Analizador Java Inteligente

López De Luise María Daniela, miembro IT-Lab de la Universidad de Palermo, mlopez74@palermo.edu
Agüero Martín Jorge, miembro IT-Lab de la Universidad de Palermo, agüero.martin@gmail.com

Universidad de Palermo, Buenos Aires, Argentina, TE/FAX (54 11) 5199 4520

Abstract
Varios intentos se han realizado para mejorar la producción de programas en diversos lenguajes. No es difícil imaginar cuán conveniente sería poder aplicar técnicas inteligentes para colaborar en este sentido. El presente trabajo realiza el desarrollo y evaluación de un prototipo analizador de métricas de calidad para código escrito en lenguaje de programación Java. El análisis comprende la descripción básica del diseño, descripción de los principios de funcionamiento y presentación de resultados preliminares.

Keywords
Métricas, neural networks, calidad de software

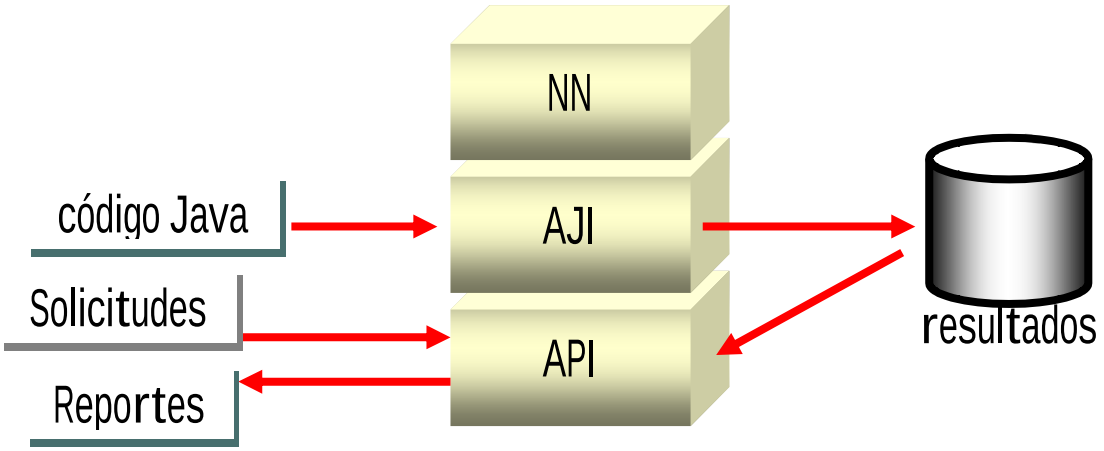
Introducción

Hoy en día la evolución del software requiere que las empresas se vean implicadas en una abultada inversión alrededor del desarrollo y mantenimiento de sistemas. Surge entonces la necesidad de contar con herramientas válidas que permitan dar soporte a dichas tareas. La calificación automática del software, a fin de determinar si un módulo de código está suficientemente limpio o libre de errores, es una meta preciada que lograría en muchos casos facilitar el proceso de puesta en marcha y mantenimiento.
Esta actividad, por demás tediosa y artesanal, cuenta con pocas herramientas de apoyo como JUnit, AgitarONE, Bugzilla, JCosmo, las que, en ningún caso son abarcativas de todo el proceso involucrado.
El prototipo que aquí se presenta es el resultado de un año y medio de trabajo fusionando técnicas históricamente conocidas (métricas de calidad de software, buenas prácticas, estándares de codificación, etc.) con nuevas métricas y conceptos propios, a fin de evaluar código Java, definir defectos más probables, y asesorar sobre cuestiones concretas dentro del código auditado.

Descripción del prototipo

El prototipo AJI (Analizador Java Inteligente) se basa en el concepto de detección automática del perfil del codificador para poder predecir sus equivocaciones y errores más probables. Está compuesto por tres módulos como se muestra en la figura 1.

Fig. 1: componentes AJI



El módulo **API** se centra en el manejo de toda la interacción básica con el usuario. El módulo **NN** en cambio es una red neuronal configurada para poder aprender dinámicamente los perfiles de usuario a partir del análisis de 14 variables extraídas de programas codificados en Java. Parte de estas variables son descriptivas (ej: cantidad de clases abstractas, cantidad de interfaces, etc.) y otras son calificativas (ej: clases con nombres mal escritos, balanceos de comentarios vs. código, etc.).
El módulo **AJI** es el que realiza la actividad principal. Sus componentes se presentan en la Fig. 2.

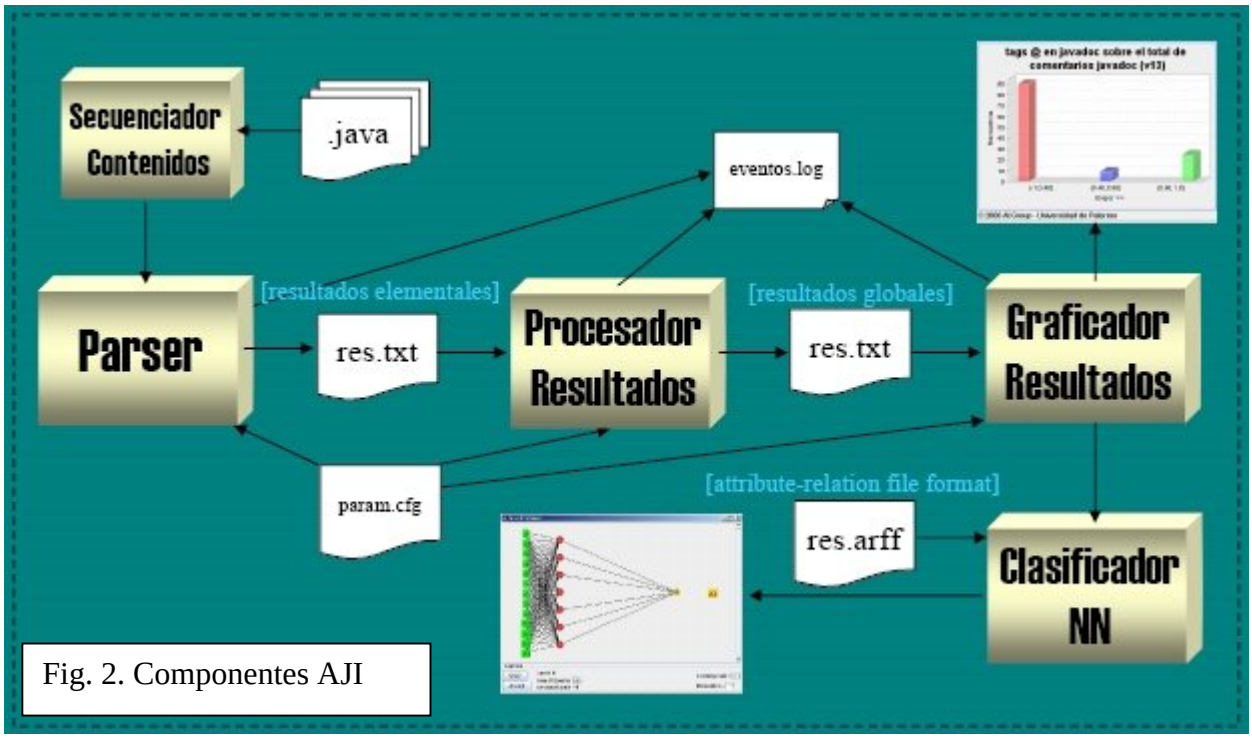


Fig. 2. Componentes AJI

Como se puede apreciar en la figura, existe un componente *Secuenciador* que captura el código fuente y lo preprocesa para facilitar la tarea del *Parser*. El *Parser* produce una serie de datos en bruto que luego son usados por el *Procesador de resultados* para elaboraciones más complicadas. Finalmente estos resultados serán entregados al *Graficador de resultados* del módulo API.

Funcionamiento del prototipo

El prototipo tiene dos modalidades de funcionamiento: *aprendizaje* y *evaluación*. Cuando está en modalidad *aprendizaje*, El secuenciador recibe un lote de archivos java en ASCII y la red neuronal cambia dinámicamente sus pesos hasta llegar a detectar los perfiles. Actualmente el prototipo ha corrido con más de 200 archivos y la red tiene un poder de clasificación de perfiles con una certeza cercana al 99%.
Cuando el prototipo ha estabilizado su aprendizaje está habilitado para comenzar la fase de evaluación por un período definido por el administrador (y podrá re-entrenarse a solicitud del mismo).
La actividad en etapa de *evaluación* utiliza los mismos componentes, salvo por el hecho de que la red neuronal es usada para clasificar los perfiles previamente detectados. Cada perfil a su vez tiene asociada una serie de *características más probables*, las cuales serán contrastadas contra el código actual y serán la base de las conclusiones y recomendaciones que el sistema le informará convenientemente al usuario a través del componente API.

Resultados

Se realizó un proceso de aprendizaje completo, comenzando con la ponderación y clasificación de las características de un lote de código fuente (ver fig. 3).

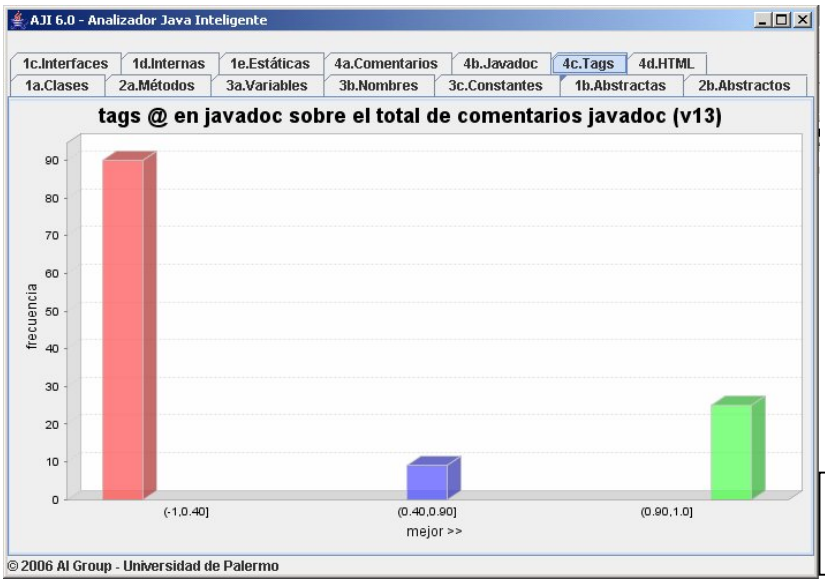


Fig. 3. Ponderación y clasificación de código

Luego, la herramienta AJI procesó y evaluó automáticamente los perfiles usando técnicas de inteligencia artificial, específicamente una red neuronal cuya configuración es la siguiente:

- Perceptrón de 1 capa
- 14 nodos de entrada
- 9 nodos ocultos
- 5 nodos de salida (un nodo por cada perfil esperado)

En la figura 4 se muestra el perceptrón funcionando durante un proceso de clasificación.

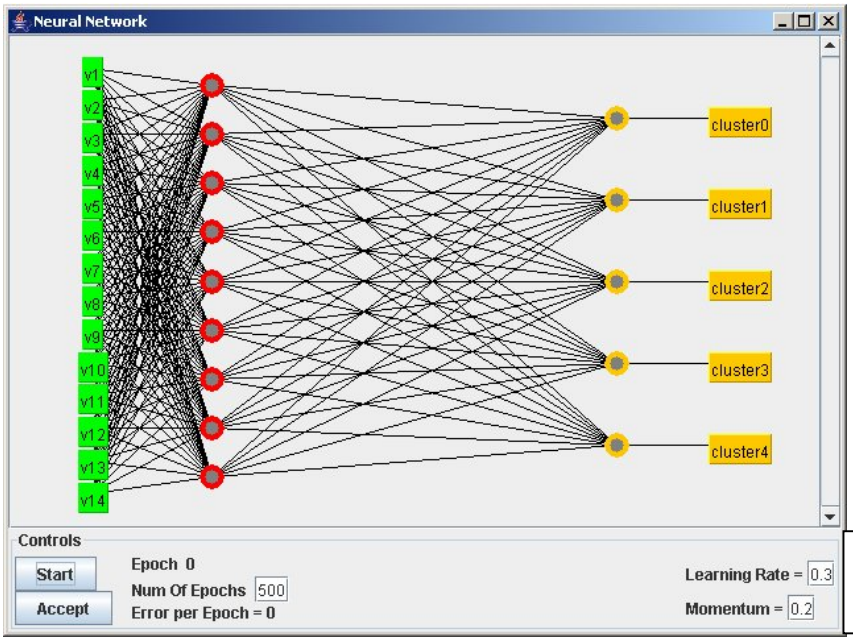


Fig. 4. Red neuronal del módulo NN

La cantidad de perfiles resulta de una investigación previa realizada en base a más de 200 archivos de código, en la que se aplicó técnicas de clustering. El cuadro de la figura 4 compara los resultados alcanzados al clasificar con diferentes algoritmos de clustering:

Clustering	*bins*	*%ok*	*k*	*MAE*	*Root MSE*	*H*	*I*
EM	5	87.1	0.826	0.071	0.200	14	5
Kmeans	10	94.4	0.922	0.023	0.092	22	10
Kmeans	**5**	**95.2**	**0.926**	**0.028**	**0.120**	**14**	**5**
FartherFirst	10	92.7	0.854	0.020	0.105	22	10
FartherFirst	5	87.1	0.826	0.071	0.200	15	5

Como se puede apreciar, KMeans obtiene la mejor clasificación cuando se escojen 5 clusters, con un nivel de confianza muy cercano a la predicción perfecta (ver valores K[1], MAE[2], Root MSE[3], H[4], I[5]).
Usando ésto, el Analizador Java Inteligente clasificó cada uno de los archivos de código fuente según atributos relevados a partir de las métricas predefinidas y el training set generado por clustering.

El proceso de la red neuronal, genera una calificación cercana al 99% de exactitud en la precisión. Es decir, de los 213 archivos Java, sólo 3 de ellos fueron clasificados erróneamente.

En la fig. 5 se muestra un cuadro con los resultados finales obtenidos del proceso de aprendizaje.

```
=== Confusion Matrix ===

  a  b  c  d  e   <-- classified as
 84  1  0  0  0 |  a = cluster0
  1 62  0  0  0 |  b = cluster1
  0  1 16  0  0 |  c = cluster2
  0  0  0 23  0 |  d = cluster3
  1  0  0  0 24 |  e = cluster4

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.988     0.016       0.977    0.988      0.982   cluster0
  0.984     0.013       0.969    0.984      0.976   cluster1
  0.941     0           1        0.941      0.97    cluster2
  1         0           1        1          1       cluster3
  0.96      0           1        0.96       0.98    cluster4
```

Fig. 5. resultado de clasificación

A la fecha, el grupo de investigación está avocado a la implementación de un Módulo de Recomendación que utilice éstos perfiles detectados para interactuar con el usuario.

[1] K es el valor kappa que mide la concordancia de categorizaciones predecidas vs. observadas.
[2] MAE mean absolute error, promedio de valor absoluto del error.

[3] Root MSE, raíz cuadrada del promedio de errores al cuadrado, tiende a exagerar el error de los outlayer.
[4] H, cantidad de nodos ocultos de la red neuronal.
[5] I, cantidad de nodos de entrada de la red neuronal.

Bibliografía:

[1] ”Aplicación de Métricas Categóricas en Sistemas Difusos”, M. D. López De Luise, Martín Agüero. IEEE Latin America. 2006.
[2] ”Finite State Machines”, R. C. Martin.Engineering Notebook Column. 1997
[3] ”Analyzing Java Software by Combining Metrics and Program Visualization”, T. Syst, P. Yu, H. Müller. 2000.
[4] ”NASA Coding Standards for C, C++, and Java”. NASA. 1999.
[5] ”Thinking in Java 3rd edition”, B. Eckel. Prentice-Hall. 2002.
[6] “ Data Mining: Practical machine learning tools and techniques (Weka)”, Ian H. Witten and Eibe Frank. 2005
[7] ”Applied Software Measurement”, Capers Jones. McGraw-Hill. 1997