Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

Tipo de artículo: Artículo original
Temática: Inteligencia artificial
Recibido: 08/07/2019 | Aceptado: 20/08/2019 | Publicado: 22/08/2019

Algoritmo *Random Forest* para la detección de fallos en redes de computadoras

Random Forests Algorithm for fail detecting on computer´s networks

Selianne Labañino Urbina[1*], Hugo Alberto Valencia Zayas[2], Orlando Grabiel Toledano López[3]

[1] Universidad de las Ciencias Informáticas. Carretera de San Antonio de los Baños km 2 ½, Torrens, La Lisa, La Habana, Cuba. slabanino@estudiantes.uci.cu.

[2] Universidad de las Ciencias Informáticas. Carretera de San Antonio de los Baños km 2 ½, Torrens, La Lisa, La Habana, Cuba. havalencia@estudiantes.uci.cu.

[3] Universidad de las Ciencias Informáticas. Carretera de San Antonio de los Baños km 2 ½, Torrens, La Lisa, La Habana, Cuba. ogtoledano@uci.cu.

[*] Autor para correspondencia: ogtoledano@uci.cu

Resumen

El uso de técnicas para el procesamiento de datos masivos constituye una gran necesidad para la sociedad y el hombre. Cada día son más abundantes y heterogéneos los dispositivos que se conectan e intercambian información al interactuar con la red de servicios integrados, originando con su interacción, un gran cúmulo de datos que se puede procesar mediante la aplicación de técnicas de aprendizaje automático. En las redes de datos, donde se ofrecen servicios a los usuarios, es de vital importancia el mantenimiento de la disponibilidad de los mismos, para que puedan ser utilizados en cualquier momento. En muchos casos la disponibilidad de los servicios puede estar afectada por factores de los componentes de la red y los servidores, pues estos dispositivos pueden fallar si ocurren determinados eventos en la red por su utilización. Para dar respuesta a dicha problemática se presenta en este trabajo la aplicación del algoritmo *Random Forest* como clasificador para resolver tareas predictivas. Para su implementación se hace uso de la herramienta de procesamiento *Apache Spark* y sus bibliotecas de aprendizaje automático Spark MLlib, sobre la

plataforma Java 8. De forma general se explica cómo aplicar el algoritmo para predecir fallos en la disponibilidad de los servicios de la red.

Palabras clave: *Apache Spark*, aprendizaje automático, *Random Forest.*

Abstract

The use of mass data processing techniques is a great necessity for society and man. The devices that connect and exchange information by interacting with the network of integrated services are becoming more abundant and heterogeneous every day, originating with their interaction a large accumulation of data that can be processed through the application of automatic learning techniques. In data networks, where services are offered to users, it is vitally important to maintain the availability of the same, so that they can be used at any time. In many cases the availability of services may be affected by factors of network components and servers, as these devices may fail if certain events occur in the network due to their use. In order to respond to this problem, this paper presents the application of the Random Forest algorithm as a classifier to solve predictive tasks. For its implementation, the Apache Spark processing tool and its automatic learning libraries Spark MLlib are used on the Java 8 platform. In general, it explains how to apply the algorithm to predict failures in the availability of network services.

Keywords: *Apache Spark, Machine Learning, Random Forest.*

Introducción

En los últimos años el uso creciente y masivo de las Tecnologías de la Información y las Comunicaciones (TIC) ha provocado gran impacto en la sociedad. Con el desarrollo de los dispositivos móviles y las tecnologías de conexión inalámbrica, es muy común hoy en día que las personas interactúen constantemente en su cotidianidad con la Internet y/o Red Digital de Servicios Integrados (RDSI; en inglés: ISDN) sin importar el lugar o momento. Toda esta interacción tecnológica origina huellas digitales que van acumulándose hasta originar cantidades enormes de datos (Hussain, Akif, Tamayo, & Safdar, 2018).

Para lograr adquirir y analizar tanta información surge el área de conocimiento conocida como *Big Data*, el cual se caracteriza por conjuntos de datos cuyo tamaño va más allá de la capacidad de captura, almacenado, gestión y análisis de las herramientas de base de datos. El *Big Data* puede ser utilizado, para la mejora de procesos permitiendo la simplificación y el control de procesos actuales (reducción de costos y esfuerzos). Además, para identificar patrones

de comportamiento complejos que permitan detectar ataques informáticos en tiempo real. Enfocado al soporte a la toma de decisiones puede ser también una potente herramienta descubriendo información que antes podría estar oculta, a través de la aplicación de algoritmos automáticos de minería de datos. Todas estas ventajas se pueden agrupar en una principal de las que derivan el resto: "obtener más información/conocimiento" a partir del análisis profundo de los datos (Manyika & Chui, Michael and Brown, 2011).

El incremento de la complejidad de las redes de datos y el volumen de servicios que estas ofrecen, en conjunto con la cantidad de dispositivos heterogéneos que se conectan, hacen que se genere un cúmulo enorme de datos que pueden ser aprovechados para extraer información oculta que permita administrar de forma eficiente la red, incrementar la calidad de los servicios y actuar en tiempo real ante ataques informáticos y fallos (Shiomoto, 2013).

Este volumen enorme de información que genera el uso y el tráfico diario de datos proviene de diferentes elementos que conforman la red, tales como: datos de configuración sobre la conectividad de dispositivos de interconexión, sistemas de detección de intrusos (IDS, por sus siglas en inglés), eventos de alarmas generados por sistemas que realizan el monitoreo y/o control de la red ante fallos o funcionamientos anómalos y las propias acciones de los usuarios sobre los servicios disponibles en la red (Frydenberg, 2015; Landset, Khoshgoftaar, Richter, & Hasanin, 2015).

Teniendo en cuenta el carácter heterogéneo y variado de los elementos que conforman la red, cada día el análisis de los logs para la detección de fallos por parte de los data-driven, se hacía muy engorroso y estaba condicionado por errores humanos. Estos fallos pueden ocasionar interrupciones de los servicios y traer consigo una disminución de los indicadores de calidad en los mismos (Shiomoto, 2013).

A continuación se muestra un esquema general que indica el flujo de información que se genera en la red, los cuales pueden ser analizados por sistemas que permitan el procesamiento masivo de los datos:

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

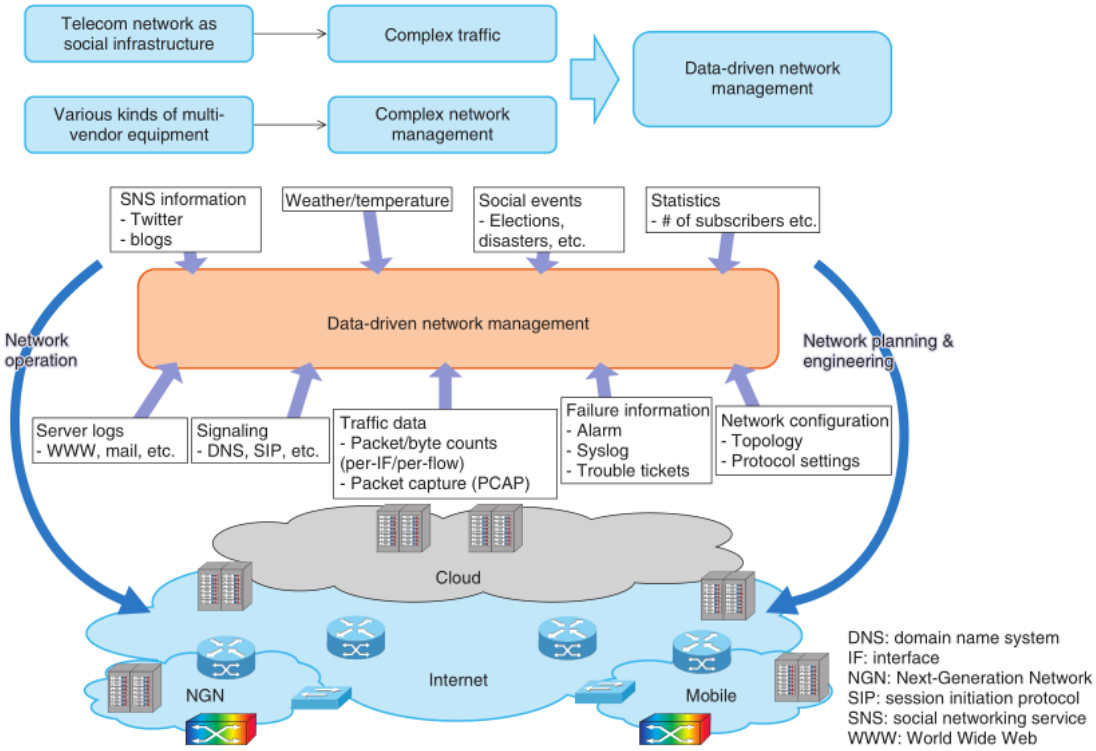

Figura 1. Flujo de información que se genera en una red para ser analizado por un data-driven (Shiomoto, 2013)

Con las técnicas de aprendizaje automático en el contexto del volumen masivo de datos, surge la analítica avanzada de datos, la cual hace referencia a las modernas y sofisticadas técnicas empleadas para descubrir patrones relevantes dentro de los datos, con el fin de ayudar a las organizaciones a tomar decisiones inteligentes que les sirvan para cumplir con metas operativas y específicas del negocio (Elibeth Eduardo, 2017).

Big Data posee tres métodos analíticos los cuales aplican técnicas de Inteligencia Artificial, y son los siguientes (Elibeth Eduardo, 2017):

- **Descriptivo**. Interpreta datos históricos para determinar eventos ocurridos.
- **Predictivo**. Busca resultados que predicen la ocurrencia de eventos futuros con base en patrones históricos que a veces se combinan con datos externos.

- **Prescriptivo**. Predice múltiples resultados para un escenario específico, de acuerdo con las acciones tomadas. La idea es mostrar cómo un conjunto diferente de acciones afectará la situación y dirigirá al usuario hacia la mejor opción posible.

Para un operador de redes, resulta de gran efectividad incorporar algoritmos y técnicas avanzadas de aprendizaje automatizado. Algunos ejemplos de aplicaciones analíticas incluyen (Elibeth Eduardo, 2017):

- Garantía de redes o servicios.
- Seguridad de redes; gestión de tráfico.
- Planificación de capacidad.

En la medida que las redes se vuelven más complejas y dinámicas, los operadores no solamente deben escalarlas con fines de crecimiento, sino también para poder adaptarlas en tiempo real para gestionar las demandas de tráfico que a menudo son impredecibles y cambian súbitamente (Elibeth Eduardo, 2017). Para ello se hace necesario el uso de técnicas de aprendizaje automático que permitan el reconocimiento de patrones en los datos que se generan con el uso de la red.

Los métodos analíticos-predictivos, aplicados al estado de la red, permiten a los operadores o administradores mantenerlas en óptimas condiciones y cumplir de manera eficiente con los niveles de servicios establecidos (Elibeth Eduardo, 2017).

El presente trabajo se centra en la aplicación de técnicas de *Big Data* para realizar tareas predictivas que permitan acometer acciones para garantizar la disponibilidad de las redes y servicios, mediante la predicción del estado de la red. Para ello se hace uso de la plataforma *Apache Spark* y se ejecuta en dicha plataforma el algoritmo *Random Forest* como clasificador.

Materiales y métodos o Metodología computacional

En este acápite se explican los elementos relacionados con el proceso de gestión de fallos en la red de forma proactiva, determinando a partir de un escenario, posibles anomalías o fallos que tendrá la red en un determinado instante de tiempo. Para ello se explica el uso de la herramienta *Apache Spark* y con esta se aplica un conjunto de entrenamiento para el algoritmo *Random Forest* con el fin de aplicarlo para predecir un fallo en la red. Se explica el proceso general de predicción de fallos, y el conjunto de datos elegido para ejecutar el algoritmo.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

Proceso de predicción de fallos en la red

Es un proceso también denominado gestión de fallos proactiva el cual tiene como objetivo predecir dichos fallos momentos antes de que ocurran. En una red de datos, no basta con asegurar la seguridad y la disponibilidad de los servicios, también deben existir elementos que permitan un modo de actuación proactivo que asegure la completa disponibilidad de la red (Esteso, 2015)

Para ello se realizan los siguientes pasos, de forma general (Esteso, 2015)

1. **Predicción de fallos online**: consiste en identificar posibles situaciones en las que el sistema puede fallar. Normalmente este punto se lleva a cabo mediante técnicas de data mining y machine learning.
2. **Diagnóstico**: una vez que se sabe que va a producirse un fallo, hay que saber dónde y por qué.
3. **Planificación**: definición de las medidas a tomar para responder al futuro fallo del sistema.
4. **Reacción**: ejecutar las medidas propuestas en el paso anterior.

Este esquema de pasos de forma general provee de un marco de trabajo para que el administrador de redes organice la forma en la que se establecerá el proceso de predicción de fallos. Se debe tener en cuenta los elementos que conforman la red e identificar los posibles fallos que puede tener cada activo. Con estos posibles fallos se debe crear una vista minable de los datos recogidos en la red y definir el formato de su representación, luego elegir el algoritmo de minería de datos más adecuado para resolver el problema. Con la salida del algoritmo aplicado se debe hacer un diagnóstico e interpretación de los resultados; y en caso de que el fallo se produzca tener bien definido el plan de medidas a tomar y su ejecución, la cual puede ser de forma manual o automática.

Algoritmo *Random Forest*

Random Forest es un algoritmo de clasificación muy usado por los científicos de datos, el mismo consiste en combinación de árboles predictores en la que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos. El algoritmo fue desarrollado por Leo Breiman y Adele Cutler, siendo *Random Forest* la marca de fábrica. Su funcionamiento es basado en los árboles de decisión pero mejora la precisión en la clasificación mediante la incorporación de aleatoriedad en la construcción de cada clasificador individual. (Bishop, 2006; Walker, 2013)

Esta aleatorización puede introducirse en la partición del espacio (construcción del árbol), así como en la muestra de entrenamiento. El *Random Forest* comienza con la propia técnica del árbol de decisión, el cual, en cuanto al conjunto, corresponde a un aprendizaje. (Bishop, 2006; Walker, 2013)

Principales ventajas que ofrece el algoritmo *Random Forest* (Walker, 2013):

- Precisión.
- Funciona de manera eficiente con bases de datos de gran tamaño.
- Maneja miles de variables de entrada sin necesidad de borrado.
- Da estimaciones de qué variables son importantes en la clasificación.
- Genera una estimación objetiva interna de la generalización de errores.
- Proporciona métodos eficaces para estimar datos que faltan.
- Los "bosques" generados se pueden guardar para uso futuro en otros datos.
- Los prototipos se calculan de manera que proporcionan información acerca de la relación entre las variables y la clasificación.
- Calcula proximidades entre pares de casos que se pueden utilizar en la agrupación, la localización de los valores extremos, o (en escala) dan interesantes vistas de los datos.
- Ofrece un método experimental para la detección de interacciones de variables.

Aspectos técnicos del funcionamiento del algoritmo

El funcionamiento de *Random Forest* consiste en entrenar un conjunto de árboles de decisión por separado, por lo que el entrenamiento se puede hacer en paralelo. El algoritmo inyecta la aleatoriedad en el proceso de entrenamiento para que cada árbol de decisión sea un poco diferente. La combinación de las predicciones de cada árbol reduce la varianza de las predicciones, mejorando el rendimiento en los datos de prueba (Spark, 2017).

El proceso de entrenamiento consiste en (Spark, 2017):

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

- Realizar un submuestreo a partir de los datos originales para obtener conjuntos de entrenamientos diferentes por cada árbol de decisión que se generará con *Random Forest*.
- Considerar cada conjunto aleatorio de características diferentes para dividir en cada nodo del árbol.

Además de estas aleatorizaciones, el entrenamiento del árbol de decisión se hace de la misma manera que para los árboles de decisión individuales. Se debe tener en cuenta que para el entrenamiento de cada árbol de decisión se pueden aplicar las siguientes métricas:

- **Node impurity** (Impureza de un nodo). Es una medida de la homogeneidad de las etiquetas en el nodo. Esta se puede calcular mediante la Impureza de Gini o aplicando entropía (Spark, 2017).
- **Information Gain** (Ganancia de información). La ganancia de información es la diferencia entre la impureza del nodo padre y la suma ponderada de las dos impurezas del nodo hijo (Spark, 2017).

Tabla 1. Cálculo de impureza de un nodo (Pentreath, 2015; Spark, 2017).

Impuresa	Tipo de tarea	Fórmula	Descripción
Impureza de Gini	Clasificación	$\sum_{i=1}^{C} f_i(1-f_i)$	*fi* es la frecuencia de la etiqueta *i* en un nodo y C es el número de etiquetas únicas.
Entropía	Clasificación	$\sum_{i=1}^{C} -f_i \log(f_i)$	*fi* es la frecuencia de la etiqueta *i* en un nodo y C es el número de etiquetas únicas.

Cabe destacar que para poder resolver tareas de regresión se debe aplicar como criterio de impureza: la varianza.

A partir de lo anterior, la ganancia de información se calcula como sigue (Spark, 2017):

$$IG(D,S)=Impurity(D)-(Nleft/N)Impurity(Dleft)-(Nright/N)Impurity(Dright)$$

Donde se calcula la ganancia de información para una división de S particiones del conjunto de datos D de tamaño N, en dos conjuntos de datos Dleft y Dright de tamaños Nleft y Nright, respectivamente.

Predicción y clasificación

Para hacer una predicción en una nueva instancia, un *Random Forest* debe agregar las predicciones de su conjunto de árboles de decisión. Esta agregación se hace de manera diferente para la clasificación y la regresión (Spark, 2017). Cabe destacar que dicho algoritmo puede ser aplicado para resolver ambos problemas, pero el desarrollo del trabajo se enfocará en la tarea de clasificación y dentro de esta la predicción.

En la tarea de clasificación, la predicción de cada árbol se cuenta como un voto para una clase. Se predice la etiqueta que es la clase que recibe la mayoría de los votos (Spark, 2017). Por tanto se escoge el resultado que corresponde a la mayoría de los resultados obtenidos por cada árbol de decisión.

Herramienta Apache Spark

Apache Spark es un *framework* de código abierto para *clusters* de computadoras, provee de APIs para clusterización y la aplicación de algoritmos de aprendizaje automático. Soporta procesamiento en paralelo y es tolerante a fallos. En comparación con *Apache Hadoop* este *framework* es 100 veces más rápido en memoria y 10 veces más rápido en cuanto acceso a disco. (Pentreath, 2015; Spark, 2017)

Características de *Apache Spark* (Pentreath, 2015; Spark, 2017)

- Es fácil de usar y se pueden aplicar los lenguajes: *Java*, *Scala, Phython* y R.
- Ofrece 80 operaciones de alto nivel que facilitan la construcción de programas paralelizados.
- Combina para el acceso a los conjunto de datos fuentes en SQL, streaming en tiempo real y analítica compleja.
- La herramienta puede ejecutarse dentro del propio entorno *Hadoop* y acceder a los HDFS (*Hadoop Distributed File Systems*).
- Posee una gran comunidad y abundante documentación para su uso.

Con la herramienta para la minería de datos se podrá acceder a la implementación que esta ofrece para el algoritmo *Random Forest*, disponible en su componente *Spark* MLlib. Con esta se procederá a crear una aplicación que permita

procesar un fichero que contiene datos recogidos del funcionamiento de la red en un determinado instante de tiempo. A partir de este, se realizará la predicción.

Resultados y discusión

Para el desarrollo de la propuesta de solución se instaló la herramienta *Apache Spark* 2.2.0 y *Apache Hadoop* 2.8.0. Utilizando la herramienta *Apache Maven* 3.3.9 se descargaron las dependencias pertenecientes al núcleo de *Apache Spark* y las bibliotecas de aprendizaje automático (*Spark* MLlib), bajo el entorno de desarrollo *Netbeans* 8.2 y usando la plataforma de *Java* 8 (JDK 8).

Las dependencias del archivo de configuración del proyecto *Maven* (pom.xml) donde se implementa el algoritmo son las siguientes:

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.2.0</version>
    <type>jar</type>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.11</artifactId>
    <version>2.2.1</version>
    <scope>compile</scope>
</dependency>
```

Figura 2. Dependencias necesarias para utilizar en desarrollo las API *Spark MLlib*

Posterior a esto se cargó un conjunto de datos de prueba para realizar el entrenamiento del algoritmo. Dicho conjunto de datos estaba conformado con valores que representaban cada tipo posible en las clases de clasificación que se podían obtener. Estos valores son datos representativos de las métricas más generales que se miden con programas que permiten la monitorización de servidores que ofrecen servicios por el protocolo HTTP.

A continuación se describen los atributos del conjunto de datos para el entrenamiento:

- Uso del CPU. Tanto por ciento de utilización del CPU.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

- Uso de RAM. Tanto por ciento de utilización de la RAM.
- Uso de disco duro. Tanto por ciento de utilización del disco duro. Cuando se realizan envíos de archivos de forma remota u operaciones sobre las bases de datos.
- Espacio utilizado en disco. Tanto por ciento del espacio utilizado en el disco duro.
- Cantidad de peticiones concurrentes. Número de transacciones realizadas en un mismo instante de tiempo.
- Cantidad de usuarios conectados. Cantidad de usuarios conectados en un mismo instante de tiempo.

A partir de estos atributos, las clases que fueron aplicadas en el proceso de predicción fueron binarias:

- Si el servicio se mantiene estable toma valor 1.
- Si el servicio falla toma valor 0.

Del propio conjunto de datos, se seleccionaron de forma aleatoria el 70 % de las instancias para el entrenamiento del algoritmo y el resto se aplicó para las pruebas al modelo obtenido. El modelo aplica el algoritmo *Random Forest* con tres árboles de decisión, utilizando el criterio de impureza de Gini y con una máxima profundidad de 5 niveles por cada árbol, como condición de parada.

Para la carga de un conjunto de datos almacenados en disco utilizado la API Spark MLlib, se realiza lo siguiente:

```
SparkConf sparkConf = new SparkConf().setAppName("JavaRandomForestForNetworking")
        .setMaster("local[2]").set("spark.executor.memory","1g");

JavaSparkContext jsc = new JavaSparkContext(sparkConf);
String datapath = "C:\\Users\\Archimaestre\\Desktop\\dataLog.txt";
JavaRDD<LabeledPoint> data = MLUtils.loadLibSVMFile(jsc.sc(), datapath).toJavaRDD();
```

Figura 3. Creación de un contexto de ejecución de *Spark* y carga del conjunto de datos almacenado en disco

Para elegir de forma aleatoria los datos de entrenamiento del conjunto y los datos de prueba se puede aplicar lo siguiente:

```
JavaRDD<LabeledPoint>[] splits = data.randomSplit(new double[]{0.7, 0.3});
JavaRDD<LabeledPoint> trainingData = splits[0];
JavaRDD<LabeledPoint> testData = splits[1];
```

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

Figura 4. Selección aleatoria de los datos de entrenamiento y prueba

Se puede observar que para este ejemplo, se aplican un 30 % del conjunto de datos para las pruebas y un 70% para el entrenamiento del clasificador.

Para el entrenamiento del clasificador con el resto de los datos del conjunto se especifican los parámetros necesarios del algoritmo y se procede a ejecutar el entrenamiento:

```
// Entrenamiento del modelo
Integer numClasses = 2;// Número de clases para la predicción
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
Integer numTrees = 3; // Se puede emplear un mayor número de árboles
String featureSubsetStrategy = "auto"; // El clasificador elige su propia estrategia
String impurity = "gini";// Se puede elegir otro criterio de impureza gini/entropy
Integer maxDepth = 5; //Condición de parada dada por la profundidad máxima del árbol
Integer maxBins = 32;
Integer seed = 12345;

RandomForestModel model = RandomForest.trainClassifier(trainingData, numClasses,
  categoricalFeaturesInfo, numTrees, featureSubsetStrategy,
  impurity, maxDepth, maxBins,seed);
```

Figura 5. Entrenamiento del algoritmo *Random Forest*

Se puede observar que se especifican dos clases para la predicción ya que se desea conocer si la red falla o se mantiene estable. Se aplica un algoritmo donde se generan tres árboles de decisión y se usa el criterio de Gini como impureza de un nodo.

A continuación se muestra el monitoreo que provee la herramienta *Apache Spark*, donde se representa en un diagrama de tiempo la ejecución del entrenamiento del clasificador (*Random Forest*) y el tiempo que demoró dicha tarea, este rendimiento depende en gran medida de la potencia del servidor donde se realiza el procesamiento de los datos:

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No. 8, Mes Agosto, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 27-41
http://seriecientifica.uci.cu

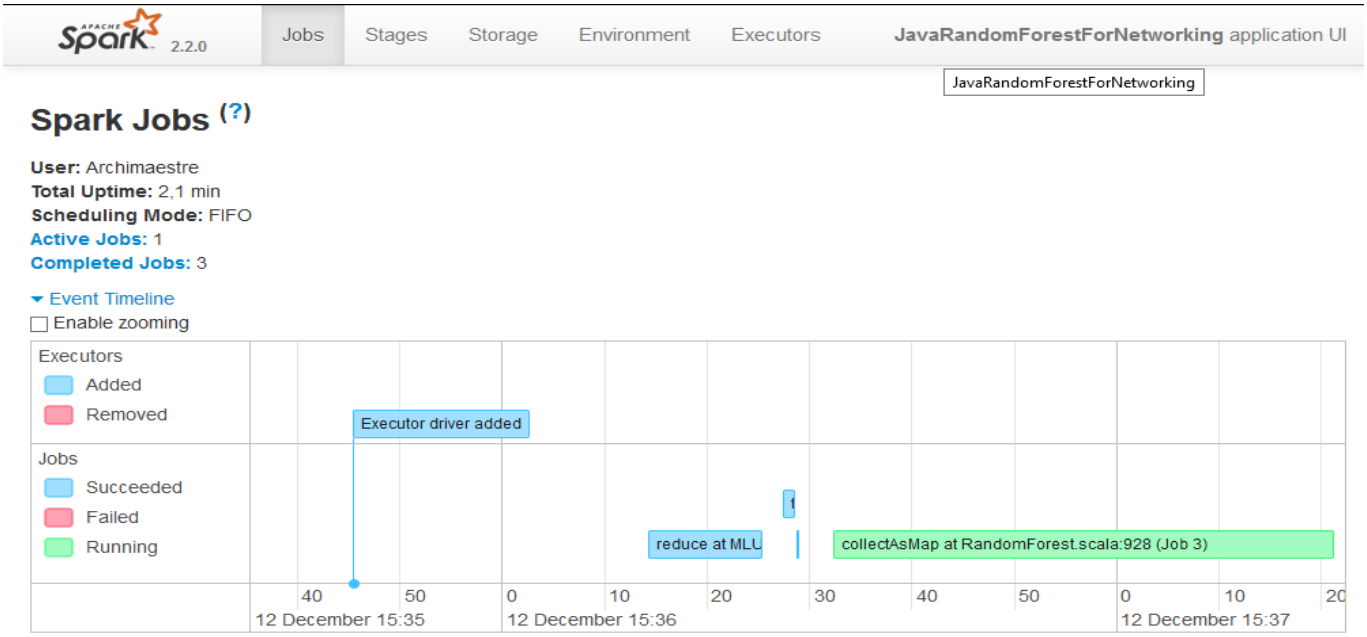

Figura 6. Diagrama de tiempo que representa la ejecución del entrenamiento del clasificador

Al ejecutar el entrenamiento, se serializa un modelo en disco que representa el bosque aleatorio ya entrenado. Este se puede cargar posteriormente para ser aplicado en el proceso de predicción. Donde dada una instancia que posee un grupo de parámetros que representan el funcionamiento del servidor, se puede predecir si este fallará o no.

Se puede mejorar el resultado de la clasificación aumentando el tamaño del conjunto de entrenamiento, para ello se debe hacer un estudio profundo de los *logs* registrados por los programas de monitoreo para conocer, con mayor detalle, durante qué eventos se manifestaron fallos en la disponibilidad de los servicios HTTP en la red. Se puede mejorar además la predicción dada por el clasificador, si se incrementa el número de árboles o se mapean los atributos de cada registro en el conjunto de entrenamiento para designar categorías a cada atributo, esto a su vez mejora el rendimiento del clasificador.

Después de ejecutar el algoritmo en cuestión con un total de 87 instancias, se obtiene como resultado la siguiente matriz de confusión:

Tabla 2 Matriz de confusión

	Ocurre fallo (Sí)	**No ocurre fallo (No)**
Ocurre fallo (Sí)	9	0
No ocurre fallo (No)	1	16

A partir de la matriz de confusión, se puede observar que de las instancias utilizadas en la prueba fueron clasificadas correctamente 9 en la clase de decisión **Sí** y 16 en la clase de decisión **No**. Clasificando así de manera incorrecta solo una de las instancias. Con esto se obtiene un modelo entrenado con una precisión de 96.15 %.

Conclusiones

Al término de este trabajo se arriba a las siguientes conclusiones:

- El estudio de las diferentes herramientas y tecnologías para la ejecución de tareas predictivas en el contexto de datos masivos permitió elegir las herramientas más adecuadas para este proceso que facilitarán la obtención de un clasificador para la predicción de fallas en la red. Para ello se hizo uso de la herramienta *Apache Spark*.

- El uso del algoritmo *Random Forest* para la solución de tareas predictivas permitirá predecir si la red fallará en un determinado instante de tiempo y permitirá a los administradores de red acometer acciones previas para evitar dicho suceso.

- Se puede mejorar el resultado del clasificador si se emplea un conjunto de entrenamiento mejor analizado en correspondencia con las características de la red y un estudio estadístico realizado sobre los eventos que ocurrieron durante el monitoreo del servidor o los servidores donde se produjeron los fallos. Con esto se pueden adaptar los parámetros del algoritmo en función de su rendimiento y la necesidad de la red.

Referencias

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Cambridge, UK: Springer Science+Business Media, LLC.

Elibeth Eduardo, G. (2017). El uso del big data para predecir el estado de las redes. Retrieved September 10, 2017, from www.cioal.com/2017/10/26/el-uso-del-big-data-para-predecir-el-estado-de-la-red/.

Esteso, M. P. (2015). *Análisis de arquitecturas de procesado de Streaming Big data.* Madrid.

Frydenberg, M. (2015). Introducing Big Data Concepts in an Introductory Technology Course, *13*.

Hussain, S., Akif, M., Tamayo, J. D., & Safdar, A. (2018). Big Data and Learning Analytics Model. *International Journal of Computer Sciences and Engineering*, *6*(7).

Landset, S., Khoshgoftaar, T. M., Richter, A. N., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, *2*(1), 1–36. https://doi.org/10.1186/s40537-015-0032-1

Manyika, J., & Chui, Michael and Brown, B. (2011). Big data: The next frontier for innovation, competition, and productivity. Retrieved January 20, 2005, from http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation.

Pentreath, N. (2015). *Machine Learning with Spark*. Birmingham, UK: Packt Publishing.

Shiomoto, K. (2013). Applications of Big Data Analytics Technologies for Traffic and Network Management Data, *11*.

Spark, A. (2017). Decision Trees - RDD-based API. Retrieved December 13, 2017, from https://spark.apache.org/docs/2.1.0/mllib-decision-tree.html#classification

Walker, M. (2013). Random Forests Algorithm. Retrieved January 1, 2018, from https://www.datasciencecentral.com/profiles/blogs/random-forests-algorithm.