MODELO DE PREDICCIÓN SECUENCIAL DE WEB ACCESS LOG BASADO EN ALGORITMO DE COMPRESIÓN Y MACHINE LEARNING

JAIME GUZMÁN

Tesis para optar al título de Ingeniero Civil en Informática y Telecomunicaciones

Profesor guía: Adin Ramirez

FACULTAD DE INGENIERÍA
ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Santiago, Chile
2015

MODELO DE PREDICCIÓN SECUENCIAL DE WEB ACCESS LOG BASADO EN ALGORITMO DE COMPRESIÓN Y MACHINE LEARNING

JAIME GUZMÁN

Tesis para optar al título de Ingeniero Civil en Informática y Telecomunicaciones

Profesor guía:	**Adin Ramirez**
Comité:	**Francisco Claude**
Comité:	**Darth Vader**

FACULTAD DE INGENIERÍA
ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Santiago, Chile
2015

Jaime Guzmán
mail@jguzman.cl

Utilice un par de oraciones para dedicar su tesis, o una frase de alguién importante.

Agradecimientos

Nota redactada sobriamente en la cual se agradece a quienes han colaborado en la elaboración del trabajo. No puede exceder más de una página.

Resumen

El siguiente documento comprende el estudio y creación de un modelo híbrido entre Machine Learning y Algoritmos de tipo Lossless Data Compression. Internet crece cada día y los datos aumentan en volúmenes del orden de los Terabytes, por lo cual es de interés usar técnicas de compresión para realizar procesamiento a mayor escala de información con la menor cantidad de recursos posibles.

Hoy existe variados tipos de web, redes sociales, microbloging, web informativas, etc. El contenido proporcionado a los usuarios finales ya no es estático y esto permite que los mismos puedan generar, aportar y modificar contenido, dado esto la industria del desarrollo nos ofrece variadas alternativas para construir web estando siempre en constante evolución lo que ha ayudado a generar mas y mejores recursos. Muchas de estas nuevas tecnologías han permitido entregar una mejor experiencia al momento de navegar, aún cuando se ha desarrollado un gran avance sobre el mismo esto no ha permitido crear Web que sean por sí mismas inteligente y puedan ir anticipando su comportamiento, por ejemplo; disminuir la latencia desde que se abre una web ya visitada o desde que se navega dentro de un sitio con alta demanda; también desde el punto de vista de la arquitectura como servicio que las hospedan no se ha visto abarcada, dando un aspecto económico a los recursos utilizados. Si bien el crecimiento de los recursos de almacenamiento en la nube se encuentran en apogeo, las redes no crecen a la misma velocidad.

Este trabajo busca predecir la siguiente página que un usuario pueda acceder dentro de una web. Usando técnicas de entrenamiento y algoritmo de compresión. Con este propósito se trabajará para crear un modelo predictivo que use estas dos áreas.

Abstract

Abstract is the summary in english of the subject your are presenting in this thesis. Should not exceed one page.

Contenido

Lista de tablas

Lista de figuras

Capítulo 1

Introducción

Los nuevos avances tecnológicos y la inclusión de la ciencia de la computación en distintas campos han permitido hacer colecciones de datos muy grandes, las cuales se deben procesar, clasificar y analizar. Encontrar información útil dentro de estos grandes volúmenes de datos significa poder mejorar las decisiones teniendo como premisa la base de conocimientos históricos que se van constantemente almacenando.

La minería de datos basada en web access se ha convertido en una área importante de investigación en los últimos años. Se puede utilizar para mejorar el rendimiento de la caché web, la detección de intereses de los usuarios y así recomendar páginas o bienes relacionados para los sitios web de comercio electrónico, mejorar los resultados de los motores de búsqueda y por ejemplo personalizar el contenido de la web con las preferencias deseadas para el usuario. En la predicción de navegación web logrando entender el patrón de navegación del usuario y luego la predicción de las páginas siguientes es el principal problema que buscamos solucionar. Con un sistema de predicción fiable podemos ver la siguiente acción de los usuarios de Internet y tomar acciones.

Normalmente se ocupan muchos algoritmos de Machine Learning para poder hacer predicciones en variadas áreas. Hoy en día las aplicaciones en que los usuarios se enfrentan no pueden ser estáticas y sin tener un comportamiento que ayuden a predecir la forma en que actúan los usuarios. Esta área toma mucha relevancia al encontrarnos en un auge de la información

generada por usuarios, redes sociales y variadas plataformas. Podemos mencionar que ésta es una de las razones para poder disponer de herramientas para análisis y predicción que nos permitan saber cómo se comportan los usuarios dentro de una web, conocer la frecuencia en que se accede a un recurso en Internet, etc.

El momento en que un usuario entra a una web se establece una conexión cliente-servidor, una gran cantidad de servicios proporcionan datos de accesos de los usuarios que se encuentran activos en una *web*, sobre estos mismos *webaccess* y se pueden realizar variadas investigaciones sobre cómo predecir cuál es la siguiente página que podrá visitar.

Las predicciones en los registros de web access ha atraído una significativa atención de varios investigadores en los últimos años. Muchas técnicas de recuperación de datos y algunos sistemas de personalización usan algoritmos de predicción. La mayoría de las aplicaciones actuales que predicen la siguiente página web de un usuario tienen una componente en línea que hace la tarea de preparación de datos y una sección en línea que proporciona contenido personalizado a los usuarios en función de sus actividades de navegación actuales. En este trabajo se presenta un modelo de predicción en línea que se puede consumir como un servicio de API el cual da una integración ha variadas plataformas y sistemas que no tenga un componente en línea y con una buena precisión de la predicción. Nuestro algoritmo se basa en algoritmos LZ78 que están adaptados para modelar y representar la navegación secuencia del usuario en una web. Nuestro modelo disminuye la complejidad computacional y de implementación, que es un problema grave en el desarrollo de sistemas predictivos en línea.

1.1. Contexto Preliminar

La Web crece constantemente y por ende su infraestructura, también la información que podemos obtener de los usuarios y concurrencia de los sistemas, la cual para los usuarios finales se traduce en latencia y una mejor o peor experiencia de usuario. Paralelamente se suma un costo exponencial de recursos tanto en tecnologías de desarrollo como servicio que no son optimizados para poder dar una experiencia de usuario con calidad de servicio (QoS). Podemos reflexionar, entonces, que tener mayores recursos no mejorará el rendimiento, ni tampoco será lo óptimo para dar una calidad de servicio web ya que el ancho de banda de Internet no crecerá en la misma

proporción.

Adicionalmente, las tecnologías para la creación de web dinámicas y asíncronas han evolucionado a favor de traspasar la carga cliente. Hoy en día ya se posee lenguajes y framework que disminuyen considerablemente la carga de un servidor, por lo cual, un buen servicio web es proveer una balanceada carga dentro del cliente y el servidor, pero cuando se poseen un volumen de datos grandes es fundamental tomar decisiones que los recursos y lenguajes no cubren, es ahí el interés de dar inteligencia a los servicios web.

Predecir los futuros accesos que un usuario tendrá en una determinada web. Interpretaremos que la manera en que un usuario navega es su comportamiento registrado en una web, y que se puede analizar, estudiar y registrar mediante *Web Access Log*.

Sobre estos mismo se puede hacer representaciones eficientes [1] las de los registros y una minería de datos, Web Usage Mining (WUM), El por qué de hacer minería de datos es que cada día la web genera una innumerable cantidad de datos, por lo cual usar un algoritmos que se puedan operar comprimidos presenta un interés ya que además de disminuir el espacio físico o recursos utilizados, este se puede usar como un algoritmo de predicción y trabajar con una mayor cantidad de datos.

Tener conocimiento de la predicción de los *webaccess logs* de manera procesada o pre-procesada, ayudaría a ingenieros de desarrollo web y diseñadores de experiencia de usuarios, como también mejorar la experiencia de usuario final, disminuyendo por ejemplo la latencia en respuestas por parte de cada petición que realizan, con técnicas de *pre-fetching predictivo*.

Actualmente, las web no pueden ser simplemente dinámicas en contenidos, deben poseer una adaptabilidad a la demanda del usuario o proveer información que permita adaptarse a los eventos, por lo tanto, es de interés el hacer un estudio sobre ésto y poder hacer integraciones en áreas como compresión y maquinas de aprendizaje. Han sido abordadas independientemente para un problema en común que se puede resolver de manera eficiente.

Durante este trabajo se usarán técnicas de compresión de datos, se utilizará una infraestructura y patrón de implementación para modelos de Machine Learning. Adicionalmente toda la experimentación se llevará acabo ofreciendo los algoritmos y modelos como servicio REST (el cual se explicará más adelante) y así es posible implementarlo en áreas productivas las cuales pueden presentar interés.

Definiremos el concepto de sesión cuando el usuario se conecta a un servi-

cio web, estas pueden ser páginas informativas, redes sociales, generalmente web dinámicas, contenido colaborativo, etc. Este usuario establece una conexión directa con una página al momento de realizar esta operación. Dado esto es posible almacenar datos muy relevantes los cuales vamos a llamar "webaccess log" ó registros de accesos web, durante el texto se mantendrán las referencias en inglés.

Un ejemplo de *web access log* es el siguiente:

```
172.31.33.116 - - [26/Nov/2015:00:12:12 +0000] "HTTP/1.1" 200 1784 "http://localhost/home"
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
172.31.33.116 - - [26/Nov/2015:00:12:12 +0000] "HTTP/1.1" 200 179333 "http://localhost/news"
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
172.31.33.116 - - [26/Nov/2015:00:12:12 +0000] "HTTP/1.1" 200 24660 "http://localhost/health
     "
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
172.31.33.116 - - [26/Nov/2015:00:15:12 +0000] "HTTP/1.1" 200 24604 "http://localhost/sports
     "
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
172.31.33.116 - - [26/Nov/2015:00:20:12 +0000] "HTTP/1.1" 200 4860 "http://localhost/home"
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
172.31.33.116 - - [26/Nov/2015:00:22:19 +0000] "HTTP/1.1" 200 4841 "http://localhost/
     finances"
"Mozilla/5.0 (Linux; Android 5.1.1; SAMSUNG SM-G920I Build/LMY47X)
SamsungBrowser/3.2 Chrome/38.0.2125.102 Mobile Safari/537.36"
```

El ejemplo anterior nos da mucha información interesante como la IP desde donde se conecta, el tipo de navegador, el dispositivo si es un teléfono inteligente o un navegador de escritorio, la fecha en que se realizó el acceso y también lo más relevante el destino del usuario.

1.2. Definición del Problema

El problema de la Predicción, ha surgido hace años y diversos investigadores han trabajado con distintos enfoques. Rissman [2] y Langdom [3] en los laboratorios Bell al realizar pruebas y experimentar con un robot que tiraba una moneda compitiendo con humano, el robot realizaba todos los cálculos markovianos y cálculos de las probabilidades condicionales para que cierto evento ocurra, a diferencia del sujeto que sólo estaba esperando un resultado. Predecir no es trivial, pero sí podemos llegar a acercarnos y minimizar el error de equivocarnos. Sin embargo, dos áreas han tratado de resolver el problema; LDC y Machine Learning de manera separada. Por parte de LDC los mayores problemas son que los predictores funcionan totalmente desconectados y no dan una de disponibilidad inmediata de los resultados, en cambios en el área

de Machine Learning debemos crear un modelo para entrenar y luego poder generar una función predictiva.

Planteamos que el problema se podría resolver teniendo un modelo híbrido juntando los patrones de cada área y disponerlo como un servicio inmediato; dando una predictibilidad inmediata que hoy en la industria es necesaria para poder hacer útiles estos algoritmo y dar un valor a los avances.

1.3. Algoritmos como servicio web

Los avances en el desarrollo de nuevas tecnologías que brinden mejores experiencias en su uso día a día, deriva en cómo podamos llevar varios escenarios idealizados a implementaciones empresariales reales. Es bastante común encontrar librerías que son bastante útiles para hacer Minería de Datos, Clustering y muchas operaciones que pueden recurrir en cálculos muy complejos pero no se pueden ofrecer como servicio. Ya en pleno auge de las infraestructuras Cloud, la capacidad de cómputo que se puede alcanzar no es un problema como antes lo era para un Cientista de Datos.

Una API es un interfaz de programación de Aplicaciones que nos permiten intermediar el *Servicio A* con el *Servicio B*. Respectivamente A puede ser el proveedor y B el Demandante de servicios. Si quisiéramos analizar datos que se encuentran dentro de un servidor específico, estos se podrían consumir por esta interfaz. Existen variados clientes que nos permiten ayudar en esta comunicación, incluso se pueden utilizar por una terminal de Unix que es posible dialogar mediante el programa *curl*.

Ya se dispone de infraestructura como servicio ($IaaS$) , software como servicio ($Saas$), plataformas como servicios ($PaaS$). Dado lo anterior ofrecer estos algoritmos para hacer que las soluciones de desarrollo den valor agregado a la experiencia requerida por el usuario final. Por esto hemos decidido utilizar una librería y framework que nos de esta posibilidad. Ofrecer algoritmos a la industria como un servicio que ayuda de manera eficiente e Inteligente en un formato API, que permitirá una fácil integración.

Todas las ventajas de este patrón son heredados de las características que ofrece una API, interoperabilidad, evitar problemas de Infraestructura, Resiliencia de Datos, Persistencia de Datos, Análisis y Procesamiento sin afectar un curso operacional de una aplicación. Un ejemplo claro de esto es el análisis de datos en sistemas legados los cuales en plan de mejoras, no poseen la compatibilidad para poder realizarlo. Por otro lado, los algoritmos

de compresión o algoritmos de Machine Learning tienden a ocupar recursos y este hecho pueden ser la razón para no implementarlos.

1.4. Predicción

En esta sección se presenta formalmente el framework que se utilizará durante este trabajo. PredictionIO es un servidor de Machine Learning Open Source para Cientistas de Datos y Desarrolladores que permite crear motores de predicción para ambientes de producción, con un bajo tiempo de entrenamiento y despliegue en ambientes productivos. Principalmente está construido en Apache Spark, HBase y Spray.

Como ya se ha señalado este ambiente de trabajo se encuentra en un maduración completa que permite tanto disponer servidores con motores predictivos, como también toda una infraestructura distribuida para hacer que complejos algoritmos sean utilizados para solucionar problemas reales.

1.4.1. Arquitectura DASE

Un motor de predicción es un tipo de proceso en Machine Learning. Siguiendo una arquitectura de tipo DASE, contendríamos los siguientes componentes.

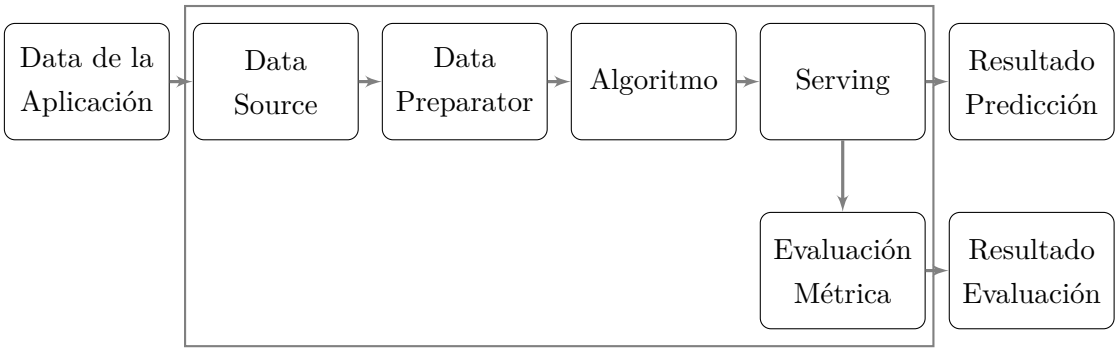
- [*D*] **Data Source y Data Preparator**
  Los Data Source leen la data desde la entrada original y la transforman en un formato deseado para hacer análisis de estos. En cambio *Data Preparator* pre-procesa la información y la reenvía a los algoritmos para hacer el modelo de entrenamiento.

- [*A*] **Algoritmo**
  Los componentes de algoritmos incluyen algoritmos de Machine Learning, estos, por los componentes que son de predicitionIO pueden ser provistos por Apache Spark o se pueden incluir algoritmos propios como también de terceros. Adicionalmente a los algoritmos podemos asignarle parámetros, para determinar como debiese ser construido el modelo predictivo.

- [*S*] **Servicio**
  El componente servicio toma las consultas ó *querys* de predicción y

retorna los resultados, en nuestro modelo propuesto en la etapa experimental veremos el siguiente símbolo de una secuencia. Si el motor de predicción tiene múltiples algoritmos, combinará los resultados en uno. Adicionalmente, lógica específica de negocios puede ser añadida para especificar aún más el resultado final.

- $[E]$ **Evaluación de Métricas**
  Las métricas de evaluación cuantifican la precisión de la predicción con una puntuación numérica. Puede ser utilizado para la comparación de algoritmos o ajustes de los parámetros del algoritmo.



PredictionIO ayuda a tener componentes modulares de fácil uso, que hemos descrito para que se puedan construir modelos de predicción de manera más sencilla, también poder integrarlos con gran facilidad a cualquier sistema o plataforma, por ejemplo, es posible elegir cual de todos los componentes se podrá desplegar al momento de crear un *Engine* (Motor de Predicción.)

Despliegue de Engine

Un Engine pone todos los componentes *text*DASE en un estado especifico de despliegue

1. Data Source
2. Data Preparator
3. Uno o más Algoritmos
4. Un Servicio

Si se especifica más de un algoritmo, cada uno de los resultados de los modelos de predicción se entregará para ser consumido por cualquier cliente. Cada *Engine* procesa los datos y construye modelos predictivos de forma independiente. Por lo tanto, todos los Engine sirven a su propio conjunto de resultados de predicción. Por ejemplo, puede desplegar dos Engine para su aplicación móvil: uno para recomendar noticias para los usuarios y otro para sugerir nuevos amigos a los usuarios.

Evaluación del Engine

Para evaluar el Accurracy de un Engine, solo se debe especificar la métrica cuando se corre el motor de evaluación, en los capítulos experimentales usted podrá ver como se generan estás métricas y como se usa este motor de predicción.

1.4.2. Modelamiento de Eventos

Modelamiento de Eventos

El modelamiento de eventos, es simplemente el hecho de poder llevar un feature[1] que es del mundo del ML, es en realidad una representación de como se debe tener la data de manera RDD para poder acceder a ella posteriormente.

Un evento lo definiremos como entidad que nos permite dar una representación temporalizada de información que será procesada por un motor de predicción. Analizaremos los eventos que un usuarios realiza para poder acceder a una web. Adicionalmente cuando cada usuario ingresa a una web automáticamente este genera una sesión, desde que que llega hasta que abandona la web.

Usaremos un *Dataset* con información que esta totalmente depurada y recuperada de los access log(Provista por Claude *et al.* [1]), los cuales a efectos de temporalidad nos interesa conocer la secuencialidad discreta de estos accesos.

El modelamiento que se realizará contempla que el usuario :

- Tipo de Evento: Visitar
- Entidad que ejecuta el evento: Usuario

[1] Característica de un cierto dataset para entrenar.

- Propiedades:

    1. Página actual
    2. Página siguiente
    3. Cierre de Sesión

El interés de tener un modelo totalmente atómico es poder contemplar la información que nos entrega, destacando sus variables y propiedades como restricciones.

1.4.3. Ventajas

Es posible mezclar y aplicar distintas característica si el modelo no puede ser ser persistido por PredictionIO automáticamente. Se requiere un objeto acompañante heredado de una clase que permite lograr la persistencia en memoria (IPersistentModelLoader), esto permite PredictionIO cargar el modelo persistentemente y automáticamente durante la implementación.

Comprendiendo el concepto de RDD, esta es la abstracción básica de Spark, aún más, esto es una de las grandes cualidades de PredictionIO, ya que no solamente podemos disponer de una máquina para hacer estudios o implementar algoritmos, este servidor de Machine Learning permite, gracias a sus componentes poder hacer un cluster para entregar mayor eficiencia acorde a los datos o algoritmo a implementar.

Ya hemos mencionado que un *Resilient Distributed Dataset* (RDD) es una representación inmutable, una colección particionada de elementos que pueden ser operadas en paralelo. Internamente cada RDD tiene cinco principales propiedades:

1. Una lista de particiones.
2. Una función para procesar cada split de datos.
3. Una lista de dependencias en otros RDD's.
4. Opcionalmente una partición de un RDD puede ser representada como $\{llave, valor\}$
5. Opcionalmente, una lista de los lugares preferidos para calcular cada una dividida en (por ejemplo, lugares de bloque para un archivo HDFS), para procesamiento en Clustering.

1.5. Descripción del Contenido

Este trabajo esta organizado de la siguiente manera: En el capitulo 1 describiremos el contexto preliminar y definermos el problema de la predicción de secuencias discretas para webaccess log. Veremos como poder dar una inducción a un framework llamada *PredictionIO* el cual nos ayudará a entregar los algoritmos como servicios consumibles por cualquier aplicación cliente que pueda comunicarse con un servidor. En capitulo 2 explicaremos todos los conceptos básicos para el entendimiento sobre esta investigación, como también conceptos para el uso de *PredictionIO* y cerraremos con todos los trabajos reaccionados mas recientes que involucran nuestra investigación de interés. En capitulo 3 veremos las predicciones sobre webaccess log, los modelos propuestos por varios investigadores y sus limitaciones, daremos una revisión del trabajo realizado por *Rissanen* [4] que da el inicio a esta área de Investigación. En el capitulo 4 veremos los temas de *Machine Learning* y *Lossless Compression Data* y como pretendemos crear un modelo de predicción con recursos de ambas áreas. Para cerrar este capitulo explicaremos el uso de del algoritmo Lempel & Ziv, el uso para secuencias discretas su convergencia a un modelo de predicción eficiente y exacto.

Finalmente el capitulo 5, presentará nuestros experimentos realizados sobre la implementación de un algoritmo de compresión en un servidor de *Machine Learning* como es *PredictionIO*, analizaremos el comportamiento del algoritmo y como se desempeña en este ambiente, propondremos discusiones de como mejorar nuestra implementación y los trabajos a futuro que puede presentar esta investigación.

Se deja como anexo una guía básica de uso para *PredictionIO*, todos los datos y nuestra implementación se puede encontrar en nuestro repositorio git público [2], en donde usted encontrará todos los datos para replicar las experiencias experimentales que hemos realizado.

[2] `https://github.com/jaimeguzman/PredictionIO-LZmodel`

Capítulo 2

Conceptos Básicos

En este capitulo se introducirá los conceptos principales que se trabajarán en los siguientes capítulos:

2.0.1. Access Log

Son los registros que se almacenan en un servidor, los cuales dependiendo del sistema operativo pueden tener mayor o menor información. Cuando los usuarios acceden a los sitios web, estos suelen dejar una gran cantidad de información de acceso, la cual si es extraída en forma razonable puede ayudar a los administradores de sitio web para obtener acceso a los patrones de los usuarios.

2.0.2. Árboles Trie

Son estructuras de datos en forma de árbol que almacenan datos en nodos y muy fácil la recuperación de información de estos. Se caracterizan por ser un conjunto de llaves que se representan en el árbol y sus nodos internos representan la información, en nuestro caso un carácter o *String* de largo 1.

Un árbol es una estructura general de nodos recursivos. Hay muchos tipos de árboles. Los populares son árbol binario y el árbol de equilibrado. Un Trie es una especie de árbol, conocido por muchos nombres incluyendo árbol prefijo, árbol de búsqueda digital, árbol de la recuperación (de ahí el nombre de ”trie”).

Cada especie de árbol tiene distinta finalidad, estructura y comportamiento. Por ejemplo, un árbol binario almacena una colección de elementos comparables (por ejemplo, números). Por lo tanto, se puede utilizar para almacenar un conjunto de números, o al índice de otros datos que pueden ser representados por los números (por ejemplo, objetos que pueden ser hash). Su estructura está ordenada por lo que se puede buscar rápidamente para encontrar nodo. Otras estructuras de árbol, como un árbol balanceado son similares en principio al trie que se implementará en la etapa experimental.

Un *trie* representa una forma estructurada de nodos, la cual puede almacenar secuencias. Es muy diferente cuando almacena secuencias de valores en lugar de valores individuales. Cada nivel representa un incremento en la altura del árbol.

Durante este trabajo mostraremos que nuestro modelo de predicción usa un *Trie*, para representar un diccionario, en los cuales podemos señalar las siguientes operaciones disponibles:

- **findByPrefix(x) :** Retornar una una lista de todos los nodos hasta llegar a un nodo que posea un *String* de parámetro equivalente a x.
- **contains(x) :** Retornar una lista de nodos intermedios hasta que el contenido del nodo final sea hoja o el intermedio corresponda a valor de la secuencia *String* x.
- **remove(x) :** Retorna *true* ó *false* cuando es posible remover el nodo con el contenido equivalente a x.
- **pathTo(x) :** Retorna una lista de nodos hasta llegar a un nodo que posea el contenido equivalente del valor x.

2.0.3. Alfabeto

Un alfabeto discreto lo representaremos por Σ y una de las características básica de este alfabeto es consiste en símbolos σ que pertenecen a Σ, durante el resto de nuestra investigación usaremos solo un $\Sigma = \sigma_i, \sigma_{i+1} = 17$ símbolos.

Dado un volumen de datos experimental, nuestro alfabeto es mostrado simbólicamente como la representación de varios nodos contenido en un *Trie* que modela la navegación de usuarios para un sitio web. Donde σ_1, puede ser definido como la página inical de una cierta secuencia o sesión de usuario. Este alfabeto es finito y acotado por una minería de datos ya realizada por [1] en *Efficient Indexing and Representation of Web Access Logs.*

2.0.4. Secuencias discretas

Definimos una secuencia de accesos discreta y finita, dado los accesos que tiene un usuario frente a una web, lo anterior es acotado por el concepto de sesión, el cual es desde que se inicia la navegación por parte de un usuario, es decir secuencia de tamaño $Seq \leq 1$ y de tamaño no superior a un alfabeto Σ.

2.0.5. Lossless Data Compression(LDC)

La compresión sin pérdida o LDC, es el arte de poder comprimir bits y poder hacer el proceso inverso, es decir poder codificar y decodificar. En capítulos posteriores se detallará más sobre el tema compresión y como esta área ayuda a crear un modelo de predicción.

2.0.6. Motor de Predicción

Es la parte fundamental de un sistema dirigida a adivinar el futuro acceso de un usuario. La salida del motor de predicción es la siguiente página, que se compone de un símbolo representando la dirección *url* o una sección en particular de una cierta web.

2.0.7. Resilient Distributed Datasets

Los RDD, permiten que en un servidor de Machine Learning pueda mantener un modelo o motor de aprendizaje persistente sin importar en el flujo que se encuentre.

Esta estructura es fundamental dentro de la librería que se introducirá mas adelante, Apache Spark. Esta estructura es una colección distribuida de objetos inmutables, cada set de datos en un RDD es divido en particiones lógicas, las cuales puedes ser computadas en distintos Clusters. Los RDD pueden contener cualquier tipo de objeto de los siguientes lenguajes: Python, Scala y Java, incluyendo clases definidas por el usuario.

Formalmente los RDD son solo de lectura, una colección de objetos divididos. Estos pueden ser creados a través de operaciones deterministas en una cierta tabla o un almacenamiento externo u otra RDD. Otra de las características de los RDD, es que son colecciones de elementos tolerantes a fallas que pueden ser operadas en si mismas o en paralelo. Apache Spark hace el uso del concepto de RDD para lograr rapidez y eficiencia en las operaciones

de MapReduce, de ser requeridas. Destacamos la escabilidad de esta librería para un gran nivel de cómputo, pero en este trabajo no se explicará el uso de Spark, pero si se utilizarán algunos conceptos como RDD y otros.

2.0.8. Data Source y Dataset

Ambos conceptos están enfocados a proveer información tanto para el servidor de Machine Learning, como para el procesamiento y análisis. En este trabajo el dataset con que realizaremos nuestro estudio son los registros de accesos de la web española *msnbc* [1], los cuales representan 1.000.000 de registros correspondientes.

Nuestro set de datos esta basado en los registros (webaccess log), ya previamente depurados con una representación numérica desde 0 hasta 17, el cual como antes ya se ha señalado será nuestro alfabeto. También crearemos dataset sintéticos para poder realizar depuraciones de nuestra implementación y casos de bordes.

2.1. Trabajos Relacionados

En la literatura, el tema de la predicción en la web se ha presentado como un tema recurrente provocando bastante atención durante los último años y ha sido tratado por varios autores. Tenemos los siguientes trabajos de interés:

1. **Dynamic and memory efficient web page prediction model using LZ78 and LZW algorithms**

   Moghaddam y Kabir [5] realizan una comparación de LZ78 y el algoritmo LZW, el cual es una derivación del anterior. La mayoría de las aplicaciones actuales que predicen el siguiente acceso a un página web posee un componente offline que hace la tarea de preparar data y luego disponer una sección en línea que permite personalizar cierto contenido para un usuario en particular basado en las actividades de navegación.

   En la mayoría de las técnicas de *Web Usage Mining*, las secuencias se utilizan, ya sea para producir las reglas de asociación o para producir estructuras de datos de tipo árbol o cadenas de Markov para represen-

tar patrones de navegación. Los Modelos de Markov, se basan en una teoría bien establecida y son fáciles de entender.

La propuesta es no crear un modelo predictivo por usuarios. Moghaddam y Kabir proponen modelar la navegación de usuarios mediante un Trie creado por un algoritmo de la familia LZ y usando muchas sesiones de usuarios, para tener un modelo predictivo de navegación.

2. **Prediction Algorithms for User Actions** (Hartmann & Schreiber,2007 *et al.* [6]) ,

   Se requiere la predicción de la siguiente acción del usuario basada en la historia de interacción que ha tenido con una interfaz. En su trabajo dan una revisión a los algoritmos de predicción Discreta (SPA) y desarrollan dos propuestas de algoritmos basadas en Modelos de Markov que combinan distintos ordenes de Markov. Y desarrollan una librería en PERL para su propuesta y evaluación.

3. **ActiveLezi** (Gopalratnam & Cook,2007 *et al.* [7]) Proponen un Algoritmo *On-Demand* que considera varios modelos de Markov. El funcionamiento es basado en almacenar la frecuencia del patrón de *input* en un *Trie* acorde al algoritmo de compresión de *LZ78* para superar algunos de los problemas que surgen con *LZ78*, se usa una ventana de largo variable de los símbolos previamente usados en la construcción del *Trie*. El tamaño de la ventana crece con el número de las diferentes subsecuencias que se van viendo en la entrada de cada secuencia nueva que ingresa. Sea $suff_l$ el sufijo de largo $l+1$ el sufijo de longitud $l=1$ de las inmediatamente historial de interacción.

4. Dongshan y Junyi [8] Destacan que un modelo de Markov puede ayudar a predecir el comportamiento de un usuario, pero con ciertas limitaciones. Para solucionarlo presentan un nuevo modelo de Markov basado en una representación de *Tree Order Model*, el cual es un híbrido entre un modelo de Markov tradicional y una representación de árbol, bautizada como HTMM (por sus siglas en inglés, *Hybrid-Order Tree Markov Model*). Su modelo fue presentado en 2002, y es relevante conocer la predicción de los *web access*, dada la importancia de creación de redes, la minería de datos, e-commerce, y otras áreas.

5. Domenech *et al.* [9]

Muestran un estudio de los rendimientos de técnicas de recuperación de datos. Las mismas se pueden utilizar para dar una entrada ideal a algoritmos de aprendizaje o algoritmos de predicción. Los conceptos más importantes son las nuevas variables de caracterización, temporalidad, espacio y geografía, que se le suman a la predicción. Además de comenzar un trabajo más elaborado de como tomar una predicción, se introducen conceptos como predicciones genéricas o específicas, variables de uso de recursos a nivel de red o nivel de procesamiento. Finalmente, se presenta un modelo predictivo que puede ayudar a disminuir la latencia entre la petición del cliente y la respuesta de la web, dando así un mejor rendimiento y *QoS*.

6. Chen *et al*. [10]

   Dan una nueva perspectiva enfocada a entregar una clara recomendación a los usuarios basada en la misma propuesta de este proyecto, los access log. El primer análisis realizado por los autores cubre las reglas asociativas que requiere un sistema de recomendación, pero en las pruebas propiamente tales encuentran que el análisis de los patrones detectadados dan una representación clara de como optimizar la web, y finalmente mediante sus pruebas logran una recomendación de calidad.

7. Rajimol y Raju [11] Minaron los patrones de los accesos web, donde el enfoque es usar los registros de acceso para crear subsecuencias y realizar comparaciones. La literatura existente es relevante para poder anticipar el patrón de comportamiento de la web.

8. Kewen [12]

   Realizó un análisis más profundo del *web usage minning*. Parte de la importancia de este trabajo, es que después de minar los registros de accesos, logran reducir la "*bad data*".

9. Poornalatha y Raghavendra [13]

   Establecen que se pueden utilizar máquinas de aprendizaje para predecir basándose en métricas de distancia entre distintos clusters. Estos autores, al igual que Domenech *et al*. [9] y Dongshan y Junyi [8], comparan el objetivo de optimizar los recursos tanto en redes (disminución de latencia) y experiencia de usuario.

10. Claude *et al*. [1]

    presentan una estructura de representación eficiente que permite dar una representación de *web access log* y ofrecen las operaciones básicas de WUM.

Capítulo 3

Predicciones sobre Web Access

Dado que en los nuevos sistemas es cada vez más común conocer sistemas inteligentes y diversos en variadas áreas, sistemas que tienen cualidades como tener la posibilidad de predecir ocurrencia de eventos para así adaptarse y tener versatilidad al tomar decisiones en variadas situaciones. Aún mas necesaria es esta propiedad en problemas que requieren predicciones secuenciales, es decir, dada una secuencia de eventos, predecir el siguiente evento basado en nuestro conocimiento histórico limitado. En los últimos años se ha observado que los contenido de muchos sitios web son dinámicos y nuevas páginas también se añaden al sitio de forma dinámica. Así es como se hace necesario un modelo predictivo que pueda servir como un modelo predictivo online, que considere tanto los cambios web que se van produciendo, como también el comportamiento de los usuarios que interactúan con ella. Moghaddam *et al.* [5] proponen un modelo predictivo online que cubre la eficiencia de la memoria como un factor importante para un algoritmo en línea.

Para cualquier secuencia de eventos, estas se pueden modelar como procesos estocásticos, estos algoritmos emplean Modelos de Markov para optimizar las predicciones del siguiente símbolo, en cualquier secuencia estocástica. Otros escenarios requieren que un algoritmo de predicción sea capaz de incrementar su recuperación de información y poder dar resultados de forma inmediata, es decir predicciones *online* .

El problema de la predicción secuencial se puede establecer de la siguien-

te manera, dado un secuencia de símbolos $x_1, x_2.....x_n$, ¿Cuál es el siguiente símbolo x_{i+1}? Este problema ha sido formulado gracias al trabajo de Risseman [2] y Langdom [3]. un buen compresor es aquel que al comprimir datos sin pérdida se aproxima a un *predictor*.

Para poder crear un modelo predictivo acorde a la Teoría de la Información, un predictor que construye un modelo cuya entropía se aproxima a la de la fuente de datos, consigue una mayor precisión predictiva.

Las predicciones son un área importante dentro del dominio de las Machine Learning y la Inteligencia Artificial, las cuales pueden ofrecer un sistema de inteligencia que las aplicaciones necesitan para un óptimo desempeño, también ayudan a dar información para la toma de decisiones. Ciertos dominios requieren que la predicción se pueda realizar en las secuencias de eventos que, por lo general, se pueden modelar como un proceso estocástico. Nuestro interés se centra en las predicciones de secuencias discretas, y en este punto, demostrar la convergencia en la cual un modelo de compresión (como el caso de LZ78 que se explicará más detalladamente en el Capítulo 4) y la eficiencia de un algoritmo de compresión, ofrece una nueva perspectiva a las predicciones.

Nos enfocaremos en el caso de los acceso que un usuario realiza a un sitio web, el tiempo en el que pasa en este es registrado por el servidor, en esta investigación no se requiere indagar en temas de *Information Retrieval*, ya que se entrega una colección de datos ya procesada, la cual es representada por las secuencias de acceso por parte de usuarios. Dicho de otro modo, dada una secuencia de acceso por un usuario que entra a la web, sus accesos web determinarán cual será la predicción de su sesión. Dentro de los registros existen respaldos de sesiones de usuarios, páginas no encontradas, accesos denegados y otra información en relación al funcionamiento. Sin importar el tipo de servidor que utilicemos podemos identificar a usuarios con distintas técnicas y/o combinaciones, por ejemplo, podemos interpretar que el *request* que ha realizado el usuario más su sesión, nos da la información necesaria para poder crear un modelo predictivo.

Dado lo anterior podemos hacer minería de los datos que un servidor ha recolectado durante un periodo de tiempo, de lo anterior podemos mencionar que existen varias técnicas para poder hacer *Web Access Pattern* y también "Web Usage Minning", de sus siglas en inglés es minería del uso de una *web*. Nuestro interés no es abordar este tema pero si explicarlo para poder realizar un estudio predictivo de secuencias de acceso, con el objetivo de

poder predecir el siguiente comportamiento que tiene un usuario al momento de visitar un sitio web.

3.1. Predictores de Estado finito

Sea Σ un alfabeto finito. Para poder entrenar un secuencia $q_1^n = q_1 q_2 \dots q_n$, donde $q_i \in \Sigma$ y $q_i q_{i+1}$ es la concatenación de los símbolos q_i y q_{i+1}. Dado lo anterior el objetivo es poder entrenar un modelo M que entregue como resultado la probabilidad respectiva de cualquier futuro símbolo dado algún pasado. Específicamente, para cualquier contexto de secuencia $s \in \Sigma^*$ y un símbolo $\sigma \in \Sigma$, el aprendizaje de la secuencia dado por el entrenamiento debe dar una distribución de probabilidad $M(\sigma|s)$. El rendimiento del modelo predictivo se puede medir mediante una función del promedio de registro de errores $L(M, x_1^T)$ de $M(\cdot|\cdot)$, con respecto a una secuencia $s = x_1^T$ con $x_1^T = x_1, x_2, ..., x_n$por lo tanto podemos definir L como

$$L(M, x_1^T) = -\frac{1}{T}\sum_{i=1}^{T} \log M(x_i|x_i \cdots x_{i-1}) \tag{3.1}$$

, donde el logaritmo es en base 2.

El promedio de L es directamente relacionado a $M(x_1^R) = \prod_{i=1}^{T} M(x_i|x_i \cdots x_{i-1})$ y minimizar el promedio de L es completamente equivalente a maximizar la asignación de probabilidades a una secuencia de pruebas $x_1^T = x_1, x_2, ..., x_n$, teniendo en cuenta que esta equivalencia es totalmente válida. Sea $M(x_1^T)$ una asignación de probabilidad consistente para una secuencia completa, la cual satisface que

$$M(x_1^{t-1}) = \sum_{x_t \in \Sigma}^{T} M(x_1 \cdots x_{t-1} x_t) \tag{3.2}$$

, para todo $t = 1, ..., T$, induce la asignación de probabilidad,

$$M(x_t|x_1^{t-1}) = M(x_1^t)/M(x_1^{t1}),\ t = 1, ..., T \tag{3.3}$$

.

El registro de pérdida L tiene variadas interpretaciones. Tal vez la más importante se encuentra en su equivalencia a la compresión sin pérdidas LDC. La cantidad $-\log M(x_i|x_1 \cdots x_{i-1})$, que también se llama la $auto-información$, puede ser la de compresión ideal o "largo de secuencia"de x_i, en

bits por símbolo, con respecto a la distribución de probabilidad condicional

$$M(X|x_1 \cdots x_{i-1}) \tag{3.4}$$

, esta puede ser implementada *online* (con pequeña redundancia arbitraria) usando codificación aritmética (Rissanen y Langdon, 1979) [14] .

Por lo tanto, el promedio de L también mide la tasa de compresión media de una secuencia de prueba, cuando se utilizan las predicciones generadas por M, es decir, un bajo promedio de L sobre la secuencia x_1^T puede implicar una buena compresión de esta secuencia [15].

Suponiendo que los entrenamientos y las secuencias de pruebas fueron generados de una fuente desconocida[1] P. Sea una secuencia dada por valores aleatorios $X_1^T = X_1 \cdots X_T$, podemos decir que claramente la distribución P minimiza unicamente *log-loss* o como la hemos llamado anteriormente L, lo cual es

$$P = arg\ min_M\{-E_P\{\log M(X_1^T)\}\} \tag{3.5}$$

Dada la equivalencia de *log-loss* y la compresión, como se ha visto anteriomente, el significado de *log-loss* de P logra la mejor compresión posible, o logra una entropía

$$H_T(P) = -E \log P(X_1^T) \tag{3.6}$$

. Aún no conociendo realmente cual es la distribución de probabilidad de P, un entrenamiento genera una aproximación a M usando una secuencia de entrenamiento. La pérdida extra que podemos obtener la llamaremos *Redundancia* y esta dada por el valor de

$$D_T(P||M) = E_P\{-\log M(X_1^T - (-\log P(X_{1^T})))\} \tag{3.7}$$

.

Para normalizar la *redundancia* $D_T(P||M)/T$, de una secuencia de largo T, da los *bits* extra por símbolo (sobre la tasa de la entropía) al comprimir una secuencia utilizando P.

Este ajuste probabilístico motiva un objetivo deseable, al entregar un algoritmo de propósito general para la predicción: minimizar la redundancia de manera uniforme, con respecto a todas las posibles distribuciones.

[1] Mas adelante usaremos el término en ingles Data Source, para referirnos a fuentes de datos , tanto conocidas como desconocidas.

Un algoritmo de predicción el cual pueda acotar la redundancia de manera uniforme, con respecto a todas las distribuciones dada una clase.

Una cota inferior de la redundancia para cualquier *Predictor Universal* y *Compresor Universal*

$$\Omega(K\frac{\log T}{2T}) \tag{3.8}$$

, donde K es (más o menos) el número de parámetros del modelo que codifica la distribución P (Rissanen [4], 1984).

Si llamamos al siguiente símbolo b_t, diremos que el resultado de nuestro predictor es entregar este valor. Dado esto, existe una función de pérdida asociada $L(b_t, x_t)$ para cada predicción realizada.

El objetivo de cada predictor es tener una función de minimización tal que minimize la fracción de predicciones erróneas, a lo anterior lo llamaremos T.

3.2. Modelos tradicionales

En secciones anteriores se ha hablado de como los eventos secuenciales se pueden modelar como un proceso estocástico.

Ciertamente para reconocer frecuencia en patrones secuenciales y reglas de asociación puede requerir mucha información y entregar mejor precisión en la información de los usuarios y su comportamiento en la web. Pero un enfoque en que la temporalidad de los datos se debe modelar con un modelo de Markov es la clave para poder realizar esto, y estos datos ya han ayudado a predecir los accesos de los usuarios, como señala Dongshan *et al.* [8] pero en la práctica existen muchas limitaciones técnicas que no permiten que se implementen.

3.2.1. Limitaciones de los modelos tradicionales de Markov

Los modelos tradicionales de Markov predicen la siguiente página Web que un usuario puede acceder considerando el acceso más probable, se itera para coincidir su secuencia de acceso actual con secuencias de accesos Web históricas.

Usando estos modelos los investigadores como Dongshan *et al.* [8] han comparado el máximo de elementos prefijos de cada secuencia histórica de webaccess, con los elementos sufijos de la misma longitud de secuencia de

webaccess actual del usuario y obteniendo secuencias dado su secuencia histórica con la probabilidad más alta en la cual los elementos coinciden.

El modelo de Markov de orden cero es la tasa base de probabilidad incondicional, la cual es la probabilidad de la página visitada, dada por

$$p(x_n) = Pr(X_n), \tag{3.9}$$

donde x_n es xn y Xn es otra Xn.

El modelo de Markov de orden uno observa la probabilidad de la transición de un página a otra, la podemos interpretar así:

$$p(x_2|x_1) = Pr(X_2 = x_2|X_1 = x_1) \tag{3.10}$$

El K-ésimo orden del Modelo de Markov considera la probabilidad condicional que un usuario cambie a una nueva n-ésima página dado su anterior página visitada, teniendo que $k = n - 1$ páginas vistas:

$$p(x_n|x_{n-1}, ..., x_{n-k}) = Pr(Xn = x_n|X_{n-1} = x_{n-1}, ..., X_{n-k} = x_{n-k}) \tag{3.11}$$

Modelos de Markov, en (3.11), de orden inferior no pueden predecir con éxito total el futuro de los webaccess log, ya que no van lo suficientemente atrás en el pasado para discriminar el modo en que se comporta el usuario. Este comportamiento de los usuarios requiere de buenas predicciones, las que a la vez requieren modelos de Markov de orden superior, pero los modelos de orden superior resultan de alta complejidad en espacio de estado y cobertura.

Modelos con mayor orden de estados son distintas combinaciones de las acciones observadas en un set de datos, entonces, el número de estados tiende a crecer exponencialmente al igual que el orden del modelo. Este aumento puede limitar significativamente la aplicabilidad de los modelos de Markov para aplicaciones en las que las predicciones rápidas son críticas para el rendimiento en tiempo real o para aplicaciones con restricciones de uso de memoria. Además, muchos ejemplos en los test podrían no tener estados correspondientes en los modelos de Markov de mayor orden, por lo que reduciría su alcance.

Capítulo 4

Compresión y Machine Learning

El uso de algoritmos de compresión en tareas de Machine Learning como clusterización y clasificación ha tenido presencia en variados campos de las ciencias de la computación. La intención de reducir problemas de selección explícita de ciertas características que se usan en estudios y algoritmos de Machine Learning. Un punto de vista de esta inclusión de áreas muestra que los algoritmos de compresión mapean implícitamente string en representaciones vectoriales de dichas características, las cuales a su vez son cotas superiores. Podemos señalar que trabajos como los de Langdon y Rissman han sido claves para determinar la comprensibilidad de un algoritmo y la predictivilidad que poseen los modelos de compresión. Con estos modelos se pueden realizar predicciones muy interesantes en el área de Machine Learning.

Desde otro punto de vista, los algoritmos de compresión mapean implícitamente strings en espacios vectoriales implícitos por features de un cierto dominio, y debido a las propiedades que usa la compresión, podemos usarlos para los dominios de predicción y aprendizaje de ciertas características que nos entrega un entrenamiento típico de algoritmos de Machine Learning.

Esta idea de usar algoritmos de compresión en máquinas de aprendizaje no es nueva, pero no ha sido mayormente explorada. Los algoritmos de compresión han sido estudiados e investigados durante varios años, la motivación fundamental es poder optimizar el espacio generado por ambas áreas, para un uso eficiente o mayor almacenamiento de datos. Estos algoritmos se

encuentran, sin saberlo, en nuestro día a día, desde el núcleo de un sistema operativo como Linux hasta por ejemplo los formatos *zip*, *rar*, *7z*, también en formatos de imágenes y audios, etc. los cuales son útiles para poder optimizar una transferencia de archivos de un equipo a otro mediante Internet o simplemente comprimir datos para respaldar, por ejemplo, en dispositivos físicos.

La motivación de profundizar en el área de compresión de datos se debe a una de las razones mencionadas con anterioridad, Internet. Esta red de redes, constantemente crea nuevos contenidos, registros, imágenes, entre otros tipos de datos, los cuales sin tener buenos recursos de red y/o Infraestructura para cargar de un lugar a otro mediante una transferencia directa implicaría un demora exponencial en un escenario típico de una web. Dichos archivos crecen innumerablemente(mayormente generados por usuarios) y he aquí uno de los mayores aportes que poseen los algoritmos de compresión con relación a nuestra red de redes, precisamente la compresión de datos optimiza la transferencia de archivos como la carga de archivos en el lado del cliente.

A diferencia de la velocidad de conexión las infraestructura de redes crece proporcional más rápidos que las transferencias de volúmenes de datos, esto genera un sin fin de problemas para los usuarios e industria web. La Latencia es el tiempo de respuesta que demora un usuario en solicitar, hacer un *request*, a un servidor, simplemente es un el tiempo de respuesta desde iniciada una acción demandada. Uno de los grandes ejemplos que tenemos en la web es la proliferación de archivos comprimidos para su descarga, los cuales en su interior poseen variados recursos de tipo, audio, video, imágen, texto, etc.

Las propiedades de estos algoritmos no solo permiten juntar un set de archivos y lograr un tasa de compresión óptima para ser transmitida por Internet, también pueden ayudar a realizar análisis en grandes volúmenes de información, por ejemplo; el análisis de texto, clasificación de proteínas, moderación de contenidos en web y predicciones del comportamiento de usuarios que navegan en un sitio de Internet. Sobre este último punto es nuestro mayor interés, las predicciones de la siguientes webaccess de una determinada web. Existen varios prominentes algoritmos de compresión que se pueden usar para realizar modelos predictivos, nos enfocaremos en la familia Lempel Ziv. Estos son en sí modelos variables de Markov lo cual nos ayudará en nuestra etapa experimental a dar un modelamiento secuencial de la navegación de un usuario como también crear funciones de predicciones

en base a la probabilidad de ver cada nodo dado a su frecuencia.

Para introducir el camino se debe presentar formalmente los algoritmos de compresión y su clasificación más general. Entre ellos tenemos los algoritmos con pérdida y sin pérdida, nos enfocaremos en los algoritmos *Lossless Compression Algorithm*, algoritmos de compresión sin pérdida.

El aprender acerca de la secuencia de datos continuas sigue siendo un ítem fundamental y un desafío en patrones de reconocimiento y aprendizaje automático.

4.1. Modelos de Compresión

Existen muchos modelos y algoritmos de compresión, nuestro enfoque es usar los algoritmos de compresión que tengan un espacio vectorial de características conjunto con Machine Learning y además tengan propiedades para ser candidatos a un predictor.

4.1.1. Prediction by Partial Match (PPM)

El algoritmo de predicción por certeza parcial es considerado uno de los mejores algoritmos del tipo *Lossless Compression Algoriths*. El algoritmo requiere un tope superior D en el maximo del orden de Markov de un modelo variable de Markov (VMM) para construirse. PPM maneja el problema de frecuencia cero usando dos mecanismo llamados

- Escape
- Exclusion

Para un método que considera diferentes órdenes de modelos, retomamos una vez más a la compresión de datos y la familia de predictores PPM (por sus siglas en inglés de Predicción Parcial de Partido). Esto ha sido usado con gran efecto, para un marco predictivo basado en LZ78.

Algoritmos PPM consideran modelos de Markov de diferente orden, con el fin de construir una distribución de probabilidad mediante la ponderación de modelos de diferente orden. En nuestro escenario predictivo, Active LeZi

construye un orden-k del modelo de Markov. Ahora empleamos la estrategia PPM de exclusión para reunir información de los modelos de orden 1 a k para asignar el siguiente símbolo de su valor de probabilidad. Este método se ilustra considerando la secuencia de ejemplo utilizado en los apartados anteriores: "aaababbbbbaabccddcbaaaa".

La ventana mantenida por Active Lezi representa el conjunto de contextos utilizado para calcular la probabilidad del siguiente símbolo. En nuestro ejemplo, el último se utiliza la frase "AAA". Dentro de esta frase, los contextos que pueden ser utilizados son todos sufijos dentro de la frase, excepto la ventana en sí (es decir, *aa* , *a*, y el contexto nulo).

4.1.2. Probabilistic Suffix Tree (PST)

Los arboles de sufijos implementados como un algoritmo de predicción intentan construir el único y mejor *VMM* con límite superior D, acorde a la secuencia de entrenamiento de entrada. Esto asume que un límite superior a la orden de Markov de un "fuente certer" es conocida como *learner*.

4.1.3. Cadenas de Markov Dinámicas

Los Algoritmos DMC o *Dinamyc Markov Compression* son modelos de información con máquinas de estados finitos. Las asociaciones están hechas entre todos los símbolos posibles en el alfabeto origen y la distribución de probabilidad sobre todos los símbolos en el alfabeto. Esta distribución de probabilidad es usada para predecir el siguiente digito binario. Los *DMC* comienzan en un estado ya previamente definido, cambiando de estado cuando nuevos bits son leídos desde la entrada. La frecuencia de transmisión ya sea un 0 or un 1 son sumados cuando un nuevo símbolo entra. La estructura puede también ser actualizada usando *state cloning method*.

4.1.4. Lempel & Ziv

El algoritmo LZ78 es propuesto por Jacob Ziv y Abraham Lempel [16] en 1977. Un método de predicción en línea no necesita depender de pre-procesamiento tiempo de los datos históricos disponibles con el fin de construir un modelo de predicción. El pre-procesamiento se hace cuando tenemos una nueva petición. LZW y LZ78 básicamente son los algoritmos de compresión de datos sin pérdidas con una buena funcionalidad. La parte más

importante de estos algoritmos es la construcción de un diccionario de algoritmos que usamos para la creación del modelo predictivo.

```
initialize dictionary := null
initialize phrase w := null
loop
wait for next symbol v
if ((w.v) in dictionary):
w := w.v
else
add (w.v) to dictionary
w := null
increment frequency for every possible prefix of phrase

endif
forever
```

LZ78 es un algoritmo de compresión sin pérdidas, el algoritmo anteriormente muestra como el diccionario es construido a partir de secuencias, utilizando LZ78. En el entorno web utilizamos frecuencia de webaccess de páginas web del usuario como secuencia de entrada al algoritmo LZ78. La variable w es la secuencia que es recuperada en cada sesión de usuario. Este algoritmo puede insertar largas secuencias, pero en general, el número total de secuencias que inserta en el árbol es menos que el algoritmo PPM. Explicamos este algoritmo con un ejemplo. Supongamos que el usuario solicita las páginas $ABABCBC$ secuencialmente. Si utilizamos el algoritmo LZ78, entonces generaríamos un trie con nodos A, B, AB, BC y C, que deberán insertar. Cuando se inserta una secuencia en el árbol los contadores de las aristas que representan el paso desde la raíz hasta la última petición de secuencia se incrementa en cada inserción. Supongamos ahora que el usuario B pide a la secuencia de páginas $ABCABCD$, estos generaría cambios en el trie y de cumplir las condiciones se incrementarían los contadores.

Sin embargo se presentan ciertos problemas de análisis con LZ78. Cualquier aplicación práctica de *LZ78* sufre a partir de los siguientes inconvenientes:

- En cualquier análisis LZ una cadena de entrada, toda la información

cruzada de los bordes de las frases se pierden. En muchos casos, serian patrones y éstos afectarían el siguiente símbolo en la secuencia.

- La tasa de convergencia de *LZ78* a la previsibilidad óptima como se definió anteriormente es lento. Los resultados experimentales que realizaremos describirán que *LZ78* se acerca de forma asintótica a un óptima previsible.

Capítulo 5

Experimental

Lo que se busca es un modelo de accesos de navegación secuencial creado por un algoritmo de compresión, Lempel Ziv, y usarlo como un modelo predicción. Se usará para predecir secuencias finitas discretas de accesos webs. Al momento de generar el árbol este creará una representación Trie de un diccionario de símbolos, el cual utilizaremos para poder hacer predicciones.

En base a lo anterior y teniendo en funcionamiento el algoritmo para crear un modelo de predicción, se integrará con el servidor de Machine Learning Prediction.IO que ya se ha explicado anteriormente.

Usaremos la misma API que ofrece Prediction.IO para poder realizar las evaluaciones de nuestras métricas propuestas, dado el dominio de nuestro problema la métrica a usar será Accurracy (Exactitud) frente a distintas porciones de datos.

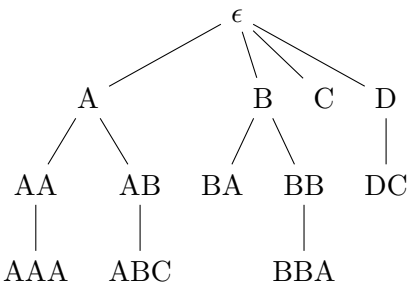
Con el dataset que tenemos haremos variadas pruebas para ver métricas como Accurracy, usaremos Cross Validation en distintos escenarios.

5.0.5. Nuestro Modelo de Predicción ML-LDC

Basados en un servidor de Machine Learning, hemos desarrollado un motor de predicción con un algoritmo basado en Lempel Ziv 78. Cada sesión es representada por un nodo y sus hijos. Sea la sesión

$$\{A, A, A, B, A, B, B, B, B, B, A, A, B, C, C, D, D, C, B, A, A, A, A\} \quad (5.1)$$

, la cual representaremos con nuestro modelo de predicción



Dado nuestro modelo de predicción podemos determinar que nuestra salida resultante será:

$$\{A, AA, B, AB, BB, BBA, ABC, C, D, DC, BA, AAA\} \tag{5.2}$$

Acotamos los siguientes casos en que nuestro Modelo de predicción funcionará

1. **Nodos Intermedios probabilidad Equivalente**

   Dada la $P(x|A\epsilon)$, siendo x la probabilidad de encontrar el siguiente símbolo. Este caso nos muestra que tenemos dos secuencias posibles $\{A, AB\}$, por lo cual podemos hacer la función $arc_max(AA, AB)$ ó calcular una función $random(AA, AB)$, al ser un caso con solamente dos posibilidades tenemos solo un 50 % de éxito.

2. **Nodos intermedios con un nodo hijo**

   Dada la $P(x|AB) = ABC$, ya que el nodo C, es el único hijo por tanto tiene toda la certeza de ser la predicción acorde al árbol de entrenamiento.

3. **Nodo hoja y vuelta a la raíz**

   Este caso es uno de los más sencillos ya que nuestro modelo al no tener más secuencias para poder hacer las operaciones, se proyectará el árbol para todos los posibles accesos dentro de nuestro alfabeto. Para el caso $\Sigma = \{A, B, C, D\}$ y la probabilidad para cada acceso representado por símbolos es de 25 %.

   Dada la probabilidad $P(x)$ al momento de retornar a la raíz, la siguiente secuencia es absorbida por ϵ.

5.0.6. Ambientes Experimental

Para los ambientes experimentales se han dispuestos dos máquinas para realizar las pruebas.

Máquinas

- Procesador 2,8 GHz Intel Core i7, 16 GB de Memoria RAM y Sistema Operativo OSX
- Procesadores Intel Xeon E5-2670 v2 (Ivy Bridge) de alta frecuencia 32 vCPU, 244 GB de Memoria RAM y Sistema Operativo Ubuntu 14.14

Para el proceso de desarrollo con IntelliJ y dataset menores a 500 sesiones se usará la máquina con 16GB y para experimentos con sesiones mayores a 1000 sesiones de usuarios se usará la máquina 240GB.

Software

- C++11
- Java 1.8
- Java(TM) SE Runtime Environment (build 1.8)
- Java HotSpot(TM) 64-Bit Server VM (build $25.51 - b03$, mixed mode)
- Scala code runner version 2.11.7 – Copyright 2002-2013, LAMP/EPFL
- SBT 0.13.9
- Python 2.7.10
- GNU bash 3.2.57 ($x86_64 - apple - darwin15$)
- Prediction.IO 0.9.4
- Elasticsearch 1.4.4
- Apache Spark-1.4.1
- Hbase 1.0.0
- Zookeeper

El motor de predicción es Prediction.IO, como se ha señalado anteriormente es un framework para desplegar servidores con algoritmos de Maquinas de Aprendizaje, Decission Tree, K-Means, RNN y todos los algoritmos ofrecidos por la suite de Apache Spark y MLib.

Para desarrollar un motor que se acople con *PIO* se deberá seguir el patrón *DASE* y crear un modelo con persistencia en memoria RAM que nos permita un acceso rápido a las predicciones por consultar.

Usaremos *SBT* para gestionar todas las librerías que se requieran como dependencia. Inherentemente usaremos Java, ya que el lenguaje Scala corre sobre la Java Virtual Machine. Prediction.IO no solo ocupa Scala, adicionalmente provee el uso de Apache Spark con sus librerías de MLIB (Machine Learning Library), Zookeeper, Hbase (Hadoop) y Elasticsearch. Hemos utilizado Python para realizar un cliente por línea de comando en el cual poder hacer pruebas y adicionalmente incluir un cliente que realice la carga de eventos desde nuestro set de datos experimental. Para mayor referencia de los clientes python ver los fuentes en los anexos.

5.0.7. Datos Experimentales

Se usarán las secuencias disponibles de *webaccess logs* pública de los sitios MSNBC. El set de datos provienen de los registros de un servidor IIS (Internet Information Services) msnbc.com de un día completo de la fecha 28 de Septiembre de 1999. Contiene secuencias de acceso web de 989,818 usuarios con un promedio de 5,7 categorías Página web visitas por secuencia, el tamaño de letras de este conjunto de datos es $\sigma = 17$.

Ejemplo de estructura de Dataset MSNBC :

```
% Different categories found in input file:

frontpage news tech local opinion on-air misc weather msn-news health living business msn-
     sports sports summary bbs travel

% Sequences:

A A
B
C B B D B B B C C
E
A
F
A A
F
F G G G F F H H H H
F I D D D J C J E J D D D
A A A K A A A
L L
```

```
A A
H H H H H H
```

Utilizaremos un simple programa en C++ para filtrar y listar los dataset y transformarlos a símbolos, caracteres en mayúscula, por ejemplo: [1]

```
1 1
2
3 2 2 4 2 2 2 3 3
5
1
6
1 1
6
6 7 7 7 6 6 8 8 8 8
6 9 4 4 4 10 3 10 5 10 4 4 4
```

Ejemplo de sub-dataset creados:

- Secuencias de webaccess con sesiones de más de 5 secciones web visitadas
- Secuencias de webaccess con sesiones de más de 10 secciones web visitadas
- Secuencias de webaccess con sesiones de más de 100 secciones web visitadas
- Secuencias de webaccess con sesiones equivalentes a 10 secciones web visitadas
- Secuencias de webaccess con sesiones menores a 5 secciones web visitadas

Trabajaremos con 1.000.000 de registros de los cuales hemos realizados distintas subconjuntos para hacer una validación cruzada:

- 10 Sesiones de usuarios
- 100 Sesiones de usuarios
- 500 Sesiones de usuarios
- 1.000 Sesiones de usuarios
- 5.000 Sesiones de usuarios
- 10.000 Sesiones de usuarios

[1] Este programa se encuentra junto en el anexo de la memoria

Page	Symbol
frontpage	A
news	B
tech	C
local	D
opinion	E
on-air	F
misc	G
weather	H
msn-news	I
health	J
living	K
business	L
msn-sports	M
sports	N
summary	O
bbs	P
travel	Q

- 50.000 Sesiones de usuarios
- 100.000 Sesiones de usuarios
- 500.000 Sesiones de usuarios
- 750.000 Sesiones de usuarios
- 1.000.000 Sesiones de usuarios

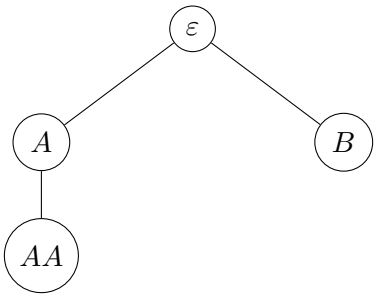
Es importante señalar que la división de nuestro set de datos es debido a que no conocemos la naturaleza de los mismo, pero con nuestro modelo podemos hacer un estudio para ir detectando patrones que puedan ser claves para medir el rendimiento en función a los criterios de volumen de datos versus el Accurracy(Exactitud).

5.0.8. Experimentos

Sesiones con un valor mínimo de visitas a secciones

Dentro de nuestro experimentos para probar la exactitud de nuestro modelo podemos demostrar que al tener una menor cantidad de secciones visitadas, la probabilidad de no acertar crece considerablemente.

Sea la siguiente sesión de un usuario $\{A, B, A\}$ para un alfabeto $\Sigma = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q\}$



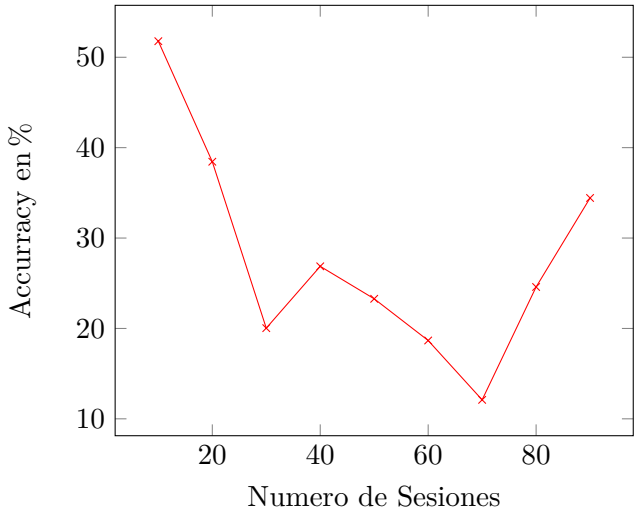
Como usted puede ver a menor cantidad de símbolos en la sesión estos generan un árbol de menor altura y menos nodos, cada nodo como se ha señalado en el capitulo 4 representa un visita a una sección en particular de la web de MSNBC, pero al ser un sesión tan corta y en la extensión de probabilidades esto hace que la probabilidad del siguiente acceso sea equiprobable dentro del los símbolos de nuestro diccionario.

Sea x el evento a predecir dada una secuencia discreta Eventualmente la probabilidad de $P(x|AB) = A$ para esta sesión de entrenamiento, pero la probabilidad $P(x|AAB) =?$ si extendieramos los simbolos de este nodo cada nodo hijo tendría un probabilidad de $\frac{1}{\Sigma} = \frac{1}{17} = 0.0588$.

Para corroborar este comportamiento haremos dos experimentos en distintos volumenes de datos, usaremos una validación cruzada para medir el Accurracy. Con esto demostraremos que lamentablemente la sesiones con menor visitas generar un tipo de ruido a nuestra métrica la cual se ve totalmente afecta en su rendimiento esperado.

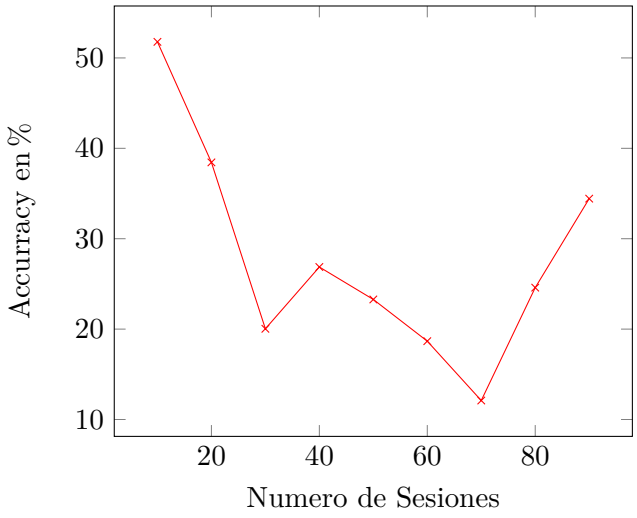
1. Sesiones con menos de 5 webaccess para generar el Trie

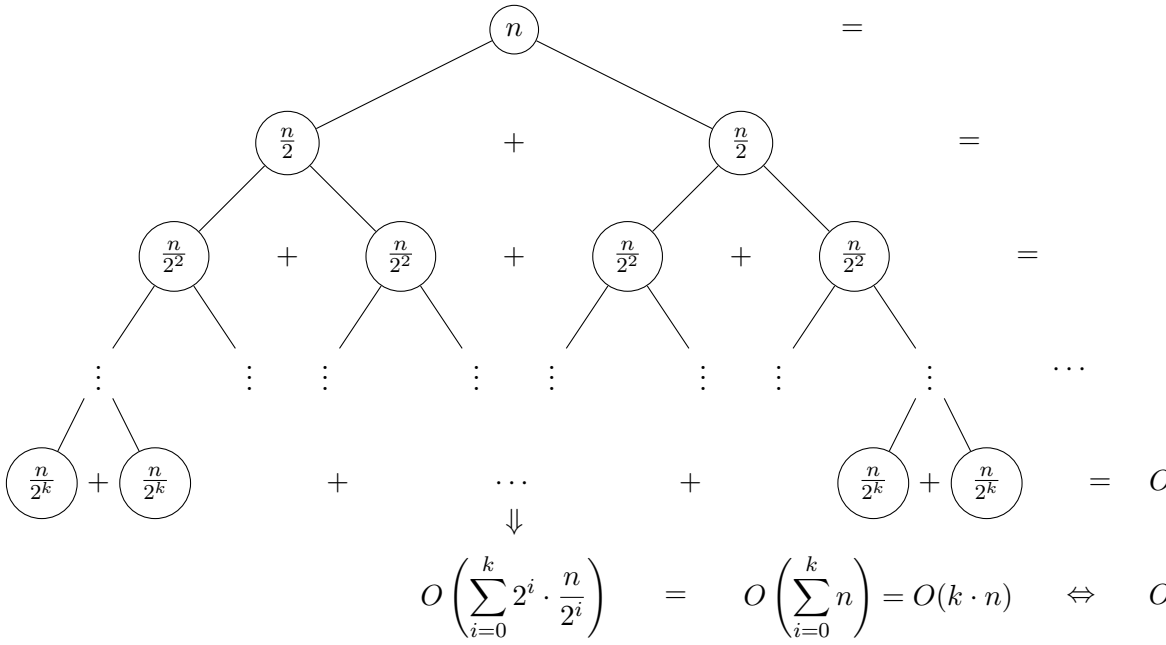
   Hacemos una validación cruzada para una muestra de data de 100 sesiones de usuarios para probar como se comporta el algoritmo con muestras de pruebas en una escala de 10.



2. Sesiones con mas de 10 webaccess para generar el Trie

Lo que se busca es ver como se comporta el algoritmo con datos discretos de cinco símbolos o más, es decir sesiones de usuarios que por lo menos han visitado cinco secciones de una web. Aquí se mide también lo estocástico de cada evento.





5.0.9. Detección de Ruido en sesiones

Al modelar una navegación de usuario mediante un trie basado en un algoritmo como LZ78, adoptamos un enfoque basado en la frecuencia por lo cual si realizamos experimentos para poder encontrar ruido veremos que que son secuencias de acceso comunes y estas no dan relevancia o aportan a la exactitud ó precisión del algoritmo.

Sea la secuencia $\{A, A, A, A, A, A, A, A, A, A, A, A\}$ a la cual llamaremos secuencia R, si A es representado por el *home* ó página de inicio, esto da a interpretación que existe un usuario en su sesión R que se encuentra accediendo constantemente a esta sección. Podemos señalar que esta es una sesión ruidosa si:

$$P(x|AAAAAAA) = A \tag{5.3}$$

, pero siendo la sesión R de tamaño = 12, en el siguiente acceso tendremos una probabilidad equiprobable dentro de las secciones en nuestro alfabeto, la cual generará un probabilidad de éxito "falso positivo".

En el siguiente experimento veremos como se comporta nuestro modelo dado entradas *Ruidosas*. Además se mostrará como el árbol suele perder su balance a medida que va creciendo los niveles de altura.

Por otro lado teniendo la noción de como es el funcionamiento de un

servidor IIS, y al ser una página con un gran número de visitas, podemos señalar que los datos proporcionados no son totalmente representativos de usuarios reales, ya que la web al bien indexada en los buscadores existen un cantidad indeterminada de *Crawlers* ó *Bots* que están constantemente generando accesos tanto para almacenar en caché páginas o generando accesos automatizados a ciertos secciones sin ser datos representativo. Dejamos como discusión que algoritmo se podría implementar para detección de estos patrones por las ramas que hemos generado para la detección de *bots* o *robots*

Hacemos un experimento con datos totalmente aleatorios de una muestra de 100 acceso aparte de darle mas diveridad a las sessiones acotamos también a sesiones que por lo menos tengan 3 webaccess

El orden de como se ingresan las sesiones afecta directamente proporcional a la construcción del modelo LZ Trie, por lo cual es un factor ya que es como un FIFO al momento de crear, lo primero que lee es lo primero que entrena por lo cual se debise tener un criterior para ordenar los webaccess antes de poder pasarlos al entrenador

@Discusión: Como LZ Trie, es en sí un compresor este no toma decisión, una posible mejora sería implementar intrinsicamente un Decission tree dentro de subarboles dentro del trie para poder elegir el más optimo acorde a los criterios historicos, asi podría darse el caso de ser un compresor-predictor "menos tonto"

@Pregunta : ¿Porque se presenta el patron que a menos datos de que tenga el trie mayor es accurracy?

5.0.10. Experimento con Largo de Ventana

5.1. Conclusiones

Aún cuando se presenta varios antecedentes, podemos decir que nuestro modelo ocupa bastante menos memoria pero esto va directamente relacionado el tamaño del trieNode de predicción generado.

Aun cuando el trieNode que representa totalmente el modelo de navegación del los usuarios de un sitio web, este no puede generalizar completamente el comportamiento estocástico de los usuarios y/o agentes que acceden a los

recursos de MSNBC o un cualquier web en general.

Una de las mayores ventajas de nuestro modelo es que al estar embebido en un servidor de Machine Learning, cada nuevo evento que ingresa para la siguiente nueva ejecución esta estará mejor preparado, podríamos decir que la recolección de data de cada evento en particular nos ayudaría a que nuestro modelo en futuros trabajos vaya aprendiendo mas y más precisamente.

A medida que la altura del árbol va creciendo este genera mayor demora en cuanto a la creación del *LZTrie*

Hemos validado que hacer un modelo de navegación de usuarios es un perfecta implementación para el uso de un árbol de la familia de Lempel & Ziv.

A nuestro modelo le es afectado secuencias de menor tamaño, por lo cual se debe trabajar para hacer un aprendizaje de que momento omitirlo o no. Estas sesiones de bajo número de secuencias genera un bajo porcetaje de accurracy.

Hemos presentado un modelo liviano el cual puede ser utilizado para predecir secuencias de webaccess en demanda.

lz effectively models sequential processes, and is extremely useful for prediction of processes where events are dependent on the previous event history. This is because of the ability of the algorithm to build an accurate model of the source of the events being generated, a feature inherited from its information theoretic background and the LZ78 text compression algorithm. The effectiveness of the method for learning a measure of time can also be attributed to fact that ALZ is a strong sequential predictor. The sound theoretical principles on which ALZ is founded also mean that ALZ is an optimal Universal Predictor, and can be used in a variety of prediction scenarios.

Dado que la myora de los predictores funcionan de manera offline, uno de los aporte de tener estar estructura de algoritmos como servicios es poder tener un motor de prediccion en linea.

Referencias bibliográficas

[1] F. Claude, R. Konow, and G. Navarro, "Efficient indexing and representation of web access logs," 2014.

[2] J. Rissanen, "A universal data compression system," *Information Theory, IEEE Transactions on*, vol. 29, pp. 656–664, Sep 1983.

[3] J. Langdon, G.G., "A note on the ziv - lempel model for compressing individual sequences (corresp.)," *Information Theory, IEEE Transactions on*, vol. 29, pp. 284–287, Mar 1983.

[4] J. Rissanen, "Universal coding, information, prediction, and estimation," *Information Theory, IEEE Transactions on*, vol. 30, pp. 629–636, Jul 1984.

[5] A. Moghaddam and E. Kabir, "Dynamic and memory efficient web page prediction model using lz78 and lzw algorithms," pp. 676–681, Oct 2009.

[6] M. Hartmann and D. Schreiber, "Prediction algorithms for user actions.," pp. 349–354, 2007.

[7] K. Gopalratnam and D. Cook, "Online sequential prediction via incremental parsing: The active lezi algorithm," *Intelligent Systems, IEEE*, vol. 22, pp. 52–58, Jan 2007.

[8] X. Dongshan and S. Junyi, "A new markov model for web access prediction," *Computing in Science Engineering*, vol. 4, pp. 34–39, Nov 2002.

[9] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "Web prefetching performance metrics: A survey," *Performance Evaluation*, vol. 63, no. 9–10, pp. 988–1004, 2006.

[10] Y. Chen, X. Chen, and H. Chen, "Improve on frequent access path algorithm in web page personalized recommendation model," pp. 83–86, March 2011.

[11] A. Rajimol and G. Raju, "Web access pattern mining, a survey," 2012.

[12] L. Kewen, "Analysis of preprocessing methods for web usage data," *Intemational Conference on Measurement, Information and Control (MIC)*, 2012.

[13] G. Poornalatha and P. Raghavendra, "Web page prediction by clustering and integrated distance measure," in *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pp. 1349–1354, Aug 2012.

[14] J. Rissanen and J. Langdon, G.G., "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, pp. 149–162, March 1979.

[15] R. Begleiter, R. El-Yaniv, and G. Yona, "On prediction using variable order markov models," *J. Artif. Int. Res.*, vol. 22, pp. 385–421, Dec. 2004.

[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *Information Theory, IEEE Transactions on*, vol. 23, pp. 337–343, May 1977.

UNIVERSIDAD DIEGO PORTALES

ANEXOS

Anexo A

Primer anexo

A.1. Uso de linea de Comando Prediction.IO

La interación con *PIO* es a través de una interface de linea de comando, esta sigue el siguiente formato de uso:

```
pio <command> [options] <args>...
```

En caso de tener duda usted al igual que los comandos de bash puede ejecutar

```
pio help <command>
```

, para ver mas detalles de cada detalle de los comandos disponibles.

Los comandos de PredictionIO se pueden separar en tres categorías:

- **help** Muestra un resumen del uso *pio help <command>* para leer sobre un sub comando
- **Version** Muestra la version instalada de PredictionIO
- **Status** Muesta la ruta de instalación y el estatus de ejecución de sistema, como también sus dependencias

Comandos del servidor de eventos

- **app** Muestra un resumen del uso *pio help <command>* para leer sobre un sub comando

- **Version** Muestra la version instalada de PredictionIO
- **app** Administra todas las aplicaciones que usa el servidor de eventos
- **pio app data-delete <name>** Borra toda la data contenida por una aplicación específica
- **pio app delete <name>** Borra una aplicación completa
- **eventserver** Lanza el servidor de eventos
- **–ip <value>** Une la IP seleccionada, el valor por defecto es *localhost*
- **accesskey** Administra todas las llaves de acceso al servidor de eventos ó app

A.2. Comandos del Motor de Predicción

Es requerido que estos comandos se ejecuten desde la misma carpeta que contiene el proyecto ó aplicación desplegada.

Las opciones –debug y –verbose muestran información detallada sobre los conectoes de las aplicaciones complementarias a las que esta compuesta PredictionIO

- **build** Construye y compila el proyecto completo desde la carpeta fuente, tiene un flag adicional – *clean*, para una compilación limpia.
- **train** Ejecuta el entrenamiento declarado en el motor
- **deploy** Despliega el motor para ser usado meiante REST como Algoritmo como Servicio. Si no existe ninguna nueva instancia desplegada, por defecto usará la última creada.

A.3. Configuraciones para hacer correr IntelliJ con Apache SPARK y Prediction.IO 0.94

```
Main class: io.prediction.workflow.CreateWorkflow

VM options: -Dspark.master=local -Dlog4j.configuration=file:/Users/jguzman/PredictionIO/conf
    /log4j.properties


Program arguments: --engine-id dummy --engine-version dummy --engine-variant engine.json
```

```
io.prediction.workflow.CreateWorkflow
-Dspark.master=local -Dlog4j.configuration=file:/Users/jguzman/PredictionIO/conf/log4j.
     properties -Dorg.xerial.snappy.lib.name=libsnappyjava.jnilib
--engine-id dummy --engine-version dummy --engine-variant engine.json



SPARK_HOME=/Users/jguzman/PredictionIO/vendors/spark-1.4.1/bin
PIO_FS_BASEDIR=/Users/jguzman/.pio_store
PIO_FS_ENGINESDIR=/Users/jguzman/.pio_store/engines
PIO_FS_TMPDIR=/Users/jguzman/.pio_store/tmp
PIO_STORAGE_REPOSITORIES_METADATA_NAME=pio_meta
PIO_STORAGE_REPOSITORIES_METADATA_SOURCE=ELASTICSEARCH
PIO_STORAGE_REPOSITORIES_MODELDATA_NAME=pio_model
PIO_STORAGE_REPOSITORIES_MODELDATA_SOURCE=LOCALFS
PIO_STORAGE_REPOSITORIES_APPDATA_NAME=pio_appdata
PIO_STORAGE_REPOSITORIES_APPDATA_SOURCE=ELASTICSEARCH
PIO_STORAGE_REPOSITORIES_EVENTDATA_NAME=pio_event
PIO_STORAGE_REPOSITORIES_EVENTDATA_SOURCE=HBASE
PIO_STORAGE_SOURCES_ELASTICSEARCH_TYPE=elasticsearch
PIO_STORAGE_SOURCES_ELASTICSEARCH_HOSTS=localhost
PIO_STORAGE_SOURCES_ELASTICSEARCH_PORTS=9300
PIO_STORAGE_SOURCES_LOCALFS_TYPE=localfs
PIO_STORAGE_SOURCES_LOCALFS_HOSTS=/Users/jguzman/.pio_store/models
PIO_STORAGE_SOURCES_LOCALFS_PORTS=0
PIO_STORAGE_SOURCES_HBASE_TYPE=hbase
PIO_STORAGE_SOURCES_HBASE_HOSTS=0
PIO_STORAGE_SOURCES_HBASE_PORTS=0


Main class: io.prediction.workflow.CreateServer
Program Arguments: --engineInstanceId **replace_with_the_id_from_pio_train**



Try -- for more information.
Usage: pio train [--batch <value>] [--skip-sanity-check]
                 [--stop-after-read] [--stop-after-prepare]
                 [--engine-factory <value>] [--engine-params-key <value>]
                 [--scratch-uri <value>]
                 [common options...]

Kick off a training using an engine (variant) to produce an engine instance.
This command will pass all pass-through arguments to its underlying spark-submit
command.

  --batch <value>
      Batch label of the run.
  --skip-sanity-check
      Disable all data sanity check. Useful for speeding up training in
      production.
  --stop-after-read
      Stop the training process after DataSource.read(). Useful for debugging.
  --stop-after-prepare
      Stop the training process after Preparator.prepare(). Useful for
      debugging.
  --engine-factory
      Override engine factory class.
  --engine-params-key
      Retrieve engine parameters programmatically from the engine factory class.
  --scratch-uri
      URI of the working scratch space. Specify this when you want to have all
      necessary files transferred to a remote location. You will usually want to
      specify this when you use --deploy-mode cluster.
```

A.4. Llamadas al Servidor de Machine Learning mediante curl

```
curl -H "Content-Type: application/json"  -d '{"webaccess" : "AC","
    num" : 10}' http://52.33.180.212:8000/queries.json
```

A.5. Python SDK para PredictionIO

Para hacer uso de estos scripts en python es necesario tener instalado el package de prediction para python sdk.

Si se tiene pip instalado correctamente se puede utilizar

```
pip install predictionio
```

ó

```
$ easy_install predictionio
```

Es recomendable tener acceso sudo para evitar problemas con permisos

```
(ie. sudo pip install predictionio)
```

```
import predictionio

engine_client = predictionio.EngineClient(url="http://localhost:8000")

print engine_client.send_query({"webaccess": "A", "num": 10})
```

```
"""
Send sample query to prediction engine
"""

import predictionio
import readline

engine_client = predictionio.EngineClient(url="http://localhost:8000")
while True:
    word = raw_input('Enter a Sequences or a single page to predict the next user webaccess:
          \n')
    print engine_client.send_query({"webaccess": word, "num": 10})
```

A.6. Programa C++ para hacer splits dentro del Dataset

Programa para poder pasar la data de msnbc a una representación de símbolos.

```
#include <iostream>     // cout
#include <fstream>      // ifstream
#include <sstream>
#include <algorithm>
#include <string>
#include <cmath>
#include <cstdio>
#include <vector>
#include <map>
#include <iterator>

using namespace std;

/**
  alias lseq = g++ -std=c++11 letterSequences.cpp -o letterSequences
  ./letterSequences

% Different categories found in input file:

frontpage news tech local opinion on-air misc weather msn-news health living business msn-
     sports sports summary bbs travel
**/


int main()
{
   map<string, int> mapCategories;


   // Inserting data in map
  mapCategories.insert(make_pair("frontpage", 1));
    mapCategories.insert(make_pair("news",    2));
    mapCategories.insert(make_pair("tech",    3));
    mapCategories.insert(make_pair("local",   4));
    mapCategories.insert(make_pair("opinion",   5));
    mapCategories.insert(make_pair("on-air",  6));
    mapCategories.insert(make_pair("misc",    7));
    mapCategories.insert(make_pair("weather",   8));
    mapCategories.insert(make_pair("msn-news",  9));
    mapCategories.insert(make_pair("health",  10));
    mapCategories.insert(make_pair("living",  11));
    mapCategories.insert(make_pair("business",  12));
    mapCategories.insert(make_pair("msn-sports",13));
    mapCategories.insert(make_pair("sports",  14));
    mapCategories.insert(make_pair("summary",   15));
    mapCategories.insert(make_pair("bbs",     16));
    mapCategories.insert(make_pair("travel",  17));


   vector<char> alphabet = { 'A','B','C','D','E','F','G',
                'H','I','J','K','L','M','N','O',
                'P','Q','R','S','T','U','V','W',
                'X','Y','Z'};

   // Iterate through all elements in map
   map<string, int>::iterator it = mapCategories.begin();

   ifstream  fin("msnbc990928.seq");
   string    file_line;
   int fold = 0 ;

   while(getline(fin, file_line)){

    string    buf; // Have a buffer string
    stringstream  ss(file_line); // Insert the string into a stream
    vector<string> tokens; // Create vector to hold our words

    while (ss >> buf) tokens.push_back(buf);

    if( tokens.size() < 6 ){
```

```
        ++fold;
        for (int i = 0; i < tokens.size(); ++i){
          string tmp = tokens.at(i);
          cout << alphabet.at( stoi(tmp) - 1) << " ";
        }cout<< endl;

      }

      //this value is for make the size of the folds of data
      if( fold == 1000000 ) break;

     }
      return 0;
}
```