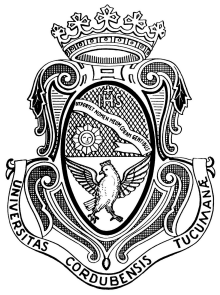
Universidad Nacional de Córdoba

Facultad de Matemática, Astronomía, Física y Computación



Trabajo final de la Licenciatura en Ciencias de la Computación

Modelado automático de trayectorias de aprendizaje: ¿Cuándo generar ayuda personalizada para principiantes en programación?

Estudiante: Marco, Moresi
Director: Lic. Marcos Javier Gómez
Profesora Representante: Dra. Luciana Benotti

CÓRDOBA ARGENTINA

Marzo 2018

Agradecimientos

A mis directores Marcos Gómez y Luciana Benotti, por su infinita paciencia y predisposición. Por brindarme sus perspectivas de las situaciones que fueron surgiendo a lo largo del trabajo y permitirme ver más allá de la problemática y seguir aprendiendo. Sin el apoyo de ellos este trabajo no hubiese sido posible. Además ellos lograron transmitirme la pasión por la investigación. Tarea en la cual quiero seguir desempeñandomé.

A Cecilia Martínez por su colaboración con sus aportes desde su mirada pedagógica de la problemática. Ella es mi mentora en esta área y no tengo más que agradecimientos para ella.

A Pedro D'Argenio y Raul Fervari por su colaboración en la presentación de este trabajo formando parte del tribunal evaluador.

A mi familia, por el apoyo brindado a lo largo de estos años de carrera.

A mis amigos de la facultad con quienes compartí muchos momentos a lo largo de estos años desde asados hasta domingos enteros entre mates y apuntes.

A mis amigos de toda la vida quienes fueron mi apoyo en todo lo que fui necesitando en estos años.

A mis amigos del club donde siempre encontré un momento de distensión pero aprendí el valor del trabajo y el esfuerzo que pude volcar en este trabajo.

A la facultad, a sus docentes y su personal no docente quienes ante cada necesidad siempre estuvieron predispuestos.

A la educación pública, qué me dió la oportunidad de ser hoy un profesional y poder seguir desarrollándome como persona.

Y a todas las personas con la que tuve la oportunidad de trabajar en distintas actividades, de ellos aprendí muchísimo y me permitieron convertirme en el profesional en el que me convierto hoy.

- **Clasificación de Biblioteca:**

  I.2.7

- **Palabras Clave:**

Resumen

Entender cuál es la trayectoria que realiza un estudiante durante la utilización de un sistema de enseñanza web permite tomar decisiones basadas en evidencia. Pero analizar estos datos a mano sería prohibitivamente costoso. El modelado automático de datos educativos con técnicas de aprendizaje automático prometen un mejor entendimiento del comportamiento y el aprendizaje por parte del estudiante. Para este trabajo se cuenta con un conjunto de datos provisto por Mumuki, un sistema de enseñanza online. Esta tesis propone una forma de modelar automáticamente cuándo un estudiante está en riesgo de abandonar un ejercicio y necesita ayuda personalizada. Para esto se realizó un análisis de cuáles son las características más significativas para modelar la trayectoria de los estudiantes comparando entornos de educación formal e informal. Se entrenaron diversos modelos de aprendizaje automático midiendo el desempeño en la tarea propuesta. Por último, con el objetivo de imitar la secuencialidad de la tarea de resolver un ejercicio de programación se entrenó una red neuronal recurrente en esta misma tarea. Luego de esto se realizó una comparación entre modelos para intentar determinar cuál es la mejor opción en esta tarea. El resultado de esta tesis es la comparación de un modelo neuronal sencillo con una propuesta concreta de ingeniería de características basada en teorías pedagógicas, para modelar automáticamente la trayectoria de estudiantes. Estos modelos fueron construidos en base a los datos generados en el sistema Mumuki. Se logró construir un modelo lineal, rápido de entrenar, con un desempeño similar al obtenido con un modelo más complejo como una red neuronal. Los modelos construidos son capaces de clasificar cada solución en tiempos realmente inferiores a los 20 milisegundos. Sería impensable que un humano consiga estos tiempos de clasificación. Haber conseguido clasificadores con tiempos de respuesta bajos permitirá que la integración al sistema Mumuki sea posible en tiempo real.

Abstract

Understanding which is the path that a student takes when using a web-based coding tool would allow a teacher to make decisions based on evidence. However, analyzing such data by hand would be prohibitively expensive, and the teacher feedback would probably arrive too late. The automatic modeling of educative data with machine learning techniques promises a better understanding of the behaviour and knowledge about student. For this work we have a dataset provided by Mumuki, an online teaching system. This thesis proposes a way to automatically model when a student is at risk of leaving an exercise and needs personalized help. Based on a pedagogical theoretical framework we automatically design and extract from the dataset representative features to model the path of students comparing formal and informal education environments. We trained several machine learning models comparing their performance in this task. Finally, with the aim of imitating the sequentiality of solving a programming exercise task, a recurrent neural network was trained in this same task. After this, a comparison between models was made to try to determine which is the best option for this task. The result of this thesis is a comparison between a simple sequential neural model and a concrete proposal of feature engineering for the task, in order to model automatically the path of the students. The models were built based on the data generated in the Mumuki system. It was possible to build a linear model, fast to train, with similar performance obtained with a more complex model such as a neural network.

Índice general

Capítulo 1

Introducción

1.1. Descripción de la Tesis

Poder detectar a tiempo cuando un estudiante está por abandonar un ejercicio de programación no es un desafío sencillo. En los últimos años diversos intentos fueron realizados para tratar de modelar este desafío, pero estos modelos son dependientes de ejercicios de programación particulares [35, 74, 15]. Esta tesis modela, de forma independiente de un ejercicio dado, cuándo un estudiante está en riesgo de abandono y necesita ayuda personalizada mediante el minado de datos de trayectorias de aprendizaje sobre el sistema educativo Mumuki.

A lo largo de esta tesis compararé diversos modelos de aprendizaje automático que sean capaces de detectar tempranamente abandonos de estudiantes en sistemas de enseñanza de programación. Puntualmente estaré trabajando sobre el sistema Mumuki [8]. Mumuki[1] es un sistema de soporte para la enseñanza de la programación. Es usado dentro del sistema formal de la enseñanza por diferentes universidades y colegios en todos los niveles. Además, Mumuki tiene una versión abierta que incluye miles de ejercicios en distintos lenguajes de programción para todos aquellos que deseen aprender a programar de forma autodidacta. Gracias a la colaboración de IKUMI SRL, la empresa que desarrolla Mumuki cuento con un conjunto de datos de más de un millón de soluciones enviadas por estudiantes en este sistema. Mumuki soporta una gran cantidad de lenguajes de programación y para cada uno de ellos tiene construido una serie de guías de ejercicios y para cada ejercicio tiene definido una cantidad de tests.

Consideremos la siguiente situación, un estudiante se encuentra trabajando en Mumuki, a medida que avanza sobre los ejercicios puede encontrarse con alguno que le resulte difícil, por diversas razones, y luego de varios intentos fallidos abandone el ejercicio. En esos casos contar con este sistema que detecte la posibilidad de abandono tempranamente, permitirá que al momento de ser detectado un posible abandono el sistema pueda tomar una decisión sobre qué acción hacer para evitar que el estudiante se frustre. Esta acción puede ser humana o automática. Por ejemplo, un docente puede ofrecer ayuda al estudiante o el sistema puede ofrecer bibliografía relevante al error cometido o sugerir un ejercicio distinto que ayude a repasar algún concepto. Para ello se analizará el conjunto de datos proveniente de Mumuki con el objetivo de encontrar características que nos permitan representar la trayectoria del estudiante. Para luego entrenar modelos de aprendizaje automático que permitan predecir un posible abandono antes de que ocurra. En breves palabras esta tesis propone cómo detectar automáticamente **cuándo** generar ayuda personalizada para principiantes en programación.

[1] https://mumuki.io/

1.2. Objetivo principal

El principal objetivo de este trabajo es comparar modelos de aprendizaje automático que sean capaces de predecir cuando un estudiante está por abandonar un ejercicio dentro de Mumuki. Con el fin de cumplir este objetivo se plantean las siguientes tareas a realizar.

- **Tarea 1:** Analizar la base de datos, conocer su granularidad de logeo, su estructura y la relación con las funcionalidades del sistema.
- **Tarea 2**: Crear un baseline para el clasificador a partir de un modelo regresión lineal y anotar automáticamente los datos mirando hacia "el futuro" dentro del conjunto de datos.
- **Tarea 3**: Diseñar y programar la extracción automática de características considerando dos dimensiones basadas en teorías pedagógicas: estudiante y ejercicio.
- **Tarea 4**: Comparar modelos con distintas características, decidir si la performance con respecto al baseline es estadísticamente significativa.
- **Tarea 5**: Comparar la performance obtenida por regresión lineal contra un modelo de aprendizaje neuronal profundo. Comparar la eficiencia temporal de los modelos.

1.3. Plan de Trabajo

A continuación se detalla cada una de las actividades a realizar para llevar a cabo las tareas enumeradas en la sección anterior.

Tarea 1: Como en toda tarea de aprendizaje automático es importante conocer con qué datos se está trabajando. Es la primera vez que se va a trabajar con datos recolectados del sistema Mumuki, por lo tanto es todo nuevo y no hay referencias sobre cómo utilizar los datos. Es un sistema en producción, no un prototipo con más de 70mil usuarios. Por ello se analizará en profundidad la distribución y características particulares de la base de datos. En particular se comparará su granularidad con otros sistemas de registro de trayectorias educativas. Se obtendrán estadísticas iniciales sobre frecuencia y regularidad de uso, uso dentro y fuera del sistema de educación formal, etc.

Tarea 2: Luego de haber completado la Tarea 1, se esperan conocer ciertas características del conjunto de datos. Lo que nos permitirá poder formalizar las definiciones de sesión activa, qué se considera abandono y qué tipos de abandonos existen. Además con estas definiciones formalizadas se podrá anotar el conjunto de datos, marcando cuáles de las soluciones pertenecen a un abandono. Para poder realizar pruebas y comparaciones definiremos un baseline a partir a un modelo de aprendizaje automático llamado regresión lineal [58], el cual es rápido de entrenar y por lo tanto poco costoso para ver el efecto que generan las características que se utilizan.

Tarea 3: Con el objetivo de construir un clasificador que detecte posibles abandonos, necesitaremos definir diferentes características, estas serán construidas a partir de la información de cada solución enviada. Para construir estas características nos interesa capturar dos dimensiones, estudiante y ejercicio, que hipotetizamos como influyentes a la hora de que un estudiante abandone el ejercicio. De acuerdo a Paper y Turkle [73] existen dos tipos de estudiantes según su comportamiento, Tinkerers y Planners las características que construiremos estarán construidas a partir de las particularidades que presentan estos tipos de estudiantes.

Dimensión Estudiante:

- Nivel de Experiencia: Intentaremos capturar la experiencia en programación del estudiante. Utilizamos las soluciones enviadas previamente dentro del sistema para otros ejercicios resueltos por ese estudiante. Distinguimos ejercicios completados satisfactoriamente de aquellos incorrectos.
- Nivel de insistencia: Este nivel tiene como objetivo analizar el comportamiento del estudiante dentro del sistema midiendo distancia en tiempo entre soluciones consecutivas, distancia de Levenshtein entre soluciones consecutivas, entre otras.
- Nivel de abandonos: Esta característica será la encargada de representar si es un estudiante que abandona pronto y/o frecuentemente los ejercicios.

Dimensión Ejercicio:

- Nivel dificultad: En este característica intentaremos representar que tan difícil es el ejercicio a partir del concepto que trabaja y la performance de los estudiantes en el mismo considerando el promedio de la cantidad de soluciones enviadas para lograr resolver ese ejercicio para todos los estudiantes, la cantidad de abandonos para el ejercicio, entre otras.
- Nivel conceptual: cada ejercicio presente en el sistema Mumuki trabaja sobre un concepto puntual, para poder representar esto utilizaremos el identificado único que tiene cada ejercicio dentro del sistema como una característica para los modelos.

Tarea 4: La tarea que se llevará a cabo será probar cada uno de estas características y evaluar el impacto que genera en el clasificador. Luego se realizarán combinaciones de las mismas para obtener el mejor clasificador posible con estas características. Para medir el desempeño de cada uno de estas características en el clasificador utilizaremos la métrica F1, la cual considera tanto la precisión con el recall del clasificador. A precisión lo podemos definir como el número de ejemplos clasificados positivos y que son correctos divididos sobre todos los ejemplos clasificados. Mientras que recall lo definimos como el número de ejemplos clasificados positivos sobre todos los ejemplos relevantes, es decir todos los que debían haber sido marcados positivos.

Tarea 5: Luego de haber identificado cuáles son las características que mejoran el desempeño del clasificador. Se utilizará esta información para entrenar una red profunda, para probar cuál es el desempeño que se obtiene para esta tarea.

1.4. Relevancia de la temática

En la actualidad diversos tipos de sistemas de enseñanza de la programación son utilizados tanto en el sistema formal de educación como de modo autodidacta por los estudiantes. Ninguno de esos sistemas es capaz de detectar la posibilidad de abandono por parte del estudiante, contar con esta característica dentro del sistema permitirá seguir avanzando en el objetivo general que comparte la empresa creadora de Mumuki y los diferentes vínculos que motivaron esta tesis. Poder predecir cuándo un estudiante está frustrado al intentar resolver un ejercicio, permitirá sugerir que resuelva otro ejercicio construyendo así una trayectoria adaptada para cada tipo de estudiante.

1.5. Links que motivan este trabajo

La principal motivación para este trabajo fue el desafío que supone trabajar con un conjunto de datos no explorado, sobre el cual no se ha realizado ninguna tarea previa. Además del desafío de construir una característica innovadora para un sistema que es utilizado actualmente. Trabajar sobre algo que tiene impacto y aplicación directa fue otro de los motivantes principales de este trabajo. El contacto con Bulgarelli Franco, CEO de Mumuki, es fundamental para este trabajo ya que él es uno de los primeros diseñadores y programados de Mumuki y quien nos facilitó el dataset con el cual trabajar. Más aún es uno de los interesados en este trabajo y quien planteó la necesidad de detectar cuándo un estudiante necesita ayuda como un objetivo general para Mumuki. El trabajo realizado por Benotti Luciana en una de su publicación [9] realizado en conjunto con investigadores de la Universidad de Stanford y Google. Es de vital importancia para este trabajo ya que brinda diversos lineamientos sobre cómo realizar los experimentos. Otro de los motivos por los que surgue este trabajo es una problemática reportada por Martinez Cecilia et al en [54], que es la falta de docentes capacitados y lo difícil que resulta capacitarlos en el área. Parte de la motivación de este trabajo es fruto de las diversas actividades realizadas por el equipo de extensión UNC++ a cargo de Benotti Luciana y Martinez Cecilia. Contar con la experiencia de campo sobre cómo utilizan los estudiantes el sistema Mumuki es muy importante para este trabajo y eso es gracias al vínculo con el estudiante de doctorado Marcos Gómez, el director de esta tesis. Esta tesis también sigue muchas recomendaciones realizadas por el trabajo realizado por Georgiana Haldeman [35] quien es estudiante de doctorado dirigida por Thu Nguyen en el departamento de Ciencias de la Computación en la Universidad de Rutgers.

1.6. Estructura de la Tesis

Concluida la introducción en este capítulo, a continuación se detalla la estructura de los siguientes capítulos. En el *Capítulo 2* se desarrollarán las diferentes motivaciones y problemáticas que dieron lugar a este trabajo, las posibles soluciones y las herramientas que están disponibles hoy. Discutiremos sobre las ventajas de los diversos sistemas tanto cerrados como abiertos. Mumuki, el sistema que será soporte de este trabajo será descrito a lo largo del *Capítulo 3*. Luego en el *Capítulo 4* hablaremos sobre el estado actual del área, daremos una categorización de los trabajos relacionados, además mostraremos algunas de las diferentes aplicaciones sobre sistemas de enseñanza. A lo largo del *Capítulo 5* se presentará el desarrollo de los modelos de aprendizaje automático, se propondrán diversas características en base a lo analizado en el capítulo previo y se sentaran los lineamientos para los experimentos a realizar. En el *Capítulo 6* discutiremos los resultados obtenidos y por último en el *Capítulo 7* presentaremos nuestras conclusiones y hablaremos sobre el trabajo futuro que da pie este trabajo.

Capítulo 2

Sistemas de corrección automática para la enseñanza de programación

En este capítulo se describe la problemática por la que atraviesa hoy la enseñanza de la Ciencias de la Computación (CC) con el objetivo de entender qué motiva la investigación en sistemas de soporte a la enseñanza de programación realizada en esta tesis. En la Sección 2.1 se detalla la situación actual de la enseñanza de la programación. Luego, en la Sección 2.2 se describe un estudio que investiga cómo los docentes aprenden a enseñar programación y qué necesitan de un sistema de soporte a la enseñanza. Por último en la Sección 2.3 se comparan dos tipos de sistemas que se utilizan para enseñanza de programación: abiertos y cerrados.

2.1. ¿Por qué se necesitan estos sistemas?

A lo largo de los años la concepción de cómo enseñar ciencias de la computación ha ido mutando conforme a diversos intereses e ideas de cómo se debe enseñar. De acuerdo con Levis [49] se pueden identificar tres distintas concepciones socioeducativas de la enseñanza y el aprendizaje de las tecnologías de la información y la comunicación (TIC) en general y en las CC en particular:

Concepción Técnico-operativo: la enseñanza y el aprendizaje se restringen a la dimensión operativa de los medios informáticos. Renueva la tradición que entiende que la escuela debe enseñar a usar la computadora. Defiende la necesidad de formar a los estudiantes en la operación de equipos y programas informáticos de uso corriente en el ámbito laboral, sin tener en cuenta la obsolescencia de estos conocimientos dado la constante evolución de equipos y aplicaciones.

Concepción Instrumental-integradora: promueve la utilización de las TIC como recurso didáctico para facilitar la enseñanza y el aprendizaje en todas las disciplinas. Propone que las computadoras y redes deben ser utilizadas para desarrollar prácticas pedagógicas innovadoras.

Concepción Tecno-lingüística: Se plantea la necesidad de enseñar los principios de los lenguaje que regula el funcionamiento de las computadoras como son los lenguajes de programación.

Durante el final de los años 70 las escuelas de Argentina entendían que la enseñanza de la Ciencias de la Computación consistía en enseñar programación con herramientas como Logo [60]

o Basic [47] basando su enseñanza en la concepción tecno-lingüística. Para la década de los 90 esto se modificó, se tornó la concepción a una forma técnico-operativo y se empezó a formar a los alumnos en el uso de herramientas de oficina como procesadores de texto, hojas de cálculo, etcétera con el objetivo que los estudiantes estuviesen capacitados para los trabajos nuevos. Con esta modalidad de enseñanza se generaba en los estudiantes una percepción de la computadora como una simple herramienta de oficina. Para el año 2000 esta situación mutó, virando la perspectiva a una concepción instrumental-integradora. A principios de esta década se volvió a una concepción tecno-lingüística.

Como consecuencia de esta reciente vuelta a un enfoque tecno-lingüista, en los últimos años la demanda de aprender a programar en el país ha crecido mucho más que la oferta de docentes capacitados. Esta realidad no es única a nuestro país. Del mismo modo ocurrió en Francia [6], en Nueva Zelanda [7], en Estados Unidos [40], entre otros países. La iniciativa de enseñar y aprender a programar está fuertemente incentivada a través de millonarias inversiones públicas e innumerables iniciativas en países como Estados Unidos, Inglaterra, Alemania, y también, en Argentina con el proyecto Program.ar[1] del Ministerio de Ciencia y Tecnología de la Nación. Todos estos países comparten un desafío: la falta de docentes capacitados para enseñar a programar. Los conocimientos con respecto a las secuenciación de contenidos y la didáctica para ser aplicada en la escuela está tratando de recuperarse de las décadas de inactividad debido a los enfoques técnicos operativos e instrumentales. Por lo tanto, los recursos didácticos existentes no terminan de satisfacer las necesidades de los docentes y de los estudiantes.

A medida que los diferentes países han puesto el esfuerzo en introducir Ciencias de la Computación y Programación en la currícula obligatoria en los diferentes niveles, un tema de debate entre el sector académico, responsables políticos y toda la comunidad educativa es: ¿Quiénes van a enseñar programación en las escuelas? Y, ¿Cómo serán capacitados los docentes que estarán a cargo de esas aulas?. Actualmente uno de los problemas más grandes que presenta la tarea de enseñar programación es la falta de conocimientos específicos de los docentes con respecto a los contenidos relacionados con CC [48]. Mientas que la mayoría de investigadores y responsables políticos concuerdan que los títulos de grado ofrecidos por las universidades son la mejor opción para preparar docentes altamente calificados [64], este enfoque no escala a la demanda actual del sistema educativo.

Como consecuencia muchos países capacitan a sus profesores en actividad. Por ejemplo el Reino Unido, Nueva Zelanda [72], Estados Unidos [33, 29] y Alemania [57]. Estos países enfrentan el dilema de seguir adelante con su reforma curricular de CC con escasez de profesionales de CC dispuestos a enseñar. Por lo tanto, optan por formar docentes que ya están en actividad.

Trabajo previo mostró que experiencias cortas de formación en servicio no son suficientes para capacitar docentes con un conocimiento suficiente para la enseñanza de programación. [29]. La próxima sección describe trabajo previo que explora cómo aprenden los docentes y qué tipo de ayuda necesitan al momento de poder aprender a enseñar programación.

2.2. ¿Qué se necesita de ellos?

Los resultados principales reportados en esta sección sobre cómo aprenden los docentes fueron publicados en nuestro artículo [54]. Estos resultados, que provienen de la implementación de un curso de capacitación, constituyen las razones de la elección del sistema para el soporte de la enseñanza de la programación que se presenta en el Capítulo 3 y la propuesta de minado de los datos educativos, descritos en los Capítulos 4 y 5

[1] http://program.ar/

En la Sección 2.2.1 describimos el curso de capacitación, su diseño e implementación. Luego en la Sección 2.2.2 se presentan los instrumentos de recolección de información que fueron diseñados y utilizados en la experiencia. Por último en la siguiente Sección 2.2.3 se discuten algunos de los hallazgos obtenidos a lo largo de esta experiencia.

2.2.1. Diseño e implementación de la capacitación

En los años 2014 y 2015 se realizaron cursos de formación docentes para la enseñanza de programación en la provincia de Córdoba. En total participaron 106 docentes de nivel inicial, primario, secundario y terciario, de distintas áreas de enseñanza, tanto de escuelas públicas como privadas. La convocatoria para poder inscribirse al curso de formación fue abierta, sin importar nivel o área de enseñanza. Se le solicitó a los profesores que se registren de a pares dentro del mismo colegio para reducir el aislamiento de los profesores a la hora de implementar las innovaciones dentro del aula y la consecuente deserción asociada [54].

El curso duró 50 horas en total, distribuidas a lo largo de un semestre. 40 de esas horas fueron impartidas en la Facultad de Matemática, Astronomía, Física y Computación (FAMAF) de la Universidad Nacional de Córdoba (UNC), distribuidas en encuentros presenciales de entre 4 y 6 horas, las 10 horas restantes fueron completadas por los docentes realizando prácticas en sus escuelas. Durante estas prácticas los docentes debían enseñar en sus aulas conceptos de ciencias de la computación trabajados durante las clases presenciales. El dictado de las clases presenciales en la universidad estuvo a cargo de docentes de la UNC. Estudiantes avanzados de la Licenciatura en Ciencias de la Computación de FAMAF trabajaron como tutores. Su tarea fue acompañar en las diferentes actividades propuestas durante las 40 horas que duró el curso. Además brindaban 4 horas de acompañamiento presencial en la escuela y asistencia dentro de la puesta en práctica de los docentes en sus aulas. También ayudaron con la elección de las herramientas de software a usar en el aula y de las estrategias para profundizar sobre la enseñanza de los conceptos del curso.

El curso consistió en 3 módulos: Robótica, Chatbot y Animaciones, con el principal objetivo de enseñar los principales conceptos de programación como secuencia, evento, condicional, ciclo, variable, métodos, parámetros, entre otros. Se promovió el uso de diferentes sistemas diseñados para enseñar programación como lo son Alice [21], Code.org [40], la herramienta de código abierto Chatbot la cual permite programar un bot que responde un chat automáticamente [13, 12] y UNCDuino [11], la cual es un sistema multilenguaje para la programación de robots construidos en base a la placa Arduino [5]. Cada vez que se trabajaba sobre un sistema nuevo se retomaban los mismos conceptos de CC, haciendo foco en aprender CC desde el aspecto conceptual, con el objetivo que los docentes puedan abstraer los conceptos sin importar qué sistema utilicen, para que no dependan de un sistema en particular.

La siguiente subsección describe los instrumentos de recolección de información que fueron diseñados y utilizados en el curso.

2.2.2. Instrumentos de recolección y datos obtenidos

Como se mencionó en el inicio de esta sección, la convocatoria era abierta para todos los niveles educativos y sin restricción de espacio curricular en la que se desempeñaba el profesor. A la hora de matricularse en el curso los docentes debían completar cierta información personal, en la Tabla 2.1 se muestra la distribución de los docentes según el tipo de escuela donde trabaja y el nivel.

La distribución de los docentes que formaron parte del curso de formación nos muestra que

Perfil del docente inscriptos	**%**
Profesor de escuela Pública	75 %
Profesor de escuela Privada	25 %
Profesor de escuela Secundaria	60 %
Profesor de escuela Primaria	20 %
Profesor de escuela Terciaria	20 %

Tabla 2.1: Distribución de la matrícula de docentes en los cursos según el sector, público o privado, y según el nivel de enseñanza, primaria, secundaria y terciaria (n=106)

el interés por aprender la temática proviene de diversas áreas no exclusivamente de docentes relacionados con las CC. Por lo tanto, el grupo de docentes matriculados era heterogéneo y con un 78 % de docentes sin conocimientos previos formales en CC. Con el objetivo de entender cómo aprenden los docentes con diferentes experiencias previas en CC, se documentó qué estrategias y conceptos eligen a la hora de hacer sus prácticas. Para ello se diseñaron tres cuestionarios que los profesores completaron.

Además de contar con esta información los tutores que acompañaron a los docentes en sus puestas en práctica dentro del aula, escribieron una reseña semi estructurada en base a la observación de cada clase a la que asistieron. Los cuestionarios respondidos por los docentes fueron analizados y se fueron etiquetando los temas emergentes que aparecieron en las respuestas. Validar las respuestas de los profesores contra las observaciones realizadas por los tutores contribuyó a la validez del estudio y permitió confeccionar la Tabla 2.2 la cual contiene los temas emergentes sobre la experiencia de los profesores. En la siguiente sección discutiremos los resultados obtenidos a partir de los datos extraídos con estos instrumentos.

2.2.3. Discusión sobre resultados

A continuación se discuten los hallazgos obtenidos a partir de analizar las respuestas de los profesores a los diferentes cuestionarios. En ese análisis se identificó que el 75 % de las clases realizadas por los profesores en el aula intentaron incorporar y explicar los conceptos fundamentales de CC aprendidos a lo largo del curso. Analizando cada observación realizada por los tutores se identificaron diferentes niveles de explicaciones en la enseñanza de conceptos de CC. En la Tabla 2.2 se observa la proporción según el nivel.

Profesor no puede explicar el concepto	5 %
Profesor explica el concepto de memoria	3 %
Profesor explica el concepto correctamente pero comete errores	17 %
Profesor explica el concepto correctamente pero superficialmente	35 %
Profesor explica el concepto correctamente usando analogías y ejemplos	23 %
Otros	17 %

Tabla 2.2: Explicación de los profesores de conceptos de CC

58 % de los profesores pudieron explicar conceptos de CC correctamente. Los conceptos más frecuentes que se enseñaron fueron conceptos de programación como condicional, ciclo, variables, secuencia, método, entre otros. Pero solo un 23 % pudo explicar los conceptos en profundidad usando analogías, ejemplos y situaciones practicas. Observamos también, a partir del relevamiento realizado por los tutores, una fuerte correlación entre los docentes con formación previa

en CC y su desempeño en el aula. Concluimos que el diseño y la duración de este curso no fue suficiente para para preparar a los docentes sin conocimientos previos en CC ya que, la mayoría abordaron de manera deficiente o superficial los conceptos de CC en sus clases. Mientras que observamos que los profesores con experiencia previa en CC fueron mejores explicando conceptos de CC.

Idealmente los docentes que enseñen CC en estos niveles educativos deberían recibir una formación formal de larga duración. Sin embargo, en Argentina y en el mundo se está intentando empezar a enseñar CC en todos los niveles educativos, comenzando en nivel inicial y no se cuenta con docentes capacitados. Dada la alta demanda por recursos humanos capacitados en CC no sólo desde el sector educativo sino también del industrial, este problema no parece factible de ser solucionado en el corto plazo.

Esta tesis propone aliviar este problema mediante sistemas de soporte a la enseñanza para que profesores sin conocimientos previos en CC pueden recibir asistencia al llevar la programación al aula. Dada la falta de conocimientos profundos en el área se les dificulta diseñar posibles secuencias didácticas para sus clases, por ello creemos que una de las opciones posibles es que esos profesores implementen sus clases mediante un sistema que provea una secuenciación de contenidos con ejercicios ya establecidos, de este modo el docente no se tendría que preocupar por qué conceptos enseñar y en qué orden. Es importante destacar que si el docente no tiene conocimientos profundos en la temática difícilmente pueda realizar un seguimiento en tiempo de todos sus estudiantes, por lo que es importante que el sistema que acompañe al docente en el aula cuente con un sistema de evaluación automática para los ejercicios. Para aquellos docentes que tengan conocimientos en la temática este tipo de sistemas les permitiría hacer foco en los estudiantes que más lo necesiten, al ya tener solucionada la secuenciación de contenidos y la evaluación de las entregas, puede por ejemplo mejorar el seguimiento de sus alumnos en cursos de gran concurrencia, o en cursos donde los niveles de conocimientos previos sean muy dispares, cosa que ocurre actualmente en las aulas, especialmente en las terciarias y las universitarias.

A continuación, en la Sección 2.3, analizaremos brevemente los tipos de sistemas actualmente disponibles para enseñanza de CC.

2.3. Tipos de sistemas de enseñanza de programación

Como describimos en las Seccion 2.1 la enseñanza de la programación comienza a ser una tendencia en todos los niveles educativos. El problema es la falta de docentes con formación específica en el área específica de programación. Y en este punto nos encontramos en un problema. Las instituciones educativas y políticas de estado quieren implementar la enseñanza de programación de las escuelas a corto plazo, lo cual no se condice con los tiempos necesarios para poder capacitar a nuevos docentes en el área. Ante esto se desarrollaron diferentes propuestas de reconversión docente con cursos de formación docente para docentes en actividad. Como vimos en la Sección 2.2, no hay buenos resultados en la incorporación conceptual en cursos cortos de capacitación. Para poder fomentar la enseñanza de programación en diversos niveles existen hoy numerosos sistemas para el soporte de la enseñanza de programación [17]. A partir de la revisión bibliográfica podemos encontrar dos grandes ejes: sistemas cerrados y sistemas abiertos. A continuación se mencionan ciertas características de estos sistemas que son importantes considerar a la hora de elegir alguno para llevar al aula.

En los sistemas abiertos, la posibilidad de crear ejercicios es ilimitada, solo depende de la imaginación y creatividad del docente, permitiendo trabajar una gran cantidad de conceptos de CC. Habitualmente este tipo de sistemas no provee ejercicios predeterminados para trabajar.

Como no tiene una guía de ejercicios para seguir, se hace más difícil que profesores sin experiencia previa, o buen manejo de conceptos de CC, puedan utilizarlas porque ellos son quienes deben definir las actividades. Más aún la infinita posibilidad de crear un ejercicio pone al docente en una decisión difícil de tomar como es: ¿Qué conceptos debo enseñar?, ¿En qué orden debe realizarse la secuenciación de contenidos?, ¿Cuál es una buena forma de enseñar un concepto en particular?. Otro aspecto no menor a mencionar, es que si el docente copia ejercicios de internet, lo que ocurre frecuentemente, en un sistema abierto se encuentra con el problema de evaluar las soluciones realizadas por sus alumnos. Habitualmente este tipo de sistemas abiertos no cuentan con un sistema de corrección automática que chequee que lo propuesto en la actividad se esté cumpliendo o no, por lo tanto el mismo docente es quien debe tener la capacidad de poder leer el código realizado por los alumnos y poder corregirlo. Esta tarea puede ser complicada para profesores sin formación previa. En el caso que el sistema abierto no cuente con un sistema de corrección automática, esta tarea debe realizarse de manera manual, lo cual hace que sea tedioso cuando la cantidad de alumnos es grande, además es importante que el docente cuente con conocimientos lo suficientemente profundos como para corregir las diversas posibles formas de solucionar los ejercicios propuestos.

En cambio un sistema cerrado cuenta con ejercicios predeterminados diseñados de tal manera que existe una secuenciación predefinida en los contenidos, donde cada ejercicio puede trabajar sobre un concepto fundamental de CC o combinar varios. Esto hace que el docente que desee utilizarla no deba preocuparse sobre ¿Qué conceptos debe enseñar?, ¿En qué orden? O, ¿de qué forma estructurar los ejercicios y contenidos?. A diferencia de los sistemas abiertos, donde el docente es quien tiene que corregir los ejercicios, los sistemas cerrados suelen incluir sistemas de corrección de ejercicios, los cuales permiten evaluar las soluciones enviadas por los estudiantes y corregirlos automáticamente. Pudiendo el sistema por sí solo definir si la solución enviada es correcta o no y determinar si cumple con ciertos lineamientos que el diseñador del ejercicio puede requerir necesarias en las soluciones, todo de modo automático.

Más aún, como este tipo de plataformas tienen ejercicios pre-construidos por lo tanto permite luego de evaluar si la solución es correcta o no y generar feedback automático. Una característica importante del feedback automático generado por la plataforma es que el estudiante no se siente inhibido por la presencia del profesor, esto se reporta en entrevistas realizadas a estudiantes en [8]. Habitualmente los sistemas cerrados proveen una secuenciación de contenidos de modo incremental, por lo tanto el profesor no debe preocuparse por generar los contenidos sino que el rol del docente se cambia por un rol el cual consiste en acompañar el aprendizaje de los alumnos. De este modo no es fundamental que el docente tenga formación específica en CC de gran profundidad para llevar a cabo actividades en el aula. Contar con la característica de los sistemas cerrados que pueden corregir las soluciones enviadas automáticamente es importante ya que si el sistema se encarga de almacenar las soluciones enviadas junto con su correspondiente correcciones permitirá al docente realizar un seguimiento personalizado y la reconstrucción del trayecto de cada estudiante en su curso. Por otro lado, si el sistema es capaz de reconstruir la trayectoria de los alumnos dentro del sistema, el sistema podría ser capaz de generar feedback a los alumnos en base a su trayectoria. Esta característica le permitirá al docente el manejo de aulas heterogéneas donde los estudiantes, en sus primeras experiencias dentro del aula, suelen presentar una gran diversidad de experiencias previas, desde quienes es realmente su primer experiencia programando hasta quienes ya la programación es algo habitual. La detección automática de cuando una secuenciación no es apropiada para un estudiante como se modela en esta tesis permitiría que un sistema cerrado se adapte a distintos tipos de estudiantes.

En la actualidad se dispone de una gran variedad de sistemas abiertos como lo son los que

se utilizaron en la capacitación docente que se describe en la Subsección 2.2.1: Alice[2], UNC Duino [11], Scratch [52], entre otros . A partir de esta experiencia creemos que es importante contar con sistemas cerrados que faciliten la tarea de los docentes. La oferta de herramientas cerradas no es tan variada, entre ellas se encuentran Coursera[3], que no logea a un nivel de granularidad apropiado para nuestros objetivos y CodeAcademy[4], que ofrece la solución correcta al estudiante luego de un número de soluciones fallidas, entre otras.

Mumuki es el sistema cerrado que elegimos para realizar la investigación y desarrollo en este trabajo, en en el Capitulo 3 se describe la plataforma en detalle con todas sus características tanto desde la perspectiva del docente como la del estudiante. Además se presenta el conjunto de datos con el que se cuenta para este trabajo. Luego en el Capitulo 4 discutiremos sobre el aporte que brinda la mineria de datos sobre este tipos de sistemas cerrados.

[2]https://www.alice.org/
[3]https://www.coursera.org
[4]https://www.codecademy.com

Capítulo 3

Mumuki: Un sistema que registra trayectorias de aprendizaje

En este capítulo se describe el sistema online Mumuki para aprender a programar y se describe un análisis preliminar del conjunto de datos. Los datos usados para desarrollar el trabajo descripto en esta tesis se obtuvieron de estudiantes interactuando con Mumuki. Los usuarios de Mumuki se pueden clasificar en aquellos que tienen permisos para resolver ejercicios, a los que llamaremos **estudiantes**. En la Sección 3.1 se detalla la plataforma desde el punto de vista de este tipo de usuario. Por otro lado se encuentran los usuarios con permisos docentes, es decir aquellos que pueden crear cursos, definir ejercicios de programación, entre otras cosas. De ahora en más llamaremos **docentes** a estos usuarios y en la Sección 3.2 se detalla su perspectiva. Por último en la Sección 3.3 se describe el conjunto de datos registrados por Mumuki que se usan para este el desarrollo de este trabajo.

3.1. Perspectiva del estudiante

Mumuki es un sistema cerrado de acuerdo a la clasificación realizada en la Sección 2.3. Mumuki es usado para la enseñanza de programación, es de código libre con licencia MIT[1] y corrige automáticamente ejercicios de programación. Mumuki fue completamente desarrollada en Argentina por docentes universitarios nacionales. Esta plataforma brinda asistencia a profesores y estudiantes en el proceso de aprendizaje. La presentación de conceptos claves se realiza de modo incremental a medida que avanzan los ejercicios.

Actualmente Mumuki soporta 18 lenguajes de programación, incluyendo Haskell, Prolog, Python, JavaScript, C y Ruby entre otros. Además es utilizado por más de 50 instituciones educativas. Entre ellas se destacan 8 universidades de Argentina, y programas de capacitación de los gobiernos de las provincias de San Luis y Tierra del Fuego. Además se destaca su participación en el plan Ceibal que se lleva a cabo en Uruguay. El plan Ceibal en Uruguay es similar a lo que fue Conectar Igualdad en Argentina. Distribuye laptops en las escuelas públicas de todo el país en la búsqueda de achicar la brecha digital de la sociedad [65].

A continuación se describe la experiencia que tiene el estudiante cuando trabaja sobre el sistema que está disponible online de forma gratuita para estudiantes autodidactas llamado Mumuki.io. Los ejercicios, como se puede ver en la Figura 3.1, están distribuidos en capítulos, de forma similar a un libro de texto. En cada capítulo se introducen, de modo incremental, nuevos

[1] https://opensource.org/licenses/MIT

conceptos de programación con un lenguaje de programación específico. Desde fundamentos de programación donde se ven los conceptos de condicional, procedimiento, entro otros. Luego en los siguientes capítulos se ven los tres paradigmas de programación, funcional, imperativo y orientado a objetos con sus conceptos propios. Y por último se trabaja sobre el concepto de Testing. Cada institución formal educativa puede, diseñar sus propios capítulos eligiendo ejercicios de una biblioteca de miles de ejercicios o diseñando los propios.

Figura 3.1: Capítulos disponibles en la interfaz online abierta de Mumuki.io para autodidactas

Cada capítulo está compuesto por varias lecciones, las mismas están compuestas por ejercicios. Cada lección presenta un concepto nuevo o se integran conceptos nuevos a los presentados en las lecciones anteriores. En la Figura 3.2 se muestran la primera y la última lección disponible para el Capítulo 3, correspondientes al lenguaje de programación Haskell.[2]

Cada lección cuenta con un conjunto de ejercicios, que está diseñados para trabajar un concepto en particular. En general el título de la lección está directamente relacionado con el concepto a trabajar.

Como puede verse en la Figura 3.2 cada ejercicio está precedido por un ícono con posibles diferentes colores. Los mismos hacen referencia a la evaluación obtenida en ese ejercicio, los detalles de los colores se explican más adelante en esta misma sección. Esto permite al estudiante tener una visión general de su avance dentro de cada lección.

Un ejercicio particular en la plataforma es presentado como se ve en la Figura 3.3. La interfaz de la plataforma incluye una barra de progreso, una sección de enunciado y un editor donde el estudiante debe escribir su solución. La barra de progreso muestra qué ejercicios han sido resueltos (verde con tilde), cuáles han sido intentados pero no completados (rojo claro u oscuro con cruz o guión) y muestra aquellos ejercicios que no han sido intentados (gris y vacío), estos colores son los mismos que acompañan los nombres de los ejercicios en la Figura 3.2. En la Figura 3.3 se observa que el ejercicio sobre el cual está trabajando el estudiante es marcado con un punto azul sobre la barra de progreso. Mumuki le sugiere al estudiante el orden en el cual debe resolver los ejercicios mediante la barra de progreso y de modo global con el orden de las lecciones, el orden propuesto es de acuerdo a la secuencialidad de los conceptos que se trabajan. De todos modos, el usuario estudiante puede elegir el orden en el cual resolver los ejercicios como él desee, permitiéndole repetir las veces que sea necesario un ejercicio hasta lograr completarlo

[2]Puede ver la lista completa de lecciones aquí https://mumuki.io/central/chapters/82-programacion-funcional

Figura 3.2: Primera y última lección correspondientes al lenguaje de programación Haskell

o dejarlo momentáneamente y retomarlo luego. Es importante destacar que no existe límite de soluciones enviadas por un estudiante para resolver un ejercicio. Esto no es así en otros sistemas cerrados como Codeacademy.

Desde el punto de vista del estudiante, un ejercicio incluye la descripción del problema con al menos un ejemplo y puede, o no, incluir una lista de funciones necesarias para la resolución del ejercicio, como se puede ver en la parte izquierda de la Figura 3.3. En el panel de la derecha, incluye una pestaña con un editor para la solución y otra pestaña con una consola interactiva donde la solución y las funciones reutilizables pueden ser probadas. El feedback automático es dado al estudiante luego que presiona el botón “Enviar”. La solución es probada bajo ciertos casos de test y expectativas desarrolladas por el diseñador del ejercicio. Los test tienen como objetivo chequear el correcto funcionamiento del programa mientras que las expectativas permiten requerir que la solución sea programada usando algún concepto específico como recursión o que reutilice alguna función dada anteriormente. En la Sección 3.2 se explican en detalle las características de los test y expectativas desde la perspectiva del docente.

El ejercicio puede ser evaluado de 4 diferentes maneras: rojo oscuro, rojo claro, amarillo o verde. Rojo Oscuro indica que la solución enviada no puede ser compilada por algún error de sintaxis por lo tanto no puede ser evaluada. Rojo claro indica que la solución pudo ser compilada pero algún resultado de un test no es el correcto. Amarillo significa que todos los test fueron correctos pero alguna expectativa no fue alcanzada. Por último cuando un ejercicio es marcado como verde significa que todos los test y las expectativas fueron evaluadas correctamente. La Figura 3.3 muestra un ejercicio evaluado en verde. Este ejercicio pide al estudiante definir una función que tome una lista de tuits y trunque su contenido a quince caracteres. El estudiante sabe de un ejercicio anterior que cada tuit se representa como una 2-upla donde, el primer elemento es el nombre del usuario que escribió el tuit y el segundo elemento contiene el texto del tuit. Dado que se introdujo la función take, se espera que el estudiante la utilice para recortar un tuit

Figura 3.3: Ejercicio Mumuki desde la vista de usuario

a la longitud de quince caracteres. Los ejercicios disponibles en el sistema tienen definidos casos de test para determinar su corrección, y pueden o no tener definidas expectativas.

En este ejercicio, están definidos los siguientes casos de test que serán evaluados luego de que se compile la solución:

- recortar [] == []
- recortar [(“@tuitero”, “Solo un tweet de prueba”)] == [(“@tuitero”, “Solo un tweet d”)]

En la Figura 3.4 el estudiante envió una solución en donde recorta los primeros catorce caracteres en vez de quince, por lo tanto el segundo test definido no es satisfecho y la solución es evaluada con rojo claro.

En caso que el estudiante envíe una solución que no es correcta sintácticamente Mumuki calificará esta solución con rojo oscuro dado que no puede compilar el programa en la Figura 3.5 se puede ver que el estudiante olvidó usar doble : dentro de la signatura de la función por lo tanto el programa no se puede compilar.

Además de los casos de test este ejercicio también tiene definidas expectativas las cuales en lenguaje natural se pueden leer de la siguiente manera:

- la solución debe usar la función take
- la solución debe declarar explícitamente el tipo de la función recortar

Las expectativas son evaluadas luego de que los casos de test son evaluados, en la Figura 3.6 se observa como el estudiante no declaró explícitamente el tipado de la función entonces la expectativa no es satisfecha y la solución es evaluada como amarillo. Notar que en este caso la solución satisface todos los test unitarios definidos para este ejercicio, pero al no satisfacer las expectativas no cumple con el estilo requerido.

```
SOLUCION:
recortar::[(String,String)] -> [(String,String)]
recortar [] = []
recortar ((x,y):xs) = (x,(take 14 y)):(recortar xs)
```

Feedback:

Figura 3.4: Solución evaluada como rojo claro correspondiente al ejercicio de la Figura 3.3.

```
SOLUCION:
recortar [(String,String)] -> [(String,String)]
recortar [] = []
recortar ((x,y):xs) = (x,(take 15 y)):(recortar xs)
```

Feedback:

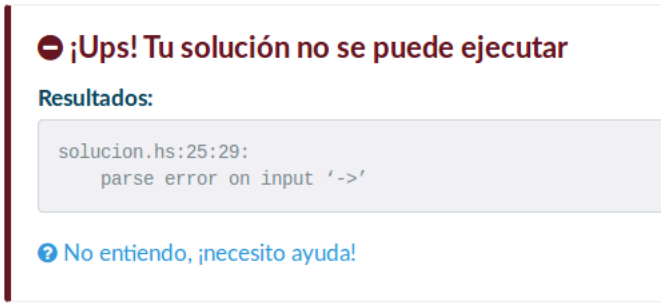


Figura 3.5: Solución evaluada como rojo oscuro correspondiente al ejercicio de la Figura 3.3.

```
SOLUCION:
recortar [] = []
recortar ((x,y):xs) = (x,(take 15 y)):(recortar xs)
```

Feedback:

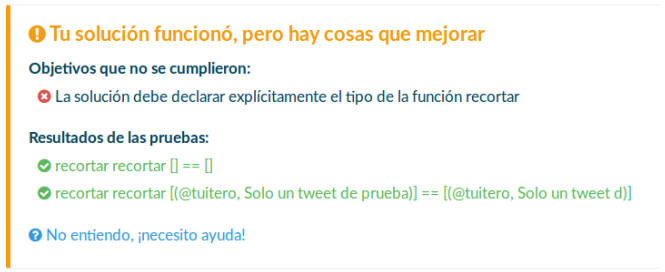


Figura 3.6: Solución evaluada como amarillo correspondiente al ejercicio de la Figura 3.3.

A continuación en la siguiente Sección 3.2 analizaremos el sistema Mumuki desde la perspectiva del docente, se describen cuáles son las herramientas que se utilizan para construir los

ejercicios, definir los casos de tests y las expectativas. Además se hace mención sobre cómo el docente puede realizar un seguimiento de los estudiantes y reconstruir la trayectoria realizada por cada uno de ellos dentro del sistema a partir de las soluciones enviadas.

3.2. Perspectiva del docente

Desde el punto de vista del usuario docente, la plataforma presenta dos principales funcionalidades llamadas **Biblioteca** y **Classroom**.

En la Biblioteca el docente tiene el control sobre las guías de ejercicios. Esta funcionalidad es la que le permite diseñar las guías para trabajar los conceptos que él mismo desee. A la hora de crear un ejercicio, el mismo consta de dos partes, por un lado una descripción del problema que debe resolver el estudiante mediante un programa, que por lo general es un programa de pocas líneas. Por otro lado, la herramienta para diseñar ejercicios proporciona una forma de calcular automáticamente el feedback para la solución del ejercicio en dos niveles diferentes: corrección y calidad. Para ser correctos, el docente debe definir un conjunto de **Tests**, los cuales son ejecutados luego de que el estudiante envía la solución, estos tienen por objetivo comprobar que la solución enviada tenga un comportamiento determinado sobre ciertos escenarios. Si bien la corrección de los programas no puede ser asegurada con este método, es lo suficientemente buena dada la complejidad de los ejercicios propuestos.

Para poder evaluar la calidad, a nivel sintáctico de la solución, el docente cuenta con un conjunto de características llamadas **Expectativas** las cuales pueden ser seleccionadas como necesarias dentro de la solución o que las mismas no deben estar presentes en la solución enviada. Las mismas deben ser definidas definidas por el docente o seleccionadas de las ya existentes a la hora de diseñar el ejercicio. Estas permiten de algún modo mantener ciertos lineamiento sobre las soluciones esperadas. Por ejemplo, mediante una expectativa, se puede definir qué función se espera que un ejercicio reuse. O evitar ciertas características del código como comparaciones innecesarias de valores booleanos.

%endfigure

Otra de las funcionalidades que provee Mumuki al docente, es el seguimiento para sus cursos, llamada **Classroom** donde puede ver el progreso detallado de cada uno de los alumnos registrados en sus cursos tal cual se muestra en la Figura 3.7.

Esta herramienta le permite ver al docente la cantidad de ejercicios intentados, la cantidad de soluciones enviadas y el resultado obtenido para cada una de esas soluciones enviadas por los estudiantes. El docente a cargo del curso puede ver todas las soluciones enviadas por un estudiante en un sistema de versiones de las soluciones enviadas y de este modo puede reconstruir el trayecto de aprendizaje que ha llevado a cabo el usuario con la desventaja de tener que hacerlo manualmente. En la Figura 3.8 se muestra una secuencia de imágenes las cuales corresponden a un fragmento de las soluciones enviadas por un usuario para el mismo ejercicio, el texto resaltado en verde indica una parte nueva en el programa, respecto de la solución previa mientras que lo resaltado en rojo es lo que se quitó respecto de la solución anterior.

Esta secuencia de imágenes muestra parte del trayecto realizado por un estudiante para intentar completar el ejercicio “Recortar Tuits”, el cual se presentó en la Figura 3.3. Las tres soluciones que se muestran en la Figura 3.8 son consecutivas y todas tienen errores de tipo. Si nos enfocamos en el tiempo transcurrido entre las soluciones, podemos observar que la distancia es corta, lo cual nos da una idea que el estudiante se puede estar frustrando. Y más aún si tenemos en cuenta la diferencia entre las soluciones enviadas es muy pequeña considerando la cantidad de caracteres modificados o eliminados entre soluciones. Notemos también que el estudiante envía

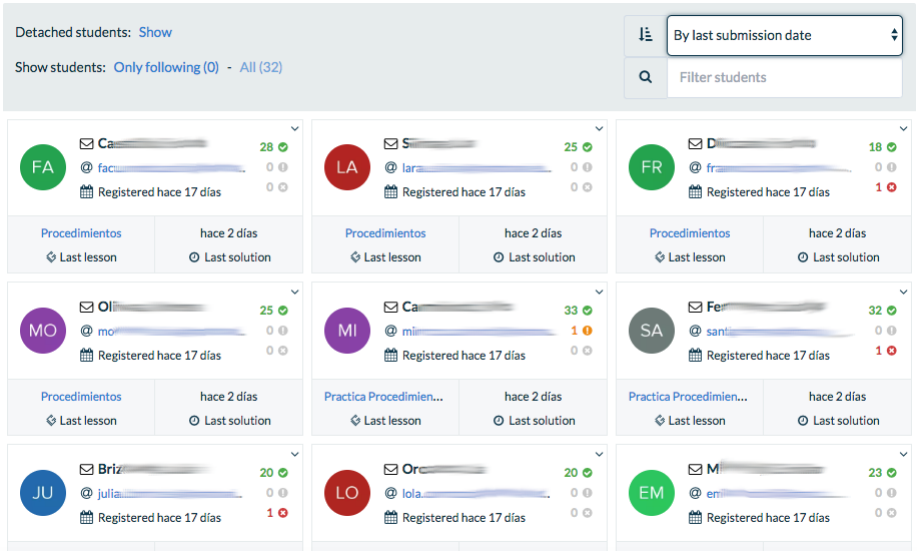


Figura 3.7: Vista del curso desde la perspectiva del docente

Figura 3.8: Extracto del trayecto realizado por un estudiante antes de abandonar un ejercicio

una solución sin realizarle ningún cambio a pesar de que su solución previa no cumplía con todos los test. Luego de esto introduce algunos cambios de modo que la solución tiene más errores de tipos la primera de esta secuencia. Esta secuencia nos está mostrando que el estudiante no entiende como arreglar su solución generando de algún modo frustración y aumentando así la probabilidad de abandono del ejercicio.

Actualmente esta es la forma que tiene el docente a cargo de un curso de detectar si un alumno está o no frustrado y con posibilidad de abandonar el ejercicio. El análisis de la situación del estudiante tiene que ser realizado de modo manual por parte del docente, claramente esta es una limitación del sistema Mumuki cuando se trata de cursos grandes.

La secuenciación de contenidos propuestas por las guías que componen a Mumuki trabajan conceptos fundamentales de programación de modo incremental, empezando por conceptos simples como variables, funciones, hasta llegar a conceptos más complejos como recursión, listas, entre otros. Esta secuenciación junto con la posibilidad de que los alumnos decidan su proceso

```
{
    "content" : "recortar :: [(String,String)] -> [(String,String)] \r\n recortar [] = [] \r\n
recortar ((x,y):xs) = (x,(take 15 y)):(recortar xs)",
    "status" : "Passed",
    "created_at" : "2016-04-17T02:32:26",
    "name" : "Recortar tuits",
    "guide.slug.name" : "mumuki-guia-funcional-listas",
    "submissions_count" : 1,
    "expectation_results" : [
        {
            "binding" : "take",
            "inspection" : "HasBinding",
            "result" : "passed"
        },
        {
            "binding" : "recortar",
            "inspection" : "HasTypeSignature",
            "result" : "passed"
        }
    ],
    "test_results" : [
        {
            "title" : "recortar [] == []",
            "status" : "passed",
        },
        {
            "title" : "recortar [(“@tuitero”, “Solo un tweet de prueba”)] == [(“@tuitero”, “Solo un
tweet d”)]",
            "status" : "passed",

        }
    ],
}
```


Figura 3.9: Ejemplo de datos almacenados por el sistema luego de que un estudiante envía una solución.

de aprendizaje, hacen que el rol del docente se transforme en acompañante de los alumnos. Esto lo puede realizar gracias a las herramientas de seguimiento que provee el sistema. El seguimiento debe realizarlo manualmente, y el problema surge cuando el curso es numeroso. Por lo tanto es importante poder agregar un nuevo funcionamiento en el sistema que permita realizar el seguimiento de modo automático, detectando posibles abandonos de ejercicios por parte de los alumnos para que el docente pueda acompañar al alumno de mejor manera. Como vimos en el Capítulo 2 este funcionamiento puede ser particularmente útil para los docentes con pocos conocimientos previos que necesitan un soporte mayor a la hora de dar clases de CC. A continuación en la Sección 3.3 se describe las características del conjunto de datos con el que trabajaremos.

3.3. Conjunto de datos Mumuki

Cada vez que un estudiante presiona el botón enviar, como el de la Figura 3.3, la solución es corregida por Mumuki y se guarda en la base de datos la solución enviada junto con información de la evaluación de los test unitarios, la evaluación de las expectativas y cierta información referida al usuario.

Diariamente acceden a la plataforma alrededor de 750 usuarios. Esto hace un total de alrededor de 16 mil usuarios activos mensualmente, contando al día de la fecha con 70 mil usuarios registrados, siendo el 54 % mujeres. Además cuenta con alrededor de 100 usuarios con permisos de docentes.

En la Figura 3.9 se muestran los campos relevantes de un ejemplo correspondiente a una solución enviada para el ejercicio “Recortar tuits” el mismo se encuentra en la guía que trabaja sobre el tipo de datos Listas, correspondiente al lenguaje de programación Haskell.

La información relevante, para nuestro trabajo, se encuentra en los siguientes campos:

- **content**: Contenido escrito por el estudiante para la solución enviada.
- **created_at**: Fecha y hora en la cual el estudiante envió la solución.
- **status**: Resultado de la evaluación realizada por la plataforma, la misma se constituye de compilación, tests y comprobación de expectativas.
- **exercise.name**: Nombre del ejercicio como lo ve el estudiante al resolverlo.
- **submission.count**: Número de solución enviada por ese usuario para ese ejercicio.
- **expectations_results**: Guarda los resultados de evaluar las expectativas definidas para el ejercicio.
- **test_results**: Guarda los resultados de los casos de test definidos para cada ejercicio, de forma similar en expectations results son guardadas las evaluaciones de las expectativas.

El campo status de las soluciones enviadas tienen cuatro opciones posibles, que se explicaron previamente en la Sección 3.1: rojo oscuro, rojo claro, amarillo y verde.

Para este trabajo se cuenta con un conjunto de más de 1 millón de soluciones enviadas por estudiantes en Mumuki para ejercicios de Haskell. Dentro del mismo se encuentran soluciones enviadas por estudiantes dentro del sistema de educación formal de la república Argentina, y de Uruguay. También se pueden encontrar soluciones enviadas por trabajadores de empresas donde se capacita a los empleados usando la plataforma y estudiantes autodidactas que utilizan la herramienta por su cuenta.

En la Tabla 3.1 se muestran la cantidad de soluciones según el color con el que fueron evaluados. Junto con otros datos como cantidad de usuarios diferentes, cantidad de ejercicios intentados y el tiempo promedio de uso del sistema.

Cantidad de usuarios	4114
Ejercicios intentados	165734
Soluciones rojo claro	515819
Soluciones rojo oscuro	327817
Soluciones amarillas	62868
Soluciones verdes	345378
Total de soluciones	1251882

Tabla 3.1: Datos cuantitativos del conjunto de datos.

Este conjunto de soluciones enviadas por los usuarios, constituirá el dataset que utilizaremos para entrenar algunos modelos de aprendizaje automático. Con el objetivo de detectar la probabilidad de que un estudiante abandone un ejercicio de modo automático. Esta funcionalidad tiene como objetivo acompañar a los docentes con poca formación en el aula como lo vimos en el Capítulo 2. En el siguiente Capitulo 4 discutiremos sobre la actualidad de las técnicas de aprendizaje automático y extracción de información sobre sistemas diseñados para educación, además analizaremos el conjunto de datos en profundidad, con el objetivo de decidir que técnicas utilizaremos para intentar resolver el problema planteado. Y más adelante en el Capitulo 5 definiremos los diferentes experimentos que se llevarán a cabo.

Capítulo 4

Análisis del conjunto de datos de Mumuki en base a experiencias previas

En este capítulo se describen trabajos relacionados de minado de datos sobre enseñanza de programación. En los últimos años se han multiplicado las experiencias en esta área con el objetivo de analizar y visualizar la experiencia de los alumnos en cursos de programación. En la Sección 4.1 presentaremos una clasificación de los distintos tipos de trabajos relacionados realizados en el área. En la Sección 4.2 se describen aquellos trabajos que están más relacionados con los objetivos de esta tesis. Por último en la Sección 4.3 se analizan los conjunto de datos con los que se cuenta para realizar este trabajo y se definen conceptos básicos adaptándolos del trabajo previo.

4.1. Situación con respecto al estado del arte

En los últimos años una de las mayores innovaciones a nivel educativo fue la emergente aceptación y utilización de sistemas online de enseñanza. Gracias a esto se han desarrollado diversas técnicas de recolección automática de la información generada en estos sistemas. Contar con estas herramientas de recolección de información permiten generar grandes volúmenes de datos dando lugar al uso de técnicas de minado de datos.

Diversas experiencias muestran cómo los estudiantes, en situaciones de aprendizaje de la programación, tienen diferentes formas de aprender según su género, su experiencia previa entre otros factores. Esta condición se da sin importar el nivel educativo donde se encuentren, desde primaria [53] , secundaria [22] hasta en el nivel universitario [69]. Más aún el uso de minado de datos se ha convertido en una tarea muy popular en investigaciones realizadas en *Massive open online course* (MOOCS) dada la gran cantidad de datos que generan los mismos. En todas estas experiencias se evidencia que los alumnos trabajan de diversas maneras, ya sea por su experiencia previa en CC como en otras disciplinas.

Por lo tanto es importante poder entender de qué forma aprenden distintos alumnos y que factores influyen en la tarea de aprender. Intentar hacer este análisis a mano con la cantidad de datos que se generan es prohibitivamente costoso. El minado de datos educativo surge como una opción más económica que permite encontrar patrones de comportamiento y así intentar personalizar la experiencia educativa.

Tomando como referencia el trabajo realizado por Ihantola et. al [36] podemos clasificar los trabajos dentro del área de la siguiente manera como puede verse en la Tabla 4.1.

Categoría	**Objetivo de Investigación**	**Referencias**
Estudiantes		
Habilidades y conocimientos	Evaluar el conocimiento de los estudiantes y conceptos específicos	[76, 42, 14]
Estados emocionales	Evaluar los estados de ánimo de los estudiantes en ejercicios de programación	[66]
Comportamiento	Determinar estilos de comportamiento de los estudiantes	[31, 69, 61]
Dificultades	Encontrar conceptos que son difíciles de incorporar	[19, 20, 41]
Desempeño y riesgo de abandono	Identificar el riesgo de abandono de un curso y tratar de predecir su desempeño	[30, 45, 70]
Entorno		
Evaluación y devolución automática	Beneficios de la evaluación automática y cuanto impacta la devolución	[4, 2, 24]
Uso del entorno de desarrollo	Análisis de las interacciones entre el estudiante y el sistema	[25, 55, 1]
Testing	Enfoque para mejorar el testing automático sobre soluciones de estudiantes	[26, 27, 68]
Programación		
Comportamiento	Análisis de la soluciones, compilaciones, debugging y su métrica asociada con relación al progreso del estudiante	[28, 37, 38]
Errores	Entender los errores cometidos durante el proceso de programación	[3, 16, 23]
Patrones	Entender como los estudiantes aprenden a programar	[59, 63]
Métricas	Enfocado en métricas, no necesariamente con el objetivo de aprender sobre el comportamiento de los estudiantes	[18, 71]
Estrategias	Determinar que estrategias usan los estudiantes para resolver los ejercicios	[43, 44, 67]

Tabla 4.1: Clasificación de trabajos de minado de datos a en base a su objetivo de investigación

De acuerdo a esta Tabla 4.1 los trabajos realizados en el área se pueden distinguir según el objetivo de investigación en tres grandes grupos:

Estudiantes: Busca entender las habilidades, dificultades y comportamiento de los mismos ante los ejercicios. Por ejemplo artículos que trabajen en predecir la calificación de estudiantes en base a su trayecto previo.

Entorno: Hace especial foco en el sistema sobre el cual el estudiante trabaja, por ejemplo el uso de IDE, la corrección automática, entre otros temas.

Programación: Por último la categoría Programación tiene como objeto de investigación el descubrimiento de comportamientos, patrones, estrategias y errores cometidos durante el proceso de "coding" entre otros aspectos.

Dentro de la categoría Programación hay varias investigaciones que permiten contextualizar este trabajo, pero hay dos ejemplos que son de particular interés para este trabajo, ya que ambos modelan la trayectoria de los estudiantes en tareas de programación con el objetivo de predecir diferentes situaciones, ellos son el trabajo realizado por Blikstein [15], y el realizado Wang et al [74]. Los cuales serán analizados y descriptos en detalle en la siguiente sección.

4.2. Trabajos previos sobre modelado de trayectorias de estudiantes de programación

Como se mencionó en la sección previa, la cantidad de trabajos relacionado con el minado de datos en temáticas educativas ha aumentado en el último tiempo, principalmente por dos situaciones, la generación de grandes volúmenes de datos de modo automático y el interés tanto de la comunidad científica como la empresarial.

Es de interés para este trabajo entender trabajos previos realizados en el área como lo son los realizados por Blikstein et al. [15], Wang et al. [74] ya que ambos hacen un intento por modelar la trayectoria de los estudiantes en tareas de programación con el objetivo de predecir diferentes situaciones.

Ambos trabajos aplican técnicas de aprendizaje automático sobre datos de educación con el objetivo de generar una representación de la evolución en el aprendizaje de los estudiantes conforme resuelven diferentes ejercicios y a partir de eso permitir cada trabajo persigue un objetivo específico en particular. Por ejemplo el trabajo de Wang [74] trabaja en modelar el conocimiento que adquiere un estudiante sobre los nuevos conceptos a medida que avanza sobre la resolución de los diferentes ejercicios propuestos con el objetivo de poder entender de forma aprenden los estudiantes y generar feedback de acuerdo a la trayectoria de aprendizaje en la que se encuentra. Este trabajo fue realizado con datos proveniente de ejercicios que pertenecen al curso *"Hour of Code"*, un MOOC en *Code.org*. El mismo contó con un conjunto de datos de más de un millón de soluciones enviada las cuales fueron utilizadas para entrenar una red neuronal profunda, con dos tareas específicas, identificar la trayectoria que está realizando el estudiante y poder predecir si completará satisfactoriamente o no el próximo ejercicio dentro del mismo curso. Por otro lado, dada una solución intermedia dentro de la trayectoria enviada por el estudiante predecir si esté completará satisfactoriamente el ejercicio o no. Es importante notar que este trabajo está realizado sobre un lenguaje de bloques y la plataforma donde se resuelven los ejercicios tiene un límite en la cantidad de bloques que pueden ser utilizados, por lo que el espacio de soluciones posibles es finito. En este trabajo se explota el hecho de que las soluciones posibles sean finitas. Esta estrategia no es posible en nuestro trabajo porque el número de posibles soluciones a nuestros ejercicios es infinita.

Por otro lado el trabajo de Blikstein et al [15] tiene como objetivo, encontrar una representación de la trayectoria del estudiante que le sirva predecir cuál será su nota final de ahí se desprende poder predecir si el estudiante completará con éxito o no el curso.

Otra similitud que comparten los trabajos es la granularidad con la que fue recabada la información, ambos trabajos construyeron su conjunto de datos de forma similar. Wang recolectó

todas las soluciones enviadas junto con información como la fecha y la hora de la solución, entre otros datos. Del mismo modo Blikstein recabó cada una de las soluciones compiladas dentro el entorno de desarrollo ya que la experiencia se llevó a cabo de modo offline, recabando casi 200 mil programas enviados por los estudiantes junto con su fecha y hora de la solución compilada. Contar con todas las soluciones para cada estudiante para cada ejercicio junto con la información adicional permite recrear la trayectoria realizada por el estudiante de forma más fiel.

A diferencia del trabajo de Wang, el trabajo [15] fue realizado sobre versiones de código, escritos en Java, recolectados en cada compilación realizadas por los estudiantes, pero sin ninguna restricción sobre el uso del lenguaje. Este tipo de ejercicios son llamados open-ended ya que el espacio de soluciones, es decir los programas que lo resuelven, puede ser infinito. Esta es una de las principales diferencias que hay entre estos trabajos. Trabajar sobre un espacio más grande puede hacer más dificultosa la tarea de minado de datos.

En base a esa información recolectada por Blikstein et al [15] se hizo un análisis incremental de herramientas y características extraidas de los datos. Con el objetivo de descubrir si estas características, como puede ser considerar la cantidad de cambios en el código entre una solución y la otra, pueden dar cuenta del tipo de trayectoria que recorre el estudiante para solucionar el ejercicio. Considerando como objetivo global poder predecir la nota final del curso en donde estaban inscriptos.

En nuestro trabajo tenemos en cuenta estas y otras caracteristicas como se describe en el proximo capitulo. La diferencia entre este trabajo y el nuestro es que Blikstein considera sólo un enunciado de ejercicio y su modelo es dependiente del mismo. Nuestros modelos son independientes del enunciado particular de un ejercicio.

En la siguiente sección analizaremos el conjunto de datos con el objetivo de entender sus características. Asi podremos construir modelos de aprendizaje automático para modelar la trayectoria de los estudiantes. De esta manera nuestro objetivo es predecir el momento donde abandonará un ejercicio de forma independiente del enunciado.

4.3. El conjunto de datos de Mumuki: definiciones y estadísticas básicas

Tal como mencionamos en la sección anterior, cada trabajo realizado está directamente relacionado con el conjunto de datos sobre el que se realizó el análisis por eso es fundamental entender la estructura del mismo. Esto es crucial para poder entender los datos y en base a ello poder generar las características de entrenamiento para los modelos de aprendizaje automático.

Con el objetivo de poder construir un modelo de aprendizaje automático que permita predecir abandonos necesitamos realizar algunas definiciones.

1. Sesión: soluciones enviadas de forma continuada donde el tiempo de inactividad no supera un cierto umbral que definimos empíricamente en esta sección.

2. Abandono en sesión: consideramos abandono en sesión cuando el estudiante supera el umbral de tiempo de inactividad sobre un ejercicio dejando en rojo su última solución enviada dentro de la sesión.

3. Abandono por cambio de ejercicio: durante una sesión el estudiante deja un ejercicio en rojo y cambia a trabajar sobre otro ejercicio dentro de la plataforma Mumuki.

Es de interés analizar la dimensión temporal del conjunto de datos con el que contamos para este trabajo para poder encontrar un umbral para nuestra definición de sesión, entender la

	Intro Algo 2018		**Mumuki io**	
Submissions Status	**Cantidad**	**Proporción**	**Cantidad**	**Proporción**
Errored (rojo oscuro)	7457	38.5	69249	29.3
Failed (rojo claro)	7855	40.5	86525	36.7
Passed (verde)	3270	16.9	68107	28.9
Warnings (amarillo)	790	4.1	11821	5.1
Cantidad de submissions	19372		235742	
Estudiantes	75		3915	
Ejercicios intentados	3043		155450	

Tabla 4.2: Comparativa de la distribución de los dataset según la evaluación de las soluciones enviadas. Ambos grupos trabajaron con el lenguaje Haskell.

forma del mismo y en base a ello proponer diversos características. El conjunto de datos provisto por Mumuki está compuesto de soluciones enviadas por los usuarios dentro del sistema, como vimos en el Capitulo 3 cada solución enviada cuenta con los siguiente datos: Content, Status, Created_at, Exercise.name, Submission.count, Test_results.

En el Capitulo 3 se hace una descripción en detalle de cada uno de los campos. El conjunto de datos con el que contamos para este trabajo puede ser dividido en dos partes, por un lado el correspondiente al curso de Introducción a los Algoritmos de FAMAF del segundo cuatrimestre del año 2018, el cual es el primer curso de programación correspondiente a la carrera Licenciatura en Ciencias de la Computación. Por otro lado contamos con el conjunto de datos recolectados en Mumuki.io durante el año 2018 fuera del ámbito de la enseñanza formal. Estas soluciones son enviadas por usuarios no enmarcados en un sistema formal de enseñanza, es decir, autodidactas. Nos interesa analizar estos dos dataset dado que son dos formas distintas de la utilización del sistema. Queremos comparar nuestros modelos en los dos tipos de enseñanza. Por lo tanto, restringimos nuestro análisis en Mumuki.io a aquellas submisiones realizadas en Haskell.

Del conjunto de datos de Introducción a los Algoritmos contamos con entrevistas *think out-aloud* [51] realizadas a los estudiantes del año 2018, las cuales nos permitieron conocer cómo los estudiantes utilizan Mumuki siguiendo una metodología de investigación cualitativa. Estas entrevistas nos permitieron empezar a entender las definiciones planteadas al inicio de esta sección. En particular nos hicieron sospechar de la frecuencia de abandonos por cambio de ejercicio que luego encontramos en los datos.

Cuando un estudiante envía una solución dentro de un ejercicio, esta es corregida por el sistema de acuerdo con lo descripto en el Capitulo 3. A continuación en la Tabla 4.2 se muestra, para cada uno de los conjuntos de datos, la distribución de las soluciones de acuerdo a como fueron evaluadas.

Se observa que en Mumuki.io hay una mayor proporción de submissiones passed (verdes) haciendo evidente que los estudiantes de Intro Algo 2018 son todos principiantes mientras que Mumuki.io incluye a un grupo con experiencia más heterogénea. Luego nos interesa saber si varía la forma en que los estudiantes usan el sistema, considerando que están siendo utilizados en contextos diferentes, para ello analizamos la cantidad de soluciones enviadas por día y en que horario fueron enviadas. En la Figura 4.3 se muestra la utilización por día de cada uno de los conjuntos de datos. Podemos ver que la utilización es pareja a lo largo de los días para el conjunto de datos de Mumuki io mientras que para el conjunto de datos de Intro Algo la mayoría de las soluciones se encuentran entre los días que se dictó la materia.

Luego de este análisis donde vimos que la distribución por días varia según el conjunto de

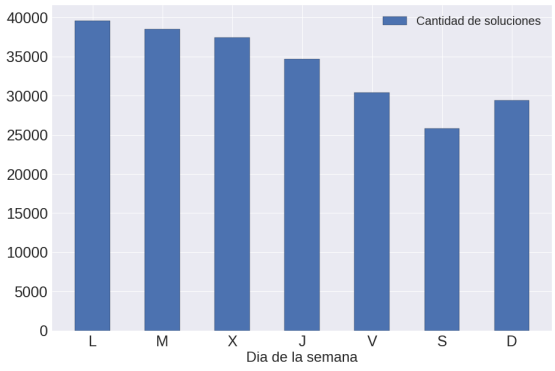


Figura 4.1: Distribución por día Mumuki io.

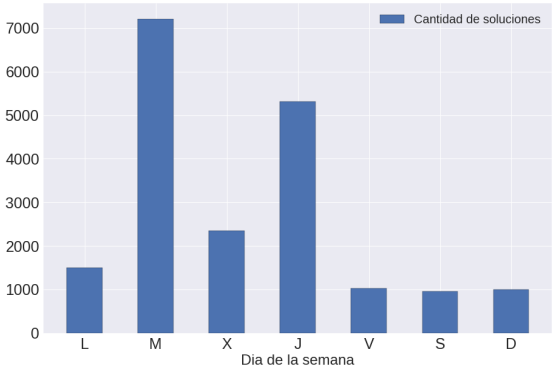


Figura 4.2: Distribución por día Intro Algo.

Figura 4.3: Comparación de la utilización del sistema Mumuki por día.

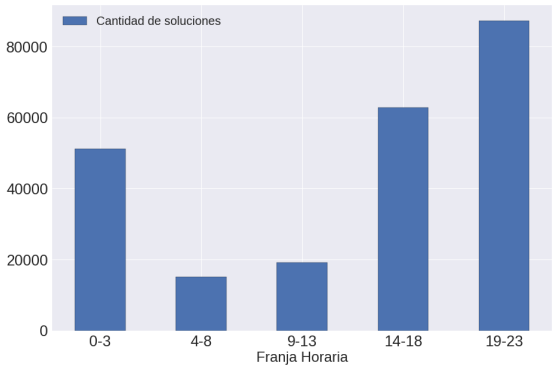


Figura 4.4: Distribución por hora Mumuki io.

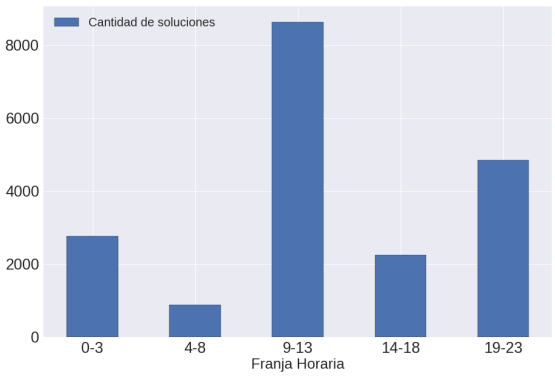


Figura 4.5: Distribución por hora Intro algo.

Figura 4.6: Comparación de la utilización del sistema Mumuki por hora.

datos que estemos usando nos interesa ver que pasas en la utilización por hora. En la Figura 4.6 se comparan la distribución por hora de la utilización del sistema Mumuki en cada uno delos conjuntos de datos

Como podemos ver en la Figura 4.2, la mayor cantidad de soluciones enviadas fueron hechas los Martes y Jueves. Además si observamos la distribución por hora podemos ver en la Figura 4.5 que en la fraja horaria de 9 a 13, es la franja donde se concentra la mayor cantidad de soluciones enviadas. Estos fueron los días y el horario en los cuales se dictaba la materia. Es importante considerar que este ámbito los estudiantes contaban con la asistencia de la docente a cargo de la materia, ayudantes de alumnos y tutores. Mientras que en el conjunto de datos de Mumuki.io la distribución es mas pareja como puede verse en la Figura 4.1 no hay un día en particular donde la utilización sea considerablemente más grande que las otras. Pero si podemos ver que a la utilización del sistema crece considerablemente luego de las 19 horas de acuerdo a la Figura 4.4.

Con el objetivo de encontrar el valor del umbral de tiempo para la definición de sesión que resulte relevante según el conjunto de datos, se calcula la distribución de las distancias de tiempo entre soluciones enviadas por estudiante considerando por separado las soluciones enviadas en Introducción a los Algoritmos y Mumuki io.

En la Figura 4.9 vemos que los percentiles de 90 % de los tiempos entre soluciones varían. Para el dataset correspondiente a Introducción a los Algoritmos el tiempo es de 454 segundos (7.5 minutos) mientras que para Mumuki IO el tiempo es de 565 segundos (9.41 minutos).

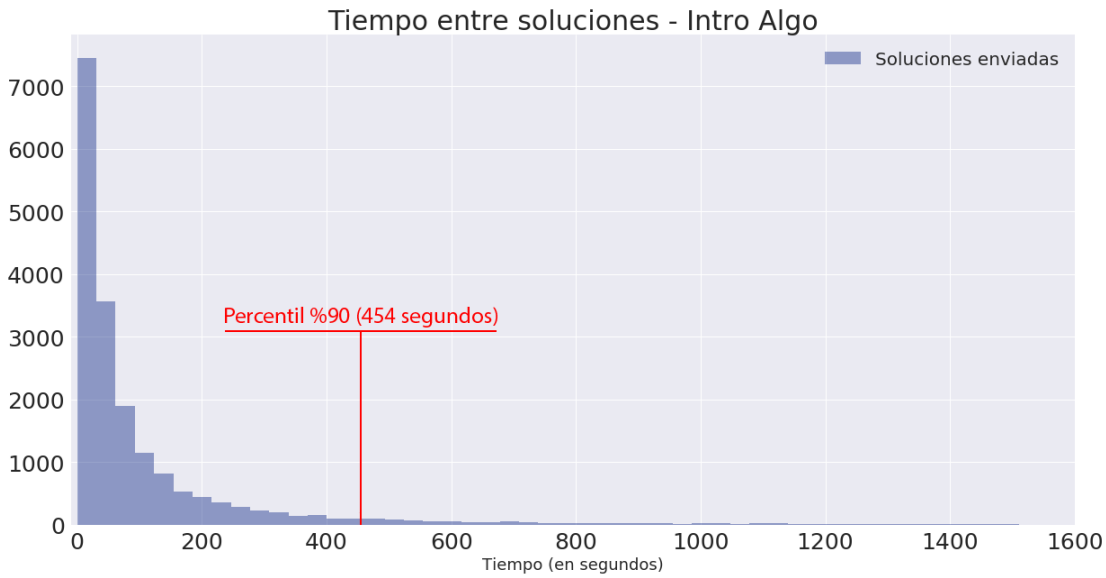


Figura 4.7: Tiempo entre soluciones para el conjunto de datos de Introducción a los Algoritmos 2018.

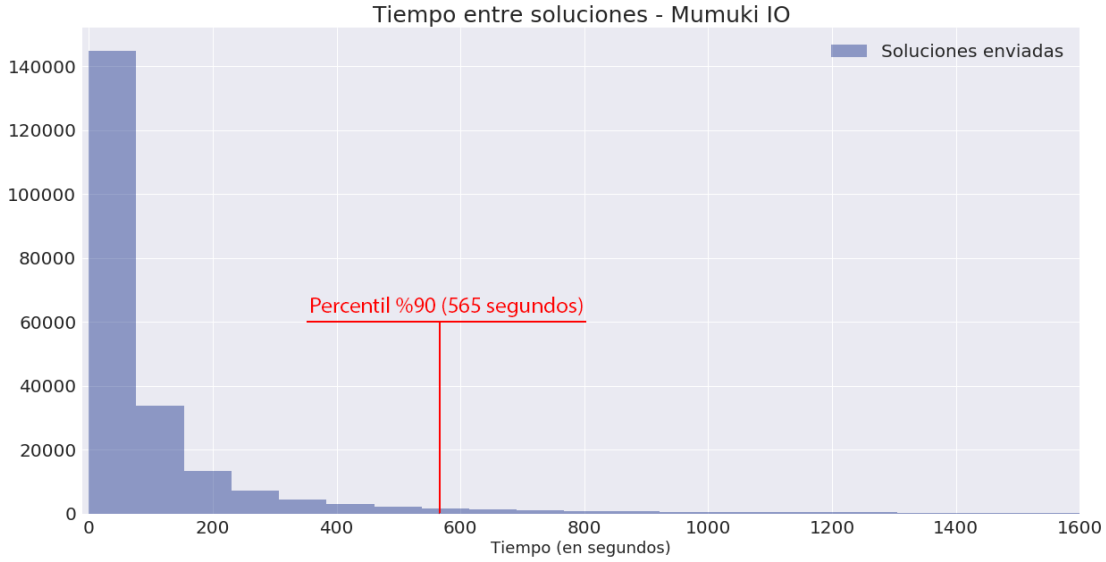


Figura 4.8: Tiempo entre soluciones para el dataset de Mumuki IO.

Figura 4.9: Comparación de tiempo entre soluciones para ambos datasets.

A partir de este análisis proponemos este umbral que cubre el 90 % de las submisiones para definir el concepto de sesión. En Mumuki.io definimos el umbral igual a 565 segundos y en el otro dataset igual a 454 segundos. Luego de este tiempo la probabilidad de que el estudiante vuelva a hacer una submisión en el mismo ejercicio dentro del mismo día es menor al 10 %. A continuación en el Capitulo 5 se describe como se extraen automáticamente diversas características de estos dos conjuntos de datos.

Capítulo 5

Aprendizaje automático para el modelado de trayectorias de aprendizaje

En este capítulo se describen el diseño de los experimentos a realizar con el objetivo de predecir la deserción en ejercicios de programación y así modelar las trayectorias de aprendizaje. Primero en la Sección 5.1 se justifica por qué el problema planteado se puede modelar como un problema de clasificación de aprendizaje supervisado. Además se presentan los métodos que se utilizan para anotar el conjunto de datos. Luego en la Sección 5.2 se explican las diversas características que se extraerán automáticamente con el objetivo de encontrar la mejor forma de representar el comportamiento del estudiante para detectar a tiempo la posibilidad de abandono. Luego en la Sección 5.3 se presentan los diversos modelos de aprendizaje automático con los que se llevarán a cabo los experimentos, se realiza una breve descripción de cada uno de los modelos y se explica la métrica con la cual se evalúa el impacto de cada característica dentro del modelo. En el próximo capítulo se presentan los resultados.

5.1. Aprendizaje supervisado y proceso de anotación

La tarea de detectar el abandono de un estudiante a lo largo de la utilización de un sistema como Mumuki, puede ser enmarcada en una tarea de aprendizaje supervisado [46]. El aprendizaje supervisado es una técnica de aprendizaje automático que consta de aprender una función que mapea una entrada con una salida a partir de un conjunto de pares de ejemplos entrada-salida. Esta técnica infiere una función a partir de ejemplos de entrenamiento etiquetados, habitualmente estos ejemplos son una tupla que consiste de un elemento de entrada, habitualmente un vector y un valor de salida esperado. La Figura 5.1 esquematiza el problema de aprendizaje supervisado para el problema de clasificación binaria para los valores si abandonó y no abandonó. El objetivo es que, dado un conjunto de entrenamiento, se aprenda una función h, donde

$$h : X -> Y$$

Tal que $h(X)$ es una buena predicción para el valor de Y. En el caso de que Y solo pueda tomar valores discretos decimos que el problema de aprendizaje supervisado es una problema de clasificación binaria. En nuestro caso, modelamos nuestro problema como una clasificación binaria donde X es una formalización de ciertas características de las soluciones enviadas.

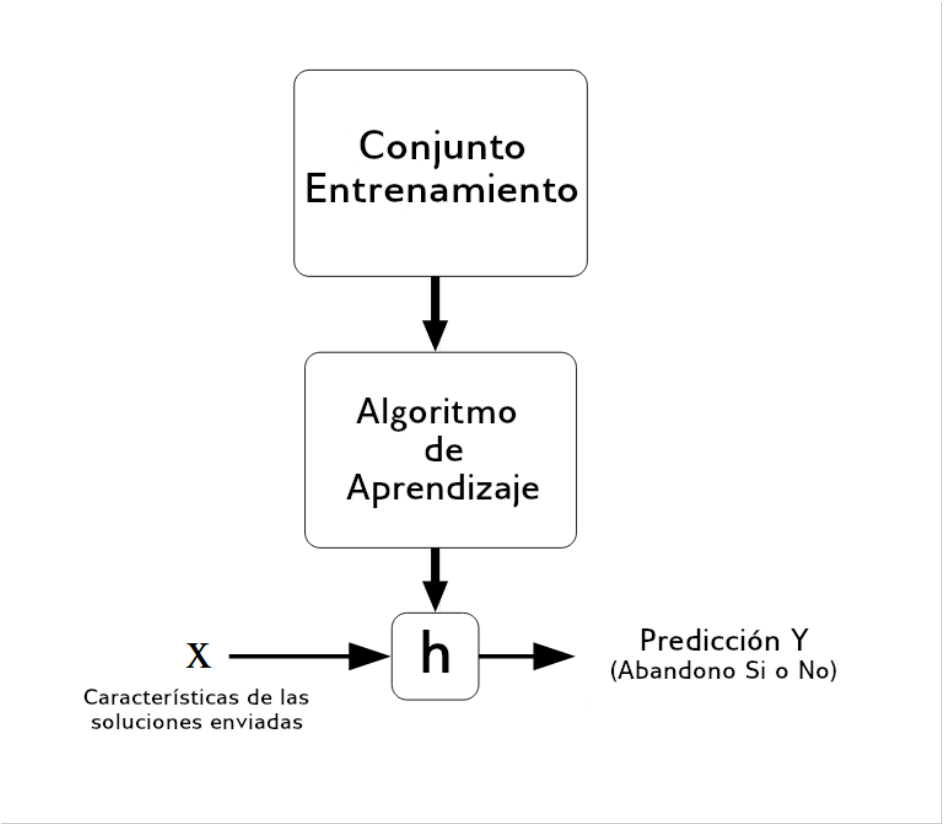


Figura 5.1: Esquema de este problema de aprendizaje supervisado

Para poder aplicar este tipo de algoritmos y entrenar el modelo, como propondremos en la Sección 5.3, es necesario tener una gran cantidad de ejemplos de pares (X,Y). En otras palabras en necesario anotar el conjunto de datos. En muchos casos los conjuntos de datos requieren anotación humana manual (por ejemplo al anotar datasets de fotos de animales para clasificarlas en el tipo de animal). En este trabajo es posible extraer las anotaciones de abandono automáticamente de nuestro dataset. Es posible "mirar hacia el futuro" en el dataset y saber si un estudiante pudo resolver un ejercicio correctamente o lo abandonó modificando así su trayectoria de aprendizaje.

Definimos nuestras variables como: X es cada solución que envía un estudiante para un ejercicio dado. En la Sección 5.2 definiremos las características con las cuales representaremos esta solución y su contexto. Y es una etiqueta binaria que indica si ocurrió o no abandono.

A continuación, mostraremos los criterios seleccionados para anotar el conjunto de datos que describimos en la Sección 4.3. Recordemos la definición de sesión que se propuso en la Sección 4.3, consideraremos una sesión al conjunto de soluciones enviadas de forma continuada donde el tiempo de inactividad no supera un cierto umbral que definimos empíricamente para cada uno de los conjuntos de datos.

Consideramos dos formas de anotar el conjunto de datos. En la primera forma, asociamos el abandono del ejercicio a la última solución enviada por el estudiante inmediatamente antes del abandono. En segundo lugar, asociamos el abandono del ejercicio a todas las soluciones enviadas durante la sesión que termina en abandono.

1. Primera Forma

   - **Abandono por cambio de ejercicio:** Consideraremos un abandono por cambio de ejercicio cuando el estudiante cambie de ejercicio habiendo dejado la última solución

en el ejercicio que estaba trabajando evaluada en rojo. Acá no se hace distinción si dejó el ejercicio en rojo oscuro o rojo claro. En el Capítulo 3 se explicó en detalle cada una de estas posibles evaluaciones que puede recibir un ejercicio.

- **Abandono en sesión:** Consideraremos un abandono en sesión cuando la última solución enviada en la sesión fue evaluada en rojo, sin distinción entre rojo oscuro y rojo claro.

De ese modo los conjuntos de datos quedan distribuidos como se muestra en la Tabla 5.1.

	Mumuki io		**Intro Algo**	
	#	%	#	%
Abandonos en sesión	4983	2.1	644	3.3
Abandonos por cambio de ejercicio	6939	2.9	1086	5.6
Total abandonos	11922	5.1	1730	8.9
Total no abandonos	223780	94.9	17567	91.1
Total soluciones	235702	100	19297	100

Tabla 5.1: Distribución de ambos conjunto de datos luego de haber sido anotados de la primer forma propuesta

2. Segunda Forma
   En esta segunda forma de anotar el conjunto de datos, en vez de considerar sólo la última solución como el abandono, se hace un cambio de perspectiva considerando todas las soluciones que hizo hasta llegar a ese momento de abandono. El objetivo es considerar la trayectoria que realizó el estudiante para llegar a esa situación. A continuación se redefine la forma de anotar el conjunto de datos de la siguiente manera:

   - **Abandono por cambio de ejercicio:** un estudiante está resolviendo el ejercicio K. Al cabo de una cantidad N de soluciones decide cambiar de ejercicio dejando la solución N-esima evaluada en rojo, sin importar si es rojo oscuro o claro, anotaremos entonces las N soluciones de esa sesión como abandono por cambio de ejercicio.

   - **Abandono en sesión:** anotaremos como abandono todas las soluciones enviadas dentro de una sesión donde la última solución antes de terminar la sesión fue evaluada en rojo.

   De este modo ambos conjuntos de datos quedan anotados de la siguiente forma y las clases tienen la siguiente distribución:

	Mumuki io		Intro Algo	
	#	%	#	%
Abandonos en sesión	32189	13.7	3382	17.5
Abandonos por cambio de ejercicio	39358	16.7	5781	29.9
Total abandonos	71547	30.4	9163	47.4
Total no abandonos	164155	69.6	10134	52.6
Total soluciones	235702	100	19297	100

Tabla 5.2: Distribución de ambos conjunto de datos luego de haber sido anotados de la segunda forma propuesta

Como se puede ver en la Tabla 5.1 considerando cada tipo de abandono como una clase, las proporciones se encuentran desbalanceadas lo que puede introducir un problema a la hora de entrenar modelos de aprendizaje automático como se explica en [39]. Por el contrario en la Tabla 5.2, donde se reportan los datos de las soluciones anotadas con el segundo método, podemos ver que las proporciones por cada tipo de abandono son mayores. Más aún si vemos la distribución de soluciones dentro del conjunto de datos obtenidos de Introducción a los Algoritmos (Intro Algo) podemos ver que considerando los dos tipos de abandonos en conjunto, la clases de Abandono y No Abandono tienen casi la misma proporción. En el Capítulo 6 veremos el impacto que tiene esto a la hora de construir el clasificador. A continuación, en la siguiente sección discutiremos las diversas características que utilizaremos para entrenar los modelos de aprendizaje automático. Es interesante notar que los abandono por cambio de ejercicio son el tipo más frecuente de abandono. Esto puede motivar que la acción a tomar al momento de detectar un riesgo de abandono sea sugerir un ejercicio más apropiado para ese estudiante.

5.2. Construcción de características

Luego de haber anotado ambos conjunto de datos surge la necesidad de encontrar las características que modelan cuándo el estudiante está en un estado de posible abandono del ejercicio. Para ello consideraremos características de dos dimensiones relevantes para este problema: características del estudiante y características del ejercicio. En esta sección proponemos cómo representar ambas dimensiones extrayendo información automáticamente a partir de los conjuntos de datos descriptos en los Capitulos 3y 4.

De acuerdo con Papert y Turkle [73] podemos dividir los estudiantes en dos tipos, *tinkerer* y *planner*. Esta clasificación surge a partir del comportamiento de los estudiantes a la hora de resolver ejercicios de programación. Por un lado, los estudiantes clasificados como *tinkerers* hacen muchos cambios incrementales a través de prueba y error con el objetivo de crear una solución final válida. Mientras que los estudiantes *planners* identifican un camino de acción y luego lo implementan con el objetivo de llegar a la solución que consideran válida.

Considerando esta división realizada por Papert y Turkle, nos interesa construir características que permitan modelar el tipo de estudiante y su comportamiento. Para ello hipotetizamos que aspectos como la experiencia del estudiante, la frecuencia con la que envía soluciones, entre otras, serán relevantes para esta tarea.

Además de la dimensión de estudiante, donde se hace foco en el estudiante y sus características como individuo, nos interesa analizar la dimensión de los ejercicios. Es decir, ver las particularidades de cada uno de ellos, por ejemplo el nivel de dificultad, la cantidad de veces que

fue resuelto exitosamente, entre otras cosas.

A continuación se realiza una descripción de cada una de ambas dimensiones en profundidad.

5.2.1. Dimensión estudiante

Para poder capturar el comportamiento de los estudiantes, se tienen en cuenta 3 niveles que representan su comportamiento al momento de programar: llamamos a cada uno de ellos nivel de experiencia, nivel de abandono y nivel de insistencia. A continuación presentamos cada uno de los niveles, y las características que los representan.

Nivel de Experiencia

El nivel de experiencia intenta capturar la experiencia en programación del estudiante en base a las soluciones enviadas previamente definiendo diversas caracteristicas. Al explicar cada una de las características, para poder ejemplificarlas, usaremos un conjunto de datos de muestra inventado para mantener la confidencialidad de los datosy sobre él se realiza el cálculo de cada una de ellas. En la Tabla 5.3 se muestra una porción del conjunto de datos, sólo se muestran las columnas involucradas en el cálculo de las características correspondientes a este nivel para facilitar la lectura.

id	created_at	student.email	exercise.name	Status
1	1/1/18 17:13:23	st1@mail.com	calcular	Failed
2	1/1/18 17:13:30	st1@mail.com	calcular	Passed
3	2/1/18 17:16:23	st1@mail.com	esBisiesto	Passed
4	4/1/18 17:20:20	st1@mail.com	pinos	Errored
5	4/1/18 17:20:25	st1@mail.com	pinos	Failed

Tabla 5.3: Ejemplo de porción del conjunto de datos inventado para ejemplificar el cálculo de las métricas correspondiente a la experiencia

- **Promedio sin aplazos:** sean $\{e_1, .., e_n\}$ los ejercicios resueltos de forma **correcta** por el estudiante. Sean $\{s_1, ..., s_n\}$, las cantidades de soluciones que el estudiante necesitó para resolver cada ejercicio uno de los n ejercicios. Definimos promedio sin aplazos como:

$$PSA = \frac{n}{\sum(s_1, .., s_n)}$$

Como podemos ver en la Tabla 5.3 el estudiante con email `st1@email.com` envió 5 soluciones dentro del sistema. Estas soluciones están distribuidas en 3 ejercicios intentados: `esBisiesto`, `calcular` y `pinos`. Hay 2 ejercicios que se resolvieron correctamente (es decir, su status es passed): `calcular` y `esBisiesto`. Para resolver correctamente `calcular` el estudiante necesitó realizar 2 soluciones. Para resolver `esBisiesto` solo le hizo falta 1. Por lo tanto para el estudiante `st1@email.com`, el PSA sería

$$PSA(st1@mail.com) = \frac{2}{2+1} = 0{,}66$$

Notar que PSA siempre será un número entre 0 y 1 porque todo estudiante necesitará enviar al menos una solución para resolver de forma correcta un ejercicio. Si un estudiante, siempre resuelve los ejercicios de forma correcta en su primer intento, su PSA será 1.

- **Promedio con aplazos**: sean $\{e_1, .., e_m\}$ los ejercicios **intentados** por un estudiante. Un ejercicio se define como intentado por el estudiante j si existe registro de al menos una solución enviada para ese ejercicio por el estudiante j. n es la sumatoria de la cantidad total de soluciones enviada por el estudiante para intentar resolver los m ejercicios. Definimos promedio con aplazos de la siguiente manera:

$$PCA = \frac{m}{n}$$

En la Tabla 5.3, podemos ver que el estudiante intentó resolver tres ejercicios diferentes para los cuales envió un total de cinco soluciones. Por lo tanto el valor del Promedio con aplazos es el siguiente:

$$PCA(st1@mail.com) = \frac{3}{5} = 0{,}6$$

Nivel de Abandono

El Nivel de Abandono tiene como objetivo representar si es un estudiante que abandona frecuentemente los ejercicios. Además intenta modelar bajo que condiciones decide abandonar un ejercicio. Para ello definimos tres caracteristicas, proporción de abandonos, cantidad de ejercicios abandonados y proporción ponderada de abandonos. Del mismo modo que hicimos con el nivel de experiencia luego de definir cada una de las características usaremos un conjunto de datos de muestra para ejemplificar el cálculo de cada una de ellas.

En la Tabla 5.4 se muestra una porción del conjunto de datos, sólo se muestran las columnas involucradas en el cálculo de las características correspondientes a este nivel para facilitar la lectura. El conjunto de datos está anotado con la segunda forma de anotar presentada en la Sección 5.1. Este pequeño conjunto de datos inventado incluye a un estudiante, a su intento por resolver 3 ejercicios, un abandono por cambio de ejercicio y un ababndono en sesión.

id	created_at	Time Dist	student email	exercise name	Status	Abandono
1	1/1/18 17:13:23	98293	st1@mail.com	calcular	Failed	No
2	1/1/18 17:13:30	7	st1@mail.com	calcular	Passed	No
3	2/1/18 17:16:23	86573	st1@mail.com	esBisiesto	Errored	Si
4	2/1/18 17:16:30	7	st1@mail.com	esBisiesto	Errored	Si
5	2/1/18 17:20:20	230	st1@mail.com	Pinos	Errored	Si
6	2/1/18 17:20:25	5	st1@mail.com	Pinos	Failed	Si
7	3/1/18 17:20:25	86400	st1@mail.com	Pinos	Failed	No
8	3/1/18 17:20:30	5	st1@mail.com	Pinos	Passed	No

Tabla 5.4: Soluciones enviadas por el mismo usuario para ejemplificar el cálculo de las características de nivel de Abandonos

A continuación presentamos las diferentes métricas que permiten capturar el nivel de abandono de los estudiantes.

- **Proporción de soluciones abandonas (PA)**, es de interés modelar que proporción de abandonos tienen en su trayectoria el estudiante dentro del sistema para ello se calcula la cantidad de soluciones marcadas como abandono sobre soluciones enviadas.

La Tabla 5.4 muestra las soluciones enviadas por el estudiante `st1@mail.com`, como podemos ver el estudiante envió en total ocho soluciones de las cuales cuatro están anotadas como abandono por lo tanto la característica proporción de abandonos se cálcula de la siguiente manera.

$$PA(st1@mail.com) = \frac{4}{8} = 0{,}5$$

- **Proporción de ejercicios abandonados (EA)**, se considera un ejercicio abandonado aquel que al menos tiene una solución marcada como abandono, sobre cantidad de ejercicios intentados. A diferencia de la característica anterior, esta característica se calcula a nivel de ejercicio y no de solución.

  Como se puede ver en la Tabla 5.4 el estudiante intentó tres ejercicios `calcular`, `esBisiesto` y `Pinos`. De los cuales `esBisiesto` y `Pinos` tienen al menos una solución anotada como abandono, por lo tanto estos dos ejercicios son considerados como abandonados al menos una vez, a pesar que `Pinos` lo logró solucionar luego en una sesión nueva. Notar que la línea que separa las soluciones 6 y 7 está denotando un cambio de sesión. Por lo tanto la característica de cantidad de ejercicios abandonados se calcula de la siguiente manera,

$$EA(st1@mail.com) = \frac{2}{3} = 0{,}66$$

Nivel de insistencia

A través del Nivel de Insistencia queremos capturar la intensidad del estudiante al momento de enviar soluciones al no poder resolver un ejercicio puntual. Analizaremos el comportamiento del estudiante dentro del sistema para generar esta característica, para ello consideraremos todas las soluciones enviadas por estudiante. Y definiremos tres características, promedio de tiempo transcurrido entre soluciones consecutivas, promedio de distancia de Levenshtein [56] e insistencia ponderada por abandono. A continuación se presenta y se ejemplifica cada una de las características a partir de un extracto conjunto de datos. El cual es presentado en la Tabla 5.5

id	created_at	Dist tiempo	Dist Lev	student email	exercise name	Cant soluc	Status
1	1/1/18 17:13:23	98293	24	st1@mail.com	calcular	1	Failed
2	1/1/18 17:13:30	7	5	st1@mail.com	calcular	2	Passed
3	2/1/18 17:16:23	86573	80	st1@mail.com	esBisiesto	1	Errored
4	2/1/18 17:16:25	2	4	st1@mail.com	esBisiesto	2	Errored
5	2/1/18 17:16:30	5	2	st1@mail.com	esBisiesto	3	Errored
6	2/1/18 17:20:20	230	40	st1@mail.com	pinos	1	Errored
7	2/1/18 17:20:25	5	2	st1@mail.com	pinos	2	Failed
8	3/1/18 17:20:25	86400	1	st1@mail.com	pinos	3	Failed
9	3/1/18 17:20:30	5	2	st1@mail.com	pinos	4	Errored
10	3/1/18 17:20:50	20	20	st1@mail.com	pinos	5	Failed
11	3/1/18 17:21:00	10	2	st1@mail.com	pinos	6	Passed

Tabla 5.5: Soluciones enviadas por el mismo usario para ejemplificar el cálculo de las métricas correspondiente a la Insistencia

- **Promedio de tiempo transcurrido (PTT)**, con el objetivo de medir con qué frecuencia el estudiante envía sus soluciones mediante el sistema dentro de una sesión, calcularemos

el promedio de tiempo transcurrido entre soluciones consecutivas (PTT), dentro de todas las sesiones llevadas a cabo por el estudiante, sin considerar el tiempo de inactividad, es decir el tiempo entre sesiones.

En la Tabla 5.5 la columna `Dist tiempo` corresponde al tiempo transcurrido entre la solución previa y la que corresponde a esa entrada en el conjunto de datos, el valor está expresado en segundos. Por lo tanto el promedio de tiempo transcurrido entre soluciones consecutivas para este estudiante es el siguiente,

$$PTT = (\frac{7+2+5+230+5+5+20+10}{8}) = 35,5$$

Notar que para el cálculo de PTT no se incluyó el tiempo correspondiente a las soluciones con id 1, 3 y 8. Ya que ellas son las primeras soluciones de cada sesión y ese tiempo es el transcurrido entre sesiones. Tiempo en el que el estudiante no estuvo trabajando en el sistema. El denominador de este promedio será la cantidad de soluciones menos la cantidad de sesiones, para este caso el estudiante envió 11 soluciónes en 3 sesiones, por lo tanto el denominador es 8.

- **Promedio distancia de Levenshtein (PDL)**, Intentando capturar que tanto piensa la solución antes de ser enviada calcularemos el promedio distancia de Levenshtein [56] entre el contenido de soluciones sobre un mismo ejercicio, para todas las sesiones realizadas por el estudiante.

  La columna Dist Lev en la Tabla 5.5 muestra la distancia de Levenshtein [56] entre soluciones del mismo ejercicio. En las primeras soluciones de cada ejercicio, soluciones con id 1, 3 y 5, la distancia de Levenshtein se calcula contra una solución vacía lo cual permite ver que tan grande es la primer solución enviada. Luego las consecutivas soluciones enviadas para el mismo ejercicio la comparación se realiza contra la inmediata anterior. De esta forma el valor de PDL se calcula de la siguiente forma,

$$PDL(st1@mail.com) = (\frac{24+5+80+4+2+40+2+1+2+20+2}{11}) = 16,54$$

- **Insistencia Ponderada por Abandono (IPA)**, con el objetivo de intentar capturar cuán insistente es el estudiante x cada vez que comienza a resolver un ejercicio construimos la siguiente característica, insistencia ponderada por abandono (IPA). Sean $\{e_1, .., e_n\}$ los ejercicios que intento resolver un estudiante. Sean, $\{t_1, .., t_n\}$ la cantidad de sesiones que el estudiante realizó para ese ejercicio. Sean $\{s_1, .., s_n\}$ la sumatoria de soluciones realizadas por cada vez que intento resolverlo llegando a ese estado de finalización por ejercicio. Definimos nuestra métrica se la siguiente manera,

$$\frac{\sum \frac{t_1}{s_1} .... \frac{t_n}{s_n}}{n}$$

En la Tabla 5.5 podemos ver que el estudiante intentó completar 3 ejercicios, `calcular`, `esBisiesto` y `pinos`. Para el ejercicio `calcular` llegó a un estado de finalización correcto, luego para el ejercicio `esBisiesto` abandonó por cambio de ejercicio. Por lo tanto para cada uno de esos ejercicios tiene un estado de finalización. Luego para `Pinos` tiene dos estados de finalización, el primero es cuando abandona en sesión y luego en la última solución cuando logra resolverlo. De modo que el cálculo de la característica es el siguiente,

$$IPA(st1@mail.com) = \frac{(\frac{1}{2} + \frac{1}{3} + \frac{2}{6})}{3} = 0,38$$

5.2.2. Dimensión ejercicio

En esta dimensión se trabaja con el objetivo de encontrar características propias de los ejercicios que permitan generar información relevante para poder tener una medida de la dificultad del ejercicio. A continuación se presenta cada una de las características que componen el nivel de dificultad y el nivel conceptual.

Nivel Conceptual

Este nivel tiene como objetivo representar que concepto es el que se trabaja en el ejercicio. En el sistema Mumuki como vimos en el Capitulo 3 cada ejercicio trabaja un concepto en particular como puede ser recursión, composición de funciones, entre otros. Para ello utilizaremos como característica del nivel conceptual el identificador único de ejercicio que usa el sistema internamente (Exercise id).

Nivel de Dificultad

En este nivel se intenta representar que tan dificultoso es cada uno de los ejercicios a partir del desempeño de los estudiantes en los mismos. Para ello se definen cuatro características, promedio de cantidad de soluciones para aprobar (PCSA), abandonos por estudiante (APE) y cantidad de abandonos por ejercicio (CAPE) y completitud (COMP). Para ejemplificar las definiciones y facilitar la comprensión de cada una de las características usaremos un extracto del conjunto de datos correspondiente al ejercicio `calcular` el cual se puede ver en la Tabla 5.6. Para este ejemplo estamos usando la segunda forma de anotar abandonos.

Student email	Exercise Name	Status	Abandono	Exercise id
e1@mail.com	calcular	Failed	Si	1
e1@mail.com	calcular	Failed	Si	1
e2@mail.com	calcular	Failed	No	1
e2@mail.com	calcular	Passed	No	1
e3@mail.com	calcular	Passed	No	1
e4@mail.com	calcular	Errored	Si	1
e4@mail.com	calcular	Errored	Si	1
e4@mail.com	calcular	Errored	Si	1

Tabla 5.6: Soluciones enviadas correspondientes al ejercicio *calcular* para ejemplificar el cálculo de las métricas correspondiente a la dificultad del ejercicio

- **Promedio de cantidad soluciones para aprobar (PCSA)**, con esta característica se intenta ver qué cantidad de soluciones son necesarias para completar con éxito cada ejercicio. Para ello se consideran todas las trayectorias que finalizaron con el ejercicio evaluado en verde y se calculará el promedio de la cantidad necesarias de soluciones enviadas para lograr resolver ese ejercicio.

  En la Tabla 5.6 podemos ver que los estudiantes `e2@mail.com` y `e3@mail.com` son quienes completaron el ejercicio `calcular` de modo correcto, necesitando dos y una solución respectivamente. Por lo tanto el promedio de cantidad soluciones para aprobar del ejercicio `calcular` queda de la siguiente forma,

$$PCSA(calcular) = \frac{2+1}{2} = 1{,}5$$

- **Abandonos por estudiante (APE)**, una forma de ver la dificultad del ejercicio puede ser viendo cuántas soluciones realizan los estudiantes antes de abandonarlo para ello se calcula la proporción entre el número de estudiantes que abandonaron el ejercicio sobre la cantidad de soluciones marcadas como abandono. Esta característica intenta modelar cuantas soluciones en promedio envía el estudiante intentando solucionarlo hasta que decide abandonarlo.

  Para el ejercicio que estamos usando de ejemplo sólo dos estudiantes lo abandonaron, el estudiante `e1@mail.com` quien envió dos soluciones antes de abandonarlo definitivamente y el estudiante `e4@mail.com` quien envió tres solucionarlo antes de abandonarlo. Habiendo observado estos datos podemos calcular la característica de abandonos por estudiante de la siguiente forma,

$$APE(calcular) = \frac{2+3}{2} = 2{,}5$$

- **Cantidad de abandonos por ejercicio (CAPE)**, en esta característica nos interesa analizar la distribución de las soluciones enviadas para el ejercicio. En particular nos interesa ver la proporción de soluciones etiquetadas como abandono sobre cantidad de soluciones enviadas. De este modo si esa proporción está cerca de 1 implica que sería un ejercicio que ha sido abandonado frecuentemente.

  Como se puede ver en la Tabla 5.6 se registraron ocho soluciones para el ejercicio `calcular` de las cuales cinco están anotadas como abandono, por lo tanto la característica CAPE se calcula de la siguiente forma,

$$CAPE(calcular) = \frac{5}{8} = 0{,}625$$

- **Completitud (COMP)**, con esta característica nos interesa modelar cuántos estudiantes pudieron completar el ejercicio. Para ello se calcula la proporción de estudiantes que completaron con éxito el ejercicio, es decir tiene al menos una solución en verde para ese ejercicio, sobre la cantidad estudiantes que intentaron el ejercicio.

  Como vimos previamente, sólo dos de cuatro estudiantes que intentaron el ejercicio pudieron completarlo satisfactoriamente. Considerando eso podemos calcular la característica completitud de la siguiente manera,

$$COMP(calcular) = \frac{2}{4} = 0{,}5$$

Hasta este momento se definieron dos dimensiones que son de interés para explorar. Dentro de cada una de esas dimensiones se hipotetizan diferentes características las cuales pueden ayudar a modelar el trayecto de un estudiante. A continuación en la Sección 5.3 se presentan los modelos de aprendizaje automático donde se pondrán a prueba esas características.

5.3. Modelos de aprendizaje automático a utilizar

En las secciones previas se definieron dos formas de anotar el conjunto de datos y se definieron un conjunto de características construidas en base a la información que posee cada una de las

entradas del conjunto de datos. Para poder medir cuáles de estas son más representativas en la tarea de predecir deserción de los estudiantes en ejercicios de programación se construirá un baseline con el cual comparar.

Cada una de las características serán utilizadas para entrenar un modelo de Regresión Logística. Se elige este modelo porque es rápido de entrenar para clasificación mientras que su desempeño es comparable con el de modelos computacionalmente más costoso como *support vector machines* (SVM). Este modelo será utilizado para comparar el desempeño de cada característica presentada previamente en esta tarea. Y luego se compararán estos resultados contra una implementación de una red neuronal recurrente LSTM *Long short-term memory*. A continuación en la Subsección 5.3.1 se explica el modelo de Regresión Logística y luego en la Subsección 5.3.2 se explica brevemente este tipo de redes neuronales y la estructura de la red que se utilizará.

5.3.1. Modelo de regresión logística

A continuación se realiza una breve explicación del modelo de aprendizaje automático de regresión logística. Un modelo de regresión logística tiene una variable dependiente que tiene dos posibles valores que pueden ser éxito/falla, ganar/perder o en nuestro caso abandono/no abandono; esas son representadas habitualmente como 0 y 1 o True y False. En este tipo de modelos, la función objetivo es una combinación lineal de una o más variables independientes (características), las variables independientes pueden ser valores binarios o una variable continua.

Entonces el modelo de regresión logística puede verse de la siguiente manera Dada $X = (x_1, x_2, \ldots, x_n)$ con $X \in \mathbb{R}^n$ queremos calcular $\widehat{y} = p(y = 1|x)$ con $0 \leqslant \widehat{y} \leqslant 1$. Los parámetros del modelo será un vector de la misma longitud que el vector de características X. $\omega = (\omega_1, \omega_2, .., \omega_n)$ con $\omega \in \mathbb{R}^n$. Hasta el momento tenemos definido lo siguiente:

- $X = x_1, x_2, x_3, x_4, \ldots, x_n$: Vector de variables independientes, este vector se construye para cada dato de entrenamiento con los resultados de las características descritas en la Sección 5.2.

- $\omega_1, \omega_2, \ldots, \omega_n$: parámetros, miden la influencia que las variables explicativas tienen sobre la regresión. Cuando algún $\omega_i = 0$ significa que esa variable independiente se está desestimando. Estos parámetros son los que debe aprender el modelo.

Entonces dado el vector X de entrada, los parámetros ω y b (término de bias), generamos $\hat{y}$ de la siguiente manera: $\widehat{y} = \sigma(\omega^T X + b)$

Notar que en la ecuación aparece una función $\sigma(z)$ la cual se describe a continuación

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Es importante la utilización de la función $\sigma(z)$ ya que es la encargada de mantener la condición que $0 \leqslant \widehat{y} \leqslant 1$. Para poder evaluar las características construidas previamente sobre un modelo lineal. Implementaremos un modelo de *Regresión Lineal* provisto por la librería scikit-learn[1]. El clasificador se construyó con los siguientes parámetros[2].

El clasificador será entrenado durante 1 millón de iteraciones o hasta que se cumpla el criterio de parada para la tolerancia del error el cual es 10^{-6}. Y se utilizará la métrica F1 para medir el desempeño del clasificador, la métrica se explica en la Sección 5.3.3. Para asegurar que el desempeño del clasificador no depende de la partición del conjunto de datos luego de entrenar el

[1] https://scikit-learn.org/

[2] C:1, penalty:l2, solver:liblinear, tol:1e-6, class_weight:balanced, max_iter:1e6

clasificador se realizará una validación cruzada utilizando el algoritmo de *k-fold cross validation* el cual se explica en la Sección 5.3.4.

5.3.2. Modelo neuronal

A continuación se realiza un breve descripción de las redes neuronales que utilizan unidades Long Short-term memory (LSTM). Antes de entrar en profundidad sobre las unidades LSTM, primero hablaremos de las Redes Neuronales Recurrentes. Este tipo de redes llamadas Redes Neuronales Recurrentes (RNN) son un tipo particular de red neuronal, las RNN tienen la característica que sus unidades están conectadas de modo que forman un grafo dirigido a lo largo de una secuencia. Esto le permite exhibir un comportamiento dinámico para una secuencia de tiempo. A diferencia de las redes feedforward, RNNs pueden usar su estado interno (memoria) para procesar secuencias. Esto hace que este tipo de redes sean útiles para tareas en las cuales el orden dentro de la secuencia importa como por ejemplo el reconocimiento de voz [50] o el reconocimiento de texto escrito a mano [34], entre otras aplicaciones.

Long Short-term memory (LSTM) son unidades de las RNN, habitualmente las redes compuestas por unidades LSTM son llamas redes LSTM. Las unidades LSTM están compuestas por una celda, una puerta de entrada, una puerta de salida y una puerta de olvido. La unidad recuerda valores por intervalos de tiempo, las tres puertas son las encargadas de regular el flujo de información que entra y sale de las unidades.

Este tipo de redes son utilizadas con diferentes objetivos como son clasificación, procesamiento y predicción siempre basado en series de tiempo. Para cada tipo de uso existen diversas formas de estructurar la red. En particular para este trabajo es de interés construir una red que siguiendo la arquitectura conocida como many-to-one que se ilustra en la Figura 5.2 a continuación se explica el diseño de la red.

El experimento consiste en entrenar un modelo neuronal con el contenido de las soluciones enviadas por los estudiantes, es decir el código generado. Esta técnica, de usar como característica principal el contenido de las soluciones, es diferente a la que se propone en la Sección 5.2 que utiliza caracteristicas contextuales pero no el programa escrito por el estudiante. Recientemente Wang et. al [74] propusieron usar LSTM directamente sobre los programas escritos por estudiantes para representar el conocimiento de los estudiantes de programación. El modelo que proponemos en la Figura 5.2 está inspirado en este trabajo. A diferencia de [74] nosotros usamos el texto plano del código mientras que Wang et. al usan los Abstract Syntax Trees (AST) del programa. Wang et al puede usar los AST porque los programas están escritos en lenguajes de bloques y por lo tanto son sintácticamente correctos por construcción. En estos lenguajes los errores de sintaxis no son posibles. En cambio, nuestro dataset está en Haskell y escrito por estudiantes principiantes donde aproximadamente el 40 % de los errores son de sintaxis. Otra diferencia es que nosotros alimentamos cada embedding con un token somo suele hacerse en el área de procesamiento de lenguaje natural mientras que ellos alimentan cada embedding con un programa completo. Para ellos es posible hacerlo dado que la cantidad de programas es finita. En nuestro caso es infinito. Para ello construiremos una red recurrente con unidades LSTM. Cada una de las soluciones serán tokenizadas utilizando un vocabulario de 35000 tokens. Luego de que la solución es tokenizada se la rellena (padding) de modo que resulten en vectores de la misma longitud. Luego los tokens de estos vectores son representados por embeddings. Acá probaremos tres tamaños de representación, embeddings de 128, 256 y 512 dimensiones. Esta elección es exploratoria ya que a priori no sabes que tamaño de embeddings pueden brindarnos buen desempeño. Estos embeddings serán la entrada de una capa de 100 unidades LSTM. Esta capa LSTM estará conectada a una capa con una función de activación sigmoide ya que como

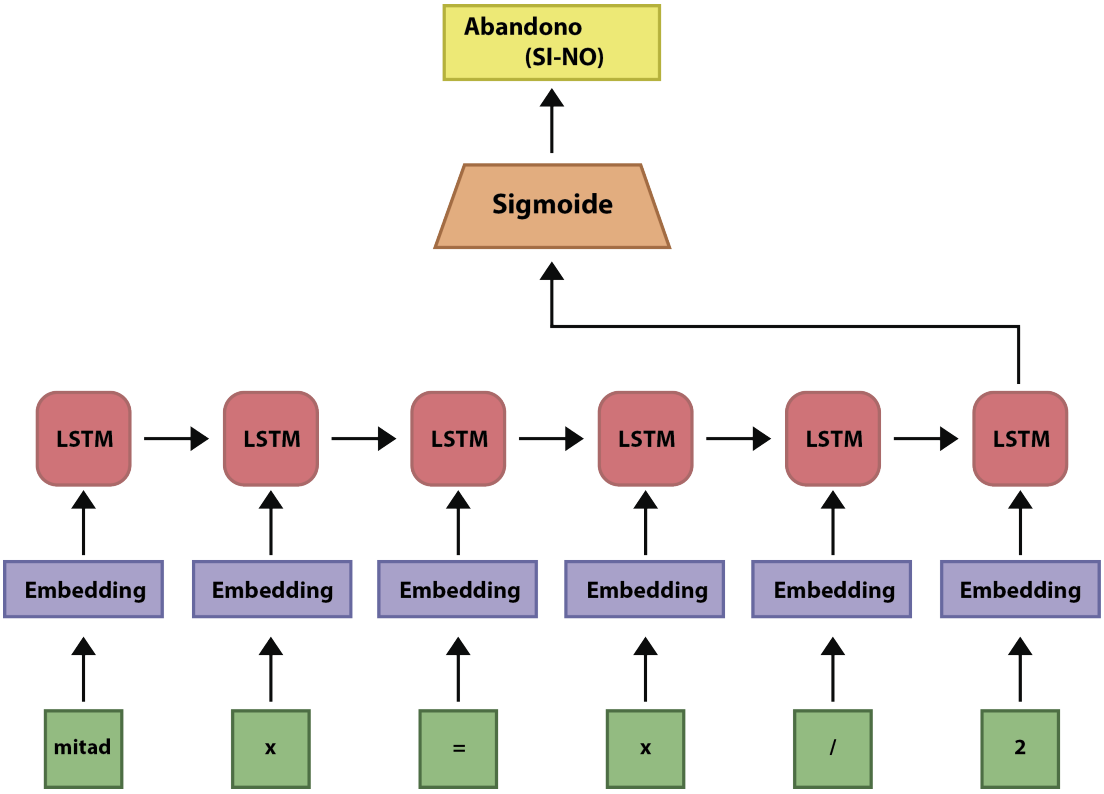


Figura 5.2: Red neuronal recurrente arquitectura Many to one

vimos previamente este es un problema de clasificación. Entonces necesitamos que la salida de la red sea 0 o 1 indicando si es un abandono o no. En la Figura 5.2 se muestra una representación de la entrada de la red y la estructura de la misma. En el próximo capitulo explicamos que esta red puede ser enriquecida con las caracteristicas definidas en la Sección 5.2. Para poder medir el desempeño de la red y que sea comparable con el modelo de regresión logística utilizaremos la métrica F1 la cual se explica en detalle en la sección siguiente.

5.3.3. Métricas de éxito

Para poder hacer una comparación entre las diferentes características utilizadas para entrenar los dos tipos de modelos se utiliza la métrica F1. Antes de profundizar en la métrica F1, debemos ver cuales son las posibilidades que se presentan a la hora de que el clasificador haga su trabajo sobre el conjunto de datos. Para ello en la Tabla 5.7 se representa la llamada matriz de confusión la cual se utiliza para medir el desempeño de un clasificador.

		Valor Real	
	Población total	Positivo	Negativo
Valor predicho	Positivo	Verdadero Positivo (VP)	Falso Positivo (FP)
	Negativo	Falso Negativo (FN)	Verdadero Negativo (VN)

Tabla 5.7: Matriz de Confusión

Acá podemos ver que cuando el clasificador marca a una solución como un posible abandono, pueden ocurrir dos cosas, o que esté en lo correcto (VP) o que sea un falso positivo (FP). En caso que el clasificador marque la solución como no abandonó osea su valor de verdad en Falso nuevamente pueden ocurrir dos cosas, que esté bien clasificado y sea Falso (VN) o que lo clasifique como Falso cuando debiera haberlo clasificado como Verdadero en ese caso es un Falso Negativo (FN). También podemos ver la matriz de confusión en la Figura 5.3 como un gráfico para facilitar su interpretación

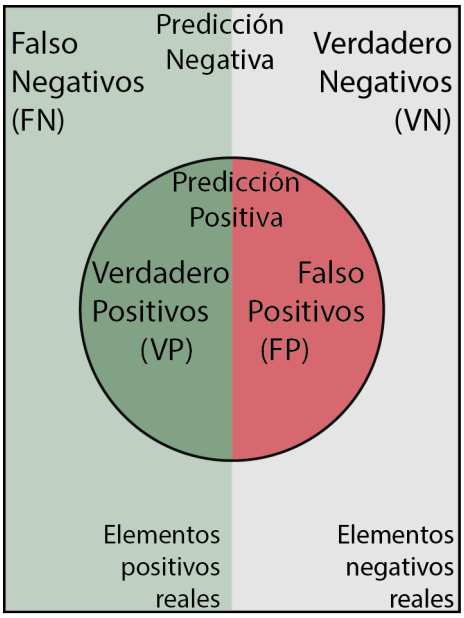


Figura 5.3: Matriz Confusión versión gráfica

A partir de estas posibilidades surgen dos métricas, **Precisión** y **Recall** las cuales nos permiten entender cómo se está comportando el clasificador. Estas métricas se definen de la siguiente manera.

- Precisión: La razón entre la cantidad de Verdaderos Positivos (VP) y la suma de ejemplos clasificados positivamente (VP + FP)
- Recall: La razón entre la cantidad de Verdaderos Positivos (VP) y todos los ejemplos que deberían clasificarse positivos (VP + FN)

Figura 5.4: Componentes de la métrica F1

Nuevamente para entender estas métricas pueden verse como en la Figura 5.4. Precisión es representada como una fracción entre los ejemplos Verdaderos positivos (Semicirculo verde) sobre los ejemplos predecidos como positivos (i.e Verdadero Positivo y Falso Positivo), esta métrica tiene por objetivo mostrar que tan preciso es el modelo, es decir de los ejemplos que clasificó como posivos, cuantos realmente lo son. Mientras que Recall está representado por una fracción

entre los valores Verdaderos positivos sobre los todos los elementos positivos reales. La métrica de recall representa que proporción de ejemplos positivos fueron clasificados como positivos.

Estas dos métricas dan lugar a la que utilizaremos que es F1 la cual sigue la siguiente fórmula.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

La métrica F1 es el promedio armónico de la precisión y recall, donde F1 alcanza su mejor valor en 1 y el peor en 0.

5.3.4. Validación cruzada

La técnica de validación cruzada o cross-validation es una técnica utilizada para evaluar los resultados de un modelo estadístico y garantizar qué son independientes de la partición entre datos de entrenamiento y prueba. Como podemos ver en la Figura 5.5 consiste en un procedimiento iterativo, en el cuál se calculan las métricas de evaluación sobre diferentes particiones, todas provenientes del mismo conjunto de datos. Los resultados de cada iteración son promediados en un resultado final y ésta será la métrica considerada para el modelo en cuestión. La cantidad de veces que se realiza la partición es parametrizada. Es ahí donde surge K-validación-cruzada o K-fold cross-validation donde los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se calcula el promedio de las métricas reportada para cada partición para obtener un único resultado. Este método es más confiable puesto que evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, pero aun así tiene una desventaja, y es que es lento desde el punto de vista computacional.

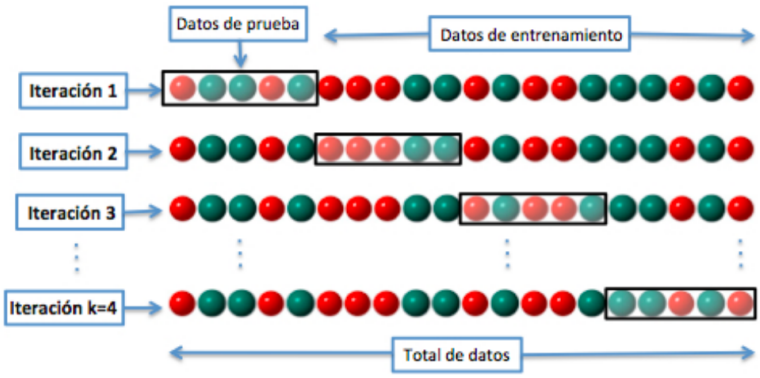


Figura 5.5: Procedimiento de validación cruzada con k iteraciones

Todos los resultados que se reportan en el Capitulo 6 que corresponden al modelo de regresión logística son reportados luego de realizar validación cruzada con diez iteraciones.

Hasta este momento hemos definido las características que serán parte de nuestro modelo. Además presentamos los modelos de aprendizaje automático que utilizaremos y su estructura junto con la métrica que usaremos para evaluar el desempeño. A continuación en el, Capitulo 6, se muestran los resultados obtenidos para cada uno de los experimentos. Luego en el Capitulo 7 discutiremos sobre los resultados obtenidos y hablaremos del trabajo futuro.

Capítulo 6

Resultados

Con el objetivo de poder predecir cuando un estudiante está en riesgo de abandonar un ejercicio en el sistema Mumuki, en el Capitulo 5 se propusieron diferentes experimentos. A lo largo de este capítulo se muestran los resultados obtenidos luego de haber realizado los mismos. Primero, en la Sección 6.1, discutiremos sobre la forma de anotar el conjunto de datos que elegimos entre las dos que propusimos previamente. En la Sección 6.2 se muestran los resultados de los baselines para cada una de las características propuestas. Luego de analizar cuáles son las características que mejoran el desempeño de los baselines, en la Sección 6.3, buscamos combinar algunas de ellas con el objetivo de seguir mejorando el desempeño del clasificador hasta lograr construir un clasificador independiente del conjunto de datos. Luego en la Sección 6.4 se muestra el desempeño obtenido con la red neuronal. Por último en la Sección 6.5 se realiza un análisis sobre los tiempos que demanda cada uno de los experimentos.

6.1. Anotación del conjunto de datos

En el Capitulo 5 se propusieron dos maneras diferentes de anotar los conjuntos de datos. Por un lado, la primer forma, solo considerando la última solución enviada como abandono y por otro lado, la segunda forma considerando la trayectoria del estudiante. En la Tabla 6.1 podemos ver la distribución de las clases luego de anotar cada uno de los conjunto de datos con las dos formas de anotar propuestas. Con el objetivo de poder decidir con cuál de estas dos formas se trabajará de ahora en más se realizó un experimento sobre el conjunto de datos de Intro Algo con ambas formas de anotación.

Recordemos que en el Capítulo 5 definimos el problema de predecir el abandono de estudiantes como una clasificación binaria considerando ambos tipos de abandono como el mismo. Es decir las clases de interés son, Abandono y No abandono. Como podemos ver en la Tabla 6.1 para la primera forma de anotar el conjunto de datos podemos ver que las clases quedan desbalanceadas. La clase de abandono representan el sólo el 9 % del total de soluciones para el conjunto de datos de Intro Algo. Esto puede generar el problema de clases desbalanceadas en aprendizaje supervisado [39]. Mientras que para la segunda forma de anotar la clase de abandono representa al rededor del 47 % de soluciones.

Para decidir con cual de las dos formas de anotar el conjunto de datos trabajaremos se entrenó un clasificador a partir de un modelo de *Regresión Lineal* provisto por la librería Scikit Learn[1]. Se entrenó un modelo de estos para cada forma de anotar y para el conjunto de datos de Into Algo. Para su entrenamiento se utilizó una característica, `submissions_count`. Este dato está

[1] https://scikit-learn.org/

	Intro Algo			
	Primera forma		Segunda forma	
	#	%	#	%
Abandonos en sesión	644	3.3	3382	17.5
Abanadonos por cambio de ejercicio	1086	5.6	5781	29.9
Total abandonos	1730	8.9	9163	47.4
Total no abandonos	17567	91.1	10134	52.6
Total soluciones	19279	100	19279	100

Tabla 6.1: Distribución de las clases para el conjunto de datos de Intro Algo, luego de haber sido anotados con cada una de las formas propuestas.

presente en cada solución registrada en el conjunto de datos. Esta hace referencia a que número de soluciones fueron enviadas previamente por ese estudiante para ese ejercicio. La tarea de este clasificador es predecir el abandono por parte de los estudiantes. Los resultados obtenidos para cada uno de los conjuntos de datos se reportan en la Tabla 6.2

	Intro Algo		Mumuki io	
	Primera forma	**Segunda forma**	**Primera forma**	**Segunda forma**
F1 score	0.91	0.58	0.94	0.67
Clase Mayoritaria	91.1	52.6	94.9	69.6

Tabla 6.2: Valores de métrica F1 obtenida para cada forma de anotar aplicada a cada conjunto de datos.

Como se puede ver en la Tabla 6.2 para ambos conjuntos de datos el valor de F1 para la primer forma de anotar no es mayor a la clase mayoritaria. Lo cual nos indeca que no está aprendiendo nada el clasificador. Lo que está haciendo el clasificador en este experimento es clasificar todas las soluciones como la clase mayoritaria, en este caso No Abandono. Por este motivo es necesario utilizar otra forma de anotar. Si observamos la columna de la segunda forma de anotar para Intro Algo podemos ver que el valor de la métrica F1 es mayor que la proporción de la clase mayoritaria lo cual nos da un indicio que el clasificador está aprendiendo, es más en este caso los ejemplos clasificados por el modelo no son todos No Abandonos.

Además de lo discutido recién, decidimos usar la segunda forma de anotar porque consideramos que pedagógicamente tiene más sentido clasificar a toda una sesión como abandono en lugar de hacerlo únicamente a la última solución de esa sesión

Teniendo en cuenta estos resultados, a partir de este momento utilizaremos la segunda forma de anotar el conjunto de datos y consideraremos las dos clases de abandonos como una sola. A continuación en la siguiente Sección 6.2 se muestran los resultados de los baselines que utilizaremos para luego medir el desempeño de cada característica propuesta en el Capitulo 5 intentando encontrar cuales son las características más significativas para modelar el comportamiento de los estudiantes.

6.2. Ingeniería de características

En el Capitulo 5 definimos dos dimensiones, estudiante y ejercicio, a partir de los cuales generamos diversas características con el objetivo de representar el comportamiento de los estudiantes en el sistema Mumuki. Primero entrenamos un un clasificador aleatorio (Dummy Classifier) el cual sera el limite inferior de desempeño para nuestras características. Luego de esto proponemos dos baselines sobre los cuales se medirá el desempeño de cada característica. Ambos baselines son modelos de Regresión Logística.

El primer baseline está construido con una características *submissions_ count*, este número representa la cantidad de soluciones enviadas previamente para ese ejercicio por ese estudiante, ya que suponemos que este es un indicador de posible abandono. Con este baseline queremos ver si podemos abstraernos del contenido de la solución del estudiante y aún así tener un buen desempeño para el clasificador.

El otro baseline que construimos tiene como característica principal el contenido de la solución del estudiante. Para poder utilizar el contenido de la solución que está escrita en Haskell, necesitamos tener una representación de la misma, para ello representaremos a las soluciones mediante un *bag of words*. Cada solución será transformada en una lista de tokens junto con la cantidad de ocurrencias de cada uno de esos tokens. La desventaja de este modelo, es se pierde el orden original de los tokens. Entonces no se logra mantener la estructura del programa enviado. Como veremos más adelante esto es diferente con una red LSTM.

A continuación en la Tabla 6.3 se muestra el desempeño de cada baseline propuesto para ambos conjuntos de datos, Mumuki io e Intro Algo. El desempeño se mide en base a la métrica F1 la cual se explicó en la Sección 5.3.3, para ello reportamos esta métrica luego de realizar una validación cruzada de diez iteraciones la cual se explicó en la Sección 5.3.4. De ahora en adelante para todos los experimentos realizados con el modelo de regresión lineal los valores que se reportan de la métrica F1 se reportan luego de la validación cruzada de diez iteraciones.

	F1 Score	
Modelo	**Intro Algo**	**Mumuki io**
Clasificador dummy	0.51	0.57
Submission count	0.58	0.67
Submission content	0.59	0.69

Tabla 6.3: Desempeño de los diferentes baselines para cada conjunto de datos

Como mencionamos previamente el objetivo de este experimento es encontrar cuales de las características definidas mejoran el desempeño de cada uno de los baselines. Como podemos ver en la Tabla 6.3 ambas características logran una mejora de desempeño sobre el clasificador Dummy.

Los baselines construidos con las características de las soluciones, *Submission count* y *Submission content*, serán los clasificadores sobre los cuales se pondrán a prueba las características propuestas en la Sección 5.2. Recordemos que definimos dos dimensiones, estudiante y ejercicio. Además dentro de cada dimensión definimos diversas características.

A continuación en la Sección 6.2.1 se muestran los resultados obtenidos para las características de la dimensión estudiante. Luego en la Sección 6.2.2 se muestran los resultados para la dimensión ejercicio. Estos resultados se reportan para ambos conjuntos de datos por separado.

6.2.1. Dimensión Estudiante

En esta sección se muestran los resultados para cada característica propuesta para la dimensión de estudiante, con el objetivo de encontrar cual modela de mejor manera la trayectoria, permitiendo predecir con mayor certeza cuando la posibilidad que un estudiante abandone el ejercicio.

Para la dimensión estudiante, donde las características hacen foco en el comportamiento del estudiante, definimos tres niveles: experiencia, insistencia y abandono. Para cada una de ellas, a continuación, se presentan los resultados.

Nivel Experiencia

En esta subsección se muestran los resultados para las características correspondientes al nivel de experiencia. Estas características intentan modelar que cantidad de experiencia previa tiene el estudiante en base a los ejercicios resueltos previamente. En la Tabla 6.4 se muestra el desempeño, reportando la métrica F1, para cada una de las características definidas para este nivel sobre ambos conjuntos de datos.

Nivel experiencia			
Baseline	**Característica**	**Intro Algo**	**Mumuki io**
Submission Count	PCA	**0.61**	0.64
	PSA	0.59	**0.65**
Content	PCA	0.62	**0.68**
	PSA	**0.63**	0.67

Tabla 6.4: Desempeño de las características del nivel de experiencia sobre ambos conjuntos de datos

Podemos ver a partir de los resultados resaltados que las características promedio con aplazos (PCA) y promedio sin aplazos (PSA) son las que mejoran el desempeño del clasificador.

Nivel Abandono

Las características correspondientes al nivel de abandono intentan modelar con que frecuencia un estudiante abandona el ejercicio. A continuación en la Tabla 6.5 se muestran los resultados sobre ambos conjuntos de datos.

Nivel Abandono			
Baseline	**Característica**	**Intro Algo**	**Mumuki io**
Submission Count	PA	**0.64**	**0.73**
	EA	0.62	0.72
Content	PA	**0.65**	**0.76**
	EA	0.62	0.73

Tabla 6.5: Desempeño de las características del nivel de abandono sobre ambos conjuntos de datos

Como se puede ver en la Tabla 6.5, sin importar el baseline ni el conjunto de datos donde se está entrenando el clasificador la característica de proporción de abandona (PA) es la que aumenta en mayor medida el desempeño del clasificador. Esto evidencia que esta característica tiene alto grado de importancia en la tarea que lleva a cabo el clasificador.

Nivel Insistencia

Recordemos que a través del Nivel de Insistencia queremos capturar la intensidad del estudiante al momento de enviar soluciones al no poder resolver un ejercicio puntual. En la Tabla 6.6 se muestran el desempeño de estas características sobre ambos conjuntos de datos.

Nivel Insistencia			
Baseline	**Característica**	**Intro Algo**	**Mumuki io**
Submission Count	PTT	**0.59**	**0.67**
	PDL	0.58	0.66
	IPA	0.58	0.66
Content	PTT	0.60	0.64
	PDL	**0.61**	0.63
	IPA	0.60	**0.67**

Tabla 6.6: Desempeño de las características del nivel de insistencia sobre ambos conjuntos de datos

Como podemos ver en los valores resaltados en la Tabla 6.6 las características de promedio de tiempo transcurrido (PTT) y el promedio de distancia de Levenshtein (PDL) son las que mejoran el desempeño del clasificador con en el baseline con *submission count* y *content* como primer característica respectivamente, dentro del conjunto de datos de Intro Algo.

Del mismo modo ocurre para el baseline con *submission count* como primer característica, sobre el conjunto de datos de Mumuki io, donde la característica PTT es la que mejora el desempeño del clasificador. Pero si consideramos el baseline con *content* como primer característica, la característica que aumenta el desempeño es la insistencia ponderada por abandono (IPA) como se puede ver en la Tabla 6.6.

6.2.2. Dimensión Ejercicio

Dentro de la dimensión de ejercicio, se propusieron características que permitan modelar el nivel de dificultad de cada ejercicio a partir de las soluciones enviadas para cada uno de los ejercicios. Además se propuso el nivel conceptual, usando como caracteristica principal el identificador del ejercicio. A continuación en la Tabla 6.7 se muestran el desempeño de las características, de ambos niveles, sobre ambos conjuntos de datos.

Nivel Dificultad			
Baseline	**Característica**	**Intro Algo**	**Mumuki io**
Submission Count	PCSA	0.62	0.65
	APE	**0.64**	0.65
	CAPE	0.62	0.63
	COMP	0.60	**0.67**
Content	PCSA	0.62	**0.65**
	APE	**0.63**	0.64
	CAPE	0.61	0.64
	COMP	0.60	0.63
Nivel Conceptual			
Submission Count	Exercise id	0.59	0.49
Content	Exercise id	0.59	0.60

Tabla 6.7: Desempeño de las características del nivel de dificultad y conceptual correspondientes a la dimensión ejercicio, sobre ambos conjuntos de datos

Como se puede observar en la Tabla 6.7 la característica abandonos por ejercicio (APE) la cual modela cuantas soluciones envía un estudiante antes de abandonar el ejercicio, es la que mejor desempeño tuvo sobre ambos baselines para el conjunto de datos de Intro Algo. Por otro lado en la Tabla 6.7 podemos ver que distintas características mejoraron el desempeño para los dos baselines. Por un lado la característica completitud (COMP) mejoró el desempeño sobre el baseline usando *submission count* como característica base. Mientras que el promedio de cantidad de soluciones para aprobar un ejercicio (PCSA) fue la característica que mejoró el desempeño utilizando el contenido de la solución como baseline.

6.3. Características: buscando la mejor combinación

En la Sección 6.2 se reportó el desempeño de cada característica para cada uno de los conjuntos de datos. En esta sección se trabaja en combinar aquellas características que mejor desempeño tuvieron sobre cada conjunto de datos y cada baseline. El principal objetivo de esto es encontrar cual es la mejor combinación de características para cada conjunto de datos. Por mejor combinación entendemos aquella que maximiza el desempeño del clasificador en la tarea propuesta.

A continuación en la Tabla 6.8 podemos ver de acuerdo a cada conjunto de datos cuales fueron las características que mejor funcionaron para cada baseline. Notar que sólo se seleccionó una característica por nivel de interés con el objetivo que la misma represente lo suficientemente bien el nivel al cual pertenece, las que están en negrita en las Tablas 6.4-6.7.

Conjunto de datos	**Intro Algo**		**Mumuki io**	
Baseline	Submission count	Submission content	Submission count	Submission content
Dimensión Estudiante				
Nivel Experiencia	PCA	PSA	PSA	PCA
Nivel Abandono	PA	PA	PA	PA
Nivel Insistencia	PTT	PDL	PTT	IPA
Dimesión Ejercicio				
Nivel Dificultad	APE	APE	COMP	PCSA
Mejor F1 logrado				
F1 Score	0.69	0.67	0.74	0.77

Tabla 6.8: Características con mejor desempeño según baseline, conjunto de datos, y nivel.

En la última fila de la Tabla 6.8 se reporta el valor de la métrica F1 para los modelos construidos con la combinación de las mejores características. Es decir, por ejemplo, para el conjunto de datos de Intro Algo, se entrenó sobre el baseline de Submission count la combinación de las siguientes características PCA + PA + PTT + APE. Y así para cada uno de los baselines de cada conjunto de datos, con las características que aparecen en la columna correspondiente a cada baseline.

De acuerdo a la Tabla 6.8 podemos ver que para la dimensión del estudiante dentro del nivel de abandono la característica proporción de abandonos (PA) es la que mejor desempeño tuvo en ambos baselines sin importar el conjunto de datos. Luego para el nivel de experiencia, hay dos características que emergen. El promedio con aplazos (PCA) y promedio sin aplazos (PSA) son los más representativos para esta dimensión. En cambio para el nivel de insistencia, el promedio de tiempo transcurrido entre soluciones (PTT), es la característica que mejor desempeño tuvo sobre el baseline con Submission count para ambos conjunto de datos. Pero si vemos el baseline con Submission content como primer característica, las características con mejor desempeño para ese nivel fueron dos, por un lado el promedio de distancia de Levenshtein (PDL) para el conjunto de datos de Intro Algo y el nivel de insistencia ponderada por abandonos (IPA) para el conjunto de datos de Mumuki io.

Por último para el nivel de dificultad correspondiente a la dimensión de ejercicio sobre el conjunto de datos de Intro Algo la características que mejor desempeño obtuvo fue el abandono ponderado por ejercicio (APE). Mientras que para el conjunto de datos de Mumuki io varió según el baseline. Las características con mejor desempeño fueron completitud (COMP) y promedio de cantidad de soluciones para aprobar (PCSA) sobre el baseline con Submission count y Submission content respectivamente.

Como se puede ver en la Tabla 6.8 para cada conjunto de datos y para cada baseline las características que mejores funcionaron en ciertos niveles coinciden pero en otros no. A partir de estos resultados surge la necesidad de encontrar cuales son las características que permiten maximizar el desempeño del clasificador y que nos permita abstraernos del conjunto sobre el cual se está entrenando para así poder generalizar el modelo.

Para ello a continuación realizaremos un *Ablation Study*, esta técnica consiste en ir quitando características del modelo con mayor desempeño, re-entrenar y medir el desempeño para tratar de identificar qué características aportan más cuando son combinadas. Realizaremos este análisis sobre el conjunto de datos de Intro Algo, porque sus clases se encuentran balanceadas como se puede ver en la Tabla 6.1. Además utilizaremos el baseline Submission count ya que

Baseline Submission Count	
Caracteristicas	**F1 Score**
PCA + PA + PTT + APE	**0.697**
PCA + PA + PTT	0.64
PCA + PTT + APE	0.65
PCA + PA + APE	0.694
PA + PTT + APE	0.697
PCA + PA	0.64
PCA + PTT	0.61
PCA + APE	0.65
PA + PTT	0.64
PA + APE	0.696
PTT + APE	0.64

Tabla 6.9: *Ablation study* sobre las características con mejor desempeño en el baseline Submission count para el conjunto de datos Intro Algo

es el que mejor desempeño tuvo para ese conjunto de datos como como se puede ver en la Tabla 6.8. Partiendo desde la mejor combinación obtenida y sucesivamente iremos quitando de a una característica por vez. En la Tabla 6.9 se reportan los resultados de este experimento

Como podemos ver en la Tabla 6.9 la característica proporción de abandonos (PA) correspondiente a la dimensión de estudiante, nivel de abandono, es una de las que más aporta a la tarea. Además el promedio de tiempo transcurrido (PTT) aparece como una de las características que aumenta el desempeño en el baseline. En lo que respecta a la dimensión de ejercicio, la característica abandonos por ejercicio (APE) es la que mejor desempeño obtuvo en el conjunto de Intro Algo.

Con el objetivo de construir un clasificador con características que permitan generalizar respecto del conjunto de datos con el que se trabaja, usaremos características que tuvieron buen desempeño para construir un clasificador que su desempeño no esté ligado al conjunto de datos. Para ello usamos como baseline Submission Count y usamos las características PA y APE. Ya que estas dos características juntas tienen un desempeño similar al mejor obtenido (i.e PCA + PA + PTT + APE, F1 0.697). A continuación en la Tabla 6.10 se muestra el desempeño de este modelo, dentro de cada uno de los conjuntos de datos.

	Intro Algo	**Mumuki io**
Submission Count + PA + APE	0.68	0.75

Tabla 6.10: Desempeño del modelo Submissions count + PA + APE, sobre ambos conjuntos de datos. Con el objetivo de construir un clasificador abstrayendonos del conjunto de datos.

Como se puede ver en la Tabla 6.10 el valor de la métrica F1 utilizando características de ambas dimensiones propuestas estudiante y ejercicio. Las cuales fueron comunes a los mejores modelos reportados en la Tabla 6.8 es menor que las mejores combinaciones. Pero logramos construir un modelo sin la necesidad de tener que usar las características especificas que mejor funcionaron para cada conjunto de datos. Si bien el desempeño es entre 1 y 2 puntos menor que los mejores modelos obtenidos previamente, utilizar combinación de estas características nos permitirá generalizar la solución. Por lo tanto creemos que este modelo será lo suficientemente robusto ante un nuevo conjunto de datos.

6.4. Una alternativa neuronal

Tal como se describió en la Capitulo 5 este experimento consta de entrenar una red neuronal recurrente construida de unidades LSTM con el objetivo de predecir si el estudiante abandonará el ejercicio o no. Este experimento está motivado por una desventaja que tiene la utilización de *bag of words* como la representación de la solución. Esta representación no respeta el orden de la aparición de los tokens. Teniendo en cuenta que en una solución enviada por un estudiante el orden de los tokens es fundamental esperamos que representando la solución como una secuencia ordenada de tokens el desempeño de la red en su función de clasificador mejore por sobre el desempeño obtenido previamente con el modelo de regresión logística.

Utilizando solamente el código enviado por los estudiantes como característica. Dado que la representación del contenido de la solución enviada será un vector ordenado de tokens, necesitamos primero realizar un análisis del contenido de las soluciones enviadas. Para ello calculamos la cantidad de tokens únicos para cada uno de los conjuntos de datos. Luego de tokenizar cada una de las soluciones se calculó la longitud de cada una de ellas y se calcula el percentil %99 siguiendo la metodología propuesta en [10]. El objetivo de utilizar el este percentil nos da la seguridad que sólo el %1 de las soluciones tendrán más tokens que esta longitud. A continuación en la Tabla 6.11 se muestran los resultados de este análisis.

	Intro Algo	**Mumuki io**
Cantidad de tokens únicos	4316	33316
Percentil %99 longitud de soluciones	100	96

Tabla 6.11: Análisis de las soluciones para la construcción de la entrada de la red

A partir de esta información, podemos construir la entrada para la red. Cada una de las soluciones será tokenizada con un vocabulario de 35000 tokens, ya que el conjunto de datos de Mumuki io cuenta con más de 33 mil tokens únicos. Cada una de las soluciones luego de ser tokenizadas, aquellas soluciones que tengan menos de 100 tokens, sus vectores serán rellenados con 0 de modo que todos los vectores sean de dimensión 100 (padding). Como vimos en el Capitulo 5, probaremos dos tamaños de *embeddings* los cuales serán la entrada para las 100 unidades de LSTM. Para cada uno de los conjuntos de datos se entrenará una red durante 20 épocas y se utilizará la métrica de éxito F1 para poder tener información que sea comparable con los experimentos previos. Para poder validar el desempeño de nuestro modelo neuronal partimos el conjunto de datos en dos partes, %80 de las soluciones para entrenamiento y %20 para validación.

A continuación en la Tabla 6.12 se muestran el valor de la métrica F1 al cabo de entrenar por 20 épocas cada una de las configuraciones de la red. La misma fue alimentada sólo con la solución generada por los estudiantes. El desempeño de la red se reporta sobre el conjunto de validación de cada uno de los conjuntos de datos.

	Intro Algo	**Mumuki io**
	F1 Score	
LSTM (Embeddings 128)	0.84	0.78
LSTM (Embeddings 256)	0.85	0.81
LSTM (Embeddings 512)	0.85	0.80

Tabla 6.12: Valores de F1 obtenidos para cada una de las configuraciones de la red.

Como podemos ver en la Tabla 6.12, el modelo neuronal mejoró sustancialmente su desempeño

respecto al modelo de mayor desempeño de regresión logística sobre el conjunto de datos de Intro Algo. Como se puede ver en la Tabla 6.8 el mejor desempeño se alcanzó con la combinación *Submission count + PCA +PA + PTT + APE* con un valor de F1 de 0.697 mientras que con la opción neuronal el valor de F1 alcanzado fue de 0.85 con embeddigngs de 256 de largo. A diferencia de lo que ocurrió con Mumuki io, donde no es tan grande la mejora ya que la mejor combinación para el modelo lineal fue *Submission Content + PCA + PA + IPA + PCSA* con un valor de F1 de 0.77 mientras que el mejor modelo neuronal alcanzó un valor de F1 de 0.81. Aún así es prometedor este resultado ya que la red sólo esta entrenada con el texto plano de la solución. A continuación en la Sección 6.5 discutiremos los costos de entrenar ambos modelos, lo cual nos permitirá elegir de mejor manera considerando el *trade off* entre tiempos de entrenamiento y desempeño.

6.5. Análisis de tiempos

En esta sección analizaremos cuanto tiempo demanda entrenar cada uno de los modelos con mejor desempeño que se construyeron hasta el momento. A pesar de que los modelos se entrenaron sobre diferentes tipos de Hardware. Los modelos neuronales se entrenaron sobre GPU mientras que los modelos de regresión logística fueron entrenados sobre CPU. Comparar los tiempos, nos brindará una idea de la diferencia de costos a la hora de entrenar los modelos. Los modelos neuronales se entrenaron sobre una GPU Nvidia Tesla K80, mientras que los modelos de regresión lineal fueron entrenados sobre un CPU Intel Xeon E5-2620. En la Tabla 6.13 se muestra el tiempo que demandó entrenar cada una de las redes durante veinte épocas, y los tiempos de entrenamiento para cada uno de los modelos lineales con mejor desempeño para cada conjunto de datos. Los tiempos se reportan en segundos.

	Intro Algo	**Mumuki io**
	Tiempo en segundos	
Modelo lineal mejor desempeño	0.41	72.16
LSTM (Embeddings 128)	1060	6000
LSTM (Embeddings 256)	1200	6240
LSTM (Embeddings 512)	1860	21460

Tabla 6.13: Tiempos de entrenamientos en segundos para cada una de las redes.

Como vimos en la Sección 6.3 la mejor combinación obtenida para el conjunto de datos de Intro Algo fue la combinación *Submission count + PCA +PA + PTT + APE* con un valor de F1 de 0.697 la cual demoró menos de medio segundo. Mientras que para el conjunto de datos de Mumuki io el mejor desempeño se obtuvo con *Submission Content + PCA + PA + IPA + PCSA* con un valor de F1 de 0.77 demoró más de un minuto para terminar de entrenarse. Con esto podemos ver que cada vez que se utiliza el contenido de la solución como característica para entrenar los modelos, el tiempo de entrenamiento crece considerablemente. Por otro lado podemos ver que el tiempo de entrenamiento para las redes neuronales aumenta a medida que crece el tamaño de los embeddings y el tamaño del conjunto de datos con el que se trabaja. En la Tabla 6.13 podemos ver que para el conjunto de datos de Intro Algo la red con embeddings de 128 demoró 17,6 minutos entrenarse, la red con embeddings de 256 demoró 20 minutos y por último la red con embeddings de 512 demoró 31 minutos. Ocurre del mismo modo sobre el conjunto de datos de Mumuki io, con los siguientes tiempos, 100 minutos para embeddings de 128, 104 minutos para embeddings de 256 y 357 minutos (cerca de seis horas) para embeddings

de 512.

Si bien el entrenamiento comúnmente se realiza de modo offline, entonces podemos tomar un experimento que demore más tiempo pero que mejore el desempeño no siempre el experimento que demore más es porque está mejorando su desempeño. Acá es importante notar que la red a medida que aumentamos el tamaño de los embeddings los tiempos de entrenamiento aumentan, pero no así la métrica F1. Como podemos ver en las Tablas 6.12y6.13 el mejor *trade-off* entre tiempo de entrenamiento y desempeño obtenido lo logramos con embeddings de 256 de largo.

Teniendo en cuenta que una de las particularidades de este trabajo es que el modelo propuesto puede ser integrado a modo de prueba en el sistema Mumuki, es importante que el tiempo de respuesta de la solución propuesta sea lo más chico posible. A continuación en la Tabla 6.14 se muestra cuanto tarda en clasificar una solución cada uno de los modelos.

	Intro Algo	**Mumuki io**
	Tiempo de predicción	
Mejor modelo con baseline Submission count	1 ms	2 ms
Mejor modelo con baseline Submission content	4 ms	16 ms
LSTM (Embeddings 128)	1.7 ms	1.7 ms
LSTM (Embeddings 256)	1.9 ms	1.8 ms
LSTM (Embeddings 512)	2.5 ms	2 ms

Tabla 6.14: Tiempos de predicción en milisegundos para cada una de los modelos con mejor desempeño.

Como podemos ver en la Tabla 6.14 a medida que crece el tamaño del conjunto de datos los modelos lineales demoran más en predecir el valor de cada solución. Mientras que para los modelos neuronales los tiempos de predicción sufren un pequeño aumento.

En el próximo Capitulo 7 discutiremos conclusiones sobre el trabajo realizado a lo largo de este trabajo en base a los resultados obtenidos. Además, propondremos diversos trabajos futuros que son de interés para seguir avanzando sobre esta temática

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

Este trabajo presentó un conjunto de datos novedoso sobre el cual todavía no se había trabajado. Propusimos dos formas de anotar el conjunto de datos automáticamente y formalizamos la tarea de modelar la trayectoria realizada por los estudiantes en el sistema de enseñanza online Mumuki. Estos modelos predicen en qué momento el estudiante está a punto de abandonar el ejercicio. Para ello se formalizaron dos dimensiones para construir características que permitan modelar la trayectoria del estudiante y así poder entrenar diversos modelos de aprendizaje automático y aprendizaje profundo que sean capaces de llevar adelante esa tarea de predicción. Logramos construir un modelo de aprendizaje automático que nos permite realizar esta tarea en un tiempo realmente corto, entre 1 y 16 milisegundos. Si la tarea de predecir abandonos fuera realizada por un humano sería prohibitivamente costosa, ya que el tiempo que demandaría y la cantidad de información que se debe procesar es muy grande. Esto nos permite realizar clasificación a gran escala, con un valor de F1 por encima de 80

Para el conjunto de datos de Introducción a los Algoritmos hemos logrado crear un modelo de aprendizaje automático con un alto desempeño sin utilizar el contenido de las soluciones enviadas por los estudiantes. Lo cual nos da una idea que para entornos de utilización del sistema donde está presente la ayuda de docentes, se pueden utilizar características del estudiante y del ejercicio para construir un modelo de aprendizaje automático donde el tiempo de entrenamiento es considerablemente menor a que si se utiliza el código enviado en las soluciones como característica. La característica abandonos por ejercicio (APE) fue una de las que aumentó considerablemente el desempeño sobre el conjunto de datos de Intro Algo, creemos que esto se debe a que los estudiantes están dentro de un sistema formal de educación entonces el comportamiento de ellos es homogéneo ante los ejercicios, además considerando que la mayoría de estudiantes que conformaron el curso de Introducción a los Algoritmos de 2018, eran principiantes en programación.

No funcionó de igual manera en el conjunto de datos no controlado. Sobre el conjunto de datos de Mumuki IO, fue de vital importancia considerar el contenido de la solución como una característica principal. Para este conjunto de datos, el mejor clasificador lineal que pudimos construir tuvo como base el contenido de la solución enviada por el estudiante. Características como PA e IPA fueron algunas de las que mejoraron el desempeño del clasificador, poder medir que tan insistente es el estudiante antes de abandonar un ejercicio, esto es capturado por IPA. Y contar con la característica de cuántas soluciones tiene como abandonos (PA), mejoraron considerablemente el desempeño ya que la experiencia previa usuarios del sistema Mumuki que no se encuentran dentro del sistema formal de educación es heterogénea.

Luego de construir clasificadores para cada uno de los conjuntos de datos por separados encontramos una combinación tal de características que nos permitieron construir un clasificador independiente del conjunto de datos con el que se está trabajando. Hemos logrado definir características pedagógicamente motivadas que permitieron construir modelos de aprendizaje automático con buen desempeño en la tarea. Esta es una perspectiva diferente al trabajo previo en el área sobre qué ver en los datos además del contenido generado por el estudiante.

Los experimentos con la red neuronal tuvieron una gran mejoría de desempeño sobre ambos conjuntos de datos sin necesidad del diseñar características específicas. El desempeño en Mumuki.io no fue tan alto como en el conjunto de datos de Introducción a los algoritmos. Probablemente esto se debe a que las soluciones enviadas dentro de Mumuki.o son generadas por un grupo de estudiantes mucho más heterogéneo en su conocimiento previo, esto se nota en la cantidad de tokens únicos que tiene cada uno de los conjuntos de datos.

Como conclusión general, encontramos que la tarea de decidir cuándo un estudiante necesita ayuda personalizada porque está por abandonar un ejercicio no es una tarea fácil. Pero aún así decidimos abordarlo de dos formas distintas por un lado con un diseño minucioso de características pedagógicamente motivadas y por otro lado utilizando técnicas de procesamiento de lenguaje natural aplicado al texto plano generado por los estudiantes y usando esto como entrada para una red neuronal secuencial. Con esta segunda forma obtuvimos un mejor desempeño que con el diseño de características. A pesar de esto el aprendizaje obtenido a partir del diseño de características es interesante ya que motiva parte del trabajo futuro y motiva también un trabajo extra que es clasificar el tipo de estudiante. Por último logramos construir modelos que están listos para ser puestos a prueba en el sistema Mumuki ya que el tiempo de clasificación de los modelos es realmente corto, lo cual permite que sea integrado en un sistema web sin agregar delay a la experiencia de usuario. Además los tiempos de clasificación conseguidos serían muy difícil de alcanzar si la tarea fuese realizada por un humano, pero aún es necesario seguir trabajando sobre la misma.

7.2. Trabajo Futuro

Luego de haber formalizado el problema y lograr construir modelos que pueden ser puestos en funcionamiento dentro del sistema Mumuki, surgen algunas posibles mejoras.

1. **Registrar el tiempo de inicio de trabajo:** Actualmente el sistema Mumuki registra el momento cuándo el estudiante envía la primer solución, pero no cuando accede al ejercicio. Acá se está perdiendo información valiosa, sobre cuánto tiempo el estudiante se toma para leer la consigna hasta que envia la primer solución. Esta información ayudaría a modelar qué tipo de estudiante es el que está resolviendo el ejercicio, según la categorización que propone Papert y Turkle [73].

2. **Probar otra estructura de red neuronal:** A pesar de haber conseguido un buen desempeño con la opción neuronal, se puede seguir mejorando. Una posibilidad sería probar el desempeño de una red neuronal LSTM pero bidireccional la cual permite que la capa de salida de la red obtenga información de estados pasados y futuros. Esto tiene sentido dado que no sólo las palabras siguientes sino también las anteriores dan significado a un token dentro de un programa.

3. **Combinar experimentos:** Logramos definir características con gran potencial a la hora de clasificar las soluciones con un modelo de regresión logística. Sería interesante construir un modelo neuronal que permita combinar las características construidas junto con una

representación de la solución como dato de entrenamiento. Y analizar si se consiguen mejoras en el desempeño.

4. **Mejorar representación del texto:** La representación que usamos de las soluciones se puede mejorar aún más, usar el texto de la solución tiene una desventaja, cada estudiante usa los nombres de variables que desea en la construcción de sus soluciones. Entonces hacer comparaciones entre soluciones de estudiantes distintos para el mismo ejercicio se dificulta ya que pueden haber elegido diferentes nombres de identificadores. Para evitar esto podríamos representar las soluciones mediantes *Abstract Syntax-tree* los cuales nos permitirían poder comparar entre soluciones de un modo más eficiente. Usar este tipo de datos implica una problemática que es qué pasará con los programas enviados que no compilen por errores de sintaxis. Otra opción es simplemente reemplazar los nombre de identificadores por identificadores genéricos.

5. **Anotar el conjunto de datos de otra manera:** en este trabajo propusimos dos formas de anotar el conjunto de datos, la primera considerando sólo la última solución como abandono y la otra considerando toda la sesión como abandono. La alternativa que proponemos es analizar el desempeño que va teniendo el estudiante a lo largo de la sesión y a partir de un momento donde el desempeño desciende considerar de ese punto en adelante como abandono hasta que concluya la sesión.

6. **Identificar tipo de estudiante:** Haber construido características que modelan nivel de experiencia, insistencia, entre otras, nos abre una puerta a identificar qué tipo de estudiante está trabajando en el sistema.

7. **Trayectorias adaptativas:** Cuando un estudiante está en riesgo de abandono de un ejercicio, situación que predice nuestro modelo, sería posible analizar qué trayectoria siguieron otros estudiantes en una situación similar. Además, podemos saber si esa trayectoria fue exitosa o no para otros estudiantes. En base a esto podríamos proponer, a un estudiante en riesgo, como modificar su trayectoria indicándole qué ejercicio le conviene practicar antes de regresar al que está por abandonar. Para ello se le podría sugerir retomar algún ejercicio previamente resuelto donde se trabaje el concepto que puede estar haciéndole falta o simplemente enviarlo a un ejercicio más fácil de modo que lo motive a seguir practicando.

Bibliografía

[1] Anthony Allevato, Stephen H Edwards, and Manuel A Pérez-Quiñones. Dereferee: Exploring pointer mismanagement in student code. *ACM SIGCSE Bulletin*, 41(1):173–177, 2009.

[2] Anthony Allevato, Matthew Thornton, Stephen Edwards, and Manuel Perez-Quinones. Mining data from an automated grading and testing system by adding rich reporting capabilities. In *Educational Data Mining 2008*, 2008.

[3] Amjad Altadmri and Neil CC Brown. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 522–527. ACM, 2015.

[4] Paolo Antonucci, Christian Estler, Durica Nikolić, Marco Piccioni, and Bertrand Meyer. An incremental hint system for automated programming assignments. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 320–325. ACM, 2015.

[5] SA Arduino. Arduino. *Arduino LLC*, 2015.

[6] Georges-Louis Baron, Beatrice Drot-Delange, Monique Grandbastien, and Françoise Tort. Computer science education in french secondary schools: Historical and didactical perspectives. *ACM Transactions on Computing Education (TOCE)*, 14(2):11, 2014.

[7] Tim Bell, Peter Andreae, and Anthony Robins. A case study of the introduction of computer science in nz schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):10, 2014.

[8] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J Gomez. The effect of a web-based coding tool with automatic feedback on students' performance and perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 2–7. ACM, 2018.

[9] Luciana Benotti, Jayadev Bhaskaran, Sigtryggur Kjartansson, and David Lang. Modeling student response times: Towards efficient one-on-one tutoring dialogues. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 121–131, 2018.

[10] Luciana Benotti, Jayadev Bhaskaran, Sigtryggur Kjartansson, and David Lang. Modeling student response times: Towards efficient one-on-one tutoring dialogues. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 121–131, 2018.

[11] Luciana Benotti, Marcos J Gómez, and Cecilia Martínez. Unc++ duino: A kit for learning to program robots in python and c++ starting from blocks. In *Robotics in Education*, pages 181–192. Springer, 2017.

[12] Luciana Benotti, Cecilia Martinez, and Fernando Schapachnik. A tool for introducing computer science with automatic formative assessment. *IEEE Transactions on Learning Technologies*, pages 179–192, 2018.

[13] Luciana Benotti, María Cecilia Martínez, and Fernando Schapachnik. Engaging high school students using chatbots. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 63–68. ACM, 2014.

[14] Marc Berges and Peter Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 51–56. ACM, 2015.

[15] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599, 2014.

[16] Neil CC Brown and Amjad Altadmri. Investigating novice programming mistakes: educator beliefs vs. student data. In *Proceedings of the tenth annual conference on International computing education research*, pages 43–50. ACM, 2014.

[17] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, et al. Increasing adoption of smart learning content for computer science education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, pages 31–57. ACM, 2014.

[18] Rachel Cardell-Oliver. How can software metrics help novice programmers? In *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114*, pages 55–62. Australian Computer Society, Inc., 2011.

[19] Jason Carter, Prasun Dewan, and Mauro Pichiliani. Towards incremental separation of surmountable and insurmountable programming difficulties. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 241–246. ACM, 2015.

[20] Yuliya Cherenkova, Daniel Zingaro, and Andrew Petersen. Identifying challenging cs1 concepts in a large problem dataset. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 695–700. ACM, 2014.

[21] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. *SIGCSE Bull.*, 35(1):191–195, January 2003.

[22] V. Dagiene, L. Mannila, T. Poranen, L. Rolandsson, and P. Söderhjelm. Students' performance on programming-related tasks in an informatics contest in finland, sweden and lithuania. In *Proceedings of the 2014 Conference on Innovation; Technology in Computer Science Education*, pages 153–158. ACM, 2014.

[23] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. All syntax errors are not equal. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 75–80. ACM, 2012.

[24] Anna Katrina Dominguez, Kalina Yacef, and James R Curran. Data mining for individualised hints in e-learning. In *Proceedings of the International Conference on Educational Data Mining*, pages 91–100, 2010.

[25] Gregory Dyke. Which aspects of novice programmers' usage of an ide predict learning outcomes. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 505–510. ACM, 2011.

[26] Stephen H Edwards, Zalia Shams, Michael Cogswell, and Robert C Senkbeil. Running students' software tests against each others' code: new life for an old gimmick. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 221–226. ACM, 2012.

[27] Stephen H Edwards, Zalia Shams, and Craig Estep. Adaptively identifying non-terminating code when testing student programs. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 15–20. ACM, 2014.

[28] Stephen H Edwards, Jason Snyder, Manuel A Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 3–14. ACM, 2009.

[29] Barbara Ericson, Mark Guzdial, and Maureen Biggers. Improving secondary cs education: progress and problems. *ACM SIGCSE Bulletin*, 39(1):298–301, 2007.

[30] Nickolas JG Falkner and Katrina E Falkner. A fast measure for identifying at-risk students in computer science. In *Proceedings of the ninth annual international conference on International computing education research*, pages 55–62. ACM, 2012.

[31] James B Fenwick Jr, Cindy Norris, Frank E Barry, Josh Rountree, Cole J Spicer, and Scott D Cheek. Another look at the behaviors of novice programmers. In *ACM SIGCSE Bulletin*, volume 41, pages 296–300. ACM, 2009.

[32] Michael S Garet, Andrew C Porter, Laura Desimone, Beatrice F Birman, and Kwang Suk Yoon. What makes professional development effective? results from a national sample of teachers. *American educational research journal*, 38(4):915–945, 2001.

[33] Joanna Goode. If you build teachers, will students come? the role of teachers in broadening computer science learning for urban youth. *Journal of Educational Computing Research*, 36(1):65–88, 2007.

[34] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2009.

[35] Georgiana Haldeman, Andrew Tjang, Monica Babes-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D. Nguyen. Providing meaningful feedback for autograding of programming assignments. pages 278–283, 02 2018.

[36] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63. ACM, 2015.

[37] Matthew C Jadud. A first look at novice compilation behaviour using bluej. *Computer Science Education*, 15(1):25–40, 2005.

[38] Matthew C Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*, pages 73–84. ACM, 2006.

[39] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[40] Filiz Kalelioğlu. A new way of teaching programming skills to k-12 students: Code. org. *Computers in Human Behavior*, 52:200–210, 2015.

[41] Amela Karahasanović and Richard C Thomas. Difficulties experienced by students in maintaining object-oriented systems: an empirical study. In *Proceedings of the ninth Australasian conference on Computing education-Volume 66*, pages 81–87. Australian Computer Society, Inc., 2007.

[42] Jussi Kasurinen and Uolevi Nikula. Estimating programming knowledge with bayesian knowledge tracing. In *ACM SIGCSE Bulletin*, volume 41, pages 313–317. ACM, 2009.

[43] Ulrich Kiesmueller, Sebastian Sossalla, Torsten Brinda, and Korbinian Riedhammer. Online identification of learner problem solving strategies using pattern recognition methods. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 274–278. ACM, 2010.

[44] Ulrich Kiesmüller. Diagnosing learners' problem-solving strategies using learning environments with algorithmic problems in secondary education. *ACM Transactions on Computing Education (TOCE)*, 9(3):17, 2009.

[45] Irena Koprinska, Joshua Stretton, and Kalina Yacef. Students at risk: Detection and remediation. In *EDM*, pages 512–515, 2015.

[46] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[47] Thomas E Kurtz. Basic. In *History of programming languages I*, pages 515–537. ACM, 1978.

[48] Karen Lang, Ria Galanos, Joanna Goode, Deborah Seehorn, Fran Trees, P Phillips, and C Stephenson. Bugs in the system: Computer science teacher certification in the us. *The Computer Science Teachers Association and The Association for Computing Machinery*, 2013.

[49] Diego Levis and Roxana Cabello. *Medios informáticos en la educación a principios del siglo XXI.* Prometeo Libros Editorial, 2007.

[50] Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4520–4524. IEEE, 2015.

[51] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. Not seeing the forest for the trees: novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin*, 38(3):118–122, 2006.

[52] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[53] Cecilia Martinez, Marcos J Gomez, and Luciana Benotti. A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 159–164. ACM, 2015.

[54] Cecilia Martinez, Marcos J Gomez, Marco Moresi, and Luciana Benotti. Lessons learned on computer science teachers professional development. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 77–82. ACM, 2016.

[55] Yoshiaki Matsuzawa, Ken Okada, and Sanshiro Sakai. Programming process visualizer: a proposal of the tool for students to observe their programming process. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 46–51. ACM, 2013.

[56] Frederic P Miller, Agnes F Vandome, and John McBrewster. Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? levenshtein distance, spell checker, hamming distance. 2009.

[57] Andreas Mühling, Peter Hubwieser, and Torsten Brinda. Exploring teachers' attitudes towards object oriented modelling and programming in secondary schools. In *Proceedings of the Sixth international workshop on Computing education research*, pages 59–68. ACM, 2010.

[58] John Neter, William Wasserman, and Michael H Kutner. Applied linear regression models. 1989.

[59] Cindy Norris, Frank Barry, James B Fenwick Jr, Kathryn Reid, and Josh Rountree. Clockit: collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin*, 40(3):37–41, 2008.

[60] Seymour Papert. Teaching children thinking. *Programmed Learning and Educational Technology*, 9(5):245–255, 1972.

[61] Marco Piccioni, Christian Estler, and Bertrand Meyer. Spoc-supported introduction to programming. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 3–8. ACM, 2014.

[62] Chris Piech, Mehran Sahami, Jonathan Huang, and Leonidas Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 195–204. ACM, 2015.

[63] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160. ACM, 2012.

[64] N Ragonis, O Hazzan, and J Gal-Ezer. A survey of cs teacher preparation programs in israel tells us: Cs deserves a designated high school teacher preparation. In *Proceedings of the 41st ACM Technical Symposium on CS Education*, pages 401–405, 2010.

[65] Ana Rivoir. Innovación para la inclusión digital. el plan ceibal en uruguay. 2009.

[66] Ma Mercedes T Rodrigo and Ryan SJd Baker. Coarse-grained detection of student frustration in an introductory programming course. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 75–80. ACM, 2009.

[67] Maria Satratzemi, Vassilios Dagdilelis, and Georgios Evagelidis. A system for program visualization and problem-solving path assessment of novice programmers. In *ACM SIGCSE Bulletin*, volume 33, pages 137–140. ACM, 2001.

[68] Bronius Skupas and Valentina Dagiene. Observations from semi-automatic testing of program codes in the high school student maturity exam. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 31–36. ACM, 2010.

[69] Jaime Spacco, Davide Fossati, John Stamper, and Kelly Rivers. Towards improving programming habits to create better computer science course outcomes. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 243–248. ACM, 2013.

[70] Emily S Tabanao, Ma Mercedes T Rodrigo, and Matthew C Jadud. Predicting at-risk novice java programmers through the analysis of online protocols. In *Proceedings of the seventh international workshop on Computing education research*, pages 85–92. ACM, 2011.

[71] Richard C Thomas, Amela Karahasanovic, and Gregor E Kennedy. An investigation into keystroke latency metrics as an indicator of programming performance. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 127–134. Australian Computer Society, Inc., 2005.

[72] David Thompson and Tim Bell. Adoption of new computer science high school standards by new zealand teachers. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, pages 87–90. ACM, 2013.

[73] Sherry Turkle and Seymour Papert. Epistemological pluralism and the revaluation of the concrete. *Journal of Mathematical Behavior*, 11(1):3–33, 1992.

[74] Lisa Wang, Angela Sy, Larry Liu, and Chris Piech. Learning to represent student knowledge on programming exercises using deep learning. In *Proceedings of the 10th International Conference on Educational Data Mining; Wuhan, China*, pages 324–329, 2017.

[75] Mark Windschitl and Kurt Sahl. Tracing teachers' use of technology in a laptop computer school: The interplay of teacher beliefs, social dynamics, and institutional culture. *American educational research journal*, 39(1):165–205, 2002.

[76] Michael Yudelson, Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. Investigating automated student modeling in a java mooc. *Educational Data Mining 2014*, pages 261–264, 2014.