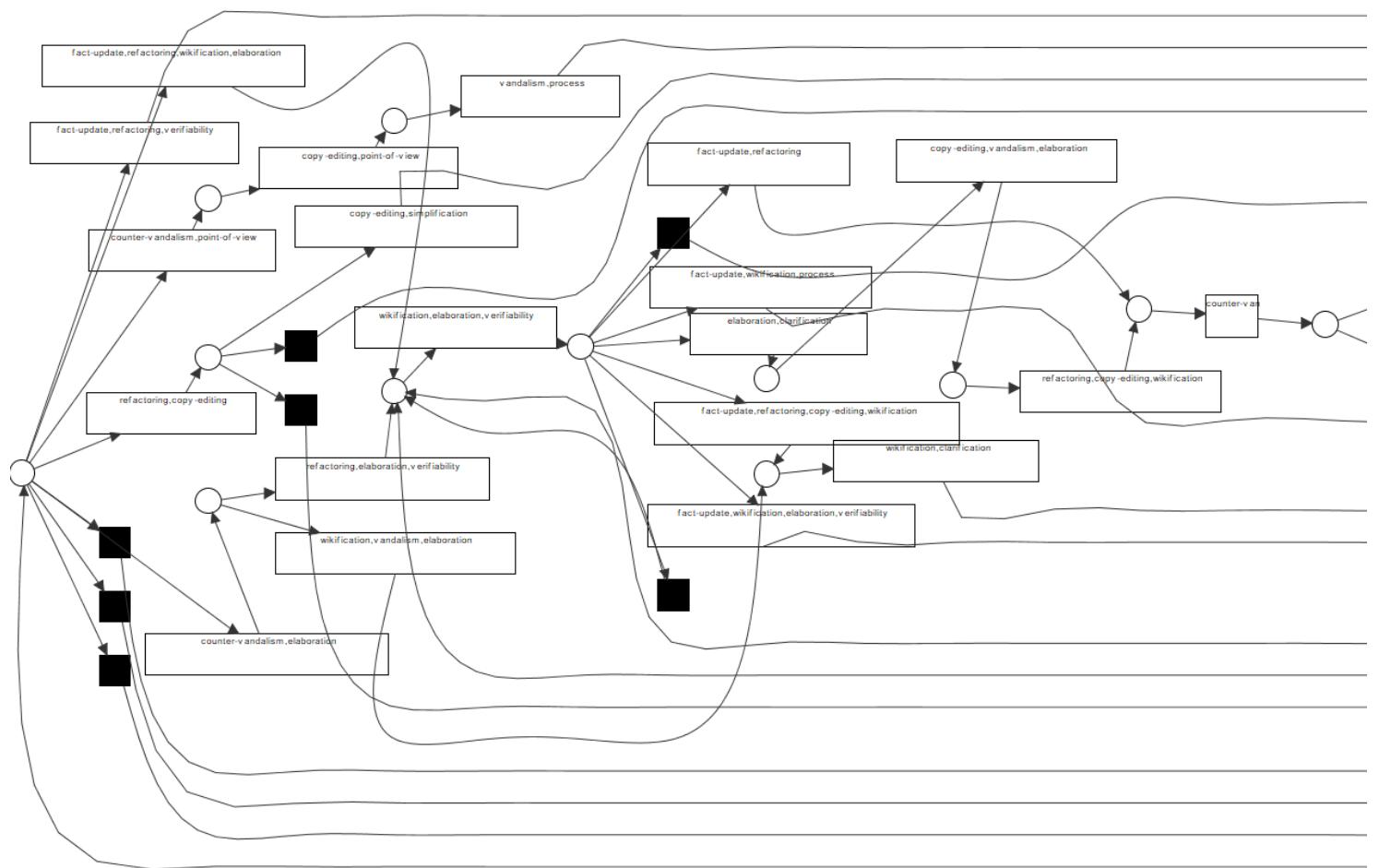


Figura 6.14: Inicio de la red de petri del proceso de edición tras su descomposición 1

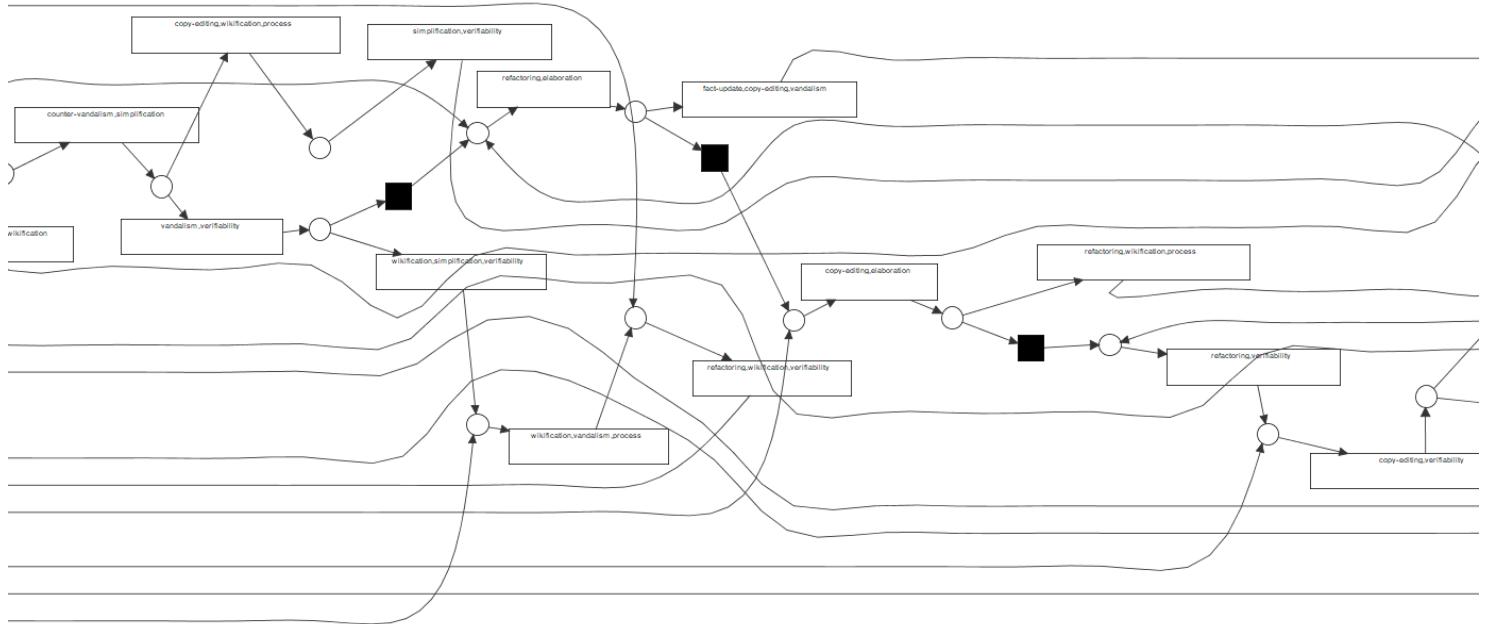


Figura 6.15: Sección intermedia de la red de petri del proceso de edición tras su descomposición
1

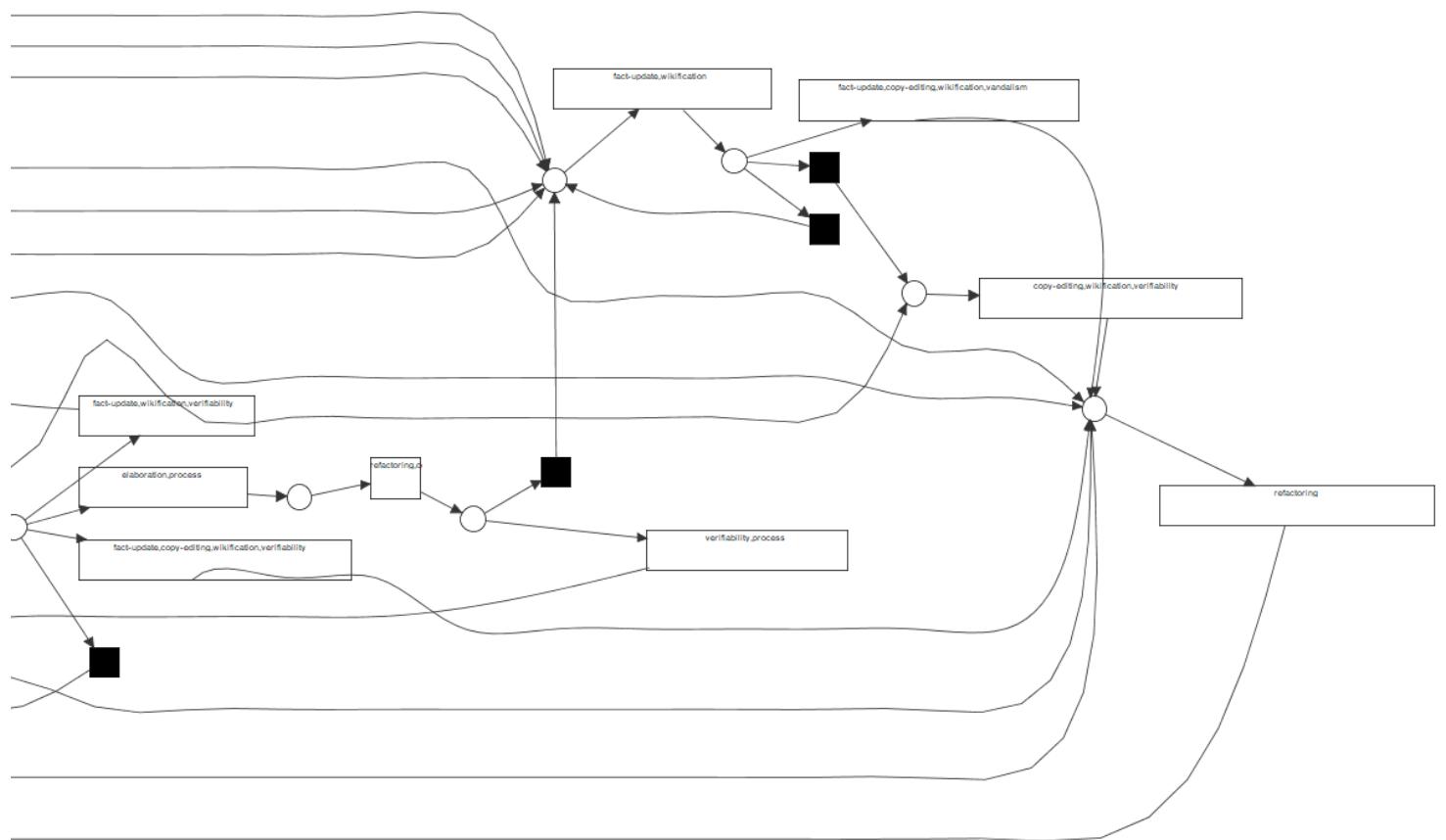


Figura 6.16: Final de la red de petri del proceso de edición tras su descomposición 1

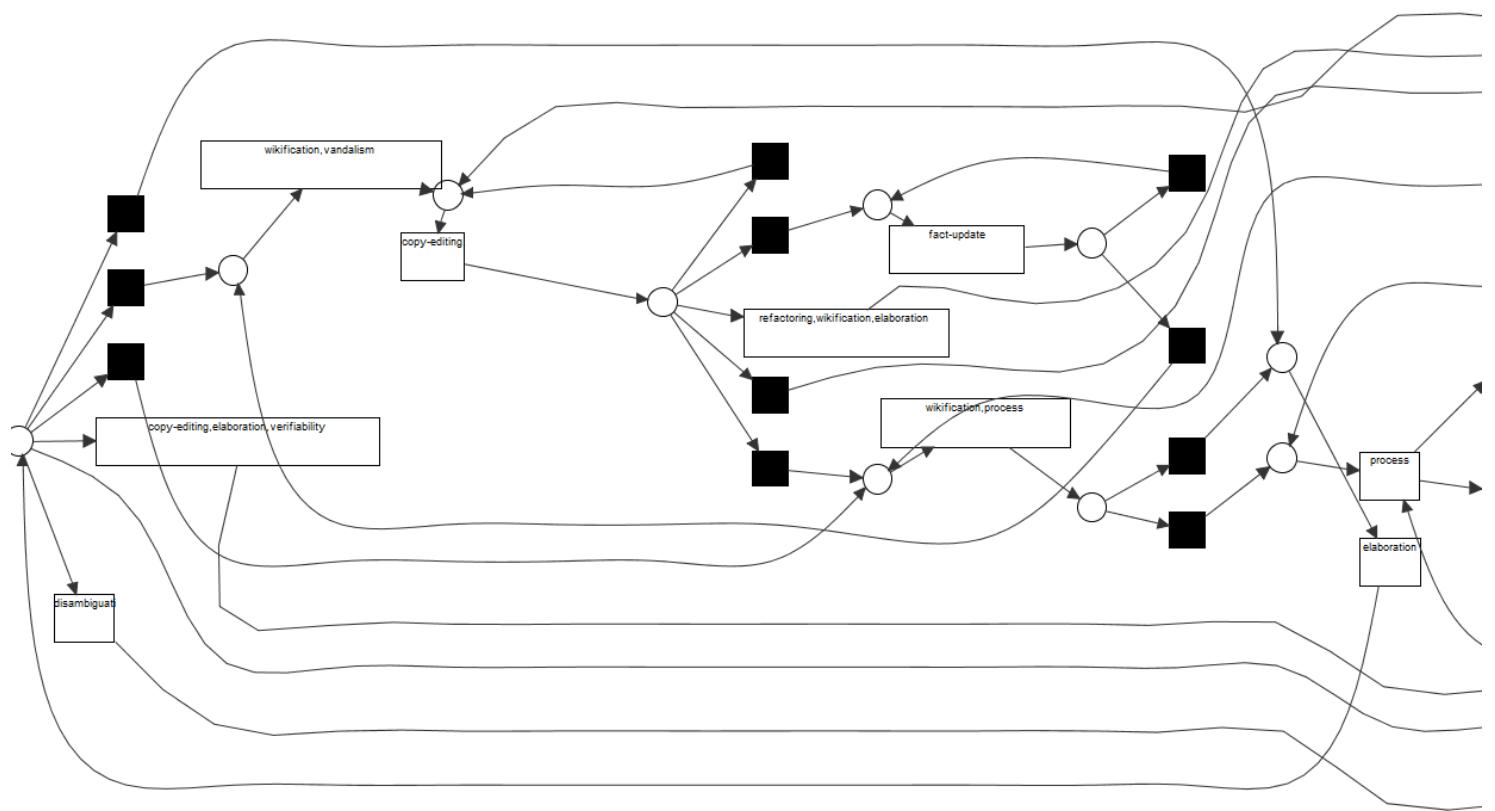


Figura 6.17: Inicio de la red de petri del proceso de edición tras su descomposición 2

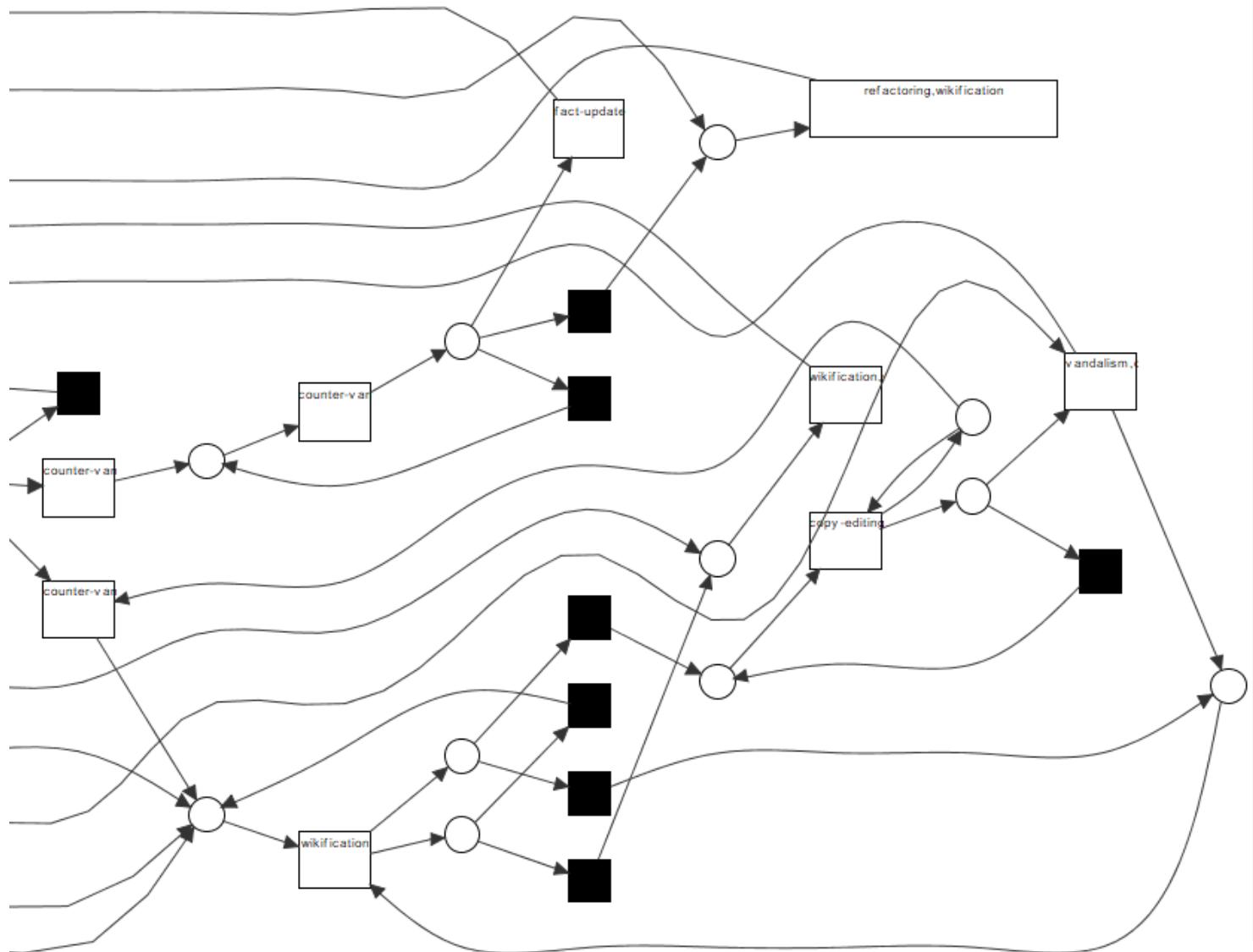


Figura 6.18: Final de la red de petri del proceso de edición tras su descomposición 2

6.5. Análisis a nivel editor

En esta sección se hará uso de los 3 log de eventos obtenidos a partir del **corpus** durante la sección Obtención de datos en formato XES. Es decir, los ficheros de revisiones de editores con actividad baja/intermedia/alta. De esta manera el objetivo es descubrir y estudiar los procesos seguidos por los propios usuarios en base a las intenciones de sus revisiones. La minería de procesos cobra especial interés de cara a este análisis pues podría ayudar a descubrir si los propios usuarios en función de su rol o actividad en la red siguen un patrón concreto de comportamiento más allá de editar eminentemente con un tipo u otro de intención.

Así los log de eventos a utilizar separan a los editores en 3 categorías⁸ para poder descubrir si existe un proceso común a cada categoría pero diferente a las demás. Por lo que al dividir de la siguiente manera evitamos el ruido de las demás categorías. Refrescando la definición dada previamente de cada log de eventos:

- Editores de actividad baja: engloba solo revisiones realizadas por autores con menos de 5 revisiones.
- Editores de actividad intermedia: contiene exclusivamente revisiones hechas por autores que han realizado entre 5 y 50 revisiones.
- Editores de actividad alta: Compuesto solo por aquellos con más de 50 revisiones.

A lo largo de las sucesivas secciones estos log de eventos serán referidos de la misma manera: fichero de revisiones de editores con actividad baja/intermedia/alta.

Estos 3 ficheros, sin embargo, aún no están listos para generar un proceso de determinar los diferentes caminos que siguen los editores. Necesitamos cambiar su estructura de modo que cada caso dentro del log de eventos venga determinado por cada editor en lugar de cada artículo. Para ello, hacemos uso de una herramienta de ProM cuyo objetivo es precisamente modificar cada log de eventos en base a org:resource, los editores, llamado 'Generate log from org:perspective'.

En esta sección además se aplicará el minero inductivo para el descubrimiento de cada proceso ya que permite intentar generar redes con fitness de 1/1.

Sin embargo, un fitness perfecto puede dar lugar a redes que engloban muchos comportamientos diferentes y que por lo tanto imposibilitan detectar comportamientos específicos, como las redes en forma de flor. Estas redes en forma de flor son aquellas que permiten cualquier tipo de comportamiento en base a un conjunto de actividades dado. En este caso las actividades serían los eventos, es decir, la intencionalidad que determina cada revisión.

6.5.1. Editores de actividad baja

Una vez que contamos con el **log de eventos de los editores con baja actividad**: solo los revisores que han realizado menos de 5 revisiones. Se aplica 'Generate log from org:perspective' y obtenemos un event log compuesto por 1367 casos y 2100 eventos. Es decir, tenemos 1367 editores diferentes que han realizado 2100 revisiones. Descomponemos en

⁸Haciendo uso de corpus_filter.py (11)

clusters del mismo modo que en la sección anterior haciendo uso de la herramienta 'Discover Clusters' y aplicamos 'Discover using Decomposition' en este caso seleccionando el Inductive Miner en su variante 'Perfect Fitness'. Así, obtenemos diferentes redes como vemos en las figuras 6.19,6.20,6.21,6.22,6.25

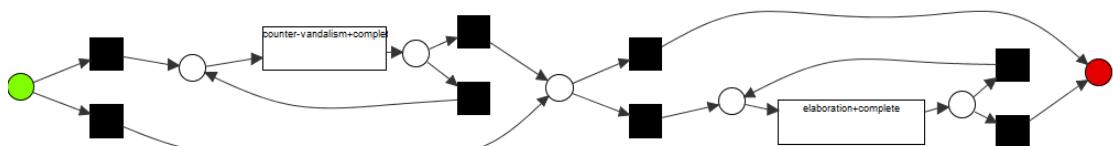


Figura 6.19: 1º Petri net del proceso seguido por los editores de baja actividad

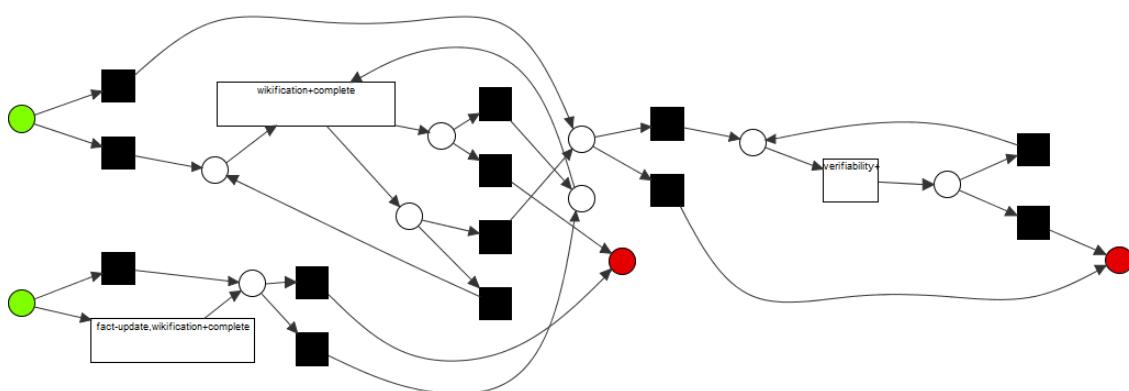


Figura 6.20: 2º Petri net del proceso seguido por los editores de baja actividad

Observamos que hay un total de 5 redes diferentes, 3 de ellas teniendo 2 puntos de inicio diferentes. Con esto, podemos decir que se han registrado 5 tipos de comportamientos observados entre los 1367 editores de baja actividad.

- La red 6.19 registra un flujo de ediciones donde se aplica counter vandalism y elaboration, sin embargo, dentro del flujo es posible que sólo se realice una de las dos o ninguna. Por este motivo, se puede inferir que los usuarios que siguen este workflow pueden ser englobados dentro de la categoría Watchdogs, usuarios que se dedican a hacer revert tras el vandalismo[6]. Por otro lado, elaboration es una categoría relacionada con la baja supervivencia de los nuevos usuarios[21], lo cual podría ser una explicación a la existencia de este flujo de trabajo dentro de los usuarios con menos de 5 revisiones. Otra posibilidad es que los usuarios tras realizar tareas de contra vandalismo, realicen pequeñas elaboraciones puntuales.
- La red 6.20 representa una actividad donde se edita con la intención de Wikification o Wikification+fact update donde ocasionalmente los usuarios realizaban después ediciones con intención Verifiability. En este caso, esto puede relacionarse con el rol de 'Content Shaper' o 'Layout Shaper' de la taxonomía de roles creada por Daxenberger et al en [6].

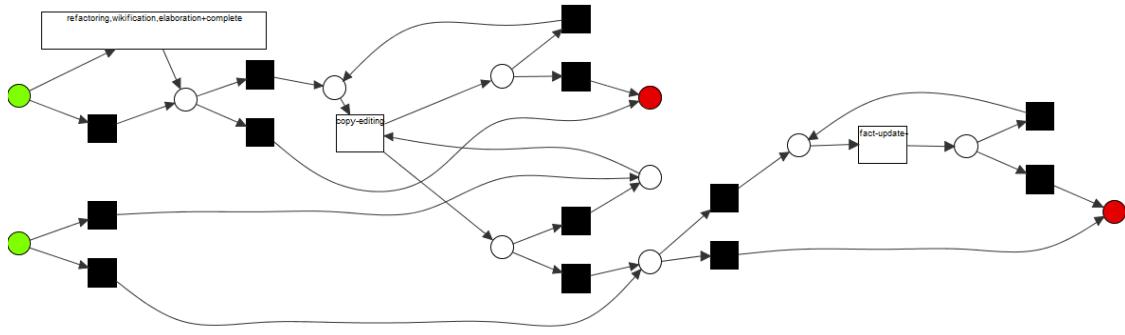


Figura 6.21: 3º Petri net del proceso seguido por los editores de actividad baja

- En cuanto a la red 6.21, nos encontramos con dos puntos de inicio diferentes, los usuarios que se engloban bajo esta forma de trabajar o bien comienzan con intenciones sofisticadas como refactoring+wikification+elaboration o bien copy-editing o fact-update. En general las diferentes posibilidades que ofrece esta red y el hecho de que por su estructura puede haber usuarios que solo hayan realizado una de estas tres intenciones hace difícil poder determinar un comportamiento específico de esta red.
- La red 6.22 no muestra mucha información respecto a ningún comportamiento real pues representa el resultado de la asociación de todos los usuarios que no han seguido un proceso específico, como aquellos que solo han realizado una revisión, entre las opciones posibles dentro de esta red, se encuentra el vandalismo. Esto es una de las desventajas comentadas anteriormente del minero inductivo. Sin embargo, se ve que entre toda esta red, hay 2 caminos que merece la pena analizar. Estos son los caminos que se observan en la parte superior e inferior.
 - En la figura 6.23 vemos la sección inferior de la red donde se observa que existe cierto posible comportamiento. Se ve un conjunto muy diverso de intenciones dentro de un bucle iterativo. Destaca como mientras que los caminos superiores cuentan con counter-vandalism+wikification en los inferiores se ve vandalismo en unión con otras intenciones como fact-update o verifiability. Este vandalismo puede ir de la mano de usuarios novatos que vandalizan al realizar ediciones sin ser ese su objetivo. Por otro lado la naturaleza iterativa de esta sección hace complicado extraer conclusiones respecto al flujo seguido por los propios usuarios.
 - En la sección superior ilustrada en 6.24 vemos un bucle de copy-editing+vandalism. Dado que copy-editing se centra en mejorar gramática, puntuación etc... del mismo modo que anteriormente, resulta difícil determinar si esto va determinado por un usuario novato que desconoce las reglas, o algo realizado a conciencia con el objetivo de vandalizar el artículo.
- Por último la red 6.25 muestra usuarios que realizan tareas de Wikification y Refactoring, en algunos casos finalizando con Process. La rama superior de la red muestra usuarios que han realizado solo revisiones de refactoring+Wikification. El hecho de que solo hayan realizado esa intención tiene sentido pues refactoring es una intención no

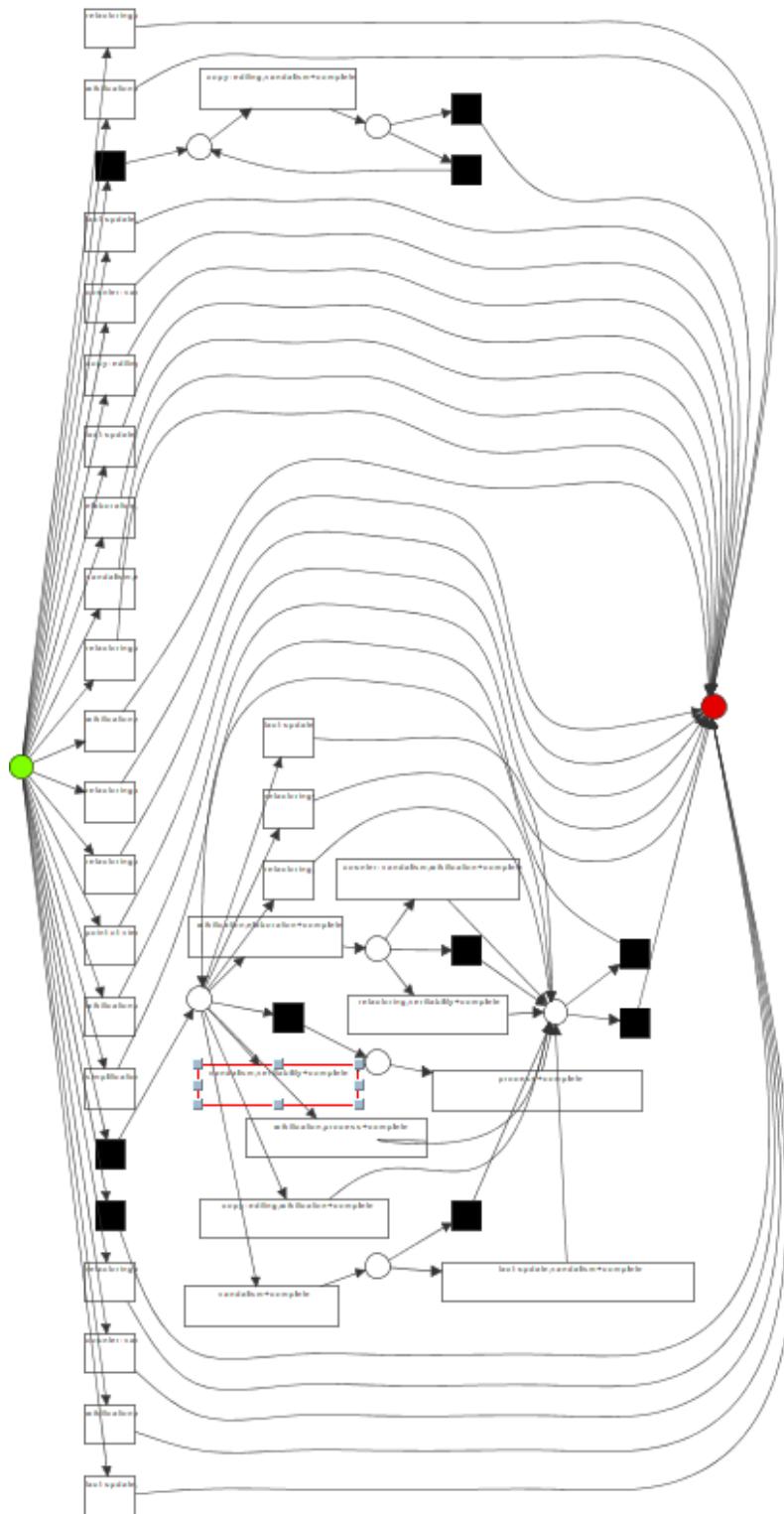


Figura 6.22: 4º Petri net del proceso seguido por los editores de actividad baja

muy apropiada para principiantes[21], aunque no hay garantía desde el alcance de este análisis para saber si los usuarios que han realizado esto son principiantes.

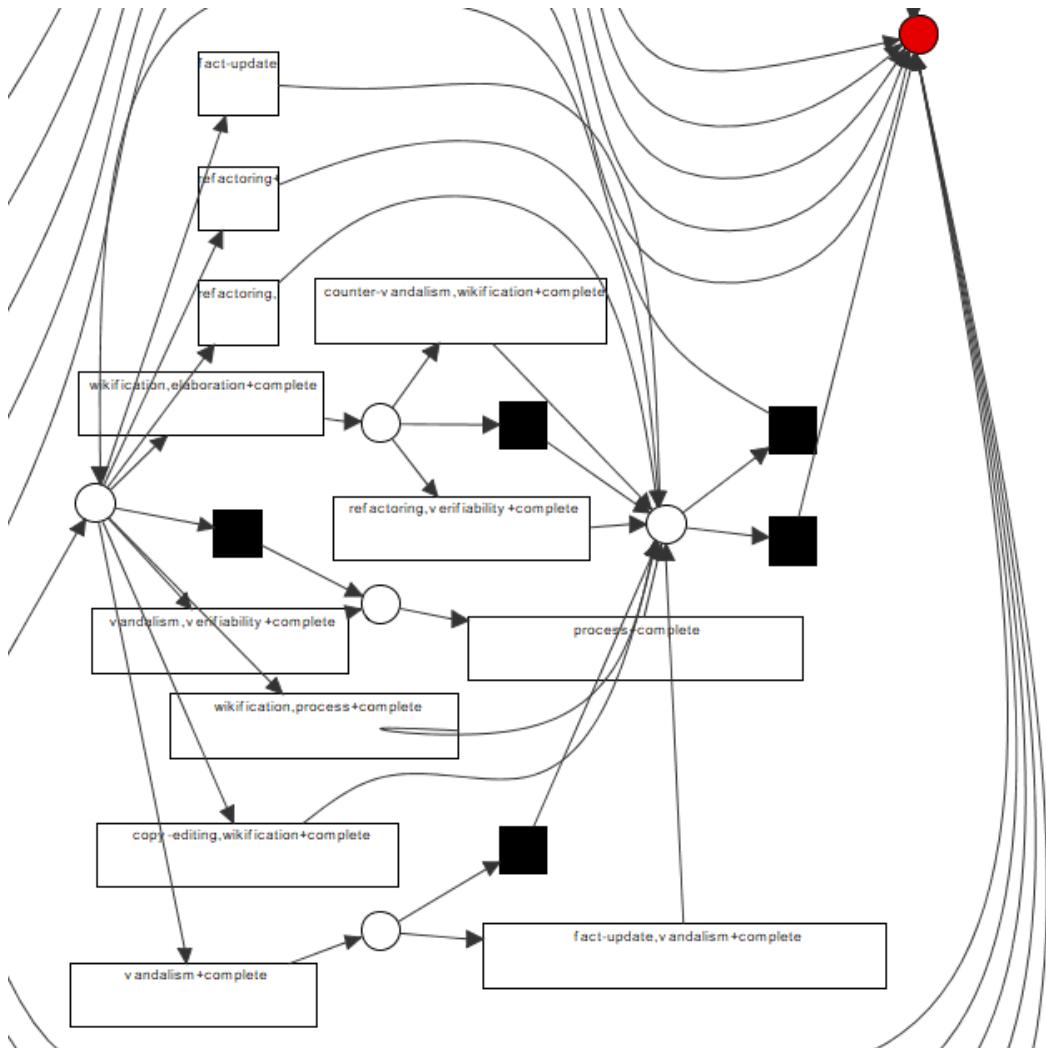


Figura 6.23: Sección inferior de la 4º Petri net del proceso seguido por los editores de actividad baja

6.5.2. Editores de actividad intermedia

Una vez que contamos con el **log de eventos de los editores con actividad intermedia**: que han realizado entre 5 y 50 revisiones y lo importamos en ProM aplicamos 'Generate log from org:perspective' de nuevo y obtenemos un log de eventos compuesto por 343 casos y 4429 eventos. Es decir, tenemos 343 editores diferentes que han realizado 4429 revisiones. Descomponemos en cluster haciendo uso de la herramienta 'Discover Clusters' y aplicamos 'Discover using Decomposition' en este caso seleccionando el Inductive Miner en su variante 'Perfect Fitness'. Así, obtenemos diferentes redes como vemos en las figuras 6.26,6.27,6.28,6.29

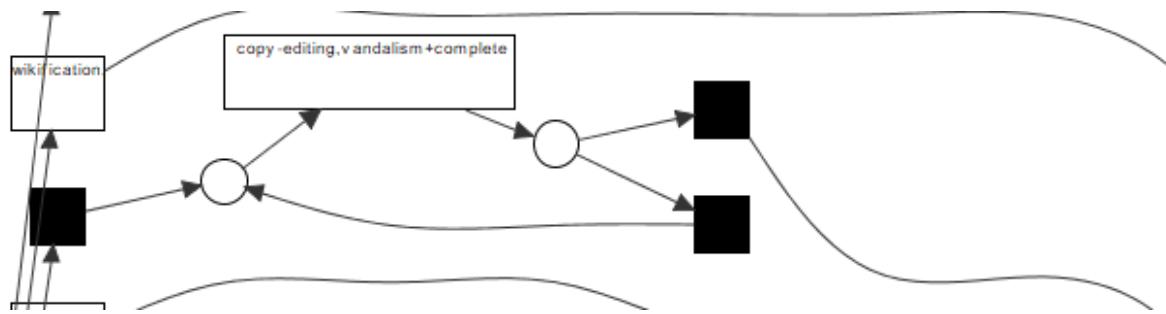


Figura 6.24: Sección superior de la 4º Petri net del proceso seguido por los editores de actividad baja

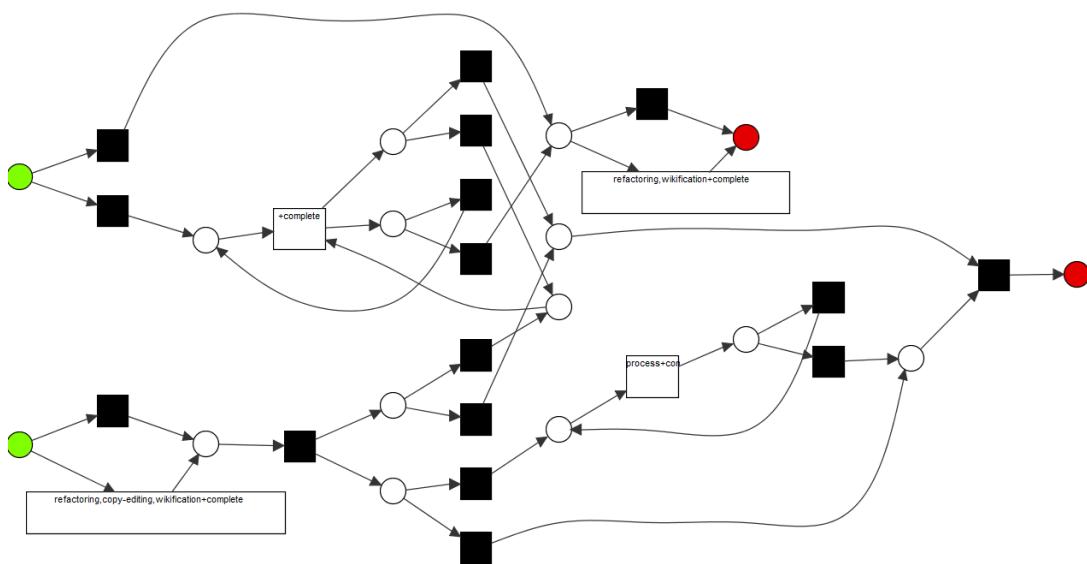


Figura 6.25: 5º Petri net del proceso seguido por los editores de actividad baja

En total, nos encontramos con un conjunto de redes bastante diverso, con flujos de trabajo complejos y otros razonablemente simples.

- La figura 6.26 muestra 2 flujos de trabajo diferentes. En el superior, vemos dos posibles opciones, o los usuarios editan realizando fact-update+Wikification o bien realizan revisiones con la intención de elaborar. Dado que nos encontramos en un flujo seguido por revisores que realizan entre 5 y 50 revisiones, esto da lugar a que esta red superior esté compuesta por usuarios que de manera repetitiva se han centrado en un tipo de intención específica tras sus revisiones. Más concretamente, entrarían dentro del rol de 'Quick and dirty editors' aquellos que realizan elaboration iterativamente y de 'Content Shapers' aquellos que realizan fact-update+Wikification, siguiendo la taxonomía de Daxenberger et al [6]. En cuanto a la red inferior, los editores o bien hacen labores de Wikification o Refactoring+Copy-editing+Wikification seguido de Wikification, lo cual indica claramente un comportamiento propio de los 'Content Shapers'.

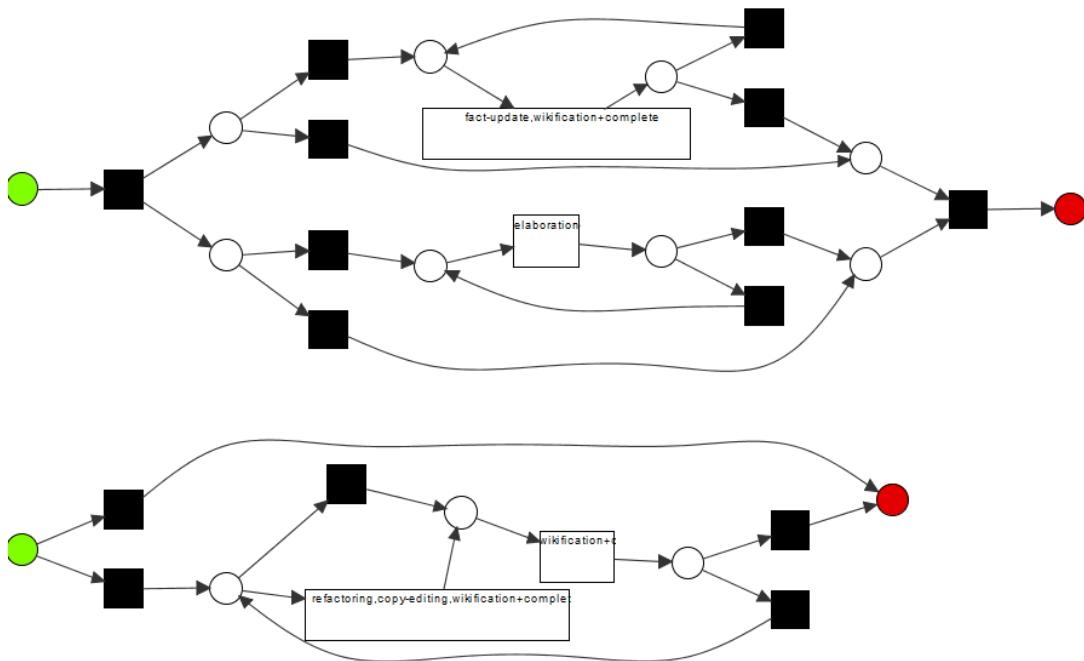


Figura 6.26: 1º Petri net del proceso seguido por los editores de actividad intermedia

- La red 6.27 tiene dos puntos de inicio y es compleja en tanto que el flujo seguido es caótico.
 - En el punto de inicio superior observamos que de nuevo vemos una rama cuya única intención es el vandalismo, que puede suceder de modo iterativo, por lo que se infiere que no es una intención propia solo momentos casuales si no que hay editores que se dedican, continuamente, a realizar vandalismo en un artículo o varios. Contrastando con esto, la otra opción dentro de esta rama de la red es copy-editing+wikification iterativamente, indicando de nuevo la existencia de los usuarios denominados 'Content Shapers'.
 - Por otro lado, en la rama que comienza en el punto de inicio inferior de la red se ve un flujo iterativo de revisiones bajo la intención Process, es decir, hay revisores específicamente centrados en realizar tareas relacionadas con Process. Además se ve como el resto de intenciones están relacionadas con la elaboración y la verificación con la ocasional wikification, de nuevo, 'Quick and dirty editors'. Sin embargo en este caso y en contra de lo comentado por Daxenberger et al no se observa vandalismo asociado a esto, probablemente relacionado con la veteranía de los editores[6].
- La figura 6.28 nos muestra una red como la obtenida con los editores de baja actividad (6.22) donde se observa que la cantidad de posibilidades existentes implica que representa una acumulación de todos aquellos workflows individuales que no ha logrado agrupar debido al funcionamiento del algoritmo Inductive Miner con perfect Fitness.
- La red 6.29 representa un conjunto amplio de diferentes posibilidades, sin embargo, mayoritariamente las intenciones que nos muestra son copy-editing, fact-update y wi-

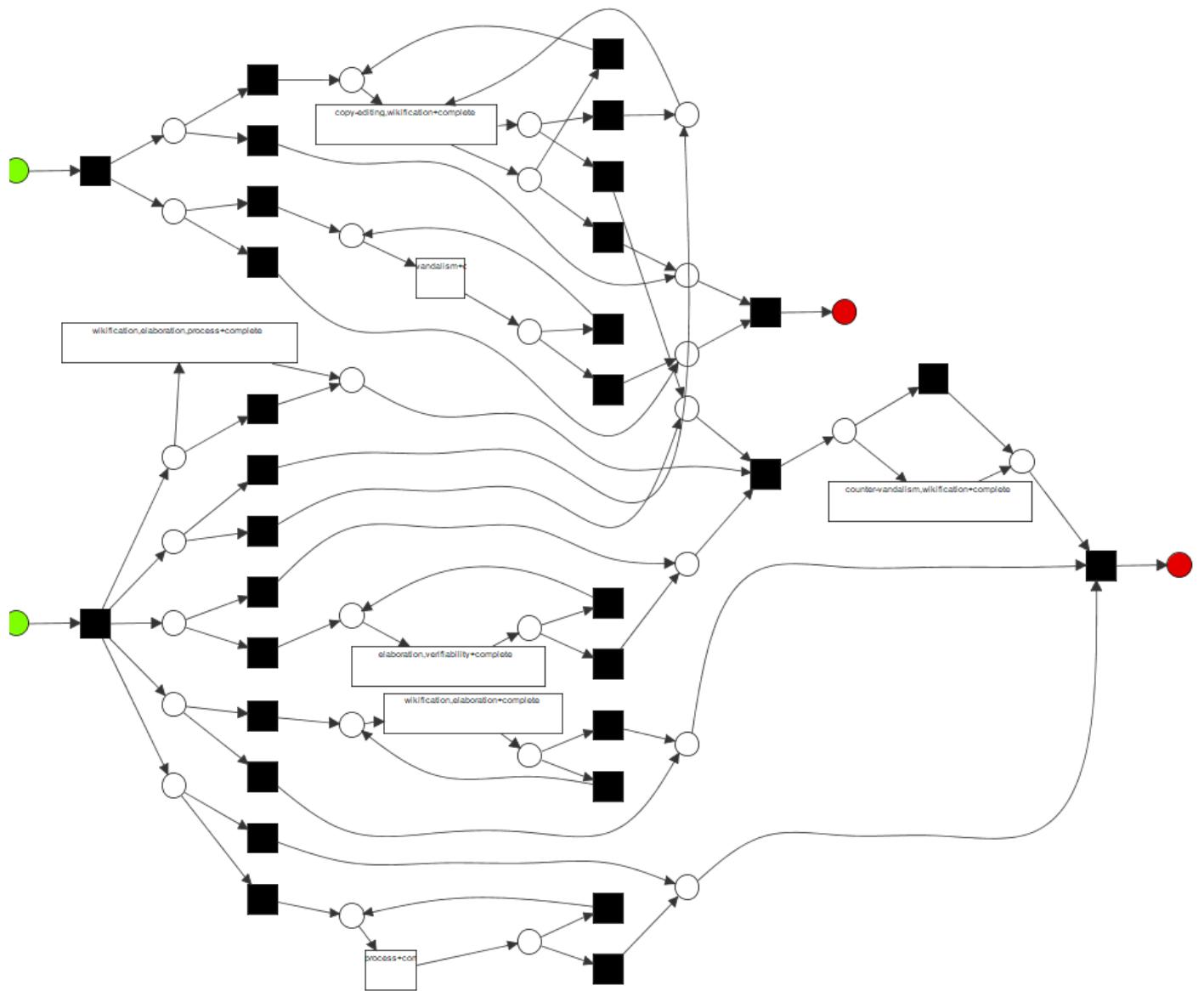


Figura 6.27: 2º Petri net del proceso seguido por los editores de actividad intermedia

kification, tanto independientemente como combinadas entre sí. Además, este proceso incluye la intención de Counter-Vandalism de modo que tras realizar contra-vandalismo se vuelve de nuevo a realizar las intenciones anteriores. Esto, encaja con la descripción del rol 'All round contributors' de Daxenberger et al donde este tipo de usuarios realizan tareas de adición de contenido y cambios en el texto actual además del formato y actualización de referencias. [6].

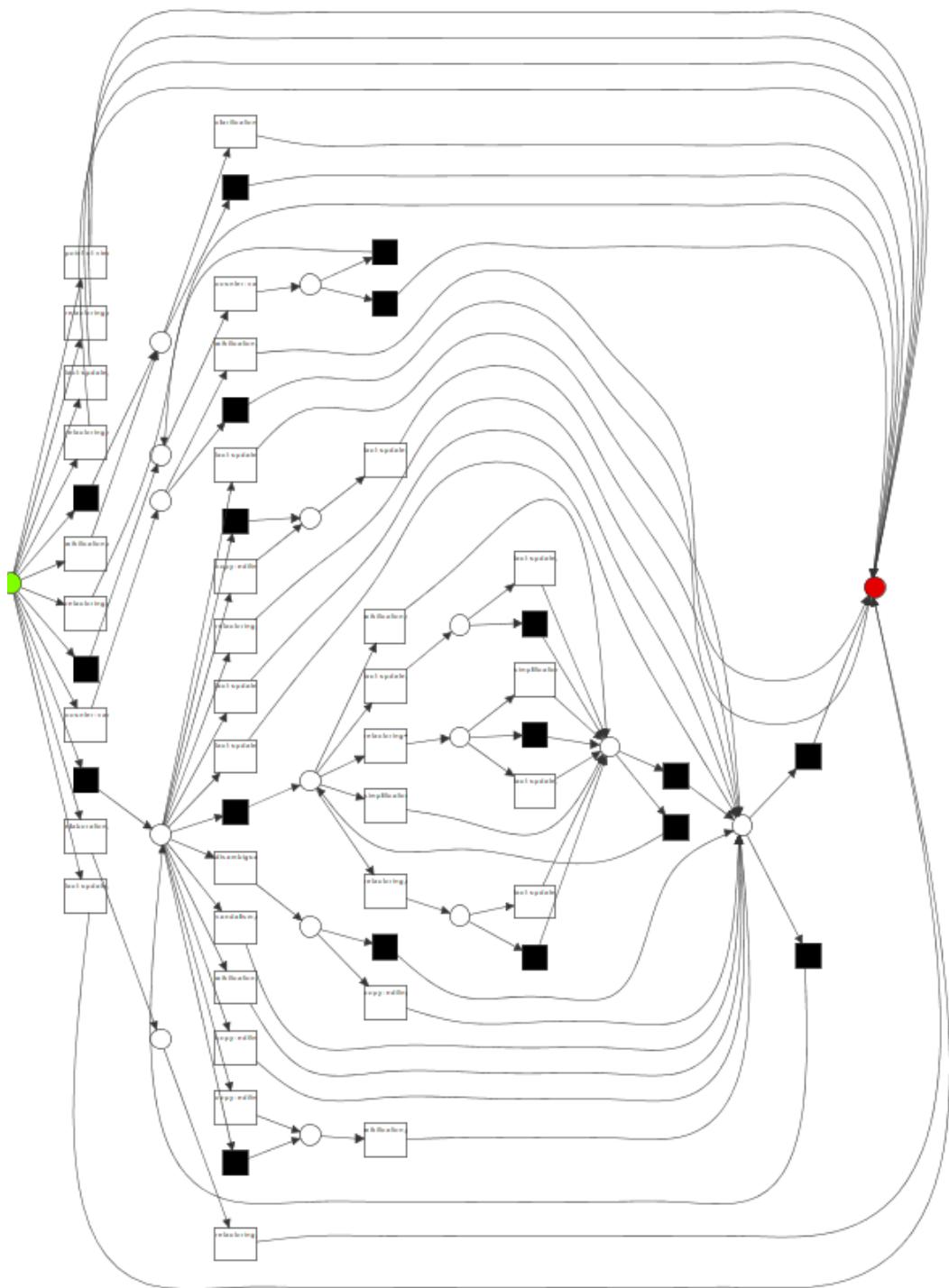


Figura 6.28: 3º Petri net del proceso seguido por los editores de actividad intermedia

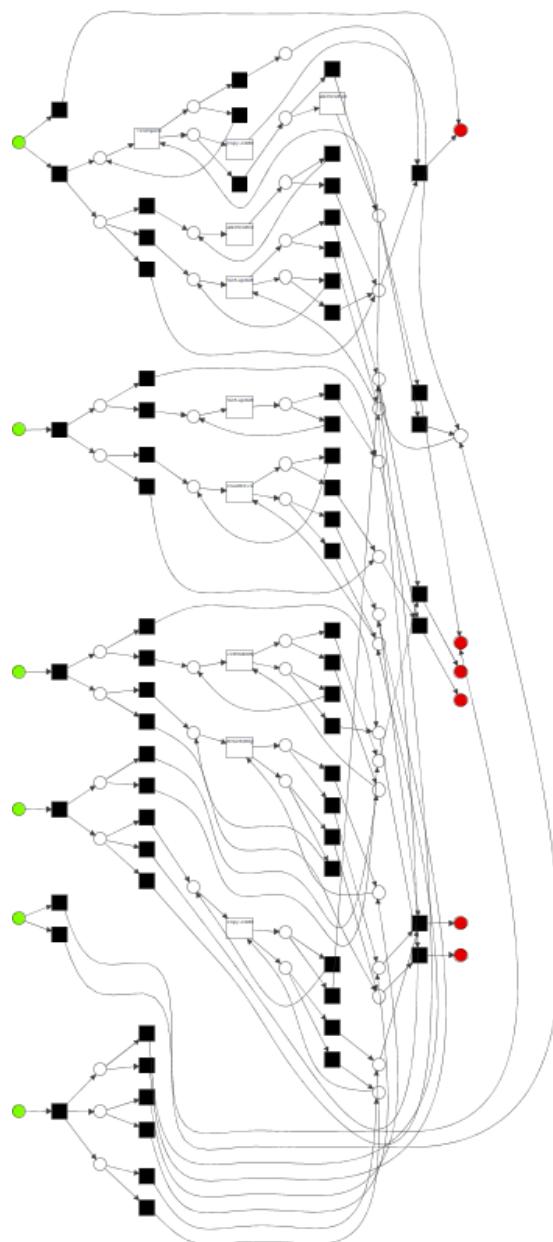


Figura 6.29: 4º Petri net del proceso seguido por los editores de actividad intermedia

6.5.3. Editores de actividad alta

Por último, nos encontramos ante el **log de eventos de los editores con actividad alta** compuesto por aquellos revisores con más de 50 revisiones a sus espaldas, es decir, aquellos con una alta actividad. Excluyendo, además, los agrupados como Anónimo. Esto es debido a que no se espera que los anónimos puedan aportar un flujo coherente de trabajo, además de que por el tipo de algoritmo de minería aplicado, sólo contaminaría las redes. Aplicamos 'Generate log from org:perspective' de nuevo y obtenemos un event log compuesto por 37 trazas y 3712 eventos. Es decir, tenemos 37 editores diferentes que han realizado 3712 revisiones.

Descomponemos en cluster haciendo uso de la herramienta 'Discover Clusters' y aplicamos 'Discover using Decomposition' en este caso seleccionando el Inductive Miner en su variante 'Perfect Fitness'. Así, obtenemos diferentes redes como vemos en las figuras 6.30,6.31,6.32

Contamos con 5 redes diferentes, de las cuales 2 son de considerable complejidad mientras que las otras 3 son sorprendentemente simples.

- Las redes de la figura 6.30 sorprenden por su simplicidad.
 1. La red superior se compone de simplemente de Refactoring+Copy-editing. Estos editores, entrarían en el rol de 'copy-editors' y 'content-shapers' al mismo tiempo de la taxonomía de Daxenberger et al [6]. Esto, claramente denota usuarios completamente dedicados a unas intenciones específicas, sin embargo, al representar dos roles diferentes, se propone añadir a la taxonomía el rol de 'Article fixers' ya que arreglan errores gramáticos y erratas además de organizar el texto existente.
 2. La red intermedia representa tres comportamientos diferentes entre sí y excluyentes:
 - a) El primero se basa usuarios que sólo editan con la intención de Copy-editing+Wikification los cuales podrían entrar dentro del rol propuesto previamente 'Article Fixers' ya que además de arreglar errores gramáticos y erratas, arreglan el formato del artículo con Wikification.
 - b) El segundo comportamiento está determinado por Vandalismo+Elaboration. Esto, a diferencia de la sección anterior con los usuarios de actividad intermedia, corrobora el comportamiento esperado por aquellos usuarios bajo el rol 'Quick and dirty editors' de la taxonomía. Dando lugar a que efectivamente este fenómeno de realizar vandalismo+Elaboration no es dependiente de la veteranía del usuario, si no que es algo propio de este estilo de edición.
 - c) El tercer comportamiento está formado únicamente por el contra-vandalismo, por lo que hay usuarios que actúan bajo el rol de 'Watchdog' salvaguardando el bienestar de artículos determinados a lo largo del tiempo. Por último, la red inferior se compone de diferentes posibilidades sin embargo todas las intenciones encontradas (Wikification, fact-update+verifiability, simplification y fact-update+wikification+refactoring) son diversas y afectan tanto a formato como contenido por lo que podrían ser propias de 'All round contributors'.
- La red 6.31 cuenta con 5 puntos de origen diferentes y es bastante compleja pues los caminos se entrelazan entre sí en numerosos y diferentes bucles. En primer lugar y debido a su estructura, hay muchas intenciones diferentes dentro de esta red. Esto, sumado a la gran cantidad de caminos y bucles existentes da lugar a que los usuarios bajo el rol 'All-round contributors' entren en este flujo. Sin embargo, se observan más roles diferentes. En el punto de inicio superior se observan bucles en las intenciones fact-update y Elaboration. La ruta de elaboration puede ir directamente hacia el final del flujo por lo que este proceso lo seguirían aquellos bajo el rol 'Quick and dirty editors', sin embargo y de nuevo, no se observa que haya vandalismo asociado a sus revisiones a diferencia de lo encontrado por Daxenberger [6]. Por otro lado las demás intenciones están relacionadas con formato y arreglo del artículo, por lo que de nuevo dentro de este flujo se pueden encontrar también 'Article Fixers'.

- La 6.32 no sirve para extraer un comportamiento determinado pues representa una acumulación de todos aquellos flujos de trabajo individuales que no ha logrado agrupar debido al funcionamiento del algoritmo Inductive Miner con perfect Fitness al igual que en las redes 6.28 y 6.22 de el log de eventos de los autores intermedios y los peores respectivamente.

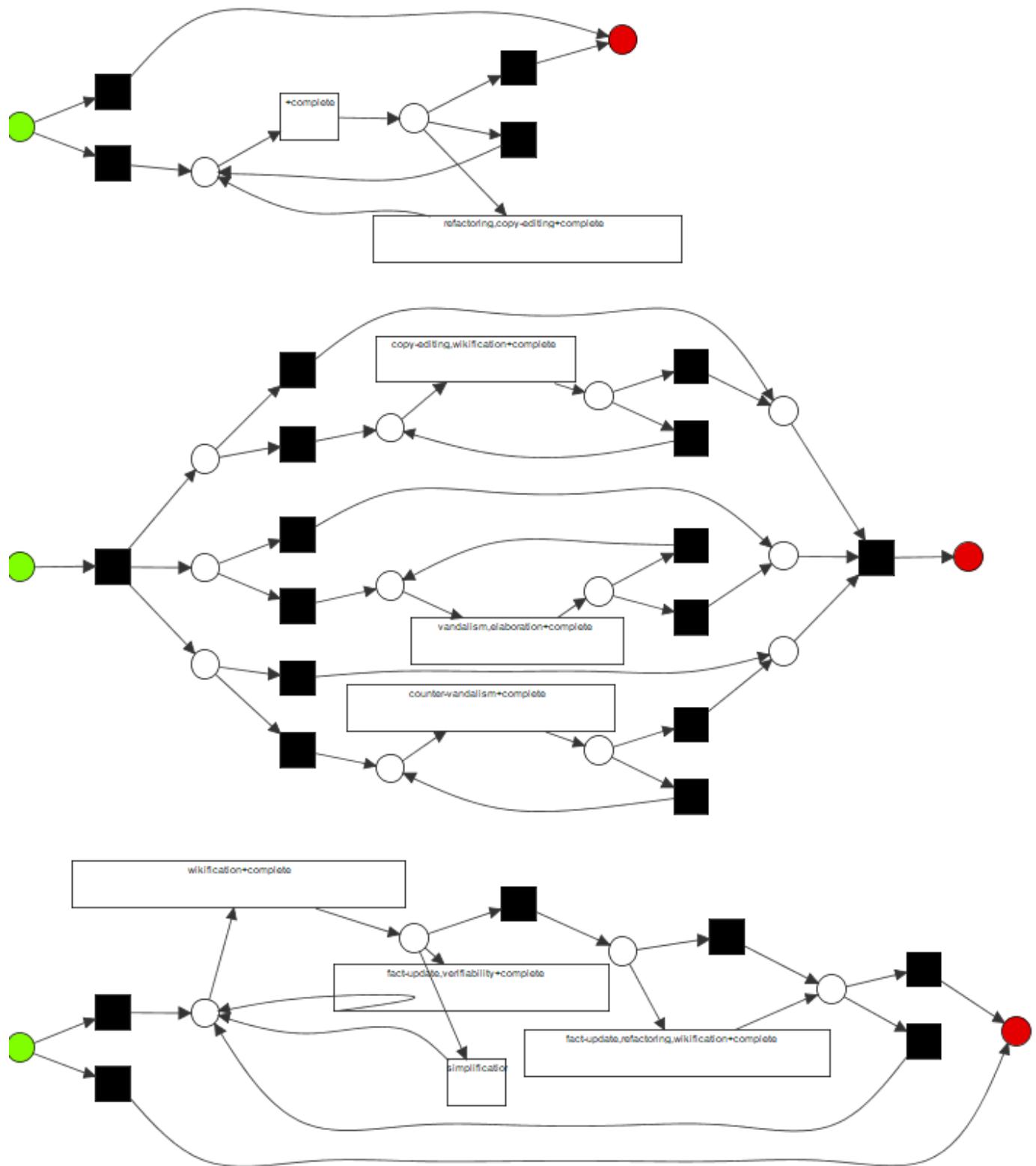


Figura 6.30: 1º Petri net del proceso seguido por los editores de actividad alta

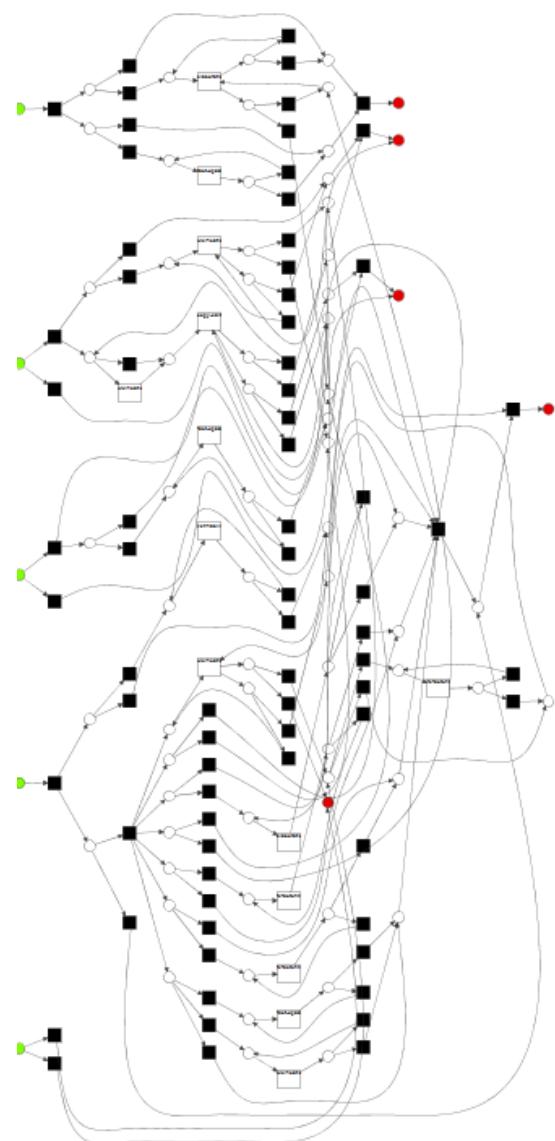


Figura 6.31: 2º Petri net del proceso seguido por los editores de actividad alta

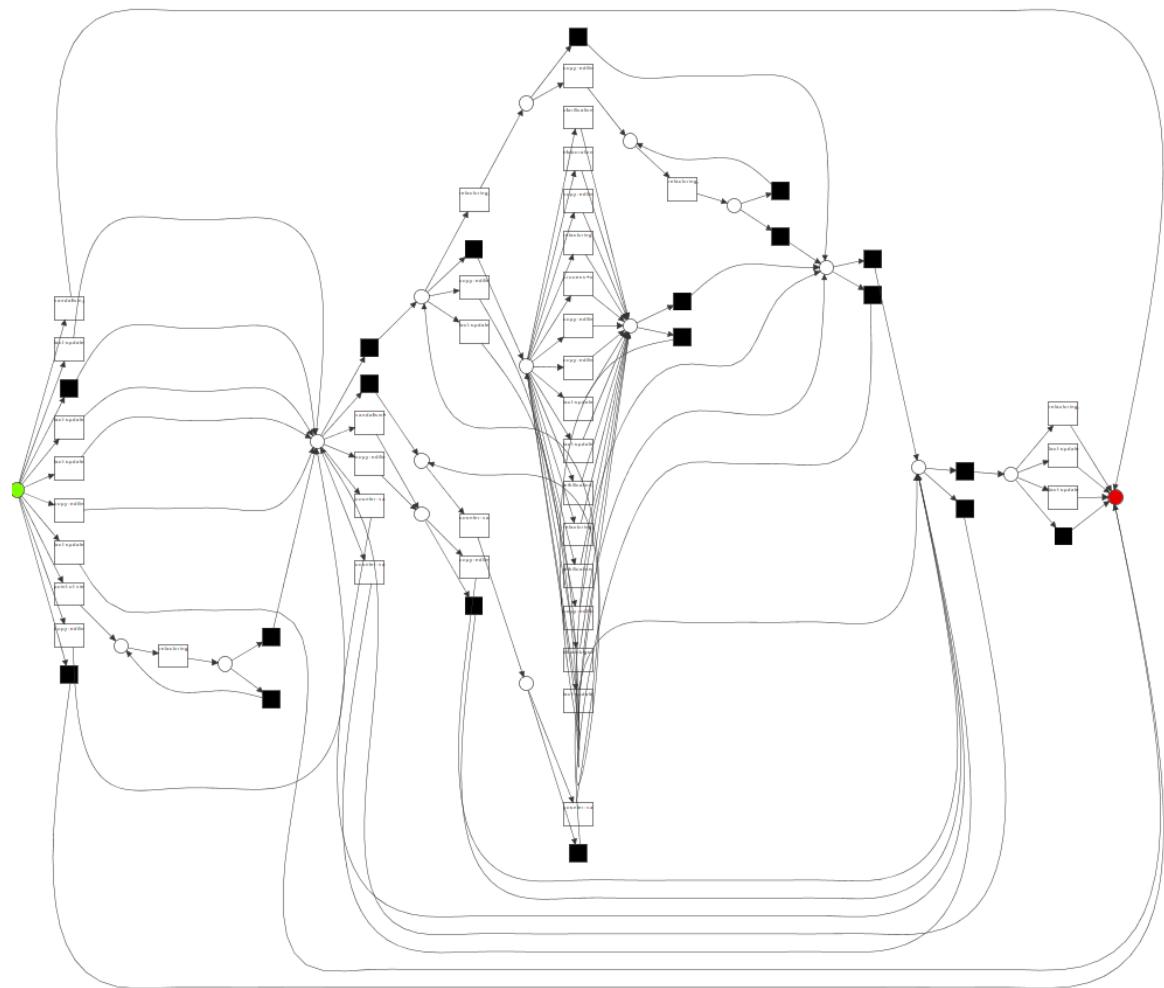


Figura 6.32: 3^o Petri net del proceso seguido por los editores de actividad alta

Capítulo 7

Minería Social

Retomando lo detallado en el capítulo 4 la minería social hace uso de técnicas de socio-metría y análisis de redes sociales[4] para observar y conocer las posibles relaciones existentes entre los diferentes elementos existentes en un log de eventos. De esta manera, se tratará de realizar un estudio de las relaciones existentes entre los editores de un artículo de Wikipedia.

La herramienta ProM 6.8 brinda una librería basada en métricas establecidas por Wil M.P. van der Aalst et al en [3]. Estas métricas son:

1. Handover of Work: Se define como Handover Of Work como el traspaso de trabajo de un individuo i a un individuo j si hay dos actividades subsecuentes entre ellos dentro de un log de actividad.[3] Es decir, si después de editar i edita j se establece una relación entre ellos. Estos individuos i y j son representados mediante nodos y su conexión mediante aristas, variando el peso en función de lo fuerte que sea la relación entre ellos.
2. Subcontracting: Subcontracting cuenta el número de veces que un individuo j ejecuta una actividad entre dos actividades ejecutadas por el individuo i[3]. Es decir, si i edita, j edita e i vuelve a editar, se establece una relación de i a j. Así, siendo i y j representados como nodos su relación se establece mediante una arista común cuyo peso dependerá de las veces que suceda la relación.
3. Working-Together: Working Together simplemente tiene en cuenta la frecuencia con la cual dos resources realizan actividades en el mismo caso (traza)[3] dentro del log de eventos. De este modo, dos resources que trabajen en el mismo caso estarán relacionados y el peso de su arista dependerá de la frecuencia con la que suceda siendo representados por nodos.
4. Similar Task: Similar Task se centra en el tipo de actividades que realizan los diferentes resources. Así, se establecerá una relación entre resources que están realizando un tipo similar de actividades, el peso de su arista al representar los resources mediante nodos dependerá de la similitud de las tareas realizadas. La similitud entre las tareas realizadas por los diferentes resources se puede calcular mediante 4 métricas diferentes:
 - Distancia Euclidea: mide la distancia en el espacio entre 2 puntos.
 - Pearson Correlation Coefficient: es una métrica de correlación lineal utilizada frecuentemente para encontrar relaciones entre casos.

- Jaccard Similarity Coefficient: compara los diferentes grupos posibles y mide que miembros son compartidos entre los grupos y que miembros no.
- Hamming distance por otro lado, comprueba el número mínimo de sustituciones requerido para transformar un string en otro.

Estas métricas representan las relaciones existentes en forma de grafo donde cada resource es un nodo y cada arista una relación, dependiendo el peso de la intensidad de esta relación y hacen uso de los mismos ficheros de datos que la minería de procesos: log de eventos.

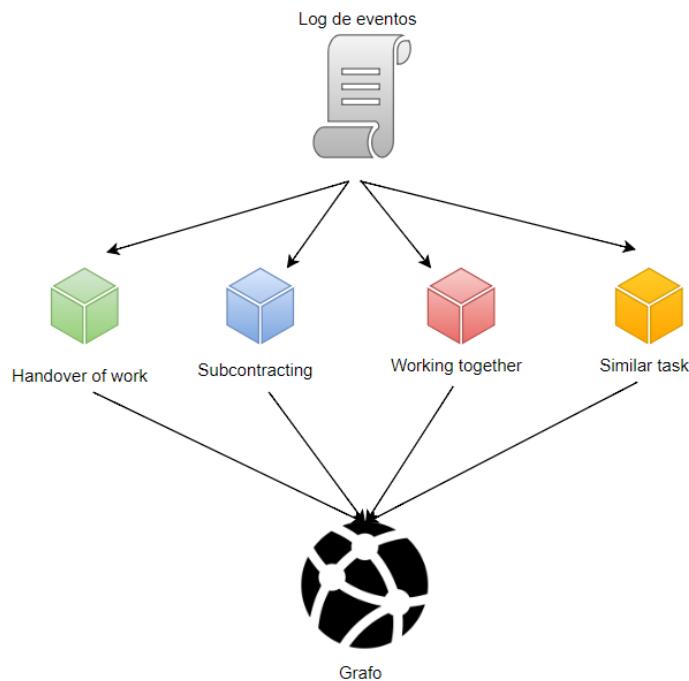


Figura 7.1: Estructura de la minería social en ProM

Durante esta sección se hará uso de dos *artículos destacados* de Wikipedia como log de eventos. Estos artículos serán analizados y comparados individualmente para determinar diferencias y similaridades. En concreto el proceso a seguir será el siguiente:

1. Obtención de datos: Se determinará qué artículos de Wikipedia serán analizados y el modo en el cual se han obtenido.
2. Handover of Work: Aplica handover of work a cada artículo para observar el traspaso de trabajo entre los editores.
3. Subcontracting: Del mismo modo que aplicamos handover or work se aplicará subcontracting a ambos artículos por separado para determinar la 'subcontratación' entre editores.

Se ha determinado que la aplicación de Working Together no aporta información al estudiar artículos individuales por lo que esta no se aplicará. Esto es debido a que al medir

la frecuencia con la cual los editores trabajan juntos en el mismo artículo, para que pueda aportar resultados interesantes se necesita un conjunto de artículos de mayor tamaño, como por ejemplo, una Wiki entera.

Adicionalmente, tampoco se hará uso de Similar-Task. En los criterios de similitud de similar task la frecuencia tiene un peso importante, haciendo que aunque dos editores editen de la misma manera, una frecuencia de edición ligeramente menor podría hacer que se computen como usuarios de diferente comportamiento y dada la gran variedad de cantidad de revisiones en nuestros usuarios esto puede afectar a la interpretación de los resultados.

7.1. Obtención de datos

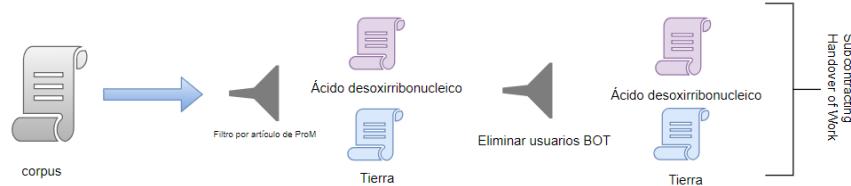


Figura 7.2: Proceso a seguir para la obtención de los datos usados en la minería social

En este capítulo la obtención de los datos es considerablemente más simple que en la sección anterior. Esto es debido a que el punto de partida es el propio **corpus** utilizado como log de eventos en la sección anterior. El proceso a seguir será el observado en el esquema 7.2

Los datos a utilizar serán:

- Handover of Work y Subcontracting harán uso de dos artículos de entre los existentes en el **corpus** por separado. Estos serán separados del **corpus** mediante un filtro de ProM que permite separar por traza los log de eventos y los usuarios BOT serán eliminados mediante un script de filtrado.¹

Los archivos seleccionados han sido, de entre los 8 existentes en el **corpus**: el artículo Tierra y Ácido desoxirribonucleico pues tienen un período de vida similar, existiendo desde los inicios de Wikipedia hasta el día de hoy y por tanto teniendo una rica historia de ediciones. Sin embargo, existe la necesidad de filtrar a sus editores BOT.

Los usuarios BOT son programas informáticos automatizados que realizan tareas repetitivas de revisión por lo que de cara a este análisis esto puede embotar las existentes colaboraciones entre los editores reales. Por este motivo, han sido eliminados de cara a estudiar el handover of Work y el Subcontracting de los artículos seleccionados.

¹corpus_filter.py (11)

7.2. Handover of Work

En esta sección se aplicará el algoritmo de minería social Handover of Work a los 2 artículos seleccionados del **corpus**: Ácido desoxirribonucleico y Tierra con los usuarios BOT filtrados. Así, sus representaciones gráficas son representadas siguiendo los mismos parámetros en ambos casos. Los colores representan los diferentes grupos existentes y están organizado de modo que todos los grupos se encuentren juntos. Debido a que la paleta de colores es reducida se pueden observar grupos del mismo color separados, por lo que se debe tener en cuenta color y región en la que se encuentre a la hora de determinar los distintos grupos existentes. Esta agrupación se basa en el peso de las aristas, cuantas más veces el editor j haya editado tras i y viceversa, mayor será el peso de la arista que los une.

Por otro lado el tamaño del nodo dependerá de su grado. En este caso, aquellos editores que hayan realizado más revisiones antes o después que otros serán representados por un nodo de mayor tamaño.

7.2.1. Artículo Tierra

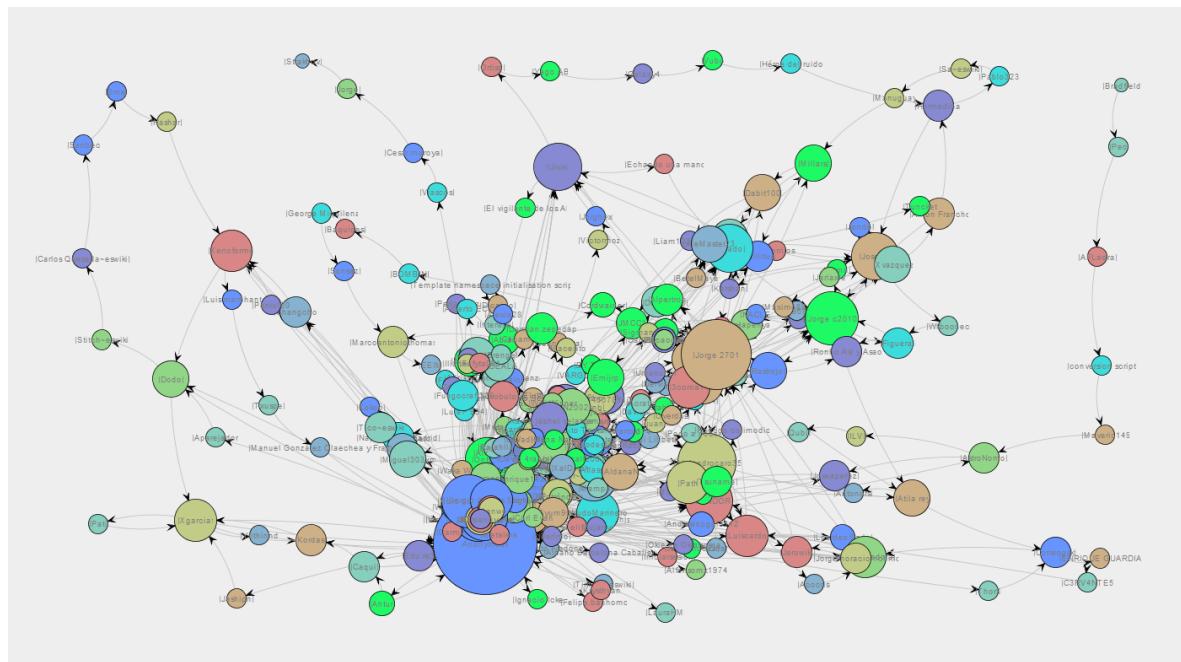


Figura 7.3: Grafo handover of work del artículo Tierra

El artículo tierra consta de 2654 revisiones y 460 editores. Por lo tanto contamos con una gran cantidad de editores propia de la antigüedad del artículo. Esto va a dar lugar a un grafo de handover complejo, con un gran número de nodos y aristas

En la figura 7.3 se encuentra el grafo de handover of work generado para el artículo Tierra tras filtrar a los usuarios BOT. Una gran cantidad del conjunto de editores se encuentra en la zona central del grafo mientras que hay una porción de los usuarios que se encuentran en la periferia estando conectados únicamente con 1 o 2 revisores y con un tamaño reducido. Esto

pueden ser ediciones realizadas por usuarios que han trabajado en momentos puntuales, sin embargo los motivos de esto son desconocidos dentro del alcance de este análisis.

Se ve como sin embargo no todos los nodos de la periferia han realizado pocas revisiones, algunos tienen un mayor tamaño implicando un mayor grado y su conexión con los nodos del conjunto central es casi directa, sin embargo sus aristas de mayor peso están conectadas con los nodos periféricos y por ello se encuentran en esa localización. Esto podría implicar alta actividad por parte de estos usuarios pero en una franja de tiempo reducida dando lugar a colaboraciones con una piscina de usuarios menor que un editor que haya podido estar realizando pocas ediciones en el tiempo pero a lo largo de toda la vida del artículo, que en este caso, son más de 15 años.

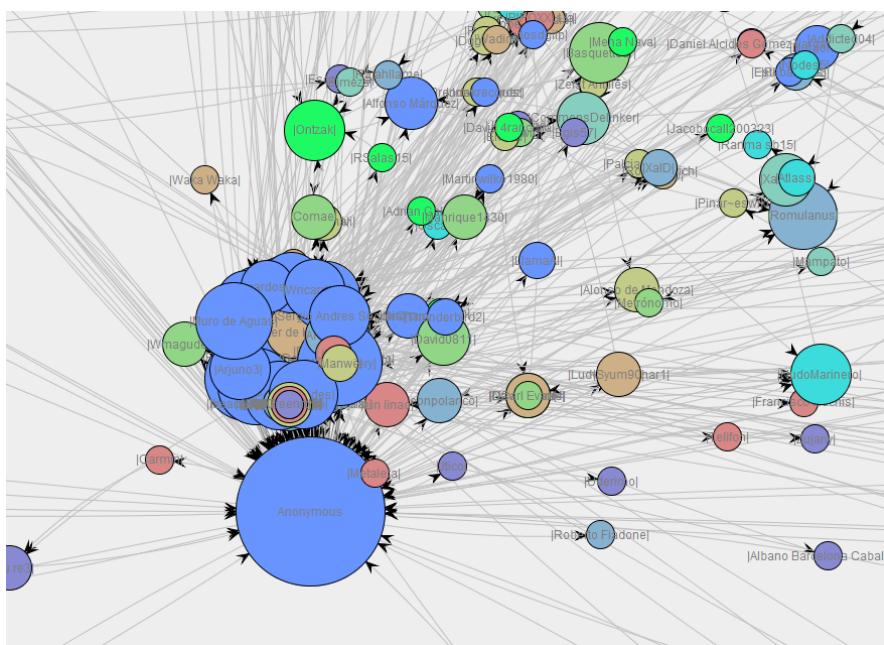


Figura 7.4: Zoom grupo central del grafo handover of work del artículo Tierra

Centrando el foco de atención en el grupo central, tenemos la ampliación del grafo anterior en la figura 7.4. Claramente se observa un grupo muy cercano de usuarios con un grado alto a juzgar por su tamaño en conjunto con otros de menor tamaño. Debido a su centralidad dentro del grafo estos nodos están relacionados con los usuarios de más peso dentro del artículo tal y como se puede ver por el nodo que representan los usuarios anónimos y su enorme cantidad de aristas. Del mismo modo que aunque en menor medida se observa una gran cantidad de aristas conectando con el conjunto de usuarios agrupados que se observa.

Tal y como se comenta antes del comienzo de la sección, el color de los nodos y su localización determina su pertenencia a un grupo específico. Aquí vemos como todo este conjunto central de usuarios influyentes se encuentra coloreado con el mismo color, indicando que entre sí forman un único conjunto. De esto se extrae que aquellos usuarios que hacen numerosas ediciones, en la mayoría de los casos, son activos durante un largo periodo de tiempo, cosechando un alto número de conexiones bajo la métrica handover of work.

7.2.2. Artículo Ácido desoxirribonucleico

En este caso contamos con un artículo compuesto por 3308 ediciones y 396 editores. Del mismo modo que anteriormente, esto da lugar dará lugar a un grafo de handover complejo, con un gran número de nodos y aristas.

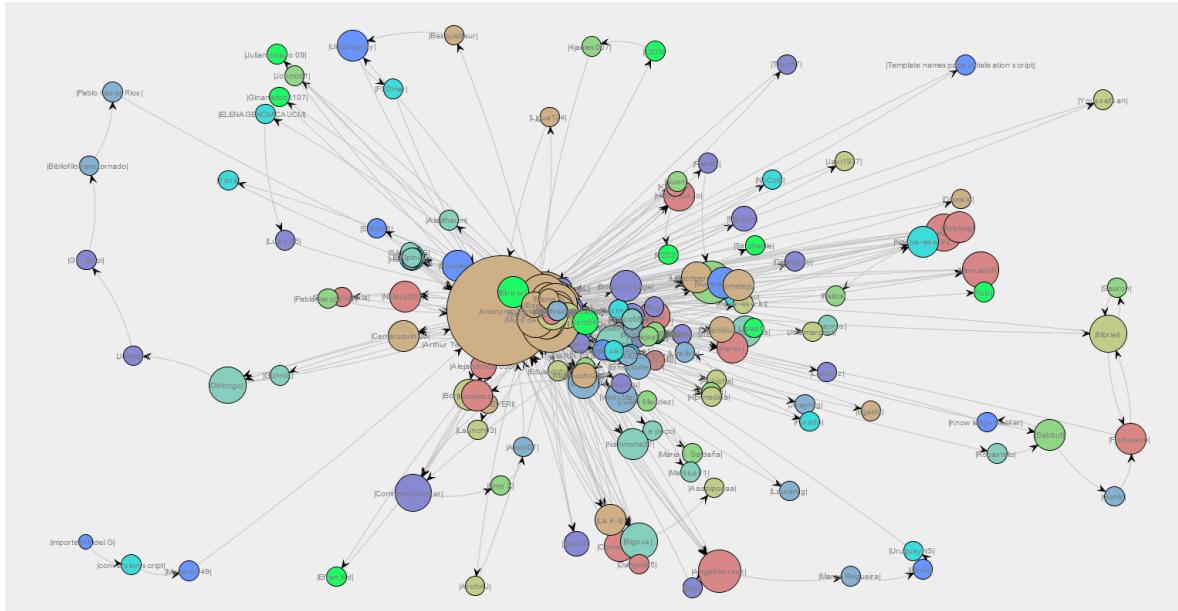


Figura 7.5: Grafo handover of work del artículo Ácido desoxirribonucleico

El grafo resultante, representado en la figura 7.5, muestra una situación similar al observado en el artículo Tierra: la mayoría de editores se encuentran agrupados en el centro del grafo mientras que una pequeña porción de los mismos se encuentra en la periferia.

Estos nodos que componen la periferia siguen la misma estructura que en el caso del artículo Tierra. La mayoría son nodos pequeños con conexiones a uno o dos nodos con una porción de nodos de mayor tamaño y por ende grado probablemente debido a editar durante un espacio pequeño en el tiempo aunque con un grado de actividad mayor que los demás.

Poniendo el foco en el grupo central, visible en ??, vemos de nuevo un fenómeno similar al anterior artículo, un conjunto de editores pertenecientes al mismo grupo como denota su color y localización de un tamaño grande indicando una gran cantidad de conexiones. Es decir, los usuarios más influyentes a lo largo de la evolución del artículo se encuentran aquí. Además se encuentran relacionados entre ellos, indicando colaboración entre los mismos. Básicamente la conclusión que puede extraerse es la misma que antes: estos editores que hacen numerosas ediciones, en la mayoría de los casos, son activos durante un largo periodo de tiempo, cosechando un alto número de conexiones bajo la métrica handover of work.

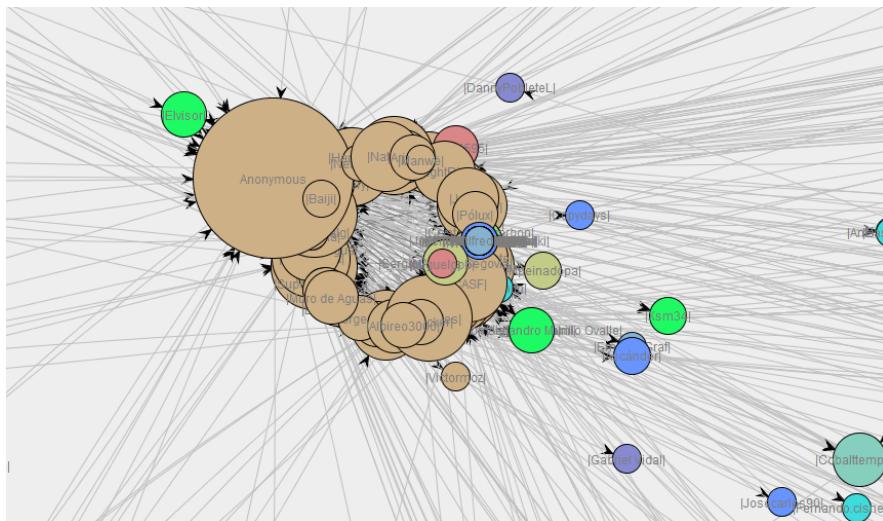


Figura 7.6: Zoom grupo central del grafo handover of work del artículo Ácido desoxirribonucleico

7.3. Subcontracting

En esta sección se aplicará el algoritmo de minería social Subcontracting a los 2 artículos seleccionados del **corpus**: Ácido desoxirribonucleico y Tierra con los usuarios BOT filtrados. Así, sus representaciones gráficas son representadas siguiendo los mismos parámetros que en el caso del Handover of Work: los colores representan los diferentes grupos existentes y están organizados de modo que todos los grupos se encuentren juntos. Esta agrupación de nuevo se basa en el peso de las aristas, cuantas más veces el editor j haya editado entre dos ediciones del editor i y viceversa, mayor será el peso de la arista que los une.

Por otro lado el tamaño del nodo también dependerá de su grado y su situación geográfica dentro de la red afecta a su pertenencia a determinado grupo a pesar de su color.

7.3.1. Artículo Tierra

Observando el grafo resultante en la figura 7.7 se ve que la mayor parte de los nodos se encuentran aislados. Esto, en el subcontracting, se traduce a que son usuarios que han editado durante una sola sesión. En subcontracting los vértices se forman entre los editores i y j si i edita entre dos ediciones de j. Dado que en este caso no hay conexiones, solo realizaron revisiones una vez ininterrumpidamente. El número de revisiones realizado no está determinado, puede ser una o más, pero carecemos de esa información bajo esta perspectiva. Sin embargo en resumen se puede extraer lo siguiente, la mayoría de editores editan durante una sola sesión en este artículo. Sin embargo, también encontramos un grupo de usuarios en el centro del grafo.

Este grupo central, visualizable en 7.8, vemos que en realidad está compuesto de 2 grupos diferentes.

- El primer grupo, compuesto por nodos grandes en color rojo. Vemos como se trata de un grupo de editores que tienen un grado alto por su tamaño y que además suelen hacer

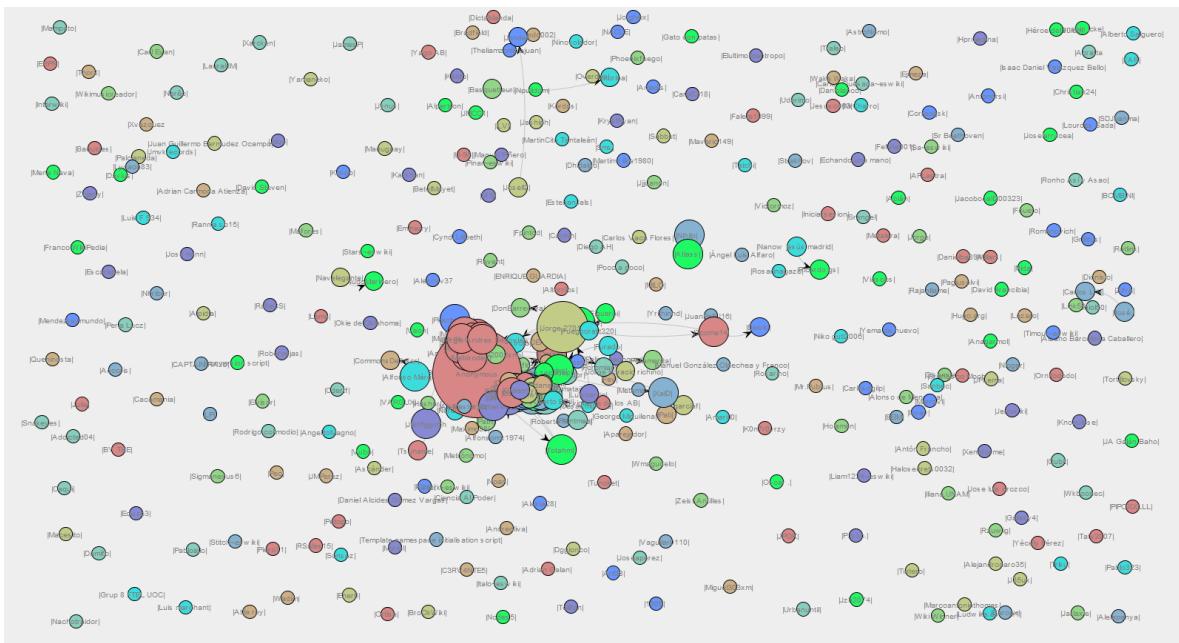


Figura 7.7: Grafo subcontracting del artículo Tierra

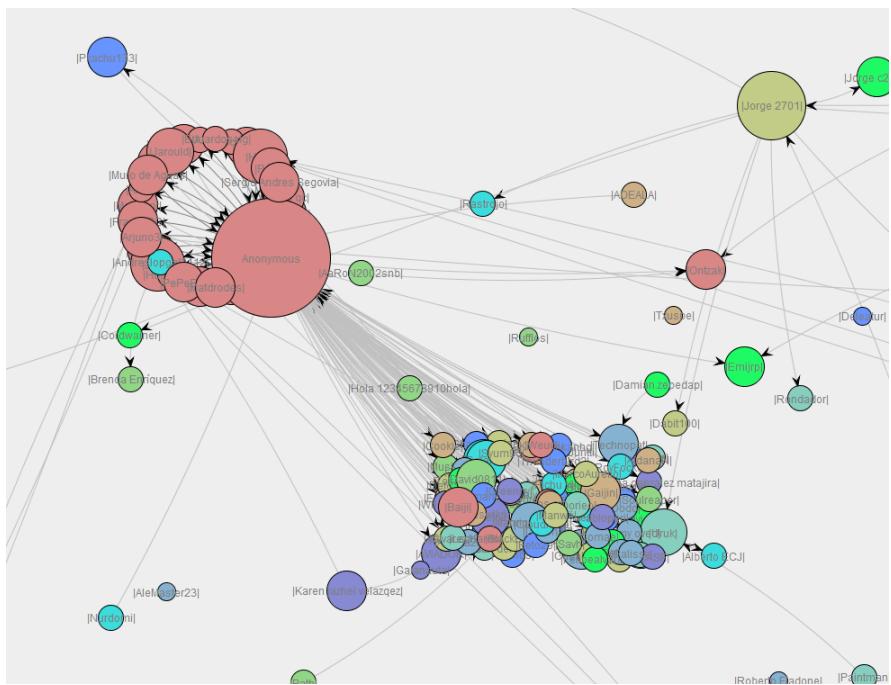


Figura 7.8: Zoom grupo central del grafo subcontracting del artículo Tierra

subcontracting entre ellos. Sin embargo no solo se reducen a su mismo grupo si no que hay numerosas aristas hacia otros nodos diferentes localizados fuera del grupo central. Esto de nuevo podría implicar un largo historial de ediciones en el tiempo por parte de estos usuarios.

- En segundo lugar tenemos al conjunto de nodos localizado a la derecha. Estos nodos se encuentran agrupados entre sí por la cercanía y el peso de sus aristas sin embargo se observa que no necesariamente pertenece al mismo grupo si no que hay varios grupos entremezclados. Sus conexiones se observa van principalmente al nodo que representa a los usuarios anónimos por lo que este conjunto más que denotar un comportamiento específico muestra los editores que hicieron revisiones entre las revisiones realizadas por usuarios anónimos.

7.3.2. Artículo Ácido desoxirribonucleico

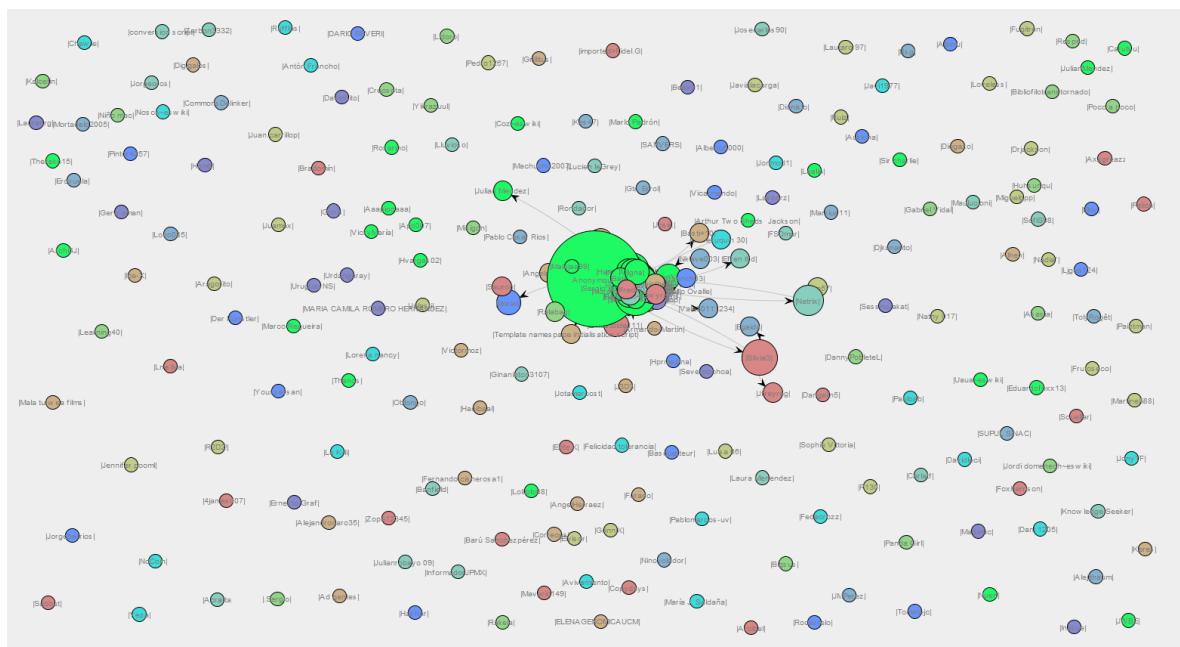


Figura 7.9: Grafo subcontracting del artículo Ácido desoxirribonucleico

El grafo de subcontracting del artículo Ácido Desoxirribonucleico, 7.9, muestra de nuevo, un comportamiento muy similar al subcontracting obtenido en el artículo Tierra. Se observa una gran cantidad de nodos aislados y un grupo central. Estos nodos se encuentran aislados debido al mismo motivo que aquellos en la subsección previa: Solo han realizado edits durante una sesión, dando lugar a una secuencia de 1 o más ediciones ininterrumpidas por otro editor.

En el conjunto central, 7.10, otra vez encontramos la misma estructura que en el artículo Tierra. Un grupo central de editores pertenecientes al mismo grupo, en verde (en el artículo Tierra en rojo) y otro compuesto de usuarios de menor grado de diferentes grupos agrupados en el centro del grafo debido a sus interacciones con los usuarios anónimos.

La diferencia en este caso es el tamaño de los nodos del grupo central verde. Estos tienen menor tamaño en comparación con los del anterior artículo. Sin embargo, este artículo cuenta con 3308 ediciones y 396 mientras que el artículo tierra con 2654 revisiones y 460 editores. Al medir subcontracting, un mayor número de editores existentes da lugar a un mayor número posible de conexiones y con esto un mayor tamaño en sus nodos más influyentes. Debido a

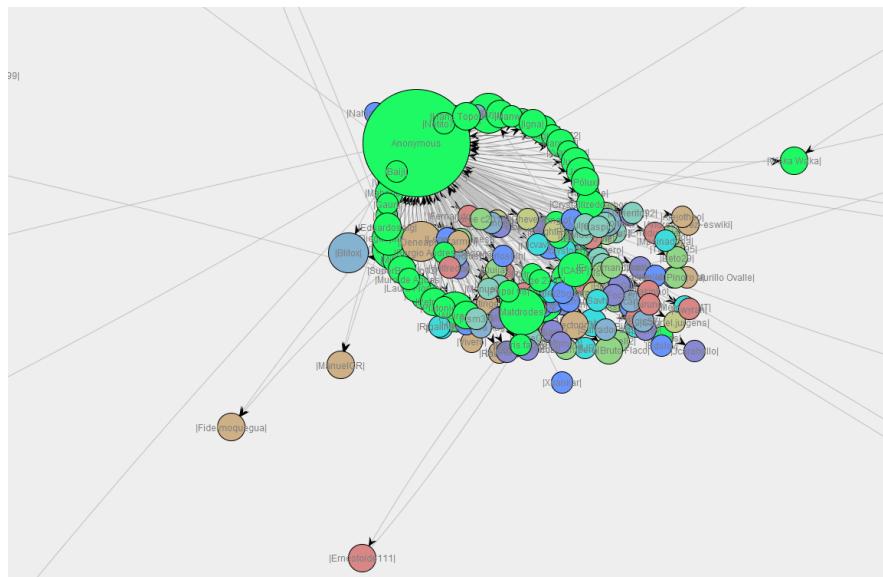


Figura 7.10: Zoom grupo central del grafo Subcontracting del artículo Ácido desoxirribonucleico

esto, el artículo ácido desoxirribonucleico cuenta con un menor número de editores y un mayor número de ediciones que el artículo Tierra, dando lugar así a menos posibles conexiones entre usuarios reduciendo el tamaño de los nodos.

Capítulo 8

Conclusiones

8.1. Conclusiones Minería de Procesos a nivel artículo

Se ha aplicado minería de procesos a un **corpus** compuesto de 8 *artículos destacados* de la Wikipedia Española desde el punto de vista del artículo. Es decir, conformando un log de eventos donde cada caso es representado por un artículo y cada 'actividad' realizada llamada evento es representada por la intencionalidad semántica tras cada revisión. El resultado ha sido una red de gran complejidad e imposible de analizar para el ojo humano.

Esta red obtenida representa la complejidad del proceso de edición que siguen los artículos. Al tratarse Wikipedia de una comunidad de conocimiento colaborativo abierta, cualquier persona puede convertirse en un editor. Esto intuitivamente se traduce en una gran cantidad de usuarios con variados rangos de conocimiento y habilidad y por ende muchos estilos de edición diferente. Sin embargo, para obtener más información acerca de esta red se descompuso en sub-redes de mayor simplicidad.

Estas sub-redes muestran un panorama similar al anterior, su complejidad es alta aunque con diferencias entre ellas. Mientras que no representan un proceso específico que se siga a raja tabla y de ahí su complejidad, muestran comportamientos diferentes.

Una de las redes obtenida muestra intencionalidades propias de usuarios todo-terreno con intenciones combinadas de cierta complejidad desde el comienzo mientras que la otra red muestra intenciones más específicas propias de usuarios con mayor grado de especialización.

En resumen, se puede concluir que (i) no hay un proceso unificado que se siga en la creación y evolución de los artículos en Wikipedia y (ii) la especialización o el generalismo de los editores en etapas tempranas de un artículo tiene influencia en la evolución de los mismos como vemos por las diferentes redes obtenidas tras la descomposición dando lugar a diferentes flujos de trabajo.

8.2. Conclusiones Minería de Procesos a nivel editor

En este caso la minería de procesos fue aplicada dándole un giro al log de eventos de la sección anterior orientandolo al editor. De esta manera, se agrupan los editores en 3 categorías

diferentes en función de su actividad dentro del conjunto 8 artículos utilizado (Baja/Intermedia/Alta actividad) medido por el número de ediciones realizado.

Analizando los procesos seguidos por los usuarios en sus sesiones de edición, se observan grandes similaridades con aquellos comportamientos que identifican Daxenberger et al[6] en 'Turbulent Stability of Emergent Roles'.

Esta taxonomía estaba compuesta por 'Watchdogs', 'All round contributors', 'Quick and dirty editors', 'Copy editors', 'Content shapers', 'Layout shapers' y 'Vandals'.

De esta manera, a lo largo de las 3 agrupaciones realizadas, se observan cambios importantes en los roles observados en función del número de ediciones

Entre aquellos usuarios con baja actividad, se observa un poco de todo, aunque por supuesto no hay gente que se pueda identificar como 'All round contributors' pues no han realizado una cantidad suficiente de ediciones para poder determinar esto. En general, categorizados por intenciones no demasiado complejas.

En aquellos editores con actividad intermedia se comienzan a observar intenciones más refinadas y aparecen los 'All round contributors'. Sigue habiendo vándalos dentro de esta categoría. Por otro lado, se comienza a ver que los usuarios que encajarían con los roles 'Copy editors', 'Content shapers' y 'Layout shapers' con frecuencia realizan tareas de cualquiera de los 3 roles.

En el caso de los editores con alta actividad, es decir, más de 50 revisiones se observan no sólo todo los roles, si no que se refuerza ese solapamiento de los roles 'Copy editors', 'Content shapers' y 'Layout shapers'.

De esta manera, se ha comprobado la existencia de diferentes roles entre los usuarios de Wikipedia desde el punto de vista de la minería de procesos, reforzando aquellos resultados obtenidos por Daxenberger et al. Sin embargo, debido al solapamiento de los 3 roles 'Copy editors', 'Content shapers' y 'Layout shapers' se propone una alteración en su taxonomía de roles añadiendo un nuevo rol denominado 'Article fixers'.

Estos 'Article fixers' se encargan tanto de realizar tareas de arreglo de formato de Wikipedia y texto como de faltas de ortografía o mejoras en sintaxis y han sido observados tanto en aquellos autores con una actividad intermedia (en menor medida) como en aquellos con una alta actividad.

8.3. Conclusiones Minería Social

Dentro de la minería social, se han aplicado los algoritmos de Subcontracting y Handover of work a dos artículos diferentes de la Wikipedia obteniendo en ambos casos resultados muy similares verificando mutuamente los resultados obtenidos.

El algoritmo working together mide el traspaso de trabajo entre editores. Así, ambos artículos hemos obtenido estructuras de organización muy similares.

CAPÍTULO 8. CONCLUSIONES

Un porcentaje de los editores transpasan trabajo a un número muy reducido de editores, entre 1 y 2, implicando que sus contribuciones al artículo son realizadas durante un momento específico del tiempo y no han editado en el artículo a lo largo de una temporada. Por otro lado, aquellos editores más influyentes y con mayor número de revisiones muestran muchas conexiones implicando que han trabajado con muchos otros editores dando lugar a ediciones durante un periodo de tiempo extendido.

Por otro lado el Subcontracting nos muestra la 'subcontratación' entre editores, es decir, si un editor edita entre dos ediciones de otro, lo cual para suceder de modo consistente requiere de cierta colaboración o un número muy reducido de usuarios. Los resultados obtenidos son similares de nuevo entre ambos artículos.

La mayoría de usuarios realiza una sola sesión de edit donde realizan una cantidad variada de ediciones sin interrupciones de otros editores. Por otro lado existe un núcleo de editores de mayor influencia que realizan subcontrataciones entre ellos, mostrando que posiblemente exista una colaboración explícita entre estos usuarios.

Sin embargo, para afianzar estas conclusiones debemos observar los resultados de ambas métricas conjuntamente. Así, vemos que aquellos usuarios aislados en subcontracting, son aquellos que en handover of work cuentan con 2 conexiones pues han realizado una sesión de edit y nada más, de manera que su trabajo es continuado por otro editor dando lugar a la conexión en el handover of work.

Siguiendo esta misma línea de razonamiento, se ve por lo tanto que aquellos editores de mayor influencia representan diferente cara de la misma moneda en ambas métricas. En handover of work, estos usuarios forman un conjunto con numerosas conexiones entre ellos y a numerosos nodos mientras que en subcontracting estas conexiones se observan en mayor medida entre ellos. Es decir, mientras que realizan ediciones en general, hay momentos donde se realizan colaboraciones con otros editores de gran influencia dentro del artículo. Es decir existe una colaboración explícita entre los usuarios más influyentes dentro de un artículo.

En resumen, nos encontramos con que hay un gran porcentaje de editores dentro de Wikipedia que realizan una sola sesión de edición en un artículo para no volver, mientras que hay una minoría de editores asiduos que realizan tareas de edición durante períodos de tiempo grandes tanto casual como organizadamente con otros editores.

8.4. Conclusiones globales

De modo general, se pueden resumir los hallazgos realizados en:

1. No existe un proceso unificado de edición durante la evolución de un artículo.
2. A pesar de ello, la generalidad o especificidad de los editores en etapas tempranas del artículo muestra diferentes maneras de proceder. Es decir, los editores iniciales tienen influencia en la evolución posterior del artículo.
3. Se observan los diferentes roles de editor de la taxonomía desarrollada por Daxenberger et al. y se propone la adición del rol 'Article Fixer': editor que realiza tareas de formato

del artículo y texto así como mejoras en gramática y sintáxis. Es decir, hacen presentable el artículo sin entrar en la corrección de la información y sin añadir más contenido.

4. La mayoría de editores lo hacen de modo casual, con una sola sesión de ediciones en un artículo al que no vuelven.
5. Por otro lado, existen editores dedicados que editan durante la evolución de los artículos realizando colaboraciones organizadas puntuales con otros editores. Estos editores, además, suelen ser aquellos con la mayor influencia en los artículos (es decir, aquellos con mayor número de revisiones).

Capítulo 9

Conclusions

9.1. Process mining applied to the article: conclusions

Process mining was applied to a **corpus** composed of 8 *featured articles* from the Spanish Wikipedia. The event log represented each case as each article and each event as the revision. The resulting petri net was too complex to analyze by the human eye.

This complex net shows the complexity of the process followed by the articles. Wikipedia is an open community of shared knowledge. This means that anybody can perform an edit on a specific article anytime with no previous organization with other users. Hence, the number of existing editors in Wikipedia is incredibly high.

The high complexity and multiple paths in the net show that there is not an specific process followed in Wikipedia's article's edition history. However, in order to obtain any insight from the net, it was decomposed into simpler sub-nets. This sub-net are sill complex yet understandable and show different behaviours.

One of the sub-nets show intentionalities behind each revision akin to 'All round contributors' since the beginning of the net while the other show more specific intentions.

With this information, the following can be concluded: (i) There is not a unified process in the edition and evolution of Wikipedia articles. (ii) The specialization of the editors in the early stages of an article have an influence in the evolution of the article.

9.2. Process mining applied to the editor: conclusions

In this approach the process mining techniques were applied to discover the processes followed by the editors instead of the article. The editors were grouped based on the quantity of the revisions they made: low, medium and high activity.

The processes followed by the editors in their edit sessions show great similarities with the taxonomy of roles developed by Daxenberger et al[6] in 'Turbulent Stability of Emergent Roles'.

his taxonomy is composed by: 'Watchdogs', 'All round contributors', 'Quick and dirty editors', 'Copy editors', 'Content shapers', 'Layout shapers' and 'Vandals'.

In addition to this, in each set of grouped editors different behaviours can be appreciated.

Among the low activity users, those that performed less than 5 editions, all roles are found with the exception of 'All round-contributors'. However not very complex intentions are observed in this group.

Medium activity editors, with 5 to 50 revisions, show more refined combination of intentions and the 'all round contributors' start to appear. Also there is still some vandalism in this category. On the other hand, editors with the role of 'Copy editors', 'Content shapers' and 'Layout shapers' are seen to be performing tasks associated with any of those 3 roles instead of just sticking to one of them.

When it comes to the high activity users, those with more than 50 revisions, all the roles are observed. Again editors performing tasks associated with the role of 'Copy editors', 'Content shapers' and 'Layout shapers' are not sticking their activity to only one role but the three of them.

In conclusion, the editor roles developed by Daxenberger et al have been clearly seen with the methodology of the process mining, reinforcing its veracity. However, given the observed behaviour of editors performing tasks of layout and content shape modification and copy editing we propose the addition of a new role called 'Article fixers'.

'Article fixers' are those editors in charge of performing task related with the formatting of content and article and fixing typos, grammar and syntax. This role is similar to an 'All round contributor' but without performing any task related to the value and depth of the information given.

9.3. Social mining conclusions

Using social mining techniques, the subcontracting and handover of work algorithms were applied to two different articles of Wikipedia out of our used **corpus**. In both cases the obtained results were very similar.

Handover of work measures the flow of work from one editor to the other and both article show very similar organizational structures.

A percentage of the editors work only with a very reduced number of editors, 1 or 2 maximum. This means that their contributions to the article were probably perform in an specific moment of time and not during a large time span. On the other hand the most influential editors (those with the biggest amount of editions) have a lot of different connections meaning they edited prior to a lot of different users meaning that its relation with the article extends over time.

Subcontracting establishes a connection between two editors if one of the editors performed an edit between 2 edits of the other user. In communities with a lot of users such as Wikipedia, consistent connections of subcontracting between 2 editors might mean an explicit collaboration between said users as the probability of editing between edits of another user routinely is very low. Interestingly enough, both article yielded similar results.

The majority of the editors make only one edit session composed of one or more edits without interruption from other users. On the other hand, there is a cluster of editors with higher influence that is performing subcontracting with each other, showing that there is probably an explicit collaboration between those users.

However, in order to obtain the full picture, attention to both metrics must be paid jointly.

In the handover of work a big percentage of the users were seen to have handed over work to only 2 users and in subcontracting a big percentage of the users were isolated. In both cases the editors portrayed were the same, as it shows two faces of the same coin.

Following this very same line of thought applied to the cluster of influential editors in both metrics is exactly the same. Those that are the most influential in the community handed work over to a lot of users however their connections in subcontracting are more limited to the editors that composed that very same cluster. This can be translated into the following: the editors that put time and dedication into this process makes editions as they see it necessary with specific moments of explicit collaboration with other users. However this collaborations seem primarily restricted to prolific editors.

To sum up, there is a huge percentage of editors in Wikipedia that make editions in a specific moment of time only to never come back to that article while there is a minority of habitual users that edit during large time spans sometimes even collaborating explicitly with other editors.

9.4. Global conclusions

The findings of this document can be summed up in the following list:

1. There is not a unified editing process during the evolution of an article.
2. Despite that, the skill set of the editors in early stages of the article show different procedures. This means that the skill set of the initial editors have an influence in the later evolution of the article.
3. The different roles presented in the taxonomy developed by Daxenberger et al can be observed in the processes followed by the editors. However, the addition of a new role is suggested: 'Article fixer'. This role is for editors focused on the formatting of the content and the article that also fix grammar, syntax and typos without paying attention to the actual content of the article.
4. Most editors are casuals. They make an edit session to never come back to the article.

5. On the other hand, there are some dedicated editors that perform edits consistently throughout large timestamps with specific explicit collaborations with other editors. This editors are also the most influential ones when it comes to the amount of editions they make.

Capítulo 10

Future Work

Este proyecto ha sido realizado generando un corpus basado en 8 artículos destacados diferentes de la Wikipedia española. Sin embargo, el propio tamaño reducido del corpus limita en gran parte la posibilidad de generalizar los resultados obtenidos. Es debido a esto que como trabajo futuro sería muy interesante poder hacer uso de un corpus de mucho mayor tamaño o hacer uso de una Wiki de tamaño reducido como las de Wikia (por ejemplo Wiki Cocktails o Hitchikers Wiki) ya que poder analizar una comunidad entera en su conjunto aporta resultados muchos mas significativos que un fragmento de la misma.

El motivo por el cual esto no se ha realizado es porque cada Wiki existente hace uso de una API propia derivada de la encontrada en https://www.mediawiki.org/wiki/API:Main_page como por ejemplo <https://cocktails.fandom.com/api.php>. Sin embargo, como bien indican se encuentra en desarrollo y no funcionaba correctamente bajo el script desarrollado para 'Identifying Semantic Edit Intentions from Revisions in Wikipedia' [21] (más en: 11). Concretamente, la salida del diff entre revisiones se encontraba vacío.

Capítulo 11

Código

Este proyecto cuenta con código de autoría propia así como código realizado por terceras partes.

De este modo, el código desarrollado se encuentra alojado en https://github.com/FRYoussef/TFG_Wiki. Este código de autoría propia cuenta con licencia MIT y se encuentra compuesto por los siguientes scripts:

- `corpus_filter.py`: Este script contiene 4 filtros posibles de entre casual, low, intermediate y high para filtrar los editores de un historial de revisiones en función de su número de revisiones o para agruparlos como es el caso del filtro casual. También un filtro para eliminar las revisiones realizadas por BOTs. En este caso no requiere más input que un parámetro que especifique el tipo de filtro a aplicar pues hace uso automáticamente del corpus y del fichero generado por el siguiente script de la lista `editor_count_aggregator.py` y su output es el corpus filtrado.
- `editor_count_aggregator.py`: Agrega el número de revisiones realizado por cada autor, generando un fichero de texto donde se encuentran los autores y su conteo total de revisiones a través del corpus.
- `model_generation.ipynb`: Notebook donde se realiza un análisis detallado en busca del mejor modelo predictivo posible para detectar las intenciones inherentes a cada revisión, los mejores modelos determinados son exportados y usados por el siguiente script de la lista: `generate_predictions.py`
- `generate_predictions.py`: Hace uso de los 13 modelos generados (uno por cada intención) para predecir las intenciones tras cada revisión del archivo que se pase como input y da formato a los resultados modificando el csv inicial del dump de revisiones como output.
- `revision_id_extractor.py`: Su utilidad es crear un fichero csv que almacene el id de revisión y la intencionalidad. Debido a que en este punto la intención aún no se conoce, añade un 0 como placeholder en su lugar. El motivo de esto es que un script de una tercera parte hace uso de un archivo con este formato como input.
- `wikipedia_dump_downloader.py`: Se trata de una modificación de https://phabricator.wikimedia.org/diffusion/PWBC/browse/master/scripts/maintenance/download_dump.py.

py desarrollada por el autor de este proyecto en conjunto con Youssef El Faqir El Rha-zoui. La modificación consiste en la eliminación de dependencias con la librería PyWiki además de la implementación de utilidades como selección de idioma de la wiki de descarga, descargar una lista de artículos y la unión de los diferentes fragmentos descargados pues los artículos son divididos en fragmentos para su adecuada descarga.

Así, el código de terceras partes utilizado ha sido:

- `wiki_dump_parser.py`: Localizable en <https://github.com/Grasia/wiki-scripts> y desarrollado por Abel Serrano Juste como parte de un conjunto de scripts para obtener y procesar datos de una wiki. El programa en cuestión se trata de un script para obtener información útil y dar formato a los datos descargados por `wiki_dump_downloader.py` en forma de csv.
- `arffToCsv.py` es un script simple y sencillo para transformar un archivo arff a formato csv desarrollado por Haloboy777 y localizable en <https://github.com/haloboy777/arfftocsv> bajo licencia MIT
- Código desarrollado por Diyyi para el paper [21] con el objetivo de obtener las diferentes características de cada revisión en función de la anterior en una Wiki. Ha habido que realizar ligeros cambios como librerías deprecated o añadir diferentes funcionalidades como la posibilidad de añadir el id de revisión al dataset o decidir el idioma o tipo de Wiki. Se encuentra alojado en https://github.com/diyyi/Wiki_Semantic_Intention y se ha realizado un fork con los cambios en https://github.com/ignacioGarsami/Wiki_Semantic_Intention.

Apéndice A

Scripts

A.1. wiki_dump_downloader.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
This script downloads wikipedia's dumps

This script supports the following command line parameters:

    -article:#      The article's name to download (e.g. Scene7)
    -list:#         The list of articles to download
    -storepath:#    The stored file's path (without extension, it's going to
                    store as .xml)
    -lang:#        The language of the wiki to use, default "es"

#
# Authors: Youssef El Faqir El Rhazoui, Ignacio Garcia Sanchez-Migallon
# Date: 21/09/2018
# Distributed under the terms of the GPLv3 license.
#
# Based on: https://phabricator.wikimedia.org/diffusion/PWBC/browse/master/
#             scripts/maintenance/download_dump.py
#

from __future__ import absolute_import, division, unicode_literals
import xml.dom.minidom as minidom
import binascii
import os.path
import sys
import requests
import datetime
import shutil
from os import remove, symlink, urandom

try:
```

APÉNDICE A. SCRIPTS

```
from os import replace
except ImportError:    # py2
    if sys.platform == 'win32':
        import os

def replace(src, dst):
    """Rename a file or directory, overwriting the destination."""
    try:
        os.rename(src, dst)
    except OSError:
        remove(dst)
        os.rename(src, dst)
else:
    from os import rename as replace

class Dump_downloader():

    part = 1
    final_timestamp = 'undefined'
    curr_timestamp = 'first'
    total_size = 0
    url = ''
    availableOptions = {
        'wikiname': 'Wikipedia',
        'article': '',
        'list': '',
        'lang': 'es',
        'storepath': './',
        'download_limit': '$wgExportMaxHistory'
    }

    def __init__(self, params):

        self.availableOptions['article'] = params['article']
        if('storepath' in params):
            self.availableOptions['storepath'] = params['storepath'] + '/' +
                params['article']
        else:
            self.availableOptions['storepath'] += params['article']
        if('lang' in params):
            self.availableOptions['lang'] = params['lang']
        self.url = 'https://'+self.availableOptions['lang']+'.wikipedia.org
            /w/index.php?title=Special:Export'

    def download_chunk(self, download_file, data_request):
        """
        It downloads maxHistory of revisions as a part of final file
        """

        #print('Downloading dump from ' + self.availableOptions['wikiname'])
        # download_file = '{article}_part{part}.xml'.format(
        #     article=self.availableOptions['article'],
        #     part=self.part)
        temp_file = download_file + '_' + \
            binascii.b2a_hex(urandom(8)).decode('ascii') + '.part'
```

```

file_final_storepath = os.path.join(
    self.availableOptions['storepath'], download_file)
file_current_storepath = os.path.join(
    self.availableOptions['storepath'], temp_file)

# First iteration for atomic download with temporary file
# Second iteration for fallback non-atomic download
for non_atomic in range(2):
    try:
        print('Downloading file from: {}'.format(self.url))
        response = requests.post(self.url, data_request)
        if response.status_code == 200:
            with open(file_current_storepath, 'wb') as result_file:
                for data in response.iter_content(100 * 1024):
                    result_file.write(data)
        elif response.status_code == 404:
            print(
                'File with name "{}", '
                'and wiki "{}" ({}) isn\'t '
                'available'.format(
                    article=self.availableOptions['article'],
                    url=self.url,
                    wikiname=self.availableOptions['wikiname']))
        return
    else:
        return
# Rename the temporary file to the target file
# if the download completes successfully
if not non_atomic:
    replace(file_current_storepath, file_final_storepath)
    break
except (OSError, IOError):
    print('Error' + (OSError.strerror or IOError.strerror))

try:
    remove(file_current_storepath)
except (OSError, IOError):
    print('Error' + (OSError.strerror or IOError.strerror))

# If the atomic download fails, try without a temporary file
# If the non-atomic download also fails, exit the script
if not non_atomic:
    print('Cannot make temporary file, falling back to non-'
          'atomic download')
    file_current_storepath = file_final_storepath
else:
    return False

size = (sum(len(chunk) for chunk in response.iter_content(8196))) /1000000
self.total_size += size
print('\nDone! File stored as {}\nTotal: {}MB'.format(
    file_final_storepath, round(size, 3)))
return

def get_final_timestamp(self):
    """

```

```

It gets the last article's revision and updates the final timestamp
"""
ret = 0
data={'pages': self.availableOptions['article'], 'curonly': 'true',
       'action': 'submit'}
file_name = '{}_temp.xml'.format(self.availableOptions['article'])
self.download_chunk(file_name, data)

# Now, we are going to find the revision's timestamp.
file_name = os.path.join(self.availableOptions['storepath'], file_name)
doc = minidom.parse(file_name)
if not doc.getElementsByTagName('timestamp').length:
    print('Timestamp not found')
    ret=1
else:
    self.final_timestamp = timestamp_to_datetime(
        doc.getElementsByTagName('timestamp')[0].firstChild.nodeValue)
remove(file_name)
self.total_size = 0
print('Removed the file {}'.format(file_name))
return ret

def join_chunks(self):
    """
    This method joins the different chunks of one article into one
    xml file
    """
    print('Joining chunks....')
    file_name = os.path.join(self.availableOptions['storepath'], ,
                           '{}.xml'.format(self.availableOptions['article']))

    with open(file_name, 'w+', encoding='utf8') as file:
        for i in range(1, self.part):
            chunk_name = os.path.join(self.availableOptions['',
                                         'storepath'], '{0}_part{1}.xml',
                                         .format(self.availableOptions['article'], i))
            with open(chunk_name, encoding='utf8', errors = 'ignore',
                      ) as chunk_file:
                content_file = chunk_file.read()
                content = content_file.splitlines()
                if i != 1:
                    content = content[44:]
                for line in content:
                    if i != self.part - 1:
                        if( "</page>" not in line and "</mediawiki>" not in line):
                            file.write(line + '\n')
                    else:
                        file.write(line + '\n')
            os.remove(chunk_name)
        file.close()
        print('
        ')
    print('The article {} has been merged into one file'.format(
        self.availableOptions['article']))

```

```

        print( '
        ')
def run(self):
    """
    It manages article's download joining the parts
    """
    if not os.path.exists(self.availableOptions['storepath']):
        os.makedirs(self.availableOptions['storepath'])

    if self.get_final_timestamp():
        print('The article \'{0}\' isn't available'.format(self.availableOptions['article']))
        return 1
    else:
        curr = ''
        while curr == '' or self.final_timestamp > curr:
            file_name = '{art}-part{part}.xml'.format(art=self.availableOptions['article'],
                                                       part=self.part)
            data={'pages': self.availableOptions['article'], 'offset': self.curr_timestamp,
                  'limit': self.availableOptions['download_limit'], 'action': 'submit'}
            self.download_chunk(file_name, data)
            timestamps = minidom.parse(os.path.join(self.availableOptions['storepath'],
                                                      file_name)).getElementsByTagName('timestamp')
            if not timestamps.length:
                #No timestamps, you have to handle the error
                return 1
            else:
                curr = timestamp_to_datetime(timestamps.length - 1).firstChild.nodeValue
                + datetime.timedelta(seconds=1)
                self.curr_timestamp = '{0}-{1}-{2}T{3}:{4}:{5}Z'.format(
                    curr.strftime('%Y'),
                    curr.strftime('%m'), curr.strftime('%d'), curr.strftime('%H'),
                    curr.strftime('%M'), curr.strftime('%S'))
        self.part += 1

        print( '
        ')
        print('The article \'{0}\' has been downloaded as {1} chunks in {2}',
              .format(self.availableOptions['article'], self.part, self.availableOptions['storepath']))
        print('Total downloaded: {}MB'.format(round(self.total_size), 3))
        print( '
        ')
        self.join_chunks()

```

```
def timestamp_to_datetime(timestamp):
    """
    It takes a ISO 8601 date from text and returns a object datetime
    """
    timestamp = timestamp.split('T')
    date = timestamp[0].split('-')
    time = timestamp[1].replace('Z', '').split(':')
    return datetime.datetime(int(date[0]), int(date[1]), int(date[2]), int(time[0]),
                            int(time[1]), int(time[2]), 0)

def dump_list(name):
    """
    This method process the file with the list of articles to download
    It reads the file splitting it in lines and stores each article name
    """
    with open(name) as file:
        content = file.read().splitlines()

    articleList = []

    for line in content:
        split = line.split('/')
        #Directions are "https://en.wikipedia.org/wiki/articleName hence the
        #split[4] hardcoded.
        articleList.append(split[4])

    return articleList

def main(*args):
    """
    Process command line arguments and invoke bot.

    If args is an empty list, sys.argv is used.

    @param args: command line arguments
    @type args: unicode
    """
    opts = {}
    unknown_args = []

    local_args = sys.argv[1:]
    for arg in local_args:
        option, sep, value = arg.partition(':')
        if option.startswith('-'):
            option = option[1:]
            if option == 'article':
                opts[option] = value or input(
                    'Enter the article: ')
                continue
            elif option == 'list':
```

```

        opts[option] = value or input(
            'Enter the list of articles: ')
        continue
    elif option == 'storepath':
        opts[option] = os.path.abspath(value) or input(
            'Enter the store path: ')
        continue
    elif option == 'lang':
        opts[option] = value or input(
            'Enter the language of the wiki to use: ')
        continue

unknown_args += [arg]

missing = []
if 'list' not in opts and 'article' not in opts:
    missing += ['-article']
    missing += ['-list']

if missing or unknown_args:
    print('HELP: python script_path -article:article_name [-storepath:path]')
    print('HELP: It is also possible to use -list instead of -article for')
    print('lists of articles')
    return 1

if 'list' in opts:
    print('Processing list of links')
    if 'storepath' not in opts:
        folder = opts['list']
        folderName = folder.split('.')
        if os.path.exists(folderName[0]):
            shutil.rmtree(folderName[0])
        os.mkdir(folderName[0])
        opts['storepath'] = folderName[0]

    articleList = dump_list(opts['list'])
    for article in articleList:
        opts['article'] = article
        dump_down = Dump_downloader(opts)
        dump_down.run()
    return 2
else:
    dump_down = Dump_downloader(opts)
    dump_down.run()

return 0

if __name__ == '__main__':
    sys.exit(main())

```

A.2. revision_id_extractor.py

```
import numpy as np
```

APÉNDICE A. SCRIPTS

```
import pandas as pd
import sys
import os

title = ''

def main(*args):
    """
    Process command line arguments and invoke bot.

    If args is an empty list, sys.argv is used.

    @param args: command line arguments
    @type args: unicode
    """
    opts = {}
    local_args = sys.argv[1:]
    for arg in local_args:
        option, sep, value = arg.partition(':')
        if option.startswith('-'):
            option = option[1:]
            if option == 'file':
                opts[option] = value or input(
                    'Enter the file: ')
            continue
    article_ids = revision_extractor(opts)
    article_ids.revision_id_extractor()

    return 0

class revision_extractor():

    article = {'file': ''}

    def __init__(self, params):
        self.article['file'] = params['file']

    def revision_id_extractor(self):

        base_file_name = 'data/' + self.article['file'] + '/' + self.article['file'] + '.csv'
        base_file = pd.read_csv(base_file_name, error_bad_lines=False)

        output_file_name = 'revision_ids' + self.article['file'] + '.csv'
        output_file = pd.DataFrame(columns=['rev_id', 'labels'])

        for rev in base_file['revision_id']:
            output_file = output_file.append({'rev_id': rev, 'labels': 0},
                                             ignore_index=True)

        output_file.to_csv(path_or_buf='revision_ids_files/' +
                           output_file_name, index=False)

        print('Revision ids file of ' + self.article['file'] + ' created')
```

```
def dummy_labels_adder(filename):
    return 0

if not os.path.exists('revision_ids_files'):
    os.makedirs('revision_ids_files')
main()

\end{lstlisting}

\section{generate\_predictions.py}

\begin{lstlisting}[language=python]
import numpy as np
import pandas as pd
from sklearn.externals import joblib
import sys
import os

title = ''

def main(*args):
    """
    Process command line arguments and invoke bot.
    If args is an empty list, sys.argv is used.
    @param args: command line arguments
    @type args: unicode
    """
    opts = {}
    local_args = sys.argv[1:]
    for arg in local_args:
        option, sep, value = arg.partition(':')
        if option.startswith('-'):
            option = option[1:]
            if option == 'file':
                opts[option] = value or input(
                    'Enter the file: ')
                continue
            if option == 'base_file':
                opts[option] = value or input(
                    'Enter the base file: ')
                continue
    predictions_process = predictions_generator(opts)
    predictions_process.generate_predictions()

    return 0

class predictions_generator():

    article = {'file': '', 'base_file': ''}

    def __init__(self, params):
        self.article['file'] = params['file']
```

APÉNDICE A. SCRIPTS

```
self.article[ 'base_file' ] = params[ 'base_file' ]  
  
def generate_predictions(self):  
    base_file_name = self.article[ 'file' ]  
    base_file = pd.read_csv(base_file_name, error_bad_lines=False)  
  
    intentions = [ 'counter-vandalism', 'fact-update', 'refactoring', 'copy-  
        editing', 'wikification',  
        'vandalism', 'simplification', 'elaboration', 'verifiability',  
        'process', 'clarification', 'disambiguation',  
        'point-of-view' ]  
  
    #Load all the classifiers for each intention  
    classifiers = {}  
    for intention in intentions:  
        model = joblib.load('trained_models/' + intention + '_classifier.  
            plk')  
        classifiers[intention] = model  
  
    #To store the predicted values of each intention  
    results = { 'revision_id':[], 'counter-vandalism':[], 'fact-update':[], '  
        refactoring':[], 'copy-editing':[], 'wikification':[],  
        'vandalism':[], 'simplification':[], 'elaboration':[], 'verifiability':  
        [], 'process':[], 'clarification':[], 'disambiguation':[],  
        'point-of-view':[] }  
  
    #We add the revision_id to the results to know the revision of each  
    results[ 'revision_id' ] = base_file.iloc[:, 0]  
    #Load the features as X to predict  
    X = base_file.iloc[:, :-14]  
    for intention in intentions:  
  
        print("Predicting labels for " + intention)  
        predictions_file = classifiers[intention].predict(X)  
        for line in predictions_file:  
            results[intention].append(line)  
  
    print("Labels predicted")  
    results_dataframe = pd.DataFrame.from_dict(results)  
    output = pd.read_csv(self.article[ 'base_file' ], sep=',')  
    #create array of Nones to store the labels with the appropriate format (  
        one single column for all intentions  
    labels = [None] * len(results_dataframe.index)  
  
    for intention in intentions:  
  
        offset = 0  
  
        for line in results[intention]:  
  
            if labels[offset] == None:  
                labels[offset] = str(results[ 'revision_id' ][offset]) + ':'  
            if line == 1:  
                labels[offset] = labels[offset] + intention + ','
```

```

    offset += 1

labels_dataframe = pd.DataFrame(columns=[ 'revision_id' , 'intentionality' ,
                                         ])
for line in labels:

    data = line.split( ':' )
    data_id = data[0]
    data_intention = data[1].split( ',' )
    data_intention.remove( '' )

    data = pd.DataFrame({ 'revision_id' : [ data_id ] , 'intentionality' : [
        data_intention ] })
    labels_dataframe = labels_dataframe.append(data)

labels_dataframe[ 'revision_id' ]=labels_dataframe[ 'revision_id' ].astype(
    int)
output = output.merge(labels_dataframe , on = 'revision_id' , how = 'inner')

output.to_csv(path_or_buf = self.article[ 'base_file' ] , index = False)

main()

```

A.3. corpus_filter.py

```

import numpy as np
import pandas as pd
import sys
import os

def main(*args):
    """
    Process command line arguments and invoke bot.

    If args is an empty list, sys.argv is used.

    @param args: command line arguments
    @type args: unicode
    """
    opts = {}
    local_args = sys.argv[1:]
    for arg in local_args:
        option, sep, value = arg.partition( ':' )
        if option.startswith( '-' ):
            option = option[1:]
            if option == 'filter':
                opts[option] = value or input(
                    'Enter the filter type: ')
                continue
    filtro_corpus = corpus_filter(opts)

```

APÉNDICE A. SCRIPTS

```
filtro_corpus.filterApplier()

return 0

def low_filter(self):
    for i in range(0, len(self.count)):
        if self.count[i] == '1' or self.count[i] == '[2-5]':
            self.editors_to_keep.append(self.editors[i])

    corpus = self.corpus.loc[self.corpus['resource'].isin(self.editors_to_keep)]
    corpus.to_csv(path_or_buf = 'data/corpus/low_editors.csv', index = False)

def intermediate_filter(self):
    for i in range(0, len(self.count)):
        if self.count[i] != '1' and self.count[i] != '[2-5]' and self.count[i] != '[50-100]' and self.count[i] != '+100':
            self.editors_to_keep.append(self.editors[i])

    corpus = self.corpus.loc[self.corpus['resource'].isin(self.editors_to_keep)]
    corpus.to_csv(path_or_buf = 'data/corpus/intermediate_editors.csv', index = False)

def high_filter(self):
    for i in range(0, len(self.count)):
        if self.count[i] != '1' and self.count[i] != '[2-5]' and self.count[i] != '[5-10]' and self.count[i] != '[10-20]' and self.count[i] != '[20-50]':
            self.editors_to_keep.append(self.editors[i])

    corpus = self.corpus.loc[self.corpus['resource'].isin(self.editors_to_keep)]
    corpus.to_csv(path_or_buf = 'data/corpus/high_editors.csv', index = False)

def casual_filter(self):
    for i in range(0, len(self.count)):
        if self.count[i] == '1' or self.count[i] == '[2-5]':
            self.editors_to_keep.append(self.editors[i])
    for j in range(0, len(self.corpus)):
        if self.corpus.iloc[j, 6] in self.editors_to_keep:
            self.corpus.iloc[j, 6] = 'Casual'

    self.corpus.to_csv(path_or_buf = 'data/corpus/corpus_casual.csv', index = False)

class corpus_filter():

    filtro = {'filter': ''}
```

```
corpus = pd.read_csv('data/corpus/corpus.csv')

editor_count = pd.read_csv('data/corpus/editor_count.csv')

count = editor_count['cluster'].values

editors = editor_count['resource'].values

editors_to_keep = []

def __init__(self, params):

    self.filtro['filter'] = params['filter']

def filterApplier(self):

    call = self.filtro['filter']
    if call == 'low':
        low_filter(self)
    elif call == 'intermediate':
        intermediate_filter(self)
    elif call == 'high':
        high_filter(self)
    elif call == 'casual':
        casual_filter(self)
    else:
        print("Introduzca un filtro de entre [low, intermediate, high, casual]")

main()
```

Bibliografía

- [1] *3.2.4.3.1. sklearn.ensemble.RandomForestClassifier*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [2] Wil M. P. Aalst. *Decomposing Petri Nets for Process Mining: A Generic Approach*. Vol. 31. Ene. de 2012. DOI: 10.1007/s10619-013-7127-5.
- [3] Wil M. P. Van Der Aalst, Hajo A. Reijers y Minseok Song. “Discovering Social Networks from Event Logs”. En: *Computer Supported Cooperative Work (CSCW)* 14.6 (2005), 549–593. DOI: 10.1007/s10606-005-9005-9.
- [4] Wil M. P. Van Der Aalst y Minseok Song. “Mining Social Networks: Uncovering Interaction Patterns in Business Processes”. En: *Lecture Notes in Computer Science Business Process Management* (2004), 244–260. DOI: 10.1007/978-3-540-25970-1_16.
- [5] Wil van der Aalst y col. *Process Mining Manifesto*. 2011. URL: https://doi.org/10.1007/978-3-642-28108-2_19.
- [6] Ofer Arazy y col. “Turbulent Stability of Emergent Roles: The Dualistic Nature of Self-Organizing Knowledge Co-Production”. En: *Information Systems Research* 27 (ene. de 2017), págs. 792-812. DOI: 10.1287/isre.2016.0647.
- [7] Hicheur Awatef y col. “Process Mining in the Education Domain”. En: feb. de 2015.
- [8] Soumen Chakrabarti y col. *Data Mining Curriculum: A Proposal*. 2006. URL: https://www.kdd.org/exploration_files/CURMay06.pdf.
- [9] B. F. van Dongen y col. *The ProM Framework: A New Era in Process Mining Tool Support*. 2005. URL: https://link.springer.com/chapter/10.1007/11494744_25.
- [10] Haibo He y col. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning”. En: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (2008). DOI: 10.1109/ijcnn.2008.4633969.
- [11] James L. Peterson. “Petri Nets”. En: *ACM Computing Surveys* 9.3 (1977), 223–252. DOI: 10.1145/356698.356702.
- [12] *ProM 6.8*. 2018. URL: <http://www.promtools.org/doku.php?id=prom68>.
- [13] Michal Rosik. *3 Industries and Companies Doing Process Mining Right*. URL: <https://www.minit.io/blog/3-industries-and-companies-doing-process-mining-right>.
- [14] Pnina Shachaf y Noriko Hara. “Beyond vandalism: Wikipedia trolls”. En: *Journal of Information Science* 36.3 (2010), 357–370. DOI: 10.1177/0165551510365390.

- [15] *sklearn.dummy.DummyClassifier*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html>.
- [16] *sklearn.svm.SVC*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>.
- [17] Marina Sokolova y Guy Lapalme. “A systematic analysis of performance measures for classification tasks”. En: *Information Processing Management* 45.4 (2009), 427–437. DOI: 10.1016/j.ipm.2009.03.002.
- [18] Vilaythong Southavilay, Kalina Yacef y Rafael A. Calvo. “Analysis of Collaborative Writing Processes Using Hidden Markov Models and Semantic Heuristics”. En: *2010 IEEE International Conference on Data Mining Workshops* (2010). DOI: 10.1109/icdmw.2010.118.
- [19] A Weijters, Wil M. P. Aalst y Alves A K Medeiros. *Process Mining with the Heuristics Miner-algorithm*. Vol. 166. Ene. de 2006.
- [20] *XES*. 2018. URL: <http://xes-standard.org/>.
- [21] Diyi Yang y col. “Identifying Semantic Edit Intentions from Revisions in Wikipedia”. En: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (2017). DOI: 10.18653/v1/d17-1213.