

Predicción del Desempeño de un Usuario en el Juego “Kids Play Math”

Cecilia Baggio - Ana G. Maguitman

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca, Argentina

Resumen Con el advenimiento de la era de la información, un gran número de sistemas informáticos cuenta con la posibilidad de almacenar datos a partir de las distintas ejecuciones y generar cierto feedback para luego realizar minería de datos sobre los mismos. Un caso particular de esto último son los juegos informáticos aplicados a educación. El presente trabajo involucra la implementación de herramientas que faciliten la predicción del desempeño de un jugador (de 3 a 5 años de edad) basadas en el juego educativo “KPM: Kids Play Math” mediante la aplicación de técnicas de aprendizaje automático para el etiquetado de datos secuenciales. Esto último se realiza en base a la historia de cada jugador y tomando un modelo construido a partir de un conjunto de entrenamiento. En particular, este trabajo ha dado como resultado herramientas novedosas capaces de: (1) generar casos de entrenamiento a partir de logs del juego previamente procesados y (2) procesar nuevos conjuntos de datos con el fin de predecir etiquetas que anticipen el desempeño de un jugador.

Keywords: Minería de Datos, Campos Aleatorios Condicionales, Predicción de Desempeño, CRFSuite, Juegos Educativos, “Kids Play Math”

1. Introducción

La minería de datos puede mejorar el diseño de los juegos de distintas maneras. Entre ellas

- Balancear la economía.
- Detectar jugadores tramposos (cheaters).
- Reducir costos de producción.
- Mantener entretenido a los jugadores y no permitir que se desanimen ante resultados desalentadores. Hacer que los mismos mantengan su máxima performance durante el juego.

La minería de datos también proporciona mejores teorías. Le da al diseñador del juego mejor percepción de cómo los jugadores usan y abusan del juego. Se amplía la perspectiva, se prueban o refutan hipótesis y se reemplazan las opiniones por hechos.

El comportamiento del jugador proporciona información rica en cuanto al balance del juego, así que se debe recoger la mayor cantidad de datos posible

(una muestra grande). Al igual que cualquier otra colección de datos estadísticos, la muestra debe ser al azar o de otro modo, representativa de las proporciones reales de la población de jugadores. Cuanto más grande sea la muestra, más clara será la imagen. En un juego perfecto, un número infinito de jugadores haría un retrato perfecto del comportamiento de los jugadores. En el otro extremo, una muestra pequeña o sesgada, no genera estadísticas significativas.

Este trabajo resume los resultados obtenidos como parte de la Tesis de Licenciatura en Ciencias de la Computación llevada a cabo en el año 2013 en la Universidad Nacional del Sur. El objetivo de dicha Tesis ha sido el estudio e implementación de herramientas que faciliten la predicción del desempeño de un jugador (de 3 a 5 años de edad) basado en el juego educativo “*KPM: Kids Play Math*” mediante la aplicación de técnicas de aprendizaje automático para el etiquetado de datos secuenciales. Esto último se realiza en base a la historia de cada jugador y tomando un modelo construido a partir de un conjunto de entrenamiento. Específicamente se utilizará la herramienta *CRFSuite*, basada en la aplicación de un modelo estadístico denominado *Campo Aleatorio Condicional* (del inglés *Conditional Random Fields* o *CRF*). Este modelo a menudo se utiliza para el reconocimiento de patrones, aprendizaje automático y clasificación de elementos atómicos de un texto (tokens) en diferentes entidades (nombres, fechas, porcentajes, etc.). Mientras que los clasificadores comunes predicen una etiqueta para una sola muestra, sin tener en cuenta las muestras “vecinas”, *CRF* puede tomar en cuenta el contexto.

Este trabajo de investigación dará como resultado herramientas novedosas capaces de:

- Generar casos de entrenamiento a partir de logs del juego previamente procesados.
- Procesar nuevos conjuntos de datos con el fin de predecir etiquetas.

En las secciones siguientes, se describe el proceso de investigación y desarrollo que hemos llevado a cabo con el fin de satisfacer las metas propuestas. No se asumen conocimientos previos del juego ni de las herramientas utilizadas. En la Sección 2, comenzaremos describiendo el juego *Kids Play Math*, y a qué nos referimos cuando hablamos de *Minería de Datos aplicada a juegos*, fundamental para entender en qué se basan los estudios posteriores. Luego encontraremos una breve introducción a la herramienta de aprendizaje automático *Conditional Random Fields*, junto con la descripción de la versión utilizada: *CRFSuite* (Sección 3). En la Sección 4 describimos las funcionalidades que provee la herramienta desarrollada (detalles de su implementación pueden verse en el Apéndice B). A continuación presentamos los resultados que obtuvimos a partir de las distintas pruebas realizadas con la nueva herramienta (Sección 5) y, finalmente, las conclusiones a las que hemos arribado luego de haber realizado dichas pruebas junto con una propuesta para darle continuidad a estos estudios (Secciones 6 y 7).

2. Kids Play Math

2.1. ¿Qué es Kids Play Math?

Kids Play Math [1] es un programa de computadoras bilingüe (en Inglés y Español) que se utiliza como material complementario en la enseñanza de matemática a niños en edad preescolar (3 a 5 años).

Consiste de un software de computadoras y capacitación profesional docente. Soporta el concepto de desarrollo matemático integrado en todos los aspectos de las aulas de *Head Start*¹ y en el trabajo con las familias. Kids Play Math fue desarrollado a partir del subsidio “Head Start Innovation & Improvement” otorgado a la Universidad de Denver.

2.2. ¿Qué sabemos de Kids Play Math hasta ahora?

Estudios recientes muestran que la matemática a temprana edad es un aspecto muy importante para la preparación escolar: los niños que comienzan el jardín que trata de desarrollar más habilidades matemáticas, tienden a tener un mejor desempeño en matemática, lectura y escritura en 5to grado.

También es sabido que los niños aprenden más de lo que uno imagina, si han sido educados de acuerdo a los caminos de aprendizaje que los estudios muestran que podrían ser efectivos. Estos métodos de enseñanza pueden reducir la brecha entre niños de bajo nivel socio-económico y los equivalentes de clase media.

Dado que el programa Kids Play Math integra el aprendizaje de los niños (vía juegos de computadora y otras actividades) con el perfeccionamiento docente (en cuanto al entendimiento y la enseñanza de matemática elemental), puede acarrear efectos positivos, a corto y largo plazo: los niños estarán mejor preparados y tendrán un mejor rendimiento en los próximos años escolares, y los docentes estarán mejor preparados para enseñar a sus futuros alumnos.

Pese a que los números en los estudios piloto fueron pequeños, estos indican que los niños que juegan Kids Play Math en sus clases han presentado ganancias significativamente superiores en matemática, respecto a los niños de otras clases. Esto sucedió en menos de un año de uso del programa.

2.3. Un pantallazo general del juego

Kids Play Math existe actualmente en dos versiones: web y de escritorio, aunque la última cuenta con más funcionalidad. En la versión de escritorio existen dos formas de iniciar sesión:

- *Docente* (o teacher), con acceso a todas las funcionalidades pensadas para el mismo, tales como la creación de nuevas clases o cursos, altas, modificaciones y bajas de alumnos, generación de reportes para el docente con el grado de

¹ *Head Start* es un programa federal que promueve la preparación escolar de los niños que provienen de familias de bajos ingresos, fomentando su desarrollo cognitivo, social y emocional, desde que nacen, hasta los cinco años.

avance de los alumnos y elección sin restricciones de cualquier juego y/o nivel, entre otras;

- *Alumno*, limitado a los conocimientos y aptitudes de cada alumno en particular. Esto es así, debido a que a medida que el niño avanza en el juego, según su comportamiento en el mismo, se van destrabando (o trabando) nuevos mini-juegos o burbujas con más complejidad que los anteriores, dándole al jugador la posibilidad de aprender cosas nuevas a medida que gana o pasa burbujas, y/o repasar temas jugados con anterioridad, en caso de que no sea una partida satisfactoria.

Cada *mini-juego o burbuja* posee un número de nivel, y pertenece a una de las siguientes categorías: Numbers-Count, Numbers-Subset, Addition, Numbers-Identify, Geometry, Spatial-Sense, Data-Measurements o Comparison.



Fig. 1. Pantalla principal

Luego de que el *alumno* inicie sesión, se visualizará en la pantalla la Figura 1. Los distintos kioscos o booths, son quienes cargan, mediante un click en alguno de ellos, el juego que se encuentra activo, es decir, el juego que le correspondería jugar al alumno respecto a su historia en KPM.

En la Figura 2 puede verse un ejemplo de lo que sería una *burbuja o mini-juego*.

2.4. Minería de Datos aplicada a Kids Play Math

La minería de datos, un campo interdisciplinario de las Ciencias de la Computación, es el proceso computacional que intenta descubrir patrones en grandes volúmenes de datos. Involucra métodos de inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos. El objetivo general del proceso de minería de datos consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible para su uso posterior. Aparte de la etapa de análisis en bruto, involucra el manejo de bases de datos, gestión y



Fig. 2. «Haz click en 10». Categoría: IdentifyMixed - Nivel 4

pre-procesamiento de datos, consideraciones del modelo y de inferencia, métricas interesantes, consideraciones de la teoría de la complejidad computacional, post-procesamiento de las estructuras descubiertas, la visualización y actualización en línea.

Una de las finalidades de Kids Play Math ha sido la de explorar esta rama de las Ciencias de la Computación. Recientes actualizaciones del juego focalizan en la mejora de las facilidades provistas para incentivar la minería de datos. La metodología utilizada por KPM radica en el almacenamiento en archivos de toda la información referente a cada partida jugada por cada uno de los niños, conformando de esta forma el “historial” en KPM de cada niño en particular. La información es recuperada y almacenada en archivos en formato XML (uno para cada niño), denominados “logs”, de la siguiente manera: cuando un niño inicia sesión en el juego por primera vez, se crea el archivo en formato XML, con toda la información referente a dicha sesión, incluyendo el nombre del niño, fecha de creación, id, entre otros. A partir de ese momento y en adelante, será allí donde se reflejará el historial de jugadas.

Cada *mini-juego* o *burbuja* es descrito por una serie de atributos y se compone a su vez de un número arbitrario de tareas que el niño debe resolver para alcanzar la meta planteada por el juego. Toda esta información es almacenada a medida que el niño juega, y será posteriormente recuperada para realizar minería de datos. Un ejemplo del formato de los logs generados puede verse en el Apéndice C, Sección C.1.

Los archivos de log son pre-procesados antes de ser utilizados para el estudio del comportamiento de los jugadores. Del total de atributos de cada burbuja se recupera sólo un conjunto reducido de los mismos, aquellos que según nuestro criterio, aportan información más relevante sobre la partida. En el Apéndice C, Sección C.1, se encuentra una descripción completa de todos los atributos del juego.

Los atributos que resultan del pre-procesamiento de los archivos de log, son los siguientes: *bubbleName*, *bubbleLevel*, *bubbleGroup*, *LastOutcome*, *globalScore*, *NowIsActive* y *Outcome*

3. Herramientas

3.1. Conditional Random Fields

Los datos relacionales (relational data) tienen dos características: en primer lugar, existen dependencias estadísticas entre las entidades que deseamos modelar, y en segundo lugar, cada entidad a menudo posee un conjunto rico de atributos o características que pueden ayudar a la clasificación. Por ejemplo, en la clasificación de documentos Web, el texto de la página ofrece mucha información sobre la etiqueta de la clase, pero los hipervínculos definen una relación entre las páginas que pueden mejorar la clasificación. Los modelos gráficos son un formalismo natural para explotar la estructura de dependencias entre las entidades. Tradicionalmente los modelos gráficos se han utilizado para representar la distribución de la probabilidad conjunta $p(y, x)$, donde las variables y representan los atributos de las entidades que deseamos predecir y las variables de entrada x representan nuestro conocimiento observado acerca de las entidades. Sin embargo, el modelado de la distribución conjunta puede causar dificultades al utilizar las características locales que pueden encontrarse en los datos relacionales, ya que requiere el modelado de la distribución $p(x)$, la cual puede incluir dependencias complejas. Modelar estas dependencias entre las entradas, puede provocar modelos computacionalmente intratables, pero hacer caso omiso de las mismas puede conducir a una disminución del rendimiento. Una solución a este problema es modelar directamente la distribución condicional $p(y|x)$, que es suficiente para la clasificación. Este es el enfoque adoptado por *Conditional Random Fields* (Campos Aleatorios Condicionales). Un campo aleatorio condicional, de aquí en adelante CRF, es simplemente una distribución condicional $p(y|x)$ con una estructura gráfica asociada. Debido a que el modelo es condicional, las dependencias entre las variables de entrada x no tienen que ser explícitamente representadas, permitiendo el uso de características valiosas y globales de las entradas. Por ejemplo, en las tareas de lenguaje natural, características o atributos útiles incluyen palabras vecinas y bigramas, prefijos y sufijos, capitalización de palabras, pertenencia a léxicos específicos del dominio e información semántica de fuentes tales como WordNet. Recientemente ha habido una explosión de interés en CRFs, con aplicaciones exitosas, incluyendo procesamiento de textos, visión artificial o por computadora y bioinformática. Uno de las primeras aplicaciones a gran escala de CRFs fue desarrollada por Sha & Pereira [2] quienes alcanzaron el “estado del arte” en cuanto a performance en la segmentación de frases nominales en un texto. Desde entonces, los campos aleatorios de cadena lineal (linear-chain CRFs) han sido aplicados a muchos problemas en el procesamiento de lenguaje natural, incluyendo el reconocimiento del nombre de entidades (NER) [3], el análisis sintáctico superficial (shallow Parsing) [2,4], la identificación de nombres de proteínas en resúmenes de biología [5], la segmentación de direcciones en páginas web [6], la integración de información [7], la búsqueda de roles semánticos en el texto [8], la clasificación fonética en el procesamiento del habla [9], la identificación de fuentes de opinión [10], la alineación de palabras en la traducción automática [11], la extracción de citas en los tra-

bajos de investigación[12], la segmentación de palabras escritas en chino [13], la extracción de información de tablas en documentos de texto [14] y el análisis morfológico japonés [15], entre otros.

3.2. CRFSuite

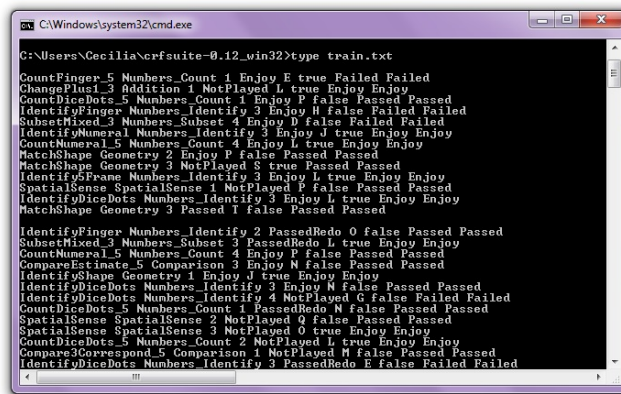
CRFSuite [16] es una implementación de CRFs para el etiquetado de datos secuenciales. Entre las diversas implementaciones de CRFs, CRFSuite consta de las siguientes características:

- *Entrenamiento y etiquetado rápido.*
- *Formato sencillo de los datos para el entrenamiento y el etiquetado.*
- *Métodos de entrenamiento de avanzada, que incluyen:*
 - Limited-memory BFGS (L-BFGS).
 - Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method.
 - Stochastic Gradient Descent (SGD).
 - Averaged Perceptron.
 - Passive Aggressive.
 - Adaptive Regularization Of Weight Vector (AROW).
- *Algoritmo forward/backward que usa el método de escalamiento (scaling method).*
- *CRF de cadena lineal (first-order Markov).*
- *Evaluación de la performance durante el entrenamiento.*
- *Formato de archivos eficiente para almacenar y acceder a los modelos CRF.*

3.3. Predicción de etiquetas en KPM utilizando CRFSuite

Descripción de la tarea: Los archivos de log, una vez pre-procesados², se transforman en secuencias de ítems, donde cada ítem es un conjunto de atributos que describe a cada burbuja jugada por un niño. Nuestro objetivo es construir un modelo que sea capaz de predecir etiquetas a partir de dichos atributos. Particularmente, una etiqueta es el resultado que obtendría un niño, para una burbuja determinada, antes de ser jugada, utilizando *CRFSuite*. Las etiquetas posibles son: *Passed*, *Failed* o *Enjoy*.

² El pre-procesamiento se explicará en detalle en Apéndice A, Sección A.1



```

C:\Windows\system32\cmd.exe
C:\Users\Cecilia\crfsuite-0.12_win32>type train.txt
CountFinger_5 Numbers_Count 1 Enjoy E true Failed Failed
ChangePlus1_3 Addition 1 NotPlayed L true Enjoy Enjoy
CountDiceDots_5 Numbers_Count 1 Enjoy P false Passed Passed
IdentifyFinger Numbers_Identify 3 Enjoy H false Failed Failed
SubsetMixed_3 Numbers_Subset 4 Enjoy P false Failed Failed
IdentifyNumeral Numbers_Identify 3 Enjoy J true Enjoy Enjoy
CountNumeral_5 Numbers_Count 4 Enjoy L true Enjoy Enjoy
MatchShape Geometry 2 Enjoy P false Passed Passed
MatchShape Geometry 3 NotPlayed S true Passed Passed
Identify5Frame Numbers_Identify 3 Enjoy L true Enjoy Enjoy
SpatialSense SpatialSense 1 NotPlayed P false Passed Passed
IdentifyDiceDots Numbers_Identify 3 Enjoy L true Enjoy Enjoy
MatchShape Geometry 3 Passed T false Passed Passed
IdentifyFinger Numbers_Identify 2 PassedRedo O false Passed Passed
SubsetMixed_3 Numbers_Subset 3 PassedRedo L true Enjoy Enjoy
CountNumeral_5 Numbers_Count 4 Enjoy P false Passed Passed
CompareEccinate_5 Comparison 3 Enjoy M false Passed Passed
IdentifyShape Geometry 1 Enjoy J true Enjoy Enjoy
IdentifyDiceDots Numbers_Identify 3 Enjoy N false Passed Passed
IdentifyDiceDots Numbers_Identify 4 NotPlayed G false Failed Failed
CountDiceDots_5 Numbers_Count 1 PassedRedo M false Passed Passed
SpatialSense SpatialSense 2 NotPlayed Q false Passed Passed
SpatialSense SpatialSense 3 NotPlayed O true Enjoy Enjoy
CountDiceDots_5 Numbers_Count 2 NotPlayed L true Enjoy Enjoy
Compare3Correspond_5 Comparison 1 NotPlayed M false Passed Passed
IdentifyDiceDots Numbers_Identify 3 PassedRedo E false Failed Failed

```

Fig. 3. Atributos que describen a una burbuja jugada en KPM

Conjunto de datos de Entrenamiento y de Test: Los datos consisten en secuencias de líneas, donde cada línea está compuesta por el nombre de una burbuja (ej. ‘CountFinger_5’, ‘ChangePlus1_3’), grupo al cual pertenece la burbuja (ej. ‘Numbers_Count’, ‘Addition’), número de nivel, resultado de la vez anterior que el niño jugó a esa burbuja (ej. ‘Enjoy’, ‘NotPlayed’), categoría de la puntuación que obtuvo con esa burbuja (ej. ‘E’, ‘L’, ‘A’), si la burbuja sigue activa luego de haber sido jugada (‘true’ o ‘false’), y por último, el resultado (*Outcome*) obtenido (ej. ‘Failed’, ‘Passed’) y la etiqueta a predecir, que en nuestro caso será igual que el *Outcome*, dado que es lo que queremos predecir, todos separados por caracteres de espacio.

Generación de atributos El próximo paso es el pre-procesamiento de los datos de entrenamiento y de test para extraer los atributos que caracterizan a las burbujas. *CRFSuite* genera internamente características de los atributos en un conjunto de datos. En general, este es el proceso más importante enfocado al aprendizaje automático, dado que un buen diseño de las características, afecta en gran medida la precisión en la predicción de etiquetas. En este ejemplo, extraemos *sólo* 19 atributos, de una burbuja en la posición t (en desplazamientos desde el comienzo de la secuencia):

- bubble[t-2], bubble[t-1], bubble[t], bubble[t+1], bubble[t+2],
- bubble[t-1]|bubble[t], bubble[t]|bubble[t+1],
- group[t-2], group[t-1], group[t], group[t+1], group[t+2],
- group[t-2]|group[t-1], group[t-1]|group[t], group[t]|group[t+1],
- group[t+1]|group[t+2], group[t-2]|group[t-1]|group[t],
- group[t-1]|group[t]|group[t+1], group[t]|group[t+1]|group[t+2]

En esta lista, **bubble[t]** y **group[t]** representan el nombre de la burbuja y el grupo al cual pertenece dicha burbuja, en la posición t de la secuencia. Estos atributos expresan características de la burbuja de la posición t , utilizando información de las

burbuja vecinas, por ejemplo, `bubble[t-1]` y `group[t+1]`. Por ejemplo, a partir del token ‘CountDiceDots_5’ en el ejemplo que sigue (Figura 4):

```

CountFinger_5 Numbers_Count 1 Enjoy E true Failed Failed
ChangePlus1_3 Addition 1 NotPlayed L true Enjoy Enjoy
t -> CountDiceDots_5 Numbers_Count 1 Enjoy P false Passed Passed
IdentifyFinger Numbers_Identify 3 Enjoy H false Failed Failed
SubsetMixed_3 Numbers_Subset 4 Enjoy D false Failed Failed

```

Fig. 4. Atributos de una burbuja en la posición t

se obtienen los siguientes atributos (la posición t es omitida por simplicidad),

- `bubble[-2]=CountFinger_5`, `bubble[-1]=ChangePlus1_3`, `bubble[0]=CountDiceDots_5`,
`bubble[1]=IdentifyFinger`, `bubble[2]=SubsetMixed_3`
- `bubble[-1]|bubble[0]=ChangePlus1_3|CountDiceDots_5`,
`bubble[0]|bubble[1]=CountDiceDots_5|IdentifyFinger`
- `group[-2]=Numbers_Count`, `group[-1]=Addition`, `group[0]=Numbers_Count`,
`group[1]=Numbers_Identify`, `group[2]=Numbers_Subset`
- `group[-2]|group[-1]=Numbers_Count|Addition`, `group[-1]|group[0]=Addition|Numbers_Count`,
`group[0]|group[1]=Numbers_Count|Numbers_Identify`,
`group[1]|group[2]=Numbers_Identify|Numbers_Subset`
- `group[-2]|group[-1]|group[0]=Numbers_Count|Addition|Numbers_Count`,
`group[-1]|group[0]|group[1]=Addition|Numbers_Count|Numbers_Identify`,
`group[0]|group[1]|group[2]=Numbers_Count|Numbers_Identify|Numbers_Count`

En este ejemplo, el atributo “`bubble[0]=CountDiceDots_5`” representa el evento donde el token actual es “CountDiceDots_5”.

El atributo “`group[0]|group[1]|group[2]=Numbers_Count|Numbers_Identify|Numbers_Count`” representa el evento donde el grupo de la posición actual, el siguiente, y el que está a dos posiciones adelante de la actual son `Numbers_Count`, `Numbers_Identify` y `Numbers_Count` respectivamente.

CRFSuite aprenderá las relaciones entre los atributos (Ej. “`group[0]|group[1]|group[2]=Numbers_Count|Numbers_Identify|Numbers_Count`”) y las etiquetas (Ej. “Passed”) para predecir una secuencia de etiquetas para un dado conjunto de datos.

CRFSuite requiere un conjunto de datos en el cuál cada línea o ítem comienza con su etiqueta, seguido de sus atributos, separados por caracteres TAB. Dado un conjunto de entrenamiento o de test, no es difícil implementar la conversión al formato esperado por *CRFSuite*. Una implementación de la conversión está dada por el script en *Python* “`chunking.py`”, detallado en el Apéndice C, Sección C.2.

A continuación, en la Figura 5 se puede ver un ejemplo de la conversión del archivo *train.txt* de la Figura 3 a *train.crf.txt*, compatible con el formato de datos de *CRFSuite*.

4. Funcionalidades

Para satisfacer las metas deseadas, se ha implementado una aplicación de escritorio³, que hace uso de las herramientas de aprendizaje automático mencionadas en los capítulos precedentes.

³ En los Apéndices A y B se halla una descripción detallada de la aplicación (modo de uso, pantallas, lenguaje de implementación, ejemplos de código, etc).

```

C:\Windows\system32\cmd.exe
C:\Users\Cecilia>crfsuite-0.12_win32>TYPE train.txt | scripts\chunking.py
Failed bubbleID=CountFinger_5 levelID=1 groupID=Numbers_Count lastOutcome[0]=Enjoy _BOS_
Enjoy bubbleID=ChangePlus_3 levelID=1 groupID=Addition lastOutcome[0]=NotPlayed
Passed bubbleID=CountDiceDots_5 levelID=1 groupID=Numbers_Count lastOutcome[0]=Enjoy
Failed bubbleID=IdentifyFinger levelID=3 groupID=Numbers_Identify lastOutcome[0]=E
Failed bubbleID=SubsetMixed_3 levelID=4 groupID=Numbers_Subset lastOutcome[0]=Enjoy globalSc
Enjoy bubbleID=IdentifyNumerals levelID=3 groupID=Numbers_Identify lastOutcome[0]=E
Enjoy bubbleID=CountNumerals_5 levelID=4 groupID=Numbers_Count lastOutcome[0]=Enjoy
Passed bubbleID=MatchShape levelID=2 groupID=Geometry lastOutcome[0]=Enjoy globalSc
Passed bubbleID=MatchShape levelID=3 groupID=Geometry lastOutcome[0]=NotPlayed
Enjoy bubbleID=IdentifyFrame levelID=3 groupID=Numbers_Identify lastOutcome[0]=E
Passed bubbleID=SpatialSense levelID=1 groupID=SpatialSense lastOutcome[0]=NotPlayed
Enjoy bubbleID=IdentifyDiceDots levelID=3 groupID=Numbers_Identify lastOutcome[0]=E
Passed bubbleID=MatchShape levelID=3 groupID=Geometry lastOutcome[0]=Passed globalSc
Passed bubbleID=IdentifyFinger levelID=2 groupID=Numbers_Identify lastOutcome[0]=P
Enjoy bubbleID=SubsetMixed_5 levelID=3 groupID=Numbers_Subset lastOutcome[0]=PassedRedo
Passed bubbleID=CountNumerals_5 levelID=4 groupID=Numbers_Count lastOutcome[0]=Enjoy
Passed bubbleID=CompareEstimate_5 levelID=3 groupID=Comparison lastOutcome[0]=Enjoy
Enjoy bubbleID=IdentifyShape levelID=1 groupID=Geometry lastOutcome[0]=Enjoy globalSc
Passed bubbleID=IdentifyDiceDots levelID=3 groupID=Numbers_Identify lastOutcome[0]=E
Failed bubbleID=IdentifyDiceDots levelID=4 groupID=Numbers_Identify lastOutcome[0]=N
Passed bubbleID=CountDiceDots_5 levelID=1 groupID=Numbers_Count lastOutcome[0]=PassedRedo

```

Fig. 5. Formato esperado por *CRFSuite*

Esta aplicación brinda distintas funcionalidades, entre ellas, la de selección, preprocesamiento y formateo de los datos provistos por el juego, de manera tal de respetar el formato estándar esperado por *CRFSuite*.

Permite también generar tres conjuntos de datos diferentes: conjunto de datos de entrenamiento, conjunto de datos de test, y conjunto total de datos, utilizado para pruebas del tipo *Cross Validation*, que se verán con más detalle en la Sección 5. Permite también crear un *modelo* a partir de los datos de entrenamiento previamente generados, el cual podrá ser testeado con posterioridad haciendo uso de los datos de test.

Cada conjunto de datos está conformado por una cantidad arbitraria de líneas o ítems, los cuales representan burbujas jugadas por algún niño. Estas líneas de datos, son representadas por una colección de atributos. Estos atributos no son, en su mayoría, significativos para nuestro objetivo de clasificación. Algunos serán mejores candidatos que otros para caracterizar a las burbujas jugadas. De este modo, la aplicación da la posibilidad de filtrar los datos de manera de utilizar sólo un subconjunto del total de atributos para realizar pruebas, y determinar cuáles serán aquellos que tengan mayor cobertura y precisión en la predicción de resultados.

Para la realización de las distintas pruebas, los modelos se conforman utilizando un conjunto de entrenamiento formado a partir de un subconjunto (80%) del total de los datos (datos que provienen de las distintas partidas jugadas por un conjunto de 46 niños). El resto de los datos (20%), es utilizado para testear los modelos, es decir, para predecir etiquetas y poder determinar la precisión, el recall y el valor-F (F1 score) de los modelos con los que estamos trabajando. Una vez conformes con el comportamiento de algún modelo obtenido, estamos en condiciones de utilizar todo el conjunto de datos para crear un modelo definitivo que nos sirva para la predicción de etiquetas en un entorno real de juego.

Las etiquetas que predice la aplicación son: *Passed*, *Failed* y *Enjoyed*. Las mismas indican si un niño que se proponga jugar a determinada burbuja, la ganará, la perderá, o si la aprovechará. Esta última indica que no hará la puntuación suficiente como para ganar, ni tampoco por debajo de la mínima como para perder.

En el Apéndice A se explica de manera sencilla e intuitiva el uso de la aplicación. Detalles de su implementación pueden hallarse en el Apéndice B.

5. Pruebas

Para nuestro análisis utilizamos un conjunto de datos consistente en los logs vinculados a 46 usuarios reales.

Como hemos explicado en la sección anterior, las pruebas que pueden realizarse a partir de la aplicación, son dos. Por un lado podemos crear un Modelo nuevo a partir de un subconjunto del total de los datos y *evaluarlo* o *testearlo* con los datos restantes para así determinar con qué precisión dicho Modelo predice las etiquetas. Por otro lado, podemos utilizar el conjunto entero de datos que poseemos, para realizar pruebas de tipo *Cross Validation*.

En nuestro análisis nos enfocamos en el segundo tipo de pruebas, más precisamente utilizamos *10-fold Cross-Validation*. A continuación, se muestra una tabla comparativa de las distintas pruebas realizadas utilizando 10-fold Cross-Validation sobre el mismo conjunto de datos, pero *variando los atributos que describen a una burbuja*. La finalidad de la tabla es exponer los resultados que obtuvimos y cómo determinamos el conjunto de atributos que resulta viable utilizar para la predicción de etiquetas, a través de la comparación de la media de los porcentajes obtenidos en cada una de las 10 iteraciones de la Validación Cruzada.

Tabla 1. Pruebas realizadas utilizando *10-fold Cross-Validation*

Atributos	Pruebas					
	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6
bubble[-2]				✓		
bubble[-1]			✓	✓		
bubble[0]	✓	✓	✓	✓	✓	✓
level[-2]				✓		
level[-1]			✓	✓		
level[0]	✓	✓	✓	✓		
group[-2]				✓		
group[-1]			✓	✓		
group[0]	✓	✓	✓	✓	✓	
lastOutcome[-2]				✓		
lastOutcome[-1]			✓	✓		
lastOutcome[0]	✓	✓	✓	✓	✓	
lobalScore[-2]				✓		
globalScore[-1]		✓	✓	✓		
globalScore[0]	✗	✗	✗	✗	✗	✗
stillActive[-2]				✓		
stillActive[-1]		✓	✓	✓		
stillActive[0]	✗	✗	✗	✗	✗	✗
Outcome[-2]				✓		
Outcome[-1]		✓	✓	✓		
Outcome[0]	✗	✗	✗	✗	✗	✗
Precisión	0.6538	0.6609	0.6605	0.6514	0.6326	0.6292

En esta tabla ⁴, presentamos 6 de las pruebas más significativas que hemos realizado. Se puede observar que el resultado de la *Prueba 2* (0.6609) difiere del resultado de la *Prueba 3* (0.6605) en menos de una milésima, y siendo que en la *Prueba 2* se tienen en cuenta sólo 7 atributos, mientras que en la *3* son 11. En la *Prueba 4* (0.6514), utilizamos todos los atributos posibles, y obtuvimos una precisión aun menor que en las pruebas anteriores, por lo tanto, hemos provisto evidencia que sugiere que contar con más atributos para describir a una entidad (burbujas en nuestro caso) puede conducir a una pérdida de precisión, dado que posiblemente estemos utilizando atributos no del todo discriminatorios y que sólo aportan ruido al clasificador. Por tal motivo, la selección de atributos apropiados es crucial para el buen desempeño del clasificador.

6. Conclusión

Luego de haber concluido con el desarrollo de las distintas tareas propuestas para alcanzar las metas de esta Tesis, estamos en condiciones de decir que los resultados que obtuvimos han sido satisfactorios. Hemos sido capaces de proponer, desarrollar y evaluar un *Clasificador* para un dominio en el que no se contaban con herramientas similares.

El método de modelado estadístico “Conditional Random Fields” se adapta a la problemática propuesta. A diferencia de los clasificadores estándar, el hecho de haber utilizado CRFs nos permitió hacer uso de las dependencias entre las distintas observaciones (burbujas del juego) y no solamente utilizar las características de las muestras individuales para realizar el etiquetado, sino también de las muestras anteriores (burbujas jugadas más atrás en el tiempo) en los historiales de jugadas de los niños.

Por último pero no menos importante, hemos provisto evidencia que sugiere que contar con muchos atributos para describir una entidad no siempre da mejores resultados. En la Tabla de la Sección 5, puede observarse cómo los porcentajes de precisión entre las distintas pruebas no varía significativamente a medida que se seleccionan más atributos. En dicha tabla puede apreciarse que en muchas de las pruebas la precisión disminuye al incrementar el número de atributos seleccionados.

7. Trabajos Futuros

7.1. Optimizaciones

En este trabajo de investigación hemos utilizado el algoritmo de entrenamiento *lbfgs*: *Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method*, el cual es utilizado en *CRFSuite* por defecto. Queda pendiente el estudio del comportamiento del etiquetado de entidades utilizando otros algoritmos de entrenamiento, también provistos por *CRFSuite*, tales como:

- *l2sgd*: Stochastic Gradient Descent (SGD) with L2 regularization.
- *ap*: Averaged Perceptron.
- *pa*: Passive Aggressive.

⁴ Una celda que contenga el símbolo \times significa que el valor de ese atributo no debe ser tenido en cuenta, dado que es información desconocida al momento de realizar el etiquetado

- *arow*: Adaptive Regularization Of Weight Vector (AROW).

Un estudio en profundidad de estos algoritmos requiere conocimientos avanzados en Probabilidad y Estadística y escapa a los objetivos propuestos para esta Tesis de grado.

7.2. Otras propuestas

Determinar si existe alguna trayectoria o patrón de burbujas jugadas que hacen que el niño avance más rápido en el aprendizaje. Por ejemplo, aprender primero los colores, luego las formas y por último los números, o las formas, los números y luego colores, etc.

Referencias

1. “Kids play math: Computer games with online teacher training for early math,” 2012.
2. F. Sha and F. Pereira, “Shallow parsing with conditional random fields,” *In Conference on Human Language Technology and North American Association for Computational Linguistics (HLT-NAACL)*, pp. 213–220, 2003.
3. A. McCallum and W. Li, “Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons,” *In Seventh Conference on Natural Language Learning (CoNLL)*, 2003.
4. C. Sutton *et al.*, “Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data,” *Journal of Machine Learning Research*, vol. 8, pp. 693–723, March 2007.
5. B. Settles, “Abner: an open source tool for automatically tagging genes, proteins, and other entity names in text,” *Bioinformatics*, vol. 21, no. 14, pp. 3191–3192, 2005.
6. A. Culotta, R. Bekkerman, and A. McCallum, *Extracting social networks and contact information from email and the web*. 2004.
7. M. Wick, K. Rohanimanesh, A. McCallum, and A. Doan, “A discriminative approach to ontology alignment,” *In International Workshop on New Trends in Information Integration (NTII)*, 2008.
8. D. Roth and W. tau Yih, “Integer linear programming inference for conditional random fields,” *In International Conference on Machine Learning (ICML)*, pp. 737–744, 2005.
9. A. Gunawardana, M. M. A. Acero, and J. C. Platt, “Hidden conditional random fields for phone classification,” *In International Conference on Speech Communication and Technology*, 2005.
10. Y. Choi *et al.*, “Identifying sources of opinions with conditional random fields and extraction patterns,” *In Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.
11. P. Blunsom and T. Cohn, “Discriminative word alignment with conditional random fields,” *In International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL)*, vol. 65–72, 2006.
12. F. Peng and A. McCallum, “Accurate information extraction from research papers using conditional random fields,” *In Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL’04)*, 2004.
13. F. Peng, F. Feng, and A. McCallum, “Chinese segmentation and new word detection using conditional random fields,” *In Proceedings of The 20th International Conference on Computational Linguistics (COLING)*, pp. 562–568, 2004.
14. D. Pinto, A. McCallum, X. Wei, and W. B. Croft, “Table extraction using conditional random fields,” *In Proceedings of the ACM SIGIR*, 2003.
15. T. Kudo, K. Yamamoto, and Y. Matsumoto, “Applying conditional random fields to japanese morphological analysis,” *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.
16. N. Okazaki, “Crfsuite: a fast implementation of conditional random fields (crfs),” 2007.

A. Breve Manual de Usuario

A.1. Pre-procesador de datos

Al abrir la aplicación, nos encontramos con la pantalla de la Figura 6. Como primer acción a realizar, nos permite seleccionar el *directorio de trabajo* que vamos a utilizar. En él se almacenan los resultados parciales y finales obtenidos por la aplicación, tales como archivos de *entrenamiento* y de *test* resultantes del pre-procesamiento, modelos y resultados obtenidos al testear los modelos, entre otros.

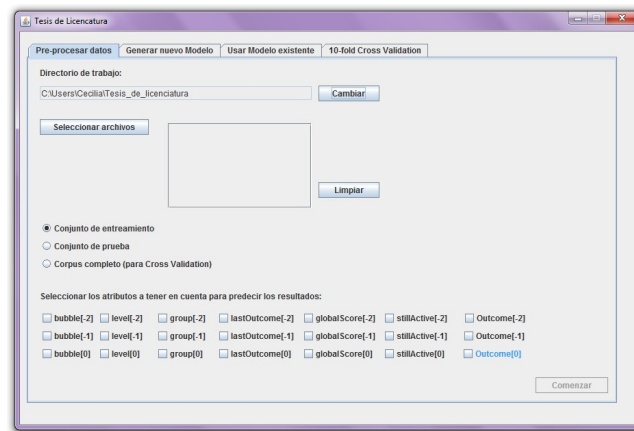


Fig. 6. Pre-procesador de datos

Luego de seleccionar el *directorio de trabajo*, se debe indicar cuáles serán los archivos a pre-procesar. Dado que KPM almacena un archivo de log por cada niño, se pueden seleccionar múltiples archivos ó directorios. Los archivos permitidos son aquellos que cuentan con extensión *.XML* y que respetan el formato establecido por KPM para el almacenamiento de logs. Ningún otro tipo de archivo es aceptado.

A continuación, se debe indicar qué tipo de pre-procesamiento realizar (Figura 7). Es decir, si los archivos seleccionados en el paso anterior serán utilizados para la generación o entrenamiento de un modelo, para testear un modelo, ó para realizar pruebas de tipo *Cross Validation* sobre el total de los datos. Por defecto, se encuentra seleccionada la opción *Conjunto de entrenamiento* (Ver: Convenciones, Sección A.5).

En caso de seleccionar la opción *conjunto de prueba*, la aplicación se fijará si en el directorio de trabajo existe un archivo de entrenamiento generado con anterioridad, dado que para poder testear un modelo, se deberán utilizar los mismos atributos que se utilizaron para su entrenamiento. En caso de que exista tal archivo, se cargarán automáticamente los atributos correspondiente. En caso contrario, se mostrará un mensaje de error al hacer click en el botón *Comenzar*.

Por último, pero no menos importante, restan seleccionar los atributos a tener en cuenta para el pre-procesamiento. Estos atributos son una especie de “template” o plantilla que se utiliza para describir *cada línea* de datos (burbuja jugada por un niño)

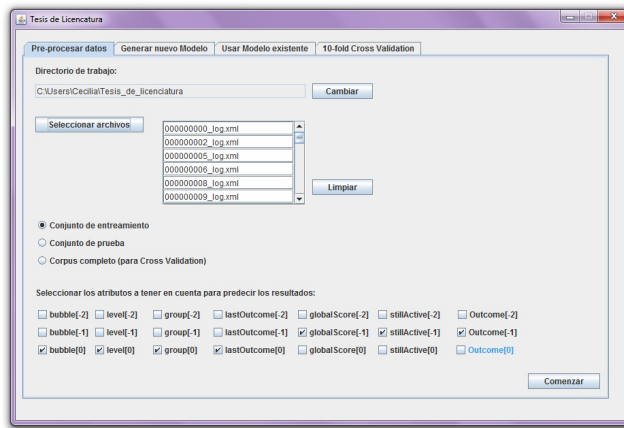


Fig. 7. Selección del tipo de pre-procesamiento: Train, Test o Cross Validation y elección de los atributos que caracterizan a una burbuja

de los archivos de log. La aplicación permite seleccionar un total de 21 atributos. En la Figura 7 se hallan seleccionados sólo 7.

Cada atributo se encuentra expresado de la siguiente manera: *nombre[índice]*, donde *nombre* indica el atributo al cuál estamos haciendo referencia, y el *índice* es un número entero que puede tomar los valores -2, -1 ó 0. De esta manera, con *nombre[0]*, estamos haciendo referencia a algún atributo de la línea que está siendo actualmente procesada del archivo. Con *nombre[-1]*, hacemos referencia a la línea anterior a la actual en el archivo. Y de la misma forma, con *nombre[-2]*, miramos el atributo especificado dos líneas hacia atrás.

Dado que el objetivo de la aplicación es predecir el Outcome de la burbuja actual (*Outcome[0]*), éste se encuentra coloreado de celeste y es deseable que el usuario de la aplicación *no* lo seleccione para tenerlo en cuenta en el template. Si este atributo es seleccionado, estaríamos diciendole de antemano el resultado que deseamos predecir, de modo que nuestro modelo no sería del todo correcto. De igual manera sucede con los atributos *globalScore[0]* y *stillActive[0]*, dado que ambos se calculan al finalizar la jugada de la burbuja actual, son datos que al momento de predecir una jugada, no estarán presentes.

Si la aplicación genera correctamente la salida esperada, se mostrará un mensaje de éxito, y los nuevos archivos estarán en el directorio de trabajo. En este ejemplo hemos generado sólo los datos de entrenamiento, de modo que en el directorio de trabajo figuran tres archivos. Estos pueden verse en la Figura 8, junto con una vista previa del archivo de entrenamiento *train.crf.txt* el cual se presenta con el formato esperado por *CRFSuite*.

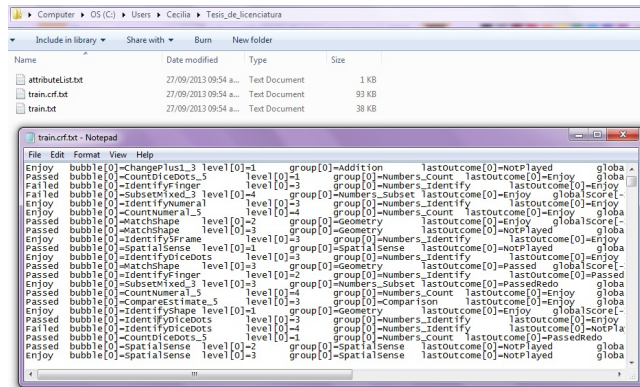


Fig. 8. Vista del directorio de trabajo luego de generar el archivo de entrenamiento

A.2. Generación de un nuevo Modelo

Para generar un nuevo modelo, basta con seleccionar un directorio de trabajo en el cual exista un archivo de entrenamiento (*train.crf.txt*). Esto es indispensable, dado que no tendría sentido generar un modelo sin datos.

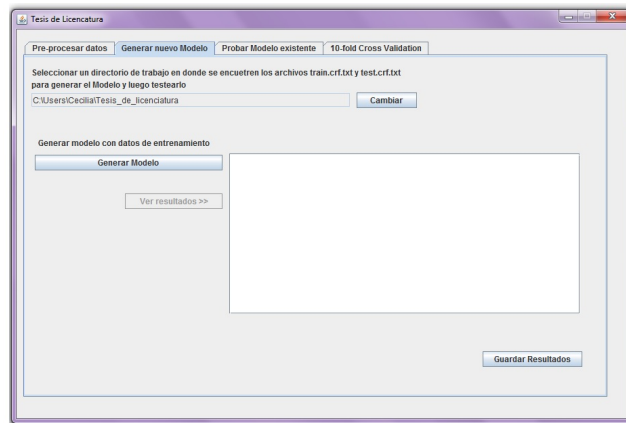


Fig. 9. Generación de un nuevo modelo

Una vez seleccionado el directorio de trabajo, se puede crear el modelo presionando el botón *Generar Modelo* de la Figura 9. Si no se halla un archivo de entrenamiento en el directorio especificado, se mostrará un mensaje informando lo sucedido.

Una vez que el *modelo* ha sido generado, puede visualizarse en el panel de la derecha, presionando *Ver resultados* (Figura 10).

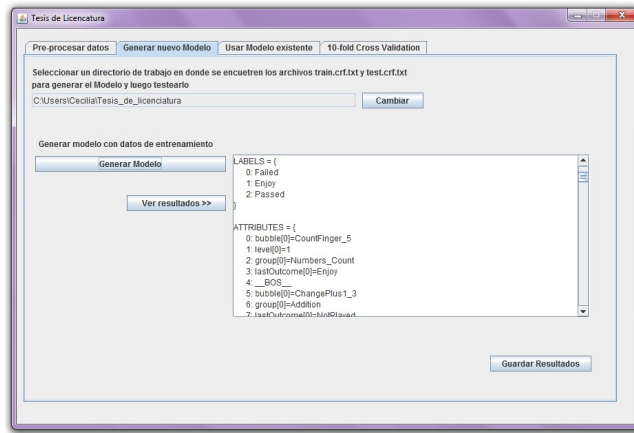


Fig. 10. Visualización de la versión legible de un modelo

A.3. Utilización de un Modelo existente

Una vez que se tiene el modelo, ya estamos en condiciones de comenzar a utilizarlo. Para ello usamos los datos que en un comienzo habíamos dejado de lado para utilizar como datos de test (20% del total de los datos recopilados). Estos datos deben respetar el formato esperado por *CRFSuite*. En la sección A.1 de este apéndice se explica cómo generar el archivo de test.

Para utilizar esta funcionalidad, en el directorio de trabajo debe existir un *modelo* (el cual deseamos poner a prueba) y un archivo de test (*test.crf.txt*) para realizar las pruebas correspondientes.

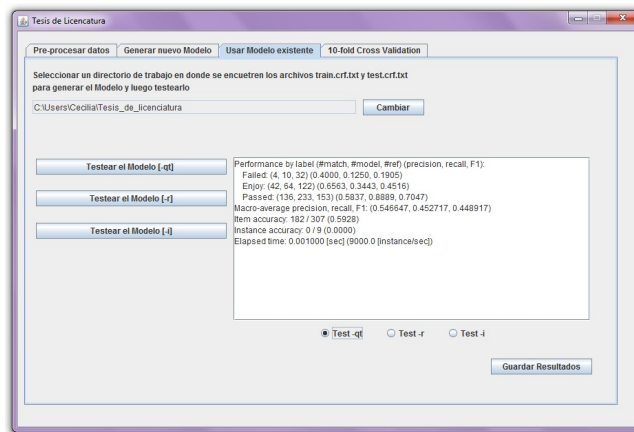


Fig. 11. Resultados obtenidos luego de testear el Modelo con la opción -qt de *CRFSuite*

Para estudiar el comportamiento de un modelo, *CRFSuite* provee tres alternativas, las cuales hacen uso del archivo *test.crf.txt* generado con anterioridad:

- Testear el modelo y obtener como salida resultados numéricos expresados en una matriz de confusión que involucra *Precisión*, *Exhaustividad o Cobertura*⁵ y *F1* de cada una de las etiquetas. Esta funcionalidad se logra presionando el botón *Testear el Modelo [-qt]*. Ver Figura 11.
- Testear el modelo y obtener como salida las etiquetas de referencia, paralelamente con las que predice el modelo, asumiendo que los datos de entrada (*test.crf.txt*) están etiquetados. Esta funcionalidad se logra presionando el botón *Testear el Modelo [-r]*. Ver Figura 12.
- Testear el modelo y obtener la probabilidad marginal de las etiquetas. Cuando se activa esta funcionalidad, cada etiqueta que se predice es seguida por “:x.xxxx”, donde “x.xxxx” representa la probabilidad de la etiqueta. Esta funcionalidad se logra presionando el botón *Testear el Modelo [-i]*. Ver Figura 13.

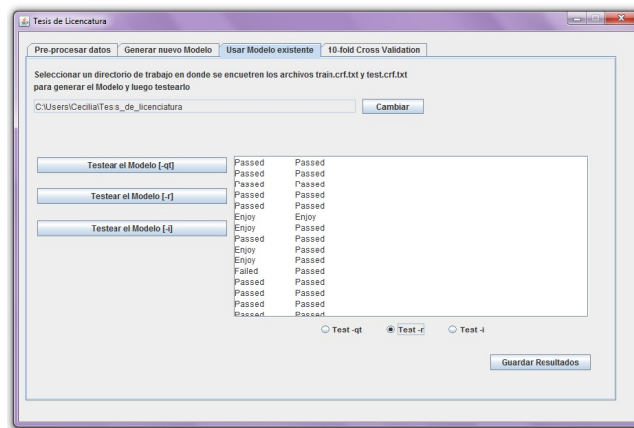


Fig. 12. Resultados obtenidos luego de testear el Modelo con la opción *-r* de *CRFSuite*

Los resultados pueden verse en el panel derecho, seleccionando la opción respectiva que se encuentra debajo del panel.

A.4. 10-fold Cross-Validation

Para realizar esta prueba, es necesario que nuestro directorio de trabajo contenga el archivo que resulta del pre-procesamiento del corpus completo (generado a partir de la primer pestaña de la aplicación, seleccionando la opción *Corpus Completo*).

Los resultados pueden verse en el panel derecho, haciendo click en el botón *Ver Resultados*.

⁵ Del inglés Recall

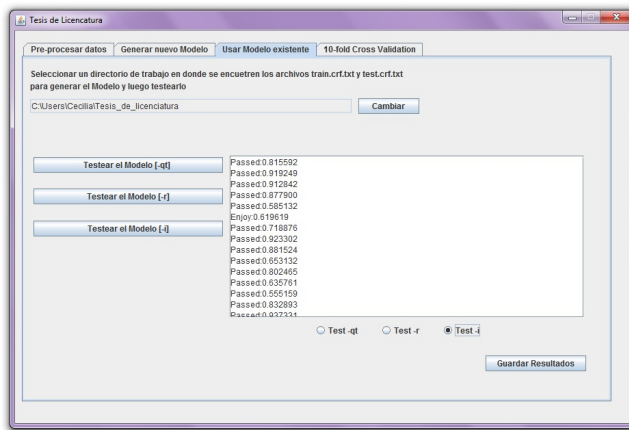


Fig. 13. Resultados obtenidos luego de testear el Modelo con la opción -i de *CRFSuite*

Cross-Validation (o Validación Cruzada) es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Para ello, lo que hace es tomar el conjunto completo de datos, y realiza rotaciones (en nuestro caso son 10) de dichos conjuntos. En estas rotaciones, el 80% de los datos son utilizados para entrenamiento, y el 20% para evaluación, y como resultado de las mismas, se obtiene un porcentaje de precisión parcial de cada una de ellas. El *promedio* de todas estas precisiones parciales, es un indicador de la precisión de un modelo basado en los atributos seleccionados con anterioridad. Este valor, se muestra en el campo “Precisión promedio” de la Figura 14.

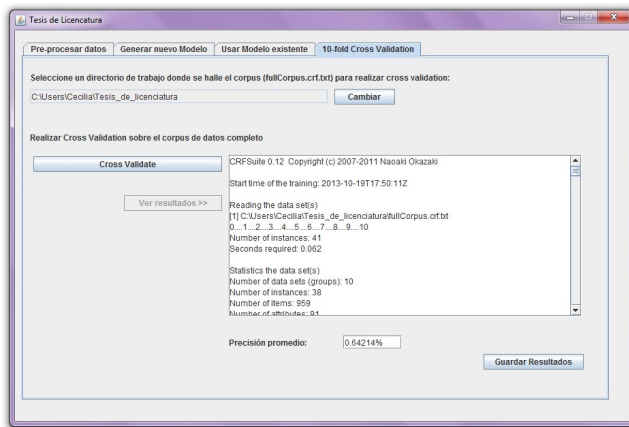


Fig. 14. Promedio de la precisión de cada una de las rotaciones de los conjuntos de datos

A.5. Convenciones

Las pestañas pueden utilizarse en cualquier orden, siempre y cuando se respeten los requerimientos para la ejecución de cada una en particular, esto es, que en el directorio de trabajo existan los archivos necesarios para realizar cada funcionalidad. De todos modos, si se viola alguno de los requerimientos mínimos, se mostrará el mensaje correspondiente.

Lo principal, es elegir un directorio de trabajo, o ser consciente de la utilización del directorio predeterminado. Allí es dónde se almacenarán los archivos intermedios del procesamiento, y de donde se leerán los datos de entrada al momento de entrenar o evaluar el Clasificador.

La formulación de los archivos que contienen los datos de Entrenamiento y/o Test debe realizarse sin excepción, a partir de la funcionalidad provista en la pantalla inicial (Figura 7), dado que se deben respetar los estándares de formato de *CRFSuite*, al igual que el nombre de estos archivos.

B. Implementación

Para la realización de la aplicación, el lenguaje de programación elegido ha sido *Java*, en conjunto con la herramienta para la predicción de etiquetas *CRFSuite*. Los datos de entrada a *CRFSuite* deben respetar un formato preestablecido, para lo cual se utilizaron scripts desarrollados en *Python*. Para la comunicación *Java-CRFSuite* se han usado scripts de tipo *Batch*, interpretados por la *Interfaz de Línea de Comandos* (cmd.exe) de *Windows*.

B.1. Arquitectura de la aplicación

Formato inicial Como primer paso hacia el desarrollo de la aplicación, se ha estudiado en profundidad el material sobre el cual trabajaríamos, es decir, el juego en sí, y los logs generados por el mismo. Estos logs son almacenados en documentos XML (un documento por cada niño) y son actualizados por el juego a medida que el niño va avanzando en el mismo. Estos documentos están compuestos por una serie de entidades, y las mismas son caracterizadas por atributos. Un ejemplo del formato de los logs puede verse en el Apéndice C.

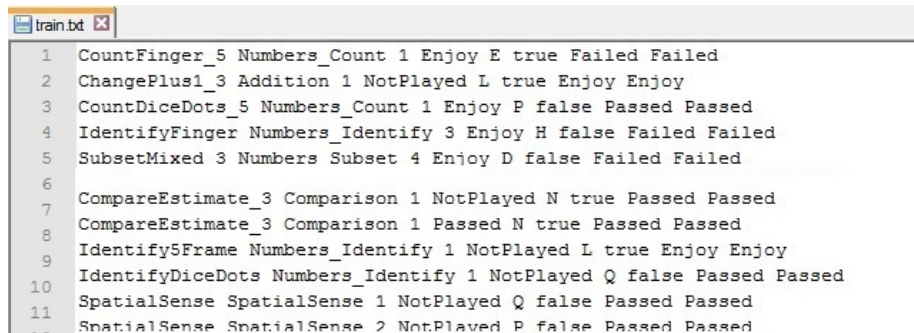
Preprocesador de datos (Parser) El objetivo principal de este módulo es la preparación de los datos que se procesarán en la aplicación, de modo que respeten el estándar esperado por la herramienta de clasificación *CRFSuite*.

Luego del formato inicial, explicado en la sección anterior, los archivos de log *temporales* respetan la estructura estándar de los documentos XML y están en condiciones de ser analizados con alguna herramienta existente. Para tal fin, en este trabajo hemos utilizado la librería *JDOM 2.0.5*⁶ para *Java*.

Gracias a las facilidades provistas por la librería *JDOM*, hemos implementado una suerte de parser, que procesa de a un documento XML a la vez, y extrae de los mismos la información que nos interesa. El parser recupera recursivamente los tags de

⁶ *JDOM 2.0.5* provee una solución completa basada en *Java* para acceder, manipular y generar datos XML desde código *Java*. URL: <http://www.jdom.org/>

un documento XML, a través de un recorrido en profundidad (del inglés Depth First Search o DFS). Por cada tag que se obtiene, se recuperan los valores de todos sus atributos y se los guarda en un único documento de texto, que contiene la concatenación de los atributos extraídos de todos los documentos XML seleccionados para el pre-procesamiento. Una vez concluida la etapa de parsing, el archivo de texto tendrá el aspecto que se aprecia en la Figura 15. En este caso, el nombre del archivo de texto dependerá de la selección hecha en la pantalla de la Figura 7 explicada en la Sección A.1, es decir, los nombres posibles para el archivo de texto serán: *train.txt*, *test.txt* o *fullCorpus.txt*.



```

1 CountFinger_5 Numbers_Count 1 Enjoy E true Failed Failed
2 ChangePlus1_3 Addition 1 NotPlayed L true Enjoy Enjoy
3 CountDiceDots_5 Numbers_Count 1 Enjoy P false Passed Passed
4 IdentifyFinger Numbers_Identify 3 Enjoy H false Failed Failed
5 SubsetMixed 3 Numbers Subset 4 Enjoy D false Failed Failed
6
7 CompareEstimate_3 Comparison 1 NotPlayed N true Passed Passed
8 CompareEstimate_3 Comparison 1 Passed N true Passed Passed
9 Identify5Frame Numbers_Identify 1 NotPlayed L true Enjoy Enjoy
10 IdentifyDiceDots Numbers_Identify 1 NotPlayed Q false Passed Passed
11 SpatialSense SpatialSense 1 NotPlayed Q false Passed Passed

```

Fig. 15. Aspecto del un archivo de texto *train.txt* luego de pre-procesar tres documentos XML (temp-000000000_log.xml, temp-000000001_log.xml y temp-000000002_log.xml) y extraer todos los atributos de los mismos

Módulo CRF Este módulo tiene el propósito de realizar todas las tareas que facilitan la utilización de la herramienta *CRFSuite*.

Creación de los scripts CMD: Estos scripts son creados con el fin de dar directivas a *CRFSuite*, las cuales son ejecutadas a través de la Interfaz de Línea de Comandos de *Windows*. Entre las directivas más usadas podemos encontrar las de entrenamiento, las de test y las de traducción de un modelo a código legible, entre otras. Por otro lado, los scripts CMD también son utilizados para invocar al intérprete de Python, el cual ejecuta los scripts que se describen en el párrafo siguiente.

Creación de los scripts en Python: El objetivo de estos scripts es el de ser utilizados como extractores de atributos. Una vez que el usuario selecciona los atributos que desea utilizar para crear un modelo, este módulo genera un script en *Python* que utiliza dichos atributos en forma de plantilla, la cual se va desplazando línea por línea sobre los datos de entrada, de manera de quedarse sólo con el valor de los atributos deseados. En el Apéndice C, sección de código C.2, presentamos un ejemplo del formato del script *chunking.py*, el cual se encarga de extraer siete atributos. Este script es ejecutado sobre los archivos generados durante la etapa de pre-procesamiento, es decir, que toma como entrada uno de los archivos *train.txt*, *test.txt* o *fullCorpus.txt* y emite como resultado *train.crf.txt*, *test.crf.txt* o *fullCorpus.crf.txt*, dependiendo de la funcionalidad seleccionada, explicadas en la Sección A.1.

Módulo Principal Es el cerebro de la aplicación. Se trata de una Clase escrita en *Java*, la cual se comporta como mediadora entre todas las demás clases. Es la encargada de delegar distintas funciones al resto de las clases, y es en ella dónde se encuentra el método `main()` (punto de entrada a la aplicación), que hace que se ejecute el programa. La primer tarea que realiza el método `main()` es la creación de la Ventana Principal.

Ventana Principal La ventana de la aplicación es una *Java Application Window*, que cuenta con diversas pestañas en donde se muestra la información que respecta a cada una de las funcionalidades. A través de la misma, el usuario final puede utilizar la aplicación.

Extras. Además de los módulos (ó clases) antes mencionadas, la aplicación utiliza algunas clases más, destinadas a procesamiento interno. Entre ellas: *Attribute*, *Group* y *Dialog*.

C. Código Interesante

C.1. Ejemplo de un archivo de Log

A continuación se puede ver una porción de un archivo de log:

```

1 <LOG>
2 <creationdate>Mon Feb 25 13:51:51 GMT-0700 2013</creationdate>
3 <UID>100000001</UID>
4 <FIRST_NAME>Michael</FIRST_NAME>
5 <LAST_NAME>Kipp</LAST_NAME>
6 </LOG>
7
8 <session startSessionTime="Mon Feb 25 13:51:51 GMT-0700 2013"/>
9
10 <bubble game_id="G4" bubbleName="CompareCorrespond_5" bubbleLevel="2" start="2013-02-25T13:52:13"
11   lastOutcome="Enjoy" currentLanguage="ENG" failedCounter="" enjoyCounter="" passCounter="" globalScore=
12   "-1.732" scoreToEnjoy="-2.2361" scoreToComplete="0.4472" bubbleDuration="90" accompanied="false">
13
14   <task success="true" taskScore="0.5774" globalScore="0.5774" time="57" timeFirstMove="57" goal="6"
15     badMoveCounter="0" numberOfObjects="2" numberOfGoal="1"/>
16
17   <task success="false" taskScore="-1.7321" globalScore="-1.1547" time="11" timeFirstMove="9" goal="6"
18     badMoveCounter="1" numberOfObjects="2" numberOfGoal="1">
19     <failed groupFailed="1" taskScore="-1.7321" time="9" goal="6" currentMove="4"/>
20   </task>
21
22   <task success="true" taskScore="0.5774" globalScore="-0.5773" time="6.25" timeFirstMove="6.25" goal="5"
23     badMoveCounter="0" numberOfObjects="2" numberOfGoal="1"/>
24   <task success="false" taskScore="-1.7321" globalScore="-2.3094" time="9.75" timeFirstMove="6.5" goal="6"
25     badMoveCounter="1" numberOfObjects="2" numberOfGoal="1">
26     <failed groupFailed="1" taskScore="-1.7321" time="6.5" goal="6" currentMove="4"/>
27   </task>
28   <task success="true" taskScore="0.5774" globalScore="-1.732" time="6" timeFirstMove="6" goal="5"
29     badMoveCounter="0" numberOfObjects="2" numberOfGoal="1"/>
30   <bubble_status_changed id="CompareCorrespond_5_2" NowIsActive="true" Outcome="Enjoy"/>
31 </bubble>
32 .
33 .
34 .

```

Fig. 16. Ejemplo de un archivo log.xml

El significado de cada uno de los atributos mencionados en el log es el siguiente:

- *creationdate*: Día de creación del log. Primer día en KPM, cuando el niño comienza a jugar.
- *UID*: Identificación unívoca de cada niño.
- *FIRST_NAME*: Nombre del niño.

- *LAST_NAME*: Apellido del niño.
- *startSessionTime*: Fecha y hora en que el niño inició una nueva sesión en KPM.
- *game_id*: Identifica el archivo Flash que se ejecutará. Rango: G1,G2,G3,G4. G1 corresponde a las tres primeras booths, G2 a la cuarta, G3 a la quinta y G4 corresponde a la sexta y la séptima booth (que están en la segunda página del juego).
- *bubbleName*: Cada instancia del juego o burbuja está definida por el nombre (*bubbleName*) y el nivel. Estos dos atributos definen la burbuja que ha sido abierta y que podrá ser *pasada*, *fallada* o *enjoyed*. Esta última se da cuando el niño no hace puntos suficientes como para pasarla, ni puntos por debajo de la puntuación mínima como para fallarla.
- *bubbleLevel*: Número que indica el nivel de la burbuja. Rango: [1, 8].
- *start*: Fecha y hora en que empieza a jugar la burbuja.
- *LastOutcome*: Resultado de la burbuja la última vez que se jugó. Rango: Enjoy, PassedRedo, Passed ó Failed. PassedRedo se obtiene cuando una burbuja es pasada luego de haber sido fallada.
- *currentLanguage*: Idioma seleccionado para la burbuja (Inglés o Español).
- *failedCounter*, *enjoyCounter*, *passCounter*: Atributos no utilizados actualmente en el juego.
- *globalScore*: Es el score que se va manteniendo para saber el *Outcome* al finalizar la burbuja, es decir, si la burbuja fue pasada, empatada (*enjoyed*) o fallada. Dado que el valor de este atributo es un número Real, para poder ser manipulado por las herramientas descriptas, lo hemos discretizado.
- *scoreToEnjoy*: Puntuación mínima para Enjoy. Una puntuación menor, hace que la burbuja sea fallada. Rango: [-3, 0].
- *scoreToComplete*: Rango: de 0 en adelante. Una puntuación mayor a cero, hace que la burbuja sea pasada.
- *bubbleDuration*: Tiempo que se tarda en terminar una burbuja. Las unidades de tiempo utilizadas son particulares de la implementación del juego.
- *accompanied*: Indica si el niño juega con compañía, o no. Posibles valores: True-False.
- *NowIsActive*: Atributo booleano cuyo valor se determina al finalizar la burbuja actual. En caso de ser pasada, la burbuja es desactivada (*NowIsActive* toma el valor false) y el *Outcome* es Passed. Además todos sus sucesores se activan, si es que tienen todos sus predecesores con estado 'Passed'. En caso de ser fallada, la burbuja es desactivada y el *Outcome* es Failed. Además todos sus predecesores se activan. Los sucesores de los predecesores de la burbuja se desactivan.
- *Outcome*: Atributo cuyo valor se determina al finalizar la burbuja actual. Es el resultado de la burbuja. Rango: Enjoy, PassedRedo, Passed o Failed.
- *id*: Atributo que corresponde al tag "bubble_status_changed". Es el resultado de concatenar *bubbleName_bubbleLevel*.
- *stillActive*: Atributo booleano, que será verdadero si luego de jugar a una burbuja específica, ésta seguirá activa. Falso en caso contrario. Este atributo no existe en el juego, sino que su valor es calculado a partir del *Outcome* de la burbuja.

Cada *task* o pregunta se almacena una vez que el niño ha respondido (correcta o incorrectamente) lo que se le ha planteado. Por lo general hay entre 4 y 8 *tasks* por burbuja. Los atributos de cada *task* son los siguientes: *Success*, *taskScore*, *globalScore*, *time*, *timefirstmove*, *goal*, *badMoveCounter*, *numberOfObjects*, *numberOfGoals*, *GroupFailed*, *CurrentMove*.

El tag *bubble_status_changed* lista las burbujas que se cambiaron de estado y de *Outcome* al finalizar la burbuja actual. En caso de ser empatada (*Enjoy*), la burbuja jugada sigue estando activa (*NowIsActive* continúa siendo true). En caso de ser pasada (*Passed*), la burbuja es desactivada (*NowIsActive* toma el valor false) y el *Outcome* es *Passed*. Además, todos sus sucesores se activan, si es que tienen todos sus predecesores con estado *Passed*. En caso de ser fallada (*Failed*), la burbuja es desactivada y el *Outcome* es *Failed*. Además todos sus predecesores se activan. Los sucesores de los predecesores de la burbuja se desactivan.

C.2. Ejemplo de un script en Python

A continuación se presenta en ejemplo de un script que permite extraer siete atributos para caracterizar cada línea del conjunto de datos: *bubble[0]*, *level[0]*, *group[0]*, *lastOutcome[0]*, *globalScore[-1]*, *stillActive[-1]* y *outcome[-1]*.

```

1  #!/usr/bin/env python
2  # Separator of field values.
3  separator = ' '
4  # Field names of the input data.
5  fields = 'bubble group level lastOutcome globalScore stillActive
6  outcome y'
7  # Attribute templates.
8  templates = (
9  (('bubble',0), ),
10 (('level',0), ),
11 (('group',0), ),
12 (('lastOutcome',0), ),
13 (('globalScore',-1), ),
14 (('stillActive',-1), ),
15 (('outcome',-1), ),
16 )
17
18 import crfutils
19
20 def feature_extractor(X):
21     # Apply attribute templates to obtain features (in fact, attributes)
22     crfutils.apply_templates(X, templates)
23     if X:
24         # Append BOS and EOS features manually
25         X[0]['F'].append('__BOS__') # BOS feature
26         X[-1]['F'].append('__EOS__') # EOS feature
27
28 if __name__ == '__main__':
29     crfutils.main(feature_extractor, fields=fields, sep=separator)

```

Fig. 17. *Chunking.py*, script extractor de siete atributos en Python