JCLAL 2.0: mejoras y nuevas funcionalidades en la herramienta Java de código abierto para el aprendizaje activo

Eduardo Pérez
Instituto Maimónides de Investigación Biomédica de Córdoba
Email: eduardo.perez@imibic.org

Luis D. González
Dpto. de Informática
Universidad de Holguín
Email: ldgonzalezo@uho.edu.cu

Luis M. Sánchez
Dpto. de Informática
Universidad de Holguín
Email: lsanchezv@uho.edu.cu

Oscar Reyes
Dpto. Informática y Análisis Numérico
Universidad de Córdoba
Email: ogreyes@uco.es

Sebastián Ventura
Dpto. Informática y Análisis Numérico
Universidad de Córdoba
Email: sventura@uco.es

***Resumen*—El aprendizaje activo tiene como objetivo la construcción de mejores clasificadores mediante el etiquetado iterativo de conjuntos de ejemplos no etiquetados. Este paradigma de aprendizaje resulta de suma importancia en la actualidad, debido principalmente al alto coste que tiene el proceso de clasificación de grandes volúmenes de datos. JCLAL es una librería Java de código abierto para la investigación en el área de aprendizaje activo, que fue creada con el objetivo de facilitar el desarrollo de algoritmos y la ejecución de experimentos en esta área de estudio. En este trabajo, se presentan las mejoras y nuevas funcionalidades que incorpora JCLAL en su versión 2.0. Esta nueva versión tiene varias características que hacen de JCLAL una herramienta más potente para la investigación en el área del aprendizaje activo. Se introduce un entorno visual que facilita aún más el uso de JCLAL por usuarios no expertos en el dominio. Además, esta nueva versión aprovecha los beneficios de la computación paralela y distribuida, posibilitando que JCLAL sea utilizado eficientemente en entornos actuales donde se manejan grandes conjuntos de datos.**

I. Introducción

El aprendizaje supervisado tiene como objetivo la construcción de modelos a partir de datos que han sido etiquetados previamente. Sin embargo, uno de los principales problemas de este paradigma de aprendizaje es que generalmente se requiere de una considerable cantidad de datos etiquetados para lograr construir modelos con una alta precisión. El etiquetado de datos no es una tarea sencilla de hacer, ya que requiere comúnmente un esfuerzo considerable por parte de los expertos que etiquetan los datos. En consecuencia, hoy en día se pueden encontrar muchas colecciones de datos que tienen un número reducido de ejemplos etiquetados en conjunto con una cantidad enorme de ejemplos que quedaron sin etiquetar [1]. En este tipo de escenario las técnicas supervisadas son insuficientes al no ser capaces de explotar la información útil que se encuentra implícita en los datos no etiquetados. Ante este tipo de problema, surge el aprendizaje semi-supervisado y el aprendizaje activo (AL), que tienen como principal objetivo la construcción de modelos precisos usando tanto los datos etiquetados como los no etiquetados.

Los métodos de AL emplean como principal hipótesis que si el algoritmo tiene la oportunidad de elegir los datos para el proceso de aprendizaje, entonces se podrán construir mejores modelos con un menor costo. Las técnicas de AL tratan de superar los obstáculos del etiquetado de datos mediante consultas donde se intenta descubrir aquellos ejemplos no etiquetados que pueden aportar información útil en la construcción de un modelo. Los métodos de AL son iterativos, donde en cada paso los ejemplos no etiquetados más significativos son presentados a un oráculo (por ejemplo, un experto), el cual los etiqueta y posteriormente estos ejemplos son incorporados al conjunto de entrenamiento a partir del cual se construye un nuevo modelo. De esta manera se intenta constuir un modelo más preciso usando la menor cantidad posible de información [2, 3].

En la actualidad, el rápido y adecuado desarrollo de un área de investigación está condicionado a contar con las herramientas computacionales que provean, al menos, las siguientes funcionalidades: (I) la disponibilidad de un API que permita reducir el tiempo en la implementación de nuevos algoritmos; (II) la facilidad de reproducción de experimentos; y (III) que tenga implementado un número considerable de métodos del estado del arte, de manera que le ahorre tiempo a los investigadores, así como disminuya la posibilidad de cometer errores en la implementación de dichos algoritmos. En este sentido, el AL ha estado bastante limitado, ya que las heramientas computacionales para la investigación en esta área son muy escasas, y la mayoría de las existentes no proveen de las funcionalidades mencionadas anteriormente. Entre las librerías que podemos encontrar en Internet destacan LibAct [4] desarollada en Python, y JCLAL [5] que está completamente desarrollada en el lenguaje Java.

Si comparamos estas dos librerías de clases, es de destacar que JCLAL contiene todos los elementos tratados en LibAct,

pero además incorpora un amplio conjunto de funcionalidades que permiten controlar todo el ciclo de AL a través de un correcto y flexible diseño de clases e interfaces. De esta manera, la implementación de nuevas estrategias de AL, así como la definición de nuevos escenarios de consulta, tipos de oráculos, condiciones de paradas, entre otras muchas funcionalidades, resulta un proceso bastante sencillo de hacer. Por otra parte, JCLAL cuenta con una detallada documentación y un conjunto amplio de ejemplos.

Como parte de la evolución a la cual está sujeto cualquier software, la librería JCLAL se ha seguido mejorando y desarrollando desde su publicación inicial. El objetivo del presente trabajo es presentar las nuevas características y funcionalidades que podemos encontrar en la nueva versión de este framework. Si se analiza la versión 1.0 de JCLAL, esta no disponía de una interfaz de usuario, obligando a los investigadores a usar el sistema de configuración de experimentos (basado en ficheros XML) o bien a codificar directamente los procedimientos usando su API. En este sentido, la nueva versión de JCLAL dispone de una interfaz de usuario que lo convierte en un producto más asequible y usable, lo que posibilita conocer sus potencialidades sin tener que entrar en el proceso de asimilación de su API. Por otra parte, la antigua versión de JCLAL está limitada a trabajar solamente con conjuntos de datos que pueden ser cargados en la memoria de estaciones de trabajo individuales, lo cual complicaba el uso de este framework en escenarios con enormes volúmenes de datos (problemas *Big Data*). En este trabajo, se explica además como la nueva versión de JCLAL supera esta última limitación gracias al desarrollo de un módulo que se integra con el conocido framework para procesamiento de datos masivos Apache Spark [6].

El resto de este trabajo se organiza de la manera siguiente. En la Sección II se detallan los cambios y nuevas funcionalidades en el API de JCLAL, así como se describe la nueva manera en la que se gestiona la modularidad del framework. En la sección III se introduce el entorno visual de JCLAL, detallando las funcionalidades del mismo. La integración de la librería con el framework Spark se explica en la Sección IV. Finalmente, en la Sección V se presentan las conclusiones del presente trabajo.

II. Estructura de clases y nuevas funcionalidades en el API de JCLAL 2.0

Desde su primer lanzamiento JCLAL ha continuado mejorándose y desarrollándose como parte del proceso de actualización y mantenimiento de cualquier software. En su versión 2.0, JCLAL cuenta con una nueva estructura de clases y funcionalidades que lo convierten en un framework cada vez más cercano a los estándares que se esperan de este tipo de herramienta computacional.

Una de las nuevas funcionalidades es la posibilidad de aprovechar todas las capacidades de cómputo de las máquinas modernas. El AL es un proceso iterativo, donde en cada iteración comúnmente los pasos más costosos son la construcción del modelo, la evaluación del modelo y el cálculo de la importancia de cada instancia no etiquetada según la estrategia de consulta. JCLAL brinda la posibilidad de paralelizar el experimento en una estación de trabajo individual y también a través del framework Apache Spark. El investigador además tiene la posibilidad de elaborar sus propias estrategias de paralelización a través de la interfaz *IParallelContext*. Si el usuario desea hacer la paralelización de forma personalizada deberá registrar su clase en un archivo de configuración, como se muestra en el ejemplo siguiente:

```
<query-strategy type="...EntropySamplingQueryStrategy">
 <parallel-unlabeled type="...TestUnlabeledDataSinglePC"/>
 <wrapper-classifier type="...WekaClassifier">
  <parallel-test type="...WekaTestModelSinglePC"/>
  <parallel-build type="...WekaBuildClassifierSinglePC"/>
  <classifier type="weka.classifiers.functions.SMO"/>
 </wrapper-classifier>
</query-strategy>
```

Anteriormente, JCLAL estaba restringido al uso de conjuntos de datos que siguieran el formato establecido por el popular framework Weka [7]. En esta nueva versión, se modificó la jerarquía de clases e interfaces que están relacionadas con la manipulación de los conjuntos de datos. De esta manera, se logra una mayor flexibilidad e independencia en el trabajo con los conjuntos de datos, haciendo posible integrar a JCLAL con datos en formato de cualquier otro framework. Si un investigador quiere aplicar JCLAL en conjuntos de datos no soportados nativamente, solamente tendrá que implementar las interfaces *IInstance* e *IDataset*; además estas interfaces dan soporte a los tipos de instancias densas y dispersas.

Por otra parte, en JCLAL 2.0 se le ha dado mayor soporte a los clasificadores del framework MOA [8], lo cual permite utilizar algoritmos de aprendizaje incremental en el proceso de AL. De esta manera, el modelo no necesita ser entrenado desde cero en cada iteración de AL (a diferencia de la mayoría de los clasificadores de Weka), lo cual mejora significativamente el rendimiento en escenarios donde se disponen de grandes volúmenes de datos no etiquetados o con datos en *streamming*.

Por último, en esta nueva versión se ha añadido un paquete de clases que implementan varios test estadísticos no paramétricos. Dicho paquete está inspirado en el módulo de test estadísticos de la librería Keel [9]. Los test estadísticos son de suma importancia para el análisis de resultados experimentales, y JCLAL incluye varios test que permiten realizar comparaciones múltiples entre estrategias de AL.

II-A. Modularidad de JCLAL

Una de las nuevas mejoras es la modularidad del framework, lo cual facilita su portabilidad y distribución. La idea surgió al observar que JCLAL crecía en tamaño (un tamaño considerable al incluir el paquete de integración con Spark), lo cual hacía que el árbol de dependencia fuera bien extenso y además afectaba la portabilidad y usabilidad de JCLAL en aquellos usuarios que solamente querían usar las características básicas del framework.

En la nueva versión se optó por un diseño modular inspirado en el diseño de Apache Spark y Spring [10], separando las características del framework en los distintos módulos que componen el sistema. De esta manera, el proyecto fue

dividido en 3 módulos: **jclal-core** que tiene las funcionalidades básicas de JCLAL, el módulo **jclal-gui** que incluye la nueva interfaz visual y **jclal-spark** que comprende el procesamiento distribuido con Spark.

La modularidad de JCLAL se gestionó mediante Apache Maven [11], el cual es una herramienta popular para la gestión de proyectos software, que facilita la administración de dependencias y un amplio conjunto de estándares para el proceso de desarrollo del software. Si el usuario decide emplear Maven como administrador de dependencias, el *groupid* para JCLAL sería **net.sf.jclal**. El siguiente ejemplo muestra cómo instalar el núcleo de JCLAL (**jclal-core**) en una aplicación; de esta manera Maven gestiona las dependencias necesarias para usar solamente el módulo **jclal-core**.

```
<dependencies>
  <dependency>
    <groupId>net.sf.jclal</groupId>
    <artifactId>jclal-core</artifactId>
   <version>2.0</version>
  </dependency>
</dependencies>
```

III. Entorno visual para JCLAL

Es común que los framework de aprendizaje automático dispongan de una interfaz gráfica de usuario (GUI, por sus siglas en inglés) para facilitar la configuración, ejecución y análisis de resultados de los experimentos, reduciendo así el tiempo y trabajo necesario de los investigadores. Tal es el caso de Weka, VisualJCLEL [12], Eva2 [13], OTA [14] y HeuristicLab [15] que permiten a los investigadores sin un amplio conocimiento de su funcionamiento interno, o incluso sin habilidades de programación, utilizar estos framework de forma intuitiva.

VisualJCLAL es un módulo GUI que permite trabajar con el framework de forma intuitiva y flexible, acercando JCLAL a los usuarios no expertos en el dominio. Este entorno visual está inspirado en la arquitectura de Weka y VisualJCLEC. VisualJCLAL permite aprovechar la mayoría de las funcionalidades que brinda su framework base, y hasta el momento brinda la opción de utilizar estrategias de AL diseñadas para crear modelos predictivos sobre datos clasificados con una sola etiqueta (*single-label learning*) y datos clasificados con múltiples etiquetas (*multi-label learning*) [2].

VisualJCLAL brinda también la posibilidad de incluir escenarios de AL, algoritmos y estrategias de consulta mediante *plugins* creados por el usuario, gracias al uso de una arquitectura basada en componentes y al diseño flexible de clases e interfaces de JCLAL. Para ello el usuario utilizará la opción de agregar un nuevo *plugin* al sistema donde sus clases serán añadidas a los archivos de configuración en formato **.props** y serán mostrados en la GUI aquellos componentes que implementen la interfaz *IPlugin*. Estos parámetros se incluirán automáticamente dentro de las opciones de la interfaz gráfica usando la estructura modular y el generador de interfaces del entorno de usuario.

Por otra parte, este módulo aprovecha las posibilidades de JCLAL en su versión 2.0 para la ejecución de los algoritmos en forma paralela y distribuida (esta última utilizando el framework Apache Spark), además de otras funcionalidades que resultan muy útiles para la realización de experimentos de AL.

La Figura 1 muestra la pantalla inicial que aparece tras ejecutar VisualJCLAL. A partir de ella se tienen tres opciones principales:

- Crear nuevo experimento (*New Experiment*): permite crear una nueva configuración desde cero para la ejecución de un experimento.
- Cargar la configuración de un experimento (*Make from XML*): permite cargar una configuración de un experimento creado anteriormente y que se encuentra almacenado en un archivo XML.
- Ver resultados (*View results*): permite analizar de forma gráfica los resultados de los experimentos ejecutados en JCLAL.

Figura 1. Interfaz inicial.

A la interfaz de configuración de un experimento se puede acceder a través de la opción *New Experiment* o cargando una configuración existente mediante la opción *Make from XML*. Como se puede observar en la Figura 2, en las diferentes pestañas de la interfaz se pueden configurar todos los parámetros necesarios para la ejecución del experimento.

En cuanto a la opción *View results*, esta nos permite cargar los ficheros generados por JCLAL y analizar gráficamente los resultados de un experimento, como se muestra en la Figura 3. Mediante la selección de una medida de evaluación se puede observar el comportamiento de varias curvas de aprendizaje al mismo tiempo, permitiendo realizar una comparación visual entre diferentes estrategias de AL. Además, esta interfaz permite configurar y analizar las gráficas mediante el cambio de colores, ocultar o mostrar curvas, definir rangos de evaluación, cambiar las formas de las líneas y calcular el área bajo una curva de aprendizaje.

Otra opción importante que brinda VisualJCLAL es la posibilidad de crear un estudio experimental con múltiples configuraciones a partir de una configuración básica (la configuración base debe contar con los parámetros mínimos

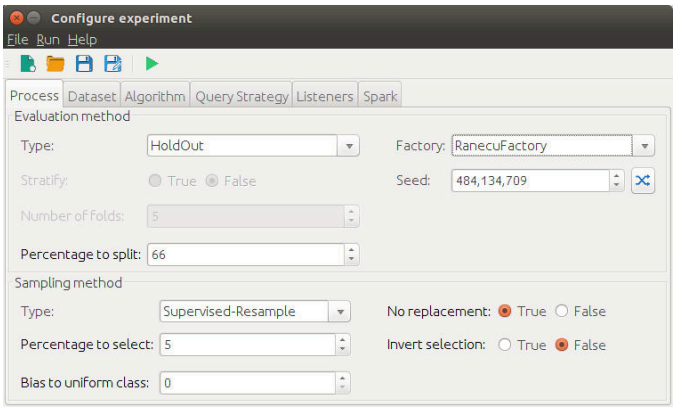

Figura 2. Interfaz para la configuración de un experimento.

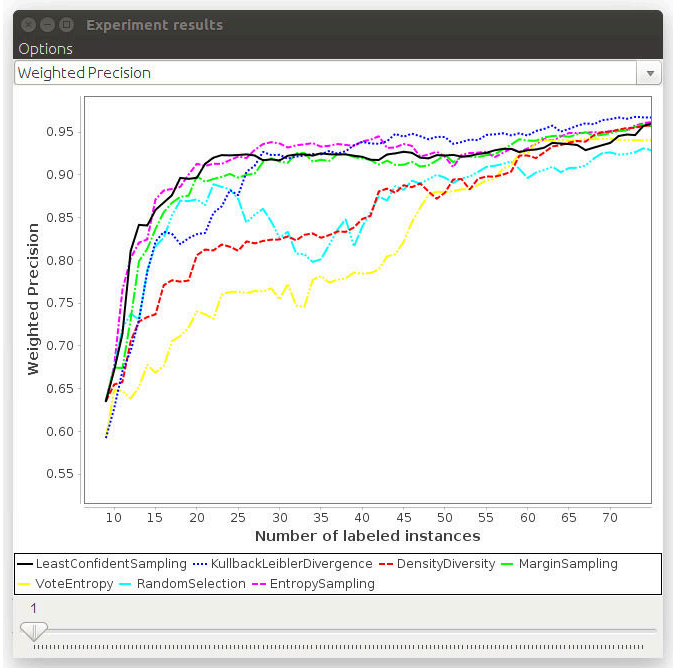

Figura 3. Interfaz de usuario para la visualización de resultados.

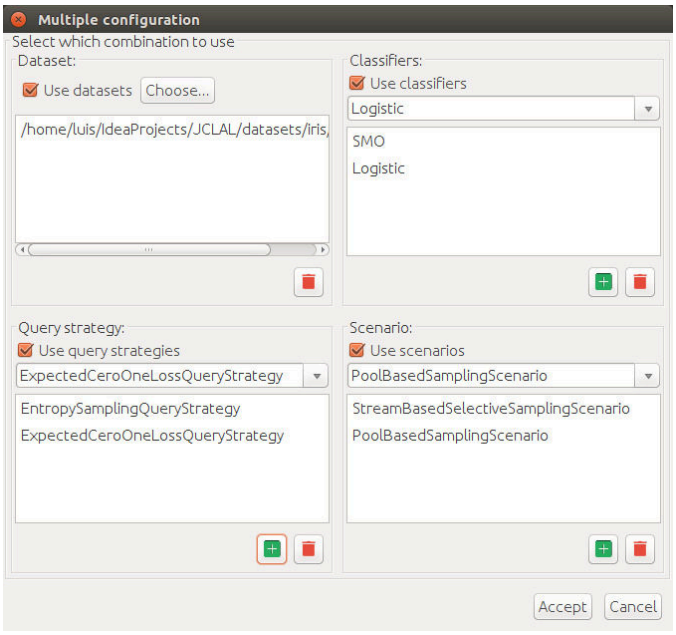

Figura 4. Creación de un estudio experimental.

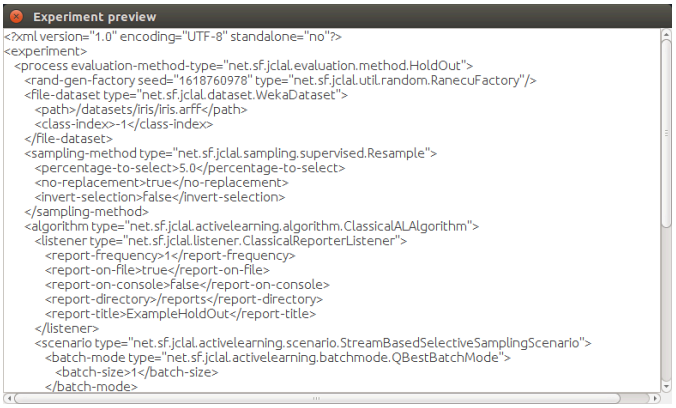

Figura 5. Vista previa del archivo de configuración.

necesarios para la ejecución de un experimento). En la Figura 4 se puede observar cómo se pueden generar N configuraciones en dependencia de las opciones escogidas.

Por último, se cuenta con la opción de chequear cómo quedaría la configuración del experimento antes de ser almacenada en un fichero, como se muestra en la Figura 5.

IV. Integración de JCLAL con el framework Spark

Big Data es un término muy popular en la actualidad, que hace referencia a problemas que involucran volúmenes de datos que superan la capacidad de los software clásicos y las máquinas convencionales para ser capturados, gestionados y procesados en un tiempo razonable. Apache Spark es uno de los framework más populares para el procesamiento de enormes conjuntos de datos, el cual implementa el paradigma de programación MapReduce [16] propuesto por Google. Spark almacena los resultados intermedios en memoria, logrando ejecutar aplicaciones mucho más rápido que las soluciones desarrolladas con Hadoop [17], lo cual hace de Spark una solución más adecuada para el desarrollo de algoritmos de aprendizaje automático altamente escalables. En este sentido, Spark cuenta con la librería MLlib [18] para el desarrollo de algoritmos distribuidos, la cual tiene implementaciones distribuidas de algunos de los algoritmos supervisados y no supervisados más populares, como Naive Bayes [19], árboles de decisión [20] y *k-means* [21].

La versión 1.0 de JCLAL se encontraba limitada a la hora de ejecutar experimentos con grandes conjuntos de datos, principalmente porque los algoritmos de aprendizaje que provee el framework Weka, o Mulan [22] para el caso de datos multi-etiqueta, no están diseñados para trabajar en este tipo escenarios. En la versión actual de JCLAL se ha desarrollado

un módulo que integra a JCLAL con Spark, aprovechando las ventajas que tiene este último para el procesamiento paralelo y distribuido de datos masivos. El módulo de AL distribuido (**jclal-spark**) está diseñado de forma tal que puede trabajar con los modelos de clasificación que provee la librería MLlib. La arquitectura es similar a la de **jclal-core**, siendo la estructura de paquetes **net.sf.jclal.spark.***.

Para utilizar **jclal-spark** a través de Maven hay que añadir el *artifactid* de este módulo, de esta manera se instalan automáticamente todas las dependencias necesarias para el trabajo con Spark. Además, en el fichero de configuración de un experimento, se debe incluir la etiqueta que indica la configuración para conectarse al servidor Spark. La etiqueta **master** especifica la dirección donde se encuentra el nodo maestro y en **app-name** se indica el nombre que se desea asignar al trabajo.

```
<spark>
  <master>spark://localhost:7077</master>
  <app-name>MyAlJob</app-name>
<spark>
```

Debido a que Spark provee sus propios modelos de clasificación, **jclal-spark** implementa las clases *SparkClassifier* y *SparkCommitteeClassifier* que sirven de interfaz para acceder a dichos clasificadores directamente (ver Figura 6). La clase encargada de realizar las evaluaciones de los clasificadores de Spark es *SparkSingleLabelEvaluation*, que además provee las medidas de tiempo de ejecución e información acerca de los conjuntos de datos.

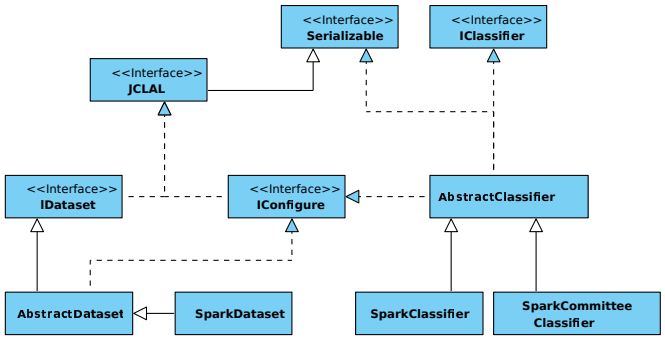

Figura 6. Diagrama de las clases principales del paquete **jclal-spark**.

Es válido destacar que el procesamiento de los conjuntos de datos en un ambiente distribuido es completamente diferente a como se hace en **jclal-core**. La clase *SparkDataset* se ha implementado para el manejo de datos distribuidos a través del conjunto de acciones y operadores que brinda la librería Spark. Los datos pueden ser cargados directamente desde disco o en un sistema de archivos distribuidos los cuales son primeramente indexados para luego ser procesados en el framework. En la nueva versión de JCLAL se añadió la etiqueta **format** para especificar el formato del conjunto de datos, logrando una mayor compatibilidad con Spark y otras librerías que se puedan incluir en el futuro. El ejemplo siguiente muestra como utilizar un conjunto de datos en formato **libsvm** empleando *SparkDataset*.

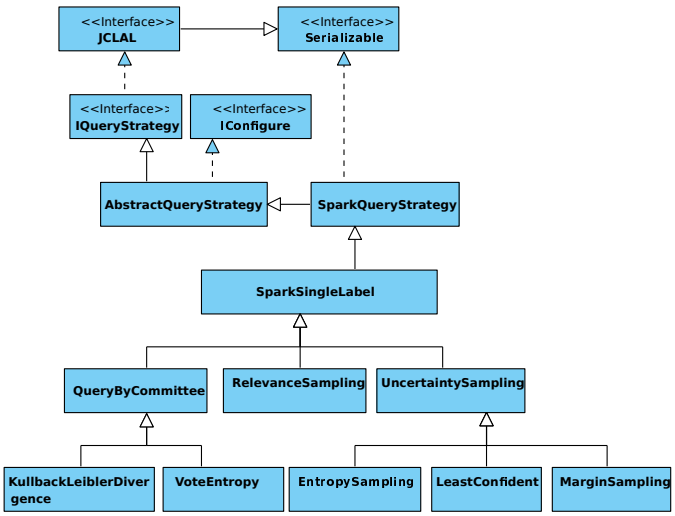

Figura 7. Diagrama de clases de las estrategias de consulta implementadas en el módulo **jclal-spark**.

```
<file-dataset type="net.sf.jclal.spark.dataset.SparkDataset">
  <path>path/to/dataset</path>
  <format>libsvm</format>
<file-dataset>
```

Respecto a las estrategias de consultas de AL implementadas en el paquete **jclal-spark**, hasta el momento están implementadas aquellas estrategias de consultas que se adaptan directamente al paradigma MapReduce, pero se espera implementar próximamente el resto de estrategias contenidas en **jclal-core**. La figura 7 muestra el diagrama de clases de las estrategias de consulta que actualmente soporta el módulo **jclal-spark**; estas son *Relevance Sampling*, y todas aquellas derivadas de *Uncertanty Sampling* (*Entropy Sampling*, *Least Confident* y *Margin Sampling*) y *Query By Committee* (*Kullback Leibler Divergence* y *Vote Entropy*).

Por último, en **jclal-spark** están implementados los métodos de evaluación clásicos: *Hold-Out*, *k-Fold Cross Validation*, *5X2 Cross Validation* y *Leave One Out Cross Validation*. En el caso de los métodos de validación cruzada se utilizaron funciones de Spark que están optimizadas para la creación de los *folds* en conjuntos de datos distribuidos.

V. Conclusiones

En este trabajo se describieron las mejoras y nuevas funcionalidades que se incluyen en la nueva versión del framework JCLAL. En su nueva versión, el núcleo de JCLAL cuenta con una nueva estructura de clases y funcionalidades que lo convierten en un framework más potente para la investigación en el área de AL. Se logró una mayor flexibilidad e independencia en la manipulación de los conjuntos de datos, haciendo posible integrar a JCLAL con datos provenientes de cualquier otro framework. Además, se le dio un mayor soporte a algoritmos de aprendizaje incremental, lo cual posibilita una mayor eficiencia en el proceso de construcción de modelos predictivos. Por otra parte, se añadió un paquete que permite realizar contrastes de hipótesis mediantes test estadísticos no

paramétricos. Otra nueva mejora radica en la descomposición del framework en módulos independientes, lo cual facilita su portabilidad y distribución. De esta manera se puede trabajar con las funciones deseadas del framework sin tener instaladas todas y cada una de las librerías del árbol de dependencias.

JCLAL 2.0 también incorpora el entorno VisualJCLAL que permite trabajar con el framework de forma intuitiva y flexible, acercando JCLAL a los usuarios no expertos en el dominio y permitiendo un ahorro significativo de tiempo. Por último, se incorporó un nuevo e importante módulo, que permite de forma sencilla aplicar programación distribuida en JCLAL. El módulo de AL distribuido utiliza la popular librería Spark, lo cual permite el uso de JCLAL 2.0 en problemas *Big Data* y posibilita una mejor acogida por parte de la comunidad científica.

Agradecimientos

Este trabajo ha sido financiado por el proyecto TIN2017-83445-P del Ministerio de Economía y Competitividad y Fondos FEDER. También ha sido financiado por el contrato i-PFIS no. IFI17/00015 otorgado por el Instituto de Salud Carlos III.

Referencias

[1] O. Reyes, A. H. Altalhi, and S. Ventura, "Statistical comparisons of active learning strategies over multiple datasets," *Knowledge-Based Systems*, vol. 145, pp. 274–288, 2018.

[2] O. Reyes, C. Morell, and S. Ventura, "Effective active learning strategy for multi-label learning," *Neurocomputing*, vol. 273, pp. 494–508, 2018.

[3] O. Reyes and S. Ventura, "Evolutionary strategy to perform batch-mode active learning on multi-label data," *ACM Trans. Intell. Syst. Technol.*, vol. 9, no. 4, pp. 46:1–46:26, 2018.

[4] Y. Y. Yang, S. C. Lee, Y. A. Chung, T. E. Wu, S. A. Chen, and H. T. Lin, "libact: Pool-based active learning in python," *arXiv preprint arXiv:1710.00379*, 2017.

[5] O. Reyes, E. Pérez, M. C. Rodríguez-Hernández, H. M. Fardoun, and S. Ventura, "JCLAL: a java framework for active learning," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3271–3275, 2016.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[8] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[9] J. Alcalá-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas *et al.*, "Keel: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009.

[10] R. Johnson, *Expert One-on-one J2EE Design and Development*. Wrox Press, 2002.

[11] F. P. Miller, A. F. Vandome, and J. McBrewster, "Apache maven," 2010.

[12] J. I. Jaen, J. R. Romero, and S. Ventura, "VisualJCLEC: A visual framework for evolutionary computation," in *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*. IEEE, 2012, pp. 119–125.

[13] M. Kronfeld, H. Planatscher, and A. Zell, "The EvA2 optimization framework," in *International Conference on Learning and Intelligent Optimization*. Springer, 2010, pp. 247–250.

[14] J. Brownlee *et al.*, "Oat: The optimization algorithm toolkit," *Centre for Information Technology Research (CITR), Swinburne University of Technology, Victoria, Australia, Technical Report A*, vol. 20071220, 2007.

[15] S. Wagner and M. Affenzeller, "Heuristiclab: A generic and extensible optimization environment," in *Adaptive and Natural Computing Algorithms*. Springer, 2005, pp. 538–541.

[16] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mccauley, M. Franklin, S. Shenker, and I. Stoica, "Fast and interactive analytics over hadoop data with spark," *Usenix Login*, vol. 37, no. 4, pp. 45–51, 2012.

[18] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[19] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[20] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[21] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[22] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *Journal of Machine Learning Research*, vol. 12, pp. 2411–2414, 2011.