Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programa de Maestría en Computación

Modelo Predictivo de Exito Académico Aplicando Algoritmos de Aprendizaje de Máquina sobre Interacciones en el TEC Digital

Tesis sometida a consideración del Departamento de Computación,
para optar por el grado de Magíster Scientiae en Computación,
con énfasis en Ciencias de la Computación

Ing. José Navas Sú

Ing. Mario Chacón Rivas, Ph.D.
Profesor Asesor

Julio 2017

RESUMEN

La teoría sobre aprendizaje de máquina y minería de datos tiene muchas aplicaciones en el ámbito educativo. En particular, en el diseño de modelos predictivos a partir de los registros masivos de datos relativos al proceso de enseñanza/aprendizaje. En el presente trabajo se diseñan y evalúan varios modelos predictivos de éxito académico, con la finalidad de crear herramientas de apoyo para la intervención temprana en casos de reprobación académica. Estos modelos *Ad hoc* se diseñaron para la plataforma educativa del TEC Digital. Los algoritmos de aprendizaje de máquina empleados son Regresión Logística, Máquinas de Soporte Vectorial y Redes Neuronales. El conjunto de datos de entrada a los modelos comparados consiste en las interacciones de los estudiantes en el TEC Digital. Los mejores resultados obtenidos corresponden al algoritmo de Redes Neuronales, pero no se encontró los parámetros ni los niveles de complejidad que permitan ajustar modelos semanales que predigan con una exactitud superior al 80 % los estudiantes que tienen más probabilidad de reprobar un curso.

ABSTRACT

Machine learning and data mining have many applications in the educational field. In particular, in the design of predictive models from massive data records related to learning process. In the present work, several predictive models of academic success are designed and evaluated, in order to create support tools for early intervention on cases of academic failure. These Ad hoc models were designed for the TEC Digital's educational platform. The machine learning algorithms used are Logistic Regression, Support Vector Machines and Neural Networks. The input dataset to the compared models consists of student's interactions within TEC Digital's educational platform. The best results obtained correspond to the algorithm of Neural Networks, but it weren't found the parameters nor the levels of complexity to adjust models that predict with an accuracy greater than 80 %, the students who are most likely to fail a course.

ÍNDICE GENERAL

A mi hija, mi esposa y mi madre

Programa de Maestría
en Computación
Teléfono: 2550-2402

ACTA DE APROBACION DE TESIS

Con fundamento en lo que establecen los **Artículos 22-24-25** del "Manual de Normas y Procedimientos para optar por el título de "MAGÍSTER SCIENTIAE EN COMPUTACION", el Tribunal Examinador de Tesis (TET), nombrado con el propósito de evaluar la tesis de grado.

"Modelo Predictivo de Éxito Académico Aplicando Algoritmos de Aprendizaje de Máquina sobre Interacciones en el TEC Digital"

Habiendo analizado el resultado general del trabajo presentado por el estudiante:

Primer Apellido	Segundo Apellido	Nombre	No. de carné
NAVAS	SU	JOSÉ DOLORES	8804886

Emite el siguiente dictamen:

☒ APROBADO

El TET, considerando que el trabajo realizado por el estudiante es SOBRESALIENTE, le otorga la siguiente MENCION HONORIFICA:

○ CUM LAUDE ☒ MAGNA CUM LAUDE ○ SUMMA CUM LAUDE

○ REPROBADO

○ SE RECOMIENDA ○ NO SE RECOMIENDA

Brindarle una nueva oportunidad para la DEFENSA PUBLICA de su Tesis

NUEVA FECHA: _____

Dando fe de lo aquí expuesto firmamos (*IDEM*: HOJAS DE APROBACION DE TESIS)

Dr. Mario Chacón Rivas
Profesor Asesor

Dr. Cesar Garita Rodríguez
Profesor Lector

Máster/DEA Luis Carlos Naranjo Zeledón
Profesional Externo

Dr. Roberto Cortés Morales
Coordinador del Programa
de Maestría en Computación

TEC | Tecnológico de Costa Rica
Maestría en Computación

26 de julio de 2017

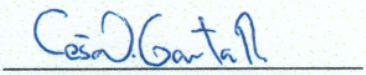
FT-07-MC

APROBACIÓN DE LA TESIS

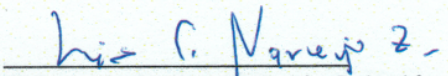
"Modelo Predictivo de Éxito Académico Aplicando Algoritmos de Aprendizaje de Máquina sobre Interacciones en el TEC Digital"

TRIBUNAL EXAMINADOR

Dr. Mario Chacón Rivas
Profesor Asesor



Dr. Cesar Garita Rodríguez
Profesor Lector



Máster/DEA Luis Carlos Naranjo Zeledón
Profesor Externo

Dr. Roberto Cortés Morales
Coordinador del Programa
de Maestría en Computación

TEC | Tecnológico de Costa Rica
Maestría en Computación

Julio, 2017

Agradezco al Dr. Mario Chacón Rivas, antes Director del TEC Digital y actual Investigador del CeNAT, la idea inicial y la asesoría brindada durante los dos últimos semestres para realizar este trabajo, y el acceso a los datos registrados en la base de datos del TEC Digital.

Del mismo modo, al Máster Agustín Francesa Alfaro, actual Director del TEC Digital, el conocimiento técnico sobre la estructura de la base de datos.

Al Dr. Esteban Meneses Rojas, actual Director del CNCA/CeNAT, el facilitar el acceso al Cluster Kabré.

También, al Ing. Willy Gerardo Villalobos Marrero, Asistente de Soporte del CNCA/CeNAT, la invaluable y oportuna asistencia técnica brindada.

Al Dr. César Garita Rodríguez, la guía metodológica sobre la documentación del trabajo realizado.

Al Dr. Luis Carlos Naranjo, sus sugerencias sobre el documento de tesis final.

Finalmente, al Máster Luis Alexander Calvo, las valiosas y atinadas sugerencias técnicas sobre los algoritmos de aprendizaje de máquina.

"It seems likely that once the machine thinking method had started, it would not take long to outstrip our feeble powers."

Alan Turing [1]

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

LISTA DE ACRÓNIMOS

.LRN	dotLRN
AA	Academic Analytics
AIHS	Adaptive and Intelligent Hypermedia Systems
AUC	Area Under the ROC Curve
CBE	Sistemas Educativos Basados en Computación
CeNAT	Centro Nacional de Alta Tecnología
CNCA	Colaboratorio Nacional de Computación Avanzada
CGPA	Cumulative Grade Point Average
CV	Cross Validation
EDM	Educational Data Mining
GLM	Generalized Linear Models
GNU	General Public License
GRM	Guaranteed Risk Minimization
HPC	High Performance Computing
ITS	Intelligent Tutoring Systems
k-fold CV	k-Fold Cross-Validation
LA	Educational Data Mining, Learning Analytics
LMS	Learning Management System
LOF	Local Outlier Factor

LOOCV Leave-One-Out Cross-Validation

LOSS Función de Pérdida de Utilidad

LR-ESOM Local Outlier Factor-Recurrent-Emergent Self Organizing Map

MDL Minimum Description Length Principle

MIT Massachusetts Institute of Technology

MPI Message Passing Interface

MSE *media del cuadrado del error*

NMF Nonnegative Matrix Factorization

OpenACS Open Architecture Community System

R-ESOM Recurrent Emergent Self-Organizing Map

R^n Repetición de Materias

SCpe Contratos de Servicio para Eventos Auditados

SGD Stochastic Gradient Descent

SNA Social Network Analysis

SOM Self-Organizing Maps

SVM Máquinas de Soporte Vectorial

TAE Tracking and Auditing Engines

TAM Tracking and Auditing Module

TEC Instituto Tecnológico de Costa Rica

TIC Tecnologías de Información y Comunicación

1 INTRODUCCIÓN

Los procesos educativos como proyecto social juegan un rol de vital importancia en toda sociedad, y con mayor énfasis en las pertenecientes al conjunto de países conocidos como "en vías de desarrollo". Los recursos materiales con que cuenta la sociedad en general son muy limitados, y más limitados son los que se asignan al mantenimiento y fortalecimiento del sistema de educación pública.

Es de suma importancia enfocar todos los esfuerzos posibles en la consecución de los objetivos del sistema educativo público, tanto en la asignación eficiente de los recursos asignados por la sociedad, como en la eficacia de los procesos de formación de los profesionales requeridos.

El presente trabajo pretende aportar en la dirección de generar información útil para el apoyo de la docencia, proporcionando una herramienta automatizada que permita identificar aquellos estudiantes que podrían requerir adecuación o intervención preventiva mientras se desarrolla el proceso de formación, en contraposición al reconocimiento de estas necesidades *a posteriori*. Con esto se podría reducir el impacto del fenómeno reconocido como Repetición de Materias (R^n) tanto en el tiempo de graduación promedio como en la cantidad promedio de recursos asignados a subvencionar la carrera de cada estudiante.

Se utilizaron técnicas de aprendizaje de máquina para entrenar un modelo de predicción del éxito académico de los estudiantes en los cursos, tomando como conjunto de datos las interacciones registradas por el módulo de rastreo y auditoría de la plataforma Learning Management System (LMS) del TEC Digital. El objetivo de este modelo predictivo es proveer de herramientas para la implementación de

estrategias de abordaje preventivo para los departamentos tanto de carrera como de apoyo psico-educativo de la institución.

1.1 CONTEXTO DE LA INVESTIGACIÓN

El proyecto TEC Digital del Instituto Tecnológico de Costa Rica (TEC) surgió en el año 2007 como una iniciativa institucional para incorporar las Tecnologías de Información y Comunicación (TIC) en los programas académicos [2] [3].

El TEC define entre sus objetivos estratégicos, "*Fortalecer los procesos académicos, mediante el mejoramiento continuo, el uso de tecnologías innovadoras, la internacionalización y el emprendedurismo*" (Objetivo Estratégico 4). Además, también en el Objetivo Estratégico Octavo: "*Fortalecer la incorporación de las tecnologías de información y comunicación en el mejoramiento del quehacer académico y las actividades de apoyo a la academia*" [4].

El 'TEC Digital es la plataforma educativa en línea del TEC. Permite la interacción entre estudiantes y profesores a través de medios como foros, calendario y noticias, entre otros; además, brinda acceso a información estudiantil y de cursos, incluyendo documentos y evaluaciones de estos" [5]. La meta que persigue es fortalecer los procesos educativos mediante la aplicación de herramientas colaborativas para la gestión del conocimiento y la investigación, el seguimiento y apoyo de los procesos de enseñanza y aprendizaje en el TEC.

El TEC Digital es una implementación de la plataforma LMS dotLRN (.LRN) / Open Architecture Community System (OpenACS) [6]. .LRN [7] es un software libre que provee soporte para *e-learning* y comunidades digitales, fue desarrollado originalmente en el Massachusetts Institute of Technology (MIT), y es utilizado a nivel mundial por más de medio millón de usuarios en instituciones de educación superior, entidades gubernamentales y organizaciones sin fines de lucro. OpenACS

[8] es un conjunto de herramientas para construir aplicaciones web escalables y orientadas a comunidades, sobre el que se fundamenta .LRN, es código abierto y está disponible bajo el tipo de licencia General Public License (GNU).

OpenACS cuenta con un módulo de rastreo y auditoría de todas las interacciones realizadas por los usuarios de la plataforma denominado Tracking and Auditing Module (TAM). En [9] y [10], Couchet et al. presentan el motor Tracking and Auditing Engines (TAE) y el módulo de rastreo y auditoría para la herramienta OpenACS. El TAE provee los medios para registrar todas las acciones realizadas tanto por los usuarios directos del sistema como del sistema mismo. Según los autores, una acción consiste en una pieza específica de funcionalidad del sistema, y la ejecución de una acción implica alguna clase de procesamiento aplicado sobre un objeto o conjunto de objetos en un conjunto de datos. Un objeto consiste en un conjunto de atributos, donde un atributo es un par ordenado único nombre-valor, no son objetos de OpenACS. El TAE (ver Figura 1) está constituido por dos partes separadas con responsabilidades diferentes, por un lado, el servicio de auditoría se encarga de la definición de los objetos que serán auditados, la definición de los datos necesarios para auditar estos objetos y el procesamiento semántico de los datos registrados, con la finalidad de extraer los objetos y el significado de la interacción entre dichos objetos, y por el otro, el servicio de rastreo que se encarga de registrar los datos definidos por el servicio de auditoría. Los otros dos componentes de la arquitectura son: un servidor JASPER encargado de la creación de reportes y la visualización de los mismos por medio de la herramienta JasperReports, y los Contratos de Servicio para Eventos Auditados (SCpe) que permiten definir los comandos que son capaces de generar meta data -cada paquete de OpenACS que utilice los servicios de rastreo y auditoría debe proveer una implementación de SCpe, así como sus propias funciones de procesamiento semántico para extraer el significado a partir de los datos registrados por el servicio de rastreo.

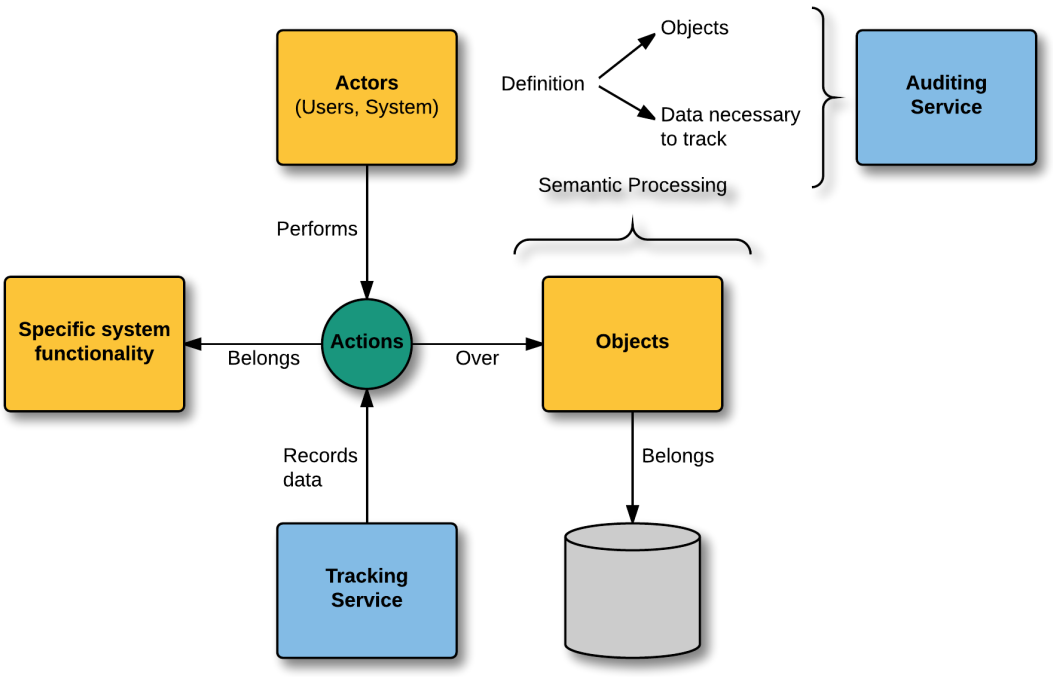


Figura 1: Procesos de rastreo y auditoría
Fuente: [10, p. 83]

Existen otros LMS similares a .LRN, como Moodle y Sakai. En [11] se realizó una evaluación de dichas plataformas con respecto a usabilidad y se encontró que .LRN supera a Moodle y Sakai, pero ninguna de las plataformas alcanza el 80 % de conformidad con los parámetros definidos para la evaluación. En las fases iniciales del proyecto TEC Digital [2] se realizó un análisis comparativo de las plataformas LMS .LRN, Moodle y Sakai, y se eligió implementar .LRN porque incluye administración de comunidades virtuales, permite la creación y personalización de portales, y por el diseño de su interface, además de que utiliza un motor de bases de datos robusto y una arquitectura escalable.

1.2 DESCRIPCIÓN DEL PROBLEMA

Realizar estudios universitarios es un proceso difícil y exigente, de gran impacto tanto a nivel personal como social. Este proceso implica una inversión importante de recursos materiales por parte del estudiante y su núcleo familiar, pero más aún por parte de la sociedad, sobre todo en el caso de las universidades que subsidian significativamente dichos estudios con recursos estatales. De ahí que el éxito académico alcanzado afecta no sólo al estudiante como tal en la gestación y desarrollo de su carrera profesional, sino también a la sociedad como proyecto social.

El problema acometido en esta propuesta es la ausencia de modelos prácticos y herramientas automatizadas que ayuden a identificar de manera oportuna los sujetos que requieren adecuación de la carga académica y/u orientación especializada.

Las variables que influyen sobre el éxito académico probablemente sean múltiples y muy complejas. Esto hace que sea una labor muy difícil el diseño de estrategias de abordaje preventivo para los departamentos de apoyo psico-educativo, para los docentes e incluso para el mismo estudiante.

La hipótesis consiste en que es posible generar un modelo predictivo a partir únicamente de las interacciones en la plataforma del TEC Digital.

1.3 JUSTIFICACIÓN

Proveer modelos y herramientas que asistan en el diseño de estrategias de abordaje preventivo de los procesos académicos para los departamentos de apoyo psico-educativo podría tener un impacto muy importante tanto en el mejoramiento de la calidad de los procesos formativos como de la eficiencia y eficacia en la utilización de los recursos destinados por la sociedad a subsidiar la formación de su fuerza de trabajo altamente calificada.

1.4 OBJETIVOS

1.4.1 *Objetivo General*

Diseñar un modelo predictivo de éxito académico universitario, aplicando algoritmos de aprendizaje de máquina, entrenados usando datos históricos de éxito académico e interacciones en el TEC Digital.

1.4.2 *Objetivos Específicos*

1. Crear un modelo de datos anónimos básico inicial con variables extraídas del historial de éxito académico y las interacciones categorizadas registradas en el TEC Digital, tomando en cuenta todos los cursos y estudiantes participantes.
2. Seleccionar el algoritmo de aprendizaje de máquina que mejor se adapte al conjunto de datos para crear el modelo predictivo.
3. Diseñar un modelo de predicción de éxito académico utilizando el algoritmo de aprendizaje de máquina seleccionado.
4. Implementar y entrenar el prototipo de predicción de éxito académico universitario.
5. Validar el modelo predictivo generado.

1.5 ALCANCES

Para este trabajo se cuenta con datos que serán obtenidos a partir de las interacciones en el TEC Digital, de los estudiantes participantes en el año 2016 en los cursos

semestrales correspondientes a planes de estudio de bachillerato universitario, y del historial académico de estos mismos estudiantes. El procedimiento a seguir para la extracción de los datos se detalla en la sección 3.1.

Los algoritmos de aprendizaje de máquina a considerar son: Regresión Lineal, Máquinas de Soporte Vectorial y Redes Neuronales.

El modelo de predicción permitiría obtener una proyección del éxito académico para cualquier estudiante en cualquiera de las semanas que constituyen el semestre, con base en las interacciones, agrupadas en categorías semánticas, que ha realizado en el TEC Digital desde la semana inicial hasta la semana en que se realiza la proyección.

Este modelo podrá luego incorporarse como un módulo en la plataforma del TEC Digital. Esto requeriría programación adicional que se sale del alcance de esta propuesta.

2 MARCO TEÓRICO

2.1 ALGORITMOS DE APRENDIZAJE DE MÁQUINA

Russell y Norvig [12], en su libro *Artificial Intelligence: A Modern Approach*[1], describen tres tipos posibles de aprendizaje de máquina: aprendizaje no supervisado, aprendizaje por refuerzos y aprendizaje supervisado.

En el aprendizaje no supervisado los algoritmos extraen patrones del conjunto de datos de entrada sin retroalimentación. La tarea de aprendizaje no supervisado más común es denominada *clustering*, y consiste en detectar grupos potencialmente útiles a partir de las muestras de entrada.

En el caso del aprendizaje por refuerzos, los algoritmos aprenden a partir de una serie de refuerzos (recompensas o castigos). Como ejemplos se menciona: la falta de propina al final de un servicio de taxi indica a un algoritmo agente de taxi un indicador de que realizó algo de manera errónea; los dos puntos por una victoria al final de un juego de ajedrez le dice al algoritmo agente que hizo algo correctamente. Es responsabilidad del agente decidir cuál de las acciones que precedieron al refuerzo fue la responsable de este.

En el aprendizaje supervisado el agente observa muestras pareadas de ejemplo entrada-salida y aprende una función que mapea la entrada con la salida.

1 Los elementos teóricos presentados en este apartado son tomados principalmente de esta fuente.

2.1.1 *Aprendizaje Supervisado*

La tarea de aprendizaje supervisado es:

Dado un *conjunto de entrenamiento* con N pares de muestras entrada-salida (x_1, y_1), $(x_2, y_2), ...(x_N, y_N)$, donde y_j fue generado por una función desconocida $y = f(x)$, descubrir una función h que aproxime la función verdadera f.

Aquí x y y pueden ser cualquier valor, no necesariamente numéricos. La función h es la *hipótesis*. El aprendizaje consiste en una búsqueda en el espacio de hipótesis posibles por una que *generalice* bien, es decir, que se desempeñe bien, incluso con muestras nuevas ausentes en el conjunto de entrenamiento. Para medir la exactitud de una hipótesis se utiliza un conjunto de prueba con muestras distintas a las del conjunto de entrenamiento.

Cuando la salida y es un valor dentro de un conjunto finito de valores, el problema de aprendizaje es denominado *clasificación*, si sólo hay dos valores posibles es *clasificación binaria*, y si hay más de dos valores es *clasificación multiclase*. Cuando y es una variable numérica continua (i.e., la temperatura de mañana), el problema de aprendizaje es denominado *regresión*.

Para un mismo conjunto de entrenamiento, es posible ajustar diferentes funciones h tomadas del *espacio de hipótesis H*. Por ejemplo, funciones de diverso grado polinomial, que se ajustarán a los datos en mayor o menor grado. Así, entre varias hipótesis que se ajustan bien a los datos de entrenamiento, se debe escoger la más simple de estas, de acuerdo con el principio de *la navaja de Ockham* (la de menor grado polinomial).

Además, existe un compromiso entre las hipótesis complejas que pueden ajustarse mejor a los datos de entrenamiento y las hipótesis más simples que pueden generalizar mejor y ajustarse más a los datos nuevos que se pretende clasificar. Desde el punto de vista de la complejidad de las hipótesis, también existe un compromiso

entre la *expresividad* de un espacio de hipótesis y la complejidad de encontrar una buena hipótesis en dicho espacio. Es posible ajustar desde funciones lineales hasta máquinas de Turing, pasando por funciones polinomiales de mayor grado.

2.1.2 *Generalización y Problemas de Ajuste*

Al ajustar un algoritmo a partir de un conjunto de datos de entrenamiento, con la pretensión de que generalice bien sobre muestras nuevas, es común que ocurran problemas con dicho ajuste, y estos pueden ser de dos tipos: *sobreajuste* (en inglés *overfitting*) y *subajuste* (en inglés *underfitting*). El sobreajuste significa que h se ajusta demasiado a los datos de entrenamiento y falla en generalizar para nuevas muestras. El subajuste es la otra cara de la moneda, significa que h no se ajusta suficientemente a los datos de entrenamiento, y también falla en generalizar para nuevas muestras. Cuando crecen tanto el espacio dimensional de la hipótesis como la cantidad de atributos de entrada, es más probable que ocurra sobreajuste; en cambio, al incrementar la cantidad de muestras para el entrenamiento es menos probable que se presente dicho problema.

Como se indica en James et al. [13], la calidad de ajuste de un algoritmo se mide mediante la *media del cuadrado del error* (MSE) (*mean squared error*), o simplemente *error*, que corresponde a

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2, \tag{1}$$

donde $h(x_i)$ es la predicción hecha utilizando h para la i-ésima observación. Cuando el error es calculado sobre las muestras utilizadas en el entrenamiento, se denomina *error de entrenamiento*, mientras que al calcularlo sobre las muestras reservadas para la prueba, se denomina *error de prueba*. El MSE será pequeño si las respuestas pre-

dichas están muy cerca de las respuestas verdaderas, y será grande si para algunas observaciones existe una diferencia significativa entre las respuestas predichas y las respuestas verdaderas.

Esta medida de ajuste puede ser utilizada para seleccionar, entre varios modelos $h_1, h_2, \ldots, h_n$, aquel que produzca el menor error de prueba al predecir con h_i.

Siguiendo a [13], el error se puede descomponer como la suma de: la *varianza* de $h(x_0)$, el cuadrado del *bias* de $h(x_0)$ y la varianza de los términos de error σ_ϵ^2. El error de prueba esperado se puede expresar de la siguiente manera

$$\begin{aligned} E[(y_0 - h(x_0))^2] &= (E[h(x_0)] - y_0)^2 + E[(h(x_0) - E[h(x_0)])^2] + \sigma_\epsilon^2 \\ Err(x) &= Bias^2 + Varianza + Error\ Irreducible. \end{aligned} \tag{2}$$

El tercer término corresponde al ruido en la relación verdadera y no puede ser eliminado por ningún modelo. Si se tiene el modelo real y una cantidad infinita de datos para calibrarlo, se podría eliminar tanto el error de bias como el error de varianza. Pero como esto no es posible de ordinario, lo mejor que se puede hacer es lograr un compromiso entre minimizar el bias y minimizar la varianza.

El error debido al bias corresponde a la diferencia entre la predicción promedio del modelo y el valor correcto que se quiere predecir, mide qué tan lejos están las predicciones del valor correcto. Es el error introducido al aproximar la función real, que puede ser muy compleja, utilizando un modelo mucho más simple. Si el error de bias es muy alto, aumentar el tamaño del conjunto de datos para entrenar el modelo no contribuiría a superar un problema de subajuste. Como norma general, los métodos más complejos y flexibles tienden a presentar bajo bias.

El error de varianza es la variabilidad del modelo predictivo, representa cuánto varían las predicciones cuando se ajusta el modelo con conjuntos de datos de entrenamiento distintos. Si el modelo sufre de alta varianza, un cambio mínimo en los datos de entrenamiento puede resultar en cambios significativos en la hipótesis h.

Como norma general, los métodos más complejos y flexibles tienden a tener alta varianza, por ejemplo, cuando se agregan atributos para aumentar el grado polinomial de los datos de entrenamiento y consecuentemente aumentar la capacidad del modelo para ajustarse a los datos.

Como puede observarse en la Figura 2, al aumentar la complejidad del modelo o hipótesis ajustado, disminuye el error de entrenamiento, mientras que el error de prueba inicialmente disminuye junto con el error de entrenamiento pero en cierto nivel de complejidad se invierte la tendencia y continúa aumentando. Es decir, el error de entrenamiento presenta una tendencia monótonamente decreciente al aumentar la complejidad, mientras el error de prueba presenta forma de U. Esta propiedad está presente en todos los métodos de aprendizaje de máquina y se sostiene para cualquier conjunto de datos y cualquier método utilizado.

El fenómeno de sobreajuste se identifica cuando el error de entrenamiento es pequeño y el error de prueba es grande, y tiene que ver con que se aprende patrones presentes en los datos de entrenamiento que no necesariamente están en los datos de prueba. El fenómeno contrario, de subajuste, se identifica cuando tanto el error de entrenamiento como el de prueba son muy cercanos entre sí pero grandes, como sucede con la hipótesis lineal. El modelo que mejor se ajusta será aquel cuyo error de prueba sea el mínimo. Existen varios enfoques para estimar este mínimo, en particular el método Cross Validation (CV), que lo hace utilizando los datos de entrenamiento.

Para minimizar el error de prueba (correspondiente a la Ecuación 2) es necesario seleccionar el modelo que a la vez consiga minimizar tanto la varianza como el bias. Esta relación entre bias, varianza y error de prueba (ver Figura 3), es conocida como *compromiso entre bias y varianza*.

Existen herramientas para el ajuste de modelos conocidas como técnicas de remuestreo. Estas consisten en extraer muestras de manera repetida desde un conjun-

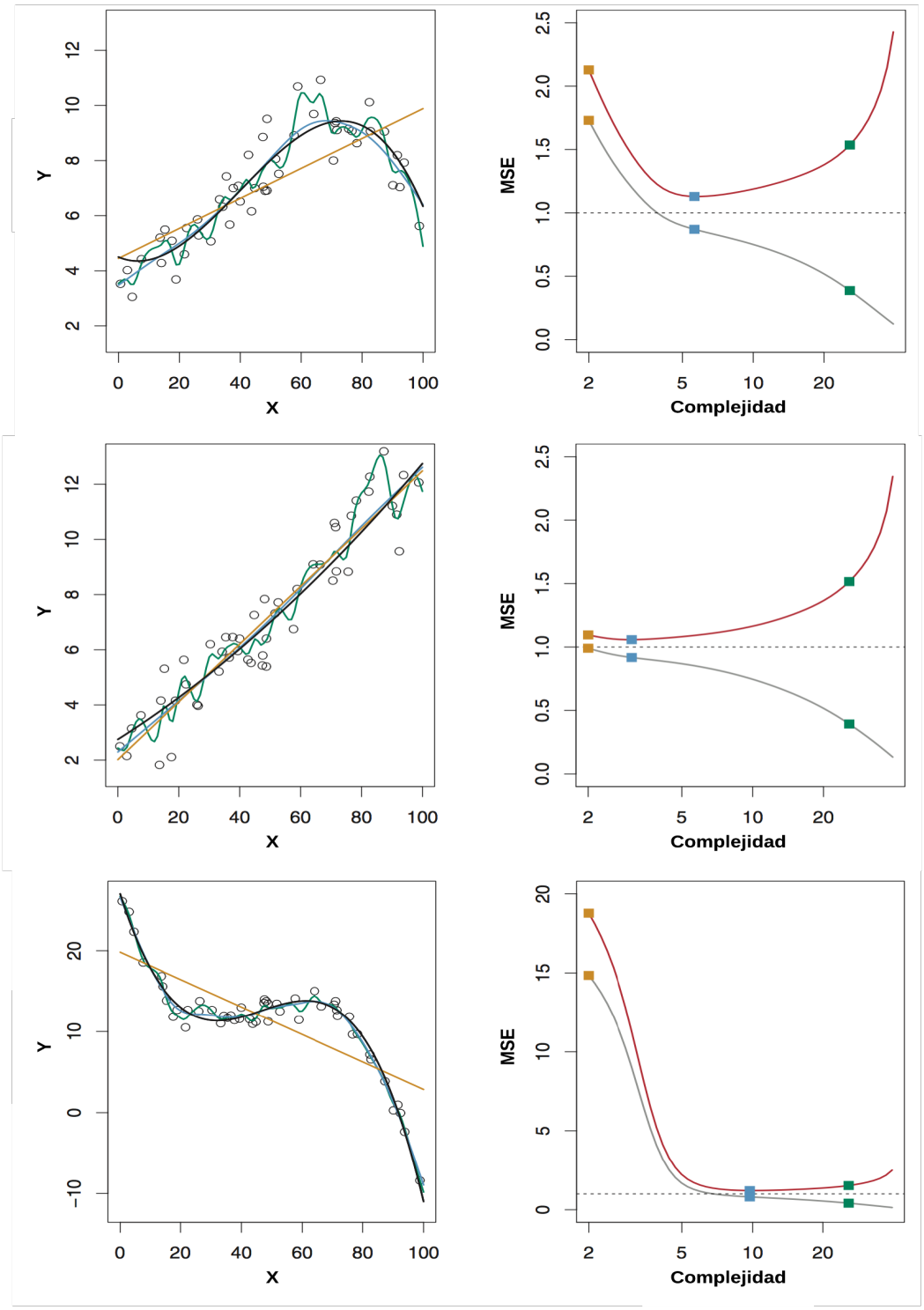


Figura 2: Izquierda: Datos simulados para tres funciones f diferentes, mostradas en negro. Tres modelos mostrados para cada f: regresión lineal (curva naranja), y dos ajustes tipo *smoothing spline* (curvas azul y verde). Derecha: error de entrenamiento (curva gris) y error de prueba (curva roja). Los cuadros de colores representan el error de entrenamiento y de prueba para los tres ajustes mostrados a la izquierda
Fuente: [13, pp. 31-34]

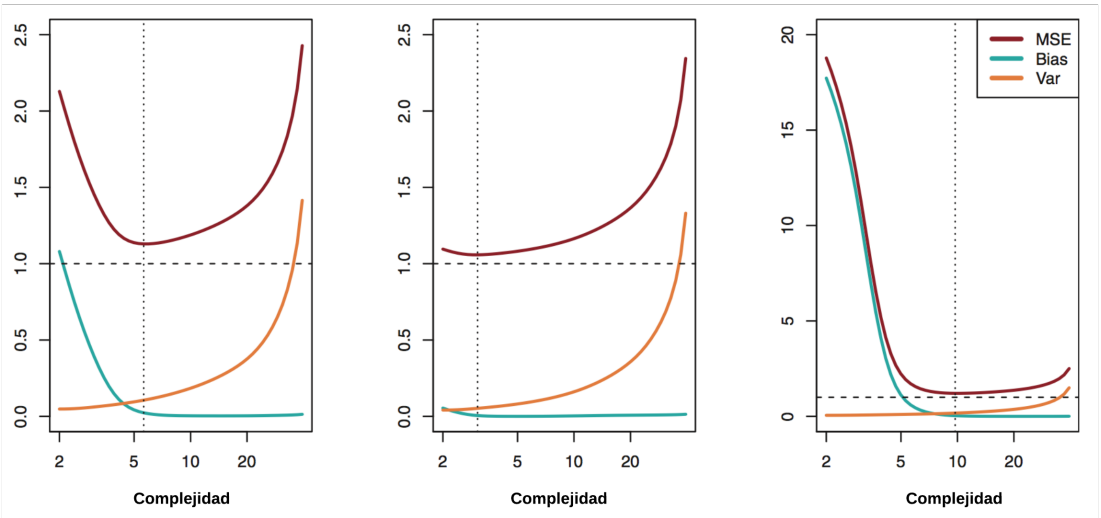


Figura 3: Bias cuadrado (curva azul), varianza (curva naranja), error irreducible (línea punteada) y error de prueba (curva roja) para los tres conjuntos de datos de la Figura 2. La línea vertical indica el modelo cuyo nivel de complejidad minimiza el menor error de prueba. Fuente: [13, p. 36]

to de datos de entrenamiento y reajustar un modelo sobre cada muestra extraída, con el fin de obtener mayor información sobre dicho modelo. Es decir, se ajusta el mismo modelo varias veces utilizando cada vez un subconjunto distinto de los datos de entrenamiento. Los métodos de remuestreo más comúnmente utilizados son CV y *Bootstrap*. CV permite estimar el error de prueba para un modelo o hipótesis h con el fin de evaluar su desempeño (*evaluación de modelo*) o de escoger el nivel de complejidad más apropiado (*selección de modelo*). El bootstrap se usa principalmente para proporcionar una medida de la exactitud de la estimación de un parámetro o de un algoritmo de aprendizaje dado.

2.1.2.1 *Selección de Modelo con el Método Cross Validation*

Este es el método más utilizado para estimar el error de prueba [13, 14]. La idea fundamental consiste en dividir el conjunto de datos de entrenamiento en dos partes, seleccionando las muestras de cada parte de manera aleatoria, una para ajustar el modelo y otra para realizar validación cruzada, es decir, para medir el error de

entrenamiento con base en muestras que no se utilizaron para el ajuste del modelo. Este es denominado el enfoque del subconjunto de validación. La clave aquí es reservar este subconjunto de datos exclusivamente para realizar la validación del modelo ajustado. La tasa de error resultante, obtenida aplicando MSE, proporciona un estimado de la tasa de error de prueba. Este enfoque es simple y fácil de implementar, pero tiene las siguientes desventajas: por un lado, el estimado de validación de la tasa de error de prueba puede ser altamente variable, y depende de cuáles observaciones se incluyeron en cada subconjunto, y por otro lado, únicamente un subconjunto es utilizado en el ajuste del modelo, lo cual es una desventaja pues el ajuste tiende a comportarse peor cuando se basa en pocas muestras, resultando en una sobreestimación de la tasa de error de prueba.

Este enfoque se puede mejorar introduciendo la noción de validación cruzada o CV, con lo cual se obtiene las siguientes variantes:

- Leave-One-Out Cross-Validation (LOOCV)
- k-Fold Cross-Validation (*k-fold CV*)

El método LOOCV pretende superar las limitaciones que presenta el enfoque del subconjunto de validación. También divide los datos de entrenamiento en dos partes, donde una consta de una única observación que será utilizada en la fase de validación, y la otra contiene el resto de los datos de entrenamiento, y es utilizada para ajustar el modelo. Esta división se repite para cada una de las muestras en el conjunto de datos de entrenamiento.

Sea $T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ el conjunto de datos de entrenamiento. Se ajusta el modelo con el subconjunto $T \setminus \{(x_i, y_i)\}$, para todo $1 \leq i \leq n$, y se calcula

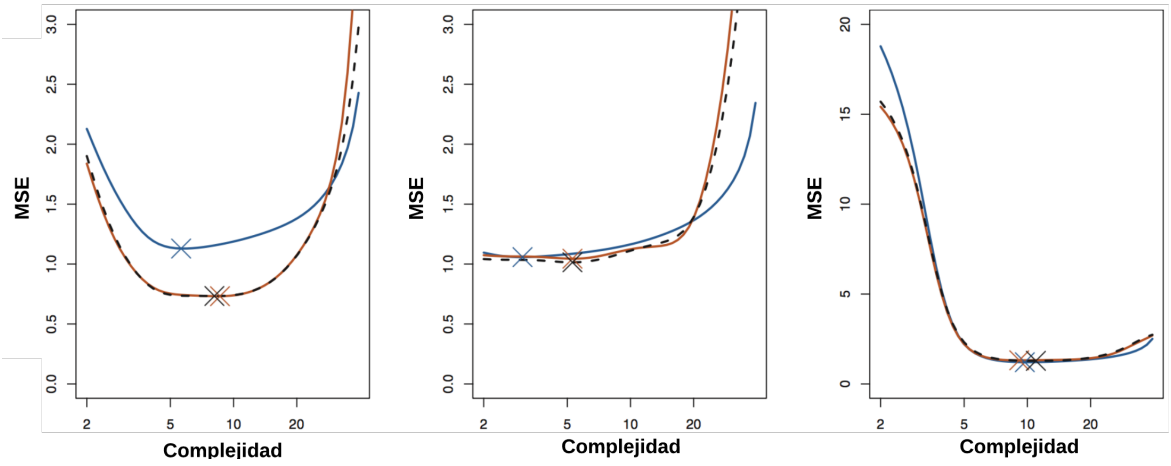


Figura 4: MSE de prueba real y estimado para los conjuntos de datos de las Figuras 2 y 3. El MSE de prueba real corresponde a la línea azul, el estimado con LOOCV a la línea discontinua negra, y el estimado con 10-fold CV a la línea anaranjada. Las marcas en forma de X indican el mínimo en cada curva MSE.
Fuente: [13, p. 182]

$MSE_i = (y_i - h(x_i))^2$. Finalmente se obtiene el estimado del MSE de prueba como el promedio de estos n estimados del error de prueba:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} MSE_i. \tag{3}$$

Este método es menos susceptible al problema de alto bias, no tiene la tendencia a sobreestimar la tasa de error de prueba, y no genera resultados distintos al ser aplicado de manera repetida, al eliminarse la aleatoriedad en la división de los datos de entrenamiento y validación. Pero tiene el problema de que requiere de muchos recursos computacionales y tiempo de procesamiento.

Si no se dispone de una cantidad de datos de entrenamiento, como para dividirlos en dos subconjuntos suficientemente grandes, uno para el ajuste del modelo y otro para validar el desempeño del modelo ajustado, y además se requiere minimizar el uso de recursos computacionales y tiempo, se puede emplear el método de *k-fold CV*. En este enfoque se divide el conjunto de datos de entrenamiento en k grupos aproximadamente del mismo tamaño, $T = T_1 \cup T_2 \cup \cdots \cup T_k$, donde cada T_i

consta de aproximadamente $\lfloor n/k \rfloor$ muestras seleccionadas de manera aleatoria sin reemplazo. Seguidamente se aplica el mismo algoritmo que LOOCV, con la variante de que en la i-ésima de k iteraciones, se ajusta el modelo con $T \setminus T_i$ y se calcula MSE_i sobre T_i. Seguidamente se obtiene el estimado del MSE de prueba como el promedio de estos k estimados del error de prueba:

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MSE_i. \tag{4}$$

Como puede verse en la Figura 4, el MSE de prueba estimado mediante *k-fold CV* es bastante cercano al MSE de prueba real, y a menudo genera estimados más exactos que LOOCV, por lo cual este método es ampliamente aplicado para selección del modelo que minimiza el error de prueba. Según [13] esto tiene que ver con un compromiso entre bias y varianza. LOOCV provee un estimado del error de prueba sin bias, mientras que *k-fold CV* provoca un nivel intermedio de bias. Si se requiere reducir el bias es mejor escoger LOOCV. Pero como el bias no es la única fuente de error, hay que considerar también la varianza. LOOCV tiene mayor varianza que *k-fold CV*. Por lo tanto, se requiere de alcanzar un equilibrio mediante la calibración de la cantidad de grupos en *k-fold CV*. Para $k = 5$ o $k = 10$ se ha mostrado de manera empírica que los estimados de la tasa de error obtenidos no sufren en exceso de alto bias ni de alta varianza.

En [15] se menciona tres métodos para la selección de modelos: Guaranteed Risk Minimization (GRM), Minimum Description Length Principle (MDL) y CV. El método a utilizar en este trabajo será *k-fold CV*.

2.1.2.2 *Función de Pérdida de Utilidad*

En [12] se introduce la función de pérdida de utilidad como una mejora sobre la tasa de error, para penalizar los errores de predicción cuando no todos los errores son iguales, diferenciando qué tan malo es un error. Si, por ejemplo, se

acomete la tarea de clasificar mensajes de correo electrónico como deseados y no deseados, es peor clasificar correo deseado como no deseado, que lo contrario. Por lo tanto, un clasificador con una tasa de error de 1 %, en donde la mayoría de los errores de clasificación consisten en clasificar correo no deseado como deseado, es mejor que un clasificador con una tasa de error de 0.5 %, si la mayoría de esos errores consisten en clasificar correo deseado como no deseado. Se define esta función de pérdida $L(y, \hat{y})$, como la cantidad de utilidad perdida al predecir $h(x) = \hat{y}$ cuando la respuesta correcta es $f(x) = y$. Así, para modelar que es 10 veces peor clasificar mensajes deseados como no deseados que lo contrario, se establece $L(spam, nospam) = 1, L(nospam, spam) = 10$. $L(y, y)$ siempre es cero, porque no hay pérdida cuando se clasifica correctamente.

Los errores pequeños son mejores que los grandes; dos funciones que implementan esta idea son el valor absoluto de la diferencia, o Función de Pérdida de Utilidad (LOSS) L_1, y el cuadrado de la diferencia, o LOSS L_2. Si es suficiente minimizar la tasa de error, se puede utilizar la función LOSS $L_{0/1}$, que tiene una pérdida de 1 para una respuesta incorrecta y es apropiada para salidas con valores discretos.

$$\begin{aligned} L_1(y, \hat{y}) &= |y - \hat{y}| \\ L_2(y, \hat{y}) &= (y - \hat{y})^2 \\ L_{0/1}(y, \hat{y}) &= 0 \text{ si } y = \hat{y}, \text{ de otro modo } 1 \end{aligned} \tag{5}$$

Existen otras funciones de pérdida, por ejemplo, en [14] se mencionan “Huberized loss” y “hinge loss”.

De acuerdo con [12], la pérdida de generalización estimada $EstLoss_{L,M}(h)$, para una hipótesis h dada una función de pérdida L y el conjunto de todas las posibles

muestras M, y la hipótesis $\hat{h}^*$, para la cual se minimiza la pérdida de generalización estimada, se pueden calcular como

$$\begin{aligned} EstLoss_{L,M}(h) &= \frac{1}{|M|} \sum_{(x,y)\in M} L(y, h(x)), \\ \hat{h}^* &= \underset{h\in H}{\operatorname{argmin}}\, EstLoss_{L,M}(h). \end{aligned} \tag{6}$$

2.1.2.3 *Regularización*

Un método alternativo a *k-fold CV* para la selección con base en la complejidad del modelo, es buscar la hipótesis $\hat{h}^*$ que minimiza la suma ponderada del LOSS y la complejidad de la hipótesis, es decir, el *costo total* [12]:

$$\begin{aligned} Costo(h) &= EstLoss(h) + \lambda Complejidad(h) \\ \hat{h}^* &= \underset{h\in H}{\operatorname{argmin}}\, Costo(h). \end{aligned} \tag{7}$$

Donde la función $Complejidad(h)$ se define a partir de los coeficientes o parámetros ajustados θ_i de la hipótesis h, $1 \leq i \leq n$ y x consta de los atributos $(x_1, x_2, \ldots, x_n)$, como

$$Complejidad(h) = \frac{1}{|M|} \sum_{i=1}^{n} \theta_i^2 \tag{8}$$

y $\lambda \geq 0$ es el parámetro de regularización, que sirve como tasa de conversión entre la pérdida de generalización y la complejidad. Ahora, en vez de buscar la hipótesis que generaliza mejor utilizando *k-fold CV* sobre la complejidad del modelo, se busca utilizando *k-fold CV* sobre el parámetro λ. El término regularización se refiere a que se premia funciones más regulares, menos complejas, o que se penaliza funciones más complejas, y esto se logra minimizando el ajuste de la función al conjunto de datos de entrenamiento.

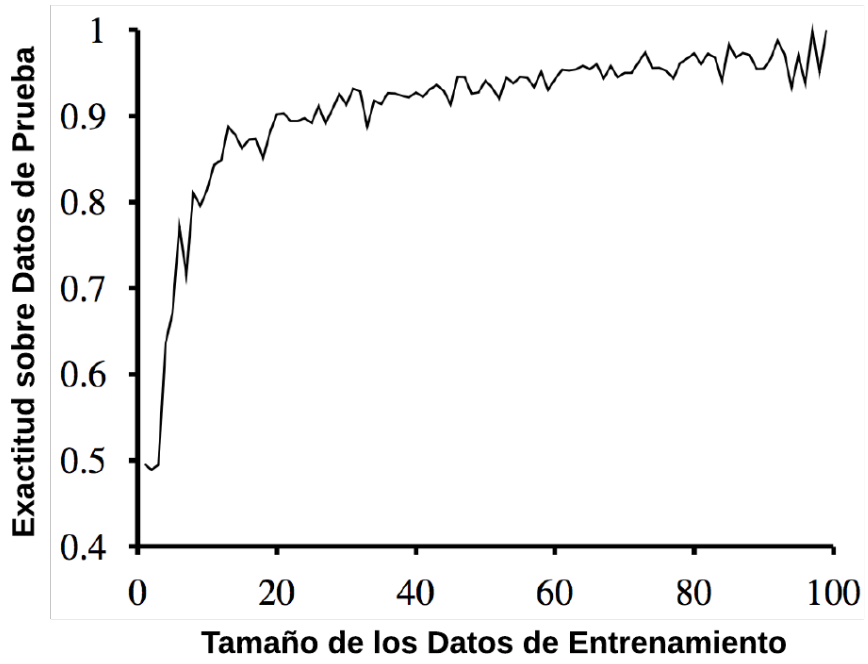


Figura 5: Curva de aprendizaje sobre 100 ejemplos generados aleatoriamente. Cada punto es el promedio de 20 intentos.
Fuente: [12, p. 703]

Otra forma de simplificar los modelos es reducir la cantidad de atributos o dimensiones del modelo. Esto es denominado *selección de características* y consiste en descartar aquellos atributos que parecen ser irrelevantes.

2.1.2.4 *Curvas de Aprendizaje*

En [12] se presenta las *curvas de aprendizaje* como una forma de medir el desempeño de un algoritmo de aprendizaje de máquina. Muestran la exactitud de predicción sobre un conjunto de datos de prueba en función del tamaño del conjunto de datos de entrenamiento (ver Figura 5). También se puede utilizar el error en vez de la exactitud. El método consiste en ajustar la hipótesis h con los datos de entrenamiento y medir su exactitud con los datos de prueba. Por ejemplo, si se cuenta con 100 muestras, se inicia con un conjunto de entrenamiento de tamaño 1 y se incre-

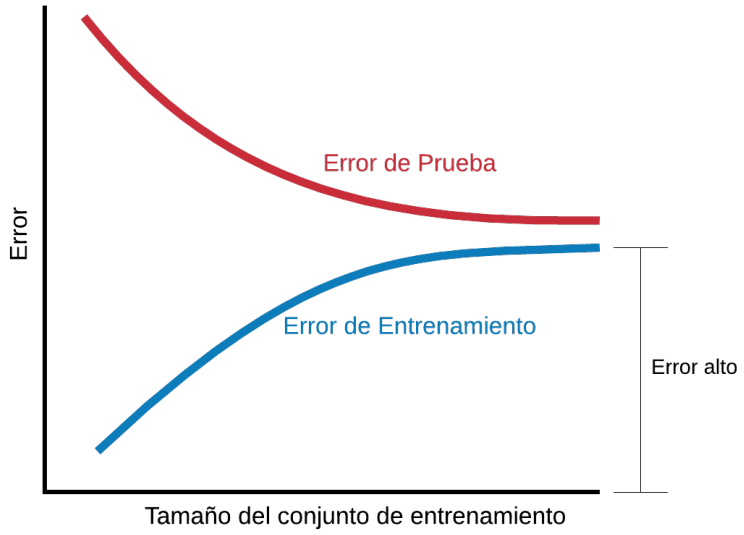


Figura 6: Curva de aprendizaje para un algoritmo con alto bias.

menta el tamaño en uno hasta llegar a 99. Para cada tamaño se repite el proceso de división aleatorio 20 veces, y se promedia el resultado de los 20 intentos.

Esta es una herramienta para diagnosticar si un algoritmo sufre de un problema de bias, un problema de varianza o un poco de ambos. Cuando se aumenta el tamaño del conjunto de entrenamiento, tanto el error de entrenamiento como el error de prueba son altos y parecidos, entonces el algoritmo tiene un problema de alto bias (ver Figura 6). Pero, si el error de entrenamiento es mucho menor que el error de prueba, entonces el algoritmo tiene un problema de alta varianza (ver Figura 7). El algoritmo podría tener un poco de ambos problemas, si la distancia entre error de entrenamiento y error de prueba es grande, y el error de entrenamiento es alto también.

Según [16], cuando la hipótesis obtenida al entrenar un algoritmo genera un error de prueba inaceptablemente alto, se hace necesario depurar el algoritmo. Hay varias rutas de acción que se pueden seguir:

- Conseguir más muestras de entrenamiento (no siempre funciona)

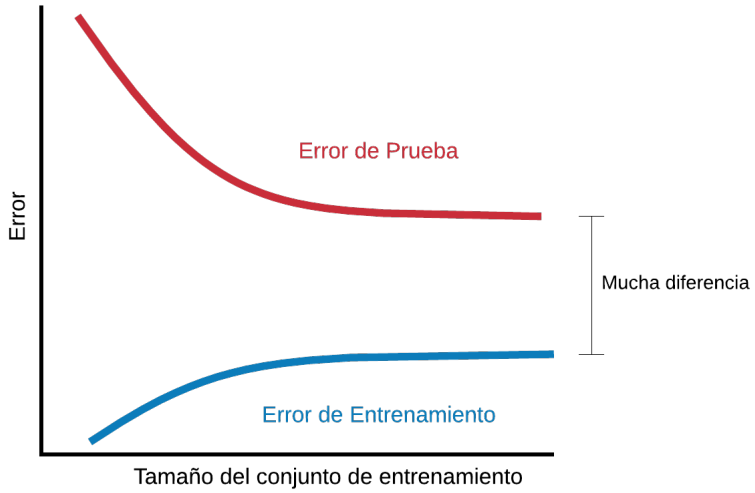


Figura 7: Curva de aprendizaje para un algoritmo con alta varianza.

- Utilizar conjuntos de características más pequeños
- Aumentar el parámetro de regularización λ
- Obtener características adicionales
- Aumentar la complejidad del modelo, por ejemplo, agregar características de tipo polinomial ($X_1^2, X_2^2, X_3^2, X_1X_2, \ldots, X_1^3, X_2^3, X_3^3, X_1X_2X_3, \ldots$, etc)
- Disminuir el parámetro de regularización λ

Las tres primeras pueden servir para corregir un problema de alta varianza, mientras que las tres últimas aplican cuando el algoritmo sufre de alto bias. Por ejemplo, si el algoritmo sufre de alto bias, conseguir más muestras de entrenamiento como única medida, no ayudará a mejorar el rendimiento del algoritmo, pues como puede verse en el comportamiento asintótico de la Figura 6, al aumentar el tamaño del conjunto de entrenamiento el error de prueba no disminuirá. En cambio, cuando el algoritmo sufre de alta varianza, y el error de entrenamiento es aceptablemen-

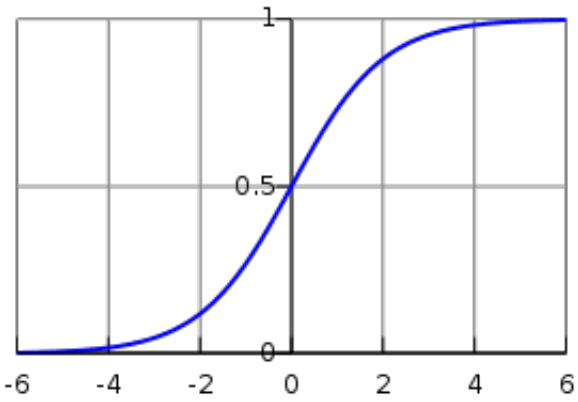


Figura 8: Función sigmoide o logística.

te bajo, conseguir más muestras para entrenar más el algoritmo puede servir para reducir el error de prueba, como se aprecia en la tendencia indicada en la Figura 7.

2.1.3 *Regresión Logística*

El algoritmo de regresión logística se puede utilizar en aprendizaje supervisado tanto para problemas de regresión como de clasificación. De acuerdo con [17], la regresión es un método estadístico en que se explica una variable dependiente o respuesta, con base en otras variables independientes o predictores. Si se tiene un único predictor, entonces el modelo se denomina regresión lineal. Para ajustar modelos mediante regresión logística, se utiliza la metodología de modelado Generalized Linear Models (GLM).

Los parámetros del modelo a ajustar son designados como θ. El modelo utiliza la función logística o sigmoide $logit$ (ver Figura 8). La hipótesis h_θ a ajustar sería

$$h_\theta(x) = logit(\theta^T x), \qquad \text{donde } logit(z) = \frac{1}{1+e^{-z}}. \tag{9}$$

$h_\theta(x)$ es la probabilidad de que la salida sea 1, dado el modelo:

$$h_\theta(x) = P(y = 1|x;\theta) = 1 - P(y = 0|x;\theta). \tag{10}$$

La función de costo que se utiliza para ajustar el modelo es la siguiente:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Costo(h_\theta(x^{(i)}), y^{(i)}), \tag{11}$$

donde

$$Costo(h_\theta(x), y) = -y\ log(h_\theta(x)) - (1 - y)\ log(1 - h_\theta(x)) \tag{12}$$

Esta función de costo garantiza que $J(\theta)$ es convexa para regresión logística, lo cual es conveniente para efectos de optimizar los parámetros de la función. Esta optimización se puede realizar aplicando la técnica *Gradient Descent* (descenso por la pendiente) sobre los parámetros θ:

```
repeat {
    # actualiza todos los θ_j a la vez
    θ_j := θ_j − (α/m) Σ_{i=1}^{m} (h_θ(x^(i)) − y^(i)) x_j^(i)
}
```

Listado 2.1: Gradient descent

El valor α es denominado tasa de aprendizaje, controla cuánto cambian los θ_j en cada iteración.

Existen otros métodos de optimización avanzados, como por ejemplo "*Conjugate Gradient*", "*BFGS*", "*L-BFGS*", "*Stochastic Gradient Descent (SGD)*", "*RMSprop*", "*Adam*", "*Adadelta*". Según sea la herramienta que se utilice para programar y ajustar los algoritmos, se tendrán unos u otros métodos disponibles.

2.1.4 *Máquinas de Soporte Vectorial*

Máquinas de Soporte Vectorial (SVM) es uno de los algoritmos más utilizados para aprendizaje supervisado, que según [12] tiene las siguientes propiedades:

1. Construye un margen separador máximo, una frontera de decisión con la máxima distancia posible a las muestras. Esto permite que generalice bien.

2. Crea un hiperplano de separación lineal, insertando los datos en un espacio dimensional mayor, mediante el *truco del kernel*. Si los datos no son linealmente separables en el espacio original, son separables en un espacio dimensional mayor.

3. Es un método no-paramétrico, los cuales retienen las muestras de entrenamiento y puede que requieran almacenarlas todas. Sin embargo, en la práctica sólo retiene una pequeña fracción de las muestras. Permite representar funciones complejas con un mecanismo resistente al problema de sobreajuste.

El método de regresión logística ajusta un modelo en función de todas las muestras, mientras que SVM identifica unas muestras que son más importantes que otras para lograr clasificarlas, como se muestra en la Figura 9. Encontrar el máximo margen que separa las muestras permite ubicar la frontera de clasificación lo más alejada posible de ambas clases, contribuyendo a que el modelo generalice mejor sobre nuevas muestras.

En vez de minimizar el *LOSS estimado*, SVM minimiza el *LOSS de generalización* esperado. El separador se define como el conjunto de puntos $\{x : w \cdot x + b = 0\}$. Lo que se busca es encontrar la solución óptima para

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (x_j \cdot x_k) \tag{13}$$

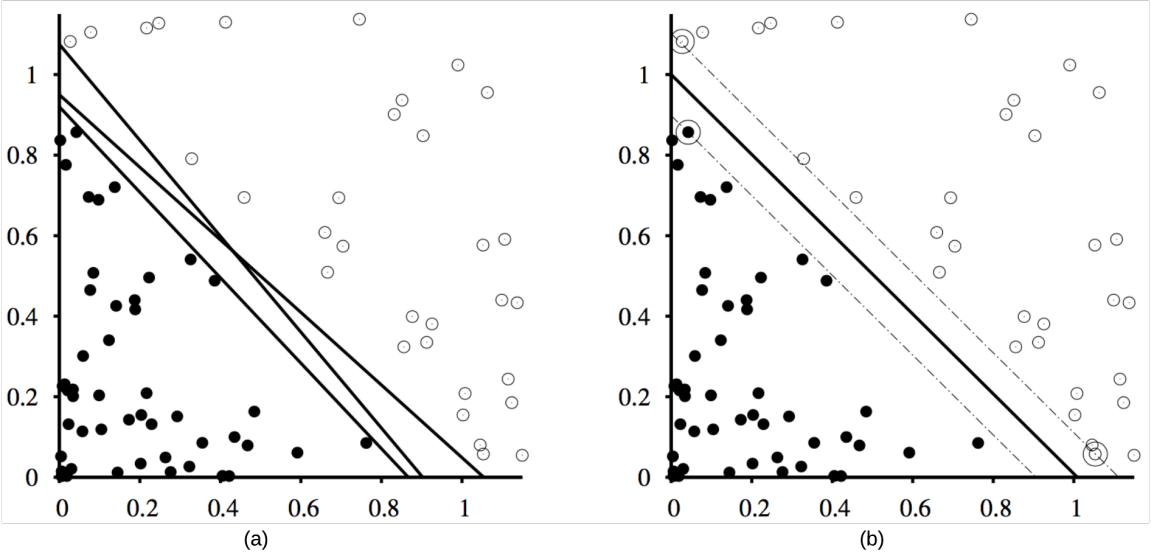


Figura 9: Clasificación con SVM: (a) Dos clases (círculos negros y blancos) y tres separadores lineales posibles. (b) El separador de máximo margen (línea más gruesa) se encuentra en el centro del margen (líneas punteadas). Los *vectores soporte* son las muestras más cercanas al separador y están señaladas con círculos más grandes.
Fuente: [12, p. 745]

donde $\alpha_j \geq 0$ y $\sum_j \alpha_j y_j = 0$. Este es un problema de optimización de *programación cuadrática*. Una vez que se encuentra el vector α se puede recuperar el vector de parámetros w con la ecuación $w = \sum_j \alpha_j x_j$. La función representada por la Ecuación (13) es convexa, por lo cual tiene un único máximo global que puede ser encontrado de manera eficiente.

Los vectores soporte son aquellos para los que el correspondiente peso α_j no es cero, y son denominados así porque sostienen el plano separador. Si los datos no son linealmente separables, se proyectan a un espacio dimensional mayor utilizando una función denominada *kernel* (ver Figura 10). Se puede buscar separadores lineales en el espacio dimensional mayor $F(x)$ reemplazando $x_j \cdot x_k$ en la Ecuación (13) con $F(x_j) \cdot F(x_k) = (x_j \cdot x_k)^2$.

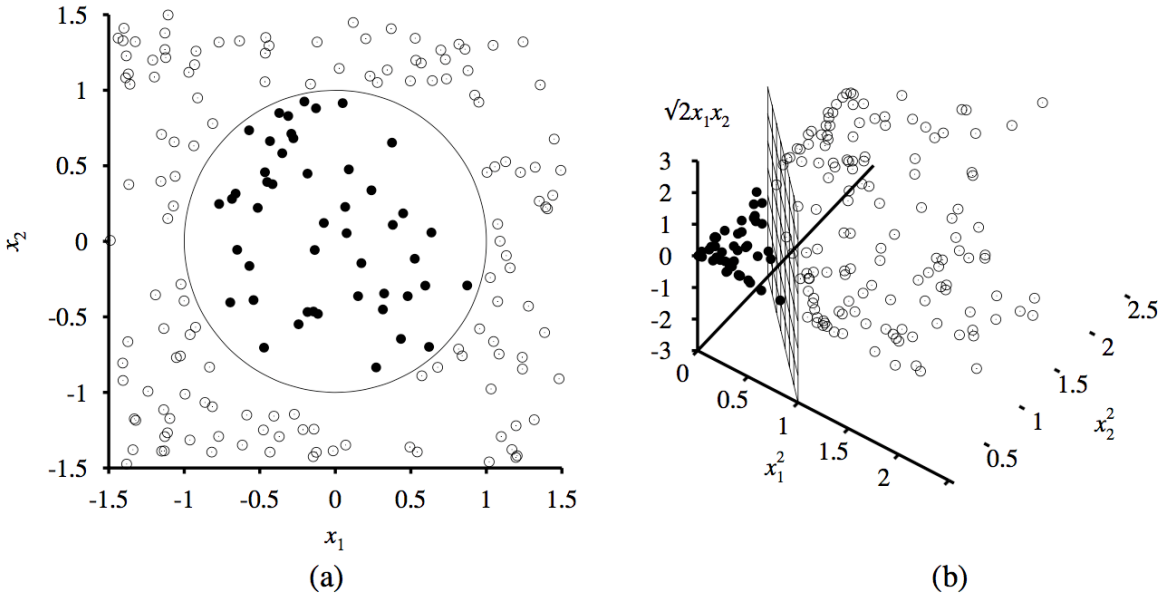


Figura 10: (a) Datos de entrenamiento bidimensional con muestras positivas como círculos negros y muestras negativas como círculos blancos. La frontera de decisión $x_1^2 + x_2^2 \leq 1$ se muestra como un círculo. (b) Los mismos datos después de proyectarlos en un espacio tridimensional $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. La frontera de decisión circular en (a) se transforma en una frontera de decisión lineal en el espacio tridimensional.
Fuente: [12, p. 747]

2.1.5 *Redes Neuronales*

Las Redes Neuronales introducen un enfoque que incorpora conocimientos sobre el funcionamiento del cerebro en la teoría y técnica para diseñar y ajustar modelos a conjuntos de datos. Se parte de la hipótesis de que la actividad mental surge de la actividad electroquímica registrada en redes de células cerebrales o neuronas [12]. El modelo más simple de neurona consiste en una función o combinación lineal de sus entradas para producir una salida que depende de un umbral o peso ajustado para cada entrada. La red neuronal es simplemente un conjunto de estos componentes interconectados para aprender funciones complejas sobre conjuntos de datos de entrenamiento.

Las neuronas en una red se denominan nodos o unidades y están interconectadas por medio de enlaces. Estos enlaces entre unidades sirven para propagar la *activa-*

ción a_i desde la unidad i hacia la unidad j. Cada enlace o sinapsis tiene un *peso* numérico asociado, $w_{i,j}$, que indica la importancia (magnitud) y la dirección (signo) de la influencia de el nodo i sobre el nodo j.

Si una neurona j tiene n entradas provenientes de otras neuronas, se calcula la función de activación sobre la suma ponderada de sus entradas: $a_j = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$. Si la función de activación $g(z)$ consiste en un umbral dado, esta función no es derivable cuando $z = 0$, se la denomina *perceptrón*, y si es una función logística, se la denomina *perceptrón sigmoide*.

Existen dos formas de interconectar las neuronas en una red:

- Red *feed-forward*: Tiene conexiones únicamente en una dirección, es un grafo dirigido acíclico.
- Red *recurrente*: Además de las conexiones hacia adelante, agrega conexiones desde las salidas de cada neurona hacia sus propias entradas.

Las redes *feed-forward* carecen de más estado interno que los pesos en sus enlaces y son una función de sus entradas actuales. En las redes *recurrentes*, los niveles de activación de la red forman un sistema dinámico que puede llegar a un estado estable o mantenerse inestable, o también puede presentar un comportamiento caótico. En las redes recurrentes, la respuesta obtenida por la red depende de su estado inicial, el cual puede estar influenciado por entradas previas. Se dice que estas podrían emular memoria de corto plazo [12].

Finalmente, las redes neuronales *feed-forward* son estructuradas en capas, de modo que cada unidad recibe entradas únicamente del nivel que la precede inmediatamente. Las unidades de las capas internas, distintos del nivel de entradas y el nivel de salidas, se denominan unidades ocultas, y las capas en que se ubican, capas ocultas.

Para abordar problemas de clasificación se ajustan modelos de redes neuronales del tipo feed-forward. El diseño de los modelos consiste en elegir las características

del modelo (entradas y salidas), la cantidad y cardinalidad de los capas ocultas, las funciones de activación, el método de optimización y la función de LOSS, y la métrica para evaluar el modelo.

El diseño de redes neuronales no es trivial, de acuerdo con [14], los modelos por lo general cuentan con demasiados parámetros y el problema a optimizar es no convexo e inestable. Los problemas más importantes a resolver son los siguientes:

2.1.5.1 *Valores Iniciales*

Si los pesos iniciales son muy cercanos a cero, la parte operativa de la función sigmoide es casi lineal y la red neuronal se convierte en un modelo aproximadamente lineal. Se recomienda por esto, iniciar con pesos aleatorios cercanos a cero. El modelo inicia casi lineal y se convierte en no lineal a medida que los pesos se incrementan. No sirve utilizar pesos iniciales en cero porque provoca que todas las neuronas del mismo nivel ajusten los pesos de manera simétrica, y utilizar pesos grandes normalmente conduce a resultados deficientes.

2.1.5.2 *Sobreajuste*

Es común que las redes neuronales tengan muchos parámetros y sobreajusten los datos en el mínimo global de la red. Para evitar esto se entrena el modelo hasta justo antes de alcanzar el mínimo global. Como los pesos inicial con una solución casi lineal, la cual corresponde a una altamente regularizada, esto tiene el efecto de ajustar el modelo lo más cercano a un modelo lineal. Para encontrar este punto óptimo se utilizan los datos de validación, detectando el momento en que empieza a incrementarse el error de validación. Otros métodos de regularización son *declive de pesos*, agregando penalización de los parámetros mediante un parámetro de afinación λ, y *eliminación de pesos*, con un efecto más pronunciado que el anterior.

2.1.5.3 *Normalización*

Escalar las características del conjunto de entrada puede tener un efecto positivo sobre la calidad del modelo ajustado. Lo común es estandarizar todas las entradas para que tengan media igual a cero y desviación estándar igual a 1. Esto asegura que todas las entradas sean tratadas igualmente en el proceso de regularización. También posibilita seleccionar un rango significativo para los pesos iniciales, lo típico es tomar pesos uniformemente aleatorios sobre el rango $[-0{,}7, +0{,}7]$.

2.1.5.4 *Cantidad de unidades y capas ocultas*

Siempre es mejor tener más unidades ocultas que menos. Si se tienen muy pocas, podría suceder que el modelo no tenga suficiente flexibilidad para captar la complejidad de los datos cuando es no lineal; y si se tienen muchas, es posible que los pesos extra se plieguen hacia el cero si se utiliza la cantidad adecuada de regularización. La cantidad típica de unidades oculta oscila entre 5 y 100, la cual se incrementa con la cantidad de entradas y de muestras de entrenamiento. Lo recomendado es colocar una cantidad razonablemente grande de unidades y ajustarlas con regularización. Se puede utilizar *k-fold CV* para estimar la cantidad óptima de unidades, pero es mejor utilizar *k-fold CV* para estimar el parámetro λ de regularización. La selección de la cantidad de capas ocultas se basa en el conocimiento del problema y en la experimentación. Cada capa extrae características de la entrada para implementar la regresión o clasificación, por lo que agregar varias capas ocultas permite que surjan características jerarquizadas en distintos niveles.

2.1.5.5 *Mínimos locales*

La función de error de la red es no convexa, normalmente posee muchos mínimos locales. La solución final depende mucho de los pesos iniciales, pues podría ser que el modelo converja en un mínimo local muy alejado del óptimo. Por esto se hace

necesario intentar varias configuraciones iniciales aleatorias y escoger la que ofrezca el error penalizado menor.

2.2 REVISIÓN DE LITERATURA

2.2.1 *Minería de Datos, Predicción y Learning Management System*

El concepto clave en el modelo de datos del módulo TAM es la interacción o evento, este es el tipo de objeto que se registra para ser rastreado. Cada interacción se define mediante una expresión regular, la instancia del paquete asociado a la interacción, la colección de comandos asociados con esta y su estado. Cada comando asociado con la interacción se relaciona con una plantilla de comando y un contexto. Una plantilla de comando pertenece a un tipo de paquete y está disponible para todas las instancias de paquetes del mismo tipo. El contexto es definido por una expresión regular que es una sub-expresión de la definida para la interacción, si la URL solicitada concuerda con la expresión regular del contexto, entonces todos los comandos asociados con este serán ejecutados por el servicio de rastreo.

Según Couchet et al. [9], la arquitectura de rastreo y auditoría para administrar los datos de las interacciones del sistema proporciona un mecanismo potente para desarrollar funcionalidades adaptativas para los usuarios finales. Los autores desarrollaron un modelo para minar, analizar y categorizar las necesidades de los usuarios de .LRN a partir de las interacciones registradas por el TAM, mediante la implementación de algoritmos de aprendizaje de máquina no supervisados, en particular mapas auto-organizados o Self-Organizing Maps (SOM) combinados con especificaciones de diseño de escenarios de aprendizaje. El modelo propuesto consiste en un *pipeline* con tres niveles secuenciales e independientes: el primer nivel aplica un Recurrent Emergent Self-Organizing Map (R-ESOM) y codificación fractal

sobre la secuencia de interacciones en una sesión .LRN para obtener sus propiedades globales emergentes. Estas propiedades surgen del comportamiento colectivo y la interrelación de las distintas secuencias de interacciones de los usuarios de .LRN. En el segundo nivel, estas propiedades son extraídas automáticamente con un algoritmo clasificador Local Outlier Factor (LOF), y son visualizadas como las diferentes regiones de un mapa Local Outlier Factor-Recurrent-Emergent Self Organizing Map (LR-ESOM). El tercer nivel aplica un algoritmo de clasificación sobre el mapa obtenido para asignar una categoría al estudiante según sus necesidades de aprendizaje.

Santos et al. [18] describen un servicio de recomendación que utiliza técnicas de aprendizaje de máquina sobre las interacciones de los estudiantes, integrado con la herramienta OpenACS/.LRN por medio de servicios web. Las técnicas empleadas son *Content Based Recommendation System* para identificar los ítems que pueden interesarle al estudiante, y *Collaborative Filtering Algorithms* para predecir la utilidad de los ítems para cada usuario particular.

En [19], Boticario y Santos proponen un enfoque de modelado de usuarios para desarrollar LMS adaptativos, mediante la aplicación de técnicas de *Collaborative Filtering* para identificar estudiantes similares con base en estilos de aprendizaje, niveles de conocimiento e interés, y las interacciones colaborativas implícitas en el curso y rastreadas por TAM. Según los autores, en este enfoque se pretende dirigir la atención del autor de los escenarios de aprendizaje hacia los elementos de administración y control dentro de los escenarios (diseño de actividades de aprendizaje, restricciones temporales, evaluaciones) y no tanto a una descripción completa de las rutas de aprendizaje alternativas para los distintos estilos de estudiantes.

Detoni et al. [20] utilizan modelos basados en algoritmos de aprendizaje de máquina para predecir el éxito o fracaso de un estudiante en un curso a partir de las interacciones registradas en la plataforma Moodle, tanto de los estudiantes,

como de los tutores y profesores. Los modelos generados emplearon las técnicas siguientes: Redes Bayesianas, Redes Neuronales, J48 (árboles de decisión) y *Random Forest*.

Hirumi [21] [22] ha profundizado en el análisis y el diseño de interacciones en LMS. En Rodrigues, Medeiros y Gomes [23] se hace referencia a la sistematización de los tipos de interacciones posibles en LMS: profesor/estudiante, estudiante/estudiante, estudiante/contenido, profesor/profesor, profesor/contenido, contenido/-contenido, profesor/herramientas, estudiante/herramientas. Los autores realizan un análisis de regresión sobre la relación entre interacciones y desempeño, con el objetivo de obtener un modelo de predicción del desempeño de los estudiantes a partir de las interacciones durante la realización de un curso sobre una implementación de la plataforma Moodle. Las variables incluidas en el análisis fueron: el desempeño final, las interacciones en foros, en video aulas, en acceso a documentos, en acceso a presentaciones y la hora promedio de acceso a la plataforma. Se concluye que al incrementar la cantidad de interacciones de los estudiantes se tiende a aumentar el desempeño académico de los mismos, y que por esto es posible estimar el desempeño de los alumnos basándose en la cantidad de interacciones en herramientas del tipo foro de discusión.

Gotardo, Massa y Hruschka [24] realizaron experimentos con una base de datos de interacciones de un conjunto de alumnos y de cursos en un ambiente Moodle de una Universidad a Distancia. Las interacciones registradas incluían entrega de tareas, quices, acceso a foros, acceso y subida de archivos, cantidad de accesos e información sobre aprobación/reprobación. En su implementación utilizaron *Collaborative Filtering* basado en los algoritmos J48 y Naïve Bayes, y la técnica de *bootstrapping* para amplificar el conjunto de datos de entrenamiento. El resultado de su trabajo es un sistema de recomendación con base en el desempeño, variables demográficas e interacciones en los cursos.

2.2.2 *Minería de Datos Educativos*

Shahiri et al [25] realizaron una revisión sistemática de literatura, siguiendo los lineamientos propuestos por [26], sobre predicción de éxito académico mediante la utilización de técnicas de minería de datos, y el uso de los algoritmos de predicción para identificar las variables con mayor contenido semántico respecto al rendimiento en la información de los estudiantes. El período de tiempo del estudio comprende desde el año 2002 hasta el 2015. Los autores encontraron que los atributos utilizados para predecir el desempeño de los estudiantes, ordenados descendentemente según la cantidad de estudios en que se usaron, son promedio ponderado o Cumulative Grade Point Average (CGPA) y evaluación interna (asignaciones, quices, laboratorios, exámenes, asistencia, etc.), seguidos por variables demográficas (género, edad, contexto familiar, discapacidades, etc.) y evaluación externa (nota promedio final del curso), y finalmente, actividades extra-curriculares, contexto educativo previo, redes de interacción social y factores psicométricos. Los métodos de predicción utilizados son clasificación, regresión y categorización. La más utilizada es clasificación, y dentro de este método los algoritmos más utilizados son *decision tree*, redes neuronales, *naive bayes*, *K-nearest neighbor* y máquinas de soporte vectorial. Con respecto al máximo porcentaje de exactitud conseguido con la aplicación de estas técnicas de clasificación en los estudios revisados, al utilizar redes neuronales con evaluación interna/externa se reporta un 98 % de exactitud, seguido por *decision tree* con CGPA y evaluación interna/externa se reporta un 90-91 % de exactitud, *K-nearest neighbor* con CGPA y evaluación interna se reporta un 83 % de exactitud, máquinas de soporte vectorial con CGPA y evaluación interna se reporta un 80 % de exactitud, y *naive bayes* con CGPA se reporta un 76 % de exactitud.

Según Al-Razgan et al [27] la aplicación de algoritmos de minería de datos al análisis de datos masivos en el ámbito educativo ha generado una área de investi-

gación multidisciplinaria denominada Educational Data Mining (EDM). Realizaron otra revisión sistemática de literatura sobre EDM entre 2006 y 2013 con base en el paper más citado en el dominio, "Educational data mining: A survey from 1995 to 2005" de Romero y Ventura [28], utilizando encadenamiento hacia adelante de las referencias a esta publicación. Los estudios realizados en esos años se orientaron a dominios de aplicación que incluyen juegos educativos, objeto de aprendizaje, aprendizaje móvil, aprendizaje personalizado, investigación científica sobre aprendizaje y sistemas inteligentes de tutoría; el foco de atención se concentra en investigación científica sobre aprendizaje, objeto de aprendizaje y aprendizaje personalizado. En cuanto a las tareas dentro del dominio de aplicación a resolver utilizando minería de datos, reportan que las categorías más direccionadas son predicción del desempeño académico, análisis y visualización de datos, modelo de estudiante y construcción automática de cursos. Finalmente señalan la necesidad de que las herramientas desarrolladas para EDM deberían ser integradas en algún LMS o proceso de aprendizaje, y ser fáciles de usar, atractivas y adaptables.

En Almosallam et al [29] se propone una distinción entre Educational Data Mining, Learning Analytics (LA) y Academic Analytics (AA). Según los autores, LA se enfoca en temas educativos y en cómo optimizar las oportunidades de mejorar el aprendizaje, AA se concentra en asuntos administrativos y en cómo ayudar a los administradores en educación universitaria a realizar mejores oportunidades y decisiones de aprendizaje. AA emplea técnicas estadísticas y modelos predictivos para ayudar a identificar estudiantes con dificultades académicas para facilitar procesos de intervención temprana. EDM se concentra en los retos técnicos y en cómo extraer valor a partir de grandes volúmenes de datos relacionados con el aprendizaje. Desde un punto de vista técnico, sugieren que EDM es la aplicación de técnicas de minería de datos con el objetivo de apoyar a educadores y estudiantes en el análisis del proceso de aprendizaje. La diferencia entre los tres campos es bastante sutil,

pues todas tienen objetivos, dominio de análisis, fuentes de datos y procesos muy similares. LA incluye otros métodos al uso de minería de datos, como herramientas estadísticas y de visualización o técnicas de Social Network Analysis (SNA), aplicadas al mejoramiento de los procesos de enseñanza-aprendizaje.

Romero y Ventura [28] [30] [31] han realizado varias revisiones de literatura sobre EDM. En *Data mining in education* señalan que EDM ha surgido como área de investigación en años recientes y pretende desarrollar, investigar y aplicar métodos computacionales para detectar patrones en conjuntos de datos educativos masivos. Es un área interdisciplinaria en la que convergen ciencias de la computación, educación y estadística (ver Figura 11). El interés creciente en el área es sugerido por un crecimiento de forma exponencial en la cantidad de artículos sobre el tema entre los años 2004 y 2011. En cuanto al tipo de ambiente educativo, se clasifican estos en dos grandes ramas, ambientes educativos tradicionales y Sistemas Educativos Basados en Computación (CBE). Los ambientes educativos tradicionales privilegian las clases presenciales, se estratifican en Pre-escolar, Primaria, Secundaria, Universitaria y Educación Especial, y recolectan información que es registrada en bases de datos tradicionales. Los CBE aprovechan la Internet mediante la implementación de sistemas de LMS, Intelligent Tutoring Systems (ITS), Adaptive and Intelligent Hypermedia Systems (AIHS), *Test and Quiz Systems* y otros. Entre los métodos más populares utilizados en EDM, identifican: predicción, *clustering*, detección de casos atípicos, minería de relaciones, SNA, minería de procesos y minería de texto. Otros métodos particulares de EDM son: destilado de datos, aplicación de modelos, trazado de conocimiento y Nonnegative Matrix Factorization (NMF).

La tesis a desarrollar se enmarcará dentro de este área de investigación, como aplicación del método de predicción en la franja correspondiente a Educación Universitaria y sobre una implementación particular de LMS.

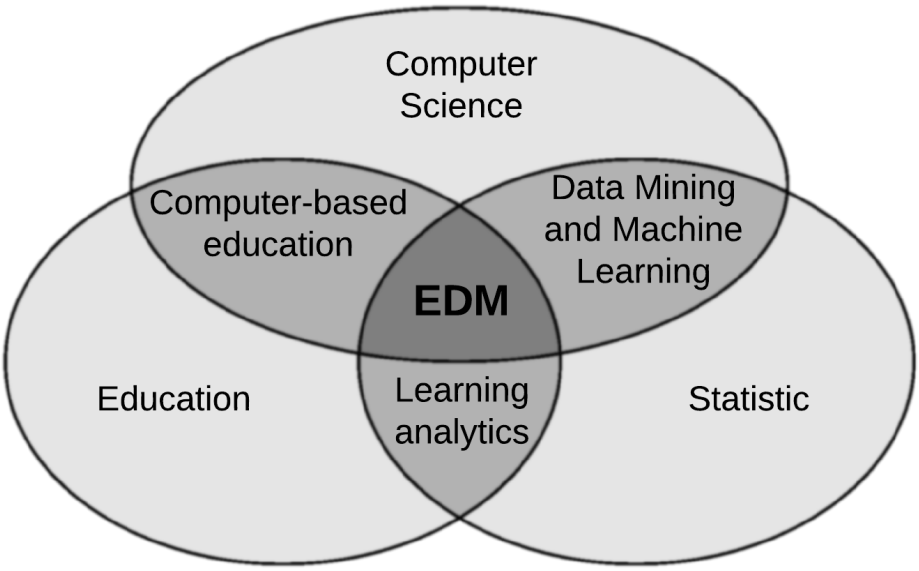


Figura 11: Áreas principales relacionadas con EDM
Fuente: [31, p. 13]

3 ANÁLISIS DE RESULTADOS

La metodología empleada para llevar a cabo los objetivos de este trabajo consiste en tres etapas. La primer etapa corresponde a la extracción y depuración del modelo de datos (ver Objetivo Específico 1). Seguidamente, la etapa de implementación y entrenamiento de los algoritmos, y comparación del desempeño obtenido por cada uno (ver Objetivos Específicos 2 y 3). Y finalmente, una etapa final de ajuste y validación del modelo diseñado y seleccionado en la etapa previa (ver Objetivos Específicos 4 y 5).

3.1 MODELO DE DATOS

En este primer momento se extrajeron todas las interacciones registradas por el módulo TAM de la plataforma del TEC Digital, y el historial académico de los estudiantes correspondientes al año 2016. Se hizo la suposición de que las muestras son independientes y tienen la misma distribución.

Seguidamente se eliminaron los registros de interacciones correspondientes a actores que no son estudiantes o que son estudiantes pero no reflejan interacciones asociadas con cursos semestrales activos y en planes de bachillerato universitario o licenciatura continua.

También se filtraron aquellos estudiantes que no finalizaron el curso, y sólo se consideraron registros de estudiantes que sí finalizaron el curso, y lo aprobaron o

Cuadro 1: Categorías empleadas por el módulo TAM

Id	Categoría	Interacciones	%
1	Assessment	14,511	0.13
2	Chat	8	0.00
3	Survey	1,442	0.01
4	Attendance	2,259	0.02
5	Evaluation	904,938	7.99
6	FAQ	1,533	0.01
7	File Storage	5,533,101	48.80
8	News	192,486	1.70
9	TDA Integration	8	0.00
10	TDA Game Engine	20,174	0.18
11	dotLRN	4,383,740	38.70
12	Forums	164,735	1.45
13	TDA GDI	6	0.00
14	dotLRN Homework	74,071	0.65
15	Attachments	4,695	0.04
16	Calendar	28,581	0.25
17	TDA SMS Configuration	292	0.00

lo reprobaron. Es decir, no se tomaron en cuenta los cursos que fueron retirados, congelados o registrados como incompletos.

Las categorías de las interacciones utilizadas en el modelo de datos (ver Cuadro 1) dependen de la categorización configurada actualmente en la plataforma del TEC Digital, y fueron incorporadas como variables en el modelo de datos. Las interacciones se contabilizaron por semana y por categoría, para cada estudiante en cada curso, de manera que el conteo de las interacciones correspondientes a una semana y una categoría constituye una variable (atributo, característica o predictor) en el modelo de datos. La aprobación o reprobación del curso será la variable a predecir o respuesta, los niveles de esta variable serán cero para reprobación y uno para aprobación.

El grueso de las interacciones caen bajo las categorías de Acceso a Archivos y Navegación en la Plataforma, con un 49 % y un 39 % del total, respectivamente, y

Cuadro 2: Cantidad y porcentaje de interacciones por semana

Semana	Interacciones	%
1	540,127	4.77
2	691,995	6.11
3	704,344	6.22
4	674,967	5.96
5	722,987	6.38
6	661,786	5.84
7	496,828	4.39
8	652,048	5.76
9	636,466	5.62
10	630,790	5.57
11	658,237	5.81
12	985,923	8.70
13	601,197	5.31
14	586,878	5.18
15	649,549	5.73
16	685,255	6.05
17	747,203	6.60

un relativamente marginal 8 % correspondiente a la categoría de Evaluaciones. Las categorías restantes abarcan poco más del 4 % del total de interacciones registradas en el sistema.

En el Cuadro 2 se muestra la cantidad de interacciones correspondientes a cada semana de ambos semestres en el año 2016. También se muestra el porcentaje del total de interacciones al que corresponde dicha cantidad. Nótese que la cantidad de interacciones es uniforme en todas las semanas del semestre.

Se utilizaron sólamente las interacciones en el TEC Digital, sin ningún otro tipo de variable (ver Sección 2.2.2 sobre variables demográficas, CGPA, evaluación externa, contexto educativo previo y factores psicométricos), porque aunque estas podrían tener cierta capacidad predictiva sobre el éxito académico, esta capacidad sería estática con respecto al transcurrir del curso en el tiempo durante el semestre. Tampoco se incluyeron variables del tipo evaluación interna o actividades extra-

curriculares porque se pretende determinar si las interacciones por sí mismas permiten predecir con exactitud el éxito académico.

Se eliminaron 5 muestras del total debido a que presentaban cantidades anómalas de interacciones, como se muestra en el Cuadro 3.

Cuadro 3: Cantidades anómalas registradas en interacciones

Muestra	**Semana**	**# Interacciones**
1	17	7,524
2	15	27,432
3	12	385,267
4	2	13,750
5	11	24,696

El conjunto de datos que se obtuvo consta de 84,808 muestras. Cada muestra contiene las interacciones contabilizadas por semana y categoría para cada materia cursada por cada estudiante. Como son 17 categorías y 12 semanas desde el inicio de cada semestre, el modelo cuenta con 204 predictores, cuyos posibles valores varían en el rango [0..5000]. La variable de respuesta es si el estudiante aprobó o no aprobó la materia, donde el valor 0 significa que aprobó y el valor 1 que reprobó.

Un modelo con menos predictores consistiría en contabilizar las interacciones por semana únicamente, con lo cual se obtendría un modelo con 17 atributos. Este modelo podría ser más rápido de ajustar, pero también sería más proclive a sufrir de problemas de alto bias.

Para poder predecir conforme transcurren las semanas del semestre, fue necesario ajustar un modelo para cada semana. Las interacciones empleadas en dichos modelos consistían en las correspondientes a las semanas transcurridas hasta entonces. Para ajustar el modelo de la tercera semana, se utilizaron las interacciones acumuladas durante cada una las primeras tres semanas según cada categoría, es decir, se tenían 51 predictores. Para ajustar el de la cuarta semana, se utilizaron las

de la semana uno a la semana cuatro, y se tenían 68 predictores. Y así sucesivamente, hasta llegar a ajustar el último modelo con todas las interacciones desde la semana uno hasta la semana doce. Se eligió la cantidad de diez modelos, porque suena poco factible que en las últimas cuatro semanas se consiga revertir lo que no se pudo en las doce primeras, y porque en la primera y segunda semana se cuenta con muy pocas variables.

Los valores de los predictores se normalizaron utilizando la fórmula $\frac{X-min(X)}{max(X)-min(X)}$ (ver Listado 4.1).

3.2 COMPARACIÓN DE ALGORITMOS

Se utilizó como plataforma computacional el *Cluster Kabré*[1] del Centro Nacional de Alta Tecnología (CeNAT) [32], que ofrece una infraestructura de High Performance Computing (HPC) para "la ejecución de simulaciones científicas y procesamiento de altos volúmenes de datos." El cluster cuenta con 2 tipos de nodos:

- 5 Nodos Cadejos: cada nodo con dos procesadores Intel Xeon E5530, con un total de 8 núcleos físicos, 16 hilos de ejecución y 32GB de RAM. Los nodos cadejos-0, cadejos-1 y cadejos-2 cuentan con 2 tarjetas NVIDIA Tesla C1060 cada uno.

- 20 Nodos Zarate: cada nodo tiene un procesador Intel Xeon Phi con 64 núcleos físicos, 256 hilos de ejecución y 96GB de RAM.

El cluster utiliza el sistema de colas denominado *Torque* para someter los trabajos, en conjunto con un software calendarizador denominado Maui. También se puede utilizar Message Passing Interface (MPI) para hacer programación paralela en el lenguaje C.

1 *Kabré* es una palabra *Ngäbe* que significa *montón*.

Los módulos instalados en el cluster que se utilizaron para programar fueron los siguientes:

- R y biblioteca *glmnet* [33] para *Regresión Logística*.
- Python y biblioteca *libsvm* (tanto la versión para CPus [34] como la que utiliza GPUs [35], escritas en C y Python) para *SVM*.
- Python, Keras [36] y TensorFlow [37] para *Redes Neuronales*.

El modelo de datos obtenido se utilizó como entrada a los algoritmos de aprendizaje de máquina: ***Regresión Logística, Máquinas de Soporte Vectorial y Redes Neuronales***. Para ver si los algoritmos sufrían de alta varianza o alto bias, se utilizó la técnica de la curva de aprendizaje (por ejemplo, ver Listado 4.2). En el caso de que los algoritmos tuviesen alguno de estos problemas, se emplearían las técnicas indicadas en la Sección 2.1.2.4. En particular, se pueden agregar características polinomiales para abordar un problema de alto bias (por ejemplo, ver Listado 4.3), y realizar una búsqueda de la complejidad del mejor modelo (por ejemplo, ver Listado 4.5). Sin embargo, al aumentar el grado polinomial de las características se aumenta la cantidad de predictores de manera exponencial (ver Cuadro 4), y el poder computacional requerido se vuelve prohibitivo, porque también se necesita aumentar la cantidad de muestras requeridas para ajustar el modelo. Este problema es conocido como la *maldición de la dimensionalidad* [38]: a medida que la cantidad de características o dimensiones crece, la cantidad de datos requerida para generalizar con exactitud crece de manera exponencial. Tomando en cuenta las 17 semanas, si la cantidad de interacciones se consolidan por semana se tendrían 17 predictores, y la cantidad aumenta según el grado polinomial como se indica en la columna 2; si la cantidad de interacciones se toman por semana y por categoría se tendrían 289 predictores, y la cantidad aumenta como se indica en la columna 3.

Cuadro 4: Cantidad de predictores según grado polinomial

Grado	**17 semanas**	**17 semanas** × **17 categorías**
2	170	2,890
3	986	16,762
4	4,862	82,654
5	20,366	346,222

Para esta etapa de comparación se utilizó un subconjunto de los datos, con el fin de reservar un subconjunto de los mismos para el ajuste y validación, en la siguiente etapa, del algoritmo seleccionado al finalizar esta. Los datos se dividieron entonces en un 25 % para la comparación de algoritmos y el restante 75 % para el ajuste y la validación del algoritmo seleccionado (ver Listado 4.1).

El subconjunto de datos para la comparación de algoritmos se subdividió a su vez en dos subconjuntos distintos: un 50 % de los datos para diagnosticar los algoritmos mediante curva de aprendizaje (25 % para entrenar y 25 % para validar) y el restante 50 % para el ajuste de modelos (selección de parámetros y nivel de complejidad) mediante *k-fold CV* (25 % para entrenar y 25 % para validar). Esta división de los datos tiene como finalidad evitar agravar cualquier problema de sobreajuste que se pueda presentar.

Para la escogencia de modelo y de algoritmo se utilizó la medida de exactitud de predicción, es decir, el algoritmo que obtenga la mayor exactitud de predicción al evaluarlo con los datos de prueba, y que sea aceptable, al menos un 80 %, para que tenga utilidad práctica. En [20] se reporta una precisión de clasificación entre el 75 % y el 95 % en un ambiente de experimentación semejante al nuestro.

A veces la calidad de la exactitud de predicción puede verse comprometida por la distribución de los datos, por ejemplo cuando el conjunto de datos tiene una cantidad mucho mayor, por ejemplo un 95 %, de una clase con respecto a la otra (*skewed*

classes), y el modelo tiende a predecir la clase mayoritaria sin desarrollar la capacidad de identificar los casos correspondientes a la otra clase. Es conveniente tener una forma de medir la calidad de la exactitud de predicción. Para esto se utiliza las medidas de precisión, $P = \frac{V^+}{V^+ + F^+}$, y de recuperación, $R = \frac{V^+}{V^+ + F^-}$, donde V^+ es la cantidad de verdaderos positivos, F^+ es la cantidad de falsos positivos y F^- es la cantidad de falsos negativos. La precisión representa la fracción de estudiantes de todos los identificados como reprobados por el modelo que realmente reprobaron. La recuperación representa la fracción de estudiantes que realmente reprobaron de todos los identificados como reprobados por el modelo. La métrica a utilizar para verificar la calidad de la exactitud sería $F_1 = \frac{2PR}{P+R}$ [39], que establece un balance entre precisión y recuperación. Si la precisión, o la recuperación, o ambas, son muy bajas, F_1 será bajo. Entre varios modelos con la misma exactitud de predicción, se debe escoger aquel que tenga el mayor F_1. Esta métrica se aplica en vez de la exactitud, cuando se hace *k-fold CV* para escoger los parámetros y la complejidad del modelo. Una métrica semejante es *Area Under the ROC Curve (AUC)* [40].

3.2.1 *Regresión Logística*

La curva de aprendizaje para el algoritmo de Regresión Logística sobre el conjunto de datos sin características polinomiales (ver Figura 12) indica que el modelo presenta el problema de alto bias (el error de entrenamiento es superior al 30 %). En la Figura 13 se muestra gráficamente las métricas de precisión, recuperación y F_1 obtenidas durante la ejecución de la curva de aprendizaje con Regresión Logística. Nótese que aunque la precisión aumenta con la cantidad de datos de entrenamiento, la recuperación por el contrario disminuye y ronda aproximadamente el 0.01, dando como resultado que la métrica F_1 sea decreciente y muy baja. Esto parece

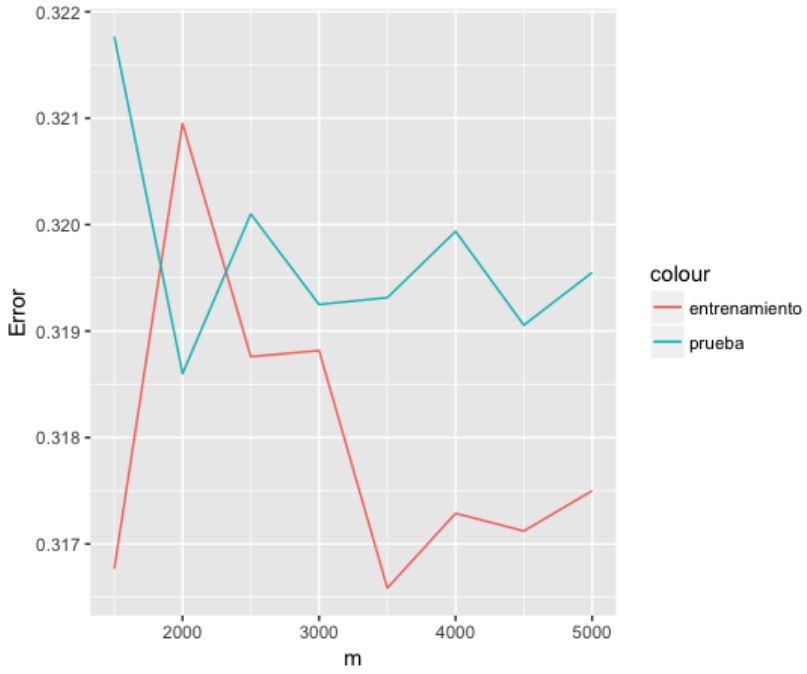


Figura 12: Curva de Aprendizaje para Regresión Logística

indicar que el modelo no muestra capacidad para aprender a reconocer los casos de reprobación.

Dado que el algoritmo de Regresión Logística parece sufrir de alto bias, se modificó el modelo agregando características polinomiales (ver Sección 2.1.2.4), pues como se muestra en la Figura 14 no hay posibilidad de disminuir el error aumentando el tamaño del conjunto de datos.

La curva de aprendizaje sobre el conjunto de datos con características polinomiales de segundo grado (ver Figura 15) indica que el algoritmo aún no supera el problema de alto bias. Se podría haber aumentado el grado polinomial de los predictores, pero esto habría requerido incrementar exponencialmente tanto el tiempo para entrenar el modelo como la cantidad de datos necesarios para esto, y no se disponía de ninguno. También se podría haber agregado otras variables al modelo, pero esto estaba fuera del alcance de este trabajo. Las métricas sobre precisión, recuperación y F_1 continúan siendo muy pobres para el nuevo modelo (ver Figura 16), por lo que conserva la incapacidad de aprender a reconocer los casos de reproba-

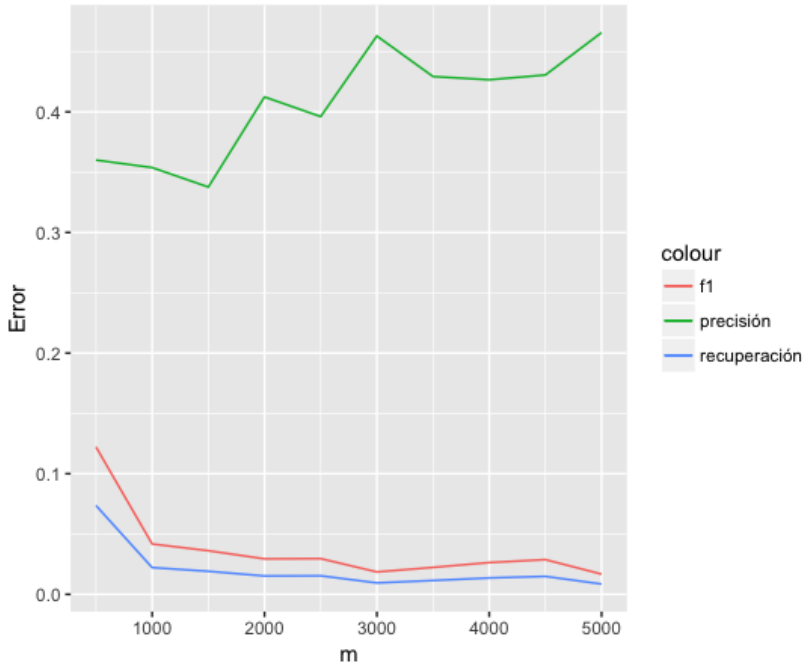


Figura 13: Calidad de Exactitud en Curva de Aprendizaje para Regresión Logística

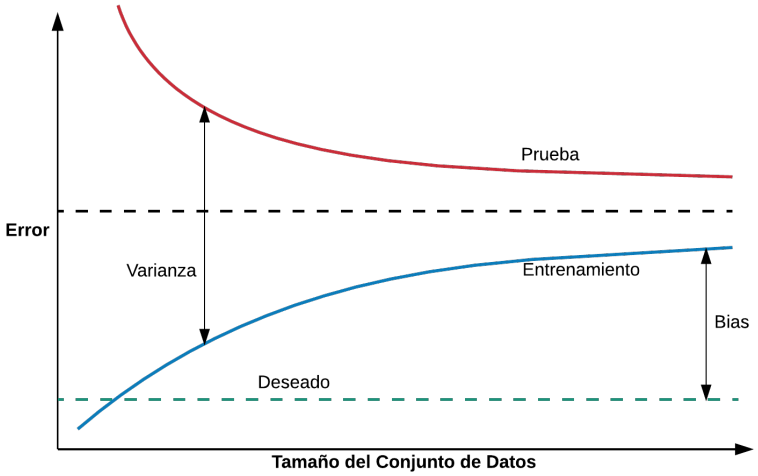


Figura 14: Componentes del error en Curva de Aprendizaje

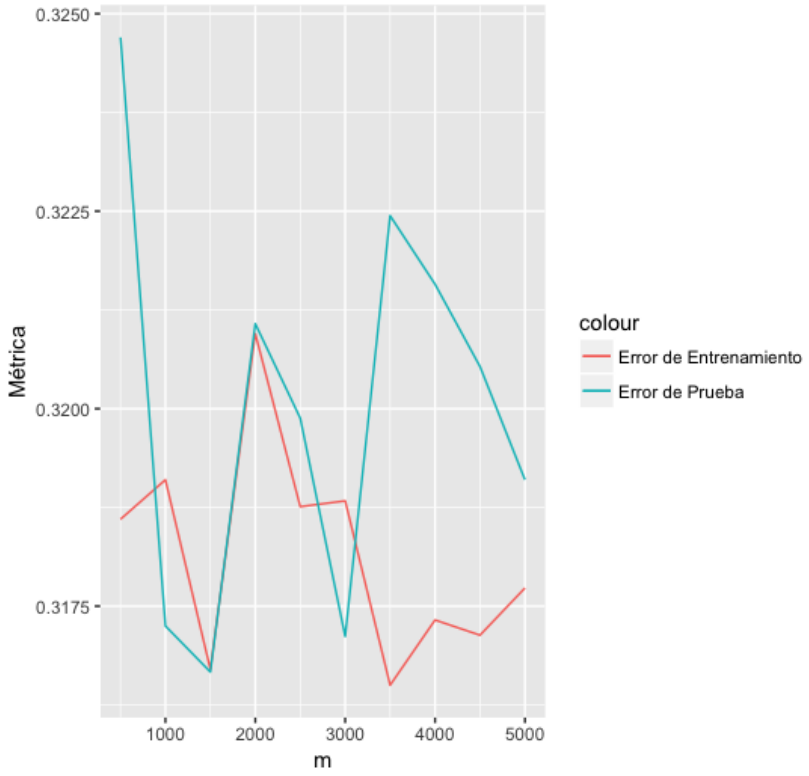


Figura 15: Curva de Aprendizaje para Regresión Logística con Características Polinomiales de Segundo Grado

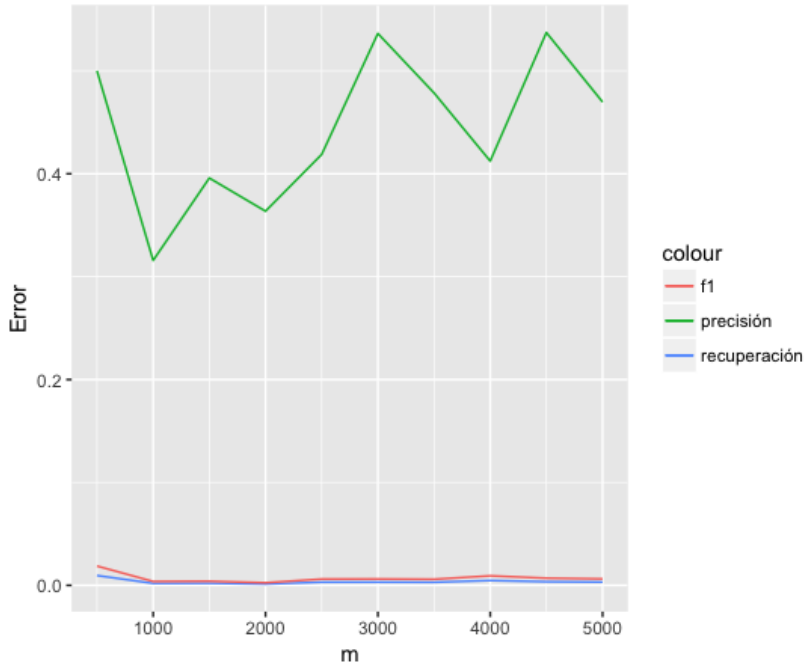


Figura 16: Calidad de Exactitud en Curva de Aprendizaje para Regresión Logística con Características Polinomiales de Segundo Grado

ción: aproximadamente la mitad de los casos que identifica como reprobados son efectivamente estudiantes reprobados, pero la proporción que representa del total de reprobados reales es demasiado baja, no alcanza siquiera el 0.005.

Los resultados obtenidos para ambos modelos nos permiten elegir entre estos el primero, pues aunque muestran prácticamente la misma tasa de error, el primero tiene mejores resultados para las métricas de precisión, recuperación y F_1, y según el principio de la *Navaja de Ockham* es preferible una hipótesis más simple que una compleja cuando ambas permiten explicar un mismo fenómeno.

Se procedió a ajustar el modelo sin características polinomiales utilizando el conjunto de datos reservado para tal fin. Se entrenó un modelo para cada semana en el rango [3..12]. La exactitud alcanzada por cada uno de estos modelos se muestra en la Figura 17, y como puede verse en todas el error supera el 30 %. La calidad de la exactitud permanece muy baja, no supera el 0.007 (ver Figura 18). Los casos de reprobación en el conjunto de prueba constituyen el 32.32 % del total, los modelos

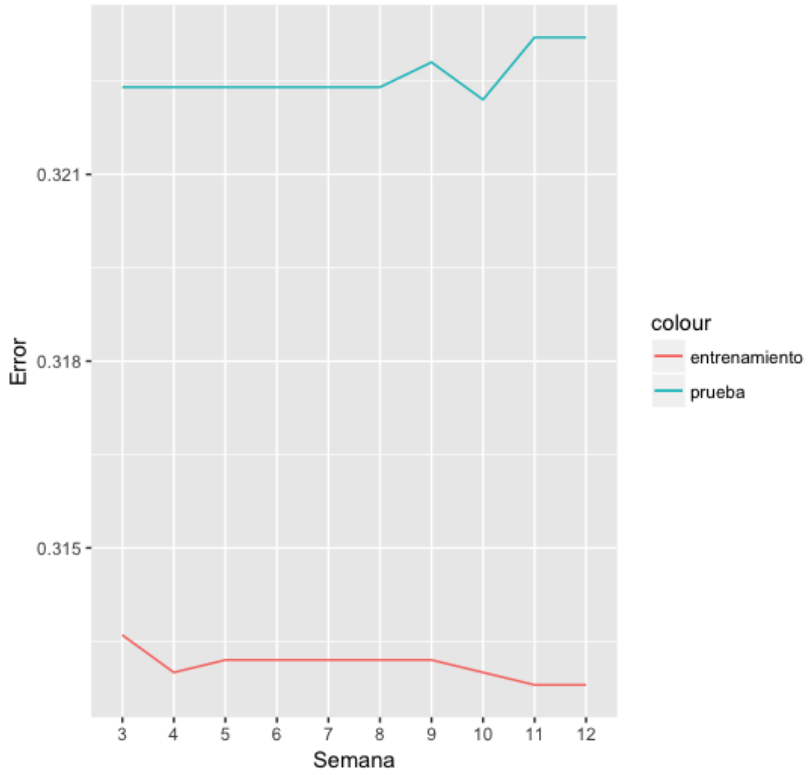


Figura 17: Exactitud con Regresión Logística para modelos de cada semana

ajustados predicen que el 99.88 % de los casos aprobarán, detectan únicamente un 0.37 % de los casos de reprobación, y clasifican erróneamente el 99.63 % restante.

3.2.2 *Máquinas de Soporte Vectorial*

Al aplicar la curva de aprendizaje (ver Listado 4.7 en Apéndice A en Apéndice A) para el algoritmo de Máquinas de Soporte Vectorial sobre el conjunto de datos sin características polinomiales (en el rango de 500 a 5000 muestras), se obtuvo que el error ronda el 41 % (ver Figura 19), y por tanto el algoritmo sufre de alto bias. Las métricas de precisión, recuperación y F_1 (ver Figura 20) indican que el modelo logra identificar casi un 30 % de los reprobados, pero de todos los identificados como reprobados sólo el 35 % son realmente reprobados.

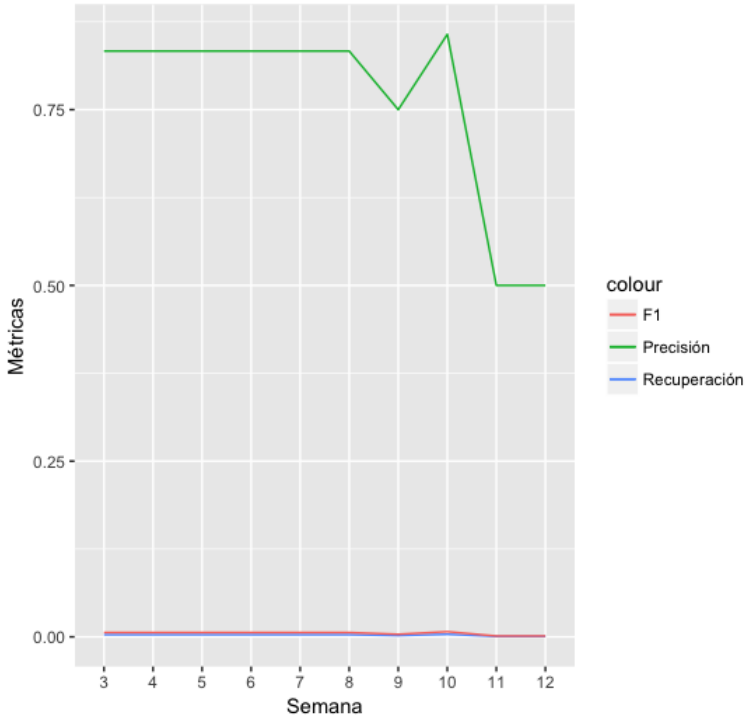


Figura 18: Calidad de la exactitud con Regresión Logística para modelos de cada semana

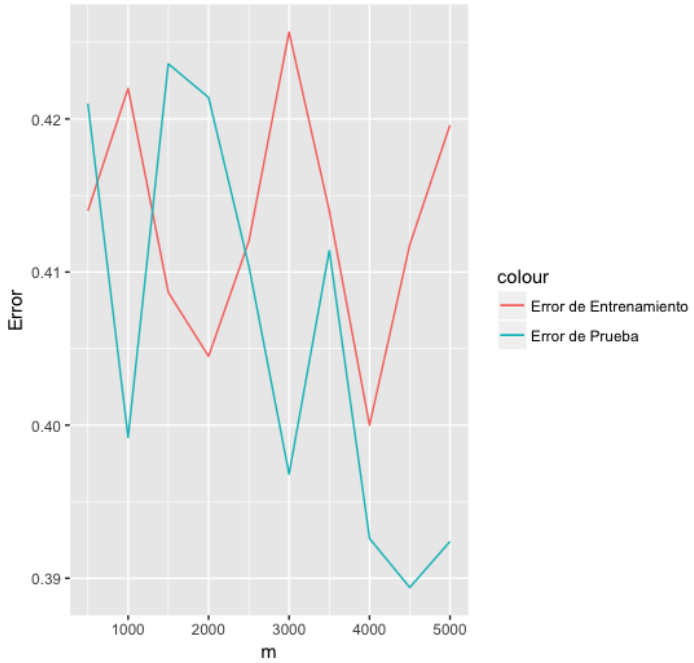


Figura 19: Curva de Aprendizaje para Máquinas de Soporte Vectorial

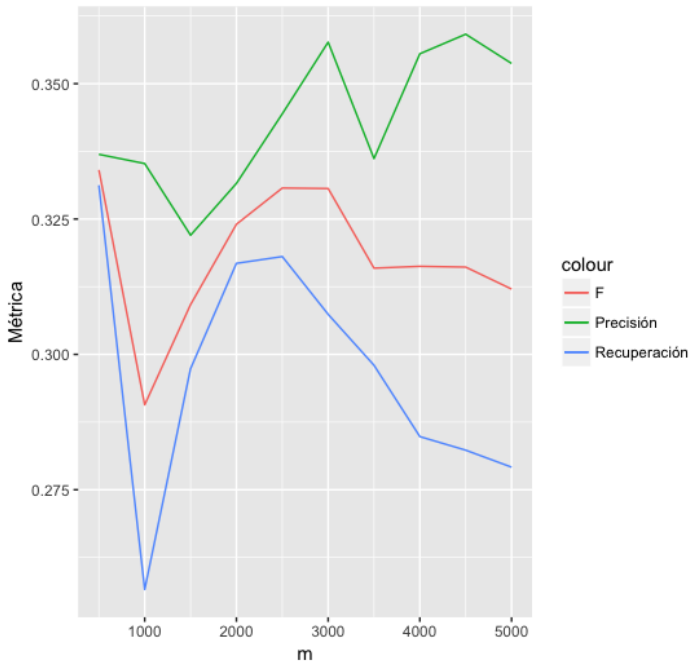


Figura 20: Calidad de Exactitud en Curva de Aprendizaje para Máquinas de Soporte Vectorial

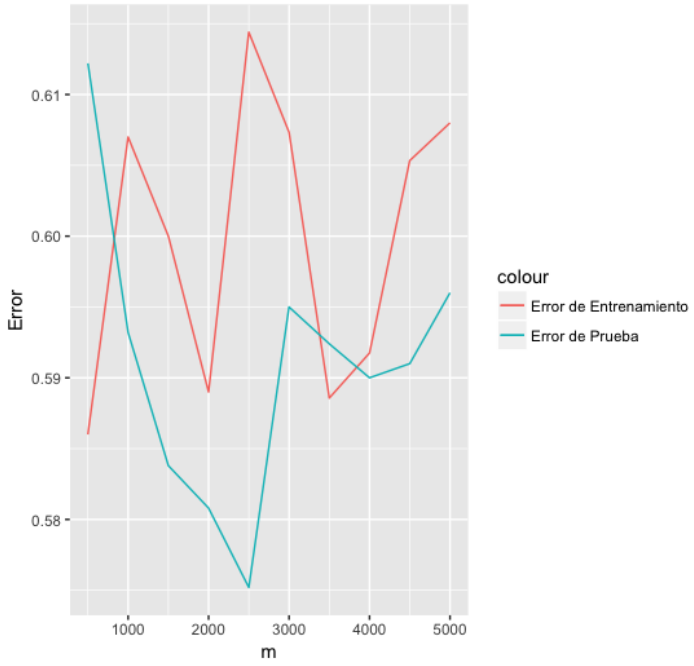


Figura 21: Curva de Aprendizaje para Máquinas de Soporte Vectorial con Características Polinomiales de Segundo Grado

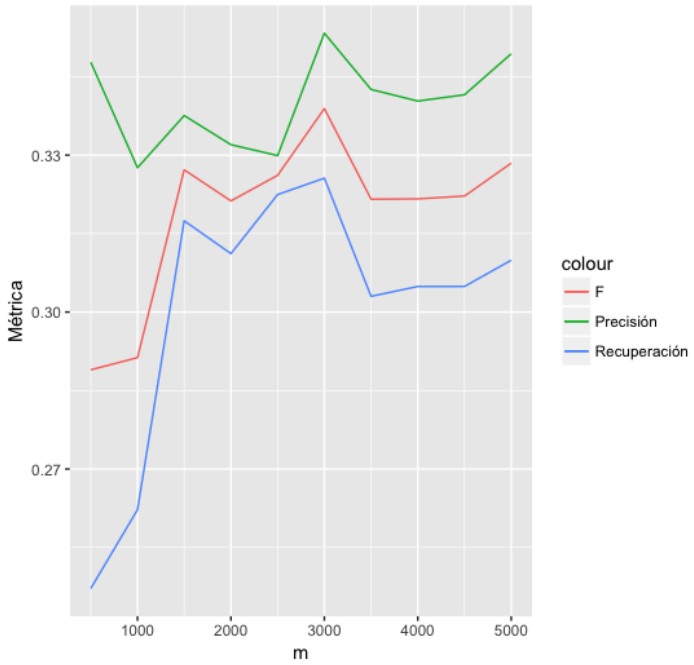


Figura 22: Calidad de Exactitud en Curva de Aprendizaje para Máquinas de Soporte Vectorial con Características Polinomiales de Segundo Grado

Se agregaron características polinomiales de segundo grado al modelo, para ver si se logra reducir el error derivado del alto bias. El diagnóstico con la curva de aprendizaje indica que el modelo no logra superar el problema de alto bias, pues el error ronda el 59 % (ver Figura 21). Las métricas de precisión, recuperación y F_1 se mantienen prácticamente iguales que en el modelo sin características polinomiales (ver Figura 22).

El modelo más simple fue seleccionado utilizando los mismos criterios que se emplearon con el algoritmo de Regresión Logística. Se procedió a ajustar el modelo para cada semana en el rango [3..12] (ver Listado 4.8 en Apéndice A). La exactitud alcanzada por cada uno de estos modelos se muestra en la Figura 23, y como puede verse en todas el error es de aproximadamente un 32 %. La calidad de la exactitud permanece bastante baja y es aproximadamente 0.065 (ver Figura 24). Los casos de reprobación en el conjunto de prueba constituyen el 31.87 % del total (956/3000), y los modelos ajustados predicen, en promedio, que el 97.27 % de los

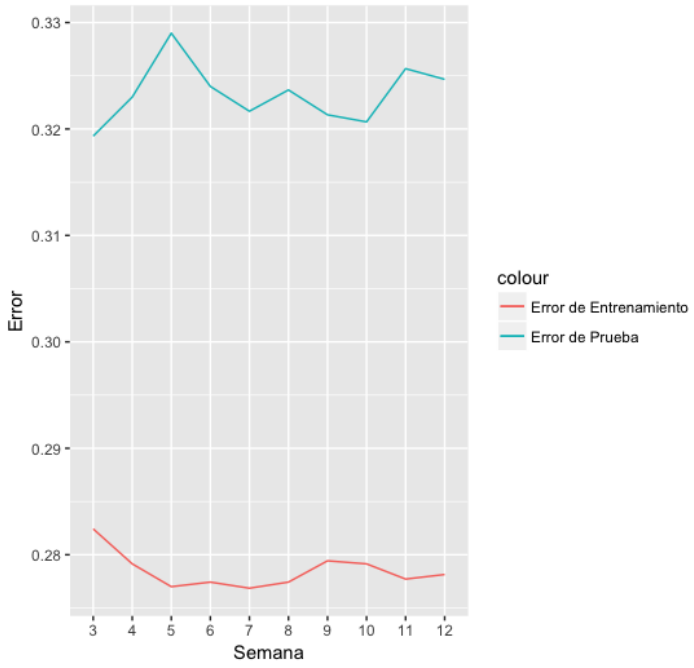


Figura 23: Exactitud con Máquinas de Soporte Vectorial para modelos de cada semana

casos aprobarán (2918/3000), detectan únicamente un 3.35 % de los casos de reprobación (32/956) con una precisión de 38.55 %, y clasifican erróneamente el 96.65 % restante de los casos de reprobación. La sensibilidad o tasa de recuperación del modelo es de 3.35 %.

3.2.3 *Redes Neuronales*

Para definir la arquitectura de los modelos que mejor se ajustan a los datos, se realizó una búsqueda en cuadrícula (producto cartesiano) para cada semana en el rango [3..12] (ver Listado 4.9 en Apéndice A), de los siguientes parámetros y nivel de complejidad:

- Dropout: Parámetro de regularización para prevenir sobreajuste,
  rango de valores [0.075, 0.05, 0.10, 0.15]

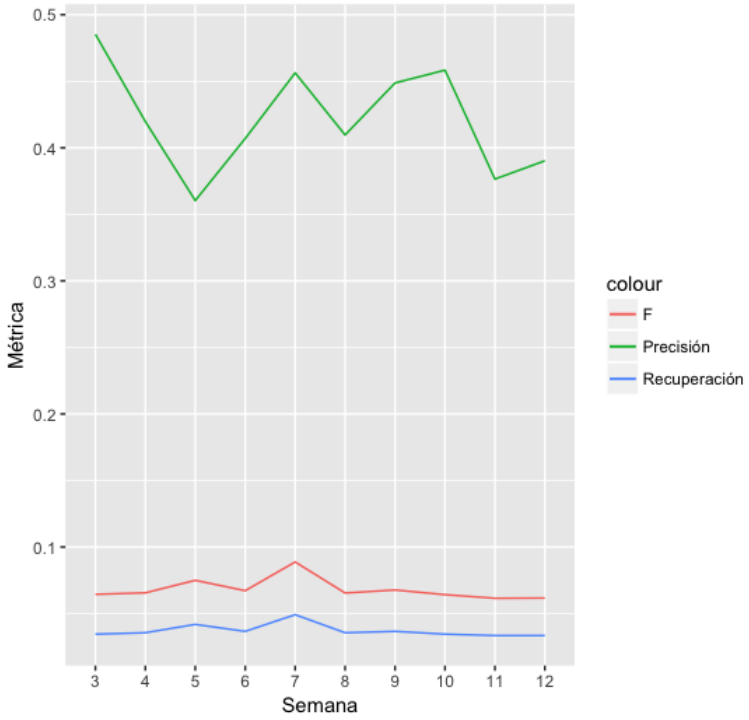


Figura 24: Calidad de la exactitud con SVM para modelos de cada semana

- Niveles Ocultos: Cantidad de niveles ocultos de la red neuronal,
  rango de valores [5, 9, 12, 15]
- Neuronas: Cantidad de neuronas en cada nivel oculto,
  rango de valores [250, 350, 450, 550]
- Epocas: Cantidad de épocas de entrenamiento de la red neuronal,
  rango de valores [500, 600, 700, 800, 900, ..., 2400, 2500]
- Tamaño de Batch: Cantidad de muestras usadas para actualizar el gradiente,
  rango de valores [64, 128, 256, 512]

Se utilizó la función de loss *Categorical CrossEntropy* y el método de optimización estocástica *Adam*, apropiado para problemas con muchos datos y parámetros [41].

Los parámetros de los modelos con el mejor resultado para cada semana se resumen en el Cuadro 5. La exactitud de los modelos se incrementa conforme avanza

Cuadro 5: Búsqueda de Parámetros con Redes Neuronales

Se *	Do *	Ni *	Ne *	Ep *	Bs *	Exactitud
3	0.150	12	550	800	256	0.682105964678
4	0.050	5	350	2500	128	0.683105631456
5	0.075	15	550	2500	128	0.681984014373
6	0.075	12	350	2000	512	0.693435521493
7	0.075	12	350	2200	512	0.750416527824
8	0.075	5	450	2500	256	0.759413528824
9	0.075	5	350	2200	256	0.758747084305
10	0.050	5	250	1600	128	0.752082639120
11	0.150	9	450	2500	512	0.763745418194
12	0.150	12	550	1600	512	0.782739086971

* Se=Semana, Do=Dropout, Ni=Niveles Ocultos, Ne=Neuronas, Ep=Epocas, Bs=Tamaño de Batch

el número de semana, lo mismo sucede con la calidad de la exactitud (ver columna F_1-Score en el Cuadro 6). En las primeras semanas la precisión es relativamente buena, pero la tasa de recuperación es muy baja, a partir de la semana 6 la tasa de recuperación supera el 50 %, con una precisión mayor al 60 %.

Se procedió a ajustar los modelos con los parámetros y la complejidad seleccionados, utilizando el conjunto de datos reservado para tal fin. Se entrenó un modelo para cada semana en el rango [3..12]. Los resultados obtenidos se resumen en el Cuadro 7 y la calidad de la exactitud de cada modelo se muestra en el Cuadro 8. La exactitud alcanzada por los modelos semanales varía entre el 68 % y el 78 %, con un F_1-Score menor al 25 % para los primeros cuatro modelos (tasa de recuperación menor al 15 % con precisión mayor al 65 %) y superior al 57 % para los últimos seis modelos (tasa de recuperación mayor al 52 % con precisión mayor al 62 %).

Cuadro 6: Calidad de la Exactitud en la Búsqueda de Parámetros

Semana	F_1**-Score**	**Precisión**	**Recuperación**
3	0.014507772021	0.8750000000	0.0073145245
4	0.016546018614	0.8000000000	0.0083594566
5	0.022657054582	0.7857142857	0.0114942528
6	0.235880398671	0.5748987854	0.1483803553
7	0.573705179283	0.6300000000	0.5266457680
8	0.604571428571	0.6670870114	0.5527690700
9	0.589569160998	0.6443618340	0.5433646813
10	0.593993325918	0.6349583829	0.5579937304
11	0.627038400842	0.6313559322	0.6227795193
12	0.622685185185	0.6977950713	0.5621734587

Cuadro 7: Ajuste de modelos con Redes Neuronales

Se *	**Do ***	**Ni ***	**Ne ***	**Ep ***	**Bs ***	**Exactitud**
3	0.150	12	550	800	256	0.683105631456
4	0.050	5	350	2500	128	0.684438520493
5	0.075	15	550	2500	128	0.681106297901
6	0.075	12	350	2000	512	0.703765411529
7	0.075	12	350	2200	512	0.749750083306
8	0.075	5	450	2500	256	0.769410196601
9	0.075	5	350	2200	256	0.761412862379
10	0.050	5	250	1600	128	0.756747750750
11	0.150	9	450	2500	512	0.782072642453
12	0.150	12	550	1600	512	0.776741086305

* Se=Semana, Do=Dropout, Ni=Niveles Ocultos, Ne=Neuronas, Ep=Epocas, Bs=Tamaño de Batch

Cuadro 8: Calidad de la Exactitud en Modelos con Redes Neuronales

Semana	F_1**-Score**	**Precisión**	**Recuperación**
3	0.012422360248	0.666666666692	0.006269592516
4	0.022703818370	0.916666666667	0.011494252874
5	0.025618374558	0.906250000000	0.012992831541
6	0.240819812126	0.658878504673	0.147335423197
7	0.574504249292	0.627475247525	0.529780564263
8	0.594837261504	0.642424242449	0.553814002132
9	0.601336302895	0.643623361144	0.564263322884
10	0.587583148559	0.625737898542	0.553814002092
11	0.645720476706	0.670416197975	0.622779519331
12	0.630650496141	0.667444574096	0.597701149425

3.2.4 *Selección de Algoritmo*

Para realizar la selección de Algoritmo se recolectaron los resultados de los experimentos previos en el Cuadro 9. Como puede observarse, los modelos ajustados con el Algoritmo de Redes Neuronales superan a los modelos correspondientes a los Algoritmos de Regresión Logística y Máquinas de Soporte Vectorial. Sin embargo, el mejor de los modelos ajustados presenta una exactitud del 78.2 % con un F_1-Score del 64.57 %, lo cual significa que tiene una tasa de recuperación del 62.28 % con una precisión del 67 %.

3.3 MODELO PREDICTIVO: AJUSTE Y VALIDACIÓN

Después de seleccionar el algoritmo de Redes Neuronales y también los parámetros y niveles de complejidad de los modelos para cada semana en el rango [3..12], se procedió a ajustar estos modelos para su validación. Se utilizó los subconjuntos de datos reservados para ajustar y para probar tanto la exactitud en la predicción

Cuadro 9: Comparación de Exactitud y $F_1 - Score$ para los tres algoritmos

#Se *	**LR ***		**SVM ***		**NN ***	
	Exact	F_1	**Exact**	F_1	**Exact**	F_1
3	0.677600	0.006165	0.680667	0.064453	0.683105	0.012422
4	0.677600	0.006165	0.677000	0.065573	0.684438	0.022704
5	0.677600	0.006165	0.671000	0.074976	0.681106	0.025618
6	0.677600	0.006165	0.676000	0.067178	0.703765	0.240820
7	0.677600	0.006165	0.678333	0.088763	0.749750	0.574504
8	0.677600	0.006165	0.676333	0.065447	0.769410	0.594837
9	0.677200	0.003704	0.678667	0.067698	0.761412	0.601336
10	0.677800	0.007394	0.679333	0.064202	0.756747	0.587583
11	0.676800	0.001236	0.674333	0.061479	0.782072	0.645720
12	0.676800	0.001236	0.675333	0.061657	0.776741	0.630650

* Se=Semana, LR=Regresión Logística, SVM=Máquinas de Soporte Vectorial, NN=Redes Neuronales

como la calidad de dicha exactitud. Los resultados obtenidos se pueden ver en los Cuadros 10 y 11. El mejor de los modelos ajustados presenta una exactitud del 75.84 % con un F_1-Score del 74.49 %, lo cual significa que tiene una tasa de recuperación del 64.69 % con una precisión del 87.78 %.

Cuadro 10: Ajuste de modelos Seleccionados con Redes Neuronales

Se *	Do *	Ni *	Ne *	Ep *	Bs *	Exactitud
3	0.150	12	550	800	256	0.683438853715
4	0.050	5	350	2500	128	0.682772409197
5	0.075	15	550	2500	128	0.681772742419
6	0.075	12	350	2000	512	0.695434855048
7	0.075	12	350	2200	512	0.747417527491
8	0.075	5	450	2500	256	0.771439520163
9	0.075	5	350	2200	256	0.758413862046
10	0.050	5	250	1600	128	0.754748417194
11	0.150	9	450	2500	512	0.762745751416
12	0.150	12	550	1600	512	0.783405531494

* Se=Semana, Do=Dropout, Ni=Niveles Ocultos, Ne=Neuronas, Ep=Epocas, Bs=Tamaño de Batch

Cuadro 11: Calidad de la Exactitud en Modelos Seleccionados con Redes Neuronales

Semana	F_1-Score	Precisión	Recuperación
3	0.022633744856	0.733333333333	0.011494252874
4	0.015117830146	1.000000000000	0.007616487455
5	0.021248339973	0.888888888889	0.010752688172
6	0.241322314052	0.577075098814	0.152560083595
7	0.579643473267	0.644501278772	0.526645768025
8	0.562060889935	0.639147802929	0.501567398119
9	0.744905855043	0.877811550152	0.646953405018
10	0.575575575576	0.552353506244	0.600835945664
11	0.632786885246	0.663230240557	0.605015673981
12	0.645948945616	0.688757396452	0.608150470219

4 CONCLUSIONES

Tomando como conjunto de entrada únicamente las interacciones de los estudiantes en la plataforma del TEC Digital, según la categorización configurada actualmente, no se encontró los parámetros ni los niveles de complejidad que permitan ajustar modelos semanales que predigan con una exactitud superior al 80 % y con una calidad de la exactitud superior al 80 %, los estudiantes que se encuentran en una situación que les conducirá a reprobar un curso.

Esto podría significar que las categorías de interacciones capturadas en el TEC Digital no tienen relación con el rendimiento académico de un estudiante en un curso. De acuerdo con el peso relativo de cada categoría de interacciones, las categorías que presentan cantidades significativas de interacciones son Almacenamiento de Documentos (48.80 %), Navegación en la Plataforma (38.70 %) y Datos de Evaluaciones (7.99 %) (ver Cuadro 1). Probablemente la mayoría de los estudiantes tienen un comportamiento similar en cuanto a recuperar o almacenar documentos, navegar en la plataforma y consultar sus resultados de evaluación, y esto sería lo que expresan las interacciones registradas por el módulo TAM de la plataforma. Además, la plataforma del TEC Digital es utilizada mayoritariamente en cursos de modalidad presencial, en los que casi toda actividad de aprendizaje sucede fuera de la plataforma.

El algoritmo de Redes Neuronales da mejores resultados que Regresión Logística y Máquinas de Soporte Vectorial, para los modelos planteados a partir de las interacciones con la plataforma del TEC Digital. Además, es mucho más flexible y

configurable, y cuenta con más *frameworks* que permiten aprovechar mejor el tiempo y los recursos destinados a ajustar y validar los modelos predictivos.

4.1 TRABAJOS FUTUROS

Como trabajo futuro queda el diseño de nuevos modelos predictivos que consideren otro tipo de variables en vez de las interacciones. Se sugiere los siguientes:

- Variables de tipo demográfico
- Promedio ponderado o CGPA
- Evaluación interna (semanal, requerida)
- Cantidad de créditos matriculados

La evaluación interna es requerida para que el modelo pueda predecir de manera progresiva conforme transcurren las semanas. Es necesario que el registro del avance de notas en la plataforma sea realizado semanalmente de manera oportuna, y podría consistir en la nota porcentual acumulada y el porcentaje del total que ha sido evaluado al finalizar cada semana. Esto requiere que todos los instructores planifiquen antes de iniciar cada curso la estructura de la evaluación y la registren en la plataforma, y vayan actualizando cada semana el avance de los rubros evaluados.

Si se quiere retomar el registro de las interacciones en la plataforma como variables para un nuevo modelo predictivo de éxito académico, será necesario replantear qué significa una interacción (por ejemplo, no es lo mismo navegar entre las etapas de una actividad interactiva de aprendizaje que navegar meramente entre las pestañas de la interfaz). No todos los *clicks* son iguales y por lo tanto deberían tener distinto peso. Habría que diseñar actividades interactivas a lo largo de todo el curso (y de todos los cursos) que permitan ir evaluando el conocimiento demostrado por los estudiantes. Las interacciones realizadas en estas actividades, podrían estar más relacionadas con el rendimiento académico final del estudiante.

Los resultados obtenidos con la implementación de los modelos predictivos servirían de insumo en un sistema de atención temprana para estudiantes en situación de riesgo académico. Esto requiere de la implementación de normativas y procesos que provean tanto a funcionarios del Departamento de Trabajo Social como a los académicos de las Escuelas de Carrera, con las herramientas necesarias para atender los casos señalados como prioritarios por el modelo predictivo.

Otro trabajo futuro que se puede realizar, como mejora adicional a cualquier modelo predictivo diseñado, es implementar la técnica de *ensemble* denominada *boosting*. Esta consiste en orquestar un conjunto de modelos y combinar las predicciones hechas por cada uno. Se generan K hipótesis utilizando conjuntos de datos con pesos, la primera hipótesis se ajusta con todos los pesos de todas las muestras en 1, los pesos son ajustados de modo que se incrementa los de las muestras mal clasificadas y se disminuye los de las muestras clasificadas correctamente. Se ajusta la segunda hipótesis con este nuevo conjunto de datos con nuevos pesos, y se repite el proceso hasta ajustar la hipótesis K-ésima. Las muestras son clasificadas de acuerdo con la predicción resultante de la mayoría de las K hipótesis.

4.2 REFLEXIONES FINALES

El modelo de datos, los algoritmos y modelos predictivos, y los resultados obtenidos en este trabajo, aplican únicamente para la población elegida y la plataforma educativa del TEC Digital (ver Sección 3.1). Para adaptar esta implementación sobre otra plataforma y otro conjunto de datos, será necesario rediseñar el conjunto de predictores a utilizar y la fase de preprocesamiento de los datos. Los resultados que se obtenga dependerán de la relación existente entre los datos seleccionados y la variable que se quiere predecir, así como del nivel de éxito en el entrenamiento de los algoritmos.

BIBLIOGRAFÍA

[1] A. M. Turing, "Intelligent Machinery, A Heretical Theory," *Philos. Math.*, vol. 4, no. 3, pp. 256–260, 1951. [En linea]. Disponible en: http://www.turingarchive.org/browse.php/B/4%5Cnhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6282777

[2] C. Garita and M. Chacón-Rivas, "TEC Digital: A case study of an e-learning environment for higher education in Costa Rica," in *Int. Conf. Inf. Technol. Based High. Educ. Training, ITHET*, no. June, 2012, pp. 1–6. [En linea]. Disponible en: https://www.researchgate.net/publication/259332961_TEC_Digital_A_case_study_of_an_e-learning_environment_for_higher_education_in_Costa_Rica

[3] J. Espinoza and M. Chacón-Rivas, "TEC Digital: Una iniciativa de implementación de e-learning en Costa Rica," in *CIESC XXXVI Conf. Latinoam. Informática*, no. October, 2010, pp. 1 – 18. [En linea]. Disponible en: http://www.tec-digital.itcr.ac.cr/servicios/investigacion/sites/default/files/publicaciones/J.Espinoza_M.Chacon_CLEI_2010.pdf

[4] Instituto Tecnológico de Costa Rica, *Plan Estratégico 2011-2015*, 2012. [En linea]. Disponible en: http://www.tec.ahttp//www.tec.ac.cr/eltec/rectoria/opi/Documents/PlanEstrat%C3%A9gico2011-2015.pdf

[5] TecDigital, "¿Quiénes somos?" 2012. [En linea]. Disponible en: http://tecdigital.tec.ac.cr/servicios/investigacion/?q=quienes_somos

[6] ——, "Why Costa Rica Institute of Technology chose .LRN," 2009. [En linea]. Disponible en: http://dotlrn.org/users/itcr

[7] DotLRN, "About .LRN," 2016. [En linea]. Disponible en: http://dotlrn.org/about/

[8] OpenACS, "The Toolkit for Online Communities," 2016. [En linea]. Disponible en: http://openacs.org/

[9] J. Couchet, O. C. Santos, E. Raffenne, J. G. Boticario, and D. Manrique, "A general tracking and auditing architecture for the OpenACS framework," in *6th OpenACS .LRN Conf. 2008. Int. Conf. Work. Community based Environ.*, 2008, pp. 179–187. [En linea]. Disponible en: http://ges.galileo.edu/fs/download/MemoriasConferenciaUniversidadGalileo?file_id=910694

[10] J. Couchet, O. C. Santos, E. Raffenne, and J. G. Boticario, "The Tracking and Auditing Module for the OpenACS Framework," in *7th OpenACS/.LRN Conf. Int. Conf. Work. Community based Environ.*, U. de Valencia, Ed. Valencia,

España: Universidad de Valencia, 2008, pp. 82–87. [En linea]. Disponible en: http://docplayer.es/866058-7th-openacs-lrn-spain-18th-19th-novembre-2008-conferencia-sobre-software-de-libre-en-educacion-superior-20-de-noviembre-2008.html

[11] L. Martin, D. R. Martínez, O. Revilla, M. J. Aguilar, O. C. Santos, and J. G. Boticario, "Usability in e-Learning Platforms: heuristics comparison between Moodle, Sakai and dotLRN," *Artif. Intell.*, vol. 509, no. Lll, pp. 75–84, 2003. [En linea]. Disponible en: http://ges.galileo.edu/conf2008/

[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, Inc., 2010.

[13] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to Statistical Learning with Applications in R*. New York: Springer, 2013.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer-Verlag New York, 2009. [En linea]. Disponible en: http://statweb.stanford.edu/$\sim$tibs/ElemStatLearn/

[15] M. Kearns, Y. Mansour, A. Ng, and D. Ron, "An experimental and theoretical comparison of model selection methods," *Mach. Learn.*, 1997. [En linea]. Disponible en: http://link.springer.com/article/10.1023/A:1007344726582

[16] A. Ng, "Machine Learning Course," 2017. [En linea]. Disponible en: https://www.coursera.org/learn/machine-learning

[17] J. M. Hilbe, *Logistic Regression Models*. New York: Chapman and Hall, 2009, vol. 53, no. 9.

[18] O. C. Santos, E. Raffenne, J. Granado, and J. G. Boticario, "Dynamic support in OpenACS/dotLRN: Technological infrastructure for providing dynamic recommendations for all in open and standard-based LMS," in *6th OpenACS .LRN Conf. 2008. Int. Conf. Work. Community based Environ.*, 2008, pp. 197–206. [En linea]. Disponible en: http://ges.galileo.edu/fs/download/MemoriasConferenciaUniversidadGalileo?file_id=910694

[19] J. G. Boticario and O. C. Santos, "An open IMS-based user modelling approach for developing adaptive learning management systems," *Interact. Media Educ.*, vol. 2007, no. 1, sep 2007. [En linea]. Disponible en: http://jime.open.ac.uk/articles/10.5334/2007-2/

[20] D. Detoni, R. M. Araujo, and C. Cechinel, "Predição de Reprovação de Alunos de Educação a Distância Utilizando Contagem de Interações," *An. do Simpósio Bras. Informática na Educ.*, vol. 25, no. 1, pp. 896–905, 2014. [En linea]. Disponible en: http://www.br-ie.org/pub/index.php/sbie/article/view/3026

[21] A. Hirumi, "A Framework for Analyzing, Designing, and Sequencing Planned Elearning Interactions." *Q. Rev. Distance Educ.*, vol. 3, no. 2, p. 141, 2002. [En linea]. Disponible en: http://proxy1.ncu.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=ehh&AN=7548956&site=eds-live

[22] ——, "The Design and Sequencing of e-Learning Interactions: A Grounded Approach," *Int. J. E-learning*, vol. 1, no. January-March, pp. 19–27, 2002. [En linea]. Disponible en: https://www.researchgate.net/publication/248580777_The_Design_and_Sequencing_of_eLearning_InteractionsA_Grounded_Approach

[23] R. L. Rodrigues, F. P. a. De Medeiros, and A. S. Gomes, "Modelo de Regressão Linear aplicado à previsão de desempenho de estudantes em ambiente de aprendizagem," *An. do Simpósio Bras. Informática na Educ.*, vol. XXIV, no. 1, pp. 607–616, 2013. [En linea]. Disponible en: http://www.br-ie.org/pub/index.php/sbie/article/view/2539

[24] R. Gotardo, P. R. M. Cereda, and E. R. H. Junior, "Predição do Desempenho do Aluno usando Sistemas de Recomendação e Acoplamento de Classificadores," *An. do Simpósio Bras. Informática na Educ.*, vol. XXIV, no. 1, pp. 657–666, 2013. [En linea]. Disponible en: http://www.br-ie.org/pub/index.php/sbie/article/view/2544

[25] A. M. Shahiri, W. Husain, and N. A. Rashid, "A Review on Predicting Student's Performance Using Data Mining Techniques," *Procedia Comput. Sci.*, vol. 72, pp. 414–422, 2015.

[26] B. Kitchenham, "Procedures for performing systematic reviews," Tech. Rep. TR/SE-0401, 2004. [En linea]. Disponible en: http://csnotes.upm.edu.my/kelasmaya/pgkm20910.nsf/0/715071a8011d4c2f482577a700386d3a/$FILE/10.1.1.122.3308[1].pdf

[27] M. Al-Razgan, A. S. Al-Khalifa, and H. S. Al-Khalifa, "Educational Data Mining: A Systematic Review of the Published Literature 2006-2013," in *Proc. First Int. Conf. Adv. Data Inf. Eng.*, T. Herawan, M. M. Deris, and J. Abawajy, Eds. Singapore: Springer Singapore, 2014, ch. 80, pp. 711–719. [En linea]. Disponible en: http://link.springer.com/10.1007/978-981-4585-18-7_80

[28] C. Romero and S. Ventura, "Educational data mining: A survey from 1995 to 2005," *Expert Syst. Appl.*, vol. 33, no. 1, pp. 135–146, jul 2007. [En linea]. Disponible en: http://linkinghub.elsevier.com/retrieve/pii/S0957417406001266

[29] E. A. Almosallam and H. C. Ouertani, "Learning Analytics: definitions, applications and related fields," in *Proc. First Int. Conf. Adv. Data Inf. Eng.*, T. Herawan, M. M. Deris, and J. Abawajy, Eds. Singapore: Springer Singapore, 2014, ch. 81, pp. 721–730. [En linea]. Disponible en: http://link.springer.com/10.1007/978-981-4585-18-7_81

[30] C. Romero and S. Ventura, "Educational Data Mining: A Review of the State of the Art," *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 40, no. 6, pp. 601–618, nov 2010. [En linea]. Disponible en: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5524021

[31] ——, "Data mining in education," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 3, no. 1, pp. 12–27, jan 2013. [En linea]. Disponible en: http://doi.wiley.com/10.1002/widm.1075

[32] CeNAT, "Cluster Computacional Kabré," 2017. [En linea]. Disponible en: http://cluster.cenat.ac.cr/wordpress/guia/

[33] J. Friedman, T. Hastie, N. Simon, and R. Tibshirani, "glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models," 2017. [En linea]. Disponible en: https://cran.r-project.org/web/packages/glmnet/

[34] C.-c. Chang and C.-j. Lin, "LIBSVM : A Library for Support Vector Machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 1–39, 2013.

[35] Multimedia Knowledge and Social Analytics Lab, "GPU-Accelerated LIBSVM," 2017. [En linea]. Disponible en: http://mklab.iti.gr/project/GPU-LIBSVM

[36] Google, "Keras: Deep Learning library for Theano and TensorFlow," 2017. [En linea]. Disponible en: https://keras.io/

[37] ——, "TensorFlow: An open-source software library for Machine Intelligence," 2017. [En linea]. Disponible en: https://www.tensorflow.org/

[38] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[39] G. Forman and M. Scholz, "Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement," *HP labs*, vol. 12, no. 1, pp. 49–57, 2009.

[40] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation," *Adv. Artif. Intell.*, vol. 4304, pp. 1015–1021, 2006. [En linea]. Disponible en: http://link.springer.com/10.1007/11941439_114

[41] D. P. Kingma and J. Lei-Ba, "Adam: A Method for Stochastic Optimization," *ICLR*, vol. 8, p. 15, 2015. [En linea]. Disponible en: https://arxiv.org/pdf/1412.6980v8.pdf

APÉNDICES

Apéndice A: Programas

```
#!/usr/bin/env Rscript
setwd("~/torque-scripts")

pnormalize <- function(X) {
  k = NCOL(X)
  cl <- makeCluster(256)
  xmin <- foreach (z = 1:k, .combine='cbind') %dopar% {
    min(X[,z])
  }
  xmax <- foreach (z = 1:k, .combine='cbind') %dopar% {
    max(X[,z])
  }
  Xright <- foreach (z = 1:k, .combine='cbind') %dopar% {
    (X[,z] - xmin[z]) / (xmax[z] - xmin[z])
  }
  stopCluster(cl)

  Xright[is.na(Xright)] = 0
  Xright[is.infinite(Xright)] = 0
  Xright = matrix(c(Xright), ncol=NCOL(X), nrow=NROW(X), byrow=FALSE)
  xmin = matrix(c(xmin), ncol=NCOL(X), nrow=1, byrow=TRUE)
  xmax = matrix(c(xmax), ncol=NCOL(X), nrow=1, byrow=TRUE)
  saveRDS(xmin, "data/datasets/Xmin.Rds")
  saveRDS(xmax, "data/datasets/Xmax.Rds")
  return(Xright)
}

data <- read.csv("clean_interactions_by_category_week.csv",  header = TRUE,
  stringsAsFactors=FALSE)
data$paso_curso = as.factor(1 - data$paso_curso)

set.seed(1010)
rsamples = sample(nrow(data))
X_data = data[, 1:289]
y_data = data[, 290]

X_data = as.data.frame(pnormalize(X_data))

X1 = X_data[rsamples[1:10000],]     # 12.5% de los datos
y1 = y_data[rsamples[1:10000]]
saveRDS(X1, sprintf("data/datasets/X1.Rds"))
saveRDS(y1, sprintf("data/datasets/y1.Rds"))

X2 = X_data[rsamples[10001:20000],] # 12.5% de los datos
y2 = y_data[rsamples[10001:20000]]
saveRDS(X2, sprintf("data/datasets/X2.Rds"))
saveRDS(y2, sprintf("data/datasets/y2.Rds"))

X3 = X_data[rsamples[20001:60000],] # 45.0% de los datos
y3 = y_data[rsamples[20001:60000]]
saveRDS(X3, sprintf("data/datasets/X3.Rds"))
saveRDS(y3, sprintf("data/datasets/y3.Rds"))

X4 = X_data[rsamples[60001:84803],] # 30.0% de los datos
y4 = y_data[rsamples[60001:84803]]
saveRDS(X4, sprintf("data/datasets/X4.Rds"))
saveRDS(y4, sprintf("data/datasets/y4.Rds"))
```

Listado 4.1: Normalización de los datos

```
#!/usr/bin/env Rscript
setwd("~/torque-scripts")
source('0_util.R')
require(doMC)
registerDoMC(cores = 1536)

Xds = readRDS("data/datasets/X1.Rds")
yds = readRDS("data/datasets/y1.Rds")
idx = 0
set.seed(2017)

for (i in 1:10) {
  idx = idx + 500

  for (c in 1:20) {
    samples = sample.int(NROW(Xds), idx*2, replace = FALSE)
    tsamples = samples[1:idx]
    vsamples = samples[(idx+1):(idx*2)]

    X = Xds[tsamples,]
    y = yds[tsamples]

    fit = cv.glmnet(data.matrix(X), y, family = "binomial",
      type.measure = "auc", parallel = TRUE, standardize = FALSE)

    trainError <- fit$cvm[which(fit$lambda == fit$lambda.min)]

    X = Xds[vsamples,]
    y = yds[vsamples]

    fitted.results <- ifelse(predict.cv.glmnet(fit,
      newx = data.matrix(X), s = c("lambda.min")) > 0.5, 1, 0)

    testError <- mean(fitted.results != as.data.frame(y))

    if (c == 1) {
      r = rbind(c(trainError, testError, idx, TP(y, fitted.results),
        FP(y, fitted.results), FN(y, fitted.results)))
    }
    else {
      r = rbind(r, c(trainError, testError, idx, TP(y, fitted.results),
        FP(y, fitted.results), FN(y, fitted.results)))
    }
  }

  tp = mean(r[,4])
  fp = mean(r[,5])
  fn = mean(r[,6])
  real = tp + fn
  pred = tp + fp
  prec = ifelse(pred == 0, 0, tp / pred)
  recall = ifelse(real == 0, 0, tp / real)
  f1score = ifelse(prec+recall == 0, 0, 2*prec*recall/(prec+recall))

  prn("(%f, %f, %d)", mean(r[,1]), mean(r[,2]), mean(r[,3]))
  prn("(T+=%f, F+=%f, F-=%f, Real(0)=%f, Predicted(0)=%f, Precision=%f,
    Recall=%f, F1Score=%f)", tp, fp, fn, real, pred, prec, recall, f1score)
}
```

Listado 4.2: Curva de Aprendizaje con Regresión Logística

```
#!/usr/bin/env Rscript

setwd('~/R')
#setwd('~/torque-scripts')

source('0_poly.R')
source('0_util.R')
source('exec.R')

# xi * xj for every category i, j within every week
addPolynomialFeatures <- function(X, p) {
  k = NROW(p)

  for (z in 1:k) {
    X = cbind(X, exec(X, z, p))
  }

  return (X)
}

run <- function() {

  p = list(a=0, b=poly2, c=poly3, d=poly4, e=poly5)

  ## polynomial degree features
  for (idx in names(p)[2]) {
    poly = asc(idx) - asc('a') + 1

    X1 = readRDS('data/datasets/X1.Rds')
    X1 = addPolynomialFeatures(X1, p[[idx]])
    saveRDS(X1, 'data/datasets/X1p.Rds')

    rm(X1)
  }
}

run()
```

Listado 4.3: Agregar características polinomiales

```
exec <- function(X, z, p) {
    cbind(
        c(X[,1]^p[z,1]*X[,2]^p[z,2]*X[,3]^p[z,3]*X[,4]^p[z,4]*X[,5]^p[z,5]*
          X[,6]^p[z,6]*X[,7]^p[z,7]*X[,8]^p[z,8]*X[,9]^p[z,9]*X[,10]^p[z,10]*
          X[,11]^p[z,11]*X[,12]^p[z,12]*X[,13]^p[z,13]*X[,14]^p[z,14]*
          X[,15]^p[z,15]*X[,16]^p[z,16]*X[,17]^p[z,17]),
        c(X[,18]^p[z,1]*X[,19]^p[z,2]*X[,20]^p[z,3]*X[,21]^p[z,4]*X[,22]^p[z,5]*X
    [,23]^p[z,6]*X[,24]^p[z,7]*X[,25]^p[z,8]*X[,26]^p[z,9]*X[,27]^p[z,10]*X[,28]^
    p[z,11]*X[,29]^p[z,12]*X[,30]^p[z,13]*X[,31]^p[z,14]*X[,32]^p[z,15]*X[,33]^p[
    z,16]*X[,34]^p[z,17]),
        c(X[,35]^p[z,1]*X[,36]^p[z,2]*X[,37]^p[z,3]*X[,38]^p[z,4]*X[,39]^p[z,5]*X
    [,40]^p[z,6]*X[,41]^p[z,7]*X[,42]^p[z,8]*X[,43]^p[z,9]*X[,44]^p[z,10]*X[,45]^
    p[z,11]*X[,46]^p[z,12]*X[,47]^p[z,13]*X[,48]^p[z,14]*X[,49]^p[z,15]*X[,50]^p[
    z,16]*X[,51]^p[z,17]),
        c(X[,52]^p[z,1]*X[,53]^p[z,2]*X[,54]^p[z,3]*X[,55]^p[z,4]*X[,56]^p[z,5]*X
    [,57]^p[z,6]*X[,58]^p[z,7]*X[,59]^p[z,8]*X[,60]^p[z,9]*X[,61]^p[z,10]*X[,62]^
    p[z,11]*X[,63]^p[z,12]*X[,64]^p[z,13]*X[,65]^p[z,14]*X[,66]^p[z,15]*X[,67]^p[
    z,16]*X[,68]^p[z,17]),
        c(X[,69]^p[z,1]*X[,70]^p[z,2]*X[,71]^p[z,3]*X[,72]^p[z,4]*X[,73]^p[z,5]*X
    [,74]^p[z,6]*X[,75]^p[z,7]*X[,76]^p[z,8]*X[,77]^p[z,9]*X[,78]^p[z,10]*X[,79]^
```

```
    p[z,11]*X[,80]^p[z,12]*X[,81]^p[z,13]*X[,82]^p[z,14]*X[,83]^p[z,15]*X[,84]^p[
    z,16]*X[,85]^p[z,17]),
        c(X[,86]^p[z,1]*X[,87]^p[z,2]*X[,88]^p[z,3]*X[,89]^p[z,4]*X[,90]^p[z,5]*X
    [,91]^p[z,6]*X[,92]^p[z,7]*X[,93]^p[z,8]*X[,94]^p[z,9]*X[,95]^p[z,10]*X[,96]^
    p[z,11]*X[,97]^p[z,12]*X[,98]^p[z,13]*X[,99]^p[z,14]*X[,100]^p[z,15]*X[,101]^
    p[z,16]*X[,102]^p[z,17]),
        c(X[,103]^p[z,1]*X[,104]^p[z,2]*X[,105]^p[z,3]*X[,106]^p[z,4]*X[,107]^p[z
    ,5]*X[,108]^p[z,6]*X[,109]^p[z,7]*X[,110]^p[z,8]*X[,111]^p[z,9]*X[,112]^p[z
    ,10]*X[,113]^p[z,11]*X[,114]^p[z,12]*X[,115]^p[z,13]*X[,116]^p[z,14]*X[,117]^
    p[z,15]*X[,118]^p[z,16]*X[,119]^p[z,17]),
        c(X[,120]^p[z,1]*X[,121]^p[z,2]*X[,122]^p[z,3]*X[,123]^p[z,4]*X[,124]^p[z
    ,5]*X[,125]^p[z,6]*X[,126]^p[z,7]*X[,127]^p[z,8]*X[,128]^p[z,9]*X[,129]^p[z
    ,10]*X[,130]^p[z,11]*X[,131]^p[z,12]*X[,132]^p[z,13]*X[,133]^p[z,14]*X[,134]^
    p[z,15]*X[,135]^p[z,16]*X[,136]^p[z,17]),
        c(X[,137]^p[z,1]*X[,138]^p[z,2]*X[,139]^p[z,3]*X[,140]^p[z,4]*X[,141]^p[z
    ,5]*X[,142]^p[z,6]*X[,143]^p[z,7]*X[,144]^p[z,8]*X[,145]^p[z,9]*X[,146]^p[z
    ,10]*X[,147]^p[z,11]*X[,148]^p[z,12]*X[,149]^p[z,13]*X[,150]^p[z,14]*X[,151]^
    p[z,15]*X[,152]^p[z,16]*X[,153]^p[z,17]),
        c(X[,154]^p[z,1]*X[,155]^p[z,2]*X[,156]^p[z,3]*X[,157]^p[z,4]*X[,158]^p[z
    ,5]*X[,159]^p[z,6]*X[,160]^p[z,7]*X[,161]^p[z,8]*X[,162]^p[z,9]*X[,163]^p[z
    ,10]*X[,164]^p[z,11]*X[,165]^p[z,12]*X[,166]^p[z,13]*X[,167]^p[z,14]*X[,168]^
    p[z,15]*X[,169]^p[z,16]*X[,170]^p[z,17]),
        c(X[,171]^p[z,1]*X[,172]^p[z,2]*X[,173]^p[z,3]*X[,174]^p[z,4]*X[,175]^p[z
    ,5]*X[,176]^p[z,6]*X[,177]^p[z,7]*X[,178]^p[z,8]*X[,179]^p[z,9]*X[,180]^p[z
    ,10]*X[,181]^p[z,11]*X[,182]^p[z,12]*X[,183]^p[z,13]*X[,184]^p[z,14]*X[,185]^
    p[z,15]*X[,186]^p[z,16]*X[,187]^p[z,17]),
        c(X[,188]^p[z,1]*X[,189]^p[z,2]*X[,190]^p[z,3]*X[,191]^p[z,4]*X[,192]^p[z
    ,5]*X[,193]^p[z,6]*X[,194]^p[z,7]*X[,195]^p[z,8]*X[,196]^p[z,9]*X[,197]^p[z
    ,10]*X[,198]^p[z,11]*X[,199]^p[z,12]*X[,200]^p[z,13]*X[,201]^p[z,14]*X[,202]^
    p[z,15]*X[,203]^p[z,16]*X[,204]^p[z,17]),
        c(X[,205]^p[z,1]*X[,206]^p[z,2]*X[,207]^p[z,3]*X[,208]^p[z,4]*X[,209]^p[z
    ,5]*X[,210]^p[z,6]*X[,211]^p[z,7]*X[,212]^p[z,8]*X[,213]^p[z,9]*X[,214]^p[z
    ,10]*X[,215]^p[z,11]*X[,216]^p[z,12]*X[,217]^p[z,13]*X[,218]^p[z,14]*X[,219]^
    p[z,15]*X[,220]^p[z,16]*X[,221]^p[z,17]),
        c(X[,222]^p[z,1]*X[,223]^p[z,2]*X[,224]^p[z,3]*X[,225]^p[z,4]*X[,226]^p[z
    ,5]*X[,227]^p[z,6]*X[,228]^p[z,7]*X[,229]^p[z,8]*X[,230]^p[z,9]*X[,231]^p[z
    ,10]*X[,232]^p[z,11]*X[,233]^p[z,12]*X[,234]^p[z,13]*X[,235]^p[z,14]*X[,236]^
    p[z,15]*X[,237]^p[z,16]*X[,238]^p[z,17]),
        c(X[,239]^p[z,1]*X[,240]^p[z,2]*X[,241]^p[z,3]*X[,242]^p[z,4]*X[,243]^p[z
    ,5]*X[,244]^p[z,6]*X[,245]^p[z,7]*X[,246]^p[z,8]*X[,247]^p[z,9]*X[,248]^p[z
    ,10]*X[,249]^p[z,11]*X[,250]^p[z,12]*X[,251]^p[z,13]*X[,252]^p[z,14]*X[,253]^
    p[z,15]*X[,254]^p[z,16]*X[,255]^p[z,17]),
        c(X[,256]^p[z,1]*X[,257]^p[z,2]*X[,258]^p[z,3]*X[,259]^p[z,4]*X[,260]^p[z
    ,5]*X[,261]^p[z,6]*X[,262]^p[z,7]*X[,263]^p[z,8]*X[,264]^p[z,9]*X[,265]^p[z
    ,10]*X[,266]^p[z,11]*X[,267]^p[z,12]*X[,268]^p[z,13]*X[,269]^p[z,14]*X[,270]^
    p[z,15]*X[,271]^p[z,16]*X[,272]^p[z,17]),
        c(X[,273]^p[z,1]*X[,274]^p[z,2]*X[,275]^p[z,3]*X[,276]^p[z,4]*X[,277]^p[z
    ,5]*X[,278]^p[z,6]*X[,279]^p[z,7]*X[,280]^p[z,8]*X[,281]^p[z,9]*X[,282]^p[z
    ,10]*X[,283]^p[z,11]*X[,284]^p[z,12]*X[,285]^p[z,13]*X[,286]^p[z,14]*X[,287]^
    p[z,15]*X[,288]^p[z,16]*X[,289]^p[z,17])
    )
}
```

Listado 4.4: exec.R - Subrutina para agregar características polinomiales a una semana con 17 categorías

```
#!/usr/bin/env Rscript
options(echo=FALSE)
suppressPackageStartupMessages(library(glmnet))
setwd("~/torque-scripts/glmnet")

## functions
source('0_util.R')

##supply default values
ds_size = '1k'
i = 2
num_cores = 255

## THIS IS TO PARALLELIZE
args = (commandArgs(TRUE))
if (length(args) > 0) {
  for (a in 1:length(args)) {
    eval(parse(text = args[[a]]))
  }
}

## parallel
require(doMC)
registerDoMC(cores = num_cores)

for (i in 2:5) {
  ## load training dataset
  Xtrain = readRDS(sprintf("~/data/Xtrain_%s_poly%d.Rds", ds_size, i))
  ytrain = readRDS(sprintf("~/data/ytrain_%s.Rds", ds_size))
  X = Xtrain
  y = ytrain

  set.seed(1010)
  fit = cv.glmnet(data.matrix(X), y, family = "binomial", type.measure =
    "class", parallel = TRUE, standardize = FALSE, maxit=10^9)
  trainError <- fit$cvm[which(fit$lambda == fit$lambda.min)]

  ## load test dataset
  Xtest = readRDS(sprintf("~/data/with_cat/Xtest_%s_poly%d.Rds", ds_size, i))
  ytest = readRDS(sprintf("~/data/with_cat/ytest_%s.Rds", ds_size))
  X = Xtest
  y = ytest

  fitted.results <- ifelse(predict.cv.glmnet(fit, newx = data.matrix(X),
    type = c("response"), s = c("lambda.min"), exact = FALSE) > 0.5, 1, 0)

  ## find out error and print accuracy
  testError <- mean(fitted.results != as.data.frame(y))

  prn("%f, %f, %d", trainError, testError, i)
}
```

Listado 4.5: Búsqueda de Complejidad y λ Optimos

```
#!/usr/bin/env Rscript

TP <- function(r, p) {
  sum(as.numeric(as.character(r[(as.numeric(as.character(r))) & (as.numeric(as.
  character(p)))])))
}

TN <- function(r, p) {
  NROW(as.numeric(as.character(p[(1 - as.numeric(as.character(r))) & (1 - as.
  numeric(as.character(p)))])))
}

FP <- function(r, p) {
  sum(as.numeric(as.character(p[(1 - as.numeric(as.character(r))) & (as.numeric
  (as.character(p)))])))
}

FN <- function(r, p) {
  sum(as.numeric(as.character(r[(as.numeric(as.character(r))) & (1 - as.numeric
  (as.character(p)))])))
}

RP <- function(r) {
  sum(as.numeric(as.character(r)))
}

PP <- function(p) {
  sum(as.numeric(as.character(p)))
}

Recall <- function(r, p) {
  recall = TP(r, p) / RP(r)
  if (is.nan(recall))
    return(0)
  else
    return(recall)
}

Precision <- function(r, p) {
  precision = TP(r, p) / PP(p)
  if (is.nan(precision))
    return(0)
  else
    return(precision)
}

F1Score <- function(r, p) {
  recall = Recall(r, p)
  precision = Precision(r, p)
  f1score = 2 * recall * precision / (recall + precision)
  if (is.nan(f1score))
    return(0)
  else
    return(f1score)
}

is.infinite.data.frame <- function(obj){
  sapply(obj,FUN = function(x) all(is.infinite(x)))
}
```

Listado 4.6: Utilitarios

```
#!/usr/bin/env python

from __future__ import print_function
import sys
import os
from subprocess import *

svmtrain_exe = "../svm-train2"
svmpredict_exe = "../svm-predict2"
grid_py = "./grid2.py"
subset_py = "./subset.py"

train_pathname = sys.argv[1]
file_name = os.path.split(train_pathname)[1]

test_pathname = sys.argv[2]
file_name = os.path.split(test_pathname)[1]

for idx in range(500, 5500, 500):
  cmd = '{0} "{1}" "{2}" "{3}"'.format(subset_py, train_pathname, idx,
      'lc{0}.crs'.format(idx))
  print('Subset training data ({0} samples)...'.format(idx))
  Popen(cmd, shell = True, stdout = PIPE).communicate()

  cmd = '{0} -svmtrain "{1}" "{3}"'.format(grid_py, svmtrain_exe,
      'lc{0}.crs'.format(idx))
  print('Cross validation...')
  f = Popen(cmd, shell = True, stdout = PIPE).stdout
  line = ''
  while True:
    last_line = line
    line = f.readline()
    if not line: break
  c,g,rate,accu = map(float,last_line.split())
  print('Best {0} c={1}, g={2} CV rate={3} accu={4}'.format(idx,c,g,rate,accu))

  cmd = '{0} -c {1} -g {2} "{3}" "{4}"'.format(svmtrain_exe,c,g,
      'lc{0}.crs'.format(idx),'{0}{1}.model'.format(file_name,idx))
  print('Training...')
  Popen(cmd, shell = True, stdout = PIPE).communicate()

  print('Output model: {0}{1}.model'.format(file_name,idx))

  cmd = '{0} "{1}" "{2}" "{3}"'.format(svmpredict_exe, test_pathname,
      '{0}{1}.model'.format(file_name,idx), '{0}{1}.predict'.format(file_name,
  idx))
  print('Testing...')
  Popen(cmd, shell = True).communicate()

  print('Output prediction: {0}{1}.predict'.format(file_name,idx))
```

Listado 4.7: Curva de Aprendizaje con SVM

```
#!/usr/bin/env python

from __future__ import print_function
import sys
import os
from subprocess import *

if len(sys.argv) <= 1:
  print('Usage: {0} week training_file testing_file'.format(sys.argv[0]))
  raise SystemExit

# svm executable file

svmtrain_exe = "../svm-train2"
svmpredict_exe = "../svm-predict2"

assert os.path.exists(svmtrain_exe),"svm-train executable not found"
assert os.path.exists(svmpredict_exe),"svm-predict executable not found"

week = sys.argv[1]
train_pathname = sys.argv[2].replace('.',week+'.')
assert os.path.exists(train_pathname),"training file not found"
file_name = os.path.split(train_pathname)[1]
test_pathname = sys.argv[3].replace('.',week+'.')
assert os.path.exists(test_pathname),"testing file not found"
testfile_name = os.path.split(test_pathname)[1]

cmd = '{0} -c 15 -g 3 "{1}" "{2}"'.format(svmtrain_exe,train_pathname,
  '{0}.model'.format(file_name))
print('Training...')
Popen(cmd, shell = True, stdout = PIPE).communicate()
print('Output model: {0}.model'.format(file_name))
cmd = '{0} "{1}" "{2}" "{3}"'.format(svmpredict_exe, train_pathname,
  '{0}.model'.format(file_name), '{0}.predict'.format(file_name))
Popen(cmd, shell = True).communicate()

cmd = '{0} "{1}" "{2}" "{3}"'.format(svmpredict_exe, test_pathname,
  '{0}.model'.format(file_name), '{0}.predict'.format(testfile_name))
print('Testing...')
Popen(cmd, shell = True).communicate()
print('Output prediction: {0}.predict'.format(testfile_name))
```

Listado 4.8: Modelo para cada semana con SVM

```
from __future__ import absolute_import
from __future__ import print_function

import tensorflow as tf
import sys
import numpy as np
import random
import keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation, GRU,
    TimeDistributed
from keras.layers.noise import GaussianDropout
from keras.utils import np_utils
from keras.models import Sequential
import keras.metrics as metrics_module
import time

num_classes = 2
input_dim = 289
loss = 'categorical_crossentropy'
optimizer = 'adam'

def time_usage(func):
    def wrapper(*args, **kwargs):
        beg_ts = time.time()
        retval = func(*args, **kwargs)
        end_ts = time.time()
        print("elapsed time: {0}".format((end_ts - beg_ts) / 60.0))
        return retval
    return wrapper

def _TP(r,p):
    return np.sum(r.argmax(1) * p.argmax(1), dtype=np.float64)

def _FP(r,p):
    return np.sum((1 - r.argmax(1)) * p.argmax(1), dtype=np.float64)

def _TN(r,p):
    return np.sum((1 - r.argmax(1)) * (1 - p.argmax(1)), dtype=np.float64)

def _FN(r,p):
    return np.sum(r.argmax(1) * (1 - p.argmax(1)), dtype=np.float64)

# How many selected items are relevant?
def _precision(y_true, y_pred):
    c1 = np.sum(y_true.argmax(1) * y_pred.argmax(1), dtype=np.float64)
    c2 = np.sum(y_pred.argmax(1), dtype=np.float64)
    # If there are no true predictions, fix the precision at 0.
    if c2 == 0:
        return 0.0
    return (c1 / c2)

# How many relevant items are selected?
def _recall(y_true, y_pred):
    c1 = np.sum(y_true.argmax(1) * y_pred.argmax(1), dtype=np.float64)
    c3 = np.sum(y_true.argmax(1), dtype=np.float64)
    # If there are no true samples, fix the recall at 0.
    if c3 == 0:
        return 0.0
    return (c1 / c3)
```

```
# Calculate f1_score
def _f1_score(y_true, y_pred):
    p = _precision(y_true, y_pred)
    r = _recall(y_true, y_pred)
    if (p + r == 0):
        return 0.0
    return (2 * (p * r) / (p + r))

def get_test_data(week):
    x1 = np.load(data_path + 'x1.npy')
    y1 = np.load(data_path + 'y1.npy')

    dataset = np.concatenate((x1, y1), axis=1)

    dataset_0 = dataset[np.where(dataset[:,-1] == 0)[0],:]
    dataset_1 = dataset[np.where(dataset[:,-1] == 1)[0],:]

    random.shuffle(dataset_0)
    random.shuffle(dataset_1)

    (r0, c0) = dataset_0.shape
    (r1, c1) = dataset_1.shape

    t0 = int(r0*0.7)
    t1 = int(r1*0.7)
    wc = week * 17

    x_train = np.concatenate((dataset_0[0:t0, 0:wc],
        dataset_1[0:t1, 0:wc]), axis=0)
    y_train = np.concatenate((dataset_0[0:t0,   -1],
        dataset_1[0:t1,   -1]), axis=0)
    x_test  = np.concatenate((dataset_0[t0:,  0:wc],
        dataset_1[t1:,  0:wc]), axis=0)
    y_test  = np.concatenate((dataset_0[t0:,    -1],
        dataset_1[t1:,    -1]), axis=0)

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    return (x_train, y_train), (x_test, y_test)

def create_model(dropout, hidden_layers, hidden_neurons):
    model = Sequential()

    model.add(Dense(hidden_neurons, input_shape=(input_dim,)))
    model.add(GaussianDropout(dropout))
    model.add(Activation('relu'))

    for layer in range(0,hidden_layers):
        model.add(Dense(hidden_neurons))
        model.add(GaussianDropout(dropout))
        model.add(Activation('relu'))

    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    return model

@time_usage
def search_params_complexity():
    data_path = '/home/jnavas/torque-scripts/keras/nn/data/'
    array_idx = int(sys.argv[1])

    idx = array_idx % 256
    week = (array_idx // 256) + 3

    layers = [5,9,12,15]
```

```
    neurons = [250,350,450,550]
    dropouts = [0.075,0.05,0.1,0.15]
    batchs = [64,128,256,512]

    np.random.seed(1337)
    hidden_layers = layers[idx % 4]
    hidden_neurons = neurons[(idx // 4) % 4]
    dropout = dropouts[(idx // 16) % 4]
    batch_size = batchs[(idx // 64) % 4]

    input_dim = week * 17

    model = create_model(dropout, hidden_layers, hidden_neurons)
    model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
    (x_train, y_train), (x_test, y_test) = get_test_data(week)

    epochs = 2500
    initial_epoch = 1

    for epoch in range(200, epochs + 100, 100):
        model.fit(x_train, y_train, epochs=epoch, initial_epoch=initial_epoch,
            batch_size=batch_size, verbose=0)
        initial_epoch = epoch
        score = model.evaluate(x_train, y_train, verbose=0)

        trained = model.predict(x_train, batch_size=batch_size, verbose=0)
        predicted = model.predict(x_test, batch_size=batch_size, verbose=0)

        score2 = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=0)

        precisiont = _precision(y_train, trained)
        recallt = _recall(y_train, trained)
        f1t = _f1_score(y_train, trained)

        precision = _precision(y_test, predicted)
        recall = _recall(y_test, predicted)
        f1 = _f1_score(y_test, predicted)

        s = y_train
        t = trained
        r = y_test
        p = predicted
        acct = (_TP(s,t)+_TN(s,t)) * 100.0/(_TP(s,t)+_FP(s,t)+_TN(s,t)+_FN(s,t))
        accp = (_TP(r,p)+_TN(r,p)) * 100.0/(_TP(r,p)+_FP(r,p)+_TN(r,p)+_FN(r,p))

        print('Week={0}, Epochs={1}, BatchSize={2}, Layers={3}, Neurons={4},
    Weight={5}, Dropout={6}'.format(week, epoch, batch_size, hidden_layers,
    hidden_neurons, high_weight, dropout))
        print('Week={0}, Train TP={1}, FP={2}, TN={3}, FN={4}'.format(week, _TP(s
    ,t),_FP(s,t),_TN(s,t),_FN(s,t)))
        print('Week={0}, Train Loss={1}, Accuracy={2}, F1-Score={3}, Precision
    ={4}, Recall={5}'.format(week, score[0], acct, f1t, precisiont, recallt))
        print('Week={0}, Test TP={1}, FP={2}, TN={3}, FN={4}'.format(week, _TP(r,
    p),_FP(r,p),_TN(r,p),_FN(r,p)))
        print('Week={0}, Test Loss={1}, Accuracy={2}, F1-Score={3}, Precision
    ={4}, Recall={5}'.format(week, score2[0], accp, f1, precision, recall))

if __name__ == '__main__':
  search_params_complexity()
```

Listado 4.9: Modelo para cada semana con NN