UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Incidencia de Idiomas Populares en la Lengua Española con Big Data: Análisis Masivo de Datos Mediante Amazon Elastic MapReduce y Google N-grams
Autores: Julián De Luca
Director: Waldo Hasperué
Codirector: Franco Chichizola
Asesor profesional: Ismael Pablo Rodriguez
Carrera: Licenciatura en Sistemas

Resumen

La presente tesina analiza, diseña e implementa una solución para el problema de la detección de neologismos y extranjerismos en la lengua Española. Para este fin, realizamos un análisis de investigaciones previas y problemáticas surgidas, y decidimos realizar un aporte mediante un sistema de cloud computing.

Nuestra solución se centra en el uso de tecnologías de análisis masivo de datos para el tratamiento de corpus grandes de una manera eficiente. Realizamos un repaso general de las tecnologías existentes, sin especificar herramientas puntuales, para luego introducir herramientas concisas. Logramos alcanzar una solución sencilla pero eficaz y escalable mediante el uso de herramientas provistas por Amazon Web Services, utilizando el corpus público llamado Google Books Ngrams.

Finalmente, abrimos la posibilidad de utilizar otras herramientas, y la misma metodología en otro tipo de estudios. Demostramos que logramos un aporte a la comunidad de lingüística computacional, y seguiremos trabajando en el tema en cuestión.

Palabras Claves

Google Books Ngrams, Amazon Web Services, AWS, Cloud Computing, Big Data, MapReduce, EMR, Hive, Neología, Extranjerismos.

Conclusiones

Se desarrolló un sistema de detección de neologismos novedoso que permite apreciar el uso de cloud computing como herramienta potente para su procesamiento.

Se otorga a futuros investigadores una herramienta de punto de partida para realizar sus estudios, con resultados claros y software extensible y modificable.

Trabajos Realizados

Se desarrolló un software para el lanzamiento de clústeres EMR en AWS, que mediante Hive permite tratar información de manera distribuida sencillamente.

Explicamos la instalación, funcionamiento, importancia y relación del sistema, y se propuso una metodología novedosa para la detección de neologismos, impulsada por teorías de otros autores.

Trabajos Futuros

Se propone:

- *Extensión para el procesamiento de información en distintos clústeres, que permita paralelizar tareas independientes.*
- *Extensión para el filtrado de extranjerismos sobre los neologismos detectados.*
- *Implementar la misma metodología en un corpus distinto, con unidades de tiempo menores a 1 año, posiblemente con información de internet como pueden ser foros.*

Fecha de la presentación: Marzo 2016

Universidad Nacional de La Plata

Facultad de Informática

Incidencia de Idiomas Populares en la Lengua Española con Big Data: Análisis Masivo de Datos Mediante Amazon Elastic MapReduce y Google N-grams

Julián De Luca

Directores:
Waldo Hasperué y
Franco Chichizola

Asesor:
Ismael Pablo Rodríguez

Marzo 2016

Si para recobrar lo recobrado
debí perder primero lo perdido,
si para conseguir lo conseguido
tuve que soportar lo soportado,

si para estar ahora enamorado
fue menester haber estado herido,
tengo por bien sufrido lo sufrido,
tengo por bien llorado lo llorado.

Porque después de todo he comprobado
que no se goza bien de lo gozado
sino después de haberlo padecido.

Porque después de todo he comprendido
por lo que el árbol tiene de florido
vive de lo que tiene sepultado.

- Francisco Luis Bernárdez

Índice

1 Introducción

1.1 Objetivos

El objetivo general de esta tesis es estudiar las herramientas de cloud computing para el tratamiento de datos masivos sobre textos en distintos idiomas. Para realizar dicho estudio se propone la utilización del corpus Google Books Ngrams (Michel et al., 2010), el cual es una base de datos de libros de más de 2TB. Como resultado de este análisis se pretende otorgar un panorama sobre el estudio de la neología[1] desde un punto de vista computacional, en especial el que prevemos que será el camino a seguir de los estudios de ésta índole que es el del análisis masivo de datos.

Introduciremos conceptos generales de lingüística y cloud computing relevantes para nuestro estudio, e intentaremos desambiguar algunos términos que otros autores suelen confundir a la hora de dar una definición.

Analizaremos las diferentes configuraciones en un ambiente de programación en la nube, el cual será el caso de Amazon Web Services (AWS), junto con los resultados de rendimiento, que serán de gran utilidad para futuros investigadores de lingüística computacional y problemas relacionados.

Con una configuración estándar, desarrollaremos un algoritmo para la búsqueda de extranjerismos y detección de neologismos en el idioma Español a partir de su intersección con otros idiomas, mediante la utilización del corpus Google Books Ngrams. Dado que el corpus es diacrónico[2], detectaremos los extranjerismos y neologismos a lo largo del tiempo tanto como el corpus nos lo permita y nos sea de interés, aprovechando la gran ventaja de poseer un corpus en varios idiomas.

Explicaremos el funcionamiento del algoritmo y cómo puede ser utilizado para analizar otros idiomas además del Español, y modificado para cumplir otros objetivos o refinar su funcionamiento.

Culminaremos con el análisis de los resultados, resumiendo el total de la investigación y los futuros pasos a seguir, presentando los neologismos y extranjerismos detectados.

1.2 Motivación

Luego de investigar numerosos artículos relacionados con el estudio de la neología, notamos que existían varios huecos a completar. Diversos autores realizaban investigaciones específicas sobre ciertos temas e idiomas, dándole una escasa reutilización a estudios anteriores relacionados, debido a que éstos no son sencillos de llevar a un entorno abstracto para ser aplicados en un estudio diferente. Algunos autores se percataron de este problema (Stenetorp, 2010) e intentaron

[1]La neología es la ciencia que estudia la aparición de nuevos términos o elementos en una lengua.
[2]La diacronicidad expresa la relación de un término en el tiempo, en éste caso los mismos términos aparecen sucesivamente año a año, pudiendo su significado cambiar según su cantidad de ocurrencias o contexto.

unificar los conocimientos de investigaciones previas, y señalan que el origen del problema podría recaer en que los pocos profesionales dedicados al tema en cuestión son contratados por editoriales que se interesan en algún idioma o tema en particular, lo que genera un problema para los estudios académicos.

Sin embargo, el software que desarrollaron no se encuentra abierto al público y futuros investigadores vuelven a encontrarse en la misma disyuntiva entre comenzar de cero o seguir investigando cuantos artículos les sea posible para concluir cómo avanzar en su investigación. Es necesario el desarrollo de software libre que permita a futuros investigadores analizar, cuestionar, extender y reutilizar el código empleado para la extracción y análisis de neologismos.

A su vez, no existen estudios que detallen de forma completa cómo tratar con el corpus de manera eficiente. Pareciera que ninguno de los investigadores se topó con un problema de rendimiento. Esto puede deberse a que muchas de las investigaciones analizan corpus pequeños (diarios y revistas) día a día, y no necesitan de una gran eficiencia o infraestructura para obtener un resultado en un tiempo razonable.

En esta tesis abordaremos dicha problemática utilizando herramientas de cloud computing que permiten el análisis de enormes volúmenes de datos en poco tiempo y con mínimo esfuerzo.

A pesar de existir numerosos estudios relacionados al análisis de influencias de una lengua en otra, encontramos interesante tanto la computación aplicada a la lingüística como la lingüística *per se* y aprovechamos los nuevos recursos disponibles de la época junto con nuestra propia motivación para impulsar la siguiente investigación.

1.3 Estructura de la Tesis

Comenzamos explicando los objetivos y motivación que impulsaron el desarrollo de esta tesis en el capítulo presente (capítulo 1). En los siguientes capítulos introduciremos conceptos de big data y cloud computing (capítulo 2) sin focalizarnos en el problema a resolver o en herramientas puntuales para permitir al lector un mejor entendimiento de estos conceptos. Dicha problemática será planteada junto con un análisis de problemas similares que otros autores han tenido, para luego introducir herramientas puntuales para el desarrollo de una solución (capítulo 3). Una vez comprendidos los conceptos anteriormente mencionados, explicaremos la configuración e implementación necesarias para la solución propuesta, y analizaremos los resultados obtenidos con esta metodología (capítulo 4). Por último realizaremos un repaso final de lo visto e intentaremos otorgar un resumen útil para la resolución de problemas similares futuros (capítulo 5).

2 Conceptos de Análisis Masivo de Datos

En este capítulo se verán los conceptos y tecnologías aplicadas en el desarrollo de este trabajo y la manera en que se relacionan.

2.1 Big Data

El término *Big Data* se refiere a la captura, gestión y procesamiento de una cantidad de datos que supera la capacidad del hardware y software habitual. Las tecnologías de Big Data dan paso a una nueva generación de tecnologías y arquitecturas diseñadas para económicamente extraer valor de volúmenes de datos largos y variados.

Algunas de las tecnologías para el procesamiento que surgieron son MapReduce (Dean&Ghemawat, 2004), y Hadoop[3], que es open source. Con ellos se puede procesar grandes cantidades de datos, pero lo más interesante es que éstos no requieren estar ordenados en una base de datos bajo cierto criterio, o ser homogéneos. Es entendible que en Big Data los datos estén desordenados y sean heterogéneos, ya que provienen de cientos de diversas fuentes. Estos pueden ser generados por las personas (al interactuar en redes sociales, enviar emails y mensajes, llenar encuestas), por transacciones cotidianas (compras con tarjeta de crédito, pago de facturas e incluso llamadas), por interacción entre máquinas (los GPS suelen transmitir datos anónimos a los servidores, sensores de temperatura, luz, sonido, entre otros), por biometría (huellas digitales, lecturas de retina ocular, y otros datos recolectados por gobiernos y agencias de inteligencia) y otros campos como la astronomía y la física (por ejemplo el CERN genera 30 petabytes de información al año, provenientes sólo de colisiones generadas con el colisionador de hadrones[4]).

El procesamiento de grandes cantidades de datos heterogéneos y desestructurados conlleva algunas desventajas. Una de las principales es que a la hora de buscar relación entre distintos datos, y encontrarlas, descubriremos el *qué* pero no el *cómo* o *por qué*. Esto puede ayudar en la generación de estadísticas que sirven para la predicción de, por ejemplo, movimientos de bolsa.

[3]https://hadoop.apache.org/(Visitado: Enero 2016)
[4]http://home.cern/about/computing/ (Visitado: Enero 2016)

Unidades de Medición de Datos

Para facilitar el entendimiento de la magnitud de Big Data, veremos en la tabla 2.1.1 la relación entre las distintas unidades para la medición de datos. Utilizaremos como unidad base el Terabyte, ya que la mayoría de los discos rígidos de las computadoras de hoy en día poseen una capacidad de almacenamiento de entre medio y 2 terabytes, y es la unidad más grande conocida por el público general.

Unidad (Abreviación)	Equivalente en Terabytes
Kilobyte (KB)	$1/1024^3 = 0.00000000093$TB
Megabyte (MB)	$1/1024^2 = 0.00000095$TB
Gigabyte (GB)	$1/1024 = 0.000976$TB
Terabyte (TB)	1TB
Petabyte (PB)	1024TB
Exabyte (EB)	$1024^2 = 1048576$TB
Zettabyte (ZB)	$1024^3 = 1073741824$TB

Tabla 2.1.1: Comparación entre Terabyte y otras unidades de datos.

2.2 Cloud Computing

Cloud Computing es una forma especializada de computación distribuida que introduce la utilización de modelos para la provisión remota de recursos escalables y medibles.

El término *nube* (*Cloud*) se refiere a un ambiente de tecnología diseñado para la provisión remota de recursos tecnológicos. Surgió como una metáfora de internet, que es en esencia una red de redes que provee acceso remoto a un conjunto de recursos informáticos descentralizados. Antes del surgimiento de *Cloud Computing*, el símbolo de la nube representaba a la internet en documentación y especificaciones de arquitecturas web de software.

Clusters

Un *cluster* es un conjunto de recursos informáticos independientes que están interconectados y trabajan como un sistema individual. Se reducen los fallos del sistema y se incrementa la confiabilidad y disponibilidad ya que la redundancia y el retomo de tareas se manejan dentro del sistema.

Un pre-requisito general de los cluster es que sus componentes posean hardware y sistema operativo similares o idénticos, para proveer niveles de rendimiento equivalentes cuando un componente que falla debe ser remplazado por otro. Los componentes del cluster son mantenidos en sincronización mediante enlaces dedicados de alta velocidad.

Grid Computing

La computación en red (*Grid Computing*) provee una plataforma en la cual los recursos de computación están organizados en una o más *pools* (piscinas/regiones) lógicas. Éstas pools son coordinadas para proveer una red distribuida de alto rendimiento, muchas veces llamadas *Super Virtual Computer.*

Grid Computing se diferencia de la computación en clusters en que el primero es mucho menos acoplado y más distribuido. Como consecuencia, los sistemas de Grid Computing involucran recursos heterogéneos y dispersos geográficamente.

Virtualización

La *virtualización* representa una plataforma de tecnología utilizada para la creación de instancias virtuales de recursos tecnológicos. Una capa de software de virtualización permite que recursos tecnológicos físicos provean varias imágenes virtuales de sí mismos para que sus capacidades subyacentes puedan ser compartidas por múltiples usuarios simultáneamente.

Previo a las tecnologías de virtualización, el software estaba limitado al hardware disponible. La virtualización rompió con estas dependencias entre el software y el hardware dado que el último y sus requerimientos pueden ser simulados mediante software de emulación ejecutado en ambientes virtualizados.

Beneficios de Cloud Computing

Existen diferentes beneficios por los cuales *cloud computing* se ha vuelto popular en las últimas décadas, en especial la última.

Al igual que en la compra de productos en un mercado mayorista, existen proveedores de servicios de *cloud* que nos permiten la contratación de infraestructura de procesamiento potente a precios proporcionales (por lo general a su tiempo de uso y tamaño) evitando la necesidad de realizar una gran inversión en la compra de ésta.

La disminución de costos por adelantado nos permite como clientes de infraestructura de *cloud* comenzar con pocos recursos y luego crecer a medida que sea necesario, contratando más recursos. Los fondos sobrantes por esta disminución pueden ahora ser invertidos en la actividad principal de la organización. Además, los proveedores de servicios de *cloud* suelen ubicar su infraestructura en lugares dónde es conveniente, por ejemplo con costos de internet bajos, para incrementar el ahorro monetario, o donde el clima sea adecuado la mayor parte del año para evitar costos de refrigeración.

Varios clientes de servicios de *cloud* comparten los mismos recursos. Los costos de operación e ineficiencias pueden ser reducidos mediante la aplicación de prácticas y patrones probados para la optimización de arquitecturas de *cloud* y su manejo.

Thomas Erl (2013) habla en su libro sobre beneficios medibles:

- Acceso por demanda a recursos, donde el pago es por consumo en tiempo real, muchas veces en tiempos cortos (por ejemplo

procesadores por hora) y la capacidad de liberar estos recursos cuando ya no se necesitan (conocido como *elasticidad*).

- El acceso a recursos cuasi-ilimitados.

- La capacidad de agregar o eliminar recursos a grado fino, como por ejemplo modificar la capacidad de almacenamiento a nivel de gigabytes.

- La abstracción de infraestructura para que las aplicaciones no estén limitadas a ciertos dispositivos o ubicaciones y sean fáciles de transportar si así se desea.

Más allá de los beneficios financieros que se pueden detectar, calcular y evaluar los costos reales puede ser una tarea difícil. La decisión de adoptar una estrategia de *cloud computing* involucra mucho más que la simple comparación entre el costo de contratación de servicios y la compra de infraestructura. También debemos tomar en cuenta los riesgos financieros de escalabilidad dinámica y los posibles problemas de sobre-provisión (adquisición exagerada de infraestructura) o infra-provisión (adquisición insuficiente de infraestructura).

Es interesante considerar que los servicios de cloud son ofrecidos listos para usar, la implementación y detalles técnicos de los clusters quedan abstraídos por el proveedor. Esto no sucede en el caso de adquirir la infraestructura. Se necesitaría invertir dinero y tiempo en preparar el sistema.

Otros factores importantes que inciden en una organización son la confiabilidad y disponibilidad de sus servicios. Cortes inesperados en éstos pueden producir desconfianza por los consumidores, así como también grandes pérdidas si se producen en períodos cortos donde se presencia un pico de uso.

Los proveedores de servicios de cloud se caracterizan por poseer una gran cantidad de recursos, los cuales minimizan o eliminan por completo la posibilidad de cortes en el servicio. Al tener una arquitectura modular, reaccionan mejor a condiciones excepcionales aumentando su confiabilidad.

Riesgos de Cloud Computing

Adoptar una estrategia de Cloud Computing es potencialmente nocivo en algunos casos particulares. Para entender éstos y comprobar que no aplican a nuestro caso de estudio, analizaremos algunos de los posibles riesgos.

El uso remoto de infraestructura implica que la información a tratar sea movida a la nube. Esto requiere que el cliente de los servicios confíe en el proveedor, y se adapte a las medidas de seguridad utilizadas por éste. Dado que los recursos son compartidos entre clientes, es posible que un cliente malicioso intente explotar una vulnerabilidad para robar o dañar información. Es por esto que muchas organizaciones gubernamentales prohíben por ley la utilización de recursos públicos y/o compartidos para el manejo de información sensible.

Existen distintas leyes y reglamentaciones en distintos países, los clientes deben conocer dónde se encuentran los recursos utilizados geográficamente (lo cual

en algunos casos es incierto, ya que por redistribución de carga podrían utilizarse recursos de distintas partes del mundo) para prevenir problemas legales. En el Acta de Protección de Datos del Reino Unido (vigente desde 1998) se establece que no se permite que información personal de ciudadanos británicos sea movida fuera del Espacio Económico Europeo (EEE), lo cual podría ser un problema serio para organizaciones que manejen este tipo de información y quieran utilizar recursos localizados en otro lugar.

Al no ser dueño de la infraestructura, el cliente pierde cierto control sobre ésta, es decir que debe limitarse a cómo opera el proveedor y a las limitaciones establecidas por éste.

En el caso de que una organización decida cambiar de proveedor de servicios, podría ser posible que su solución ya implementada sea poco portable. Los distintos proveedores de servicios de cloud siguen su propio estándar.

2.3 MapReduce

MapReduce es un paradigma de programación y, a su vez, una implementación de dicho paradigma para el procesamiento y generación de grandes data sets. Su metodología consiste en definir una función *map* que procese pares clave/valor para generar un set intermedio compuesto de otros pares de claves y valores, y una función *reduce* que una todos esos valores intermedios con sus respectivas claves intermedias. Los programas expresados con este paradigma son automáticamente paralelizables y ejecutables en clusters. Los datos de entrada son particionados y el programa se ejecuta de forma distribuida.

La intención de esta implementación es la de expresar procesamiento de manera simple, escondiendo detalles engorrosos de paralelización, tolerancia a fallos, distribución de datos y balance de carga. Fue inspirada en las primitivas de *map* y *reduce* de Lisp y otros lenguajes funcionales. Estos simplifican la implementación de algoritmos paralelos dado a la falta de estado global, que impide la presencia de posibles condiciones de carrera (*race conditions*, en inglés) a causa de accesos de manera no prevista a memoria compartida.

Visto como una caja negra, el proceso toma un set de pares clave/valor como entrada, y produce otro set de pares clave/valor como salida. El usuario expresa el proceso como dos funciones: *Map* y *Reduce*. *Map* debe tomar una entrada de pares y produce una salida intermedia. La librería por sí misma toma estos valores intermedios y los asocia con su clave, para luego enviarlos a la función *reduce. Reduce* toma una clave individual y el conjunto de todos los valores relacionados a esa clave. Estos datos son manejados mediante un iterador, permitiendo que colecciones muy grandes entren en memoria. Luego, los une para formar un conjunto posiblemente más pequeño. En general, la salida de cada llamada a *reduce* es sólo una o ninguna.

Las llamadas a *map* son distribuidas a lo largo de varias máquinas mediante la partición de la entrada de datos de manera automática. Éstos pueden ser procesados en diferentes máquinas en paralelo. Las llamadas a *reduce* son distribuidas mediante la partición de las claves intermedias utilizando una función de particionamiento. En la mayoría de las implementaciones del paradigma

MapReduce el usuario debe especificar tanto la cantidad de particiones deseadas como la función de particionamiento. Veamos el proceso:

- La librería comienza por particionar los archivos de entrada en partes de, por lo general, 16Mb o 64Mb (modificable mediante parámetros), e inicializa varias instancias del mismo programa en un cluster de máquinas.

- Una de las copias es llamada copia *master*, las demás son llamadas *workers* que trabajan para el master. Este último elige workers ociosos y les asigna una tarea de map o de reduce.

- Un worker al que se le asigna una tarea de map lee el contenido de una partición, mapea pares clave/valor y envía cada uno a la función de map definida por el usuario. Esos pares intermedios son retenidos en un búfer en memoria[5].

- Esporádicamente los pares en el búfer son escritos en el disco local, particionado en la misma cantidad de particiones de datos, con la misma función de partición provista previamente. La dirección de estos pares en dichas particiones de disco son pasadas al master, quien es responsable de enviar estas direcciones a los correspondientes workers de reduce.

- Cuando un worker de reduce es notificado por el master acerca de una de estas direcciones, se utiliza RPC[6] para leer los datos del búfer desde los discos de los workers de map. Una vez que el worker de reduce leyó toda la información, la ordena por clave intermedia para que todas las ocurrencias de la misma clave estén juntas.

- El worker de reduce itera sobre los datos intermedios y envía cada clave única encontrada a la función de reduce del usuario, junto con sus respectivos valores. La salida final de esta función es anexada a un archivo de salida final para su respectiva partición de reduce.

- Cuando todas las tareas son completadas, el master despierta al programa principal, y se retorna al código del usuario.

La fase de *map* se divide en M partes y la de *reduce* en R partes. Idealmente, M y R debieran ser mucho más grandes que el número de workers disponibles para lograr que cada worker ejecute varias tareas. Con esto se mejora el balance de carga dinámico, y se acelera la recuperación ante fallas. Debido a esto, el master realiza

[5]Un búfer es una sección de memoria utilizada para almacenar datos temporalmente.

[6]Remote Procedure Call (RPC) es un protocolo de comunicación mediante 2 nodos para que uno ejecute código en el otro remotamente.

O(M + R) [7] decisiones de scheduling y mantiene O(M * R) estados en memoria (cuya constante tiende a ser pequeña, 1 byte por par). En general, R es limitado por los usuarios ya que la salida de cada *reduce* se escribe en un archivo distinto. En la práctica, suele seleccionarse a M con un valor para que cada tarea individual maneje entre 16Mb y 64Mb de entrada (Para la optimización del almacenamiento y bandwith, mediante el principio de localidad de memoria), y se hace que R sea un múltiplo pequeño del número de workers que se espera usar.

La salida queda disponible en los archivos de salida (uno por tarea de reduce). Por lo general el usuario no necesita combinar estos archivos de salida en uno solo. Es posible pasar estos archivos como entrada de otra llamada a MapReduce, o utilizarlos desde otra aplicación distribuida que puede lidiar con varios archivos en conjunto.

[7] Utilizaremos la notación "Big O" la cual describe el límite de una función cuando la función tiende a algún valor o a infinito.

3 Problemática y recursos

En las siguientes secciones explicaremos la problemática a resolver, primero centrándonos en el problema desde un punto de vista neológico, y luego desde el punto del análisis masivo de datos.

Veremos las herramientas que nos ayudarán a resolver el problema, y plantearemos una solución sencilla pero eficaz, considerando los conceptos incorporados en éste capítulo.

Por último, resumiremos lo visto hasta ahora, para dar al lector una perspectiva completa de lo visto hasta ahora.

3.1 Introducción

En este capítulo veremos cómo se presentan varios problemas e interrogantes a la hora de analizar un lenguaje, para ello utilizaremos Google Books n-grams. Hablando sólo de la parte lingüística, puede ser difícil decidir qué definición tomar para algunos términos, y cómo tratarlos. Procesar una gran cantidad de información requiere del uso de varias herramientas, nosotros proponemos el uso de AWS (Amazon Web Services) pero también dejamos abierta la posibilidad de utilizar otras. Como explicamos anteriormente, nos inclinamos por el uso de algún proveedor de servicios para evitar tener que invertir en hardware e infraestructura, pero también sería posible que uno compre el equipo necesario y utilice tecnologías open source (u otras) para su investigación.

Analizaremos las tecnologías de AWS, especialmente EC2 (Elastic Compute Cloud) y EMR (Elastic Map Reduce) que serán las más utilizadas, pero también S3 (Simple Storage Service) para el almacenamiento y IAM (Identity and Access Management) para la autenticación. Veremos los conceptos mínimos necesarios para comprender su funcionamiento, su ciclo de trabajo y cómo se relacionan. Introduciremos algunos conceptos de la SDK, junto con figuras que ejemplificarán el código necesario para realizar algunas de las tareas deseadas, y también cómo utilizarla para crear steps de Hive para procesar la información.

Por último, presentaremos una solución con el uso de las herramientas mencionadas, mediante la utilización de la SDK para lanzar un cluster y de Hive para crear un diccionario a partir del corpus y moverlo por una ventana, actualizándolo y almacenando los resultados en S3 para su posterior reutilización y análisis ya que la información que se encuentra en el cluster en el sistema de archivos HDFS (Hadoop Distributed File System) es volátil.

3.2 Problemática

Estudios relacionados

El estudio de la neología (Apéndice A) se vio modificado en las últimas décadas por la incorporación de la informática en el campo de la lingüística. Diversos autores han desarrollado distintas herramientas de procesamiento de textos para la detección semi-automática y automática de neologismos, así como su categorización. Entre ellos podemos mencionar NeoTrack (Janssen, 2005a), Buscaneo (Cabré et al., 2009),NeoCrawler (Kerremans et al., 2011), CANeo TIP (Estupiñán et al., 2012), NeoTag (Janssen, 2012), Logoscope (Falk et al., 2014), entre otros. La mayoría de ellos se basan en alguno de los siguientes 2 métodos o algún derivado:

- Analizar un corpus junto con un conjunto de palabras conocidas llamado diccionario o lista de exclusión, identificando las palabras nuevas como potenciales neologismos, o;

- Analizar un corpus, por lo general diacrónico, empleando distintos métodos estadísticos y heurísticas para determinar si existen neologismos.

El origen del corpus en los estudios neológicos suelen ser diarios, aunque también pueden ser libros o partes de discursos. Esto depende del tipo de neologismos que se esté intentando detectar. Freixa et al. (2006) analizan las ventajas y dificultades de la extracción manual frente a la automática dependiendo del tipo de neologismo, mencionando neologismos formados por derivación, composición, truncación, neologismos sintácticos, semánticos y préstamos o extranjerismos. Janssen (2005b) compara distintas definiciones de neologismos, en especial la lexicográfica y la basada en el análisis de corpus, explicando problemas en ambas como son la ausencia de una palabra en un diccionario por razones distintas a la de ser un neologismo, y la aparición de palabras nuevas en corpus que no son neologismos.

Existen a grandes rasgos 3 métodos de extracción de neologismos:

- Manual, llevada a cabo por un profesional.

- Semi-automática, la cual utiliza recursos computacionales para facilitar la tarea al profesional, que también es necesario.

- Automática, donde no se necesita un profesional de neología y detecta neologismos por sí sólo.

Muchos autores proclaman desarrollar métodos automáticos que en realidad son semi-automáticos. Es importante remarcar esta diferencia ya que es lo que determina la necesidad de consultar a un experto o no.

Definiciones y tipos de neologismos

Entre los primeros problemas que podemos encontrarnos al enfrentar una investigación relacionada es el de definir qué es un neologismo y qué no lo es. Varios autores han intentado dar una definición robusta, y se han topado con el inconveniente de que la novedad de una palabra podría ser subjetiva, es decir, algunos podrían considerar una palabra como nueva, y otros no.

Janssen (2005b) en su investigación sobre neologismos ortográficos realiza un profundo análisis acerca de las diversas definiciones y sus falencias. Él nos señala cómo la búsqueda de una definición ya existía en 1995 cuando Alain Rey concluye que no existe definición objetiva para el término *neologismo* debido a que depende del punto de vista de cada individuo sobre qué es novedoso y qué no, y luego Cabré en 1999 define los *neologismos psicológicos*, es decir, una palabra que es percibida como nueva por la comunidad. Janssen continúa con su investigación con los *neologismos léxicos*, aquellas palabras que no aparecen en un diccionario, que conllevan como problema que los diccionarios podrían no ser completos y existen varias razones por las que una palabra podría no aparecer en él, y considera a los errores de tipeo y nombres propios como neologismos cuando en realidad no lo son. Existen también los *neologismos diacrónicos*, que son aquellas palabras que aparecen en un texto general reciente pero no en uno anterior de ese mismo lenguaje. Janssen remarca los problemas de éste último enfoque en que algunas palabras que no son neologismos podrían ser detectadas como tales, debido a que siempre van a existir errores de tipeo y nombres propios en cualquier corpus, y por más que éstos se filtren no existe corpus lo suficientemente extenso que incluya todas las palabras de un idioma. Esto puede ser discutido con un contraejemplo, es decir, un corpus suficientemente extenso, aunque esto es difícil sino imposible de demostrar.

En nuestro caso de estudio utilizamos los Google ngrams, el cual es exhaustivamente completo. Luego de una ardua búsqueda sobre el idioma inglés, encontramos que el corpus no posee la palabra “lamprophonia” (claridad de la voz) pero esto no quiere decir que el corpus no sea completo, sino que quizás nos hemos topado con un neologismo o con una palabra en desuso.

Respecto a los problemas tipográficos y nombres propios, no es necesario que sean filtrados, si el corpus es lo suficientemente grande, incluirá todos los errores tipográficos comunes y nombres posibles como palabras del idioma, y sólo restará el análisis estadístico de sus ocurrencias para detectar si se convierten en un neologismo en algún momento dado. Dado que el corpus en nuestro caso de estudio representa, como veremos más adelante, el 6%[8] de los libros alguna vez publicados abarcando distintas proveniencias y tópicos en un rango de más de 4 siglos, podríamos preguntarnos si realmente vale la pena detenernos en una palabra cuya cualidad principal sea su poco uso.

Es importante recordar por qué el estudio de la neología es interesante. Los

[8]La segunda versión representa el 8% de los libros alguna vez publicados, pero nosotros utilizaremos la primera.

neologismos mantienen vivo a un lenguaje y marcan hacia dónde éste se dirige. Mediante la observación de ellos y de los cambios en el habla es posible que academias del lenguaje como la Real Academia Española (RAE) en España y regiones hispanoparlantes o el "Rat für deutsche Rechtschreibung" (RdR) en regiones germanoparlantes, intervengan si detectan que se están adaptando malas prácticas.

Janssen prosigue con su investigación considerando el uso de internet como corpus de exclusión y menciona algunos de los posibles problemas. Entre ellos nos encontramos con que existen muchas combinaciones de letras que no son palabras y aparecen reiteradas veces, por ende perderíamos muchos neologismos si utilizáramos internet como corpus de manera directa. No obstante esto podría ser resuelto aplicando heurísticas y análisis estadísticos, por ejemplo considerar palabras que ocurran en una ventana de tiempo, e intentar predecir sus ocurrencias. Analizaremos estas posibilidades en los capítulos siguientes.

Por último Janssen nos introduce la definición de neologismo diacrónico extendida, la cual considera como neologismo a cualquier palabra que no aparece en una base de datos morfológica derivada de un diccionario, debido a ser novedosa. Entendemos entonces que un diccionario puede ser creado a partir de un corpus, y podemos utilizar la diacronicidad del corpus para detectar la novedad de una palabra o un giro en su uso en un momento determinado.

Como acabamos de ver, existe una gran discrepancia a la hora de definir cómo detectar un neologismo, qué es considerado un neologismo y qué no, y cuales métodos son válidos o no. Poco se ha debatido acerca de la eficiencia de éstos métodos, o cómo llevarlos a cabo.

Camino hacia una solución

A la hora de construir una herramienta para la detección de neologismos podemos detectar varios problemas. Elegir una definición para neologismo no es trivial, pero primero el investigador debe conocer qué fuentes de datos posee a disposición (diarios, revistas, libros, bibliotecas virtuales, blogs, internet) y cómo quiere tratarlos. Para ello, debe definir también si enfrentará el problema que desea resolver con alguno de los dos métodos más comunes que son mediante listas de exclusión y mediante el análisis de un corpus diacrónico, o algún otro método novedoso como podría ser un híbrido, lo cual depende tanto del problema como de los recursos disponibles. Cual fuere su decisión, es necesario que se mantenga en lo abstracto y analice la metodología a seguir para encontrar una solución a su problema sin limitarse por lo que le ofrecen las herramientas que conoce. Esto se debe a que con el avance de la computación en las últimas décadas la capacidad de procesamiento aumentó desmesuradamente, así también como la de almacenamiento, y nada indica que dejará de hacerlo en el futuro. Sabiendo cual es nuestra preferencia, podemos ahora definir un concepto de neologismo que se adapte a nuestros recursos, y en lugar de apegarnos a una definición preestablecida, crear la más conveniente pero también precisa definición para nuestro corpus. Por último, deberá analizar la eficiencia de su solución, y si es conveniente escalarla para analizar más recursos, o los mismos pero en menos tiempo.

En nuestra investigación tenemos disponible un gran corpus diacrónico en varios idiomas. Partiendo de esta base, puede ser muy difícil conseguir un diccionario completo para cada idioma que se desee procesar. Por ende, podríamos optar por un enfoque híbrido en el cual generemos el diccionario para cada rango de años y analizar el corpus en una ventana de tiempo. Esa podría ser nuestra metodología, en la cual analizaremos el corpus en una ventana de tiempo. El siguiente inconveniente sería el de decidir qué será considerado un neologismo, y qué no. Analizaremos algunas posibles alternativas de qué considerar un neologismo:

- Palabras que no están en nuestro diccionario generado desde el corpus, pero comienzan a utilizarse frecuentemente al mover nuestra ventana.

- Palabras que quedaron al límite de entrar en nuestro diccionario pero no lo hicieron, esto dependerá del rango de años que elijamos y las ocurrencias que creamos necesarias para considerar una palabra parte del diccionario (lo cual podría ser un porcentaje sobre el total o que aparezca al menos una vez en todos los años, entre muchas otras heurísticas)

Con este enfoque nos proponemos demostrar que existen muchas posibilidades válidas a la hora de llevar una investigación. Vemos cómo nos mantuvimos en la abstracción, y no nos limitamos por el entorno. Comprendemos ahora que este análisis podría ser muy grande, lo cual requiere varios tipos de herramientas para el procesamiento, sin mencionar la complejidad del algoritmo. Es ahora cuando comenzamos a analizar la eficiencia necesaria y la escalabilidad del problema.

3.3 Google n-grams

Los n-grams o *n-gramas* son tuplas de ítems de tamaño fijo. En el caso de los *Google n-grams* los ítems son palabras que aparecen dentro de una oración de manera consecutiva, extraídas del corpus Google Books. La *n* especifica la cantidad de palabras de cada tupla, por ejemplo un 5-gram posee 5 palabras[9].En este párrafo, un posible 4-gram es "*son tuplas de ítems*".

Los n-grams fueron generados al pasar una ventana por el texto de distintos libros digitales y generar un registro de su contenido. Aproximadamente más de 8 millones de libros fueron utilizados para esta tarea (Yuri Lin et al., 2012), los cuales representan aproximadamente el 6% de todos los libros publicados del mundo. Los idiomas que se procesaron y su cantidad de filas pueden ser observados en la tabla 3.3.1. Dichos volúmenes datan desde el siglo XVI hasta el año 2008.

El corpus no podría ser leído por seres humanos debido a su tamaño. Suponiendo que se tratara de leer sólo los textos en Inglés del año 2000 en adelante, a una velocidad de 200 palabras por minuto, tomaría 80 años de lectura (Michel et al., 2010).

Dado el formato de los n-grams, que veremos más adelante, son muy útiles a la hora de análisis lingüísticos, permiten observar el cambio de los temas de interés a lo largo del tiempo, así como también los cambios lexicográficos y de gramática.

El estudio realizado por Jean-Baptiste Michel et al. (2010) realiza un análisis acerca de la cantidad de palabras en el idioma Inglés. Para esto se analizaron los 1-grams en Inglés, estimando que una palabra se considera de uso común cuando ésta aparece al menos una vez cada mil millones de palabras, esto se debe a que palabras comunes extraídas de diccionarios reconocidos suelen tener esa frecuencia. Luego de ciertos análisis, se derivó al resultado de que el idioma Inglés poseía aproximadamente 1.022.000 palabras en el año 2000, y crece a una gran velocidad: aproximadamente 8.500 palabras por año, un aumento del tamaño del lenguaje de un 70% en los últimos 50 años.

Los n-grams representan un corpus extenso, el cual presenta 2 versiones. La primera generada en el año 2009 es la número 20090715, y una extensión, la segunda versión número 20120701 generada en el año 2012 que agrega gramas, anotaciones sintácticas como parte de discurso, y el idioma Italiano. Ambas versiones cubren el mismo rango de años (desde el año 1500 aproximadamente hasta el 2008 inclusive) con la diferencia de que la segunda es más completa. Veremos en el capítulo 4 que nosotros nos centraremos en la primera versión, en los 1-grams (que promedian los 3GB de tamaño entre los distintos idiomas) y que el tamaño total del corpus ronda los 2.2TB.

[9]Algunos lenguajes no poseen el concepto de palabras, sino de ideogramas, que son representados con sólo un carácter. Entenderemos estos casos como palabras de una letra.

Idioma	Ngrams	Filas	Tamaño
Inglés	1	472.764.897	4,8GB
	2	6.626.604.215	65,6GB
	3	23.260.642.968	218,1GB
	4	32.262.967.656	293,5GB
	5	24.492.478.978	221,5GB
Francés	1	157.551.172	1,6GB
	2	1.501.278.596	14,3GB
	3	4.124.079.420	37,3GB
	4	4.659.423.581	41,2GB
	5	3.251.347.768	28,8GB
Chino	1	7.741.178	0,1GB
	2	209.624.705	2,2GB
	3	701.822.863	7,2GB
	4	672.801.944	6,8GB
	5	325.089.783	3,4GB
Alemán	1	243.571.225	2,5GB
	2	1.939.436.935	18,3GB
	3	3.417.271.319	30,9GB
	4	2.488.516.783	21,9GB
	5	1.015.287.248	8,9GB
Hebreo	1	44.400.490	0,5GB
	2	252.069.581	2,4GB
	3	163.471.963	1,5GB
	4	43.778.747	0,4GB
	5	11.088.380	0,1GB
Ruso	1	238.494.121	2,5GB
	2	2.030.955.601	20,2GB
	3	2.707.065.011	25,8GB
	4	1.716.983.092	16,1GB
	5	800.258.450	7,6GB
Español	1	164.009.433	1,7GB
	2	1.580.350.088	15,2GB
	3	3.836.748.867	35,3GB
	4	3.731.672.912	33,6GB
	5	2.013.934.820	18,1GB

Tabla 3.3.1: Tamaño del corpus de Google Books por idioma, primera versión.

3.4 Amazon Web Services

Los Servicios Web de Amazon (AWS por sus siglas en inglés) son un conjunto de servicios de infraestructura de procesamiento en la nube que la compañía Amazon empezó a ofrecer en el año 2006. Se encuentra disponible en más de 190 países, contando con datacenters principalmente en Estados Unidos pero también en Europa, Brasil, Singapur, Japón y Australia.

No solo es utilizado por pequeñas organizaciones que desean bajar los costos de inversión en infraestructura, sino también por grandes compañías como son Netflix, Samsung, Adobe, Pinterest, Nokia, Spotify, y el mismo Amazon[10].

[10]https://aws.amazon.com/solutions/case-studies/all/ (Visitado: Enero 2016)

Algunos de sus competidores directos son Microsoft Azure[11] y Google Cloud Platform[12]. No nos detendremos a comparar las opciones de proveedores de servicios ya que extiende el alcance de esta investigación. La razón por la cual optamos por AWS es que es un servicio con una gran comunidad, ampliamente usado y que cuenta con la primera versión del corpus de ngrams en un repositorio público[13]:

```
s3://datasets.elasticmapreduce/ngrams/books/20090715/
```

Amazon Elastic Compute Cloud (EC2)

La Nube de Cómputo Elástica (abreviado EC2 por ser ECC sus siglas en inglés) es un servicio web que provee capacidad de cómputo en la nube de tamaño variable y está diseñado para facilitar el cómputo en la nube a desarrolladores.

Su principal ventaja es la sencillez con la que es posible configurar un servidor y ponerlo en funcionamiento, permitiendo una gran escalabilidad en ambos sentidos. Esto lo logra mediante el uso de AMIs (Amazon Machine Image), lo cual es un archivo de configuración que especifica lo necesario para lanzar una instancia de Amazon, que es un servidor virtual en la nube. Amazon provee algunas AMIs públicas y gratuitas, entre ellas podemos mencionar Amazon Linux, Red Hat Enterprise Linux 7.2, SUSE Linux Enterprise Server 12, Ubuntu Server 14.04, Microsoft Windows Server 2012, entre otras. Existe la opción de comprar ciertos tipos de instancia que no son gratuitos, y también la de crear una propia. En nuestro caso de estudio utilizaremos la primera de éstas, Amazon Linux AMI. Ésta AMI posee las ventajas de ser gratuita, mantenida por AWS, y diseñada especialmente para funcionar sobre EC2 de manera estable, segura y con alto rendimiento.

Las instancias de EC2 utilizan varias medidas de seguridad para evitar que el control de alguna instancia caiga en manos equivocadas, entre ellas están los pares de claves (pública y privada), grupos de seguridad (reglas de acceso para un conjunto de instancias) y *Virtual Private Cloud* (VPC, red virtual que permite aislar recursos y controlar tráfico, entre otros beneficios).

Nos detendremos a analizar los pares de claves, ya que son necesarios para conectarse al nodo master. EC2 utiliza criptografía asimétrica para manejar información de sesión, es decir que maneja un par de claves para encriptar (clave pública) y desencriptar (clave privada) la información. Existen varias maneras de crear éste par de claves que EC2 necesita, que son del tipo 2048-bit SSH-2 RSA[14], una sencilla es a través de la consola en la sección de EC2 en la sección “Key Pairs”, pero existe una manera más directa que veremos en el capítulo 4.

Existen diferentes tipos de instancias de EC2 diseñadas para optimizar distintos casos de uso. Estos tipos de instancia combinan distintos tipos de CPU,

[11]https://azure.microsoft.com/(Visitado: Enero 2016)

[12]https://cloud.google.com/(Visitado: Enero 2016)

[13]https://aws.amazon.com/datasets/google-books-ngrams/(Visitado: Enero 2016)

[14]RSA es un sistema de encriptación asimétrica ampliamente utilizado surgido en 1977.

memoria, almacenamiento y capacidad de manejo de redes. Podemos elegir entre distintos tamaños (nano, micro, small, medium, large, xlarge, 2xlarge, etc) los cuales varían en el número de vCPU[15], capacidad de almacenamiento y de manejo de red, pero mantienen el mismo tipo de hardware y características generales. Veremos cuáles son:

- Propósito General
  - T2: Proveen una línea base de procesamiento que tiene la capacidad de aumentar rápidamente si es necesario. Son recomendadas para los casos en los que no se utiliza la capacidad total de la CPU habitualmente pero necesita procesar ráfagas intermitentemente, como es el caso de servidores web y proveedores de servicios pequeños. A nivel de hardware constan con procesadores Intel Xeon de hasta 3.3Ghz de capacidad, 1 o 2 vCPUy entre 0.5GBs y 8GBs de memoria dependiendo el tamaño de la instancia.

  - M3: Proveen un balance entre todos los recursos. Son recomendadas para procesamiento en clusters, pequeñas y medianas bases de datos, entre otros. Poseen procesadores Intel Xeon E5-2670 V2 y almacenamiento en unidades de estado sólido. Según el tamaño de instancia, pueden tener entre 1 y 8 vCPU, 3.75GBs y 30GBs de memoria.

  - M4: Ofrecen las mismas características generales que las M3, pero con mejor rendimiento. Sus procesadores son Intel Xeon E5-2676 V3 de 2.4GHz, con memoria EBS[16]. Las vCPU varían entre 2 y 40, y la capacidad de almacenamiento entre 8GBs y 160GBs.

- Computación Optimizada
  - C3: Se caracteriza por tener procesadores muy potentes, los cuales son del tipo Intel Xeon E5-2680 V2, y almacenamiento en unidades de estado sólido. Las vCPU van de 2 a 32, y su capacidad de almacenamiento de 3.75GBs a 60GBs.

  - C4: Análogamente a M3 y M4, éste tipo ofrece características similares a C3 pero con mejor rendimiento. La mejor relación precio/rendimiento en EC2 es el de éste tipo. Se recomiendan

[15]Los procesadores virtuales (vCPU) en AWS son núcleos de *hyperthread*, lo cual es una tecnología propietaria de Intel diseñada para mejorar el procesamiento en paralelo de multitareas. Ciertos usuarios denuncian que esto reduce el rendimiento real de los clusters.
http://www.pythian.com/blog/virtual-cpus-with-amazon-web-services/ (Visitado: Enero 2016)

[16]Amazon Elastic Block Store (EBS) es una tecnología de almacenamiento de Amazon creada para utilizarse en la nube de AWS.

para aplicaciones de alto rendimiento, análisis distribuido, servidores web, entre otros. Los procesadores son Intel Xeon E5-2666 V3, con memoria EBS. Las vCPU varían de 2 a 36, y la memoria de 3.75GBs a 60GBs.

- Memoria Optimizada
  - R3: Este tipo esta optimizado para ser eficiente en el manejo de memoria y ofrece la mejor relación precio/almacenamiento. Se recomienda para bases de datos de alto rendimiento, memorias cache distribuidas, análisis sobre grandes cantidades de datos, entre otros. Cuenta con procesadores Intel Xeon E5-2670 V2 y almacenamiento en unidades de estado sólido. Poseen entre 2 y 32 vCPU, y de 15.25GB a 244GB dependiendo su tamaño.

- GPU (Unidad de Procesamiento de Gráficos)
  - G2: Están diseñadas para el procesamiento de gráficos. Se recomienda su uso para aplicaciones de codificación de videos, machine learning y otros que requieran procesamiento de gráficos del lado del servidor. Sus procesadores son Intel Xeon E5-2670, y GPU NVIDIA con 1536 núcleos CUDA[17]y 4GBs de memoria de video cada una. Las opciones de tamaño son sólo 2, 1 vCPU con 8 GPU y 15GBs de memoria, o 4 vCPU con 32 GPU y 60GBs de memoria.

- Almacenamiento Optimizado
  - I2: Estas son instancias de gran rendimiento en entrada y salida de datos. Sus procesadores son Intel Xeon E5-2670 V2, y su almacenamiento se encuentra en unidades de estado sólido. Se recomiendan para aplicaciones de almacenamiento de datos y hadoop, bases de datos no relacionales, entre otras. Dispone de 4 a 32 vCPU, y de 30.5GBs a 244GBs de memoria.

  - D2: Este tipo está ideado para almacenar las más grandes cantidades de datos, soportan hasta 48TBs de almacenamiento en discos rígidos. Sus procesadores son Intel Xeon E5-2676 v3, y sus vCPU varían entre 4 y 36, y su memoria entre 30.5GBs y 244GBs.

En nuestra investigación utilizaremos algunos de estos tipos para comparar el rendimiento y determinar qué tipo de hardware y configuración son las más convenientes. No todos los tipos son soportados en clusters EMR.

[17]Tecnología desarrollada por NVIDIA que da acceso al set de instrucciones virtual de la GPU y a otros elementos de procesamiento paralelo.

Amazon Simple Storage Service (S3)

El Servicio de Almacenamiento Simple (abreviado S3 por ser SSS sus siglas en inglés) es un servicio de almacenamiento para internet, y está diseñado para facilitar el cómputo web escalable.

Provee una interfaz web de servicios que pueden ser usados para almacenar y recuperar información, como vemos en la figura 3.4.1. La información es representada en forma de *objects* de hasta 5TBque se guardan en *buckets*. Sobre los buckets puede elegirse una ubicación geográfica dónde se encontrará su información, lo cual es práctico a la hora de optimizar tiempos de latencia en transferencia de datos, aunque también es importante que el bucket esté almacenado en la misma ubicación donde se planea procesar, ya que algunos servicios cobran extra por transferencia de datos, lo que también hace a nuestras aplicaciones innecesariamente más lentas.

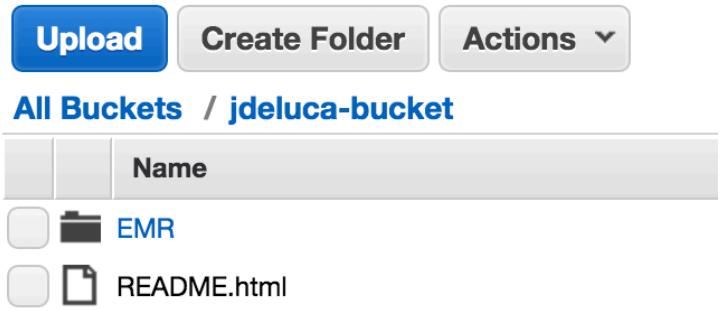

Figura 3.4.1: Ejemplo de bucket (“jdeluca-bucket”) con carpeta (“EMR”) y object(“README.html”) en la Consola web de AWS

```
AmazonS3 s3client =
            new AmazonS3Client(new ProfileCredentialsProvider());
s3client.setRegion(Region.getRegion(Regions.US_EAST_1));

//Los nombres de los buckets deben ser únicos globalmente.
if (!(s3client.doesBucketExist(bucketName))){
      s3client.createBucket(new CreateBucketRequest(bucketName));
}
```

Figura 3.4.2: Código simple de ejemplo para crear un bucket en S3 mediante la SDK de Java en AWS

Amazon Elastic MapReduce (EMR)

El servicio de MapReduce Elástico de Amazon es una plataforma de manejo de clusters diseñada para simplificar la ejecución de frameworks de big data en AWS. Mediante la combinación de éstos con algunas tecnologías open source soportadas por Amazon como lo es Apache Hive[18], podemos procesar grandes cantidades de datos con fines analíticos o de lógica de negocios. Y también soporta la transferencia de datos desde y hacia Amazon S3 y Amazon DynamoDB[19].

El componente central de EMR es un cluster de instancias EC2. Cada una es llamada nodo, los cuales emplean distintos roles llamados tipos de nodo. EMR instala componentes de software distintos dependiendo del rol a desempeñar por el nodo, los cuales pueden ser:

- Nodo master: maneja el cluster al coordinar la distribución de información y tareas en los otros nodos (llamados nodos esclavos) para su procesamiento. Mantiene información del estado del cluster y las tareas.

- Nodo core: un tipo de nodo esclavo que ejecuta tareas y almacena información en el cluster utilizando el sistema de archivos de Hadoop HDFS[20].

- Nodo task: nodo esclavo opcional que sólo ejecuta tareas de procesamiento de información.

[18]https://hive.apache.org/ (Visitado: Enero 2016)

[19]https://aws.amazon.com/documentation/dynamodb/ (Visitado: Enero 2016)

[20]https://hortonworks.com/hadoop/hdfs/(Visitado: Enero 2016)

Para procesar información debemos ejecutar un cluster eligiendo los frameworks y aplicaciones que se necesitan preinstaladas. Luego tenemos 2 opciones: enviar tareas o consultas a las aplicaciones instaladas previamente en el cluster o ejecutando *steps* en él. Los steps son unidades de trabajo que contienen instrucciones para manipular la información a procesar mediante el software preinstalado como mencionamos previamente. En general la información a procesar se almacena en forma de archivos del sistema de archivos elegido para el cluster (S3 o HDFS). La información pasa de un step al siguiente, hasta llegar al último que la almacena en donde sea especificado como puede ser un bucket de S3. Dependiendo de cómo se desarrollen las tareas especificadas el cluster puede pasar por distintos estados, como veremos en la figura 3.4.3.

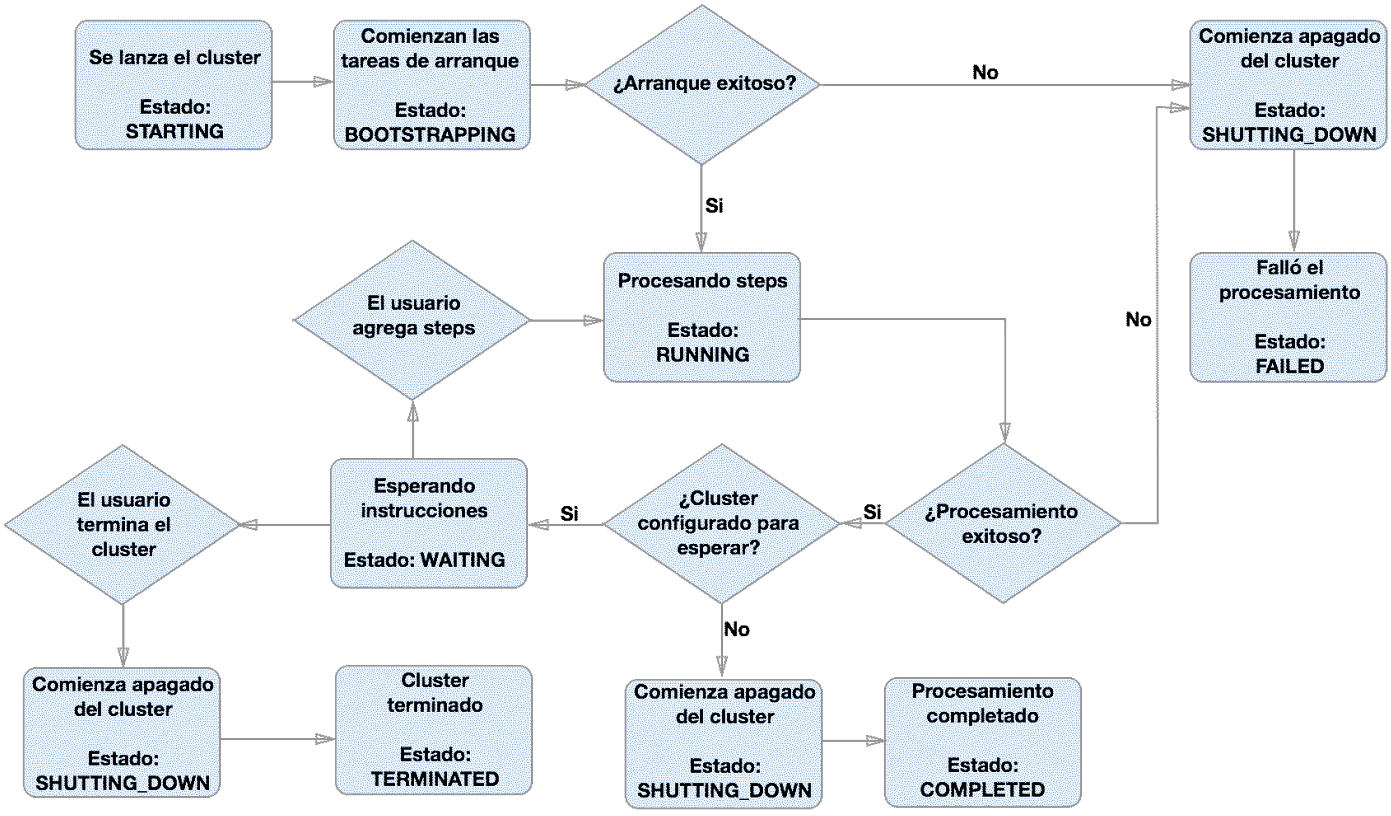

Figura 3.4.3: Ciclo de vida de un cluster EMR.

La SDK de AWS en Java provee un paquete con funcionamiento de EMR el cual se encuentra, junto con otros 2 subpaquetes, en:

```
com.amazonaws.services.elasticmapreduce
```

Como mencionamos previamente los steps son conjuntos de instrucciones que manipulan información. Pueden contener una o más tareas de Hadoop MapReduce. Un cluster puede a su vez contener entre cero y 256 steps activos o pendientes al mismo tiempo, sin embargo es posible mantener un cluster configurado para esperar instrucciones y no terminar cuando no posee steps y enviarle estos mediante la consola de Amazon o la CLI. Esto no es posible utilizando la SDK, aunque sí es posible utilizar un abordaje híbrido en el cual disparemos un cluster configurado para que no finalice luego de correr nuestros steps, por ejemplo de preparación de información.

Veremos un ejemplo de cómo lanzar un cluster con éstas características en la figura 3.4.4. Los steps son procesados de manera secuencial, primero se marcan todos en estado pendiente, y luego uno a uno son tratados cambiando su estado a en ejecución y completado. Si algún step falla, su estado es actualizado a fallado y todos los siguientes steps son marcados como cancelados. Si el cluster sigue o no ejecutando dependerá de la configuración de éste respecto a fallos. Los steps podrían utilizar información relacionada entre ellos, y la manera en la que ésta viaja de un step al siguiente es mediante archivos almacenados en el cluster, el cual utiliza el sistema de archivos HDFS. Esta información sólo permanece disponible cuando el cluster está en ejecución, una vez que se termina o finaliza, la información en HDFS se pierde. Es por esto que es común que la información sea llevada a un bucket de S3 en el último step.

```
credentials =
      new ProfileCredentialsProvider("default").getCredentials();

AmazonElasticMapReduceClient emr =
      new AmazonElasticMapReduceClient(credentials);

// Creamos la lista de steps y agregamos 1 step.
StepFactory stepFactory = new StepFactory();
List<StepConfig> steps = new LinkedList<>();
steps.add(
      new StepConfig()
            .withName("Hive QL Script Master")
            .withActionOnFailure("TERMINATE_JOB_FLOW")
            .withHadoopJarStep(
                  stepFactory.newRunHiveScriptStep(
                        "s3://mi-bucket/ruta/TareHive.q")));

Application hive = new Application();
hive.withName("Hive");

// Configuramos el cluster a lanzar.
RunJobFlowRequest request = new RunJobFlowRequest()
      .withName("Mi Cluster EMR")
      .withReleaseLabel("emr-4.2.0")
      .withApplications(hive)
      .withConfigurations(myHiveConfig)
      .withSteps(steps)
      .withServiceRole("EMR_DefaultRole")
      .withJobFlowRole("EMR_EC2_DefaultRole")
      .withInstances(
            new JobFlowInstancesConfig()
                  .withInstanceCount(5)
                  .withKeepJobFlowAliveWhenNoSteps(false)
                  .withMasterInstanceType("m3.xlarge")
                  .withSlaveInstanceType("m1.large"));

RunJobFlowResult result = emr.runJobFlow(request);
```

Figura 3.4.4: Código para lanzar una instancia de EMR con steps de Hive en la SDK de Java.

Amazon Hive y Apache Hive

Hive es una tecnología open source que funciona como almacenador y manejador de grandes cantidades de datos, permitiendo su análisis. Para ello, debe utilizarse el sistema de archivos de Hadoop para almacenar la información, llamado HDFS, o alguno compatible. Provee un lenguaje del estilo SQL llamado HiveQL (o simplemente QL) que convierte consultas en funciones de *map* y de *reduce*. Hive extiende el paradigma SQL sobre MapReduce permitiendo valores de tipos no primitivos, es decir, elementos estructurados como son los tipos definidos por el usuario, objetos JSON, o funciones escritas en Java.

Amazon Hive no es más que una *rama*[21]del proyecto de Apache Hive, en el cual los desarrolladores de Amazon intentan agregar compatibilidad con AWS, junto con otras mejoras. Podemos mencionar:

- Escribir/Leer sobre S3 directamente: la versión de Hive instalada en los clusters EMR de Amazon posee una extensión que le permite escribir directamente sobre S3, sin la necesidad de utilizar archivos temporales, lo que genera un aumento de performance. Esto genera el inconveniente de que S3 y HDFS sean tratados de maneras distintas dentro de Hive. Como consecuencia, no nos es posible leer y escribir la misma tabla en la misma sentencia de Hive, si ésta se encuentra en S3. Por ejemplo, para actualizar una tabla en S3 deberemos crear una tabla intermedia (que será almacenada en HDFS en el cluster local), volcar los resultados en ella y luego escribir en S3.

- Acceder a recursos de S3: podemos referenciar distintos recursos de S3 mediante la instancia de Hive instalada en los clusters de EMR, como archivos JAR o scripts con funciones de *Map* o *Reduce* propias. Por Ejemplo:

```
add jar s3://elasticmapreduce/samples/hive-
        ads/libs/jsonserde.jar
```

- Usar Hive para recuperar particiones: al lenguaje de consulta HiveQL le agregaron la sentencia *RECOVER PARTITIONS* con el fin de recuperar las particiones de una tabla a partir de la información de una tabla almacenada en S3.

- Uso de variables en Hive: a las instrucciones de Hive podemos agregar el uso de variables mediante el signo de dólar "$" y llaves "{}" de la forma ${var}. Dicha variable puede ser pasada por parámetro a la hora de invocar el script o sentencia con el flag "-d". Esto facilita el uso

[21]Nos referimos al concepto de *branch* de repositorios de código. El repositorio de Hive se encuentra disponible en http://svn.apache.org/viewvc/hive/branches/ (Visitado: Enero 2016)

de Hive ya que facilita su comunicación con la SDK, como veremos en la figura 3.4.5

```
StepFactory stepFactory = new StepFactory();
StepConfig runHive = new StepConfig()
        .withName("Run Hive Script")
        .withActionOnFailure("TERMINATE_JOB_FLOW")
        .withHadoopJarStep(stepFactory.newRunHiveScriptStep(
                "s3://mybucket/script.q",
                Lists.newArrayList("-d", "VAR='neo_table'"))
        );
```

```
    -- script.q
SELECT COUNT(*)
FROM ${VAR};
```

Figura 3.4.5: Pasaje de parámetros con Java SDK, dónde "script.q" posee la variable ${VAR} la cual es pasado por parámetro.

Amazon Identity and Access Management (IAM)

El servicio de IAM es una medida de seguridad obligatoria que se encarga de autenticar y autorizar usuarios a utilizar sus recursos. Cuando creamos una cuenta en AWS nos provee una cuenta del tipo *root*, esto quiere decir que el usuario tiene acceso sin restricciones a la cuenta. Las credenciales de esa cuenta son también del tipo root, y son perfectamente funcionales al igual que las credenciales que mencionamos previamente, no obstante no se recomienda su uso cotidiano por razones de seguridad, es preferible que creemos usuarios dentro de la cuenta, con restricciones, roles, y claves de acceso propias. Una práctica común es la de crear un usuario con permisos administrativos para el dueño de la cuenta, y que utilice ese usuario cotidianamente.

Dentro de una cuenta de AWS podemos crear varios usuarios, que no necesitan ser personas, sino que pueden ser aplicaciones que utilicen recursos en AWS. Por defecto estos usuarios no puede acceder a ningún recurso, éstos permisos deben ser otorgados mediante la creación de políticas de acceso o *policy/policies* las cuales son documentos con formato JSON que listan qué acciones los usuarios pueden realizar y qué tipo de recursos pueden afectar, veremos un ejemplo en la figura 3.4.6. Los usuarios pueden a su vez estar organizados en grupos IAM, y éstos tendrán los mismos permisos. Esto no quiere decir que los permisos individuales pasan a ser compartidos, sino que un grupo IAM puede tener ciertos permisos, los cuales les serán otorgados a todos los usuarios de ese grupo.

Para controlar el acceso de la SDK, AWS utiliza las credenciales IAM. Existen varias maneras de proveer las credenciales, veremos cómo lidiar con ellas en el capítulo 4.

```
{
      "Version": "2012-10-17",
      "Statement": [
            {
                  "Sid": "Stmt1234567890000",
                  "Effect": "Allow",
                  "Action": [
                        "elasticmapreduce:*"
                  ],
                  "Resource": [
                        "*"
                  ]
            }
      ]
}
```

Figura 3.4.6: Ejemplo de *policy* en IAM para otorgar acceso total a EMR.

3.5 Solución propuesta

Mediante AWS podemos procesar la información de manera fácil y eficiente. Sus servicios son compatibles entre sí, y existen múltiples alternativas de uso (consola web, SDK, API). Además, soporta el uso de otras tecnologías open source que ayudan y dan aún más alternativas para plantear una solución.

Luego de analizar los distintos tipos de investigaciones sobre neologismos, y de encontrar muy poca atención por parte de investigadores en la búsqueda y detección de extranjerismos, nos proponemos analizar ambos problemas, proponer una solución con la ayuda de tecnologías de Big Data y analizar el rendimiento que presenta en este caso de uso.

Comenzando por el análisis diacrónico, construiremos un diccionario a partir del corpus. Generaremos varios escenarios en los cuales cambiaremos el método para filtrar las palabras que permanecerán en el diccionario. No buscamos que las palabras en éste sean verdaderas palabras, sólo que no sean neologismos de la época. Con esto nos referimos a que no nos interesa si en el diccionario se encuentra un error tipográfico como palabra, o un nombre propio. De todas formas normalizaremos[22] el corpus para disminuir los niveles de ruido, es decir, los errores.

Definiremos un rango de tiempo o *ventana* en la cual analizaremos las palabras. Como algunas palabras son mucho más frecuentemente usadas que otras, nos resulta difícil encontrar un número de ocurrencias o porcentaje de ocurrencias sobre el total indicado para definir qué palabras estarán o no en el diccionario, por lo que no utilizaremos la información de la cantidad de apariciones de una palabra en un año en éste paso, sino que sólo observaremos si la palabra se utilizó o no (al menos una vez). A su vez, cambiaremos el tamaño de la ventana por ejemplo a 1, 5,

[22]En las bases de datos el concepto de normalización representa el proceso de organizar la información almacenada para eliminar redundancia y entradas no deseadas.

20 y 50 años, y el porcentaje de años que una palabra tiene que aparecer para estar en el diccionario a 100%, 80% y 50%. Luego de generar nuestro diccionario comenzaremos a correr la ventana de a 1 año, analizando qué palabras nuevas aparecen en el diccionario, es decir, la diferencia entre el diccionario generado con una ventana de tamaño *n* partiendo del año *x*, y el diccionario con una ventana del mismo tamaño partiendo del año *x+1*. Veamos un ejemplo: Si nuestra ventana es de tamaño 20, y comenzamos el análisis en el año 1950, generaremos un diccionario con el rango 1950-1970 y otro con el rango 1951-1971, analizaremos qué palabras aparecieron, y cuales desaparecieron. Notemos que algunas palabras podrían también desaparecer (posibles arcaísmos), debemos actualizar el diccionario con las palabras nuevas y quitar las viejas antes de seguir moviendo nuestra ventana.

Almacenaremos una copia de las palabras nuevas que aparecen en cada año, según la configuración de cada método, para comparar los resultados. Dependiendo de qué tan abundante y variado sea el corpus podremos tener varios candidatos a neologismos. Los ordenaremos por ocurrencias de mayor a menor para analizar si se trata realmente de neologismos o no. Sería difícil imponer un límite de palabras a analizar por nosotros mismos, y cuales descartar. Es decir, analizar cierto número de palabras cada año y descartar todas las otras, pero para esto podemos basarnos en la frecuencia de incorporación de términos por parte de academias del lenguaje en los últimos años para dar una idea aproximada de cuantas palabras suelen incorporarse, si bien este número creció a lo largo del tiempo. En el caso del español, la RAE acuñó 93111 términos en la 23ª edición de su diccionario, año 2014, cuando la anterior edición tiene 88431 términos y es 13 años anterior[23]. Con una cuenta sencilla podemos concluir que en promedio se incorporan 360 términos por año. Podemos predecir que algunas palabras aparecerán, dependiendo del método que utilicemos, no exactamente en el diccionario el primer año que se utilizaron. Por ejemplo si elegimos una ventana de 50 años y requerimos un porcentaje de ocurrencia de 80% de los años, una palabra nueva aparece en un año determinado y es ampliamente utilizada será incorporada al diccionario por primera vez 50 * 0.8 = 40 años después de su verdadera primera aparición. Sin embargo, si lo que nos interesa es saber la verdadera fecha de aparición de una palabra en un idioma, podríamos volver al corpus y aplicar distintas heurísticas, como analizar la curva de uso del candidato a neologismo. Es por esto que dependiendo de los resultados esperados no nos servirá cualquier combinación de tamaño de ventana y porcentaje de aparición de años requerida. Si quisiéramos distinguir los neologismos lo más rápido posible, deberemos aceptar tener más ruido, y utilizar una ventana más pequeña.

Una vez generado un diccionario de un lenguaje, podríamos utilizarlo para detectar extranjerismos que provoca en otros, y viceversa. Así como también eliminar el ruido generado por citas en otros idiomas en el texto. Para esto necesitaremos analizar 2 diccionarios de 2 idiomas distintos, en una misma ventana de años. Los términos que tengan en común serán posibles extranjerismos. La manera de detectar que una palabra es originaria de un idioma será tan sencilla como observar qué idioma la acuñó primero en su diccionario. Veremos qué método

[23]http://www.rae.es/sites/default/files/Preambulo.pdf (Visitado: Enero 2016)

nos otorga los mejores resultados.

Todo este procesamiento puede ser realizado mediante las herramientas vistas en éste capítulo. Utilizaremos la SDK de AWS para configurar el entorno de procesamiento, y Hive para expresar nuestras consultas y manejar la información desde y hacia S3. Configuraremos el entorno con distintos tipos de instancia, y distintos tamaños, analizando la eficiencia de los recursos utilizados y ajustando la configuración de cluster a la más apropiada.

4 Implementación y resultados

En este capítulo veremos cómo configurar el entorno de ejecución del algoritmo, explicaremos su funcionamiento, y analizaremos su rendimiento y resultados.

Analizaremos paso a paso la llamada a scripts de Hive, explicando su lógica de trabajo en el orden que es utilizada. Ejecutaremos experimentos destinados a analizar el rendimiento del algoritmo dependiendo del tamaño del cluster y los tipos de instancia.

Por último, analizaremos los resultados y los problemas encontrados.

4.1 Configuración

Command Line Interface (CLI) y Software Development Kit (SDK)

Como mencionamos en capítulos anteriores, existe una herramienta muy útil llamada AWS CLI, la cual es una interfaz de comandos que sirve para interactuar con los servicios de Amazon. Para utilizarla, necesitaremos tener una cuenta en Amazon Web Services la cual puede ser creada a través de su sitio oficial[24]. Luego deberemos dirigirnos a la consola de IAM y crear un par de claves de acceso. Una vez obtenidas, podremos proceder a instalar la CLI desde su repositorio en Github[25]. La manera más sencilla es mediante el manejador de paquetes llamado *pip*:

```
$ sudo pip install awscli
```

Una vez instalada, podremos utilizar comandos para manejar servicios de AWS. Pero primero deberemos proveer las claves de acceso obtenidas al crear la cuenta de AWS.

Una manera práctica de manejar nuestras claves de acceso es a través del archivo *credentials*. Éste es un archivo con formato de estilo INI[26], el cual debe ser ubicado en *~/.aws/credentials* que es donde la SDK lo busca por defecto. Vemos un ejemplo en la figura 4.2.1 donde tenemos un perfil "default" que es el cual AWS utilizará si no especificamos ningún rol, y otro perfil "miPerfilAlt" que se utilizará si lo especificamos. Las credenciales también pueden ser seteadas en variables de entorno, o ser explicitadas directamente en el código. Éste último método podría ser peligroso, ya que por error podríamos subir nuestro código a un repositorio donde otras personas podrían obtenerlas y utilizarlas para fines personales como por ejemplo minería de bitcoins[27].

---

[24]http://aws.amazon.com (Visitado: Enero 2016)

[25]https://github.com/aws/aws-cli (Visitado: Enero 2016)

[26]INI es un estándar informal para archivos de configuración simples, de texto plano.

[27]http://www.forbes.com/sites/runasandvik/2014/01/14/attackers-scrape-github-for-cloud-service-credentials-hijack-account-to-mine-virtual-currency/ (Visitado: Enero 2016)

```
; Soy un comentario.
; Archivo credentials.
[default]
aws_access_key_id =ID_DE_CLAVE_DE_ACCESO
aws_secret_access_key =CLAVE_SECRETA_DE_ACCESO

[miPerfilAlt]
aws_access_key_id =OTRO_ID_DE_CLAVE_DE_ACCESO
aws_secret_access_key =OTRA_CLAVE_SECRETA_DE_ACCESO
```

Figura 4.1.1: Archivo *credentials* para el manejo de credenciales AWS.

Otras de las herramientas que necesitaremos instalar es la SDK de AWS[28]. Ésta provee una API para construir aplicaciones, como la desarrollada en esta investigación. Utilizaremos la versión en Java, la cual requiere Java 1.6 o posterior. Si utilizamos Maven[29], solo es necesario que descarguemos el código desde Github, lo descomprimamos, y ejecutemos el comando estándar de Maven para instalar paquetes:

```
$ mvn clean install
```

A pesar de ser herramientas suficientes, utilizaremos otros recursos en combinación con la CLI y la SDK, como la consola web y la posibilidad de conectarnos al nodo maestro del cluster mediante SSH. Al ejecutar nuestra aplicación lanzando un cluster, nos retornará el ID necesario para que nos conectemos a éste mediante el siguiente comando:

```
aws emr ssh --cluster-id j-XXXXXXXXXX--key-pair-file ~/mi-
                        clave.pem
```

Donde "j-XXXXXXXXXX" es el ID del cluster, y "mi-clave.pem" es el archivo con la clave privada de EC2. Dicha clave debe estar en la misma región que el cluster, en nuestro caso utilizaremos la región "us-east-1", la cual puede ser configurada por defecto en el archivo *~/.credentials* mediante la propiedad "region", dentro de un perfil. Podemos entonces ejecutar el siguiente comando:

```
aws ec2 create-key-pair --key-name miClaveEC2 --query
  'KeyMaterial' --output text > miClaveEC2.pem
```

En la siguiente sección comenzaremos a analizar la configuración del cluster en detalle, mediante el código desarrollado.

[28]https://github.com/aws/aws-sdk-java (Visitado: Enero 2016)

[29]https://maven.apache.org/ (Visitado: Enero 2016)

Configuración de EMR

Existen varias combinaciones posibles para lanzar un cluster EMR, entre ellas podemos destacar la diferencia de tamaño (cantidad de instancias), el tipo de instancia para el nodo maestro y para el nodo esclavo. Existen otros parámetros que no nos detendremos a analizar, pero si mencionaremos por ser necesarios, como son los roles para el servicio de EMR y para las instancias de EC2. En estos casos, necesitaremos generar los roles por defecto mediante el siguiente comando:

```
aws emr create-default-roles
```

El cual creará el archivo *~/.aws/config*con los roles "EMR_DefaultRole" para el servicio y "EMR_EC2_DefaultRole" para la instancia a lanzar.

El tamaño de la instancia determinará el poder de procesamiento y capacidad de almacenamiento del cluster. Éste estará compuesto por 1 nodo maestro el cual no requiere demasiado poder de procesamiento, y el resto de los nodos serán del tipo core, los cuales necesitan poder de procesamiento y almacenamiento, o task, los cuales sólo necesitan poder de procesamiento. El límite por defecto para una cuenta común de AWS es de 20 nodos, por lo que no superaremos este número. Veremos en adelante cómo las diferentes combinaciones ofrecen un rendimiento distinto, así como un costo distinto.

Configuración de Hive

Hive posee distintas propiedades que pueden ser configuradas. La manera que nosotros elegimos es mediante la SDK, pero también puede ser configurado mediante la línea de comandos. Cual fuere el método elegido, estas propiedades se encontrarán en el archivo *hive-site.xml* el cual se encuentra en el nodo master en el directorio de instalación de Hive, el cual es */etc/hive/*.

Nos centraremos en 2 propiedades que utilizaremos en nuestro algoritmo, las cuales son *hive.input.format* y *mapred.min.split.size*. Éstas propiedades deben ser configuradas ya que la información de los ngrams se encuentra en un solo archivo llamado *data*, lo cual provocaría que Hive produzca sólo 1 función de mapeo, ralentizando considerablemente la ejecución. Es por eso que configuramos éstas 2 propiedades con los valores org.apache.hadoop.hive.ql.io.HiveInputFormat y el equivalente en bytes a 128Mb, respectivamente, lo cual le indica a Hive que deberá dividir este archivo en partes de 128Mb para procesarlos con varios mapeadores.

4.2 Implementación

En la presente sección explicaremos el algoritmo desarrollado para solucionar la problemática planteada. Nuestro algoritmo presenta 2 grandes perfiles, por un lado utiliza Java para obtener los parámetros necesarios para lanzar distintas instancias de EMR, y por otro utiliza Hive para manejar la información en S3 y HDFS. La combinación de estas tecnologías es lo que hace sencillo el desarrollo de una solución. Es importante remarcar que el procesamiento y la información

permanecen en el cluster, no necesitaremos ningún hardware fuera de lo común para realizar estas pruebas.

SDK para AWS

Utilizaremos la SDK en Java de AWS para interactuar con nuestro cluster. Nos aprovecharemos principalmente de 2 clases, *AmazonElasticMapReduceClient* y *AmazonS3Client* los cuales son 2 clientes con funciones para interactuar con EMR y S3. Estos clientes realizan consultas a los servicios de AWS autenticándose mediante las credenciales de acceso que vimos anteriormente.

Nuestro algoritmo cargará los scripts que utilizaremos posteriormente en S3, creando un bucket para almacenarlos junto con los resultados de nuestro procesamiento.

Utilizaremos el cliente de S3 para listar los lenguajes disponibles mediante el análisis de los directorios en el bucket de EMR donde se encuentran los ngrams, como veremos en la figura 4.2.1, y le permitiremos al usuario seleccionar los lenguajes a analizar, que en primera instancia serán solo 2: uno para buscar neologismos y otro para analizar su influencia sobre el primero (extranjerismos).

```
        ObjectListing ol = s3.listObjects(
              new ListObjectsRequest()
                    .withBucketName("datasets.elasticmapreduce")
                    .withPrefix("ngrams/books/20090715/")
                    .withDelimiter("/"));

          List<String> languagePaths = ol.getCommonPrefixes();
          Iterator<String> it = languagePaths.iterator();

for (int i = 0; i < languagePaths.size(); i++){
              System.out.println(it.next());
          }
```

Figura 4.2.1: Código Java para listar los lenguajes disponibles de ngrams, mediante AWS.

Utilizaremos un enfoque parecido para permitirle al usuario elegir los tipos de instancia para el maestro y el esclavo, en el cual listaremos los tipos posibles.

En la subsiguiente sección comenzaremos a explicar cómo es el tratado de la información, el cual se hace principalmente mediante steps de Hive. El tipo de instancia que lanzaremos tendrá instalada la aplicación Hive por defecto, por lo que no es necesario agregar steps de configuración, ya que las propiedades básicas son configuradas desde el código. No obstante, éstas pueden ser re-configuradas desde un script de Hive, como veremos en la figura 4.2.2.

```
set hive.base.inputformat=
            org.apache.hadoop.hive.ql.io.HiveInputFormat;
set mapred.min.split.size=134217728;
```

Figura 4.2.2: Script de Hive para configurar variables desde un step.

Steps de Hive

El procesamiento principal y tratado de la información se realiza mediante steps, que son scripts de Hive. Cada step puede estar compuesto por 1 o más instrucciones o consultas de HiveQL. Como vimos anteriormente, es posible crear scripts de Hive con variables del estilo *${miVar}* en ellos, lo que nos permite ejecutar diferentes pruebas.

Como vimos en capítulos anteriores los steps son procesados uno a uno, en el orden en el que fueron enlistados. Mediante nuestro algoritmo agregamos distintos scripts a modo de plantilla, modificando los parámetros o variables deseados para por ejemplo crear una tabla externa en S3 con distintos lenguajes de ngrams, como veremos en la figura 4.2.3 (ImportNgrams.q).

```
CREATE EXTERNAL TABLE raw_${ngramsTable} (
gram string,
year int,
occurrences bigint,
pages bigint,
books bigint
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS SEQUENCEFILE
LOCATION '${ngramsLocation}'
;
```

Figura 4.2.3: Instrucción de HiveQL con variables en su nombre y ubicación de información de entrada (ImportNgrams.q).

En la figura anterior vemos cómo crear una tabla externa, es decir fuera de HDFS, en S3. La ubicación de los ngrams en S3 depende del lenguaje y del ngram, por ejemplo los 3-grams del idioma Alemán se encuentran en *s3://datasets.elasticmapreduce/ngrams/books/20090715/ger-all/3gram*.

Los ngrams poseen todo tipo de gramas, debemos considerar que existen

citas a otros idiomas, errores tipográficos, y también errores de OCR [30]. Analizaremos el ruido producido por estos errores en la sección de resultados. Por ahora nos conformaremos con aplicar una expresión regular sencilla, la cual no es más que una secuencia de caracteres que demuestra un patrón a seguir, para filtrar los gramas deseados. Dicha expresión tendrá la siguiente forma:

```
^\p{Ll}+(\-)?\p{Ll}+$
```

Explicaremos brevemente su funcionamiento. El primer carácter "*^*" indica el comienzo de la cadena o secuencia de caracteres. Luego la barra invertida "\" escapa lo que viene inmediatamente después, que es un script de Unicode[31], el cual tiene la forma "*\p{Ll}*" y se encarga de filtrar cualquier letra minúscula de cualquier idioma para la cual exista una variación en mayúscula. Luego se encuentra el símbolo "*+*" que a pesar de representar la concatenación en muchos lenguajes, aquí representa un cuantificador que limita la expresión anterior a suceder al menos 1 vez. Si nos interesara limitar la expresión a cualquier cantidad de apariciones (inclusive 0) deberíamos utilizar el símbolo "***". Seguido al primer "*+*" vemos una expresión entre paréntesis, que indica que nos encontramos con una sub-expresión, seguida del símbolo "*?*" que indica que dicha expresión sucede a lo sumo 1 vez. Como vimos recientemente la barra invertida sirve para escapar expresiones o caracteres, en éste caso escapa el guión "-" para buscarlo literalmente, esto se debe a que el guión es utilizado para expresiones de rango como es "0-5" la cual filtra los caracteres del 0 al 5. Utilizamos ésta opción ya que existen expresiones con guiones medios que podrían ser de interés, como por ejemplo nosotros utilizamos el término "sub-expresión" en éste párrafo. Repetimos el script de Unicode visto con al menos una ocurrencia para evitar las expresiones terminadas en "-", y por último el carácter "$" indica el final de la cadena.

Como ejemplo la expresión aceptara términos como "hola", "χαίρετε" (forma griega de presentarse) y "aide-moi" (pedir ayuda en Francés) pero no otras como "María" (por comenzar con mayúsculas) o "#error" (por contener un carácter fuera del alfabeto).

La expresión que acabamos de ver no es la mejor para filtrar un corpus de este tamaño, permite demasiadas expresiones erróneas de distintos alfabetos. Podríamos utilizar los scripts de Unicode para filtrar sólo los gramas que utilizan letras latinas como es "*\p{Latin}*", o que mezclan alfabetos. Además, descartamos todos los gramas que contengan al menos una letra mayúscula, lo cual descarta muchos nombres propios que comienzan con mayúsculas, pero quizás otros gramas que podrían ser de interés. Como lado positivo la expresión anterior es sencilla de entender, y deja abierta la posibilidad al lector de pensar sus propias expresiones regulares de filtrado para su corpus o investigación, utilizando los conceptos introducidos en ella. Además, al no filtrar caracteres de otros alfabetos podremos

[30] OCR es una tecnología para el reconocimiento óptico de caracteres, el cual sirve principalmente para la digitalización de textos.

[31] Los scripts de Unicode se encargan de identificar la categoría de un carácter Unicode, como pueden ser letras, números, símbolos, entre otros.

observar mejor palabras extranjeras y errores en el corpus. Destacaremos que la expresión dentro del código contiene algunos caracteres de escape "\" extras.

Luego de generar la tabla externa como vimos en la figura 4.2.3, seleccionamos los campos que nos interesan analizar: gram, año de aparición y cantidad de ocurrencias en ese año. Agregamos la expresión regular para filtrar lo que nos interesa de ellos. El corpus de 1grams en Español pasa a ser de 164009433 filas, a ser de 94852669, lo que representa una reducción aproximada del 40% luego de aplicar la normalización.

El siguiente paso consiste en crear una tabla (llamada pre-diccionario) que contenga la información necesaria. Este pre-diccionario es un conjunto de datos que nos servirá para crear el verdadero diccionario. La manera en la que realizamos esto es teniendo en cuenta la variable llamada "años de ocurrencia", la cual nos sirve para considerar solo los gramas que aparecen al menos un cierto porcentaje de años dentro de nuestra ventana, y así descartar los términos menos usados que no serán considerados parte del diccionario. El pre-diccionario contendrá para una ventana todos los gramas (*gram*) que aparecen al menos una vez dentro de la misma, la cantidad de ocurrencias (*ocurrences*), la cantidad de años en la cual aparecen (*yearOcurrences*) y el año en que finaliza la ventana (*year*).

Para realizar esta tarea, realizamos un loop en el rango de años a analizar. Primero inicializamos la ventana de tamaño W años, creando la tabla pre_diccionario, la cual va a contener la información de todos los gramas que aparecieron en la primera ventana (ver ejemplo en la figura 4.2.4). Recordamos que en éste corpus, si una palabra no es utilizada en un año, entonces no se encontrará en las filas de ese año.

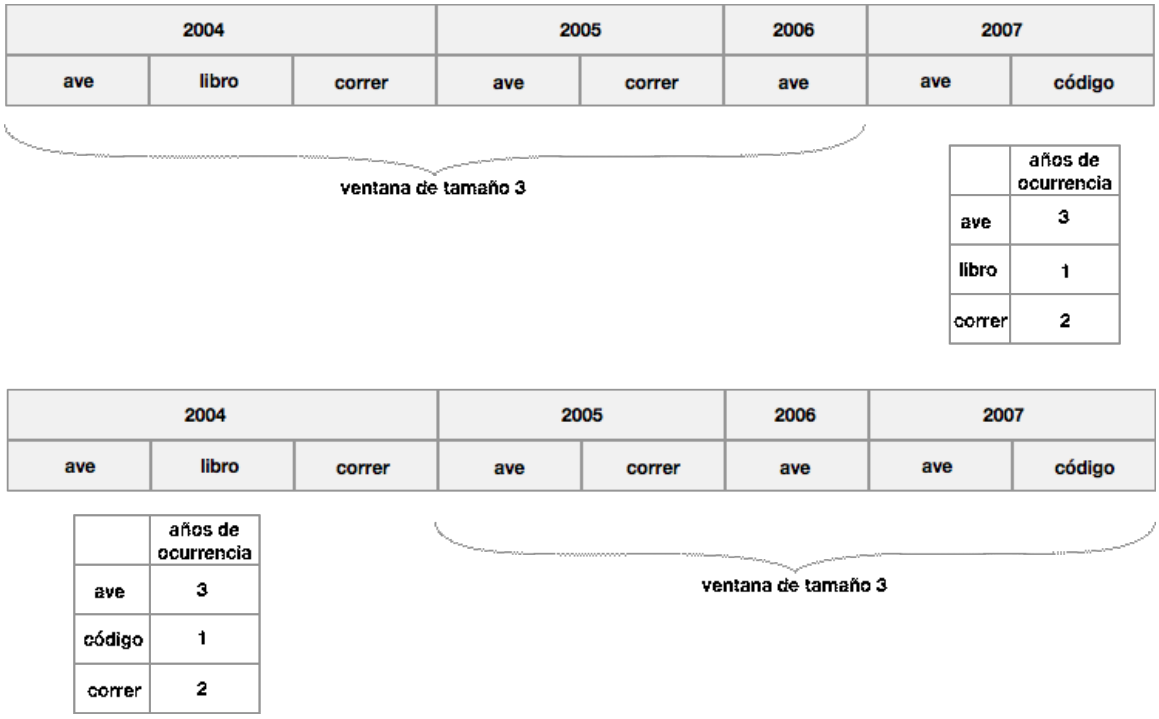

Figura 4.2.4: Movimiento de ventana sobre un corpus de ejemplo.

Para crear el pre-diccionario, utilizamos el script InitializeWindow.q (figura 4.2.5). Con él contabilizamos la cantidad de años distintos en los que aparece un grama, los cuales serán tenidos en cuenta más adelante. Creamos una tabla que contenga cada ventana, para ello tomamos la columna del año como el año en el que termina la ventana, y agregamos en ese año todos los gramas que aparecen en el rango junto con su cantidad de años de ocurrencia el cual estará entre 1 y W siendo W el tamaño de la ventana.

```
CREATE TABLE pre_dictionary_${ngramsTable} (
 gram string,
 year int,
 occurrences bigint,
 yearOccurrences int
)
;

INSERT OVERWRITE TABLE pre_dictionary_${ngramsTable}
SELECT
 t2.gram,
 ${toYear}-1,
 t2.totalOccurrences,
 t2.yearOccurrences
FROM
(
SELECT
  t1.gram as gram,
  count(DISTINCT t1.year) as yearOccurrences,
  SUM(t1.occurrences) as totalOccurrences
 FROM
  (
   SELECT
    gram, year, occurrences
   FROM
    normalized_${ngramsTable}
   WHERE
    year>=${fromYear} AND year<${toYear}
  ) t1
 GROUP BY
   gram
) t2
;
```

Figura 4.2.5: Script de Hive para inicializar la ventana del pre-diccionario (InitializeWindow.q).

El script de la figura 4.2.5 contiene 2 instrucciones, la primera simplemente crea una tabla de 4 columnas. La segunda es más compleja, contiene 2 sub-consultas anidadas que analizaremos a continuación.

La más interna de ellas, la cual es renombrada como "t1", simplemente selecciona los gramas desde la tabla "normalized_${ngramsTable}", que contiene los gramas normalizados, que se encuentran en el rango de años en el que crearemos la ventana. Podemos deducir que la diferencia entre *${toYear}* y *${fromYear}* es el tamaño de la ventana.

La segunda sub-consulta utiliza la sub-consulta "t1", y realiza la suma de las ocurrencias de cada grama en cada año, así como también la cuenta de en cuantos años distintos aparece un grama, mediante funciones de agregación[32] y la cláusula *GROUP BY*.

Una vez creado el pre-diccionario inicial, podemos empezar a mover la ventana para ir actualizandolo. Esto lo realizaremos moviendo la ventana de a 1 año, removiendo los datos del último año y añadiendo los datos del nuevo año.

Veremos que podríamos re-analizar toda la ventana en cada iteración, pero existen maneras más eficientes que evitan analizar tantas veces los mismos años (W veces cada uno, siendo W el tamaño de la ventana). En la figura 4.2.6 observamos el código con el bucle que le pasa los parámetros que el script shiftWindow.q (figura 4.2.7) requiere para agregar el corrimiento de la ventana al diccionario. La manera en que éste último funciona es obteniendo el nuevo año a analizar, re-analizando el año a ser removido de la ventana y re-analizando el último año de la ventana el cual es el que posee los datos totales de la ventana terminada en ese año, es decir que cada año es analizado a lo sumo 3 veces.

Si analizáramos todos los años de la ventana de tamaño W en cada corrimiento, realizaríamos un procesamiento con la siguiente eficiencia:

```
O((toYear – fromYear) * W) = O(N * W)
```

Es decir que cada año en el rango total a analizar es tratado a lo sumo W veces, lo cual no es muy eficiente en términos de procesamiento. Podemos realizar una pequeña mejora analizando sólo el nuevo año de la ventana, y el primero nuevamente para removerlo, actualizando los valores de la ventana en el siguiente año. Realizaremos N corrimientos de ventana, en el cual manejaremos 3 años cada vez, es decir:

```
O(3 * N) = O(N)
```

Esto se debe a lo que mencionamos previamente sobre reutilizar la información de la ventana. Podemos tener en cuenta la eficiencia a nivel espacio, lo cual también requiere de tiempo de lectura/escritura. Por ejemplo organizando la información de otra manera, en la cual no haya tanta redundancia de datos ya que con nuestro esquema cada grama frecuente aparece sucesivas veces en la misma

[32]Las funciones de agregación (UDAF, por sus siglas en inglés) son funciones de bases de datos que permiten tratar múltiples filas y agruparlas según ciertos criterios.

tabla, cuando podría aparecer sólo una vez junto a una lista de los años en el que aparece. También debemos considerar el tamaño de la entrada de datos. En nuestro caso, si utilizamos siempre la tabla normalizada con todos los gramas estaremos trabajando con aproximadamente 90 millones de filas en cada año, pero si reutilizamos la información en el diccionario la cual contendrá al menos el último año en la ventana, estaremos trabajando con una entrada mucho más pequeña. Dependiendo como hayamos creado la ventana en el paso anterior, podremos utilizar el diccionario para obtener también el primer año de la ventana que será el cual remover al correr ésta. Para mantener la sencillez del algoritmo, no nos detendremos a analizar en profundidad más optimizaciones en éste sentido.

```
for (int i = fromYear+windowSize; i<=toYear; i++){
      steps.add(
            emrh.getHiveStep(
                  getStepName(),
                  scriptsFullPath+"ShiftWindow.q",
                  createParameters(
                        "ngramsTable="+ngramsTable,
                        "newYear="+i,
                        "windowSize="+windowSize)));
}
```

Figura 4.2.6: Bucle Java para reutilizar el script “ShiftWindow.q” (Figura 4.2.7) a modo de plantilla para mover la ventana.

Para ayudar a entender mejor la figura anterior, recordaremos que luego de crear la ventana que va desde el año origen hasta el tamaño de la ventana, ejecutamos un bucle desde el año siguiente hasta el año destino. Por ejemplo, si la ventana tiene un tamaño de 5, y el rango de años es desde el 1800 al 1850, primero se creará la ventana con el rango 1800 a 1804 inclusive, y se ejecutará el bucle desde el año 1805.

Para lograr esto creamos 3 tablas auxiliares: *tmp_old_year* contiene los grams que aparecen en el año más antiguo (el que sale de la ventana), *tmp_new_year* contiene los gramas del nuevo año (el que ingresa en la ventana) y *tmp_last_year* contiene la fila del pre_diccionario correspondiente a la última ventana.

La manera en la que se crean las dos primeras tablas en cada iteración es sencilla, se toman los gramas del año deseado desde la tabla de gramas normalizados. La tabla *tmp_last_year* contiene los datos de la última ventana cuya información se extrae del último año desde la tabla pre_diccionario. Veremos en la siguiente figura el script de Hive, que realiza la tarea de actualizar la ventana.

```
INSERT INTO TABLE pre_dictionary_${ngramsTable}
SELECT
 combined_table.sq_gram,
 ${newYear} as year,
 combined_table.sq_occurrences -
  COALESCE(toy.occurrences, 0) as occurrences,
 combined_table.sq_yearOccurrences -
  COALESCE(toy.yearOccurrences, 0) as yearOccurrences
FROM
 (
  SELECT
   COALESCE(tly.gram, tny.gram) as sq_gram,
   COALESCE(tly.occurrences, 0) +
    COALESCE(tny.occurrences, 0) as sq_occurrences,
   COALESCE(tly.yearOccurrences, 0) +
    COALESCE(tny.yearOccurrences, 0) as sq_yearOccurrences
  FROM tmp_last_year as tly
  FULL JOIN tmp_new_year tny ON tly.gram=tny.gram
 ) combined_table
LEFT JOIN tmp_old_year as toy
ON combined_table.sq_gram=toy.gram
WHERE combined_table.sq_yearOccurrences -
      COALESCE(toy.yearOccurrences, 0) > 0
;
```

Figura 4.2.7: Instrucción de Hive principal para correr la ventana, actualizando la tabla con el pre-diccionario (shiftWindow.q).

La figura anterior parece mucho más compleja de entender que las anteriores, no obstante la complejidad de ella no reside en las múltiples llamadas a la función de agregación *COALESCE* la cual retorna el primer valor no nulo analizando la lista de valores pasada por parámetro, sino en la lógica de los *JOINS*.

Comenzaremos por analizar la sub-consulta interna. Todos los gramas que se encuentran en las tablas *tmp_last_year* y *tmp_new_year* deberán considerarse en la nueva ventana, realizamos un *FULL JOIN* el cual no excluirá ninguna fila. Veamos un ejemplo: si nuestra ventana es de 5 años y termina en el año 1992 y queremos moverla un año, *tmp_last_year* contendrá los gramas del año 1992, y *tmp_new_year* los del año 1993. En la siguiente sub-consulta utilizaremos la tabla *tmp_old_year* que en nuestro ejemplo serían los gramas del año 1988 ya que es el año que se deja de tener en cuenta al mover la ventana.

Las posibilidades son 3:

- El grama se encuentra en ambos años: sus ocurrencias se sumarán y se almacenará como un sólo grama. Los años de ocurrencia del grama en la tabla *tmp_last_year* será un número entre 1 y el tamaño de la ventana, y los años de ocurrencia del lado de la tabla *tmp_new_year* será 1.

- El grama sólo se encuentra en el último año de la ventana: el sector derecho de la tabla producida por la unión de ambos años se encontrará nulo, por lo que *COALESCE* retornará el valor 0 en cada llamada dentro de nuestra sub-consulta, y los valores se mantendrán iguales a los de la tabla *tmp_last_year* para ese grama.

- El grama sólo se encuentra en el nuevo año de la ventana: análogamente con el caso anterior, el lado izquierdo de la tabla producida por la unión de ambos años se encontrará nula, y los valores serán los de la tabla *tmp_new_year*. Podemos predecir que la cantidad de años de ocurrencia será 1.

La tabla resultante del *FULL JOIN* es renombrada a *combined_table*, y es utilizada en un *LEFT JOIN* con la tabla *tmp_old_year* la cual contiene los gramas del año a remover. La cláusula *LEFT JOIN* sólo conserva los gramas de la tabla de la izquierda, eliminando aquellas filas de la tabla que se encuentra a la derecha y no coinciden con la restricción especificada en *ON*. Si un grama no se encuentra en el primer año de la ventana, pero sí en algún año posterior, nos interesa conservarlo. Contrariamente no nos interesa conservarlo si el grama sólo aparece en el año a remover, lo cual debería ser imposible ya que el último año de la ventana contendrá el grama con años de ocurrencia en 1, y un *FULL JOIN* debería tener el mismo efecto. Lo que remueve los gramas que aparecen sólo en el primer año de la ventana es la cláusula *WHERE* en el final, que no permite gramas con ocurrencia 0 o menor.

Por último, mencionamos que si la ventana es pequeña existirá procesamiento en vano. Por ejemplo una ventana de tamaño 1 procesará el primer año y el último año de la ventana (los cuales son el mismo) sólo para anularlos mutuamente. Esto no representa un comportamiento inválido, solo uno innecesario, que no afecta el resultado correcto. Notar que el tipo de *INSERT* en este caso no sobrescribe los valores de la tabla (*INSERT OVERWRITE*) sino que simplemente agrega filas (*INSERT INTO*).

Una vez procesada nuestra ventana podemos crear el diccionario del lenguaje según el uso de sus palabras. El algoritmo necesita conocer cuál es el porcentaje de años de aparición necesario para que una palabra sea parte del lenguaje. Este porcentaje será expresado como un número entre 0,1 y 1,0 inclusive, lo que nos deja representar entre el 10% y el 100%. Como veremos en la figura 4.2.8 (ExportDictionary.q) los decimales serán truncados mediante la función *FLOOR* que nos retorna la cantidad mínima deseada de años de ocurrencia, la cual se calcula con el tamaño de la ventana. Por ejemplo, una ventana de 5 años con 90% de porcentaje requerido filtrará los gramas que ocurren al menos en 5 * 0,9 = 4,5 = 4 años dentro

de la ventana. Podemos destacar que existen combinaciones que son equivalentes debido a esto y no tiene sentido considerarlas por separado, así también como otras en las cuales sólo ciertos valores pueden ser utilizados. Por ejemplo el porcentaje carece de sentido cuando la ventana es de tamaño 1, el resultado será siempre el mismo.

```
CREATE EXTERNAL TABLE IF NOT EXISTS dictionary_${ngramsTable} (
 gram string,
 year int,
 occurrences bigint,
 yearoccurrences bigint
)
 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
 LINES TERMINATED BY '\n'
 STORED AS TEXTFILE
 LOCATION '${output}'
;

INSERT OVERWRITE TABLE dictionary_${ngramsTable}
SELECT *
FROM pre_dictionary_${ngramsTable}
WHERE
 yearOccurrences>=FLOOR(${windowSize} * ${percentOfYears})
ORDER BY year, occurrences DESC
;
```

Figura 4.2.8: Script que exporta un diccionario a S3 (ExportDictionary.q).

Al exportar una tabla a S3 estará disponible aunque el cluster se termine. En nuestro caso guardamos el archivo en forma de texto, separando las columnas por tabulaciones y las filas por saltos de línea.

Es posible que Hadoop cree algunos archivos con la terminación "*_$folder$*" en S3, los cuales son objetos vacíos que sirven a Hadoop como indicadores de que existe un directorio en ese lugar. Aunque el concepto de directorio no existe por sí mismo en S3, se simula con el prefijo del nombre de los objetos.

A continuación veremos en la figura 4.2.9 (ProcessNeologism.q) cómo podemos utilizar el diccionario creado para calcular los neologismos simplemente detectando los gramas que ingresaron en una ventana y que no están en la anterior.

```
INSERT INTO TABLE pre_neologisms_${ngramsTable}
SELECT *
FROM dictionary_${ngramsTable} as pdn
WHERE
 pdn.year=${year}
 AND NOT EXISTS (
  SELECT 1
  FROM dictionary_${ngramsTable} as dic
  WHERE
   dic.year=${year}-1
   AND
   dic.gram=pdn.gram
 )
;
```

Figura 4.2.9: Script para procesar posibles neologismos (ProcessNeologism.q).

Dado que la cantidad de resultados en ésta lista podría ser muy grande, aplicaremos un límite arbitrario a la hora de exportar los neologismos a S3, como veremos en la figura 4.2.10 (ExportNeologism.q). Anteriormente vimos que un límite promedio para el idioma Español podría ser 360 por año, pero podríamos en cambio concentrarnos en otras variables que afectan al algoritmo como la relación entre el tamaño de la ventana y el porcentaje de años de ocurrencia requerido, o utilizar más provechosamente la cantidad de ocurrencias para descartar palabras muy comunes o incluso pensar en una expresión regular más precisa, y limitar la salida a simplemente la cantidad de gramas que se desean analizar.

```
INSERT OVERWRITE TABLE neologisms_${ngramsTable}
SELECT gram, year, occurrences, yearOccurrences
FROM
 (
  SELECT
   *,
   rank() over (
    PARTITION BY sq.year
ORDER BY sq.occurrences DESC) as rank
  FROM pre_neologisms_${ngramsTable} as sq
 ) sq_table
WHERE sq_table.rank <= 20;
```

Figura 4.2.10: Script para seleccionar los 20 gramas más relevantes (ExportNeologism.q).

En la figura anterior utilizamos una función que no vimos hasta ahora, la función *rank()* la cual divide a la tabla según algún criterio (en nuestro caso, por año) y luego ordena esas divisiones por algún otro criterio (nosotros elegimos la cantidad de ocurrencias, de mayor a menor) asignándole a cada fila un valor entre 1 y N dentro de cada división. Esto lo realizamos para seleccionar los 20 gramas nuevos en un diccionario más utilizados de cada año. Destacamos que la función Rank asigna el mismo valor en caso de empate, por ende es posible que la consulta retorne más de 20 gramas, con algunos de ellos con la misma cantidad de ocurrencias.

De la misma manera en la que generamos el diccionario anterior, podemos generar otros diccionarios simplemente cambiando los parámetros enviados. A partir de ello realizar una consulta para comparar los gramas en común entre 2 lenguajes. Desde éste punto existen varias posibilidades en las que cierto grama podría encontrarse en ambos diccionarios:

- El grama es un extranjerismo que el idioma A tomó del idioma B: dependiendo la frecuencia de uso de un grama en un diccionario y en otro, podríamos intentar estimar su origen en alguno de los 2 idiomas.

- El grama está compuesto por la misma cadena de caracteres: existe la posibilidad de que la misma secuencia de caracteres representen distintos gramas, con distintos significados o incluso el mismo, sin necesidad de que uno haya sido tomado del otro.

- El grama es un extranjerismo en ambos idiomas, tomado de un tercero: existen varios casos en los que un grama se encuentra en varios idiomas con distinta cantidad de usos y su origen sea el de otro. Un ejemplo sucede con la palabra “garage" la cual aparece simultaneas veces tanto en Español como en Inglés, pero tiene origen Francés.

Un experto en lingüística seguramente podría detectar más razones por las cuales 2 gramas aparecen en 2 idiomas distintos. En nuestra investigación nos limitaremos a la primera alternativa, calculando arbitrariamente si una diferencia entre las ocurrencias de un grama en un idioma son mucho mayores a las del otro. Una alternativa es calcular el porcentaje que representa un grama sobre el total de gramas para ese idioma en ese año, y calcular la diferencia con las mismas variables aplicadas al otro idioma. Por sencillez, utilizaremos un enfoque menos preciso pero más simple, que es el de corroborar si la diferencia de ocurrencias es N veces mayor.

4.3 Resultados

Analizaremos el rendimiento que ofrecen las distintas instancias al ejecutar el algoritmo y los resultados obtenidos mediante éste.

Tipos de instancia

Como vimos en el capítulo 3, AWS ofrece distintos tipos de instancia con distintas prestaciones. Realizamos un experimento al ejecutar el código desarrollado con los mismos parámetros. Analizamos el rango de años desde 1990 hasta el año 2000, con una ventana de 5 años y un porcentaje de aparición anual del 80%, sobre el idioma Español influenciado por el idioma Inglés. Esto implica importar ambos ngrams, generar los diccionarios desde 1995 al año 2000 en ambos idiomas, buscar los extranjerismos, procesar los neologismos en el mismo rango que los diccionarios, y exportar todos los resultados a S3. Veremos en la figura 4.3.1 cómo el rendimiento no varía demasiado a pesar de tener una cantidad algo mayor o menor de recursos. Para estas pruebas usamos un cluster de tamaño 5, donde las instancias son del tipo xlarge para el nodo master y x2large para los 4 nodos slave restantes.

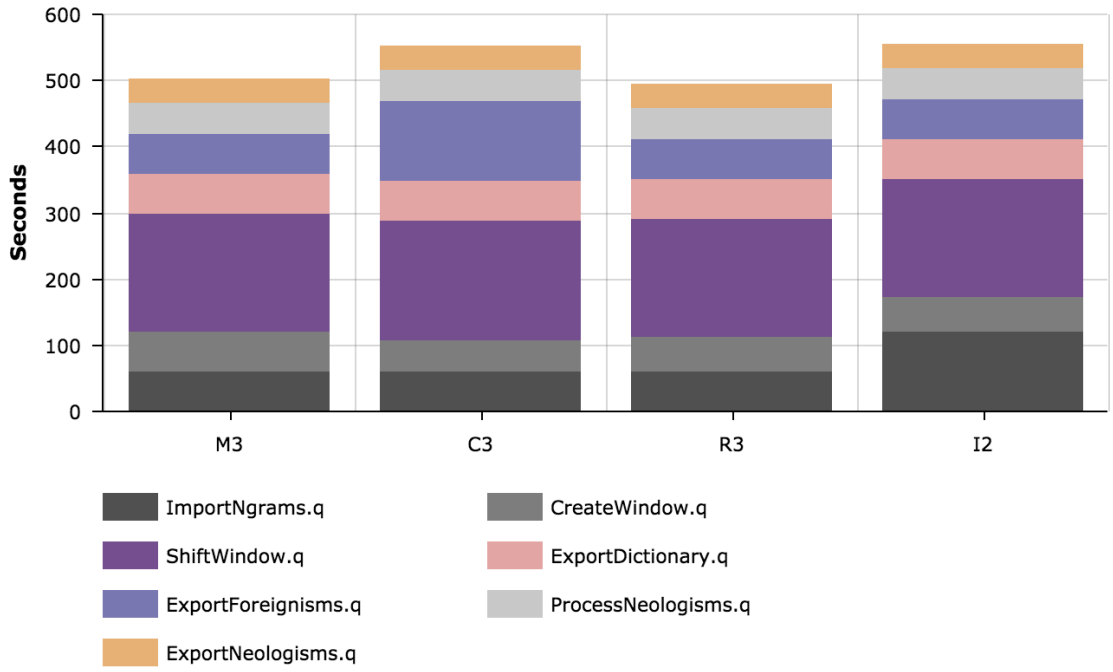

Figura 4.3.1: Tiempo de ejecución en segundos para cada step por tipo de instancia.

Si nos detenemos a analizar la consola web, en la sección de monitoreo podremos observar algunas métricas como son las cantidad de bytes leídos y escritos en S3. Y el porcentaje de HDFS utilizado. Si nos preocupa la utilización

eficiente de recursos, es decir desperdiciar la menor cantidad posible, podemos ver que en cualquiera de los casos desperdiciamos una gran cantidad de ellos. Nuestra aplicación requiere leer aproximadamente 9GB en HDFS, el sistema de archivos interno del cluster, y escribir aproximadamente 3GB. El tipo de instancia con menor cantidad de memoria es el C3, el cual ofrece 7.5GB y 15GB para instancias del tipo xlarge y x2large respectivamente, podríamos decir que es el tipo que realiza el menor desperdicio de memoria, de los que hemos analizado. El peor caso sería el del tipo de instancia I2, que ofrecen 30.5GB y 61GB para los mismo tipos, junto con una capacidad de almacenamiento de 800GB y 1600GB, el cual es desperdiciado por más del 99%, ya que nuestra aplicación solo requiere aproximadamente 9GB para todo el cluster.

Tamaño del cluster

AWS nos limita el número por defecto de instancias EC2 a 20 por cuenta, para todas las regiones. Si intentamos alocar más que ésta cantidad, obtendremos un error de validación. Éste número puede ser incrementado, pero podemos anticipar que no necesitaremos hacerlo.

Aumentar el tamaño del cluster no siempre mejorará el rendimiento, para intentar encontrar el número adecuado de instancias lanzamos 5 clusters de distintos tamaños con los mismos parámetros que en el experimento anterior.

En cada una de las pruebas usamos un nodo para el master y el resto para slaves, utilizando un proceso por nodo, excepto en el cluster de tamaño 1 donde el mismo nodo ejecuta el master y el slave. Veremos el rendimiento en la figura 4.3.2.

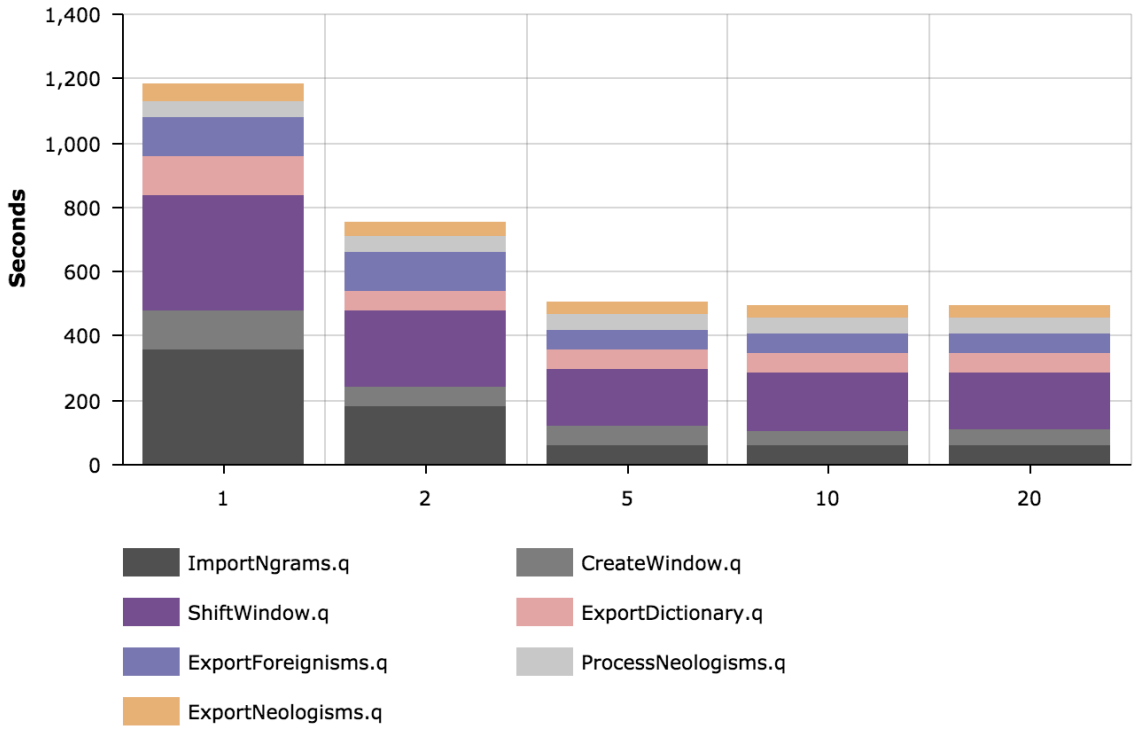

Figura 4.3.2: Mejora del rendimiento al incrementar el tamaño del cluster.

Podemos observar que al aumentar el tamaño del cluster inicialmente disminuye el tiempo de ejecución, pero encuentra un límite rápidamente al superar los 5 nodos, a partir de lo cual se estabiliza. Estos nodos son del mismo tipo que en experimentos anteriores, m3 xlarge y x2large.

Costo económico

AWS calcula el costo de ejecutar un cluster por horas de instancia. A modo de ejemplo, un cluster de 1 instancia que se ejecuta por 10 horas representará 10 horas de instancia, al igual que un cluster de 2 instancias que se ejecuta por 5 horas.

La diferencia se encuentra en el costo del tipo de instancia. En nuestros experimentos utilizamos el tipo m3 x2large la mayoría de las veces, la cual al día de la fecha tiene un costo de 0,672 dólares[33] por hora de instancia para EMR. Ejecutar todos los scripts varias veces para generar los diccionarios, extranjerismos y neologismos tomó en promedio 1 hora con 2 minutos en cada ejecución con 5 o más instancias. Podemos concluir que cada ejecución costó aproximadamente:

```
1 hora de instancia * 5 instancias * $0,672 = $3,36
```

Podemos intentar estimar el costo desperdiciado en instancias que no eran necesarias, como cuando se utilizaron 20 de ellas:

```
1 hora de instancia * 20 instancias * $0,672 = $13,44
```

En ambos casos estamos obviando un detalle menor, que es que no todas las instancias son del mismo tipo, el master puede ser de un tipo de instancia distinto. Pero dejamos en evidencia que algunos de los análisis que mencionamos, como es el de 20 instancias de tamaño, tuvieron un gasto innecesario, en el peor caso de $10,08.

Como detalle final, destacamos que si el tiempo de ejecución no nos fuera de interés podríamos realizar estos experimentos con 1 instancia para disminuir aún más el costo. El experimento que se realizó con sólo 1 instancia tardó aproximadamente 2 horas con 24 minutos, con una instancia del tipo xlarge, la cual tiene un costo de $0,336 la hora:

```
2.5 horas de instancia * 1 instancia * $0,336 = $0,84
```

Claramente si nuestro interés principal es reducir costos, podemos tener un acercamiento de éste estilo. Aunque debemos considerar que en estudios más exhaustivos esto podría ser muy lento, y no aprovecha los beneficios de MapReduce.

Diccionarios

En los experimentos realizados generamos los diccionarios en Español y en Inglés, analizamos la cantidad de gramas en cada uno. Recordamos que estos

[33]https://aws.amazon.com/elasticmapreduce/pricing/ (Visitado: Enero 2016)

diccionarios se generaron sobre los gramas filtrados con la expresión regular vista previamente, y en un rango de años de 1925 al 2000. Calculamos el promedio de palabras por año, teniendo en cuenta que la ventana consume parte del rango, y obtuvimos los resultados de la figura 4.3.3.

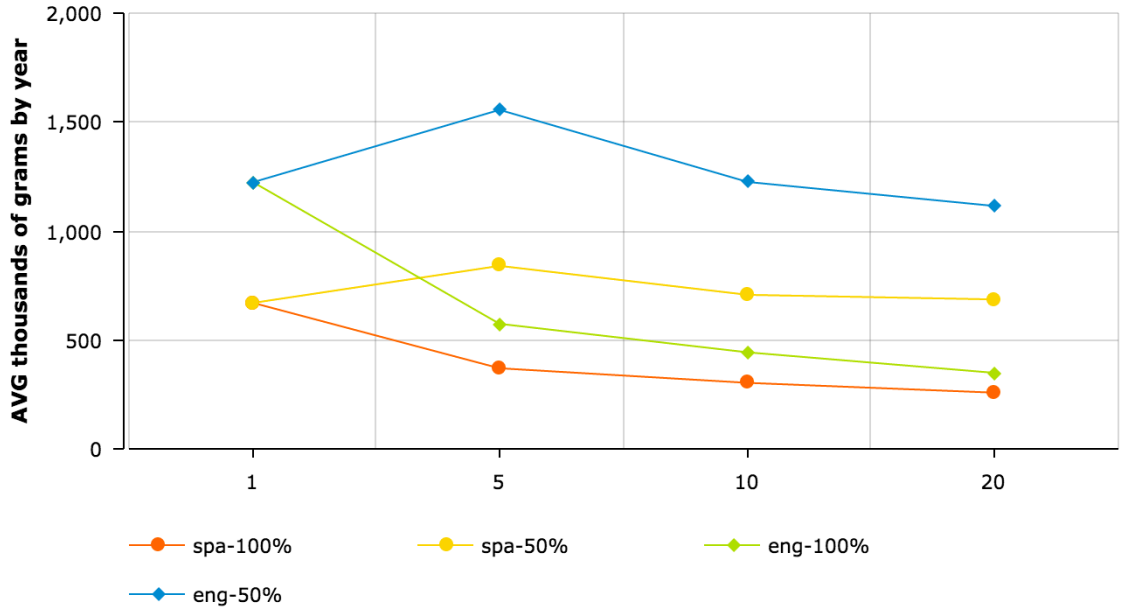

Figura 4.3.3: Promedio de miles de gramas por año, dependiendo el tamaño de ventana y años de ocurrencia (50% o 100%) requeridos.

Dado a que el corpus es más grande en el idioma Inglés que en el Español, es natural que tenga más gramas por año. Esto no significa que un lenguaje tenga más gramas que el otro en el habla común.

Cuando el tamaño de ventana es 1, considerar cualquier porcentaje de años de ocurrencia es lo mismo, dado a que esto exige el mínimo de años que un grama debe aparecer en la ventana, pero siendo el mínimo de apariciones de un grama por año igual a 1, el parámetro pierde sentido.

Vemos que si consideramos un porcentaje menor de años de ocurrencia sobre la misma información y tamaño de ventana, obtendremos más gramas, lo cual es lógico ya que un resultado incluye al otro: todos los gramas que aparecen el 100% de los años, también aparecerán en al menos el 50% de ellos.

Si consideramos el 100% de porcentaje de años de ocurrencia, al incrementar la ventana la cantidad de gramas promedio por año desciende pero se acerca a un valor, la cantidad de gramas más comunes del idioma. Si quisiéramos saber cuáles son éstos en los últimos 10 años del corpus, sería tan sencillo como calcular los gramas que aparecen en esa ventana, con un 100% de ocurrencia de años.

Por último, observamos que a pesar de lo que mencionamos en primera instancia que es que el corpus es más extenso en el idioma Inglés, nuestro algoritmo considera las palabras distintas. Cuando ajustamos los parámetros para ser más

restrictivos (incrementar el tamaño de ventana y porcentaje de años de ocurrencia) los lenguajes comienzan a acercarse, conteniendo el Inglés y el Español 349.000 y 258.000 gramas promedio por año, respectivamente.

Neologismos

Analizaremos los neologismos obtenidos en el idioma Español. En la sección 4.2 vimos cómo los calculamos, y que restringimos su resultado a 20 por año (siendo posible obtener algunos más en caso de empate en la última posición). En los diferentes experimentos, ésta cantidad varió de 1120 a 1513, decreciendo cuando la ventana crecía, lo cual es esperable ya que incluye cada vez menos gramas. Analizaremos los resultados generados a partir de diccionarios creados con 100% de porcentaje de años de ocurrencia.

En la ventana más pequeña, de 1 año, obtuvimos entre otros los siguientes neologismos: epididimarios (2000, tipo de quiste), microcalcificaciones (2000, pequeños depósitos de calcio en tejidos), himbierno (1998, error ortográfico que aumentó considerablemente su frecuencia), tafíes (1990, Gentilicio), píxeles (1974, punto de una imagen), crótalus (1965, especie de serpiente), ocitocina (1957, hormona descubierta en los 50'. Idioma portugués), dpto (1928, abreviatura de "departamento").

Podemos observar estos neologismos y su curva de uso en el visualizador de los Google Ngrams[34]. Si analizamos la lista completa uno a uno, notaremos que aparecen muchos términos de medicina, gentilicios, errores ortográficos, extranjerismos, y algunos neologismos. La ventaja de tener una ventana de 1 año es que el neologismo puede ser detectado muy rápidamente desde su aparición, pero la desventaja es la gran cantidad de ruido, es decir, todo lo que no es un verdadero neologismo. Además, existen palabras de poca frecuencia de uso, que podrían aparecer varios años si su utilización sube y baja frecuentemente. Veremos que neologismos obtenemos con la ventana de 20 años: senderista/s (2000, Partidarios de la organización peruana terrorista llamada Sendero Luminoso, surgida en 1980), fordista (1999, termino surgido en 1980 para describir un sistema socioeconómico muy discutido), antisandinistas (1998, Opositores al régimen del Frente Sandinista de Nicaragua derrocado en 1979), intis (1996, moneda peruana utilizada desde 1985 actualmente en desuso), computadora (1972, concepto de máquina surgido en los 50), estreptomicina (1963, antibiótico descubierto en 1943), penicilina (1959, antibiótico descubierto en 1928, sus creadores ganaron el premio Nobel en 1945), siudad (1946, error ortográfico de "ciudad", muy común a principios del siglo XX).

Los resultados que obtuvimos con una ventana más grande fueron mucho más acertados que con una ventana pequeña. La gran mayoría de los términos son neologismos de la época. No es difícil percibir que existe un problema con éste enfoque, y es el que en todos los casos los neologismos fueron detectados al menos 20 años (el tamaño de la ventana) luego de su verdadero surgimiento. Esto podría ser un inconveniente grande para este método ya que no es muy útil encontrar un neologismo 20 años luego de su aparición, es posible que ya no sea un neologismo

[34]https://books.google.com/ngrams/ (Visitado: Enero 2016)

en la fecha de descubrimiento. Pero esto se debe a que la unidad mínima de los Google Books Ngrams es de 1 año. Podríamos preguntarnos qué sucedería si tuviéramos un corpus por días, sería tan sencillo como utilizar una ventana de 20 días para encontrar un neologismo. Pero la neología suele interesarse en descubrir los nuevos términos tan rápido como sea posible, generalmente en el mismo día. ¿Qué sucedería entonces si tuviéramos un corpus con muchas entradas frecuentes, como puede ser un foro, y pudiéramos analizarlo por hora, o por minutos, o por segundos?

Extranjerismos

La manera en la que intentamos identificar los extranjerismos fue a partir de la generación de 2 diccionarios en 2 lenguajes distintos, identificando aquellos términos que eran ampliamente más utilizados en una lengua que en otra. No separamos los extranjerismos por año ya que no nos interesó conocer cuándo fueron acuñados, simplemente cuales lo fueron.

Entre ellos, encontramos las palabras más comunes del idioma Inglés: the, to, and, of, in, was, for...

Esto tiene varios orígenes, como veremos en la sección de ruido. Principalmente destacamos el hecho de que no hay manera sencilla de filtrar citas con los Google Books Ngrams, es decir, identificar las comillas (") para analizar esos gramas de manera distinta. Algo posible es el de utilizar, por ejemplo, 3grams, e intentar analizar si el grama intermedio está acompañado de gramas que también son extranjerismos o no. De esta manera descubriríamos los extranjerismos aislados, pero no los compuestos por más de una palabra, o que estén en contextos confusos.

Este método podría ser útil para otro tipo de corpus, pero en el caso de los ngrams el método debería ser mucho más sofisticado, y quizás no sea razonable forzar la utilización de éstos, y simplemente utilizar otro corpus.

Ruido

Existen varios problemas que llamamos ruido. Con esta expresión nos referimos a elementos indeseados que interfieren con los resultados obtenidos, ensuciándolos.

Podemos mencionar varias razones, a nivel de implementación del algoritmo podemos decir que no es infalible, y retornará algunos errores como palabras que no son neologismos, o palabras que sí lo son pero que surgieron antes del año indicado (incluso teniendo en cuenta el corrimiento de ventana). A nivel de corpus tenemos otros tipos de errores. Los errores tipográficos suelen ser comunes en cualquier corpus, palabras mal escritas o utilizadas incorrectamente. Con nuestro método, añadimos al diccionario todas las palabras que fueran comunes (incluso errores tipográficos comunes, o por ejemplo números romanos como “vi”, “vii”, “xiv” entre otros). Pero existen otros errores que descubrimos a lo largo del desarrollo de este trabajo, como por ejemplo libros marcados como pertenecientes a un idioma que en realidad pertenecen a otro. Creemos que el origen de esto es la inmigración, ya que muchos libros etiquetados como españoles se encuentran en Italiano, entre

otros, lo que produce cierta cantidad de ruido, pero no descartamos la posibilidad del error humano. También podemos mencionar los distintos dialectos, en Inglés tenemos algunas separaciones como por ejemplo en Inglés británico y americano. Pero en español existen muchos más, que se encuentran en un mismo corpus.

Otro tipo de error común que genera una gran cantidad de ruido es el error provocado por OCR, en especial en textos antiguos que fueron impresos en baja calidad. En el año 1800 hubo un gran cambio en el uso de una letra llamada "S larga" la cual es muy parecida a una "f" (ſ) y tenía la misma pronunciación que la "S" de hoy en día. Esta fue dejada de utilizar repentinamente en los comienzos del siglo XIX, provocando muchos errores de OCR como destacaron algunos usuarios en 2010 (Sullivan 2010).

Este tipo de errores nos indica que quizás los corpus generados a partir de libros no son los más indicados. Además, la ortografía fue cambiando con los años (por ejemplo "arpa" comenzó a reemplazar a "harpa", o "Sofía" a "Sophia") y los errores ortográficos eran muy comunes, incluso en escritores reconocidos. Hoy en día dependiendo el origen del corpus, los errores ortográficos pueden también ser comunes, lo que agrega complejidad para la generación de una solución totalmente automática, que no requiera a un profesional en lenguas que detecte estos casos.

5 Conclusiones

Al principio del desarrollo de esta tesis nos encontramos con varios problemas e incertidumbres por falta de investigaciones reutilizables o lo suficientemente generales relacionadas al estudio de la neología como para ser tomadas como punto de partida. Algunas de ellas sí nos fueron de utilidad, y nos introdujeron a ciertos problemas comunes que necesitaban una solución.

Nuestro foco comenzó siendo puramente el análisis de datos en una herramienta particular: AWS, la cual nos proveyó de una infraestructura barata pero poderosa, y ampliamente utilizada. Una vez interiorizados en la tecnología disponible, nos interesamos por los problemas de análisis de lingüística al conocer que hace unos pocos años existían corpus exhaustivos que podían ser utilizados para éste tipo de análisis, y por razones que desconocemos no estaban siendo tan ampliamente utilizados.

Decidimos comenzar a incorporar ciertos temas a nuestra investigación, y nos propusimos desarrollar algo que sea útil para la comunidad. Por un lado, proponer un análisis claro para el estudio de neologismos, el cual requiera sólo un corpus diacrónico. Dicho corpus nos serviría para generar un diccionario de palabras utilizadas, lo que no significa que estemos en busca de un diccionario real, sino el de una lista de exclusión con términos comunes a lo largo del tiempo, que no nos interesan ya que no son neologismos. Nos decidimos por no limitarnos por el tamaño que pueda llegar a tener esta lista, cualquier cantidad de información digitalizada es analizable, a cierto costo. Vimos cómo varias aplicaciones utilizan blogs o sitios web reducidos, y limitan mucho su fuente de recursos muchas veces sin explicar por qué. Entendemos que es posible que la razón sea falta de recursos para una cantidad de información muy grande, y también de trabajo previo existente para tomar como referencia de qué hacer o qué no hacer.

Al desarrollar un algoritmo que solucione la problemática planteada, intentamos ser claros y mantener la sencillez de la solución. Con 7 scripts de Hive pudimos procesar una gran cantidad de información, de manera simple. Además, mostramos cómo preparar el entorno de AWS para esto, lo cual puede ser engorroso en un comienzo, en nuestra experiencia nos topamos con varios problemas para realizar un algoritmo que lance una instancia generando los steps por sí misma. Creemos dejar en claro que es posible utilizar éstas tecnologías para éstos tipos de investigaciones, en especial cuando encontramos ejemplos a seguir como el que nosotros exponemos.

Al intentar buscar un número para el tamaño del cluster, nos vimos sorprendidos al descubrir que no era necesaria una gran cantidad de máquinas. Sabíamos que el tiempo de ejecución del algoritmo iba a disminuir aumentando el tamaño del cluster hasta cierto punto, pero no creímos que iba a ser tan pequeño. Descubrimos que en parte, esto sucede por la secuencialidad de los steps de Hive. Podríamos diseñar una solución que utilice algún sistema de paralelización, por ejemplo creando varios clusters y monitoreando su ejecución para saber cuándo un recurso está disponible. O también para procesar el rango de años en paralelo,

siempre que estemos dispuestos a analizar la misma información más de una vez para generar la ventana de cada sector.

Realizamos experimentos que nos otorgaron varios resultados, los cuales nos conformaron ampliamente al comprobar que eran acertados casi en su totalidad. Utilizamos esta metodología para probar que nuestro punto de vista es plausible y aplicable en corpus con unidades más pequeñas que 1 año. Apoyándonos en estudios previos implementamos una solución sencilla y eficaz, que puede ser extendida y escalada fácilmente.

Esperamos que futuros investigadores puedan basarse en éste método para sus estudios, y reutilicen la información presente. Además, que sigan en la línea de la reutilización de contenidos y desarrollen material útil para terceros, como creemos haber logrado realizar.

5.1 Trabajo futuro

Dentro de las tareas que escaparon al alcance de esta tesis podemos destacar la extensión del algoritmo a una solución más eficiente, y con más opciones de configuración. La paralelización de la generación de tareas, mediante la implementación separada de módulos independientemente ejecutables.

Disminuir el ruido al filtrar el corpus de una manera más precisa, y procesarlo en una forma menos general. Estas modificaciones también pueden ser módulos nuevos en el algoritmo.

Utilizar el enfoque diacrónico para la generación del corpus de exclusión a partir de la web, como han hecho otros autores pero con mayores limitaciones que ya hemos resuelto.

Implementar el descarte de palabras extranjeras, ya que éstas no son verdaderos neologismos, al menos no para nuestro interés. Dicho descarte puede ser realizado de la manera en la que generamos la lista de extranjerismos, ajustando los parámetros para que generen un diccionario lo suficientemente confiable para eliminar la mayoría de las citas en otros idiomas al del análisis.

Utilizar los demás ngrams, por ejemplo podríamos utilizar los 3-grams para considerar el contexto que rodea a una palabra, y tener en cuenta la relación entre las palabras que suelen ir juntas.

Generar un sistema que analice libremente la web o un subconjunto de ésta, de una manera robusta. El problema de los llamados “crawlers” (indexadores y analizadores de la web) es que suelen depender de la interfaz que ofrecen los sitios que analizan. Creemos posible el desarrollo de una lista de exclusión a partir de la información plana de la web, mediante filtros como nuestra expresión regular o similares. De la manera en la que nosotros trabajamos la información sólo era suficiente que el término sea utilizado frecuentemente para no ser considerado un neologismo, y las demás palabras que se incorporaran con el paso del tiempo (por ejemplo los días en un foro) serán neologismos.

Estudiar la posibilidad de determinar alguna métrica que dependiendo del tamaño original del archivo a analizar y las tareas a ejecutar estime el tamaño máximo del cluster de modo de optimizar el uso de los recursos económicos.

Apéndice A: Conceptos de Neología

Enumeraremos y describiremos brevemente conceptos de lingüística relacionados al tema de estudio para dar un mejor entendimiento al lector, analizando las influencias historiológicas de algunos idiomas en el mundo.

Neologismos

Un neologismo es, en términos generales, toda palabra nueva o un giro nuevo que se introduce en una lengua. En el caso concreto de la lengua Española, deben haber sido recientemente aceptadas por la Real Academia Española (RAE) y figurar en su diccionario para ser aceptados como neologismos formales, pero existen otros que aún no fueron aceptados y se consideran neologismos potenciales, que pueden caer en alguna otra categoría como por ejemplo la de extranjerismos.

Las palabras "nuevas" de una época se generalizan con el paso del tiempo y dejan ser neologismos (etimológicamente, *neo* significa *nuevo* y *logo* significa *palabra*, provenientes del griego), pasando a ser palabras de uso común.

Algunos neologismos son palabras ya existentes que adquieren significados nuevos, otros se inventan, se crean por derivación o composición, se toman de otras lenguas y se adaptan ortográficamente para que tengan una pronunciación más sencilla y más parecida al idioma original.

Ejemplos pueden ser *internet* la cual hoy en día es de uso común y es aceptada por la RAE, *piratería* que cambió su significado para referirse también a la descarga ilegal de contenidos vía internet, *descarga* en el contexto del ejemplo anterior por las mismas razones, *feminicidio* como neologismo reciente proveniente del inglés *feminicide*.

Como podemos observar, la informatización es una de las principales (sino la principal) fuente de neologismos, pero no la única.

Barbarismos

Los barbarismos consisten en la utilización de palabras mal formadas o alteradas por la influencia de una lengua extranjera y, por ende, no siguen las reglas de la lengua en las que se usa. Algunos barbarismos de la lengua española son por ejemplo el uso de *serio* por *grave* o *jugar un papel* por *desempeñar un papel*, ambos barbarismos con influencias del inglés.

El término se refiere a los *bárbaros* en el sentido a lo extranjero. La palabra *bárbaro* surgió en Grecia cuando los griegos imitaban el sonido de sus inmigrantes de otras tierras que hablaban otro idioma, y para ellos sonaba como "bar bar bar".
Muchas veces éstos son confundidos con extranjerismos y es difícil marcar un límite entre qué es considerado un barbarismo o un extranjerismo, especialmente con palabras o expresiones que siguen correctamente las reglas gramaticales de otra lengua. En estos casos, suele considerarse como barbarismo si ya existía una expresión apropiada aceptada en la lengua de origen. Por ejemplo *gourmet* del

francés en lugar de *gastronómico*, *vedette* también del francés en lugar de *estrella de cine* o *show* (ya aceptada por la RAE) del inglés en lugar de *espectáculo*.

Extranjerismos

Los extranjerismos son barbarismos que consisten en emplear palabras, frases y giros extranjeros con la ortografía original y una pronunciación que imita a la lengua de origen. Pueden ser vistos como préstamos que se toman de otras lenguas.

Suelen calificarse con distintos nombres según su proveniencia: *anglicismos*, del inglés, *galicismos*, del francés, *germanismos,* del alemán, *italianismos,* del italiano, entre otros. Cuando la RAE no los ha aceptado aún es recomendable evitar su uso formal.

A capella, proveniente del italiano, es un extranjerismo que indica que una obra o pieza musical se canta sin acompañamiento musical. *Karaoke*, del japonés, que significa interpretar la voz de una canción sobre su banda sonora. *Élite*, proveniente del francés, para referirse a un grupo selecto de personas. *Karma,* del hinduismo, para referirse a lo que uno merece en base a sus acciones. *Vudú,* proveniente de varias lenguas africanas, para referirse a un tipo de ritual espiritista.

Observamos que los ejemplos vistos no poseían una traducción sencilla, por lo que naturalmente se eligió incorporar estos términos a la lengua. Algunos mantuvieron su escritura y pronunciación, como *karaoke*, y otros fueron adaptados para que sigan las reglas del idioma, como en el caso de *vudú.*

La RAE muchas veces incorpora éstos términos en nuevas ediciones, y en otras puede eliminarlos.

Arcaísmos

Los arcaísmos son elementos de una lengua que quedaron en desuso y sólo se utilizan en contextos muy específicos. Aquel elemento o término solía tener un uso cotidiano, pero luego fue reemplazado por otro término, desaparecido en parte (arcaísmo relativo) o por completo (arcaísmo absoluto).

En nuestra investigación nos interesa considerar que existen palabras que pueden utilizarse menos con el paso de los años y luego, reaparecer. Stenetorp (2010) plantea que una palabra olvidada pero que se vuelve a utilizar sigue siendo novedosa para los hablantes actuales. No nos detendremos a considerar si este es el caso de un neologismo, o simplemente un término que dejó de ser un arcaísmo, pero los tendremos en cuenta a la hora de plantear una hipótesis para solucionar el problema planteado en el capítulo 4.

A.1 Principales influencias en la lengua española

En los siglos XV y XVI la mayor influencia era el italiano. Luego en los siglos XVII y XVIII predominó la influencia francesa. En los últimos 3 siglos la influencia principal en el idioma español es inglesa.

No es casualidad que en esos siglos el mundo estaba muy influenciado por países que hablaran esas lenguas. Esto se debe a grandes hechos históricos: El Renacimiento, surgido en Italia, hizo que se comiencen a tomar préstamos del Italiano; El Imperio Francés, comandado por Napoleón, fue la mayor potencia mundial de fines del siglo XVIII, en el cual ocurrió la Revolución Francesa, que caería a principios del siglo siguiente; El imperio Británico luego de la caída del imperio Francés en la batalla de Waterloo se transformó en la mayor potencia del siglo XIX, el cuál fue quedando de lado por el auge industrial de otros países como Estados Unidos y Alemania, de los cuales el primero es el más influyente hoy en día.

Podemos ver cómo la historia tuvo mucho que ver en cuáles eran los lenguajes en foco, pero existen muchos otros factores como la similitud y las raíces de los idiomas o la inmigración en masa de ciertos países a otros.

Los anglicismos de hoy suelen ser incorporados debido a que las principales agencias de noticias del mundo son angloparlantes, el poder del *marketing* del capitalismo y su publicidad, los deportes, y los medios audiovisuales. Todo esto acompañado por la globalización e informatización, lleva a que se reciban mucha influencia de la principal potencia mundial.

Bibliografía

Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

Maarten Janssen. NeoTrack: semiautomatic neologism detection. Paper presented at APL, 2005.

Maarten Janssen. Orthographic Neologisms: selection criteria and semi-automatic detection. Unpublished manuscript, 2005.

Freixa, J., & Solé, E. Análisis lingüístico de la detección automática de neologismos léxicos. Sendebar. Revista de Traducción e Interpretación, 17. 2006.

Maarten Janssen. Detección de Neologismos: una perspectiva computacional. Debate Terminológico, vol.: 5, pp. 68 - 75. 2009.

Pontus Stenetorp. Automated Extraction of Swedish Neologisms using a Temporally Annotated Corpus. Master of Science in Engineering (MSc Eng) Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, March 2010.

Jean-Baptiste Michel et al. Quantitative Analysis of Culture Using Millions of Digitized Books.December, 2010.

Danny Sullivan. When OCR Goes Bad: Google's Ngram Viewer & The F-Word. 2010. http://searchengineland.com/when-ocr-goes-bad-googles-ngram-viewer-the-f-word-59181 (Visitado: Enero 2016)

Garcia-Fernandez, Anne, Ligozat, Anne-Laure, Dinarelli, Marco, Bernhard, Delphine, Méthodespour l'archéologielinguistique: datation par combinaisond'indicestemporels. Atelier DEFT 2011, Actes de TALN 2011.

IDC. Big Data: What It Is and Why You Should Care. 2011.

Janssen, Maarten. NeoTag: a POS Tagger for Grammatical Neologism Detection. Proceedings of LREC, Istanbul, Turkey, 2012.

Yuri Lin et al. Syntactic Annotations for the Google Books Ngram Corpus. ACL 2012.

Kerremans, Daphné, Susanne Stegmayr and Hans-JörgSchmid, "The NeoCrawler: identifying and retrieving neologisms from the internet and monitoring on-going change". In: Kathryn Allan and Justyna A. Robinson, eds., Current methods in historical semantics, Berlin etc.: de Gruyter Mouton, 59-96. 2012.

Liu, Tsun-Jui, Shu-Kai Hsieh, and Laurent Prévot. "Observing Features of PTT Neologisms: A Corpus-driven Study with N-gram Model." ROCLING. 2013.

Thomas Erl “Cloud Computing: Concepts, Technology & Architecture” Prentice Hall 2013.

Viktor Mayer-Schonberger and Kenneth Cukier, Big Data: A Revolution That Will Transform How We Live, Work and Think. Canada: Eamon Dolan/Houghton Mifflin Harcourt, 2013

Falk, I., Bernhard, D., & Gérard, C. From Non Word to New Word: Automatically Identifying Neologisms in French Newspapers. In LREC-The 9th edition of the Language Resources and Evaluation Conference. May, 2014.