IMPLEMENTACIÓN DE UN ALGORITMO DE APRENDIZAJE AUTOMÁTICO EN APACHE SPARK

IMPLEMENTATION OF A MACHINE LEARNING ALGORITHM IN APACHE SPARK

Ricardo Sánchez Alba[1]

[1]Universidad Central "Marta Abreu" de Las Villas, Cuba, risanchez@uclv.cu

RESUMEN: *El análisis de grandes cantidades de datos, así como la extracción de conocimiento útil de estos constituye en la actualidad un reto ya que cada día crecen velozmente los volúmenes de información generada y se necesitan programas capaces de realizar esta tarea en poco tiempo.*

Durante varios años frameworks de código abierto han sido utilizados para la aplicación de técnicas de aprendizaje automático en pequeños volúmenes de datos, pero la necesidad creciente de la industria ha dado como consecuencia una evolución en el área del cómputo distribuido, surgiendo así herramientas como Apache Hadoop y Apache Spark siendo éste último entre 10 y 100 veces más rápido que su antecesor.

En este trabajo se propone un procedimiento general para la inclusión de nuevos algoritmos de aprendizaje automático en el framework Apache Spark y se implementa un algoritmo de regresión lineal con el fin de validar la metodología propuesta. Se realizaron una serie de experimentos al software implementado que permitieron valorar las ventajas del framework Apache Spark para reducir significativamente los tiempos de ejecución cuando este tipo de algoritmo se somete al procesamiento de cantidades masivas de datos.

Palabras Clave: apache spark, aprendizaje automático, algoritmo

ABSTRACT: *The analysis of large amounts of data, as well as the extraction of useful knowledge of these, is now a challenge as each day the volumes of information generated grow rapidly and programs are needed that can perform this task in a short time.*

For several years, open source frameworks have been used for the application of automated learning techniques in small volumes of data, but the growing need of the industry has resulted in an evolution in the area of distributed computing, resulting in tools such as Apache Hadoop and Apache Spark The latter being between 10 and 100 times faster than its predecessor.

In this paper we propose a general procedure for the inclusion of new algorithms of automatic learning in the Apache Spark framework and a linear regression algorithm is implemented in order to validate the proposed methodology.

A series of experiments were performed on the implemented software that allowed to evaluate the advantages of the Apache Spark framework to significantly reduce execution times when this type of algorithm is submitted to the processing of massive amounts of data.

Keywords: apache spark, machine learning, algorithm

1. INTRODUCCIÓN

Cada día la cantidad de información de cualquier tipo aumenta aceleradamente y así mismo se hace necesario no solo almacenar esta información sino poder procesarla eficiente y eficazmente para extraer conocimiento útil de esta. [1], [7]

Construir modelos para detectar fraudes de tarjetas de crédito utilizando miles de características y billones de transacciones; recomendar inteligente-mente millones de productos a millones de usuarios; estimar riesgos financieros utilizando simulaciones de portafolios incluyendo millones de instrumentos y manipular datos del genoma humano para detectar asociaciones genéticas con enfermedades eran tareas muy difíciles o imposibles de realizar hace 5 o 10 años atrás debido a la alta complejidad computacional y las cantidades de datos asociados. El modelo de programación funcional MapReduce ha sido pionero en el tratamiento de estas grandes cantidades de datos dada su facilidad para ser implementado mediante sistemas de cómputo paralelos. Sistemas distribuidos como Apache Hadoop han encontrado así su camino y han tenido una amplia aplicación en múltiples empresas. [2]

Durante un largo periodo de tiempo, frameworks de código abierto como R, PyData y Octave han sido utilizados para realizar rápidos análisis y construcciones de modelos viables sobre pequeños *datasets*. Pero un correcto aprovechamiento de estos frameworks sería extenderlos para ejecutarlos en múltiples computadoras, mantener sus modelos de programación y reescribir sus interioridades para lidiar eficientemente con características distribuidas. Sin embargo, muchos elementos que se asumían por estar en sistemas con un solo nodo requieren ser repensados para la computación distribuida. Por ejemplo, porque los datos deben estar particionados a lo largo de múltiples nodos en un clúster de computadoras, algoritmos que tenían amplias dependencias de datos sufrirían el hecho de que las tasas de transferencia de la red son más lentas que el acceso a memoria. Así como que a medida que aumentan la cantidad de computadoras trabajando en un problema, aumenta la probabilidad de fallo. [2]

Como parte de las investigaciones que se desarrollan en el laboratorio de Inteligencia Artificial del Centro de Investigaciones de Informática (CEI) se desea poder utilizar el ambiente de computación distribuida Apache Spark para disminuir significativamente los tiempos de ejecución de algoritmos de Aprendizaje Automatizado cuando

estos se someten a procesamientos de cantidades masivas de datos. Apache Spark cuenta con un módulo en desarrollo concebido para el trabajo con algoritmos de esta naturaleza, dentro de este, se tiene un conjunto de algoritmos de aprendizaje automático y utilidades afines que se brindan como un paquete pre-elaborado. Es de gran utilidad conocer el procedimiento a seguir cuando un investigador desea incluir un algoritmo propio y que este aproveche las
ventajas de la arquitectura subyacente definida por los desarrolladores de Apache Spark.

De esta forma se desea desarrollar un algoritmo de aprendizaje automático en Apache Spark mediante la definición de un conjunto de pasos generales para ajustar estos procedimientos al modelo de programación de este framework, lo cual constituye el objetivo general de este trabajo.

Para dar cumplimiento al anterior objetivo general se proponen los **objetivos específicos**:

- Determinar las características de la arquitectura de Apache Spark que permiten la implementación de algoritmos de aprendizaje automático.
- Proponer un conjunto de pasos para la implementación de algoritmos de aprendizaje automático en Apache Spark.
- Implementar un algoritmo de aprendizaje automático en Apache Spark utilizando la arquitectura del modelo del framework y los pasos propuestos en este trabajo.
- Evaluar el desempeño del algoritmo bajo el ambiente distribuido de Spark mediante la realización de experimentos.

2. CONTENIDO

Spark contiene dos bibliotecas de aprendizaje automático, Spark MLlib y Spark ML con APIs marcadamente diferentes, en conjunto incluyen utilidades para clasificación, regresión, agrupamiento, filtrado colaborativo, reducción de dimensionalidad, así como las primitivas de optimización subyacentes. Estas bibliotecas heredan consideraciones de rendimiento de la API basada en RDD y la basada en Datasets. MLlib es la primera de las dos, pero se encuentra en un estado de mantenimiento y corrección de errores solamente. Spark ML es la API nueva, donde el desarrollo activo está tomando lugar. [3], [6], [8]

En orden de agregar nuevas funcionalidades a Spark ML se tienen dos opciones, la primera es implementar los algoritmos utilizando transformaciones a los RDD siguiendo las convenciones de Spark MLlib y seguir a partir de ahí, para Spark ML este acercamiento es válido también, pero se pierden características integradas muy útiles, incluyendo la posibilidad de ejecutar meta-algoritmos como búsqueda de parámetros con cross-validación, la segunda vía es extender el modelo de Spark ML Pipeline.

Para agregar un nuevo algoritmo a una pipeline se necesita implementar ya sea Estimator o Transformer las cuales implementan a su vez la interfaz PipelineStage. Para los algoritmos que no necesitan entrenamiento, se puede implementar la interfaz Transformer, y para algoritmos con entrenamiento la interfaz Estimator, ambas en org.apache.spark.ml. [3], [6]

2.1 Modelo de herencia de clases para la API de Spark ML

La API de Spark puede ser de cierta forma vista desde dos perspectivas la API pública y la API privada. Estas están delimitadas por el paquete declarado en los archivos de código donde están implementadas las clases.

Las clases recomendadas para la extensión de Spark ML (Estimator y Transformer) pueden ser heredadas de manera natural desde un proyecto de usuario siempre que se tengan las respectivas dependencias a la biblioteca estándar de Spark.

Por otra parte, los desarrolladores de Spark implementan las nuevas funcionalidades dentro de los paquetes internos de Spark, en este caso org.apache.spark.ml que al mismo tiempo se encuentran en un nivel inferior en la jerarquía de clases respecto a las clases Estimator y Transformer, en lo que se refiere a la extensión del modelo de Pipelines.

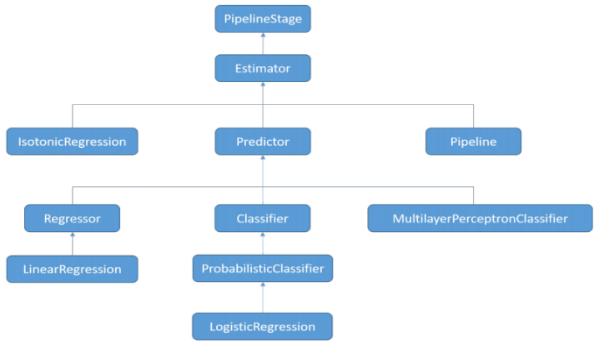

Figura. 1: Herencia asociada a Estimator.

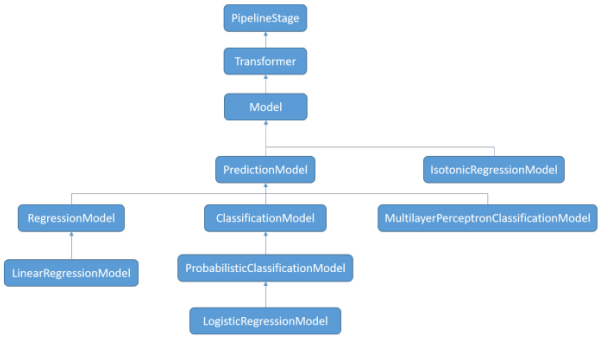

Figura. 2: Herencia asociada a Transformer.

2.2 Pasos propuestos para el ajuste de un procedimiento de Aprendizaje Automático al modelo de Pipeline de Spark ML

Dada la homogeneidad estructural presentada por la API de Spark ML, las potencialidades de desempeño de las estructuras que este utiliza y la creciente comunidad de desarrolladores que se encuentran en una intensa labor de extensión de esta API se decidió utilizarla para el caso de estudio de la inclusión de un algoritmo de aprendizaje automático que funcionara como vehículo para ejemplificar el proceso de implementación. Para ello se confeccionaron una serie de pasos generales a seguir para ajustar un algoritmo que se desee implementar utilizando los beneficios de esta API. Básicamente estos pasos son:

- Definir matemáticamente el algoritmo a implementar.
- Redefinir las estructuras y subrutinas asociadas para condicionarlas al modelado distribuido necesario para la paralelización. Esto es, condicionar la futura implementación utilizando DataFrame y Dataset de Spark SQL y el paradigma de programación funcional.
- Determinar si el algoritmo necesita entrenamiento o si es una transformación directa de los datos de entrada.
- En caso de necesitarse entrenamiento, implementar la interfaz Estimator de Spark ML en la cual el código del entrenamiento debe estar implementado en el método fit()
- Implementar la interfaz Transformer en cuyo método transform() se debe situar el código de la transformación de los datos.
- En caso de requerirse la integración del algoritmo como parte de una Pipeline se debe tener en cuenta la implementación de los métodos transformSchema() de ambas interfaces donde se explicita la transformación en la estructura de los datos en la entrada y salida de esa etapa de la Pipeline

2.3 Descripción matemática del algoritmo implementado

2.3.1 Ecuación normal

Se desea predecir el valor de una variable mediante su aproximación a la función lineal:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x \qquad (1)$$

siendo n el número de variables de entrada. Un método utilizado para escoger los parámetros θ es definir una función que estime qué tan cerca están

los valores de $h(x^{(i)})$ respecto a los valores correspondientes de $y(i)$. Sea la función de costo:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \qquad (2)$$

Esta función es comúnmente denominada función de costo de mínimos cuadrados. Sea X la matriz que contiene las instancias de entrenamiento y y el vector de valores de la función objetivo asociados a estas instancias. Una posible solución al problema de regresión lineal es utilizar la minimización explícita mediante el uso de la ecuación normal:

$$\theta = (X^T X)^{-1} X^T y \qquad (3)$$

lo que es equivalente a la resolución del sistema lineal:

$$X^T X\theta = X^T y \qquad (4)$$

2.3.2 Redefinición de la multiplicación de matrices para permitir la escalabilidad del cómputo paralelo

La resolución de la ecuación normal como método para calcular los valores de los parámetros θ incluye el cómputo de la multiplicación de matrices. Este procedimiento es implementado por varias bibliotecas locales de álgebra lineal dada la frecuencia de su utilización. Para un correcto aprovechamiento de las capacidades del cómputo distribuido en este frecuente procedimiento algebraico se hace necesario plantear la multiplicación de matrices de manera tal que las matrices puedan ser fraccionadas a lo largo de un clúster de computadoras, así como poder computar porciones de la matriz resultado de manera distribuida.

Si se utiliza la definición de producto tensorial $\otimes$ ésta multiplicación puede ser escrita como:

$$(A \cdot B) = \sum_i (\vec{a_i} \otimes \vec{b_i}) \qquad (5)$$

[4]

Esta formulación permite realizar un fraccionamiento de las matrices operando de manera que estas pueden ser distribuidas en un clúster de computadoras, pero acarrea la inconveniente de generar los pares $(\vec{a_i}, \vec{b_i})$ para aplicar el operador $\otimes$ mediante la operación map y reducir los accesos a los datos por lo que el almacenamiento necesario para una correcta paralelización es mucho mayor.

En el caso de la multiplicación $X^T X$ presente en la ecuación normal se pueden aprovechar varias bondades. Dado que la multiplicación es de la matriz por ella misma, no es necesario generar los pares $(\vec{a_i}, \vec{a_i})$ por lo que es menos costoso en cuestiones de espacio de almacenamiento. La otra particularidad de esta multiplicación es que dado que $(\vec{a_i} \otimes \vec{a_i}) = \vec{a_i}^T \cdot \vec{a_i}$ es una matriz simétrica, no es necesario calcular todos los elementos, sino solo los elementos de la triangular inferior y se disminuyen significativamente los cálculos necesarios.

2.4 Características y diseño general de la implementación

La implementación del algoritmo se realizó siguiendo el modelo de programación de la API de Spark ML basada en Pipelines. El lenguaje de programación escogido fue Scala versión 2.11.8 incluida dentro de la biblioteca de Spark versión 2.0.1. Para las tareas locales de álgebra lineal se utilizó la biblioteca LAPACK incluida en esa versión de Spark. El IDE utilizado para la implementación fue IntelliJ IDEA versión 2016.3. La versión de Java utilizada por Scala en esta implementación fue la 8 actualización 91.

2.4.1 Principales clases y funciones implementadas

Clase Instance:

Representa una instancia de entrenamiento que contiene un atributo label que representa la clase de esa instancia y un atributo features que almacena el vector de características de la instancia.

Clase CustomLinearRegression:

Esta case hereda de la clase Estimator de Spark ML. Es la encargada de construir un modelo a partir de las instancias de entrenamiento. En este caso el modelo construido es un modelo de regresión lineal utilizando la ecuación normal. El principal método implementado en esta clase es fit el cual es el encargado de construir el modelo aprendido a partir de las instancias, recibe como parámetro un Dataset de Spark SQL con los datos de entrenamiento.

Pseudocódigo de la función fit:

Entrada: dataset. Objeto que contiene las instancias de entrenamiento.

Salida: Objeto de CustomLinearRegressionModel que contiene el vector θ de coeficientes para evaluar la función lineal.

1. trainInstances ← dataset.select("features", "label")
2. l2RegularizationParam ← 0.1
3. Calcular ambos miembros de la ecuación normal $X^T X\theta = X^T y$.
4. Aplicar regularización L2 sobre el miembro $X^T X\theta$.
5. Resolver el sistema lineal $X^T X\theta = X^T y$ sobre la variable θ.
6. Devolver una nueva instancia de CustomLinearRegressionModel con el valor del vector θ.

Clase CustomLinearRegressionModel:

Esta clase hereda de la clase Model de Spark ML. Es la encargada de representar el modelo aprendido, así como de contener las funcionalidades para transformar los datos de entrada y predecir características para nuevas instancias. Concretamente esta clase contiene un método transform que recibe un Dataset de Spark SQL como datos de entrada, en este caso, instancias para las cuales se desea predecir el valor de la clase mediante la evaluación de una función lineal que utiliza los valores de las características y devuelve un DataFrame de Spark SQL que contiene los valores arrojados por la predicción.

Pseudocódigo de la función transform:

Entrada: Dataset que contiene las instancias para las cuales se desea predecir la variable continua y de la clase a partir del vector de características de cada una y el vector de coeficientes θ aprendido.

Salida: DataFrame que contiene las instancias de entrada con la columna adicional de valores arrojados por la evaluación de la función lineal para cada una de esas instancias.

1. instances ← dataset.select("features", "label")
7. Evaluar la función lineal mediante el cálculo de $\theta^T \vec{x}$ para cada instancia.
8. Agregar la columna de valores arrojados por la predicción al dataset de entrada.
9. Devolver el nuevo dataset en forma de DataFrame

También se implementaron varias funciones necesarias para las tareas de cálculo algebraico y estadísticas sobre los resultados.

Función rmse:

Esta función es la encargada de calcular el error cuadrático medio de la estimación realizada por la función lineal respecto a los valores reales de la variable y para instancias de las cuales se conoce este dato. De manera que esta medida de error

puede ser utilizada para valorar qué tan acertada está la estimación.

Función outerVecProduct(v: Vector)

Esta función es la encargada de calcular el producto externo de un vector por sí mismo, concretamente la operación $\vec{a} \otimes \vec{a}$. La implementación de este método solo calcula los elementos de la matriz triangular asociada al resultado, dado que es una matriz simétrica y devuelve esta matriz en representación vectorial por filas.

Es importante resaltar que al realizarse la implementación con Scala, que es un lenguaje de programación funcional las estructuras para la representación de colecciones como vectores y matrices, así como de las estructuras distribuidas de Spark son inmutables e implementan eficientemente el paradigma map-reduce. En el caso de la implementación de las estructuras de Spark este modelo de programación permite la paralelización implícita por lo que las operaciones de la naturaleza map están implementadas para ser escalables por definición.

Como consecuencia de haberse implementado este algoritmo siguiendo el modelo de programación de Spark ML este está condicionado para ser integrado como parte de una Pipeline y ser utilizado como parte de una cadena de procesamientos sucesivos de datos.

2.5 Diagramas de clases asociados a CustomLinearRegression y CustomLinearRegressionModel

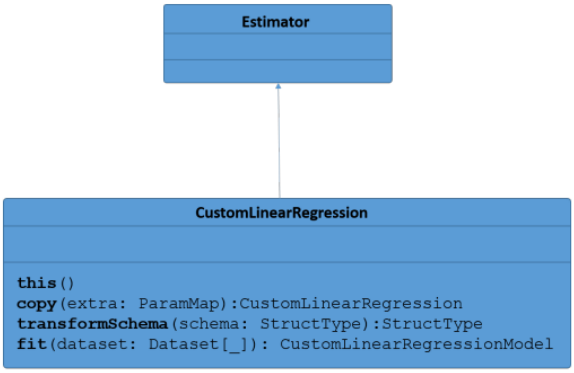

Figura. 2: Diagrama asociado a CustomLinearRegression

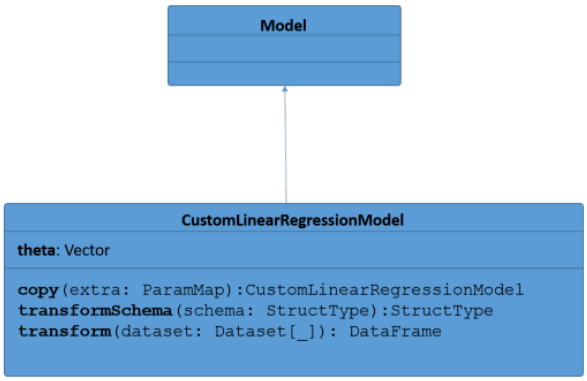

Figura. 3: Diagrama asociado a CustomLinearRegressionModel

2.6 Pruebas realizadas

2.6.1 Características del dataset

[5]

- Tamaño en disco: **553 MB.**
- Número de instancias: **463715** / **51630** (test)
- Número de características: **90**
- Localización: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html

2.6.2 Características del clúster utilizado:

El clúster utilizado para las pruebas presenta las siguientes prestaciones:

- Cantidad de Nodos: 10

- Versión de Spark: 2.1.0
- Manejador de clúster: YARN

Configuración de YARN:

Nodo:

Memoria reservada para todos los contenedores de YARN en un nodo: 12GB

CPU:

Porciento de CPU físico para todos los contenedores en un nodo: 80 %

Número de núcleos virtuales: 3

Contenedor:

Tamaño de memoria los contenedores: Mínimo 1024 MB; Máximo 12 GB

Tamaño de los contenedores (CPU): Mínimo 1; Máximo 3 (Núcleos virtuales)

Tamaño de montículo de las JVM: 3064 MB

2.6.3 Resultado de las pruebas de escalabilidad

Las pruebas se realizaron incrementando el número de ejecutores desde 1 hasta 10, utilizando los siguientes parámetros en común:

- –-master yarn
- –-driver-memory 4098m
- –-executor-memory 4098m
- –-executor-cores 3

Los tiempos de ejecución registrados por el clúster de Spark se comportaron de la siguiente manera:

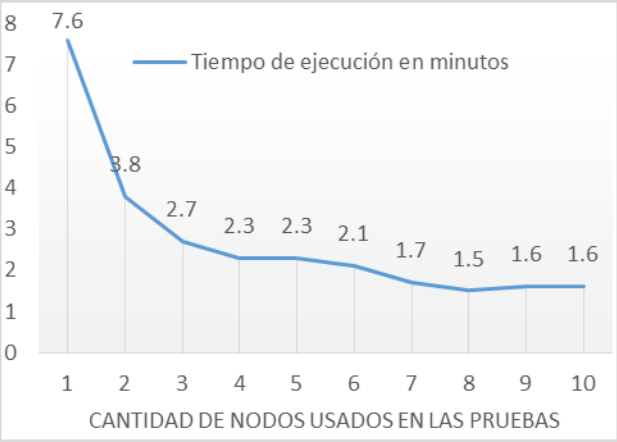

Figura. 4: Escalabilidad del algoritmo personalizado

3. CONCLUSIONES

El estudio de la arquitectura de Spark permitió reconocer las prácticas seguidas por sus desarrolladores en el proceso de construcción, así como las principales ventajas proporcionadas por la API, específicamente en los módulos relacionados con aprendizaje automático, profundizándose el conocimiento sobre este framework.

Se desarrolló un procedimiento general para guiar el proceso de implementación de nuevos algoritmos de aprendizaje automático sobre Spark ML, utilizando su modelo de programación, lo cual permite extender las funcionalidades brindadas por esta herramienta.

Se llevó a cabo la implementación de un algoritmo de regresión lineal, utilizando la ecuación normal, siguiendo los pasos propuestos con el fin de su validación.

La etapa de pruebas al algoritmo implementado permitió observar varios resultados favorables, entre ellos:

- Las capacidades del software de Apache Spark en el manejo de grandes volúmenes de datos en poco tiempo.
- El aprovechamiento por parte del algoritmo implementado de las ventajas de la plataforma subyacente.
- La escalabilidad del software implementado mediante los pasos propuestos y el modelo de programación de la API de Apache Spark, observándose una reducción de los tiempos de ejecución a medida que aumentan las prestaciones del hardware disponible en un clúster de computadoras.

4. REFERENCIAS BIBLIOGRÁFICAS

1. **Hastie, T.; R. Tibshirani and J. Friedman:** "The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Ed Springer, 2001.
2. **Ryza, S; U. Laserson; S. Owen and J.**

Wills: "Advanced Analytics with Spark", Ed O'REILLY, 2015.

3. **Karau, H:** "Extend Spark ML for your own model/transformer types", https://www.oreilly.com/learning/extend-spark-ml-for-your-own-modeltransformer-types, 2017.
4. **Golub, Gene H and Charles F. van Loan:** "Matrix computations", Ed Johns Hopkins University Press, 1996.
5. **Csie.ntu.edu.tw:** "Libsvm Official Website", http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ref.html#SK09a, 2017.
6. **Karau, H and R. Warren:** "High Performance Spark", Ed O'REILLY, 2017.
7. **Pentreath, N:** "Machine Learning with Spark", Ed PACKT, 2015.
8. **Apache Software Foundation:** "Apache Spark Official Website", https://spark.apache.org/docs/2.1.0/ml-guide.html, 2017.

5. SÍNTESIS CURRICULARES DE LOS AUTORES

Ricardo Sánchez Alba es Licenciado en Ciencia de la Computación por la Universidad Central "Marta Abreu" de Las Villas (UCLV), actualmente se encuentra en el período de adiestramiento. En el período 2013-2014 perteneció al Laboratorio de Programación e Ingeniería de Software donde investigó y desarrolló soluciones de software para varios problemas de bioinformática. En el curso 2014-2015 se desarrolló como alumno ayudante prestando servicio en la asignatura Lógica. En ese mismo período de tiempo perteneció al Laboratorio de Base de Datos, donde desarrolló un software para la tarea de la automatización del proceso de población de base de datos. Desde el 2015 hasta la actualidad trabaja como programador en el grupo de desarrollo de aplicaciones móviles KaleydoBit de la Facultad de Matemática Física y Computación. Desde el 2015 es miembro del Laboratorio de Inteligencia Artificial del Centro de Investigaciones en Informática donde investiga en las áreas del Aprendizaje Automático, Cómputo Distribuido y Ciencia de Datos.