UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

UNIVERSIDAD DE LA REPÚBLICA
Facultad de Ciencias Económicas y de Administración
Licenciatura en Estadística
Informe de Pasantía

Detección de Anomalías para el Aseguramiento de Aplicaciones Web

Nicolás Montes de Marco

Tutores:
PhD. Juan José Goyeneche
PhD. Gustavo Betarte
PhD. Álvaro Pardo

Montevideo, Marzo 2018
Uruguay.

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE CIENCIAS ECONÓMICAS Y DE ADMINISTRACIÓN

El tribunal docente integrado por los abajo firmantes aprueba el trabajo de Pasantía:

Detección de Anomalías para el Aseguramiento de Aplicaciones Web

Nicolás Montes de Marco

Tutores:
PhD. Juan José Goyeneche
PhD. Gustavo Betarte
PhD. Álvaro Pardo
Licenciatura en Estadística

Puntaje .....
Tribunal
Profesor.....(nombre y firma).
Profesor.....(nombre y firma).
Profesor.....(nombre y firma).
Fecha.....

Resumen

A medida que el mundo se vuelve cada vez más dependiente de Internet, los ataques cibernéticos en la World Wide Web están aumentando tanto en frecuencia como en gravedad. Es conocido que muchas aplicaciones web pueden presentar distintos tipos de vulnerabilidades. A menudo, estas últimas (más aún en el caso de aplicaciones web desarrolladas a medida) sólo pueden ser descubiertas como un proceso de ensayo y error por parte de un atacante. Es así que se vuelven necesarias técnicas de detección y prevención de ataques. El desarrollo de este tipo de técnicas involucran procedimientos que ayudan a discernir entre el comportamiento de un usuario válido del sistema y un agente malicioso.

Los sistemas de detección de intrusos son configurados con reglas y directivas con el fin de detectar y bloquear el tráfico web malicioso. Tradicionalmente estas reglas son diseñadas a partir de un conjunto de firmas de ataques predefinidos. Estos sistemas resultan muy efectivos para detectar ataques conocidos (aquellos que *machean* con alguna de las firmas disponibles) sin embargo uno de los problemas es la incapacidad de detectar aquellos nunca antes observados (conocidos como *zero day attacks*). Otra dificultad que tienen estos sistemas es que requieren una frecuente actualización del conjunto de firmas utilizadas. Un enfoque complementario consiste en la utilización de un sistema de detección de anomalías. El mismo se basa en modelar el perfil normal del recurso que se quiere proteger (en la fase de aprendizaje) y posteriormente (en la fase de detección) reconocer como un ataque todo comportamiento que tenga un patrón anómalo bajo dicho perfil.

En este proyecto de grado se estudiaron técnicas de aprendizaje automático y reconocimiento de patrones, utilizadas para el diseño de sistemas de detección de anomalías. Particularmente se profundizó en un algoritmo denominado **Fusión de modelos bayesiana** (*Bayesian Model Merging*). En el mismo se combinan formalismos de *Modelos Ocultos de Markov (Hidden Markov Models)*, *Autómatas Probabilísticos*, *Inferencia Bayesiana* y *Teoría de la Información* para inferir la **gramática probabilística** generadora de un lenguaje regular objetivo, a partir de ejemplos de entrenamiento. Se implementó el algoritmo y se experimentó en dos estudio de casos. El primero consistió en aprender un lenguaje creado artificialmente, a partir de 8 ejemplos de entrenamiento. El segundo es un prototipo que reconoce nombres de personas. El propósito fue simular la protección de una aplicación que tiene como tráfico normal nombres de personas. Los resultados son promisorios: en el primer estudio de caso se llegó al modelo generador del lenguaje regular artificial objetivo. En el segundo estudio de caso se logró crear una gramática capaz de reconocer nuevos nombres (sin sobre-generalizar) a partir de 11 ejemplos de entrenamiento.

Palabras Clave: Aprendizaje automático, Modelos Ocultos de Markov, Inducción de Gramáticas, Aprendizaje Bayesiano, Ciberinteligencia, Aprendizaje de la Topología de HMM, Algoritmos Inductivos, Autómatas probabilísticos, Aplicaciones Web, Web Application Firewall, ModSecurity.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Este trabajo de grado se enmarca en la actividad desarrollada dentro del proyecto de I+D **WAFINTL**(ICT4V, 2017). Este es un proyecto del ICT4V (**Information and Communication Technologies for Verticals**) que tiene como objetivo general la concepción y desarrollo de un *framework* que provea soporte metodológico y automatizado para la implementación de mecanismos de detección, identificación, análisis y prevención de ataques informáticos a las aplicaciones web y los activos sensibles de sus organizaciones.

Figura 1: Nube de etiquetas de la Web, imagen extraída de (Web Design Word Cloud Photo, 2017)

1.1. Objetivo

El objetivo de este proyecto de grado es investigar y aplicar técnicas de detección de anomalías, para complementar a los enfoques clásicos (basados en reglas) utilizados para la detección y prevención de ataques por parte de las tecnologías actuales, siendo una referencia paradigmática el módulo de Apache ModSecurity (ModSecurity, 2017).

La incorporación de técnicas de detección de anomalías permitirán mejorar tanto la capacidad de detección de ataques como también disminuir la cantidad de falsos positivos.

Específicamente se va a proponer un modelo estadístico predictivo que permita identificar comportamiento considerado anómalo. Para ello se investigará y definirá un procedimiento automático de detección de anomalías que aprenda de ciertos patrones en los datos. Dichas anomalías deberán entonces ser analizadas como indicios de la presencia de ataques en la aplicación objeto de estudio.

1.2. Contexto General

Desde el comienzo de la existencia de las redes globales de computadoras han existido usuarios maliciosos que han intentado explotar las vulnerabilidades de esas redes. Hasta hace unos años las principales amenazas a los sistemas consistían en código malicioso auto-propagante. El avance en tecnologías anti-virus contribuyó significativamente a reducir los riesgos implicados por estos artefactos.

El impacto que causó en la década pasada la Web 2.0 en transacciones sensibles corrientes generó una demanda sin precedentes de técnicas y herramientas para garantizar la seguridad de las aplicaciones web. En la era emergente de la Web 3.0, la problemática de la seguridad del ciberespacio ha devenido una agenda de investigación crítica que reclama el interés de una **fuerza de trabajo científica multidisciplinaria**.

A medida que el mundo se vuelve más dependiente de las aplicaciones Web para transacciones comerciales, financieras y médicas, los ataques cibernéticos en la World Wide Web están aumentando en frecuencia y gravedad. Las aplicaciones Web ofrecen una alternativa atractiva a las aplicaciones de escritorio tradicionales debido a su accesibilidad y facilidad de despliegue. Sin embargo, la accesibilidad de las aplicaciones Web también las hace extremadamente vulnerables a los ataques.

Esta vulnerabilidad inherente se ve intensificada por la naturaleza distribuida de estas aplicaciones y la complejidad de la configuración de los servidores de aplicaciones. Estos factores han llevado a una proliferación de ataques en los que los atacantes, por ejemplo, inyectan código en las solicitudes HTTP, permitiéndoles ejecutar comandos arbitrarios en sistemas remotos y realizar actividades maliciosas como leer, alterar o destruir información sensible.

Desde el punto de vista de la seguridad, **la World Wide Web es a la vez críticamente importante y críticamente vulnerable**. La Web es de importancia crítica porque proporciona acceso a (1) servicios cruciales (por ejemplo, salud, gobierno, comercio electrónico, educación, banca, etc.) y (2) datos sensibles (por ejemplo, privados, financieros, médicos). La presente década ha experimentado un crecimiento increíble en la disponibilidad de servicios e información en línea.

Según estimaciones recientes, el 80 % o más de los usuarios de Internet de EE. UU, es decir, cientos de millones hacen compras en línea. En Uruguay, el Centro de Respuestas a Incidentes de Seguridad del Uruguay (CERTuy) semestralmente publica estadísticas de incidentes de seguridad informática incurridos en nuestro país, lo que permite visualizar las tendencias en ciberataques y realizar acciones en consecuencia.

En el primer semestre de 2017 según datos de CERTuy se atendieron 457 incidentes, y se observa un crecimiento del 9 % con respecto al mismo período del año anterior.

La Web es críticamente vulnerable porque es inherentemente pública y accesible y además distribuida de forma ininterrumpida. El drástico incremento de ataques informáticos obliga a los gobiernos, organizaciones y empresas, a considerar la seguridad de la información, de las aplicaciones y de la infraestructura informática como un tema prioritario.

Debido a la diversidad, multiplicidad y la continua innovación de los sistemas actuales se torna necesario automatizar procesos y generar herramientas que asistan a los desarrolladores, administradores de sistemas, auditores y analistas de seguridad en esta tarea.

1.3. Problemática

Los tipos de ataques a los que se puede ver enfrentado un sistema de información, en particular los sistemas Web multi-capas, han evolucionado desde lo que eran agentes disruptivos distribuidos a la implementación de ataques multi-etapas de bajo perfil cuyo fin es lograr alcanzar objetivos tácticos y establecer una presencia persistente dentro del ecosistema de la organización amenazada.

Este tipo de ataques, conocidos como Amenazas Persistentes Avanzadas (en inglés *Advanced Persistent Threats* (APT)), requieren de una estrategia de defensa proactiva, lo que en parte contrasta con el enfoque tradicional reactivo de la ciber-seguridad, que se ha focalizado principalmente en el entendimiento y solución de vulnerabilidades, lo que sigue siendo necesario, pero ya no es suficiente. Esta estrategia defensiva se la conoce con el nombre de *cyber threat intelligence* (cyberintell, ciberinteligencia), la que se define como el entendimiento de las capacidades, acciones e intención de los adversarios en el contexto de la seguridad informática.

Una de las áreas principales dentro del campo de la ciberinteligencia es la responsable de enfocarse en los ataques a aplicaciones Web. Este campo tiene las siguientes particularidades:

- Gran exposición de los sistemas, frecuentemente abiertos a cualquier usuario en Internet
- La facilidad de acceso y la exposición de estos sistemas llevan a que las organizaciones expongan a través de éstos información sensible
- La cantidad y diversidad de atacantes hace que la tarea de entendimiento e intención de los adversarios sea muy amplia y más compleja de analizar
- En general no existen técnicas y herramientas de software que se adapten completamente a las necesidades concretas de una organización, y por lo tanto se hace necesario el desarrollo de soluciones a medida que brinden las funcionalidades necesarias a los procesos de negocio/misión de la misma, lo que conlleva a tener mayor cantidad de problemas específicos.

Es conocido que las aplicaciones web pueden presentar diferentes tipos de vulnerabilidades. Pese al surgimiento de diferentes iniciativas, como por ejemplo el TOP 10 de OWASP (OWASP Top Ten, 2017), la proliferación de las vulnerabilidades en aplicaciones web va en aumento. Siempre que se detecte una vulnerabilidad que afecte a una aplicación, la solución obvia es analizar las formas en que la vulnerabilidad puede ser arreglada y modificar correspondientemente el código de la aplicación, el diseño de la arquitectura y/o la actualización de la plataforma en la que se ejecuta la aplicación. Sin embargo, hay ciertas situaciones en las que se necesita un enfoque alternativo, por ejemplo si:

1. La aplicación fue implementada por un tercero, y sólo el código ejecutable está disponible
2. El equipo que desarrolló la aplicación ya no está disponible
3. La aplicación es heredada o se ha desarrollado utilizando una tecnología no disponible
4. El costo de detectar y corregir el error es prohibitivo

También es frecuente que la remediación de una vulnerabilidad requiera un nuevo proyecto de desarrollo, lo que implica, entre otras tareas, realizar revisiones del código fuente, realizar pruebas de penetración e identificar problemas que la corrección podría llevar adelante. **Es debido al esfuerzo que implica abordar ese tipo de problemas que se han propuesto técnicas alternativas de mitigación**.

Una técnica comúnmente utilizada es Virtual Patching (OWASP Virtual Patching, 2017). Esta técnica consiste en añadir a la aplicación una capa externa que es responsable de filtrar los datos que fluyen hacia y desde la aplicación. En cuanto a la seguridad, la idea principal es que la capa se encargará de detectar y gestionar adecuadamente un ataque que intente explotar la vulnerabilidad.

Este enfoque no requiere que el código de aplicación se modifique para generar la protección necesaria, pero se necesita experiencia para configurar la herramienta que debe hacer cumplir el parche virtual. Ejemplos de herramientas de remediación virtuales son los sistemas de prevención de intrusiones de red (NIPS) (Roesch, 2005; Suricata, 2017) que supervisan el tráfico de red y bloquean la actividad maliciosa, firewalls de base de datos (DBF) (Oracle, 2017; Sophos XG Firewall, 2017) que filtran el tráfico de red desde y hacia una base de datos y los firewalls de aplicación web (WAF) (Barracuda, 2017; ModSecurity, 2017).

En este contexto, y más aún en el caso de aplicaciones web desarrolladas a medida, es posible que muchas vulnerabilidades de software sólo puedan ser descubiertas como resultado de un proceso de ensayo y error proveniente de un atacante. Es así que se vuelven necesarias las técnicas de detección y prevención de ataques.

El desarrollo de este tipo de técnicas involucra procedimientos que ayudan a discernir entre el comportamiento de un usuario válido del sistema y un agente malicioso. La identificación y determinación de un comportamiento no contemplado debe tener en cuenta si cada evento detectado es simplemente sospechoso o en realidad se trata de un evento que es parte de un ataque.

Las técnicas de detección de ataques se vuelven vitales a la hora de determinar los umbrales adecuados para las acciones de respuesta. Este tipo de técnicas colaboran en aspectos tales como impedir que los atacantes identifiquen/verifiquen con éxito la existencia de vulnerabilidades en los sistemas y minimizar la cantidad de falsos positivos (actividad no maliciosa identificada como tal).

Al momento de implementar este tipo de técnicas, un problema no menor es la cantidad de aplicaciones ya desarrolladas en las organizaciones y que deben de ser protegidas por este tipo de soluciones. **Una alternativa tecnológica para realizar este tipo de análisis puede estar basada en el uso de un Firewall de aplicaciones Web**.

En investigaciones recientes se han aplicado técnicas de aprendizaje automático y de minería de datos para diseñar, desarrollar y mejorar algoritmos para el diseño de sistemas de seguridad cibernética. El aprendizaje automático y la minería de datos desempeñan un papel importante en la ciberseguridad (Dua y Du, 2016), especialmente a medida que aparecen más desafíos con el rápido desarrollo de técnicas de descubrimiento de información, como las que se originan en la dimensionalidad y naturaleza heterogénea de los datos de la red, el dinamismo con que cambian las amenazas y el severo desbalance entre comportamiento normal y anómalo.

1.4. Organización del Documento

El resto del documento está estructurado de la siguiente forma:

En el Capítulo 2 se da una visión general de las aplicaciones Web y una breve descripción de las principales amenazas. Posteriormente se plantean los desafíos técnicos para los que se presentan soluciones en este trabajo, como la utilización de algoritmos de aprendizaje automático y reconocimiento de patrones utilizados para sistemas de detección de intrusos. Luego se presenta el algoritmo de aprendizaje automático que se va a utilizar, los Modelos Ocultos de Markov. Por último se realizará una breve descripción de los autómatas finitos.

En el Capítulo 3 se presenta el estado del arte mediante el estudio de una compilación de investigaciones que se han realizado.

En el Capítulo 4 se presenta el problema en el contexto y se profundiza en la metodología de una técnica particular seleccionada del estado del arte. La misma consiste en ver los Modelos Ocultos de Markov como un autómata probabilístico. Dicha técnica permite inferir la estructura de estos modelos a partir de datos de entrenamiento.

En el Capítulo 5 se describen detalles relacionados a la implementación, se describen las subrutinas utilizadas y las funciones creadas para la utilización del algoritmo propuesto.

En el Capítulo 6 se expone la experimentación en dos estudio de casos. El primero para aprender un lenguaje creado artificialmente, el segundo es un prototipo que reconoce nombres de personas.

Por último, en el Capítulo 7 se establecen las conclusiones, se evalúan los resultados, se plantean dificultades y sus posibles extensiones.

2. Aplicaciones Web, Principales Amenazas, Detección de intrusos, Modelos Ocultos de Markov y Autómatas

2.1. Visión General de las aplicaciones Web

Una aplicación Web es cualquier aplicación que es accedida vía web por una red como internet o una intranet. Comúnmente se utiliza el término aplicación web para identificar el conjunto de ejecutables/scripts que ofrecen un cierto número de servicios (por ejemplo un motor de búsqueda ofrece: página principal, interrogación de bases de datos, generación de imágenes, etc.). En general (y en el contexto de este trabajo), el término también se utiliza para hacer referencia a cada programa o script que genera contenido web dinámico.

Una aplicación web consta de una gran cantidad de tecnologías; particularmente se va a prestar atención en la interacción **entre un cliente y una aplicación.** En la figura 2 extraída de (Ariu, 2010) se esquematiza una situación típica, en la cual un **navegador web** realiza una petición (de un cierto recurso -*request*-) al **servidor web**. En este caso a **search.php**

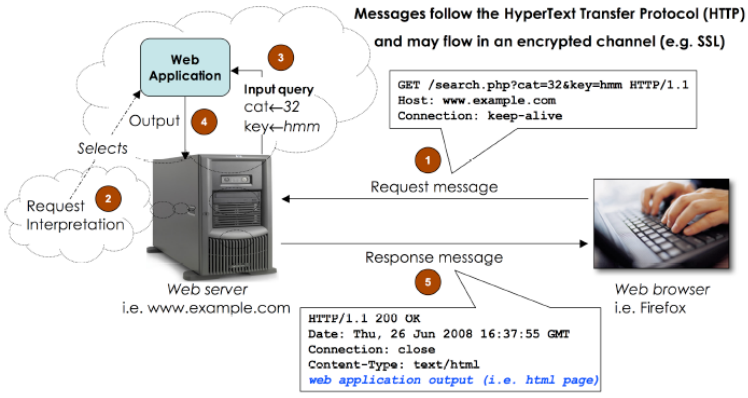


Figura 2: Ejemplo de petición a una aplicación Web, imagen extraída de (Ariu, 2010)

1. Un paquete de red es enviado desde el navegador web hacia el servidor web (www.example.com). El paquete contiene una petición a la página **search.php**. La petición es enviada de acuerdo al protocolo de transferencia de hipertexto (por sus siglas en inglés HTTP) (HTTP, 2017) entre el navegador y el servidor (basado en pedidos y respuestas, que pueden ser desde páginas HTML hasta vídeos o archivos para descargar). Se puede observar en el ejemplo, que al servidor web se le está enviando **cierta información en la petición**:

   - **El nombre del método HTTP**. Se pueden utilizar diferentes métodos, en función de lo que el cliente requiere del servidor web. En particular el método GET, es el utilizado típicamente para pedir un recurso. También existen otros métodos, por ejemplo el método POST que usualmente se utiliza para enviar información, archivos o cuerpos grandes de texto.
   - La **Request URI** que identifica una página (o varias) en un servidor web por el camino (*path*) y/o los parámetros de la consulta (también llamados atributos)

Por ejemplo, si www.example.com es desarrollada de forma **estática** y tiene una página en el sitio llamada "about.html", que está localizada en un subdirectorio del sitio, la URI de esa petición podría ser **"/prag/eng/home.html"**.
Por otra parte si **www.example.com** es desarrollada de forma **dinámica** en php, la URI de la petición a esa página sería **"/pages.php?group=prag&lang=eng&page=about.php"**.

El ejemplo de la figura 2 contiene una petición a la aplicación **"search.php"**. La aplicación recibe dos atributos: "cat" con el valor 32 y "key" con el valor *hmm*. Vale la pena remarcar que se va a considerar cada uno de los scripts en el servidor web como una aplicación diferente.

- **La versión del protocolo HTTP**. En el ejemplo de la figura 2 es enviado de acuerdo a la versión 1.1
- **Un set de cabeceras HTTP**. En el ejemplo de la figura 2 se pueden identificar dos. La cabecera del *Host* que especifica el host del recurso pedido y la cabecera de la conexión, que permite especificar como se realizara la misma. Por ejemplo en la figura 2 se envía la opción *"connection: keep-alive"*, la cual permite dejar la conexión abierta. De esta forma si posteriormente se quiere pedir un nuevo recurso no se tiene que establecer una nueva conexión.

2. Cuándo el servidor recibe la petición HTTP interpretada por un analizador sintáctico (*parser*), extrae el **nombre de la aplicación requerida y los atributos introducidos**.
3. El servidor llama a la aplicación y le pasa los atributos como argumentos
4. La aplicación genera el recurso requerido
5. El recurso es enviado al cliente dentro de un mensaje de respuesta

2.2. Ataques contra aplicaciones Web

Por lo general en el desarrollo de (complejos) sistemas informáticos los requerimientos de seguridad se encuentran embebidos entre los requerimientos funcionales. Por lo tanto, es difícil integrarlos correctamente en el proceso de desarrollo de software tradicional y rara vez son tomados en cuenta en las primeras etapas de dicho proceso.

Si bien los desarrolladores de la infraestructura de *software* (es decir de servidores webs y bases de datos) tienen un gran conocimiento de las cuestiones de seguridad; los desarrolladores de aplicaciones web a menudo pueden tener poco conocimiento en dichas cuestiones. Muchas veces estos últimos por estar focalizados en la funcionalidad del usuario final, o al trabajar con ciertas restricciones de tiempo, pasan por alto un necesario análisis de seguridad de la aplicación (Kruegel y Vigna, 2003).

Desafortunadamente, la arquitectura multi-capas de las aplicaciones Web dificulta la aplicación de mecanismos de seguridad. Estas aplicaciones usualmente son codificadas en un lenguaje principal (como

PHP o Java) que es ejecutado en una capa y dinámicamente genera programas en un lenguaje objeto (como SQL, HTML o Java Script) que es ejecutado en otra capa. Adicionalmente, y con el objetivo de facilitar/mejorar la experiencia del usuario, las aplicaciones en general usan datos de entrada provistos por el usuario y no siempre validados.

Estos datos pueden ser cargados en memoria o en una base de datos y ser utilizados por otros programas. La utilización de datos de entrada no confiables para la generación de otros cálculos puede llegar a representar una vulnerabilidad muy seria para la aplicación si estos son confundidos con código a ser ejecutado.

Este tipo de ataque es conocido como de *code injection* (inyección de código), y en forma general puede ser entendido como violación de integridad. A continuación se presentan brevemente algunos de los ataques más frecuentes.

2.2.1. Inyección SQL

Inyección SQL es una clase de ataques que ocurren cuando el usuario es capaz de cambiar la estructura de una consulta SQL (lenguaje de consulta estructurada) que al ejecutarse en la base de datos tiene un efecto que no es el deseado por la aplicación.
Por ejemplo como se explica en (Kruegel y Vigna, 2003), cuando una aplicación web le permite desplegar una lista de todas las tarjetas de crédito registradas de un usuario, el pseudo código sería el siguiente:

```
uname=getAuthenticatedUser()
cctype=getUserInput()
result=sql(SELECT nb FROM creditcards WHERE user='" + uname + "' AND type='" +cctype+"';")

print(result)
```

Si cierto usuario **Bob** desea realizar una búsqueda de todas sus tarjetas **VISA**, la consulta normal que se ejecutaría sería **SELECT nb FROM creditcards WHERE user='bob' AND type='VISA';**. Dicho ejemplo es vulnerable a inyección de código SQL.
Si Bob, malintencionadamente, quisiera conocer todas las tarjetas que pertenecen a otro usuario por ejemplo **'alice'**, podría introducir la secuencia **'OR user='alice;** logrando cambiar la consulta y arbitrariamente ejecutar:

```
SELECT nb FROM creditcards WHERE user='bob' AND type=''OR user='alice'.
```

La correcta implementación de la aplicación, no debería permitir al usuario introducir datos que cambien la estructura de una consulta SQL. Antes de introducir dichos datos se debe asegurar haberlos sanitizado correctamente.

2.2.2. Cross-Site Scripting (XSS)

Las fallas XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar secuencia de comandos en el navegador de la victima los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso (OWASP Top Ten, 2017).

2.2.3. Buffer Overflow

El desbordamiento de buffer (del inglés *buffer overflow*) se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer), de forma que si dicha cantidad es superior a la capacidad preasignada los bytes sobrantes se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original (Pavey, 1995).

2.3. Sistemas de detección de intrusos

Los Sistemas de detección de intrusos (IDS por sus siglas en inglés) según (Patcha y Park, 2007) reúnen información de distintas fuentes, como una computadora o una red, para identificar posibles brechas de seguridad. En otras palabras, se encargan de detectar acciones que intentan comprometer la confidencialidad, la integridad y la disponibilidad de un sistema o una red.

Los IDS se dividen en dos categorías que dependen del comportamiento del intruso:

- **Network-based** que monitorea el tráfico entre dispositivos de red
- **Host-based** que monitorea el entorno de *software* asociado con un *host* específico.

Otra división puede resultar por la manera de operar del IDS; las tres categorías son:

- **Sistema de detección de firmas**

  Es una técnica de detección de intrusos que se basa en un set de firmas de ataques predefinidas. Se detecta un ataque solamente si machea con alguna firma de un ataque conocido. Dicho sistema resulta efectivo para detectar ataques conocidos sin generar un número muy alto de falsas alarmas. Otras de las ventajas es que comienza a proteger al sistema/computador, inmediatamente luego de ser instalado.

  Uno de los grandes problemas es que requiere una actualización frecuente de la base de datos con reglas y firmas. El otro problema es que no puede detectar nuevos ataques (*zero-day attacks*) ya que necesita conocer las firmas de todos los posibles ataques.

- **Sistema de detección de anomalías**

  En sistemas de detección de anomalías se crea un modelo estadístico del recurso al cual se está protegiendo. Se define el comportamiento normal de un recurso como “un conjunto de características que fueron observadas en una operación normal” . Luego de crear el modelo

estadístico para el comportamiento normal, se detecta un ataque si es un patrón anómalo bajo el modelo inferido.

Su gran ventaja es la habilidad para detectar nuevos ataques. **Otra de las ventajas es que el perfil de la actividad normal es personalizada para cada aplicación**. Además cuando hay datos que estas técnicas categorizan como ataques, pueden ser utilizados como insumo para técnicas de detección de firmas.

En el pasado la popularidad de los sistemas de detección de anomalías se ha visto restringidos debido al alto porcentaje de falsas alarmas que pueden generar. Esto puede ocurrir en aquellos recursos que cambian frecuentemente. Si el modelo estadístico no tiene en cuenta esas variaciones el sistema corre riesgo de generar una alta cantidad de falsas alarmas. A pesar de eso, hoy en día muchos de los productos comerciales que existen en plaza están basados en mecanismos de detección de anomalías.

Los sistemas de detección de anomalías también tienen sus desventajas (Patcha y Park, 2007), por un lado necesitan pasar por un período de entrenamiento necesario para inferir el perfil normal. Además crear un perfil con el tráfico normal es todo un desafío, y la creación de un perfil inapropiado puede llevar a un desempeño pobre del clasificador.

- **Un sistema de detección híbrido o compuesto** en los que se combina ambos enfoques

Cualquiera haya sido la técnica utilizada, existen ciertas métricas utilizadas para evaluar la exactitud y la capacidad de detección que tiene cierto IDS. Es decir evaluar cuántos objetos se etiquetaron correctamente y cuántos no. Formalmente a menudo se identifica a la clase *ataques* como la **positiva** y la clase *normal* como la **negativa**. Se definen las siguientes situaciones que pueden ocurrir en el proceso de clasificación:

- **Verdaderos positivos**. Un verdadero positivo es un ataque que fue detectado correctamente por el IDS.
- **Falsos positivos**. Un falso positivo consiste en un objeto normal que es considerado como un ataque.
- **Verdaderos negativos**. Un verdadero negativo es un objeto normal que está correctamente clasificado como normal.
- **Falsos negativos**.Un falso negativo es un ataque que logro evadir el sistema sin ser detectado

Utilizando las definiciones mencionadas anteriormente; un sistema de detección de intrusos puede ser evaluado con las siguientes métricas:

$$Coeficiente\ de\ Falsos\ Positivos = \frac{Negativos\ clasificados\ incorrectamente}{Total\ de\ negativos}$$

$$Coeficiente\ de\ \text{Detección} = 1 - Ratio\ de\ Falsos\ Negativos$$

Donde el coeficiente de Falsos Negativos se define como

$$Coeficiente\ de\ Falsos\ Negativos = \frac{Positivos\ clasificados\ incorrectamente}{Total\ de\ positivos}$$

En este proyecto se pone foco en **seguridad de aplicaciones web**, particularmente en las denominadas **host-based solutions** mencionadas anteriormente.

Una posible solución son los Firewall de Aplicación Web (WAF). En la Subsección siguiente se explica de que se trata este tipo de tecnología.

2.3.1. Web Application Firewall

Un WAF es un componente que se instala entre los usuarios y la aplicación y se encarga de analizar todos los pedidos. Los firewalls de red tradicionales están diseñados para realizar la inspección de paquetes a nivel de red. Una lista de reglas define las acciones que deben tomarse en los paquetes inspeccionados. Las aplicaciones Web, sin embargo, intercambian mensajes en capas superiores, en su mayoría HTTP, que los firewalls tradicionales no están diseñados para entender. Un WAF es un firewall que ha sido diseñado específicamente para proteger aplicaciones web con la capacidad de detectar e identificar patrones que podrían derivar en un ataque contra la aplicación.

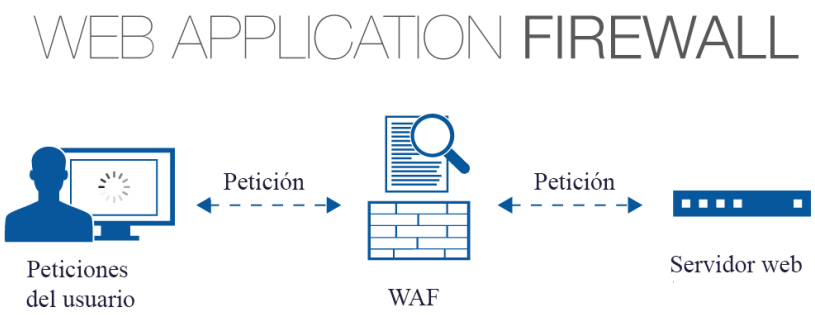


Figura 3: Como trabaja un WAF, imagen extraída de (CypherSec, 2017)

Un WAF a menudo soporta diferentes configuraciones de modelos de seguridad:

- **Modelo de seguridad positiva**
  Estos WAF por defecto deniegan todas las transacciones y solamente acepta las que identifica como seguras. Para identificar si una transacción es segura, el WAF consulta una serie de reglas que se definen de antemano, ya sea por auto-aprendizaje o una configuración manual. La ventaja es que no depende de ningún tipo de actualización y a su vez, protege a la aplicación de ataques desconocidos. Como desventaja es que son propensos a detectar falsos positivos y necesitan un proceso de aprendizaje.

- **Modelo de seguridad negativa**
  En este modelo el WAF acepta todas las transacciones y solamente deniega las que detecta como un posible ataque. Una de las desventajas es que es muy dependiente de las actualizaciones y bases de firmas de posibles ataques. A pesar de ellas estos tipos de WAF suelen ser fáciles de administrar.

ModSecurity es un WAF estándar de facto en la comunidad abierta. Es de código abierto, flexible y extensible. Se puede utilizar para controlar, monitorear y registrar el tráfico desde y hacia aplicaciones web y tiene dos modos de funcionamiento detección y prevención. En el primer modo, se generan registros para cada ataque potencial detectado y se utiliza para supervisar reglas específicas. Normalmente, este modo se utiliza cuando se añaden nuevas reglas y se controlan los falsos positivos. El segundo modo es cuando el WAF es realmente útil; mediante la correcta configuración de reglas y directivas es capaz de bloquear el potencial tráfico Web malicioso, hacia y desde las aplicaciones. El núcleo de ModSecurity implementa un motor de reglas flexible, que pueden ser aplicadas en cada transacción de la aplicación.

2.4. Reconocimiento de patrones en sistemas de detección de intrusos

Básicamente el problema en sistemas de seguridad informática es el siguiente: dado un cierto objeto (que puede ser un paquete de red, un archivo ejecutable, una petición a una página web, etc.) se debe clasificar automáticamente si es un ataque o no, es decir si pertenece a la clase positiva (maliciosa) o a la clase negativa (normal).

La incapacidad de detectar nuevos ataques por parte de sistemas basados en detección de firmas, ha impulsado la investigación de técnicas en detección de anomalías, hasta el punto tal que muchos de los sistemas comerciales de hoy en día, se valen de estas (Ariu, 2010).

Dichas técnicas utilizan algoritmos de reconocimiento de patrones, ya que estos permiten representar el problema de una manera estadística. El diseño del sistema no es trivial. Existen diferentes enfoques que tienen una gran dependencia con: la naturaleza de los datos con los que se cuenta y ciertos requerimientos necesarios para inducir el comportamiento del dominio en el cual se está trabajando (Chandola *et al.*, 2009).

Todo sistema de reconocimiento de patrones, trabaja en dos fases:

- En la fase de **aprendizaje o entrenamiento** se modela estadísticamente el comportamiento del dominio en el cual se está trabajando, en base a observaciones de una muestra representativa de la población objetivo.
- En la fase de **detección u operación** se utiliza el modelo inferido como clasificador para etiquetar nuevas observaciones.

Un aspecto fundamental para definir el enfoque a abordar es la disponibilidad de la etiqueta de los datos con los que se cuenta, es decir si para cada observación del conjunto de datos de entrenamiento se conoce si es un ataque o no.

A menudo el etiquetado es hecho a mano por un experto en el dominio. Por ende, resulta extremadamente costoso obtener datos etiquetados con información precisa y además representativa de todo tipo de ataques existentes. En cambio obtener datos etiquetados del comportamiento normal, es más accesible.

El comportamiento anómalo es dinámico por su naturaleza, ya que el atacante tiene que modificar continuamente su procedimiento para no ser detectado.

Visto estas posibilidades de etiquetados, para la clase positiva y negativa, surgen los siguientes abordajes:

- **Aprendizaje supervisado**. En esta modalidad se cuenta con la etiqueta de cada una de las observaciones. De aquí se distinguen dos enfoques:

  - ***Multi-class***. En el caso del abordaje *multi-class* es necesario definir una clase de tráfico normal y una o más clases de ataques para entrenar el clasificador. La gran ventaja, es que tienen una baja cantidad de falsos positivos.

    Estos métodos tienen dos limitaciones principales: sirven sólo para detectar ataques conocidos para un conjunto dado de aplicaciones web (las utilizadas para construir el conjunto de entrenamiento) y para su implementación se requieren de datos etiquetados tanto de tráfico normal como de ataques. Si bien obtener datos etiquetados de tráfico normal para un conjunto de aplicaciones web puede ser algo realizable recolectando datos durante períodos de prueba, no siempre es sencillo generar datos de ataques. Por lo que resulta difícil obtener un conjunto de entrenamiento balanceado y que no sobre-ajuste a las instancias normales.

  - ***One-class***. Técnicas que operan con el modo *one-class* entrenan con datos que se sabe que pertenecen a la clase *normal*.

    En vez de modelar todas las clases, normal y ataques, se concentran en modelar la normalidad e identificar los ataques como aquellas instancias que se alejan de la normalidad.

    El supuesto que está subyacente, es que la anomalía se considera como un ataque. Es necesario tener un set de datos que cubra todos los posibles comportamientos normales. La falta de un set de datos completo, puede llevar a una cantidad de falsos positivos no deseada, diciendo que una anomalía es un ataque, cuando en realidad tuvo un comportamiento normal, pero que no se tuvo en cuenta en el entrenamiento. Otro problema es la definición de un punto de corte a partir del cual una instancia se considera anómala. La definición de este umbral es crítica para poder tener buenos indicadores de *performance*, en particular de falsos positivos.

- **Aprendizaje no supervisado**. Esta modalidad no cuenta con observaciones etiquetadas; el objetivo es reconocer un patrón común en los datos (Chandola *et al.*, 2009). Bajo el supuesto que es mucho más frecuente observar instancias de comportamiento normal que de ataques, toda anomalía que se reconozca se va a identificar como ataque. Si no se cumple este supuesto, el modelo puede sufrir una cantidad de falsos positivos no deseada.

En (Ariu, 2010) trabajan de una manera **semi-supervisada**. Como no cuentan con un set de datos etiquetados para cada instancia, proponen en una fase de *testing* etiquetar el set de datos de una manera semi-automática. Para las aplicaciones que tienen un alto porcentaje de consultas asumen que las que tienen una baja probabilidad son ataques. Para las aplicaciones que tienen un bajo porcentaje de consultas, las etiquetan manualmente. Posteriormente entrenan un sistema de detección de anomalías supervisado de una clase. Parten de la premisa de que una pequeña porción del set de datos con el que cuentan son ataques. Por ejemplo uno de los ataques que encontraron fue:

```
/app?attr=<a href=http://x0.741.com>h</a>
```

Para dicha URI, «app» representa la aplicación y «attr» el atributo. La aplicación es vulnerable porque el atributo no tiene ninguna validación. En este caso el atacante agrega un *link* a su página (tal vez para hacerla más "popular") explotando el indexado para motores de búsqueda. Es importante notar que el administrador no notifica este ataque, simplemente porque no causa ningún daño permanente en el software.

Otro aspecto importante a la hora de definir el algoritmo a utilizar es la naturaleza de los datos. Los datos son una colección de **observaciones** (también denominadas objetos, patrones, eventos, casos, muestras, etc). Cada observación es definida utilizando un conjunto de **variables** (también denominadas atributos, características, dimensiones, etc.). Cada variable puede ser de diferente tipo, es decir *continua* (que son aquellas variables numéricas que tienen infinitos valores entre dos valores cualquiera), *categórica* (que tiene como valores un número finito de categorías posibles) o *binaria* (que solo admiten dos categorías). Cada observación puede tener una sola variable (univariada) o puede contener múltiples variables (multivariada).

2.5. Modelos Ocultos de Markov

Teoría de Modelos Ocultos de Markov (HMM: Ejemplo de recipientes y pelotas
Para ilustrar la idea que esta por detrás de los Modelos Ocultos de Markov se considera el siguiente *ejemplo 1* [1] presentado en (Rabiner, 1989) como *"The Urn and Ball Model"*. Supongamos que en una habitación hay 3 recipientes y en cada uno, una cierta cantidad de pelotas rojas, azules y amarillas (la cantidad de pelotas puede variar entre recipientes, así como también la proporción de pelotas de cada color -por ejemplo en el recipiente 1 puede haber un predominio de pelotas rojas, mientras que en el recipiente 3 igual cantidad de pelotas azules y amarillas-).

[1] *The Urn and ball model* fue introducido por Jack Ferguson y sus colegas, en conferencias de teoría de HMM

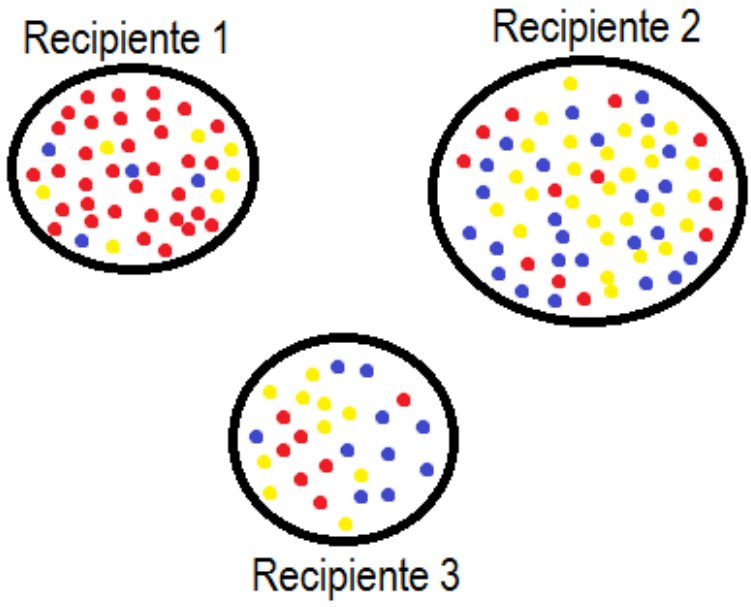


Figura 4: Ejemplo de los recipientes y las pelotas

El proceso físico generador de observaciones es el siguiente: una persona que está en otra habitación escoge un recipiente, de ahí saca una pelota y anotamos el color (primer elemento de la observación). Vuelve a reponer la pelota que sacó y en una segunda instancia elige un nuevo recipiente (con la posibilidad de escoger el mismo nuevamente), de donde saca otra pelota y anotamos el color (segundo elemento de la observación). Repitiendo este procedimiento sucesivamente, dicho proceso genera una secuencia (colores de las pelotas que fue sacando), el cual nos gustaría modelar.

En este simple ejemplo de HMM, cada recipiente representa un estado y, para cada estado, el color de la pelota escogida representa la emisión de un símbolo.

Supongamos que tenemos la siguiente observación $O = \{Roja, Roja, Amarilla, ..., Azul\}$. Físicamente se desconoce lo que pasa en la otra habitación, sabemos que hay tres recipientes y que en cada uno de ellos hay pelotas de colores. Se conoce la observación planteada, pero no se sabe nada acerca de las características de este proceso generador.

¿A que nos referimos con las características del proceso? Supongamos que por alguna razón hay más probabilidad de que el primer recipiente que escoja sea uno en particular (Ejemplo: recipiente 2). A su vez si en alguna instancia seleccioné determinado recipiente, lo más probable que el próximo en elegir sea otro específico (Ejemplo: recipiente 3 $\Longrightarrow$ recipiente 1). Y por último supongamos que en cierto recipiente es más probable que la pelota que saque sea de determinado color (Ejemplo: recipiente 2: pelota amarilla).

Modelar el proceso significa deducir todas estas características (aprendiendo con las observaciones generadas) y luego cuando tenga una nueva observación, conocer cual es la probabilidad de que haya sido generada por este modelo o calcular la secuencia de recipientes más probable que la haya generado.

2.5.1. Elementos de un HMM

Para especificar un HMM se requiere determinar [2]:

1. Una cierta cantidad de estados N, definiendo el espacio de estados como $S = \{S_1, S_2..., S_N\}$ y q_t determina el estado en tiempo t (la variable t hace referencia al *t-ésimo* elemento en una observación de T elementos). Por ejemplo si queremos indicar que el segundo elemento proviene del estado 1 denotamos $\{q_2 = S_1\}$.

   En el caso de los recipientes y las pelotas (RyP), la cantidad de estados es 3, los 3 recipientes.

2. El número de distintos símbolos observables, M. Denotando cada símbolo individual dentro del espacio de los posibles símbolos $V = \{v_1, v_2, ..., v_M\}$.

   En el caso RyP, los símbolos son diferentes colores que se pueden observar en las pelotas $\Sigma = \{Roja, Azul, Amarilla\}$.

3. Las tasas de transición de ir de un estado i hacia otro j, compuesta por una matriz A de dimensiones $N \times N$, donde cada elemento a_{ij} representa la probabilidad de ir desde el estado i al estado j al cabo de un paso.

   $$a_{ij} = P\left[q_{t+1} = S_j | q_t = S_i\right], \quad 1 \leq i, j \leq N$$

   .

   En el caso RyP es, si escogí el recipiente i, la probabilidad que el próximo recipiente que escoja sea j.

4. Las probabilidades de emisión de cada uno de los símbolos en determinado estado, representado por la matriz B en donde cada entrada tiene el valor

   $$b_j(k) = P\left[observar\ v_k\ en\ t | q_t = S_j\right], \quad 1 \leq j \leq N, \quad 1 \leq k \leq M$$

   .

   En el caso RyP es, si escogí un determinado recipiente (supongamos que el 1), qué probabilidad hay de sacar una pelota roja. Se podría escribir formalmente como $b_1(Roja)$.

5. La distribución inicial de estados π donde $\pi_i = P\left[q_1 = S_i\right], \quad 1 \leq i \leq N$.

   En el caso RyP sería la probabilidad de que el primer recipiente en escoger sea el i. Por ejemplo la probabilidad de que el primer recipiente sea el 2 es π_2.

El caso de los recipientes y las bolas puede ser descrito formalmente en términos de HMM de la siguiente manera:

[2] En las siguientes definiciones se va a utilizar la nomenclatura del tutorial (Rabiner, 1989)

	1	2	3
Probabilidades iniciales	π_1	π_2	π_3

Cuadro 1: Probabilidades iniciales π

	1	2	3
1	a_{11}	a_{12}	a_{13}
2	a_{21}	a_{22}	a_{23}
3	a_{31}	a_{32}	a_{33}

Cuadro 2: A Transiciones

	Rojo	Azul	Amarilla
1	$b_1(Roja)$	$b_1(Azul)$	$b_1(Amarilla)$
2	$b_2(Roja)$	$b_2(Azul)$	$b_2(Amarilla)$
3	$b_3(Roja)$	$b_3(Azul)$	$b_3(Amarilla)$

Cuadro 3: B Emisiones

Dados valores apropiados para N, M, las matrices A, B y el vector π, el HMM puede ser usado como el generador de una secuencia de observaciones $O = \{O_1 O_2 ... O_T\}$. En el caso RyP, es el proceso por el cual genero una observación de T pelotas.

Dicho proceso ocurre de la siguiente forma:

1. Primero se escoge un estado inicial. Cada estado tiene una determinada probabilidad de ser elegido primero, por ejemplo la probabilidad de que el estado i sea el inicial es $\pi_i = P(q_1 = S_i)$.
   En el caso RyP se escoge un recipiente del cual se saca la primer pelota
2. fijo t=1
3. Elijo $O_t = v_k$ de acuerdo a las probabilidades de observar el símbolo v_k en el estado i en que estoy posicionado, es decir $b_i(k)$
   En el caso RyP se saca una pelota del recipiente y anoto su color.
4. Paso al siguiente estado j, de acuerdo a la probabilidad de transición $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$.
   En el caso RyP se escoge otro recipiente(con la posibilidad de escoger el mismo nuevamente).
5. Incremento a t en una unidad t=t+1, si $t < T$ vuelvo al paso 3, si no termina el procedimiento.

Una completa especificación de un HMM requiere la especificación de la estructura (valores de N y M) y dada esa estructura la especificación de las distribuciones de probabilidad de las celdas de las matrices A, B y de los valores del vector π.

Por conveniencia se va a denominar dichas distribuciones de probabilidad como los parámetros del modelo $\beta = (A, B, \pi)$

2.5.2. Los tres problemas básicos de HMM a resolver:

1. Dada una observación $O = O_1 O_2 ... O_T$ y un cierto modelo específico $\lambda = (A, B, \pi)$, ¿cómo puedo calcular la probabilidad de generar esa secuencia dado el modelo $P(O|\lambda)$?

2. Dada una observacion $O = O_1O_2...O_T$ y cierto modelo $\lambda = (A, B, \pi)$, ¿cómo puedo encontrar la mejor secuencia de estados $Q = q_1q_2...q_T$ que hayan generado esa observación?

3. ¿Cómo ajusto los parámetros del modelo $\lambda = (A, B, \pi)$ de forma tal de maximizar $P(O|\lambda)$?

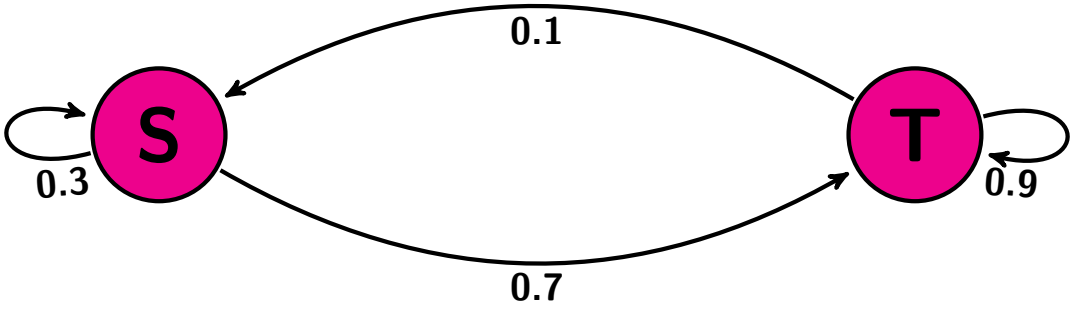
El problema 1 es conocido como el problema de **evaluación** y se puede utilizar con dos propósitos: uno es calcular la probabilidad de que una observación haya sido generada por cierto modelo, y el otro es para tener una medida de qué tan bien un cierto modelo genera determinada observación.
Supongamos que tenemos una observación pero no sabemos que modelo la generó; entonces probamos con tres modelos diferentes y nos quedamos con aquel que maximice la probabilidad de generar dicha observación.

El problema 2 es el que cubre la parte del modelo que está oculta, conocemos una observación pero queremos deducir de qué estado salió cada elemento. Para ilustrar esto nos ayudamos con el caso RyP, cuando tenemos una observación $O = \{Roja, Roja, Amarilla, Azul\}$ no sabemos cuales son los recipientes que se escogieron, es decir puede que la secuencia escogida sea "*Recipiente 1*, *Recipiente 2*, *Recipiente 2*, *Recipiente 1*", o bien todas del mismo recipiente (el 1, 2 o el 3), o diferentes combinaciones de recipientes.
Si T es la cantidad de elementos en la observación y N la cantidad de estados en el modelo, tenemos N^T posibles combinaciones de estados que pueden haber generado dicha observación. Este problema intenta identificar cual de todas las secuencias posibles de estados fue la que generó la observación.

El problema 3 cubre la parte de **aprendizaje** del modelo, es decir, la estimación de los parámetros de forma óptima. Dado un modelo con una cierta estructura (N estados y M posibles símbolos a emitir en cada estado), queremos conocer los parámetros óptimos de ese modelo $\lambda = (A, B, \pi)$.
Cuando hablamos de la estimación óptima es en referencia a las observaciones. Se buscan aquellos parámetros que maximicen la probabilidad que el modelo genere esas observaciones. Por eso se denominan observaciones de entrenamiento.

2.5.3. Solución a los tres problemas básicos de HMM

Se propone el siguiente simple *ejemplo 2* para poder ilustrar los cálculos que se van a ir haciendo. Este modelo simple tiene dos estados (S y T) y emite dos símbolos, A y B.



	S	T
Probs iniciales	0.85	0.15

Cuadro 4: Probabilidades iniciales π

	S	T
S	0.3	0.7
T	0.1	0.9

Cuadro 5: Matriz A de Transiciones

	A	B
S	0.4	0.6
T	0.5	0.5

Cuadro 6: Matriz B de Emisiones

2.5.4. Solución al problema 1

Dada una observación $O = O_1O_2...O_T$ y un cierto modelo específico $\lambda = (A, B, \pi)$, se quiere calcular la probabilidad de que el modelo genere dicha observación, es decir $P(O|\lambda)$:

Esta observación pudo haber sido generada de diferentes maneras, es decir por diferentes combinaciones de estados. Por ejemplo una posibilidad es, que todos los elementos pudieron haber sido producto de elegir sucesivamente un mismo estado en particular, o arrancar en un determinado estado y cambiar sucesivamente de estado, etc.
Una observación de T elementos, en un modelo que tiene N estados puede haber sido generada de N^T maneras diferentes.
Siguiendo con *el ejemplo 2* de los dos estados S y T, supongamos que tenemos una observación de dos elementos $O = \{A, B\}$. Ésta pudo haber sido generada de 2^2 maneras diferentes:

$$\{S_S, S_S\}, \{S_S, S_T\}, \{S_T, S_S\} \{S_T, S_T\}$$

.

La forma más sencilla de calcular $P(O|\lambda)$ es evaluar de todas las maneras en las que la observación $O = O_1O_2...O_T$ pudo haber sido generada. Esto quiere decir evaluar de todas las combinaciones posibles de T estados.
Primero consideremos una posible secuencia de estados fijos que pudieron haber generado la observación (una de las N^T maneras diferentes):

$$Q = q_1q_2...q_T.$$

La probabilidad de emitir cierto símbolo en un determinado estado es $P(O_t|q_t, \lambda) = b_{q_t}(O_t)$. Como emitir un cierto símbolo en cada uno de los estados, es independiente de lo que emitieron los demás estados; la probabilidad de generar la observación, dada esa secuencia particular de estados $Q = q_1q_2...q_T$ es:

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) = b_{q_1}(O_1)b_{q_2}(O_2)...b_{q_T}(O_T).$$

Por otro lado la probabilidad de que dicha secuencia sea elegida es:

$$P(Q|\lambda) = \pi_{q_1}a_{q_1q_2}...a_{q_{T-1}q_T}.$$

Por la ley de Bayes, la probabilidad conjunta de emitir esa observación y esa secuencia de estados es:

$$P(O, Q|\lambda) = P(O|Q, \lambda) P(Q|\lambda).$$

Hasta acá se calculó la probabilidad de que una cierta secuencia de estados fija haya generado la observación (es la probabilidad de generarla por uno de los caminos posibles). Entonces sumando para todas las combinaciones de estados posibles (todos los caminos), se calcula la probabilidad de generar la observación.

$$P(O|\lambda) = \sum_{q_1q_2\cdots q_T} P(O|Q, \lambda) P(Q|\lambda) = \sum_{q_1q_2\cdots q_T} \pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)...a_{q_{T-1}q_T}b_{q_T}(O_T).$$

La cantidad de operaciones requeridas para calcular esa probabilidad es del orden de $N^T 2T$ cálculos, precisamente $(2T-1)N^T$ multiplicaciones, y $N^T - 1$ sumas.
Para una observación con 100 elementos (T=100) y un modelo de 5 estados (N=5), la cantidad de operaciones requeridas es del orden de 10^{72}.

Se propone, por lo tanto, un procedimiento más eficiente para resolver dicho problema, con menos operaciones requeridas. Este procedimiento se denomina *Forward*

.

Se considera la variable ***forward*** $\alpha_t(i)$ definida como:

$$\alpha_t(i) = P\left(\{O_1 O_2 ... O_t\}\ ,\ \{q_t = S_i\}\,|\lambda\right) \tag{1}$$

Esta variable representa la probabilidad de haber observado la secuencia $O_1 O_2 ... O_t$, y el estado S_i en el tiempo t, dados los parámetros del modelo.

$\alpha_t(i)$ se puede resolver inductivamente:

1. el paso inicial es:
$$\alpha_t(i) = \pi_i b_i(O_1) \qquad \{1 \leq i \leq N\}$$

2. el paso inductivo es:
$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(j) a_{ji}\right] b_i(O_{t+1}) \qquad \{1 \leq i \leq N \quad ,\ \ 1 \leq t \leq T-1\}$$

3. El paso final es:
$$P\left(O|\lambda\right) = \sum_{i=1}^{N} \alpha_T(i)$$

Dicho proceso se puede ir calculando hacia delante por intermedio de una matriz, en donde cada fila representa un estado y cada columna el valor de la variable forward para cada tiempo.

Podemos ir construyendo una matriz con los elementos de la variable **forward** de modo de ilustrar su construcción.

En la siguiente matriz en cada fila están los N posibles estados y en cada columna el valor de la variable **forward** para cada tiempo t.

	$\alpha_1(.)$	$\alpha_2(.)$	...	$\alpha_T(.)$
1	$\pi_1 b_1(O_1)$	$[\alpha_1(1)a_{11} + ... + \alpha_1(N)a_{N1}]\, b_1(O_2)$	...	$[\alpha_{T-1}(1)a_{11} + ... + \alpha_{T-1}(N)a_{N1}]\, b_1(O_T)$
...	...	...	...	...
N	$\pi_N b_N(O_1)$	$[\alpha_1(1)a_{1N} + ... + \alpha_1(N)a_{NN}]\, b_N(O_2)$	...	$[\alpha_{T-1}(1)a_{1N} + ... + \alpha_{T-1}(N)a_{NN}]\, b_N(O_T)$
	t=1	t=2	...	t=T

Cuadro 7: Método iterativo de **forward**

El paso inicial es la probabilidad conjunta de arrancar en el *i-ésimo* estado y a su vez emitir el primer símbolo en él. Esto se calcula para todos los estados, como se detalla en la segunda columna $\alpha_1(i)$.

El segundo paso, el inductivo, es el corazón del método. Es la probabilidad de que cierto estado i haya generado el elemento O_{t+1}, cualquiera haya sido el estado generador del elemento anterior O_t.
Por ejemplo, si en la matriz miramos el elemento de la fila 2 y la columna 3, representa la probabilidad de emitir el segundo elemento O_2 en el estado 1, habiendo partido de cualquier otro estado. Este elemento tiene N sumandos, cada uno con la probabilidad de haber arrancado en un estado i de los N posibles (con probabilidad $\alpha_1(i)$), por la probabilidad de ir desde ese estado i hacia el 1 (a_{i1}). Toda esa suma se multiplica por la probabilidad de emitir el segundo símbolo en el estado 1 (con probabilidad $b_1(O_2)$).

Este proceso inductivo va teniendo en cuenta todas las combinaciones de estados pasadas hasta llegar al estado i, que se está evaluando en el tiempo t ($\alpha_t(i)$).
Entonces en el paso final (última columna de la matriz del cuadro 7), en cada fila se calcula la probabilidad de terminar en ese i-ésimo estado, cualquiera haya sido la combinación de estados anteriores. Sumando en todos los posibles estados (suma de la última columna) se llega a la probabilidad de haber pasado por todas las combinaciones posibles de estados.

$$P\,(O|\lambda) = \alpha_T(1) + \alpha_T(2) + ... + \alpha_T(N)$$

La cantidad de operaciones requeridas para calcular esta probabilidad es del orden de N^2T, se requieren $N(N+1)(T-1)+N$ multiplicaciones y $(N-1)(T-1)N+N$ sumas.

Considerando una observación con 100 elementos y 5 estados, con el método **forward**, se necesitan aproximadamente 2500 operaciones en comparación con las 10^{72} requeridas sin el método.

Para fijar ideas, se propone seguir con el *ejemplo 2*. Se quiere calcular la probabilidad de que una observación $O = \{A, B, B, A\}$ haya sido generada por el modelo especificado, es decir calcular $P\,(O|\lambda)$.

En la siguiente matriz se anotan los sucesivos cálculos:

	$\alpha_1(.)$	$\alpha_2(.)$	$\alpha_3(.)$	$\alpha_4(.)$
S	$0{,}85 * 0{,}4 = 0{,}340$	$0{,}06570$	$0{,}0209910$	$0{,}00618822$
T	$0{,}15 * 0{,}5 = 0{,}075$	$0{,}15275$	$0{,}0917325$	$0{,}04862648$
	t=1	t=2	t=3	t=4

Cuadro 8: método iterativo de **forward** en el ejemplo 2

Los $\alpha_1(i)$ iniciales se calculan como la probabilidad de iniciar en el estado i por la probabilidad de emitir en ese estado el primer símbolo: Como se ve en la tabla $\alpha_1(S) = \pi_S * b_S(A) = 0{,}85 * 0{,}4 = 0{,}340$, este valor es la probabilidad de arrancar en el estado S y ahí emitir una A (que es el primer elemento $O_1 = A$). El mismo cálculo también se hace para la probabilidad de que el primer estado sea el T $\alpha_1(T) = \pi_T * b_T(A) = 0{,}15 * 0{,}5 = 0{,}075$.

Los demás elementos se calculan por intermedio del paso inductivo. Por ejemplo para calcular la probabilidad de que el segundo elemento de la observación ($O_2 = B$) haya sido generado por el estado T

$$\alpha_2(T) = [\alpha_1(S)a_{ST} + \alpha_1(T)a_{TT}]\, b_T(B)$$
$$= [0{,}340 * 0{,}7 + 0{,}075 * 0{,}9] * 0{,}5 = 0{,}15275$$

La interpretación es la siguiente; la probabilidad de haber encontrado el segundo elemento en el estado T pudo haber sido generada de dos maneras:

- partiendo del estado S con probabilidad $\alpha_1(S) = 0,340$ y luego pasar al estado T con probabilidad $a_{ST} = 0,7$
- partiendo del estado T con probabilidad $\alpha_1(T) = 0,075$ y luego pasar al estado T con probabilidad $a_{TT} = 0,9$

Luego de estar en el estado T (por cualquiera de los dos caminos), emitir una B con probabilidad $b_T(B) = 0,5$.

En cada paso se toman en cuenta todos los posibles estados antecesores y a su vez los posibles estados antecesores también cumplen con dicha propiedad.
Cuando se llega al último paso hay dos opciones, o el último elemento fue generado por el estado S con probabilidad $\alpha_4(S)$ o fue generado por el estado T con probabilidad $\alpha_4(T)$. En cualquiera de los dos casos contemplan todas las secuencias posibles hasta el momento, por lo tanto sumando estas dos probabilidades calculamos la probabilidad de generar la observación $O = \{A, B, B, A\}$ (para todas las posibles secuencias de estados generadoras).

$$P\left(\{A,B,B,A\}\,|\lambda\right) = \alpha_4(S) + \alpha_4(T) = 0{,}00618822 + 0{,}04862648 = 0{,}0548147$$

De manera similar se considera la variable ***Backward***, que si bien no es necesaria para resolver el problema 1, será utilizada en la resolución del problema 3.

Se define la variable ***Backward*** como:

$$\beta_t(i) = P\left(\{O_{t+1}O_{t+2}...O_T\} \mid \{q_t = S_i\}\ ,\ \lambda\right).$$

Su interpretación es la siguiente; dado que se emitió el símbolo O_t en el estado i, la variable **Backward** es la probabilidad de emitir la secuencia parcial restante de elementos $O_{t+1}O_{t+2}...O_T$, cualquiera sea la secuencia parcial de estados $q_{t+1}q_{t+2}...q_T$ que la puedan haber generado.

Tambíen se puede resolver inductivamente:

1. el paso inicial es:
$$\beta_T(i) = 1, \qquad \{1 \le i \le N\}.$$
2. el paso inductivo:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1})\beta_{t+1} \qquad \{1 \le i \le N \quad , \quad t = T-1, T-2, ..,1\}.$$

Dicho proceso se puede ir calculando hacia atrás por intermedio de una matriz, en donde cada fila representa un estado y cada columna el valor de la variable backward para cada tiempo.

A continuación se indica la manera de construir una matriz con los elementos de la variable **Backward**.

En la siguiente matriz en cada fila están los N posibles estados y en cada columna el valor de la variable **backward** para cada tiempo t.

	$\beta_1(.)$	...	$\beta_{T-2}(.)$	$\beta_{T-1}(.)$	$\beta_T(.)$
1	$\beta_1(1)$	...	$a_{11}b_1(O_{T-1})\beta_{T-1}(1) + ... + a_{1N}b_N(O_{T-1})\beta_{T-1}(N)$	$a_{11}b_1(O_T)1 + ... + a_{1N}b_N(O_T)1$	1
...	...	...	...	...	...
N	$\beta_1(N)$	...	$a_{N1}b_1(O_{T-1})\beta_{T-1}(1) + ... + a_{NN}b_N(O_{T-1})\beta_{T-1}(N)$	$a_{N1}b_1(O_T)1 + ... + a_{NN}b_N(O_T)1$	1
	t=1	...	t=T-2	t=T-1	t=T

Cuadro 9: método iterativo de **backward**

En el paso inicial, arbitrariamente se fija $\beta_T(i) = 1$ para todos los estados. En el siguiente paso se calcula $\beta_{T-1}(i)$ para todos los i estados posibles, ¿cual es la interpretación de esta variable $\beta_{T-1}(i)$? Dado que se emitió el elemento O_{T-1} en el estado i, es la probabilidad de emitir el elemento restante O_T cualquiera sea el estado q_T que lo puedan haber generado.
Por ejemplo en la fila 1 y columna $T-1$ (penúltima) de la matriz, se calcula la variable $\beta_{T-1}(1)$ que, dado que se emitió el penúltimo elemento en el estado 1, es la probabilidad de emitir el ultimo elemento en alguno de los estados.
Esa probabilidad se calcula de la siguiente manera; dado que se emitió O_{T-1} en el estado 1, se suman las probabilidades conjuntas de ir a cada uno de los N estados y ahí emitir O_T . Cada sumando tiene la probabilidad de ir desde el estado 1 hacia el estado i a_{1i}, por la probabilidad de emitir el último elemento ahí $b_i(O_T)$

En cada paso ademas de sumar las probabilidades conjuntas de ir a cada uno de los N estados, y ahí emitir el siguiente símbolo, viene la parte recursiva, que es el corazón del algoritmo, ya que en cada sumando se asume que ahí se emitió el siguiente símbolo, se calcula la probabilidad de emitir los subsiguientes elementos restantes, en cualquiera de los estados posibles.

Para fijar ideas, se propone seguir con el *ejemplo 2*, y una observación $O = \{A, B, B, A\}$ generada por el modelo.
En la siguiente matriz se anotan los sucesivos cálculos:

	$\beta_1(.)$	$\beta_2(.)$	$\beta_3(.)$	$\beta_4(.)$
S	0,133143	0,2561	0,47	1
T	0,127281	0,2487	0,49	1
	t=1	t=2	t=3	t=4

Cuadro 10: método iterativo de **backward** en el ejemplo 2.

En el paso inicial, arbitrariamente se fija $\beta_4(S) = \beta_4(T) = 1$. En el siguiente paso se calcula $\beta_3(i)$, por ejemplo $\beta_3(S)$, que calcula la probabilidad de emitir el ultimo elemento $O_4 = A$, en cualquiera de los dos estados, dado que el tercer elemento $O_3 = B$ se emitió en el estado S.

2.5.5. Solución al problema 2

A diferencia del problema 1, el cual tiene una única solución, para resolver el problema 2 existen una variedad de maneras.

Una de las formas es encontrar el estado q_t, que es individualmente más probable. Para implementar esta solución se define la variable :

$$\gamma_t(i) = P\left(\{q_t = S_i\}\,|O, \lambda\right)$$

Dada una observación $O = \{O_1, ..., O_T\}$ y un modelo de parámetros λ, es la probabilidad de que el elemento O_t de la observación se emita en el estado S_i, es decir $\{q_t = S_i\}$.

Intuitivamente tiene la siguiente interpretación; dado que se puede generar la observación $O = \{O_1, ..., O_T\}$ de N^T combinaciones de estados posibles (para un modelo de N estados), solo me quedo con aquellas en las que en el tiempo t está el estado S_i, es decir $\{q_t = S_i\}$.
Es la probabilidad de observar el elemento O_t en el estado S_i; cualquiera hayan sido los estados $q_1, ...q_{t-1}$ antecesores que generaron la secuencia parcial de elementos $O_1, ...O_{t-1}$ y cualquiera hayan sido los estados $q_{t+1}, ...q_T$ predecesores que generaron la secuencia parcial de elementos $O_{t+1}, ...O_T$.

Por el teorema de Bayes se tiene que:

$$P\left(\{q_t = S_i\}\,|\,\{O_1, ..., O_T\}\ ,\ \lambda\right) = \frac{P\left(\{O_1, ..., O_T\}\ ,\ \{q_t = S_i\}\,|\lambda\right)}{P\left(\{O_1, ..., O_T\}\,|\lambda\right)}$$

El término que está en el numerador, utilizando la independencia condicional, se puede escribir de la siguiente manera:

$$P\left(\{O_1, ..., O_T\}\ ,\ \{q_t = S_i\}\,|\lambda\right) = P\left(\{O_1 O_2 ... O_t\}\ ,\ \{q_t = S_i\}\,|\lambda\right)\ P\left(\{O_{t+1} O_{t+2} ... O_T\}\ |\ \{q_t = S_i\}\ ,\ \lambda\right)$$

Dicha probabilidad se puede expresar en términos de las variables **forward** y **backward**:

$$P\left(\{O_1, ..., O_T\}\ ,\ \{q_t = S_i\}\,|\lambda\right) = \alpha_t(i)\beta_t(i) \tag{2}$$

Sumando esta probabilidad en todos los N estados posibles, nos da la probabilidad de generar la secuencia en cualquier combinación de estados.

$$P\left(\{O_1, ..., O_T\}\,|\lambda\right) = \sum_{i=1}^{N} P\left(\{O_1, ..., O_T\}\ ,\ \{q_t = S_i\}\,|\lambda\right) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)$$

Calculando la variable $\gamma_t(i)$ en términos de las variables **forward** y **backward**:

$$\gamma_t(i) = P\left(\{q_t = S_i\}\,|O\ ,\ \lambda\right) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

Entonces se puede encontrar la secuencia de estados más probable, hallando el estado q_t, que es individualmente más probable:

$$q_t = \underset{i}{\text{argmáx}}\,[\gamma_t(i)] \qquad \{1 \leq t \leq T\}$$

Se observa que encontrar la secuencia de estados más probable, con este método, puede llevar a algunas inconsistencias, ya que en cada tiempo se encuentra el estado individualmente más probable, cualquiera hayan sido los estados antecesores y cualquiera sean los estados predecesores.
Puede pasar que en cierto tiempo t el estado más probable sea el S_i y en t+1 sea S_j, pero que $a_{ij} = 0$ (el estado i no se conecta con el j).

Dada una observación $O = \{O_1, ..., O_T\}$ y un modelo de parámetros λ, para resolver este problema, una primera solución es encontrar la mejor combinación de estados (de las N^T posibles) generadora de esa observación, es decir la combinación de estados que hace que la probabilidad de encontrar la observación por intermedio de esos estados sea máxima.

El procedimiento propuesto por **Viterbi**, para resolver este problema computacionalmente de manera eficiente es el siguiente:

Se define la variable:

$$\sigma_t(i) = \underset{q_1, q_2.., q_{t-1}}{\text{máx}}\, P\left[q_1, .., q_{t-1}, q_t = S_i\ , O_1, ..., O_t|\lambda\right] \tag{3}$$

Es la probabilidad conjunta de generar el elemento O_t en el estado S_i y los $O_1, ..., O_{t-1}$ elementos generarlos en la secuencia de estados más probable $q_1, .., q_{t-1}$, es decir en el camino más probable.

Por inducción:

$$\sigma_{t+1}(j) = \underset{i}{\text{máx}}\,[\sigma_t(i)a_{ij}]\, b_j(O_{t+1})$$

Para encontrar la mejor secuencia, se tiene que hacer el siguiente procedimiento:

- Inicialización:

$$\sigma_1(i) = \pi_i b_i(O_1) \qquad 1 \leq\ i \leq N$$
$$\Psi_1(i) = 0$$

- Recursión:

$$\sigma_t(j) = \underset{1 \leq\ i \leq N}{\text{máx}}\,[\sigma_{t-1}(i)a_{ij}] \cdot b_j(O_t) \qquad 1 \leq\ j \leq N \qquad 2 \leq\ t \leq T-1$$
$$\Psi_t(j) = \arg \underset{1 \leq\ i \leq N}{\text{máx}}\,[\sigma_{t-1}(i)] \qquad 1 \leq\ j \leq N \qquad 2 \leq\ t \leq T-1$$

- Finalización:

$$P^* = \underset{1 \leq\ i \leq N}{\text{máx}}\,[\sigma_T(i)]$$

$$q_T^* = \arg \underset{1 \leq\ i \leq N}{\text{máx}}\,[\sigma_T(i)]$$

- Se busca la secuencia de estados, volviendo hacia atras:

$$q_t^* = \Psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, ..., 1.$$

Luego de presentar el algoritmo de **Viterbi**, se propone utilizarlo en el *ejemplo 2*. Se quiere calcular la secuencia de estados más probable que hayan generado la observación $O = \{A, B, B, A\}$.

En la siguiente matriz se anotan los sucesivos cálculos:

	$\sigma_1(.)$	$\sigma_2(.)$	$\sigma_3(.)$	$\sigma_4(.)$
S	$0{,}85 * 0{,}4 = 0{,}340$	$0{,}0612\ \Psi_2(S) = S$	$0{,}0110\ \Psi_3(S) = S$	$0{,}0021\ \Psi_4(S) = T$
T	$0{,}15 * 0{,}5 = 0{,}075$	$0{,}119\ \Psi_2(T) = S$	$0{,}0535\ \Psi_3(T) = T$	$0{,}0240\ \Psi_4(T) = T$
	t=1	t=2	t=3	t=4

Cuadro 11: Aproximación por **Viterbi** en el ejemplo 2.

Entonces la probabilidad de generar la observación $O = \{A, B, B, A\}$ por el camino más probable es $P^* = 0{,}0240$.

- El estado generador del último elemento $q_4^* = T$
- $q_3^* = \Psi_4(q_4^*) = \Psi_4(T) = T$
- $q_2^* = \Psi_3(q_3^*) = \Psi_3(T) = T$
- $q_1^* = \Psi_2(q_2^*) = \Psi_2(T) = S$

Entonces para recrear la secuencia de estados (más probable) generadores de la cadena $O = \{A, B, B, A\}$ es :

$$Q = \{q_1^* = T, q_2^* = T, q_3^* = T, q_4^* = S\}$$

2.5.6. Solución al problema 3 con el algoritmo de Baum-Welch

Baum-Welch es un algoritmo de Esperanza-Maximización o algoritmo EM (*Expectation-Maximization*), empieza asignando valores iniciales a los parámetros (transiciones y emisiones) y en cada iteración calcula el número esperado de veces en que cada una de estas probabilidades es utilizada por el modelo para generar los datos de entrenamiento (paso E). Luego se actualizan las probabilidades para que la verosimilitud de los valores esperados sea máxima (paso M). Para resolver dicho problema primero se definen las siguientes variables:

- Se estima la probabilidad de estar en cierto estado en el tiempo t

$$\gamma_t(i) = P\left(q_t = S_i | O, \lambda\right)$$

. Expresada en términos ***forward-backward***

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

- Se estima la probabilidad de pasar del estado i al estado j en el tiempo t.

$$\xi_t(i,j) = P\left(q_t = S_i, q_{t+1} = S_j | \lambda, O\right)$$

Expresada en términos de ***forward-backward***

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{4}$$

Notar que:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Para calcular el número esperado de transiciones desde i a j:
$\sum_{t=1}^{T-1} \xi_t(i,j) =$ número esperado de transiciones desde i a j.

Para calcular el número esperado de transiciones desde i hacia algún otro estado:
$\sum_{t=1}^{T-1} \gamma_t(i) =$ número esperado de transiciones desde i hacia algún otro estado.

Utilizando las fórmulas mencionadas ahora se hace la re-estimación:

- La probabilidad de estar en el estado i en tiempo inicial

$$\widehat{\pi} = \gamma_1(i)$$

- La probabilidad de pasar del estado i al estado j

$$\widehat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

- La probabilidad de emitir el símbolo v_k en el estado i

$$\widehat{b}_i(k) = \frac{\sum_{t=1}^{T} \gamma_t(i) \cdot \delta(O_t, v_k)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Donde el delta de Kronecker $\delta(x, y)$ es 1 si $x = y$ y 0 en otro caso.

2.6. Autómatas

Para presentar los autómatas finitos se va a utilizar el artículo de (Hopcroft *et al.*, 2008). Un autómata finito es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto Σ. Consiste en un conjunto finito de estados y un conjunto de transiciones entre esos estados, que dependen de los símbolos de la cadena de entrada. El autómata finito acepta una cadena x si la secuencia de transiciones correspondientes a los símbolos de x conduce desde el estado inicial a un estado final o terminal.

2.6.1. Autómata finito determinista

El autómata "determinista" (denominado AFD por sus siglas) hace referencia al hecho de que para cada entrada sólo existe un y sólo un estado al que el autómata puede hacer la transición a partir de su estado actual.

Un *autómata finito determinista* consta de:

- Un conjunto finito de estados, a menudo designado como Q.
- Un conjunto finito de símbolos, a menudo designado como Σ.
- Una *función de transición* que toma como argumento un estado y un símbolo de entrada y devuelve un estado.
- Uno de los estados de Q que es el *inicial*.
- Un subconjunto de estados de Q que son los *finales* o de *aceptación*.

El AFD "acepta" una secuencia de símbolos y el lenguaje del AFD es el conjunto de todas las cadenas que acepta.

Por ejemplo se puede especificar un AFD que acepte unicamente todas las cadenas de ceros y unos que contengan la secuencia 01 en cualquier posición de la cadena.

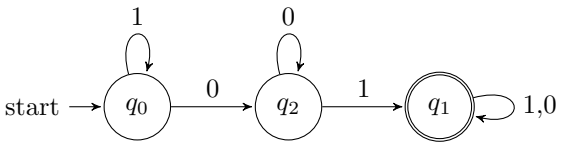


Figura 5: Diagrama de transiciones del AFD que acepta todas las cadenas que contienen la subcadena 01

En este autómata el alfabeto de entrada es $\Sigma = \{0, 1\}$. Tiene un conjunto de estados, Q, siendo el q_0 el estado inicial y el q_1 el estado de aceptación.

Si en el estado inicial se observa un 1 el autómata se mantiene en el estado q_0. Recién cuando se observa un 0, se está en condiciones de observar la secuencia 01, por lo tanto se pasa al estado q_2.

Estando en el estado q_2 si se observa un 1 se pasa al estado q_1, que es un estado de aceptación, por lo tanto se acepta cualquier cadena que se lea de ahí en adelante. De lo contrarío se mantiene en el estado q_2 hasta observar un 1.

Por ejemplo la secuencia 1000011 es aceptada por dicho autómata, ya que el último elemento termina en el estado q_1 que es de aceptación.

También existen los autómatas "no deterministas" (AFND) que tienen la capacidad de estar en varios estados a la vez. Estos autómatas aceptan los mismos lenguajes regulares que los AFD. Sin embargo a menudo los AFND son más compactos y faciles de diseñar. Además siempre es posible convertir un AFND en un AFD.

3. Estado del Arte

3.1. Aprendizaje supervisado multi-class

En (Goseva-Popstojanova *et al.*, 2012) se utilizan técnicas de **aprendizaje supervisado multi-class**, enfocándose en **caracterizar la actividad maliciosa**. Con el fin de tener datos de ataques etiquetados implementaron un *honeypot* [3]. Su objetivo es clasificar automáticamente tráfico malicioso HTTP, basándose en la hipótesis de que las actividades maliciosas tienen diferentes patrones de comportamiento. Es un enfoque distinto a métodos estándar de clasificación que se limitan a discriminar entre actividades maliciosas y no maliciosas.

Utilizando un set de datos que solo tiene sesiones Webs maliciosas, les permitió concentrarse en las características de ataques, sin estar afectados por el ruido de las actividades normales. Además utilizan un enfoque para seleccionar un subconjunto de las características más importantes (*feature selection*), *rankeando* las características con el método de ***Ganancia de información*** (en donde seleccionan las características que producen el mayor decremento de la impureza, medido por el decremento de la entropía). Para seleccionar las características utilizan una selección secuencial de avance (*Forward*), en el que se va agregando una característica en cada paso.

Es un enfoque muy completo donde utilizan varios clasificadores como: árboles de decisión C.45, método de inducción de reglas basadas en arboles de decisión parciales (PART) y máquinas de soporte vectorial multi-clases (en inglés *multiclass Support Vector Machines* (SVM)). Luego comparan el desempeño y la efectividad de cada uno de los clasificadores.

En (Gallagher y Eliassi-Rad, 2009) se puede encontrar otro **enfoque supervisado multi-class**, que identifica y **clasifica** inyecciones de código en consultas HTTP, utilizando modelos de espacio vectorial (en inglés *vector space model*) (Salton *et al.*, 1975), usado comúnmente en Búsqueda y Recuperación de Información (*Information Retrieval*). Dicho método se basa en calcular las frecuencias de palabras repetidas, para tratar de clasificar las consultas. Los autores construyen un clasificador que permita etiquetar una consulta HTTP como valida o ataque, en el caso de ataque el clasificador también identifica la tipología del mismo.

Los autores utilizaron este enfoque en un conjunto de datos proporcionado en el experimento ECML/PKDD 2007; una competencia creada en conjunto por la 18^{th} European Conference on Machine Learning (ECML) y la 11^{th} European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD). Los datos proporcionados por la competencia contenían el texto completo de consultas con el protocolo HTTP (incluyendo method, protocol, uri, query, headers y body), cada consulta fue asignada en una de las 8 posibles clases (es decir "válido" + 7 tipos de ataques).

3.2. Aprendizaje supervisado one-class

En (Kruegel y Vigna, 2003) investigadores de la Universidad de Santa Bárbara desarrollaron un sistema de detección de anomalías que aprende el perfil del acceso normal a la base de datos por las

[3] honeypot o sistema trampa es una herramienta de seguridad informática dispuesto en un sistema o una red para ser el objetivo de un ataque informático y así poder detectarlo y tener información del atacante y lo que está haciendo.

aplicaciones web, utilizando diferentes modelos de aprendizaje.

Analizan las consultas que hacen los usuarios al servidor de un programa web, **buscando correlaciones entre ciertas características en los parámetros de las consultas y algún tipo de actividad maliciosa o fuera de lo normal**.
El sistema utiliza diferentes modelos para aprender patrones característicos de un usuario normal (bien intencionado) en las consultas al servidor referenciado. Luego al analizar un nuevo usuario, detecta si su comportamiento se desvía del normal y en ese caso tener evidencia de un posible ataque.

El análisis se centra *logueos web*, específicamente parámetros de las consultas GET con el protocolo HTTP (utilizado para efectuar peticiones, pasando valores a los servidores o documentos activos).
Los datos de entrada utilizados son las URIs (identificador de recursos uniforme) de consultas GET exitosas. La URI se compone por una ruta a los recursos referenciados y una cadena de consulta con el símbolo "?", que se utiliza para pasar parámetros al recurso. La cadena de consulta consta de n pares de parámetros o atributos con sus respectivos valores.

El set de URIs es particionado de acuerdo a la ruta de sus recursos, por lo tanto cada programa referenciado tiene un set de consultas correspondientes .

El sistema de detección utiliza **diferentes modelos** para identificar anomalías usando un set de URIs para cada programa (es decir cada programa entrena sus modelos con un set de datos específico).
Los modelos propuestos son procedimientos para evaluar:

- **Características de los valores de los atributos de una consulta** (por ejemplo el largo del valor del atributo)
- **Características de la consulta como un todo** (por ejemplo presencia o ausencia de determinado atributo)

Los modelos propuestos por (Kruegel y Vigna, 2003) son los siguientes:

- **Modelos que analizan la consulta como un todo**
  - **Orden de los atributos**, hay ocaciones que las consultas que se hacen al servidor no son directamente invocadas por los usuarios, sino que muchas veces estos campos son llenados por programas del lado del cliente, por lo que este proceso debe ser regular, es decir en dichos casos, siempre tienen una cierta cantidad de parámetros con el mismo nombre y están ordenados de una determinada manera.
    Algunas evidencias muestran que muchos atacantes se focalizan en explotar las vulnerabilidades del programa, sin prestan demasiada atención al orden regular de los parámetros. El objetivo de este modelo es encontrar la forma normal en que se presentan los parámetros de los clientes legítimos; y en caso de que una consulta no tenga esa forma, será detectada como una anomalía.
  - **Presencia o ausencia de un atributo**, como se planteó en el modelo anterior, cuando son programas que llenan ciertos campos, los parámetros tienen un orden preestablecido, por ende siempre tienen los mismos parámetros, no debería de haber alguno de más (es decir ninguno que no se haya visto antes), ni tampoco ninguno de menos.

El objetivo de este modelo es similar al anterior, ambos estudian la consulta en su totalidad, en este caso evaluando que no falte algún parámetro o que no haya alguno adicional nunca antes visto.

- **Modelos para cada uno de los valores de los atributos de una consulta**

  - **Largo del atributo**, es de esperar que el largo de los valores de los parámetros no debería tener variaciones significativas entre consultas asociadas al mismo programa, por lo tanto, uno de los modelos plantea estimar el largo de los valores normales y detectar aquellas consultas que difieran significativamente de esos valores. Primero aproximan la media μ y la varianza σ^2 para los largos de los valores observados en el entrenamiento $l_1, l_2, ..., l_n$. Luego el modelo de decisión está basado en la *desigualdad de Chebyshev*. La probabilidad de cierto largo l puede ser calculada con la ayuda de dicha desigualdad. La cuál pone una cota superior a la probabilidad de la diferencia entre una variable aleatoria x y la media μ, es decir que dicha probabilidad sea mayor a cierto umbral t.

    $$p\left(|x-\mu| > t\right) < \frac{\sigma^2}{t^2}$$

    Sustituyendo t por la distancia entre el largo de una cierta instancia l y μ es decir $|l-\mu|$. Se obtiene una cota superior a la probabilidad de que el largo del parámetro se desvíe más que la instancia actual. La probabilidad resultante $p(l)$ queda expresada de la siguiente forma:

    $$p\left(|x-\mu| > |l-\mu|\right) = p(l) < \frac{\sigma^2}{(l-\mu)^2}$$

    que indica cual es la probabilidad, de que la distancia a la media, sea más grande de la que se observa en la instancia l.

  - **Distribución de los caracteres del atributo**, el algoritmo propuesto se basa en analizar la frecuencia de los caracteres en un atributo. Dicho modelo captura el concepto de consulta "normal"basándose en la distribución de los mismos.
    Es esperable que ante consultas legítimas, la frecuencia de los caracteres vaya decreciendo lentamente. Sin embargo, en algunos ataques, es probable que se utilice de forma excesiva algún caracter particular.

  - **Token Finder**, el propósito de este modelo es determinar cuando valores de un cierto parámetro, pertenecen a una lista numerada preestablecida. En caso afirmativo, se plantea un método para inferirla. Al analizar una nueva observación, se evalúa si el valor de cierto parámetro pertenece a esa lista o no (en caso de que no pertenezca se considera como anómala).

  - **Inferencia de la estructura**, para el propósito del modelo la estructura del valor de un atributo, es la gramática regular que describen todos los valores normales y legítimos. Es decir, la gramática generadora.
    Cuando se analiza la estructura del atributo de una consulta, la gramática resultante debe producir al menos todos los ejemplos de entrenamiento (como mínimo). Lamentablemente no hay una única gramática que pueda ser derivada de los elementos de entrada. Cuando no se tiene elementos negativos (es decir elementos que no se derivan de la gramática), los

dos extremos son; crear una gramática que contenga solamente los datos de entrenamiento, o una gramática que permita producciones de cadenas arbitrariamente. En el primer caso hay demasiada simplificación, solo se van a aceptar los datos vistos en el entrenamiento, por lo tanto no se deduce ninguna nueva información. En el segundo caso hay demasiada generalización, ya que la gramática puede producir cualquier cadena arbitraria.

El enfoque propuesto es partir de una gramática que sólo genere los datos de entrenamiento y generalizarla mientras parezca "razonable", es decir **dejar de generalizar cuando se está perdiendo mucha información** estructural. Dicho criterio es guiado por modelos Markovianos y probabilidades Bayesianas. Recomiendan el artículo de (Stolcke y Omohundro, 1994) en donde se detalla la técnica utilizada.

El presente trabajo tiene como objetivo el estudio y la implementación de dicha técnica, se explicará en detalle en la Sección 4.2.

Para todos los modelos se distinguen dos etapas; una de **Aprendizaje** donde el sistema infiere el modelo del perfil normal de las consultas de cada programa y el valor del umbral que discrimina entre una consulta normal o anómala para cada uno de los modelos. Otra de **Detección** donde se identifica aquellas consultas que superen el umbral de al menos uno de los *scores* de anomalía.

En la fase de detección cada modelo inferido asigna valores de probabilidad a cada una de las consultas o a determinado atributo de la consulta; esta probabilidad refleja la ocurrencia de ciertas características de los atributos o de la consulta como un todo. Asumiendo que aquellas probabilidades bajas, son indicios de un comportamiento distinto al normal.

Se evalúa un *score* de anomalía tanto para cada atributo como para cada consulta como un todo , combinando los modelos utilizados (asignándole a cada uno un determinado peso, que se fija en la fase de aprendizaje). Esto es necesario para prevenir ataques, en donde se esconde un atributo malicioso en varios atributos normales.

En (Corona *et al.*, 2009) utilizan un enfoque muy similar al de (Kruegel y Vigna, 2003). Utilizando un conjunto de Modelos Ocultos de Markov (HMM por sus siglas en inglés) modelan:

- La secuencia de atributos de la URI, recibida con el método GET por la aplicación web.
- El valor asignado para cada atributo.

Como cada servidor web puede hospedar varias aplicaciones (en donde cada una de ellas tiene sus propios atributos) **el sistema crea un módulo específico para cada aplicación** y entrena con consultas específicas, igual que en (Kruegel y Vigna, 2003)

El objetivo es detectar ataques que exploten fallas en la validación de la entrada de los usuarios, para esto proponen modelar **dos características de las consultas**:

- La posición de cada atributo en la secuencia. Igual que en (Kruegel y Vigna, 2003) (**en donde se modela la presencia o ausencia de atributos y el orden de los atributos, modelando la consulta como un todo**).

Para modelar la secuencia de atributos utilizando HMM, consideran el valor de cada atributo como un símbolo de la secuencia.

- El valor de cada atributo en la secuencia. Por ejemplo, si típicamente un atributo recibe como valor un número, y luego se observa un caracter (o viceversa) puede ser síntoma de que algo anda mal; es un enfoque parecido a la inferencia de la estructura (**Structural Inference**) utilizado en (Kruegel y Vigna, 2003). La diferencia radica en que proporcionaron una codificación más compleja de las entradas de los atributos con respecto a la simple extracción de la secuencia de caracteres. Estudiaron varios ataques contra aplicaciones web (cve.mitre.org), y al interpretar el significado de las entradas de los atributos, plantean que típicamente los caracteres no alfanuméricos tienen mayor relevancia que los caracteres alfanuméricos.
  Los caracteres no alfanuméricos podrían utilizarse como meta-caracteres con un "significado" especial durante el procesamiento realizado por aplicaciones web.

  Por lo tanto, entienden que una distinción entre ellos (por ejemplo entre "/" y "-") es definitivamente necesaria. Por el contrario, distinguir entre dígitos o letras de un alfabeto no es útil para detectar ataques de validación de entrada. En síntesis recodifican las consultas, le asignan una "A" a los caracteres alfabéticos y una "N" a los numéricos, dejando todos los demás tal como son.

El objetivo es detectar anomalías en alguna de las características mencionadas. Cuando se evalúa una nueva observación, si para algún modelo de los entrenados, alguna característica de la consulta, tiene una probabilidad baja de haber sido generada, se sospecha de ella, igual que en (Kruegel y Vigna, 2003).

Proponen atacar dos de los tres problemas básicos de HMM para la construcción de los mismos (en el anexo 2.5 se encuentra en detalles los formalismos y los tres problemas básicos extraídos de (Rabiner, 1989))

- *Aprendizaje*. Asumiendo una cantidad de estados para cada modelo utilizan el algoritmo de Baum-Welch para entrenar los parámetros de los HMM.

- *Evaluación*. Utilizado en la fase de detección, para detectar las consultas anómalas.

Plantean una heurística diferente a (Kruegel y Vigna, 2003), asumen una cantidad de estados para cada modelo y estiman las probabilidades de transición y emisión con el algoritmo de Baum-Welch. Como el algoritmo es propenso a caer en un óptimo local (Stolcke y Omohundro, 1994, p. 4) y (Rabiner, 1989, p. 265), la solución que proponen es crear un **conjunto de HMM para cada característica e inicializarlos aleatoriamente** (Ariu, 2010, p. 37). Luego en la fase de detección, al evaluar una nueva consulta, utilizan el modelo que tenga menor probabilidad de haberla creado, si está por debajo de cierto umbral se considera anómala. Con esta solución evitan caer en un mínimo local.

El conjunto de datos utilizado es de la misma forma que en (Kruegel y Vigna, 2003) compuesto por peticiones con el método GET, extrayendo la aplicación y la consulta de la URI.

En (Vamsidhar, 2013) también se propone modelar el comportamiento normal de cada una de las

aplicaciones del servidor web para detectar anomalías. Además a diferencia de (Kruegel y Vigna, 2003) y (Corona *et al.*, 2009), si la consulta se detecta como un ataque, se anuncia el tipo del mismo.

El set de datos que utilizaron fue creado por ellos mismos, consta de inicios de sesión (*logueos*) a 61 aplicaciones web, en donde algunos ataques fueron configurados manualmente para utilizarlos en la fase de detección.

Al igual que en (Kruegel y Vigna, 2003) y (Corona *et al.*, 2009) se entrena cada aplicación específica con consultas de la misma. Utilizan tres módulos para detectar anomalías:

- **Largo del atributo**. Al igual que (Kruegel y Vigna, 2003), plantean estimar el largo de los valores normales y detectar consultas que difieren de esos valores.
- **Input Validation**. Al igual que (Kruegel y Vigna, 2003) y (Corona *et al.*, 2009) modelan la presencia o ausencia de los atributos, el orden de prioridad y los valores que tiene. Entrenan un Autómata Finito Determinístico (AFD) para cada aplicación, luego en la fase de detección, verifican que la secuencia de atributos pertenezcan a dicho autómata.
- **Detección de anomalías en los valores de los atributos**. Igual que en (Kruegel y Vigna, 2003) y (Corona *et al.*, 2009) plantean inferir la estructura de los valores de los atributos. El enfoque es parecido a (Corona *et al.*, 2009), en el cual a diferencia de (Kruegel y Vigna, 2003) codifican el parámetro recibido:

  - Utiliza la letra "A" para cada caracter del alfabeto que observaron en la cadena
  - Utiliza la letra "N" para cada dígito que observaron en la cadena
  - Para cada caracter predefinido como $=, \{., ?, \&, \_, =\}$, se deja tal cual está
  - Para los demás casos, se utiliza letra "H"

  Luego se crea un autómata, en donde se va cambiando de estado al observar cada uno de los caracteres, cuando existe una transición hacia el estado H, se asigna un punto de anomalía.

4. Marco Conceptual

4.1. Problema en el Contexto

Según el Estándar de Verificación de Seguridad en Aplicaciones (ASVS) (OWASP, 2017) la debilidad más común de seguridad de las aplicaciones web es la falla en validar apropiadamente el ingreso de datos que provienen del cliente o del ambiente antes de ser utilizados. Esta debilidad conduce a casi todas las vulnerabilidades encontradas en aplicaciones web, tales como cross-site scripting (XSS), inyecciones SQL, inyección de intérprete, ataques local/Unicode, ataques a sistemas de archivos y desbordamientos de búfers.

Uno de los ataques más comunes se da cuando un usuario malintencionado intenta inyectar un código en las consultas HTTP (como se ejemplificó en la Sección 2.2) que puede ejecutarse en el servidor Web durante el procesamiento de la misma.

Hay lenguajes esperados para cierto campo de un formulario, por ejemplo nombres de personas. Sin embargo una consulta puede contener un código SQL o XPath (o algún otro programa) con intención de ejecutarse en el lado del servidor web.

Veamos un ejemplo con una actividad de un usuario en la página de la FING (Facultad de Ingeniería, UdelaR, Uruguay). Esta página permite localizar información sobre sus docentes. El usuario debe escribir el nombre y la página le devuelve la información correspondiente. Los parámetros de esta petición con el mecanismo GET se pueden observar en la figura 6:

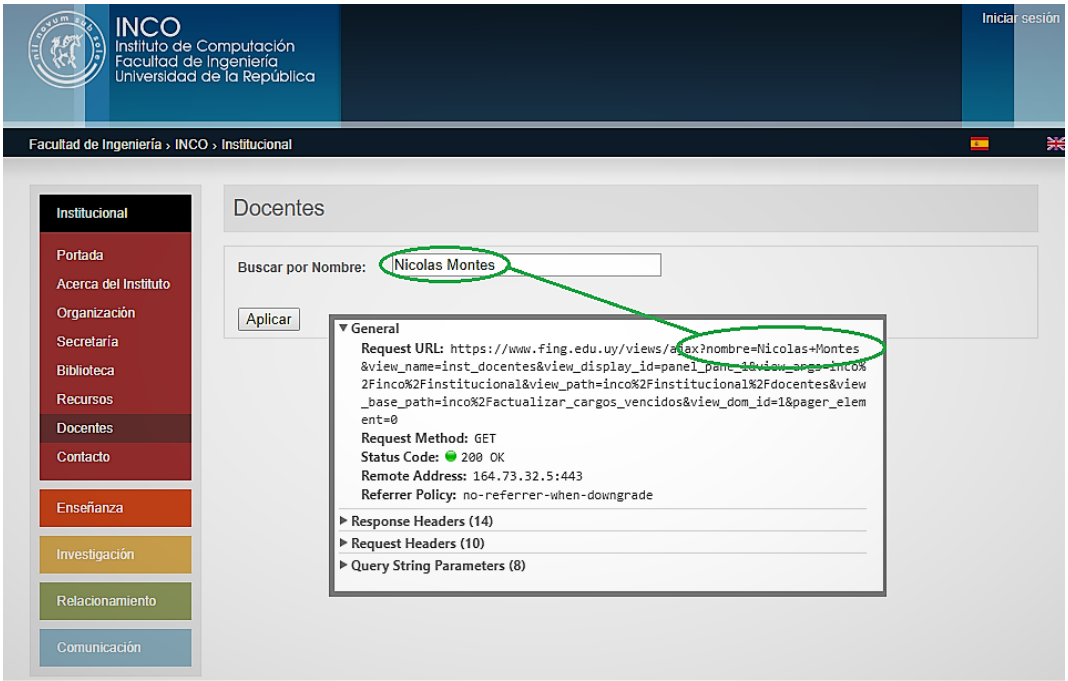


Figura 6: Ejemplo de una petición a la página

Lo esperable es que cada usuario escriba en el campo nombres de personas (palabras formadas con cadenas de caracteres). Sin embargo, un usuario malintencionado, sea éste un ser humano o un agente

computacional, puede introducir ciertas combinaciones de caracteres en dicho campo, que pueden tener otro propósito.

Supongamos que dicha aplicación tiene una vulnerabilidad al validar apropiadamente el ingreso de datos que provienen del cliente, en este contexto dicho usuario podría introducir expresiones como 'OR '1'='1 (inyectando código SQL) y de este modo modificar la consulta, comprometiendo información de la base de datos.

Este tipo de ataques puede prevenirse con la utilización de un WAF. Mediante la correcta configuración de reglas y directivas, dicho sistema es capaz de detectar y bloquear este tipo de tráfico Web malicioso.

Como se expuso en el Capítulo 3 varios investigadores han aplicado técnicas de aprendizaje automático para la detección de ataques a las aplicaciones web. Uno de los enfoques consiste en modelar el tráfico normal de la aplicación que se está protegiendo y en una siguiente fase bloquear aquellas consultas que tengan algún patrón anómalo bajo el modelo inferido.

En este trabajo se va a profundizar en una de las técnicas propuestas por (Kruegel y Vigna, 2003) denominada **Inferencia de la Estructura (Structural Inference)**. El objetivo de dicha técnica es inferir la **gramática regular** subyacente de los campos de la aplicación que se quiere proteger (utilizando datos de tráfico normal para el aprendizaje) y en una siguiente fase bloquear aquellas consultas que no sean aceptadas por dicha gramática.

La **gramática inferida** debe producir al menos todos los ejemplos vistos en el entrenamiento, pero también debe generalizar, es decir aceptar nuevas palabras no vistas antes, pero que potencialmente podrían haber sido creadas por ella.

Para inducir la **gramática regular** (Kruegel y Vigna, 2003) proponen utilizar un algoritmo denominado **"Fusión de Modelos Bayesiana" (Bayesian Model Merging)** propuesto por (Stolcke y Omohundro, 1994) en el que se combinan formalismos de *Modelos Ocultos de Markov, Autómatas Probabilísticos e Inferencia Bayesiana* para inferir el modelo generador del lenguaje regular subyacente.

En la siguiente Sección 4.2 se va a exponer el marco teórico de dicho algoritmo. En el Capítulo 5 se van a describir detalles de la implementación y posteriormente en el Capítulo 6 se va a presentar la experimentación en dos estudio de casos.

El primer estudio de casos consiste en inferir un lenguaje creado artificialmente como en (Stolcke y Omohundro, 1994), a partir de un mínimo conjunto de entrenamiento.

El segundo estudio de casos trata de un prototipo que aprenda nombres de personas a partir de un conjunto de entrenamiento.

El objetivo es simular una solución para el problema planteado en el ejemplo de la figura 6 (en el que se supone que dicha aplicación presenta una vulnerabilidad). Para esto es necesario un procedimiento automático que filtre toda cadena de caracteres que no parezca ser el nombre de una persona.

La solución que se propone investigar es utilizando el algoritmo **"Fusión de Modelos Bayesiana" (Bayesian Model Merging)**, en la cual partiendo de observaciones de tráfico normal se debe crear un modelo que acepte nuevos nombres sin sobre-generalizar, es decir que rechace cadenas de caracteres arbitrarias que no sean nombres (como por ejemplo la cadena 'OR '1'='1).

4.2. Técnica automática para inferir la estructura de HMM

En esta Sección se describe una técnica para **inducir la estructura de HMM** a partir de datos, basada en la estrategia **"Best-first Model Merging for Hidden Markov Model Induction"** propuesta por Stolcke y Omohundro (Stolcke y Omohundro, 1994).

HMM es un método muy popular, utilizado para modelar secuencias estocásticas con una estructura finita subyacente. Son algoritmos muy utilizados en muchas de las áreas de reconocimiento de patrones. Los primeros usos fueron en el área de criptoanálisis y hoy en día se utiliza en diferentes dominios tales como reconocimiento de voz, etiquetado gramatical, modelado de secuencias biológicas, etc.

HMM caracteriza una clase de procesos estocásticos que tienen una estructura Markoviana finita subyacente, pero que solo puede ser observada indirectamente (por eso la denominación estados ocultos -hidden-).

Dada una observación, es decir una secuencia de símbolos del alfabeto, la probabilidad de observar cada símbolo depende del estado subyacente, a su vez la probabilidad de estar en cada estado depende del estado predecesor inmediato, es un proceso doblemente estocástico (en la sección 2.5 se detalla el funcionamiento de HMM, mediante un resumen del tutorial de Rabiner (Rabiner, 1989)).

Asumiendo un modelo con determinada cantidad de estados, algoritmos implementados (ampliamente conocidos) permiten estimar las probabilidades de transición de ir de un estado hacia otro, y las probabilidades de emisión, es decir la probabilidad de emitir determinado símbolo dado que se está en cierto estado.

Desde otro punto de vista HMM puede ser visto como una generalización de un Autómata Finito No Determinista (AFND). Al igual que los AFND, los HMM tienen un número finito de estados, incluyendo el estado inicial, el final y un sub-conjunto de estados no terminales. Todos se pueden conectar por intermedio de transiciones. En cada estado se puede emitir un símbolo de un alfabeto discreto. Por lo tanto generan (o aceptan) cadenas de caracteres (formadas por símbolos del alfabeto), a través de un camino aleatorio entre el estado inicial y el final. Además HMM asigna probabilidades a las producciones que puede generar, computando las diferentes probabilidades de transiciones y emisiones necesarias para generar dicha cadena.

En (Dupont *et al.*, 2005) presentan la relación entre Automáta Probabilístico (AP) y HMM, ambos modelos tienen estados y símbolos. En cada estado se emite un cierto símbolo de un alfabeto (discreto) y posteriormente se puede conectar con otro estado del modelo.

Esta relación es de suma importancia, ya que permite aplicar una cantidad de técnicas inductivas de aprendizaje estudiadas en uno de los dos formalismos y aplicarlas en el otro.

Stolcke y Omohundro proponen un algoritmo denominado **"Fusión de modelos bayesiana"** utilizado para HMM. Este enfoque no sólo se limita a estimar las probabilidades de transición y emisión dado una cierta cantidad finita de estados como los enfoques tradicionales de HMM, sino que además estima la cantidad de estados que tiene el modelo.

Utilizando el algoritmo para inducir el HMM visto como un AFND a partir de datos, se está induciendo el autómata generador del lenguaje regular subyacente.

4.2.1. Definiciones

Un Modelo Markoviano de estados ocultos (HMM) con emisiones discretas, visto como un Autómata Finito No Determinista Estocástico, se especifica por: un conjunto de estados Q, un alfabeto Σ, un estado inicial q_I, un estado final q_F y un set de ***parámetros de probabilidad*** compuestos por:

- ***Probabilidades de Transición*** $P(q_i \to q_j)$ que representan la probabilidad de pasar del estado q_i al estado q_j, $\forall\ \{q_i, q_j\} \in\ Q$.
- ***Probabilidades de Emisión*** $P(q_i \uparrow \sigma)$ que representan la probabilidad de emitir el símbolo σ en el estado q_i, $\forall\ q_i \in\ Q\ y\ \sigma \in \Sigma$.

La **estructura o la topología** de HMM se especifica por: los estados Q, los posibles símbolos que se pueden emitir Σ, un subconjunto de transiciones $(q_i \to q_j)$ con $P(q_i \to q_j) = 0$ y un subconjunto de emisiones $(q_i \uparrow \sigma)$ con $P(q_i \uparrow \sigma) = 0$.
En otras palabras la **topología** especifica un subconjunto de potenciales transiciones y emisiones que tienen probabilidad cero, dejando todas las demás probabilidades sin especificar. Llamémosle **parámetros** del modelo, cuando dada una cierta estructura se especifican las probabilidades de transición y emisión que tienen probabilidad diferente de cero.

Se va a denotar un modelo $M = (M_S, \theta_M)$ como la descomposición del modelo M en su parte estructural M_S y su parte paramétrica θ_M.

Se utiliza el superíndice en estados q^t y en emisiones σ^t, para denotar que el elemento t-ésimo de una cierta cadena, fue generado por el estado q, y que emitió el símbolo σ. Entonces:

$$P(q_i \to q_j) = p\left((q_j)^{t+1} | (q_i)^t\right),\ t = 0, 1, 2, ...$$
$$P(q_i \uparrow \sigma) = p\left(\sigma^t | q^t\right),\ t = 0, 1, 2, ...$$

Dónde $P(x|y)$ indica la probabilidad condicional, de que ocurra el suceso x dado que ocurrió el suceso y.

El estado inicial q_I ocurre al inicio de cualquier secuencia y el estado final q_F al final de cada secuencia. Por conveniencia se asume que $q_I, q_F\ \notin\ Q$ ya que en ninguno de esos estados se emiten símbolos.

Se dice que un HMM genera una cadena $x = x_1 x_2 ... x_l\ \in\ \Sigma^*$ (donde Σ^* representa al conjunto de todas las cadenas posibles de un alfabeto denominada Clausura de Kleene, la cuál se encuentra explicada en la página 81) si y sólo si hay por lo menos una secuencia o camino $x = q_1 q_2 ... q_l\ \in\ Q^*$ con probabilidad conjunta diferente de cero, tal que cada estado q_t emite el símbolo correspondiente x_t con probabilidad diferente de cero, para $t = 1, ..., l$. La probabilidad de generar dicha cadena es el producto de todas las transiciones y emisiones necesarias para generarla.

La probabilidad condicional $P(x|M)$ de una cadena x dado un modelo M, es la suma de las probabilidades de todos los caminos posibles que la pueden generar:

$$P(x|M) = \sum_{q_1 q_2 \ldots q_l \in Q^*} P(q_I \to q_1) P(q_1 \uparrow x_1) P(q_1 \to q_2) P(q_2 \uparrow x_2) \ldots P(q_l \uparrow x_l) P(q_l \to q_F) \quad (5)$$

Para ilustrar estos conceptos se considera un modelo M que genera las cadenas del lenguaje regular $(a\,(a \cup b))^*$.

En este modelo se pasa del estado inicial al estado uno con probabilidad uno y en dicho estado se emite una a con probabilidad uno. En el estado dos se puede emitir una a con probabilidad 0.5 o una b con probabilidad 0.5.

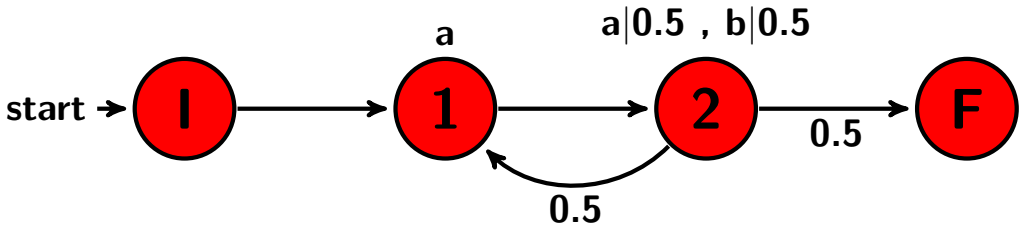


Figura 7: HMM que genera las cadenas del lenguaje regular $(a\,(a \cup b))^*$, imagen extraída de (Stolcke y Omohundro, 1994, p. 37)

La probabilidad de generar una cadena de caracteres x dado el modelo M es $P(x|M)$. Por ejemplo la probabilidad de que la cadena $x = \{a, b, a, b\}$ haya sido generada por el modelo M se computa de la siguiente forma:

$$P(\{a,b,a,b\}|M) = P(q_I \to q_1) P(q_1 \uparrow a) P(q_1 \to q_2) P(q_2 \uparrow b) P(q_2 \to q_1) P(q_1 \uparrow a) P(q_1 \to q_2) P(q_2 \uparrow b) P(q_2 \to q_F)$$

$$= (1 * 1)(1 * 0{,}5)(0{,}5 * 1)(1 * 0{,}5)0{,}5 = (0{,}5)^4 = 0{,}0625$$

El modelo M, acepta la palabra $x = \{a, b, a, b\}$ y además computa la probabilidad de generarla. Esta probabilidad es conocida como la **verosimilitud**; es un concepto muy importante que se va a utilizar para realizar la inferencia del modelo, ya que permite cuantificar qué tan probable es que el modelo M haya generado los datos x.

La verosimilitud se expresa como la probabilidad condicional $P(x|M)$, es decir la probabilidad de generar los datos x, dado que el modelo es el M (especificado en función de los parámetros θ_M).

4.2.2. Estimación de los parámetros de HMM

Para estimar los parámetros de HMM se asume una cierta topología M_S y se ajustan las probabilidades θ_M para maximizar la verosimilitud de los datos $P(X|M_S)$.

El problema principal es que ante una observación $x = x_1x_2...x_l$, no se sabe cual de los estados generó cada uno de los elementos. Es decir puede ser que la observación x_i pudo haber sido generada por algún estado, pero no se sabe cual (por eso el término estados ocultos). Por lo tanto para una secuencia de observaciones, hay varias combinaciones de estados posibles que pudieron haberla generado.

Si se supieran cuales fueron los estados generadores de cada una de las secuencias, por ejemplo que los estados generadores de la cadena x fueron $q_1q_2...q_l$, se podrían calcular los estadísticos:

- $c\left(q_i \to q_j\right)$ que representan la cantidad de transiciones del estado q_i al estado q_j, $\forall\ \ q_i, q_j \in\ Q$.
- $c\left(q_i \uparrow \sigma\right)$ que representan cantidad de emisiones del símbolo σ en el estado q_i, $\forall\ q_i \in\ Q\ y\ \sigma \in \Sigma$.

Y con esta información estimar los parámetros por máxima verosimilitud:

$$\widehat{P}\left(q_i \to q_j\right) = \frac{c\left(q_i \to q_j\right)}{\sum_{s\ \in\ Q} c\left(q_i \to s\right)} \tag{6}$$

$$\widehat{P}\left(q_i \uparrow \sigma\right) = \frac{c\left(q_i \uparrow \sigma\right)}{\sum_{\rho\ \in\ \Sigma} c\left(q_i \uparrow \rho\right)} \tag{7}$$

En la ecuación 6 se estima la probabilidad de transición de ir desde el estado q_i hacia el estado q_j. Intuitivamente es la proporción de las transiciones que se realizan desde el estado q_i al estado q_j (para todas las cadenas de entrenamiento observadas). El numerador de dicha ecuación representa la cantidad de veces que al estado q_i le sigue el estado q_j, el denominador representa la cantidad de veces que al estado q_i le sigue cualquier estado de Q.

En la ecuación 7 se estima la probabilidad de emitir el caracter σ en el estado q_i. Intuitivamente es la proporción de las emisiones de σ desde el estado q_i (para todas las cadenas de entrenamiento observadas) . El numerador de dicha ecuación representa la cantidad de veces que se emitió el caracter σ en el estado q_i, el denominador representa la cantidad de veces que se emitió cualquier caracter de Σ en el estado q_i.

El problema de no poder observar directamente los estados subyacentes, se puede abordar con una serie de técnicas denominadas algoritmos por *Expectation Maximization* (EM)(Dempster *et al.*, 1977).

La idea que está por detrás del algoritmo de EM es que los parámetros se inician de una manera arbitraria, luego se calculan los valores esperados de las variables ocultas con los parámetros escogidos para inicializar el algoritmo. Con esa información se re-calculan los parámetros por máxima verosimilitud. Se repite el proceso hasta que los parámetros del modelo convergen a un máximo local.

- En el paso **E** se calculan los valores esperados para las variables ocultas, con las aproximaciones de los parámetros

- En el paso **M**, se ajustan los parámetros para maximizar la verosimilitud de los datos, dados los valores de las variables ocultas calculadas en el paso anterior.

Algoritmos implementados de **EM**, ampliamente conocidos, como los propuestos por **Baum-Welch** y **Viterbi** permiten solucionar el problema de la estimación de los parámetros.

Algoritmo de Baum-Welch

Para cada cadena de caracteres $x = x_1x_2...x_T$ de la muestra, se puede calcular la probabilidad a posteriori $P\left((q_i)^t|x, M\right)$, es decir la probabilidad de que el t-ésimo elemento de la cadena haya sido generado por el estado q_i. También la probabilidad conjunta de pares de estados $P\left((q_i)^t, (q_j)^{t+1}|x, M\right)$, es decir de pasar del estado q_i al estado q_j en tiempo t. Esto se puede hacer con un algoritmo llamado ***forward-backward*** (ver ecuación 2 en la Sección 2.5 para más detalles). Con dichas probabilidades se pueden computar las esperanzas a posteriori:

$$\widehat{P}\left(q_i \to q_j\right) = E\left[p\left(q_i \to q_j\right)|X, M\right] \tag{8}$$

$$\widehat{P}\left(q_i \uparrow \sigma\right) = E\left[p\left(q_i \uparrow \sigma\right)|X, M\right] \tag{9}$$

Para calcular dichas esperanzas se inician los parámetros del modelo de forma aleatoria (paso **E**) y luego se calculan las siguientes variables (Rabiner, 1989):

- Se estima la probabilidad de estar en cierto estado en el tiempo t, utilizando el algoritmo de ***forward*** (el cual se detalla computacionalmente en la ecuación 1 en la Subsección 2.5.4)

$$\gamma_t(i) = P\left((q_i)^t|x, M\right)$$

  .

- Se estima la probabilidad de pasar del estado i al estado j en el tiempo t, utilizando el algoritmo de ***forward-backward*** (el cual se detalla computacionalmente en la ecuación 2 en la Sección 2.5)

$$\xi_t(i, j) = P\left((q_i)^t, (q_j)^{t+1}|x, M\right)$$

Con las estimaciones mencionadas se calcula el número esperado de transiciones desde el estado i al estado j, donde la variable t hace referencia al *t-ésimo* elemento de la cadena x observada:
$\sum_{t=1}^{T-1} \xi_t(i, j) =$ número esperado de transiciones desde i a j.

Se calcula el número esperado de transiciones que partiendo desde el estado i se llega hacia cualquier otro estado de Q:
$\sum_{t=1}^{T-1} \gamma_t(i) =$ número esperado de transiciones desde i hacia algún estado Q.

Utilizando las fórmulas mencionadas ahora se hace la re-estimación (paso **M**):

- La probabilidad de estar en el estado i partiendo del estado inicial

$$\widehat{P}\left(q_I \to q_i\right) = \gamma_1(i)$$

- La probabilidad de pasar del estado i al estado j

$$\widehat{P}\left(q_i \to q_j\right) = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

- La probabilidad de emitir el símbolo σ en el estado i

$$\widehat{P}\left(q_i \uparrow \sigma\right) = \frac{\sum_{t=1}^{T} \gamma_t(i) \cdot \delta(x_t, \sigma)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Donde el delta de Kronecker $\delta(x, y)$ es 1 si $x = y$ y 0 en otro caso.
La re-estimación de los parámetros se hace hasta llegar a un punto de convergencia (que es un óptimo local, ya que depende de los parámetros iniciales) el cual fue probado por Baum y colaboradores (Rabiner, 1989).

Aproximación por Viterbi

Una aproximación utilizada es asumir que cada cadena de caracteres $x = x_1 x_2 ... x_T$ fue generada por un único camino, el que tiene máxima probabilidad (de todos los posibles), donde el camino más probable es aquel que maximiza la siguiente expresión:

$$V\left(x|M\right) = \arg \max_{q_1 q_2 ... q_T \in Q^*} P\left(q_I \to q_1\right) P\left(q_1 \uparrow x_1\right) P\left(q_1 \to q_2\right) P\left(q_2 \uparrow x_2\right) ... P\left(q_T \uparrow x_T\right) P\left(q_T \to q_F\right)$$

Denotemos V_i al i-ésimo estado en $V(x|M)$. Asumiendo que $V_0(x|M) = q_I$ y que $V_T(x|M) = q_F$ por conveniencia.

Es decir para cada secuencia $x \in X$ se calcula la secuencia de estados más probable que la haya generado. La manera intuitiva de realizar dicho cálculo (partiendo de ciertos parámetros iniciales) es calcular todas las posibles combinaciones de estados que hayan podido generar la secuencia x, luego elegir la secuencia de estados más probable.

Viterbi plantea un procedimiento para resolver dicho cálculo de una manera eficiente, el mismo se detalla en la ecuación 3 de la Subsección 2.5.5.

Luego de tener las secuencias más probables de estados $V\left(x|M\right)$ que generaron cada una de las secuencias de símbolos $x \in X$, se aproximan los siguientes estadísticos:

$$\hat{c}\left(q_i \to q_j\right) = \sum_{x \in X} \sum_{i=0}^{T} \delta\left(q_i, V_i(x|M)\right) \delta\left(q_j, V_{i+1}(x|M)\right) \tag{10}$$

$$\hat{c}\left(q_i \uparrow \sigma\right) = \sum_{x \in X} \sum_{i=1}^{T} \delta\left(q_i, V_i(x|M)\right) \delta\left(\sigma, x_i\right) \tag{11}$$

donde el delta de Kronecker $\delta(x, y)$ es 1 si $x = y$, 0 en otro caso.

Tanto el algoritmo de Baum-Welch como el de Viterbi son técnicas estudiadas para resolver los problemas básicos de HMM. En la Sección 2.5 se detallan los formalismos de HMM desde un punto

de vista probabilístico detallando las técnicas de Baum-Welch en 2.5.6 y Viterbi en 2.5.5, así como ejemplos utilizados de cada operación requerida para las estimaciones.

4.2.3. Fusión de modelos para HMM

Como se explicó en la Subsección 4.2.1 un modelo markoviano de estados ocultos M está definido por su **estructura o topología** M_S (es decir la cantidad de estados que tiene, así como sus potenciales conexiones y posibles símbolos a emitir en cada estado) y su parte **paramétrica** θ_M que especifica con qué probabilidades se dan dichas conexiones y emisiones.

En enfoques estándar de HMM se define la estructura del modelo M_S de acuerdo a algún conocimiento del dominio donde se está trabajando. Entonces para un modelo M con una **estructura** M_S y un set de datos X (que se asume que fueron generados por el modelo) se puede aprender (estimar) los parámetros θ_M con algoritmos de Expectation-Maximization como los que se mencionaron en la sección 4.2.2 y se detallan en el anexo 2.5.

A diferencia de enfoques tradicionales de HMM, Stolcke y Omhoundro (Stolcke y Omohundro, 1994) proponen **inducir la estructura** M_S de cierto modelo a partir de un set de datos de entrenamiento X. Las observaciones son cadenas de caracteres, cada carácter es un símbolo del alfabeto Σ.

Para ello se va a utilizar un algoritmo inductivo de aprendizaje. La idea esencial es que el modelo debe generar los datos de entrenamiento X, pero además tiene que tener la capacidad de generalizar (es decir generar secuencias no vistas en el entrenamiento).

El proceso inductivo comienza con un modelo M_0 que solamente genera los datos de entrenamiento. Este modelo tiene la verosimilitud más alta de generarlos, pero solamente se limita a generar dichos datos.
Se van formando nuevos modelos sucesivamente, fusionando pares de estados. Estos nuevos modelos permiten generar nuevas cadenas, pero en contrapartida la probabilidad de generar las cadenas vistas en el entrenamiento bajo ese modelo decrece (disminuye la verosimilitud).

Inducción de la estructura de HMM fusionando estados

En el primer modelo M_0:

- El estado inicial tiene la misma cantidad de transiciones como observaciones en el set de datos X

- Tiene un estado para cada caracter de las observaciones, en el cual se emite el símbolo correspondiente a dicho caracter con probabilidad uno
- Se pasa de un estado al siguiente con probabilidad uno

El paso fundamental es fusionar estados, es decir reemplazar dos estados del modelo anterior por uno nuevo, como resultado hereda las transiciones y emisiones de sus antecesores.

Estas fusiones permiten generar nuevas secuencias, además de las ya observadas en el modelo inicial. En cada paso de la fusión:

- Se eliminan los viejos estados y se agrega un nuevo estado fusionado. Los viejos estados son llamados estados padres.
- Las transiciones desde y hacia los viejos estados son redirigidas hacia los nuevos estados. Las probabilidades de transición son re-calculadas para maximizar la verosimilitud.
- Las emisiones en el nuevo estado son la unión de las emisiones de los viejos estados, las probabilidades de emisión son reajustadas para maximizar la verosimilitud.

Por ejemplo se considera el lenguaje regular $\{ab\}^+$ y dos muestras de dicho lenguaje $X = \{ab, abab\}$, a través del algoritmo **fusión de modelos (model merging)** se infiere el modelo generador de dicho lenguaje a partir de las muestras observadas.

Como se ilustra en la figura 8 extraída de (Stolcke y Omohundro, 1994, p. 37) el modelo inicial M_0 solamente genera las dos secuencias de la muestra. Sucesivamente se colapsan estados llegando a una estructura que permite generalizar (generan nuevas cadenas no vistas en el entrenamiento, pero que comparten ciertos patrones en común).

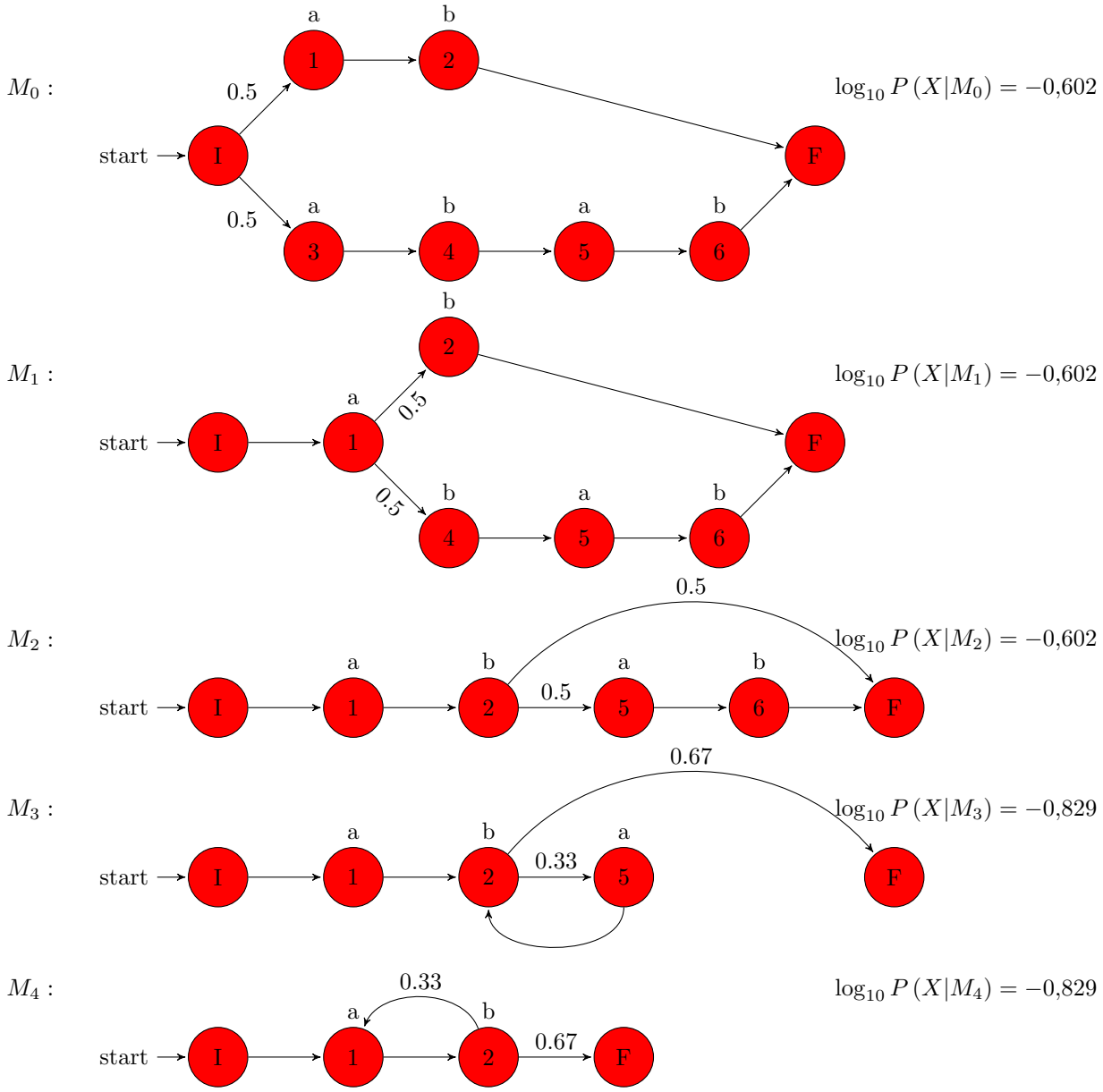


Figura 8: Secuencia de modelos obtenidos con la muestra $\{ab, abab\}$, imagen extraída de (Stolcke y Omohundro, 1994, p. 37)

En el ejemplo de la figura 8 se decidió enumerar el nuevo estado fusionado como el número del estado menor de los estados padres (como hacen en (Stolcke y Omohundro, 1994)). Por ejemplo en el M_1 se fusionan los estados **tres** y **uno** del modelo M_0, por lo tanto el nuevo estado hereda el número **uno**.

Los tres primeros modelos del ejemplo M_0, M_1, M_2 solo pueden generar las cadenas vistas en el

entrenamiento $\{ab, abab\}$. A su vez tienen la verosimilitud más alta de generar los datos de la muestra (los tres tienen la misma).

La verosimilitud del modelo M_0 es $P\left(X = \{ab, abab\}\,|M_0\right) = (0{,}5) * (0{,}5) = 0{,}25$ y el logaritmo de la verosimilitud en base 10 es:

$$\log_{10} P\left(X|M_0\right) = \log_{10}(0{,}25) = -0{,}60206 \tag{12}$$

Fusionando estados se construyen los modelos M_3 y M_4 los cuales sí pueden generalizar. Es decir pueden generar secuencias del lenguaje $\{ab\}^+$ pero con una verosimilitud más baja de observar los datos de la muestra.

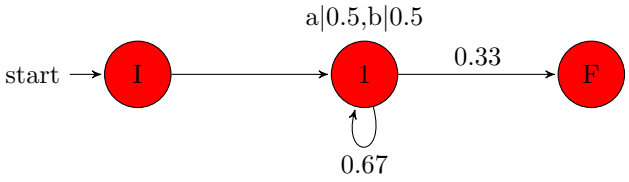
La verosimilitud del modelo M_4 es $P\left(X = \{ab, abab\}\,|M_4\right) = (0{,}67) * (0{,}33 * 0{,}67) = 0{,}148$ y el logaritmo de la verosimilitud en base 10 es:

$$\log_{10} P\left(X|M_4\right) = \log_{10}(0{,}148137) = -0{,}829 \tag{13}$$

Por lo tanto en la figura 8 se comienza con un modelo M_0 que tiene la verosimilitud más alta de generar los datos de entrenamiento, pero que sobre-ajusta a los datos. Luego se llega al modelo M_4 que si tiene la capacidad de generalizar, pero como contrapartida la verosimilitud de la muestra es menor con respecto a los otros modelos. Se observan dos tendencias contrapuestas; la caída en la verosimilitud de -0,602 a -0,829 pero como contraposición el modelo puede generalizar.

En un siguiente paso se podría realizar otra fusión y generar un nuevo modelo M_5, el cual puede generar secuencias del lenguaje $(a|b)^+$, este modelo puede generar por ejemplo cadenas que sólo tienen la letra "a" o sólo la letra "b", pero la disminución de la verosimilitud es mucho más pronunciada.

M_5 :



$\log_{10} P\left(X|M_5\right) = -3{,}465$

Figura 9: Modelo M_5

La verosimilitud del modelo M_5 es $P\left(X = \{ab, abab\}\,|M_5\right) = (0{,}5)^6 * (0{,}67)^4 * (0{,}33)^2 = 0{,}00034$ y el logaritmo de la verosimilitud en base 10 es:

$$\log_{10} P\left(X|M_0\right) = \log_{10}(0{,}00034) = -3{,}465 \tag{14}$$

Por lo tanto hacer una fusión más y pasar al M_5 tendría un decrecimiento de la verosimilitud muy pronunciado, pasando de 0,148 a 0,00034 (de tres órdenes de magnitud decimal), es decir que el nuevo modelo puede generalizar mucho más aún, pero se está perdiendo mucha verosimilitud.

Las preguntas claves son ¿hasta cuándo generalizar?, ¿hasta cuánto se acepta que baje la verosimilitud?. Se necesita un criterio que guíe cuando parar con la generalización. En el siguiente punto se formaliza dicho criterio, a través del enfoque denominado **Fusión de modelos Bayesiana (Bayesian Model Merging)**.

4.2.4. Fusión de Modelos Bayesiana

Aprender de los datos en el marco de **Fusión de modelos Bayesiana** significa generalizar a partir de ellos, esto implica una tensión entre la verosimilitud del modelo y un sesgo hacia modelos más simples. La verosimilitud representa que tan bien el modelo M ajusta a los datos observados en la muestra, pero se quiere evitar el sobre-ajuste, es decir encontrar un modelo más simple que no se limite solo a generar esos datos de entrenamiento.

Se asume que existe una probabilidad $P(M)$ independiente de los datos que asigna a cada modelo M una probabilidad a priori (antes de observar los datos), que puede entenderse como la expresión que formaliza el sesgo hacia modelos más simples.

Existe una tensión entre ajustar el modelo a los datos y la capacidad del modelo de generalizar. Se comienza arbitrariamente con un modelo inicial que solo genera los datos de entrenamiento (sobre-ajusta) y se fusionan estados para generar modelos que puedan generalizar hasta llegar al extremo donde se acepta cualquier cadena de caracteres, formada por los símbolos del alfabeto (sobre-generalizar). El criterio de parada para no llegar a dicho extremo es el comportamiento de la probabilidad a posteriori Bayesiana del modelo.

Dados ciertos datos X, cada modelo M_i tiene una probabilidad a posteriori:

$$P\left(M_i|X\right) = \frac{P\left(X|M_i\right) * P\left(M_i\right)}{P\left(X\right)} \tag{15}$$

Dicho término representa la probabilidad de que el modelo M_i haya sido el que generó la secuencia X observada en el entrenamiento.

Por lo tanto:

- Cada paso en la fusión debe maximizar la probabilidad a posteriori (no normalizada) del modelo $P\left(M|X\right)$
- Se va a seguir fusionando pares de estados siempre y cuando la probabilidad a posteriori (no normalizada) siga creciendo

La inferencia Bayesiana del modelo es una generalización de la estimación por máxima verosimilitud, agregándole una probabilidad a priori a la expresión que se quiere maximizar.

Como se dijo anteriormente el modelo inicial M_0 solamente genera los datos observados. Sucesivamente se van a ir generando nuevos modelos $\{M_1, M_2...\}$ a partir de la fusión de pares de estados (donde cada modelo tiene un estado menos que el anterior). El criterio para no realizar una fusión adicional es si la probabilidad a posteriori (no normalizada) del nuevo modelo comienza a decrecer, es decir si se cumple:

$$P\left(M_{i+1}|X\right) < P\left(M_i|X\right) \tag{16}$$

Esto se interpreta como que el modelo M_i tiene más probabilidad de haber generado los datos X que el modelo M_{i+1}. Es decir, el modelo M_i tiene más probabilidad a posteriori, dados los datos X.

Cuando ocurra lo contrario:

$$P\left(M_{i+1}|X\right) > P\left(M_i|X\right) \tag{17}$$

Se interpreta como que el modelo fusionado M_{i+1} tiene más probabilidad a posteriori de ser el modelo generador de los datos X que el modelo M_i.

Entonces se utiliza el modelo M_{i+1} siempre y cuando se cumpla:

$$P\left(M_{i+1}|X\right) > P\left(M_i|X\right) \tag{18}$$

Dicha expresión con el teorema de Bayes se define como:

$$\frac{P\left(X|M_{i+1}\right) * P\left(M_{i+1}\right)}{P\left(X\right)} > \frac{P\left(X|M_i\right) * P\left(M_i\right)}{P\left(X\right)} \tag{19}$$

El término que está en el denominador (probabilidad de los datos $P(X)$) es el mismo para cada modelo, por lo que se puede tomar como constante y no utilizarlo en la evaluación.

La productoria que está en el numerador de ambos lados de la inecuación 19 es el término que guía la decisión. La idea que está por detrás es que al fusionar (al pasar de un modelo M_i a un modelo M_{i+1}) ese término puede decrecer o crecer, ya que los elementos de la productoria se comportan de manera opuesta:

- La verosimilitud decrece, es decir, la probabilidad de generar los datos de entrenamiento con el modelo fusionado es más pequeña. Como se vio claramente en la sección 4.2.3 el primer modelo es el que mejor se ajusta a los datos X, pero sucesivamente cada modelo pierde verosimilitud al fusionar pares de estados, en general:

$$P\left(X|M_{i+1}\right) < P\left(X|M_i\right) \tag{20}$$

- El término que impulsa la generalización es la probabilidad a priori del modelo, imponiendo un sesgo hacia modelos más simples, es decir la probabilidad a priori del modelo fusionado es mayor.

$$P\left(M_{i+1}\right) > P\left(M_i\right) \tag{21}$$

Como hay un término que crece y otro que decrece, es necesario evaluar que es lo que ocurre con esa tensión en cada paso de la fusión. En el momento que la probabilidad a posteriori comienza a decrecer es cuando se finaliza la generalización y se decide por el modelo M_i.

4.2.5. Probabilidades a priori para HMM

La inferencia Bayesiana, basada en las probabilidades a posteriori (no normalizadas), tiene una formulación alternativa en términos de teoría de la información. Plantear el dualismo entre ambos formalismos es de gran utilidad; tanto para tener un entendimiento más profundo de los principios subyacentes (de ambos formalismos), como para computar las probabilidades de las distribuciones a priori P(M).

La maximización de:

$$P(M, X) = P(M) P(X|M) \tag{22}$$

implicito en la inferencia Bayesiana del modelo M es equivalente a minimizar:

$$-\log P(X, M) = -\log P(M) - \log P(X|M) \tag{23}$$

En teoría de la información (Rissanen, 1983) el valor negativo del logaritmo de la probabilidad de un evento discreto E representa el largo del código óptimo para comunicar dicha instancia, con el cual se minimiza el promedio del largo del código del respectivo mensaje.

Las regularidades que se encuentran en los datos pueden ser utilizadas para describirlos de forma compacta. La idea que guía el principio de Minimo Largo de Descripción (MDL por sus siglas en inglés) es la equivalencia entre: aprender de los datos y describirlos en forma compacta. Se entiende por "describir" los datos como la codificación mediante secuencias de bits.

¿Como se describen los datos de forma compacta?

Suponiendo que se tiene un mensaje $X = \{FAAACAGC\}$ que es una secuencia de cuatro eventos discretos (A, C, G, F) con las siguientes probabilidades respectivamente $(1/2, 1/4, 1/8, 1/8)$.

- Una primer forma de codificar ese mensaje sería utilizar dos bits para cada evento. De esa forma se codifican los eventos (A, C, G, F) con el siguiente esquema $(00, 01, 10, 11)$. El largo promedio L(C) de cada mensaje son 2 bits.

- Una forma alternativa de codificar, utilizando conceptos de teoría de la información, es utilizar menos bits para representar los eventos más probables y más bits para representar los menos probables. Según el *teorema de Shannon* (Shannon, 2001) el largo del código de cada evento está determinado por $-\log_2 p(E)$ y es el largo de código, unívocamente decodificable, mas corto para representar el mensaje.

  En el ejemplo:

  - El evento A tiene una probabilidad $P(A) = 0{,}5$. El código óptimo para A es $-\log_2 0{,}5 = 1$, es decir se codifica con un bit
  - El evento C tiene una probabilidad $P(C) = 0{,}25$. El código óptimo es $-\log_2 0{,}25 = 2$, es decir se codifica con dos bits

- Los eventos G y F que tienen una probabilidad $P(G) = P(F) = 0{,}125$. El código óptimo es $-\log_2 0{,}125 = 3$, es decir se codifican con tres bits

De esa forma se codifican los símbolos (A, C, G, F) con el siguiente esquema $(0, 10, 110, 111)$. El largo promedio L(C) de cada mensaje son 1.75 bits. Calculado de la siguiente forma:

$$L(C) = 0{,}5 * 1\ bits + 0{,}25 * 2\ bits + 0{,}125 * 3\ bits + 0{,}125 * 3\ bits = 1{,}75\ bits$$

Si se tiene un mensaje con 100 símbolos:

- Utilizando la primer codificación se necesitan 200 bits para los datos y 8 bits para especificar como se codifica cada evento. Por lo tanto se representa con un total de 208 bits.
- Utilizando la segunda codificación con el criterio de MDL se necesitan en promedio 175 bits para los datos y 9 bits para especificar como se codifica cada evento. Por lo tanto se representa con 184 bits.

Si se conoce que los datos están gobernados por una cierta distribución de probabilidad p, sabemos exactamente cómo codificarlos de forma compacta.

¿Cómo se utiliza MDL para inferir la distribución de probabilidad que gobiernan nuestros datos?

Para la tarea de modelar datos, según el principio de MDL, asignar largos de código y asignar probabilidades es esencialmente lo mismo. Dado un juego de datos $x^n = x_1, x_2, ..., x_n$, el principio de MDL sugerirá, elegir el modelo puntual, p, que minimice:

$$l(p) + l\left(x^n|p\right)$$

Donde

- $l(p)$ es el largo de un código con el cual describiríamos p.
- $l\left(x^n|p\right)$ es el largo de código con que codificaríamos x^n asumiendo una distribución p.

Volviendo nuevamente a la ecuación que se quiere minimizar:

$$-\log P\left(X, M\right) = -\log P\left(M\right) - \log P\left(X|M\right) \tag{24}$$

Específicamente, $-\log P\left(M\right)$ es el largo en bits del código de descripción del modelo M a priori y $-\log P\left(X|M\right)$ corresponde a la descripción de los datos X (utilizando M como el modelo con el cual se basan los largos del código). La probabilidad conjunta puede ser interpretada como el largo de descripción total del modelo y los datos. En el marco de MDL se busca minimizar el largo de código de los datos conjuntamente con un modelo M.

La inferencia por mínimo largo de descripción (MDL) es entonces equivalente y una alternativa de la maximización de la probabilidad a posteriori.

$$-\log P(X,M) = \underbrace{-\log P(M)}_{\substack{l(M) \\ \text{código óptimo} \\ \text{del modelo M}}} \overbrace{-\log P(X|M)}^{\substack{l(X|M) \\ \text{código óptimo} \\ \text{de los datos X} \\ \text{dado el modelo M}}} \tag{25}$$

El problema de inferir el modelo subyacente en términos de MDL es la tensión entre ajustar a los datos (minimizando el código con el que se representan los datos) y encontrar un modelo que sea más general (minimizando el código con el que se representa el modelo).

Como se mencionó en el principio de la sección, uno de los objetivos de formular el problema en el marco de MDL es utilizar algunos conceptos para evaluar las probabilidades a priori de los modelos P(M). Luego se van a utilizar en la maximización bayesiana de las probabilidades a posteriori.

Largo de descripción de las distribuciones a priori.

De forma inversa, cualquier esquema de codificación que asigna a un modelo M un largo de código $l(M)$ para describirlo, puede utilizarse para inducir la probabilidad de la distribución a priori del modelo.

Es decir, conociendo el largo del código en bits que tiene el modelo $l(M)$, se puede despejar la probabilidad de la distribución $P(M)$ de la siguiente forma:

Utilizando la siguiente igualdad:

$$l(M) = -\log P(M) \tag{26}$$

Despejo la probabilidad del modelo:

$$P(M) \propto e^{-l(M)} \tag{27}$$

Una forma natural de codificar todas las emisiones y las transiciones en HMM es simplemente enumerarlas. Cada transición puede ser codificada utilizando $\log(|Q|+1)$ bits, ya que hay $|Q|$ posibles transiciones a donde se puede ir desde cierto estado q, sumando un uno que permite conectar con el estado final.

El largo de descripción para todas las transiciones desde el estado q es $n_t^{(q)} \log(|Q|+1)$ bits. Donde $n_t^{(q)}$ representa el número de transiciones desde el estado q y $n_e^{(q)}$ representa el número de emisiones.

De la misma forma, todas las emisiones desde q pueden codificarse usando $n_e^{(q)} \log(|\Sigma| + 1)$ bits.

Entonces la probabilidad a priori de la contribución del estado q (es decir $M_S^{(q)}$) es:

$$P\left(M_S^{(q)}\right) \propto (|Q| + 1)^{-n_t^{(q)}} (|\Sigma| + 1)^{-n_e^{(q)}} \tag{28}$$

Donde la priori de la estructura del modelo M_S queda determinada por la productoria de las contribuciones de cada estado:

$$P(M_S) = \prod_{q \in Q} P\left(M_S^{(q)}\right) \tag{29}$$

4.2.6. Cálculo de las probabilidades a posteriori

Una de las formas de computar la probabilidad a posteriori (no normalizada) de un potencial modelo es evaluar la probabilidad a priori de la estructura del modelo $P(M_S)$ y luego estimar los parámetros θ_M por máxima verosimilitud para dicha estructura M_S. Con dichas estimaciones computar la verosimilitud del modelo (Stolcke y Omohundro, 1994, p. 22).

El algoritmo

1. Construcción del modelo inicial M_0 que tiene la máxima verosimilitud dado el set de datos X.
2. Se inicializa:
   - i=0
   - Parar=Falso
   - max=Cantidad de estados -1

   Mientras $\{i \leq max\}$ & $\{Parar == Falso\}$

   a) Se plantean todas las K posibles parejas de estados a fusionar del modelo M_i

   b) Para cada candidato $k \in K$, se computa el modelo fusionado $k(M_i)$, y su probabilidad a posteriori (no normalizada) $P\left(k(M_i)|X\right)$

   c) Sea k^* el candidato que maximiza $P\left(k(M_i)|X\right)$. Se define el siguiente modelo $M_{i+1} = k^*(M_i)$

   d)
   - Si $P\left(M_{i+1}|X\right) < P\left(M_i|X\right)$, Parar=Verdadero
   - Sino i=i+1
3. Devuelve M_i como el modelo inducido

4.2.7. Decisiones en la implementación

Para calcular la probabilidad a posteriori (no normalizada) de un modelo, se tiene que evaluar la priori estructural $P(M_S)$ y luego calcular las estimaciones de los parámetros θ_M.
Para estimar las probabilidades de los parámetros se puede utilizar el algoritmo iterativo de Baum-Welch que es un caso de EM. Sin embargo dicho procedimiento requiere hacer la re-estimación en cada iteración, lo que puede llevar mucho tiempo.
Este problema puede solucionarse utilizando el algoritmo de **Viterbi**, asumiendo que una muestra dada es generada por un único camino generador en el modelo (Stolcke y Omohundro, 1994, p. 22).

Estimación de los parámetros con el algoritmo de Viterbi

Para estimar los parámetros basados en el algoritmo de **Viterbi** se utilizan las ecuaciones vistas en la sección 4.2.2:

Asumiendo que cada cadena de caracteres se generó por un único camino, el más probable, que es aquel que maximiza:

$$V(x|M) = \arg \max_{q_1 q_2 ... q_l \in Q^*} P(q_I \to q_1) P(q_1 \uparrow x_1) P(q_1 \to q_2) P(q_2 \uparrow x_2) ... P(q_l \uparrow x_l) P(q_l \to q_F) \tag{30}$$

Si V_i denota el i-ésimo estado en $V(X|M)$, asumiendo que $V_0(X|M) = q_I$ y que $V_{l+1}(X|M) = q_F$, los estadísticos aproximados utilizando el algoritmo de Viterbi son:

$$\hat{c}(q_i \to q_j) = \sum_{x \in X} \sum_{i=0}^{l} \delta(q_i, V_i(x|M)) \delta(q_j, V_{i+1}(x|M)) \tag{31}$$

$$\hat{c}(q_i \uparrow \sigma) = \sum_{x \in X} \sum_{i=1}^{l} \delta(q_i, V_i(x|M)) \delta(\sigma, x_i) \tag{32}$$

Donde el delta de Kronecker $\delta(x, y)$ es 1 si $x = y$ y 0 en otro caso, y donde $\hat{c}(q_i \to q_j)$ cuenta las veces que a un estado q_i lo sigue un estado q_j para todas las secuencias $x \in X$.

Luego de calcular los estadísticos aproximados, se calculan las estimaciones de las probabilidades:

$$\widehat{P}(q_i \to q_j) = \frac{\hat{c}(q_i \to q_j)}{\sum_{s \in Q} [\hat{c}(q_i \to s)]} \tag{33}$$

$$\widehat{P}(q_i \uparrow \sigma) = \frac{\hat{c}(q_i \uparrow \sigma)}{\sum_{\rho \in \Sigma} [\hat{c}(q_i \uparrow \rho)]} \tag{34}$$

Pesos Globales para las probabilidades a priori

Como se mencionó anteriormente la clave está en encontrar un modelo que generalice y a su vez que se ajuste a los datos observados en la muestra. La generalización es gobernada por la probabilidad a priori del modelo, mientras que el ajuste, por la verosimilitud de los datos. Se plantea que en la práctica es conveniente tener un parámetro que pueda controlar cuando se debería detener la generalización. La forma logarítmica de la función a posteriori que se quiere maximizar es:

$$\log P(M) + \log P\left(X|M\right) \tag{35}$$

Para tener un control sobre la priori es necesario agregarle el peso λ a la distribución $P(M)$:

$$\lambda \log P(M) + \log P\left(X|M\right) \tag{36}$$

Entonces:

- Cuándo $\lambda > 1$ el algoritmo detiene la generalización más tarde, debido a que la priori tiene un peso mayor en la tensión. Por más que la verosimilitud tenga un descenso pronunciado, el parámetro λ hace que la priori tenga más peso, haciendo que la probabilidad a posteriori se incremente, ya que se le da más "importancia" al término que es creciente.
- Cuándo $\lambda < 1$ el algoritmo detiene la generalización antes, debido a que la priori tiene menos peso en la tensión. Cualquier descenso en la verosimilitud contribuye a que la probabilidad a posteriori no se incremente, ya que se le "quita importancia" al término que es creciente.

En la práctica es muy utilizado controlar el peso a priori λ . En (Stolcke y Omohundro, 1994) plantean que para encontrar un valor óptimo para λ es necesario pasar por un proceso de ensayo y error.

En la práctica el problema más común es cuando se incrementa la probabilidad a posteriori hasta cierto paso en el que comienza a decrecer (en el cual el algoritmo para de fusionar) pero luego de unos pasos adicionales comienza a crecer nuevamente (es un máximo relativo). La manera más sencilla de resolver este problema es agregar de dos a cinco pasos adicionales luego de ver que la probabilidad comienza a decrecer.

5. Implementación del Algoritmo

Se implementó el algoritmo **Fusión de modelos bayesiana** y se evaluó en dos estudio de casos. Primero en un ejemplo de un lenguaje creado artificialmente como el de (Stolcke y Omohundro, 1994, p. 30), en donde el algoritmo aprende dicho lenguaje a partir de un conjunto de entrenamiento. En el segundo caso se experimentó con un ejemplo real, un prototipo que aprende nombres de personas a partir de un conjunto de nombres en el entrenamiento.

Para calcular la probabilidad a posteriori de cada uno de los modelos, se evaluó la priori estructural $P(M_S)$ (utilizando el criterio de MDL) y se calcularon las estimaciones por máxima verosimilitud de los parámetros θ_M con el algoritmo de **Viterbi**.

5.1. Subrutinas y lenguaje utilizado

Para la implementación del algoritmo el lenguaje utilizado fue el Software R (R Core Team, 2017). Se construyeron nuevas funciones a partir de otras ya existentes de la subrutina HMM (Scientific Software Development - Dr. Lin Himmelmann y www.linhi.com, 2010).

La técnica propuesta se implementó para el caso particular de HMM visto como un AFNDE (en el que solo algunos estados se conectan con el final), en cambio la subrutina HMM fue creada para el caso general (en el que cualquier estado puede ser el final).

Para poder utilizar la subrutina la solución fue imponer que algunos estados sean los terminales, creando un estado final (que no va hacia ningún otro estado y no emite ningún símbolo) en el que solo ellos pueden conectarse con él.

De la subrutina HMM se utilizaron tres funciones. Una que crea un modelo M tomando como datos de entrada la especificación del mismo (es decir los estados iniciales, el alfabeto, la matriz de transiciones, la matriz de emisiones y las probabilidades iniciales). Dicha función se llama **initHMM()**.

Las otras dos se invocan luego de especificar cierto modelo M. La función **forward()** calcula la probabilidad de generar una secuencia de caracteres x y la función **viterbi()** infiere la secuencia de estados más probable que generaron una observación x.

A partir de las funciones mencionadas de dicha subrutina se crearon otras propias (los cuales están documentados en la sección 8.2) para implementar el algoritmo. La función **m0()** arma el modelo inicial M_0 a partir de datos de entrada X.

La función **baymodmeg()** decide cuando detener la generalización y retorna el modelo final. Para tomar la decisión utiliza cuatro funciones auxiliares que se invocan cada vez que se tiene un modelo M candidato a generalizar. Las mismas se detallan en el siguiente párrafo.

La función **vero()** calcula la verosimilitud de un conjunto de datos X, la función **priori()** calcula la priori estructural $P(M_S)$, la función **reesViterbi()** hace la estimación de los parámetros θ_M (con el algoritmo de Viterbi) y la función **modmeg()** decide cuales estados se van a colapsar.

Especificación del algoritmo implementado:

El algoritmo implementado, utilizando las funciones creadas (que se especificaron en el párrafo anterior), queda de la siguiente forma:

Se parte de un set de datos X con observaciones para inferir el modelo.

1. Se construye el modelo inicial M_0 con la función **m0(X)** a partir del set de datos X.
2. Se hace la inferencia del HMM con la función **baymodmeg(M_0, X, λ)**. Como datos de entrada se utiliza: el modelo inicial M_0 (construido en el paso anterior), el set de datos X y un valor para el parámetro λ

   El algoritmo funciona de la siguiente forma:

   Se inicializa:

   - i=0
   - Parar=Falso
   - max=Cantidad de estados iniciales -1

   Mientras $\{i \leq max\}$ & $\{Parar = Falso\}$

   Con la función **modmeg(M_i)**

   a) Se planean todos las K posibles parejas de estados a fusionar del modelo M_i

   b) Para cada candidato $k \in K$, se computa el modelo fusionado $k(M_i)$=**reesViterbi(M_i)**, y su probabilidad a posteriori (no normalizada) $P(k(M_i)|X)$=**vero($k(M_i)$priori($k(M_i)$)**

   c) Sea k^* el candidato que maximiza $P(k(M_i)|X)$. Se define el siguiente modelo $M_{i+1} = k^*(M_i)$

   d)
   - Si $P(M_{i+1}|X) < P(M_i|X)$, Parar=Verdadero
   - Sino i=i+1

3. Devuelve M_i como el modelo inducido

6. Experimentación y resultados

6.1. Estudio de casos I

Se propone para el primer caso de estudio aprender el lenguaje regular $ac^*a \cup bc^*b$ a partir de un mínimo conjunto de entrenamiento, constituido por 8 cadenas que pertenecen a dicho lenguaje:

Palabra
aa
bb
aca
bcb
acc
bccb
accca
bcccb

Cuadro 12: Conjunto de entrenamiento para aprender el lenguaje

En este caso de estudio se propone observar como se comporta el algoritmo utilizando diferentes pesos del parámetro a priori $\lambda = \{0{,}016; 0{,}16; 1{,}0\}$. A su vez inducir el modelo generador el lenguaje objetivo.

Como se mencionó anteriormente cuando se fija un valor de λ menor que uno, el algoritmo detiene antes la generalización, por el contrario cuando el valor es mayor que uno se detiene más tarde.

En la figura 10 de la página siguiente se observan los modelos obtenidos luego de utilizar los tres pesos diferentes del parámetro λ, en donde se representó gráficamente el modelo inferido para cada peso.

Por convención al fusionar un par de estados el nuevo modelo hereda la etiqueta (el número) del menor de los dos, es decir si en un paso se fusiona el estado 10 con el 9, el nuevo estado hereda el número 9.

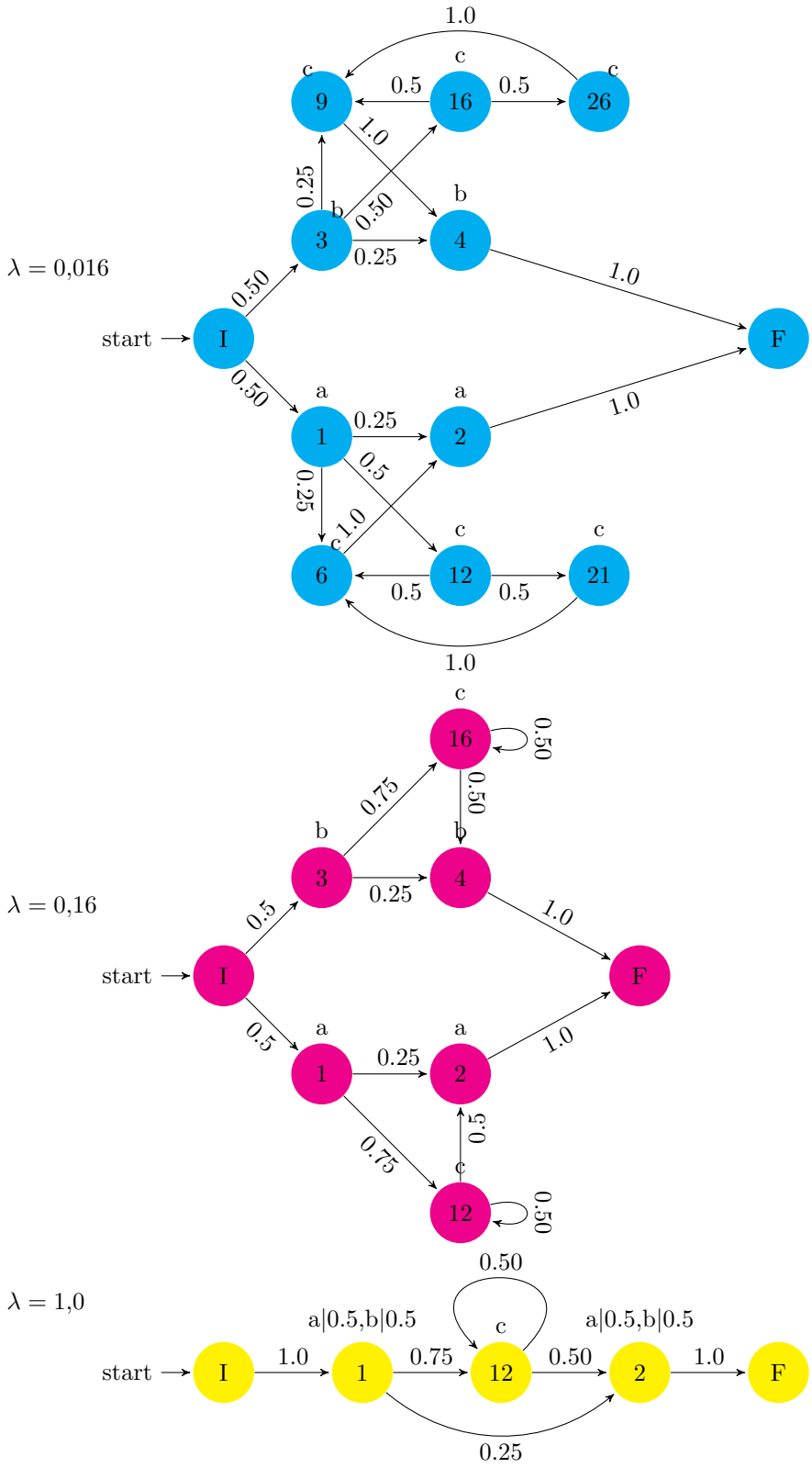


Figura 10: Modelos inducidos del estudio de caso 1, variando el peso global λ de las distribuciones a priori λ para controlar la generalización

- Cuando se fija $\lambda = 0{,}016$, un valor relativamente pequeño, el modelo que se infiere se sobre-ajusta a los datos. Solamente se pueden generar las secuencias observadas en el entrenamiento.
- Cuando se fija $\lambda = 0{,}16$ es cuando se llega al modelo generador del lenguaje objetivo. Este modelo (representado en color magenta en la figura 10) puede generar todas las secuencias del lenguaje regular $ac^*a \cup bc^*b$.
- Cuando se fija $\lambda = 1{,}0$ el modelo está sobre-generalizando.

En el cuadro 13, se imprimen para cada paso del algoritmo los indicadores utilizados para decidir cuando detener la generalización. En la primer columna se imprime el paso en el que está el algoritmo, en la segunda columna la cantidad de estados $|Q|$ que tiene el modelo, en la tercer columna el logaritmo de la verosimilitud y en la cuarta columna el logaritmo de la priori.

Desde la quinta columna hasta la séptima se imprime el logaritmo de la posteriori según diferentes valores de λ, cada uno representado con el color correspondiente al de los modelos graficados en la figura 10.

Paso	$\|Q\|$	Log Vero	Log Priori	Post$\lambda = 0,016$	Post$\lambda = 0,16$	Post$\lambda = 1,0$
2	28	-16,64	-134,06	-18,78	-38,09	-150,70
3	27	-16,64	-131,62	-18,74	-37,70	-148,26
4	26	-16,64	-125,88	-18,65	-36,78	-142,52
5	25	-16,64	-120,18	-18,56	-35,86	-136,82
6	24	-16,64	-114,51	-18,47	-34,96	-131,15
7	23	-16,64	-112,02	-18,43	-34,56	-128,66
8	22	-16,64	-106,39	-18,34	-33,66	-123,02
9	21	-16,64	-103,84	-18,30	-33,25	-120,47
10	20	-16,64	-101,23	-18,26	-32,83	-117,87
11	19	-16,64	-98,56	-18,21	-32,41	-115,20
12	18	-16,64	-92,94	-18,12	-31,51	-109,57
13	17	-16,64	-87,34	-18,03	-30,61	-103,98
14	16	-16,64	-81,79	-17,94	-29,72	-98,43
15	15	-16,64	-76,28	-17,86	-28,84	-92,91
16	14	-16,64	-70,80	-17,77	-27,96	-87,44
17	13	-16,64	-65,37	-17,68	-27,09	-82,01
18	12	-16,64	-59,98	-17,60	-26,23	-76,61
19	11	-16,64	-54,63	-17,51	-25,38	-71,26
20	10	-17,16	-49,32	-17,95	-25,05	-66,48
21	9	-17,50	-41,85	-18,17	-24,19	-59,35
22	8	-18,02	-36,74	-18,61	-23,90	-54,76
23	7	-18,36	-29,72	-18,84	-23,12	-48,08
24	6	-22,52	-24,85	-22,92	-26,50	-47,37
25	5	-23,91	-21,64	-24,25	-27,37	-45,55
26	4	-23,91	-15,25	-24,15	-26,35	-39,16
27	3	-35,00	-10,46	-35,16	-36,67	-45,46
28	2	-46,96	-4,85	-47,04	-47,74	-51,82

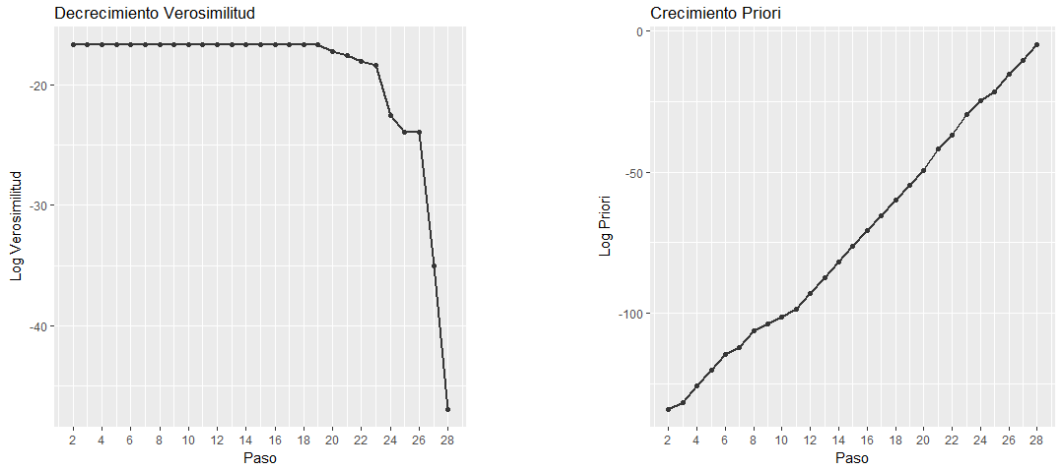
Cuadro 13: Largo del modelo, verosimilitud, priori y posteriori según diferentes valores de λ

El logaritmo de la probabilidad a posteriori no normalizada (incluyendo el peso λ) se calcula como $\log(Verosimilitud) + \lambda * \log(Priori)$. Por ejemplo cuando $\lambda = 0{,}016$ (la cuarta columna) el logaritmo de la probabilidad a posteriori (no normalizadas) en el paso dos se calcula como $-16{,}64 + 0{,}016 * (-134{,}06) = -18{,}78$.

La idea es que se fusionen estados siempre y cuando la probabilidad a posteriori (no normalizada) se incremente, como está en escala logarítmica se presentan números negativos. Para cada valor de λ en la tabla se coloreó la columna de la posteriori hasta el paso en que deja de crecer. Por ejemplo en la columna magenta con un peso de $\lambda = 0{,}16$ el algoritmo se detiene en el paso 23, en el cual se induce el modelo del lenguaje objetivo.

La tercer columna del cuadro 13 muestra los valores del logaritmo de la verosimilitud en cada paso; estos valores son monótonamente decrecientes. Al contrario del logaritmo de la priori (cuarta columna)

que es monótonamente creciente. Estas tendencias contrapuestas se muestran en la siguiente figura 11



(a) Logaritmo de la verosimilitud en cada modelo

(b) Logaritmo de la priori en cada modelo

Figura 11: Logaritmo de la verosimilitud y logaritmo de la distribución a priori del modelo en cada paso de la fusión

El logaritmo de la verosimilitud hasta el paso 19 se mantiene constante y luego comienza a decrecer. Si el algoritmo se detiene en dicho paso el modelo solamente acepta las observaciones vistas en la muestra, en la figura 10 dicho modelo está representado en color cyan.

En la figura 12 se grafica la posteriori para el caso que $\lambda = 0{,}16$, donde se visualiza claramente la presencia de un máximo en el paso 23, el cual se marcó con una línea negra. Dicho modelo es el que genera el lenguaje regular objetivo.

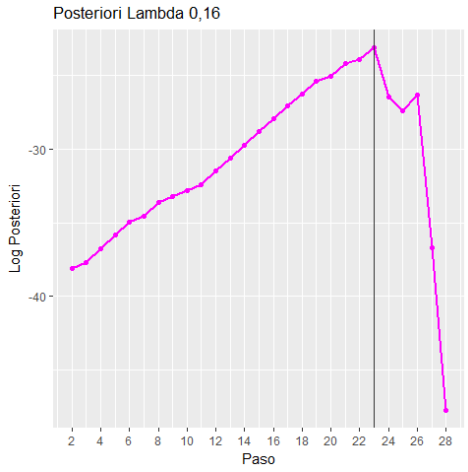
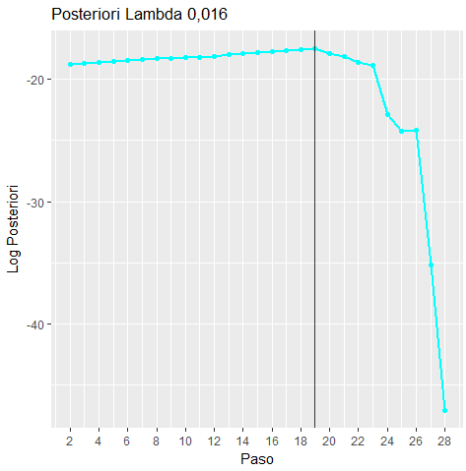


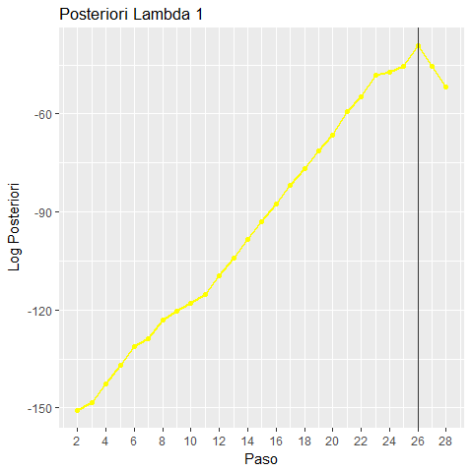
Figura 12: Logaritmo de las probabilidades a posteriori con $\lambda = 0,16$

Luego se grafican el logaritmo de las probabilidades a posterioris de los modelos que no son óptimos,

el primer modelo se sobreajusta a los datos y el segundo sobregeneraliza.



(a) Logaritmo de la posteriori en cada modelo con $\lambda = 0,016$



(b) Logaritmo de la posteriori en cada modelo con $\lambda = 1$

Figura 13: Posterioris para distintos valores de λ

El modelo que se sobreajusta a los datos tiene un máximo en el paso 19 (representado con el modelo cyan). El modelo que sobregeneraliza tiene un máximo en el paso 26 (representado con amarillo).

6.2. Estudio de casos II

En el siguiente caso de estudio se utilizó el algoritmo implementado para inferir un modelo que reconozca nombres de personas. El aprendizaje es con una muestra de 11 nombres.

La razón por la cual se utilizaron tan pocos nombres en el entrenamiento se debe al tiempo de ejecución del algoritmo.

Como se mencionó en 4.2.3 en cada paso del algoritmo se fusionan de a pares de estados, dicha acción requiere probar todas las combinaciones de estados posibles. Es decir si el algoritmo comienza con un modelo inicial M_0 que tiene 70 estados, en el primer paso tiene 2.415 candidatos de estados a fusionar. A su vez para cada candidato se re-estiman los parámetros con el algoritmo de **Viterbi** y se calcula la verosimilitud de encontrar la muestra con dicho modelo. El tiempo incurrido en dicha operación es lo que hace extremadamente lento el algoritmo.

De todas formas, mas allá que la idea es entrenar con un conjunto de nombres más grande, que puede ser llevado a cabo en un trabajo futuro, el objetivo de este caso de estudio es evaluar el funcionamiento del algoritmo en un prototipo de caso real, simulando una situación como la planteada en 4.1. El conjunto de datos utilizado fue el siguiente:

Nombres	
Florencia	Bruno
Guzman	Santiago
Pablo	Sofia
Matias	Camila
Maria	O'Hara
Maria Pia	

Cuadro 14: Nombres en el entrenamiento

Se puso en funcionamiento el algoritmo desde el primer paso hasta el último (en el que solo existe un estado) guardando en cada paso los modelos correspondientes.

Para tomar la decisión del modelo a escoger, se probó la capacidad de generalizar de cada uno de los modelos, es decir de aceptar nuevos nombres no observados en el entrenamiento, sin estar sobre-generalizando (no aceptar cualquier cadena de caracteres).

- Para probar la capacidad de generalizar del modelo, se utilizó un set de datos X^+ de 10.858 nombres extraídos de una base del Registro Civil de Montevideo, estos datos se encuentran disponibles en el sitio de datos abiertos (Catálogo de datos abiertos, 2017).

  Se utilizaron 10.858 nombres de un total de 20.000, se escogieron aquellos que sus caracteres pertenecen al alfabeto Σ determinado por todos los símbolos observados en el entrenamiento.

  $$\Sigma = \{', b, A, B, C, E, F, G, I, H, L, M, N, O, P, R, S, T, U, Z\}$$

  Aclaración: el símbolo b representa al espacio vacío.

- Para evaluar que un modelo no esté sobre-generalizando se simuló un conjunto de 10.000 cadenas de caracteres formadas aleatoriamente con símbolos de Σ.

Por ejemplo es evidente que una cadena aleatoria como $Z = \{A'EFFEA'LM\}$ no es un nombre, por lo tanto es de esperar que el modelo no la acepte.

Luego para cada modelo se registró cuántos nombres y cadenas aleatorias aceptaron cada uno. Se seleccionó el que aceptó la mayor cantidad de nombres y la menor cantidad de secuencias aleatorias.

El primer modelo M_0 representado en la figura 14 solo puede generar los 11 nombres observados en la muestra

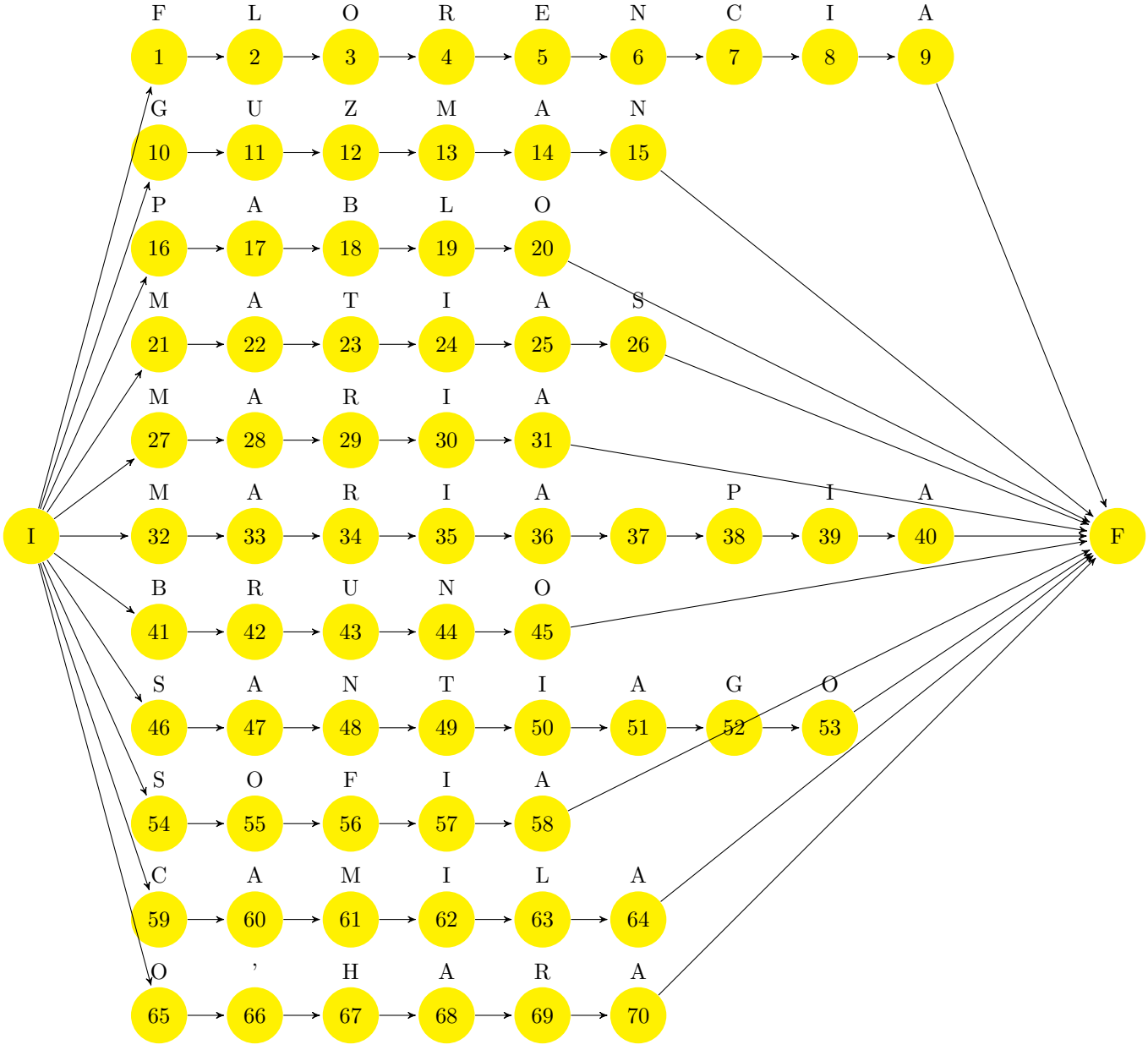


Figura 14: Modelo inicial M_0 del estudio de caso 2

En este modelo las probabilidades iniciales (en aquellos estados que emiten la primer letra de cada nombre) son las frecuencias relativas de cada uno de los nombres (es decir $1/11 = 0,09$). Todas las demás transiciones se efectúan con probabilidad uno y se emite el correspondiente símbolo con probabilidad uno. Las probabilidades iniciales son las siguientes:

Estados	1	10	16	21	27	32	41	46	54	59	65	Suma
Probabilidad Inicial	0,09	0,09	0,09	0,09	0,09	0,09	0,09	0,09	0,09	0,09	0,09	1

Cuadro 15: Probabilidades iniciales del modelo completo

Una estrategia utilizada en (Stolcke y Omohundro, 1994) para reducir los tiempos de ejecución del algoritmo, consiste en comenzar con el modelo más compacto que genere la muestra de entrenamiento. Dicho procedimiento permite evitar los primeros pasos (que tienen los mayores tiempos de ejecución).

Utilizando dicha estrategia se comenzó con un modelo inicial M_0 más compacto (que tiene 54 estados en vez de 70) y se puso en funcionamiento el algoritmo.

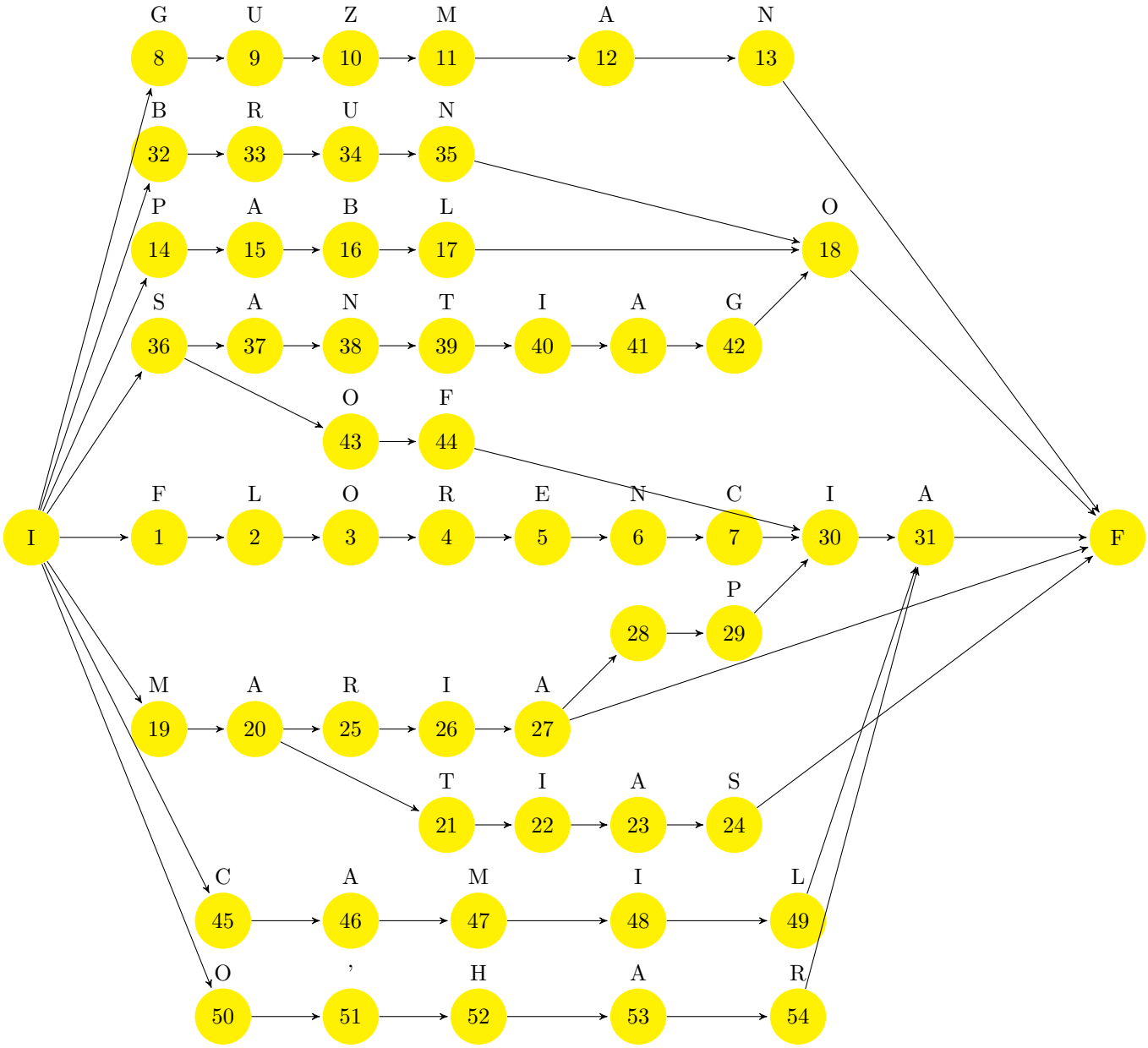


Figura 15: Modelo compacto del estudio de caso 2

En este modelo inicial cambian las probabilidades iniciales como se detalla en el siguiente cuadro:

Estado	1	8	14	19	32	36	45	50	Suma
Probabilidad Inicial	0,09	0,09	0,09	0,27	0,09	0,18	0,09	0,09	1

Cuadro 16: Probabilidades iniciales del modelo compacto

Elección del modelo

Luego de poner en funcionamiento el algoritmo se guardaron los modelos inducidos en cada paso. Para cada uno de ellos se evaluó cuantos nombres y cadenas aleatorias se aceptaron. En el cuadro 17 se muestran los últimos 8 pasos del algoritmo.

Paso	Nombres	C.Aleatorias	% Nombres	% C.Aleatorias
47	667	30	6 %	0.3 %
48	936	66	9 %	0.7 %
49	1203	109	12 %	1 %
50	2043	994	18 %	9 %
51	5565	4068	51 %	40 %
52	6835	7656	62 %	76 %
53	8369	8699	76 %	87 %
54	10885	10000	100 %	100 %

Cuadro 17: Cantidad de nombres aceptados

En el paso 48 se aceptan 936 nombres y 66 cadenas de símbolos aleatorios. Es decir el 9 % de los nombres y el 0.7 % de las secuencias aleatorias.

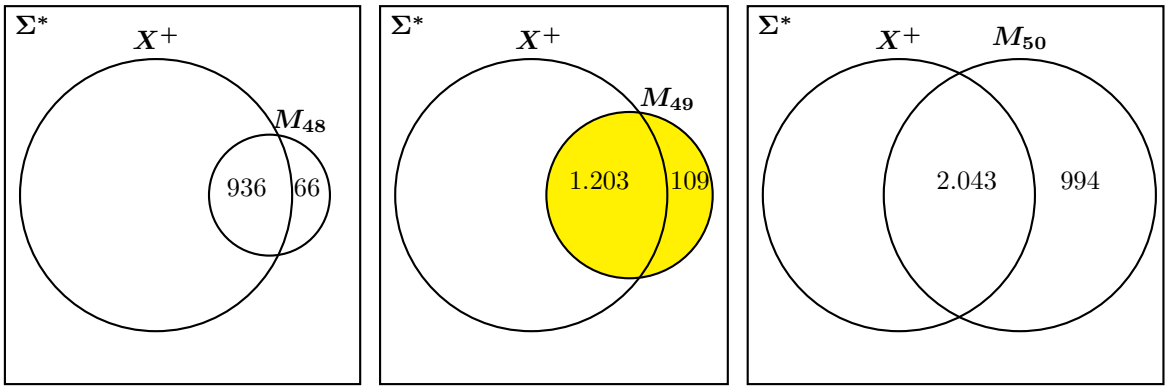
En el siguiente paso, el 49, el modelo acepta 1.203 nombres (el 12 % del total) y a su vez acepta 109 cadenas aleatorias (un 1 % del total). En comparación con el modelo anterior se están aceptando 3 % más de nombres y 0.3 % más de secuencias aleatorias. Por lo que se interpreta que el "costo" de aceptar un 3 % de nombres más (medido por el 0.3 % de secuencias aleatorias adicionales) es relativamente aceptable.

Si bien el siguiente modelo (en el paso 50) acepta 2.043 nombres (el 18 % del total) hay 994 secuencias aleatorias aceptadas (el 9 % del total). En comparación con el modelo anterior se están aceptando 6 % de nombres más, pero como contrapartida se aceptan un 8 % de secuencias aleatorias adicionales. Se concluye que el modelo acepta más nombres debido que está aceptando cualquier secuencia de símbolos.

Evidentemente el modelo en el paso 53 está sobregeneralizando ya que acepta el 87 % de las secuencias aleatorias, intepretandose como que la gran cantidad de nombres que acepta se debe a que está aceptando cualquier cadena de símbolos.

En el siguiente gráfico se ilustra para cada modelo M_{48}, M_{49} *y* M_{50}, la cantidad de secuencias aleatorias $\{\Sigma^* - X^+\}$ que acepta y la cantidad de nombres de $\{X^+\}$ que se aceptan.

La conclusión es que el modelo M_{49} es el que tiene mejor desempeño.



La estructura del modelo M_{49} obtenido se ilustra en la figura 16

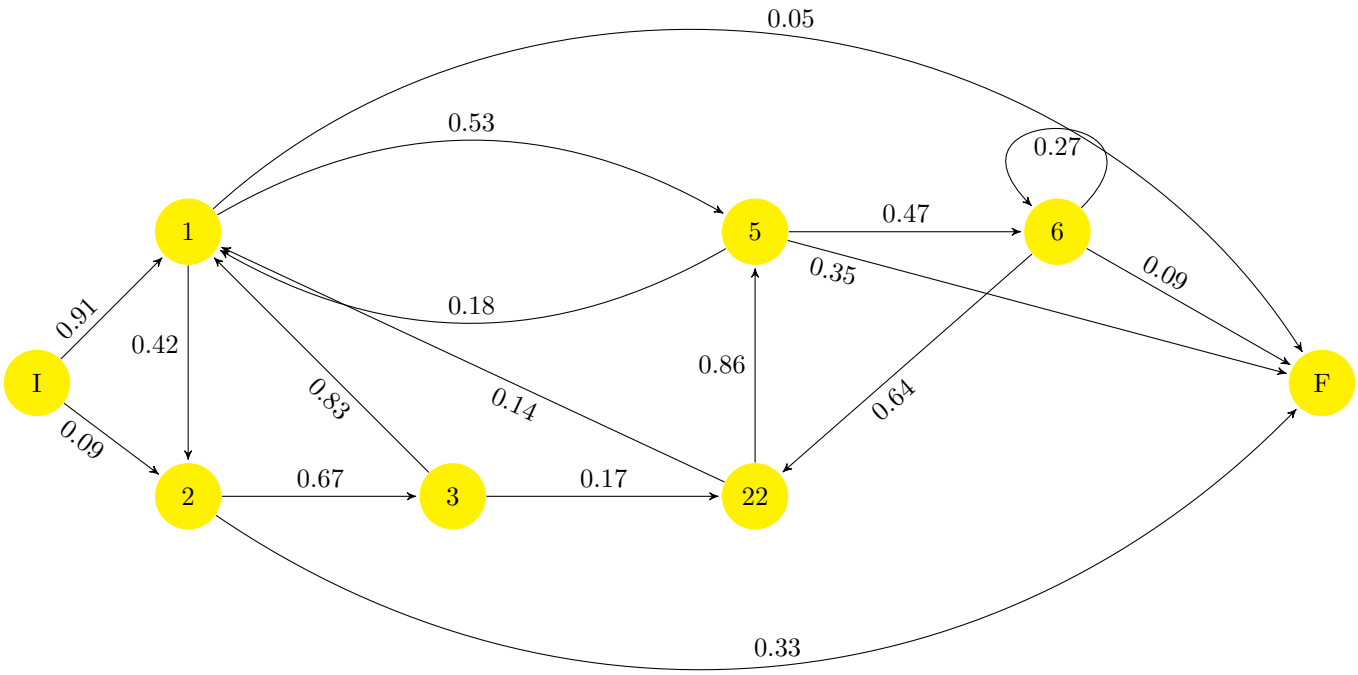


Figura 16: Modelo M_{49} obtenido en el paso 49

El cuadro 18 tiene las probabilidades de emisión en cada estado para el M_{49}. En las filas se encuentran los diferentes estados y en las columnas los diferentes símbolos de Σ. Por ejemplo el valor (0.05) de la celda que se encuentra en la fila uno columna cuatro representa la probabilidad de emitir una "B" en el estado 1.

Las celdas que están vacías representan que dicho estado tiene probabilidad 0 de emitir el correspondiente símbolo.

	’	b	A	B	C	E	F	G	I	H	L	M	N	O	P	R	S	T	U	Z
1				.05	.05		.05	.11		.05	.11	.21	.05		.05	.11	.16			
2			.11								.11			.56		.11			.11	
3	.17			.17			.17							.17					.17	.17
5			.94			.06														
6		.09			.09							.09	.27		.09	.18		.18		
22									1.0											

Cuadro 18: Probabilidades de emisión del M_{49}

En el cuadro 19 se imprimen 20 de los nuevos nombres aceptados por el modelo y una columna que indica con que estados se generan.

Nombre	Estados	Nombre	Estados
ROSA	2 3 1 5	LETICIA	1 5 6 22 1 5
SARA	1 5 1 5	RAMIRO	1 5 6 6 22 1 2
LAURA	1 2 3 1 5	SAULO	1 2 3 1 2
MARINA	1 5 6 22 1 5	SANTINO	1 5 6 6 22 1 2
CARLOS	1 5 1 2 3 1	FATIMA	1 5 6 6 22 1 5
RUBEN	2 3 1 5 6	LUCA	2 3 1 5
CESAR	1 5 1 5 6	MACARENA	1 5 1 5 1 5 1 5
RAUL	1 2 3 1	ROSEMARIE	2 3 1 5 1 5 6 22 5
LUIS	2 3 22 1	PAOLA	1 2 3 1 5
MARTIN	1 5 6 6 22 1	CARINA	1 5 6 22 1 5

Cuadro 19: 20 nuevos nombres aceptados

Por ejemplo el nombre ROSA se genera con los estados $\{2, 3, 1, 5\}$ y tiene la siguiente probabilidad:

$$P\left(\{R, O, S, A\}\,|M_{49}\right) = p\left(q_I \to q_2\right) p\left(q_2 \uparrow R\right) p\left(q_2 \to q_3\right) p\left(q_3 \uparrow O\right) p\left(q_3 \to q_1\right) p\left(q_1 \uparrow S\right) p\left(q_1 \to q_5\right) p\left(q_5 \uparrow A\right) p\left(q_5 \to q_F\right)$$

$$= (,09 * ,11)(,67 * ,17)(,83 * ,11)(,53 * ,94),35 = 0{,}00018$$

Las probabilidades de aceptar cualquier palabra bajo el modelo son relativamente bajas, incluso para los nombres que pertenecen a la muestra. Esto se debe a que las probabilidades de todas las posibles palabras generadas por el mismo deben sumar 1 (Kruegel y Vigna, 2003, p. 6). Por este motivo, es suficiente con que una palabra tenga probabilidad mayor que cero de ser generada, para ser aceptada por el mismo.

Por lo tanto el nombre ROSA es aceptado por el modelo, aunque la probabilidad de generarlo sea relativamente pequeña.

El nombre NICOLAS no es aceptado por el modelo ya que la probabilidad de generarlo es 0. Si bien se puede partir del estado 1 con probabilidad 0.91 y luego generar una N con probabilidad 0.21, necesariamente el próximo estado tiene que generar una I. El único estado que emite una I es el 22, pero como la probabilidad de ir del estado 1 al 22 es 0, la probabilidad de generar la secuencia entera es 0.

7. Conclusiones y Trabajo futuro

7.1. Conclusiones

Para este trabajo de grado se planteó como objetivo estudiar un marco teórico que provea soporte metodológico para la implementación de mecanismos de detección y prevención de ataques informáticos a las aplicaciones web.

Luego de un estudio del estado de arte se profundizó en la técnica utilizada por (Kruegel y Vigna, 2003) denominada **Inferencia de la Estructura (Structural Inference)**. Los fundamentos de dicha técnica fueron introducidos por (Stolcke y Omohundro, 1994) en el que proponen un algoritmo denominado **Fusión de Modelos Bayesiana (Bayesian Model Merging)** para inferir la estructura de HMM diferente a métodos estandar.

Metodológicamente el algoritmo propuesto combina varios formalismos de diferentes marcos teóricos como:

- *Modelos Ocultos de Markov*
- *Automátas Finitos No Deterministas*
- *Aprendizaje Bayesiano*
- *Principio de Minimo Largo de Descripción*

Una gran parte del trabajo consistió en la comprensión del marco teórico del mismo (también el de los formalismos utilizados) y posteriormente en la implementación. Luego fue utilizado y evaluado en dos estudio de casos.

En el estudio de caso del lenguaje artificial como el de (Stolcke y Omohundro, 1994) se obtuvieron buenos resultados. Se llegó al modelo capaz de generar el lenguaje regular $\{ac^*a \cup bc^*b\}$ a partir de un mínimo conjunto de entrenamiento, constituido por 8 cadenas que pertenecen al mismo.

Uno de los propósitos de experimentar con este estudio de caso fue visualizar conceptos que se venían estudiando teóricamente. La representación con varias herramientas gráficas permitieron conocer que ocurre en cada etapa de la construcción de los modelos y visualizar ciertas decisiones tomadas por el algoritmo en la inferencia del lenguaje.

El objetivo planteado en el segundo estudio de caso que aprende nombres de personas fue poner a prueba el algoritmo en un ambiente que simule una situación real como la planteada en 4.1.

El motivo por el cual no se implementó en un caso real fue por el excesivo tiempo de ejecución del algoritmo al utilizar varios ejemplos en el entrenamiento. Esto se debe a que el mismo en cada paso fusiona pares de estados, dicha acción requiere probar todas las combinaciones de estados posibles, lo que consume una gran cantidad de tiempo.

La implementación se realizó con 11 nombres en el entrenamiento. La primer solución propuesta fue utilizar un modelo más compacto que solamente genere las cadenas observadas en el inicio como el de la figura 15 en vez de comenzar con un modelo con 70 estados.

Comenzando con el modelo compacto de 54 estados el algoritmo evalúa 1.431 posibles candidatos para realizar la primer fusión. Para cada candidato reestima los parámetros con el algoritmo de Viterbi

y calcula la probabilidad a posteriori para dicho modelo. Dicha operación que realiza la primer fusión tiene un tiempo de 1 hora y 50 minutos.

De todas formas el resultado obtenido fue satisfactorio. Se indujo un modelo capaz de reconocer 1.203 nombres a partir de 11 nombres de entrenamiento sin estar generalizando (aceptando 109 cadenas aleatorias).

Ambos experimentos mostraron la capacidad del algoritmo de aprender cadenas nuevas a partir de un ejemplo de entrenamiento.

7.2. Trabajo futuro

Dadas las implicancias del tiempo de ejecución no fue posible evaluarlo en una situación real con más cantidad de datos, lo que puede realizarse en un trabajo futuro.

Una posible solución es utilizar algunas funciones implementadas en C++ desde el código en R, con el objetivo de acelerarlo. Esto se puede hacer con la subrutina Rcpp, desarrollada por Dirk Eddelbuettel y Romain Francois.

Existen otras alternativas como: utilizar otro lenguaje mas apropiado (como C o C++ por ejemplo) para implementar completamente el algoritmo o la utilizacion de otro tipo de tecnología como el procesamiento en paralelo en diferentes CPUs.

En particular, si usamos un lenguaje como C++, la subrutina StochHMM tiene funciones implementadas para HMM. En GitHub (GitHub, 2018) sus desarrolladores presentan la comparación del desempeño de algunas de sus funciones con respecto a las de la subrutina HMM de R, y la diferencia en los tiempos de ejecución es muy pronunciada. En uno de los ejemplos que exponen utilizan una secuencia de 3.000 símbolos y comparan el algoritmo de Viterbi con el de la subrutina en C++. En la subrutina StochHMM (de C++) tarda un tiempo de 0.18 segundos, mientras que la función de la subrutina de R tarda 21.09 segundos. Es decir, la de C++ es más de 100 veces más rápida que la subrutina de R.

Glosario

Ataques multi-etapas Es una intrusión que normalmente implica un ataque inicial, seguido por la instalación de una parte adicional de códigos maliciosos. Un ejemplo es un troyano que descarga e instala *adware* (Glosario de Seguridad, 2017).

Cryptanalysis es la parte de la criptología que se dedica al estudio de sistemas criptográficos con el fin de encontrar debilidades en los sistemas y romper su seguridad sin el conocimiento de información secreta.

Etiquetado gramatical Es el etiquetado gramatical (conocido también por su nombre en inglés, part-of-speech tagging, POS tagging o POST) es el proceso de asignar (o etiquetar) a cada una de las palabras de un texto su categoría gramatical (Ondetti *et al.*, 1971).

Firewall Es una aplicación de seguridad diseñada para bloquear las conexiones en determinados puertos del sistema, independientemente de si el tráfico es benigno o maligno.

ModSecurity Es un Web Application Firewall estandar de facto en la comunidad abierta, provee protección contra diversos ataques hacia aplicaciones Web y permite monitorizar tráfico HTTP, así como realizar análisis en tiempo real sin necesidad de hacer cambios a la infraestructura existente (ModSecurity, 2017).

Procesos estocásticos es un concepto matemático que sirve para caracterizar una sucesión de variables aleatorias (estocásticas) que evolucionan en función de otra variable.

Teorema de Bayes expresa la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de sólo A.

Verosimilitud es una función de los parámetros de un modelo estadístico que permite realizar inferencias acerca del valor de dichos parámetros a partir de un conjunto de observaciones.

Web Application Firewall es un Firewall que ha sido diseñado específicamente para proteger aplicaciones web con la capacidad de detectar e identificar patrones que podrían derivar en un ataque contra la aplicación. El mismo se instala entre los usuarios y la aplicación y se encarga de analizar todos los pedidis.

Zero-day attacks estos tipos de ataques aprovechan vulnerabilidades no conocidas hasta el momento de una aplicación para atacar un sistema.

Siglas

AFD Autómata Finito Determinístico.

AFNDE Autómata Finito No Determinístico Estocástico.

AP Automáta Probabilístico.

APT *Advanced Persistent Threats*.

CERTuy Centro de Respuestas a Incidentes de Seguridad del Uruguay.

DBF *Database Firewall*.

ECML *European Conference on Machine Learning*.

EM *Expectation Maximization*.

HMM Hidden Markov Models.

HTTP *HyperText Transfer Protocol*.

IDS *Intrusion Detection System*.

MDL Mínimo Largo de Descripción.

NIPS *Network Intrusion Prevention System*.

OWASP *Open Web Application Security Project*.

PKDD *European Conference on Principles and Practice of Knowledge Discovery in Databases*.

SVM *Support Vector Machines*.

URI *Uniform Resource Identifier*.

WAF Web Application Firewall.

Referencias

Ariu, D. (2010). Host and network based anomaly detectors for http attacks. *Diss. PhD thesis, PhD Program in Electronic and Computer Eng.(DRIEI), University of Cagliari, Italy.*

Barracuda (2017). Barracuda networks products, consulting and information — barraguard.com. http://www.barraguard.com. (Recuperado el 11/07/2017).

Catálogo de datos abiertos (2017). Partidas de registro civil de montevideo - conjuntos de datos - catálogo de datos abiertos. https://catalogodatos.gub.uy/dataset/partidas-de-registro-civil-de-montevideo. (Recuperado el 11/07/2017).

Chandola, V., Banerjee, A., y Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15.

Corona, I., Ariu, D., y Giacinto, G. (2009). Hmm-web: A framework for the detection of attacks against web applications. En *Communications, 2009. ICC'09. IEEE International Conference on*, pp. 1–6. IEEE.

CypherSec (2017). Cyphersec technologies llp. http://cyphersec.in/waf.html. (Recuperado el 11/24/2017).

Dempster, A. P., Laird, N. M., y Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

Dua, S. y Du, X. (2016). *Data mining and machine learning in cybersecurity*. CRC press.

Dupont, P., Denis, F., y Esposito, Y. (2005). Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern recognition*, 38(9):1349–1371.

Gallagher, B. y Eliassi-Rad, T. (2009). Classification of http attacks: a study on the ecml/pkdd 2007 discovery challenge. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

GitHub (2018). Comparison korflab/stochhmm wiki github. https://github.com/KorfLab/StochHMM/wiki/comparison. (Recuperado el 02/20/2018).

Glosario de Seguridad (2017). Glosario de seguridad. https://www.symantec.com/es/mx/theme.jsp?themeid=glosario-de-seguridad. (Recuperado el 11/15/2017).

Goseva-Popstojanova, K., Anastasovski, G., y Pantev, R. (2012). Using multiclass machine learning methods to classify malicious behaviors aimed at web systems. En *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, pp. 81–90. IEEE.

Hopcroft, J. E., Motwani, R., y Ullman, J. D. (2008). *Teoría de autómatas, lenguajes y computación*. Addison Wesley.

HTTP (2017). Rfc 2616 - hypertext transfer protocol – http/1.1. https://tools.ietf.org/html/rfc2616. (Recuperado el 12/06/2017).

ICT4V (2017). ciberseguridad — localhost. http://www.ict4v.org/page/ciberseguridad#scrollTop=519. (Recuperado el 11/13/2017).

Kruegel, C. y Vigna, G. (2003). Anomaly detection of web-based attacks. En *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 251–261. ACM.

ModSecurity (2017). Modsecurity: Open source web application firewall. https://www.modsecurity.org/. (Recuperado el 11/07/2017).

Ondetti, M. A., Williams, N. J., Sabo, E., Pluscec, J., Weaver, E. R., y Kocy, O. (1971). Angiotensin-converting enzyme inhibitors from the venom of bothrops jararaca. isolation, elucidation of structure, and synthesis. *Biochemistry*, 10(22):4033–4039.

Oracle (2017). Oracle audit vault and database firewall. https://go.oracle.com. (Recuperado el 11/07/2017).

OWASP (2017). Category:owasp application security verification standard project - owasp. https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project. (Recuperado el 11/15/2017).

OWASP Top Ten (2017). Category:owasp top ten project - owasp. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. (Recuperado el 11/07/2017).

OWASP Virtual Patching (2017). Virtual patching best practices - owasp. https://www.owasp.org/index.php/Virtual_Patching_Best_Practices. (Recuperado el 11/07/2017).

Patcha, A. y Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470.

Pavey, D. (1995). Safer c: Developing software for high-integrity and safety-critical systems. les hatton. published by mcgraw-hill book company europe, maidenhead, uk, 1995. isbn 0 07 707640 0, 229 pages. price:£ 22.95, soft cover. *Software Testing, Verification and Reliability*, 5(3):203–204.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *The Annals of statistics*, pp. 416–431.

Roesch, M. (2005). Light weight intrusion detection for networks. En *Proceedings of LISA*, volumen 99.

Salton, G., Wong, A., y Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Scientific Software Development - Dr. Lin Himmelmann y www.linhi.com (2010). *HMM: HMM - Hidden Markov Models*. R package version 1.0.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

Sophos XG Firewall (2017). Sophos xg firewall test drive. http://test-drive-now.azurewebsites.net. (Recuperado el 11/07/2017).

Stolcke, A. y Omohundro, S. (1994). Inducing probabilistic grammars by bayesian model merging. *Grammatical inference and applications*, pp. 106–118.

Suricata (2017). Suricata — open source ids / ips / nsm engine. https://suricata-ids.org/. (Recuperado el 11/07/2017).

Vamsidhar, T. (2013). Intrusion detection system for web applications with attack classification. *Journal of Global Research in Computer Science*, 3(12):44–50.

Web Design Word Cloud Photo (2017). Web design word cloud photo. https://www.featurepics.com/online/Web-Design-Word-Cloud-1650406.aspx. (Recuperado el 11/24/2017).

8. Anexo

8.1. Lenguajes Regulares

Los lenguajes regulares son aquellos que pueden escribirse mediante un autómata finito. Otra forma de describir un lenguaje regular es con la notación algebraica conocida como expresiones regulares. Lo primero que hay que entender sobre un autómata finito es como decide si aceptar o no una secuencia de símbolos de entrada. El lenguaje del autómata finito es el conjunto de todas las cadenas que acepta .

Las expresiones regulares pueden definir de forma exacta los mismos lenguajes que describen los distintos tipos de autómatas: Los lenguajes regulares.

Para tener un lenguaje L se necesita un alfabeto (símbolos) y reglas que manipulen ese alfabeto (la gramática), para generar cierto conjunto de secuencias de símbolos del alfabeto (que puede ser finito o infinito).

- **Alfabeto**, que es un conjunto de símbolos finito, convencionalmente denotado con la letra Σ, por ejemplo puede ser:
  - $\Sigma = \{a, b, ..., z\}$ el conjunto de todas las mínusculas
  - El conjunto de todos los caracteres ASCII, el conjunto de los caracteres ASCII imprimibles, etc
- **Cadena de caracteres**, también se denomina palabra, que es una secuencia finita de símbolos de algún alfabeto, por ejemplo $\{hola\}$ es una cadena de caracteres del conjunto de las minúsculas
  - La cadena vacía es aquella que no tiene ningún símbolo, puede construirse por cualquier alfabeto, el vacío se designa con la letra ϵ
  - La longitud de una cadena es el número de posiciones ocupadas por símbolos dentro de la cadena, utilizando la notación estándar para identificar la longitud de una cadena w es $|w|$, por ejemplo $|\{hola\}| = 4$
  - Potencias de un alfabeto, si Σ es un alfabeto, se pueden expresar todas las cadenas de determinado largo que se pueden formar, combinando de todas las formas posibles los símbolos del alfabeto, utilizando la notación exponencial.
    $\Sigma^0 = \{\epsilon\}$, es la unica cadena de largo 0, para todo Σ,
    Por ejemplo si $\Sigma = \{a, b\} \Rightarrow \Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$, $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$
    Al conjunto de todas las cadenas posibles de un alfabeto se denomina Σ^* (Clausura de Kleene), expresada como:

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

Si queremos excluir la cadena vacía, definida como la clausura positiva

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$$

- Concatenación de cadenas, por ejemplo si tengo una cadena $x = \{h, o\}$ y otra cadena $y = \{la\}$, la concatenación de cadenas es $x + y = \{hola\}$

- **Lenguajes**, para un determinado alfabeto Σ, un lenguaje L es un conjunto de cadenas de Σ^*. Formalmente si Σ es un alfabeto y $L \subseteq \Sigma^*$, entonces L es un lenguaje.
  En (Hopcroft *et al.*, 2008) se plantea el ejemplo de que los lenguajes habituales pueden interpretarse como conjunto de cadenas, por ejemplo el inglés, es un conjunto de palabras correctas, que es un conjunto de cadenas del alfabeto , de igual manera un lenguaje de programación, en el que que las palabras correctas son un subconjunto de las cadenas que pueden formarse con la tabla ASCII.

- **Problemas**, un problema es la decisión de si una cadena pertenece a determinado lenguaje, si L es un lenguaje del alfabeto Σ, decidir si w que es una cadena de Σ^* pertenece o no a L.

8.2. Códigos implementados

8.2.1. Función m0

Función que arma el modelo inicial M_0 a partir de datos de entrada $\mathbf{X}$.

```
m0=function(f){
#primero hago una lista en donde pongo cada nombre como un vector y cada letra un caracter
g=list()
for (i in 1:nrow(f)){
g[[i]]=unlist(strsplit(f[i,],split = ""))
}

#Pmodelo M0 a partir de los datos
#creo un vector con todos los simbolos observados concatenados
j=NULL
for (i in 1:length(g)) {j=c(j,unlist(g[i])) }
#creo un vector con todos los simbolos concatenados y al lado el estado
#que lo genero
j2=as.data.frame(cbind(j,cumsum(!j==".")),stringsAsFactors = F)
j2[j2$j==".",2]="999"

#vector con todos los estados generadores
x=as.numeric(j2[-nrow(j2),2])
# vector de los estados sin el primer elemento y con una F al final
y=as.numeric(j2[-1,2])
#junto los dos vectores para tener las transiciones
#para tener las transiciones , no tomo en cuenta cuando arranca de F
e=which(x==999)
#probabilidades iniciales
sp=c(1,y[e])
sp1=rep(0,length(unique(c(x,y))))
sp1[as.numeric(sp)]=1
sp1=sp1*(1/length(sp))
x=x[-e];y=y[-e]

lev=unique(sort(c(x,y)))
y<-factor(y,levels =lev)
x<-factor(x,levels =lev)

t=prop.table(table(x,y),margin = 1)
t[t=="NaN"]=0
#matriz de emisiones
e1=as.matrix(prop.table(table(j2[,2],j2[,1]),1))
e1=e1[mixedorder(rownames(e1)), ]
```

```
#estados
s=rownames(e1)
#alfabeto
al=colnames(e1)
#primero se identifica el modelo inicial
hmm = initHMM(States = s,Symbols = al,transProbs=t,emissionProbs=e1,startProbs =sp1)
return(hmm)
}
```

8.2.2. Función vero

Función que calcula la verosimilitud de un conjunto de datos $\mathbf{X}$.

```
vero=function(hmm,f){
g=list()
for (i in 1:nrow(f)){
g[[i]]=unlist(strsplit(f[i,],split = ""))
}

v=1
j=NULL
for (i in 1:length(g)) {v=v*sum((FW1=exp(forward(hmm = hmm
,(unlist(g[i])))))[,length((unlist(g[i])))]) }

return(v)
}
```

8.2.3. Función priori

Función que calcula la probabilidad de priori estructural $P(M_S)$

```
priori=function(hmm){
p_mod=((nrow(hmm$transProbs))^(-sum((hmm$transProbs[1,])>0)))*((nrow(hmm$emissionProbs))^(-sum((hmm$e

for (z in 2:nrow(hmm$transProbs)){
p_mod=p_mod*((nrow(hmm$transProbs))^(-sum((hmm$transProbs[z,])>0)))*((ncol(hmm$emissionProbs))^(-sum(
}
return(p_mod)
}
```

8.2.4. Función reesViterbi

Función que hace la estimación de los parámetros θ_M (con el algoritmo de Viterbi)

```
reesViterbi=function(hmm,f){
```

```
g=list()
for (i in 1:nrow(f)){
g[[i]]=unlist(strsplit(f[i,],split = ""))
}

j=NULL
for (i in 1:length(g)) {j=c(j,unlist(g[i])) }

j1=NULL
for (i in 1:length(g)) {j1=c(j1,viterbi(hmm,unlist(g[i]))) }
j2=cbind(j,j1)

#vector con todos los estados generadores
x=as.numeric(j1[-length(j1)])
# vector de los estados sin el primer elemento y con una F al final
y=as.numeric(j1[-1])
#junto los dos vectores para tener las transiciones
#para tener las transiciones , no tomo en cuenta cuando arranca de F
e=which(x==999)
#probabilidades iniciales
sp=c(x[1],y[e])
g1=as.data.frame(table(sp),stringsAsFactors = F)
sp1=rep(0,length(unique(c(x,y))))

x=x[-e];y=y[-e]

lev=unique(sort(c(x,y)))
y<-factor(y,levels =lev)
x<-factor(x,levels =lev)

t=prop.table(table(x,y),margin = 1)
t[t=="NaN"]=0
#matriz de emisiones
e1=as.matrix(prop.table(table(j2[,2],j2[,1]),1))
e1=e1[mixedorder(rownames(e1)),]
#estados
s=rownames(e1)
#alfabeto
al=colnames(e1)

#probabilidades iniciales
sp2=data.frame(cbind(s,sp1),stringsAsFactors = F)
sp2[sp2$s %in% (g1$sp),]$sp1=as.numeric(g1$Freq)
```

```
sp2$sp1=as.numeric(sp2$sp1)
sp2$sp1=sp2$sp1*(1/length(sp))

#primero se identifica el modelo inicial
hmm=initHMM(States = s,Symbols = al,transProbs=t,emissionProbs=e1,startProbs =sp2$sp1)
return(hmm)
}
```

8.2.5. Función modmeg

Función que decide cuales estados se van a colapsar

```
modmeg=function(hmm,f) {
#en u voy a ir guardando las probabilidades a posteriori
u=NULL
#en ff vo a ir guardando los indices que se van a juntar
ff=NULL
#pruebo mergear cada uno con cada uno
for (i in 1:(length(hmm$States)-1)){
j=i+1
while (   j<length(hmm$States) )
{
TP=hmm$transProbs
#elimino fila y columna j
TP[,i]=TP[,i]+TP[,j]
TP=TP[,-j]
TP[i,]=TP[i,]+TP[j,]
TP=TP[-j,]

TP[i,]=TP[i,]/sum(!TP[i,]==0)
#TP[i,]=TP[i,]/sum(TP[i,])

##reestimo emisiones
EP=hmm$emissionProbs
EP[i,]=(EP[i,]+EP[j,])/sum((EP[i,]+EP[j,]))
EP=EP[-j,]

#restimo probabilidades iniciales
SP=hmm$startProbs
SP[i]=(SP[i]+SP[j])
SP=SP[-j]

#corro el modelo con un nuevo merge
```

```
hmm2 = initHMM(hmm$States[-j],hmm$Symbols,
transProbs=TP,
emissionProbs=EP,
startProbs =SP)

#reestimo probabilidades
hmm2=reesViterbi(hmm2,f)

#print(c(i,j))
ff=c(ff,paste(i,j,sep="-"))
j=j+1
print(tail(ff,1))

v=vero(hmm2,f)

u=c(u,v)
}
}
#identifico el candidato a mergiar
e=(which.max(u))

#i y j son los candidatos a mergiar
#cuantos digitos tienen los dos estados
estados=as.vector(unlist(strsplit(ff[e],"-")[1]))

i=as.numeric(estados[1]);j=as.numeric(estados[2])

TP=hmm$transProbs
TP[,i]=TP[,i]+TP[,j]
TP=TP[,-j]
TP[i,]=TP[i,]+TP[j,]
TP[i,]=TP[i,]/sum(!TP[i,]==0)
TP[i,]=TP[i,]/sum(TP[i,])
TP=TP[-j,]

##reestimo emisiones
EP=hmm$emissionProbs
EP[i,]=(EP[i,]+EP[j,])/sum((EP[i,]+EP[j,]))
EP=EP[-j,]

#restimo probabilidades iniciales
SP=hmm$startProbs
SP[i]=(SP[i]+SP[j])
```

```
SP=SP[-j]

#corro el modelo con un nuevo merge
hmm3 = initHMM(hmm$States[-j],hmm$Symbols,
transProbs=TP,
emissionProbs=EP,
startProbs =SP)
#reestimo probabilidades
hmm3=reesViterbi(hmm3,f)

return(hmm3)
}
```

8.2.6. Función baymodmeg

Función que decide cuando detener la generalización y retorna el modelo final

```
baymodmeg=function(hmm,f,lam){
estados=(length(hmm$States)-1)
i=1
s=0
v0=vero(hmm,f)
p0=priori(hmm)

sink("historia.txt")
print(paste("paso=;",i,"; largo=;",length(hmm$States),
+ "; Log vero=;", log(v0),"; Log priori=;", log(p0),
+ "; hora=;",Sys.time()))
sink()

while  (i<estados & s==0){
#busca el mejor candidato
hmm1=modmeg(hmm,f)
v1=vero(hmm1,f)
p1=priori(hmm1)
{
if ( (log(v1)+lam*log(p1))>=(log(v0)+lam*log(p0))) {
i=i+1
hmm=hmm1
v0=v1
p0=p1
}
else {s=1}
}
```

```
sink(file = "historia.txt", append = TRUE)
print(paste("paso=;",i,"; largo=;",length(hmm$States),
+ "; Log vero=;", log(v1),"; Log priori=;", log(p1), "; hora=;",Sys.time()))
sink()

b=paste("modelos/hmm_",i,".RData",sep = "")
save(hmm,file =b)
}
return(hmm)
}
```