UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM VALLE DE MÉXICO

MODELO PREDICTIVO BASADO EN ERRORES DE COMPILACIÓN PARA EL APRENDIZAJE DE PROGRAMACIÓN EN C

T E S I S

Que para obtener el Grado de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P r e s e n t a

Lic. Víctor Gonzalo Rivero Martínez

Tutor Académico

Dra. Maricela Quintana López

Tutores Adjuntos

Dr. Asdrúbal López Chau

Dr. Víctor Manuel Landassuri Moreno

Atizapán de Zaragoza, Edo. de Méx. Agosto 2023

VALLE DE MÉXICO

Dedicatorias y Agradecimientos

Dedico el resultado del presente trabajo a mi familia y agradezco a mis asesores por su paciencia, apoyo y dedicación con su labor docente, a mis compañeros de Maestría, a la Universidad Autónoma del Estado de México y al Consejo Nacional de Humanidades Ciencias y Tecnologías por el apoyo recibido.

Resumen

Hoy en día, los cursos de programación son fundamentales en el nivel universitario porque permiten a los estudiantes desarrollar las competencias para aplicar o crear nuevas tecnologías computacionales, además de destrezas lingüísticas orales y escritas, de computación y matemáticas. Sin embargo, de acuerdo con diversos estudios, los estudiantes a menudo tienen dificultades durante el proceso de codificación, siendo uno de los factores la interpretación de los mensajes de error del compilador, los cuales al no entenderlos plenamente ya sea por el lenguaje, el idioma o su estructura, pueden llegar a frustrarlos o desanimarlos. La dificultad en la interpretación se debe a que un error especifico puede producir diferentes mensajes del compilador dependiendo del contexto del código del programa.

Este problema ha sido abordado de diferentes enfoques, por un lado, los que proponen reparar el código y por otro, los que hacen a los mensajes de error del compilador más entendibles.

En este trabajo, se considera el aprendizaje basado en errores y, en lugar de corregir errores en el código, se utilizan mensajes de error del compilador para proporcionar retroalimentación sobre errores de sintaxis a los estudiantes de programación. La retroalimentación consta de cuatro componentes: una traducción del mensaje al español, información de sintaxis sobre el elemento del idioma, la relación del error con las posibles causas y el tema a revisar.

La información proporcionada tiene como objetivo ayudar a los estudiantes a comprender el error y poder reescribir el código y compilarlo correctamente. Para lograr ese objetivo, se construyó un modelo predictivo de aprendizaje supervisado cuya función es predecir el error de sintaxis de acuerdo con el mensaje de error del compilador. El corpus o conjunto de documentos se generó mediante un proceso de inyección de errores en programas modelo, posteriormente etiquetados según el tipo de error de sintaxis. Los algoritmos de aprendizaje automático utilizados fueron Decision Tree, Support Vector Machine, Random Forest, Multi-layer Perceptron y K-Nearest Neighbors, entrenados con el 80% del corpus y evaluados con el 20% restante, logrando una precisión superior al 90% para realizar nuevas predicciones y posteriormente dar retroalimentación.

.

Abstract

Today, programming courses are essential at the university level because they allow students to develop the skills to apply or create new computer technologies, in addition to oral and written language, computing and mathematical skills. However, according to various studies, students often have difficulties during the coding process, one of the factors is the interpretation of compiler error messages, which by not fully understanding them either due to the language and its structure, can frustrate or discourage them. The difficulty in interpretation is because a specific error can produce different compiler messages depending on the context of the program code.

This problem has been addressed using different approaches, on the one hand, those that propose repairing the code and on the other, those that make the compiler error messages more understandable.

In this work, error-based learning is considered and instead of fixing errors in the code, compiler error messages are used to provide feedback on syntax errors to programming students. The feedback consists of four components: a translation of the message into Spanish, syntax information about the language element, the relationship of the error to possible causes, and the topic to be reviewed.

The information provided is intended to help students understand the error so they can rewrite the code and compile it correctly. To achieve this goal, a supervised learning predictive model was built whose function is to predict the syntax error according to the compiler error message. The corpus or set of documents was generated through a process of injecting errors into model programs, subsequently labeled according to the type of syntax error. The machine learning algorithms used were Decision Tree, Support Vector Machine, Random Forest, Multi-layer Perceptron and K-Nearest Neighbors, trained with 80% of the corpus and evaluated with the remaining 20%, achieving an accuracy greater than 90% for make new predictions and subsequently give feedback.

Contenido

Capítulo 1. Introducción 1
1.1 Antecedentes 1
1.2 Planteamiento del problema 4
1.3 Objetivos 6
1.3.1 Objetivo General 7
1.3.2 Objetivos Específicos 7
1.4 Delimitación o alcances de la investigación 7
1.5 Hipótesis 7
1.6 Justificación 7
1.7 Fundamentación Inicial 8
1.7.1 Aprendizaje basado en el error 8
1.7.2 Minería de Textos 9
1.7.3 Construcción del modelo predictivo 9
1.8 Metodología 10
1.9 Publicaciones derivadas del trabajo de investigación 11
1.10 Organización del capitulado 12
Capítulo 2. Marco Teórico y Estado del Arte 13
2.1 Análisis de datos 13
2.2 Inteligencia Artificial 13
2.3 Aprendizaje Automático 14
2.4 Minería de Datos 15
2.4.1 Introducción 15
2.4.2 Proceso de Descubrimiento de Conocimiento (KDD) 16
2.5 Minería de Textos 17
2.5.1 Introducción 17
2.5.2 Metodología para la Minería de Textos 20
2.6 Clasificación de textos 21
2.6.1 Vectorización 23
2.6.1.1 Bolsa Palabras 23
2.6.2 Algoritmos para la clasificación de textos 26

2.7 Métricas de evaluación 34
2.7.1 Exactitud 34
2.7.2 Precisión 34
2.7.3 Recall 35
2.7.4 F1 Score 35
2.7.5 Matriz de confusión 35
2.8 Estado del Arte 36
Capítulo 3. Desarrollo del modelo 42
3.1 Generación del corpus 42
3.1.2 Limpieza de datos 47
3.1.3 Vectorización 47
3.2 Proceso Inductivo 49
3.3 Evaluación del modelo 51
3.4 Métricas de evaluación 54
Capítulo 4. Desarrollo del prototipo del sistema 63
4.1 Descripción del sistema 63
4.2 Funciones del sistema 64
4.3 Características de los usuarios 64
4.4 Suposiciones y dependencia 64
4.5 Requisitos Futuros 65
4.6 Especificación de Requerimientos 65
4.7 Interfaz de usuario 65
4.8 Requerimientos funcionales 66
4.9 Diagrama de casos de uso 67
4.10 Acceso al sistema 68
Capítulo 5. Conclusiones y trabajo futuro 69
Referencias 70

Índice de tablas

Tabla 1.1 Dificultades que presentan los estudiantes al programar 2
Tabla 1.2 Clasificación de las herramientas de apoyo y su retroalimentación 3
Tabla 2.1 Ejemplo de bolsa de palabras para dos documentos 24
Tabla 3.1 Elementos removidos de los programas modelo 45
Tabla 3.2 Clases formadas 46
Tabla 3.3 Cálculo del estadístico TF-IDF de un documento inicial 48
Tabla 3.4 Normalización del vector TF-IDF 49
Tabla 3.5 Transformación del mensaje de error 49
Tabla 3.6 Métricas de evaluación de la configuración de parámetros predefinida 52
Tabla 3.7 Configuración de Hyperparámetros 53
Tabla 3.8 Resultado de las Métricas de Evaluación 54
Tabla 3.9 Matriz de confusión del clasificador Random Forest 55
Tabla 3.10 Matriz de confusión del clasificador Decision Tree 56
Tabla 3.11 Matriz de confusión del clasificador de Multi-layer perceptron 57
Tabla 3.12 Matriz de confusión para el clasificador K-Nearest Neighbors 58
Tabla 3.13 Matriz de confusión del clasificador Support Vector Machine 59
Tabla 3.14 Desempeño de clasificadores con el conjunto de prueba 59
Tabla 3.15 Desempeño de clasificadores con el nuevo conjunto de datos 60
Tabla 3.16 Predicciones del clasificador Random Forest para el conjunto de prueba 60
Tabla 3.17 Predicciones del Voting para el conjunto de prueba 61
Tabla 3.18 Predicciones del clasificador Random Forest para el nuevo conjunto de datos 61
Tabla 3.19 Predicciones del Voting para el nuevo conjunto de datos 62

Índice de figuras

Figura 1.1 Problemas al resolver un ejercicio de programación 3
Figura 1.2 Solución de los errores de compilación 5
Figura 1.3 Relación de los errores de compilación con los temas asociados 5
Figura 1.4 Relación entre el aprendizaje y el tiempo que le dedica el profesor 6
Figura 1.5 Proceso de Minería de Texto 9
Figura 1.6 Metodología 11
Figura 2.1 Campos del análisis de datos 13
Figura 2.2 Tipos de Aprendizaje Automático 15
Figura 2.3 Taxonomía de la minería de datos 16
Figura 2.4 Proceso KDD 17
Figura 2.5 Minería de textos y su interacción con otras áreas 18
Figura 2.6 Metodología de minería de textos 21
Figura 2.7 Proceso de construcción de un clasificador automático 22
Figura 2.8 Entrenamiento y evaluación de un clasificador 23
Figura 2.9 Ponderación en el modelo de Bolsa de Palabras 24
Figura 2.10 Relevancia de las palabras bajo la vectorización TF-IDF 26
Figura 2.11 Márgenes de hiperplano de separación τ. a) No óptimo, b) Óptimo .. 27
Figura 2.12 El método Kernel 28
Figura 2.13 Funciones Kernel 28
Figura 2.14 Estructura general de un árbol de decisión 29
Figura 2.15 Clasificación de una muestra con el algoritmo KNN 31
Figura 2.16 Distancias Minkowski, euclidiana, Manhattan y de Chebyshev 32
Figura 2.17 Marco conceptual del clasificador de bosque aleatorio 32
Figura 2.18 Modelo de McCulloch-Pitts para una neurona artificial 33
Figura 2.19 Perceptrón 33
Figura 2.20 Multilayer perceptron 34
Figura 2.21 Ejemplos de matrices de confusión. (a) Matriz de confusión para una clasificación binaria. (b) Matriz de confusión para la clasificación multiclase. 35
Figura 2.22 Detección y reparación de un error 36
Figura 2.23 Localización y reparación de una línea de código 37
Figura 2.24 Ilustración del gráfico de retroalimentación del programa. 37
Figura 2.25 Modelo de reparación de programas (MultiFix) 38
Figura 2.26 Arquitectura de la herramienta Synshine 38
Figura 2.27 Diagrama del modelo CLACER 39
Figura 2.28 Entrada y salida de la herramienta GrammarGuru 39
Figura 2.29 Interfaz de usuario de la herramienta TEGCER. 40
Figura 2.30 Diagrama de flujo del sistema Learnskell 40
Figura 2.31 Mensaje mejorado como salida del sistema Learnskell 41
Figura 3.1 Metodología de Minería de Textos Aplicada 42
Figura 3.2 Generación del corpus 42
Figura 3.3 Estructura básica de un programa 43
Figura 3.4 Ejemplo de programa modelo 43
Figura 3.5 Programa con error 44

Figura 3.6 Documento generado (Mensaje de error del compilador) 44
Figura 3.7 Elementos eliminados de la función printf 45
Figura 3.8 Ejemplo de limpieza de datos 47
Figura 3.9 Proceso inductivo 49
Figura 3.10 Arquitectura de la clasificación por votación (Voting) 52
Figura 4.1 Arquitectura del sistema 64
Figura 4.2 Proceso de recomendación 65
Figura 4.3 Interfaz de Usuario 66
Figura 4.4 Recomendaciones 66
Figura 4.5 Diagrama de casos de uso 67
Figura 4.6 Diagrama de Secuencia 67
Figura 4.7 Símbolo del sistema 68
Figura 4.8 Ejecución del sistema 68

Capítulo 1. Introducción

1.1 Antecedentes

El progreso tecnológico influye en diversos sectores como el laboral, el doméstico, el educativo, el industrial y el entretenimiento, entre otros. La humanidad se encuentra en un momento en el que los avances en la inteligencia artificial, el aprendizaje automático, la ciencia de datos y el internet de las cosas, entre otras, permiten innovar productos y servicios que utilizamos en nuestras actividades diarias (González A. , 2017).

La tecnología forma gran parte de nuestras actividades cotidianas, por lo tanto, se requiere de personas con un conocimiento especializado que permita crear nueva tecnología o modificar la existente con el fin de actualizarla o mejorarla (Bordes, 2021).

De acuerdo con (Sun & Sun, 2011), impulsar el avance tecnológico en la industria requiere de la formación de ingenieros que tengan la capacidad de resolver problemas a través del desarrollo de aplicaciones, por ello es importante que desde sus primeros años en la universidad los estudiantes aprendan a programar. Los cursos de programación permiten a los estudiantes desarrollar competencias para aplicar o crear nuevas tecnologías computacionales, además de destrezas lingüísticas orales y escritas, de computación y matemáticas (Tejera, Aguilera, & Miguel, 2021).

La enseñanza de la programación usualmente consiste en la impartición de conocimientos y realización de prácticas en una sala de cómputo. De esta forma, se hace hincapié en que, para aprender a programar hay que programar constantemente, aumentando progresivamente la complejidad. En este sentido, programar es crear la solución a un problema utilizando un lenguaje de programación (Fuentes & Moo, 2017), lo cual requiere capacidad de análisis y abstracción con el fin de diseñar y proponer la solución más adecuada al problema, por ello se considera que programar es una tarea compleja (Campañ, Satorre, Llorens, & Molina, 2015).

Esta dificultad al programar se ve reflejada en el alto índice de reprobación en materias de programación. Por ejemplo, en un estudio realizado en el año 2016 a estudiantes de ingeniería en la Universidad Autónoma de Ciudad Juárez se aplicó un examen departamental mostrando un bajo desempeño de los estudiantes, obteniendo un promedio de 4.17, cuando la mínima aprobatoria es de 7.0. De los 223 estudiantes de la materia sólo 170 presentaron el examen y de éstos el 83% lo reprobó (Noriega, Mendoza, Robledo, Acosta, & Esquivel, 2016).

En otro estudio realizado en el año 2016 en el Instituto Tecnológico de Mexicali, se tomaron muestras de los indicadores, de los periodos de enero 2012 a junio 2015, de la materia de Fundamentos de Programación, mostrando un alto índice de reprobación, el cual frecuentemente se encuentra en un rango de entre el 50% y el 66% (Viveros , López , & Villareal , 2016).

En (Fuentes & Moo, 2017) se determinan y clasifican las dificultades que tienen los estudiantes de las ingenierías en sistemas y electromecánica al momento de solucionar problemas mediante programación básica, así como las posibles soluciones, las cuales se presentan en la Tabla 1.1; en la primera columna de la tabla, se describen las principales dificultades encontradas, mientras que en la segunda columna, se presenta la descripción de estas dificultades y en la tercera las propuestas de solución.

Tabla 1.1 Dificultades que presentan los estudiantes al programar

Dificultad	**Descripción**	**Propuesta de solución**
Fobia a los problemas complejos	Los estudiantes tienden a no solucionar problemas que los impacten	Enseñar al estudiante a dividir el problema en pequeños subproblemas
Lógica incompleta	No consiguen establecer los pasos necesarios para llegar a una solución completa	Fomentar la solución de acertijos en los que se apliquen los pasos para solucionar un problema
Desconocen el lenguaje	Desconocen las palabras reservadas o bibliotecas del lenguaje que podrían utilizar al codificar el programa	Elaborar material didáctico que sirva como guía
Desconocer las herramientas del entorno de desarrollo IDE	Los estudiantes escribían sus códigos, pero no lograban corregir los errores de lógica	Enseñar a los estudiantes como correr el programa paso a paso empleando el entorno de programación

En el Centro Universitario UAEM Valle de México se aplicó una encuesta a 193 estudiantes que han llevado al menos un curso de programación, particularmente se les preguntó cuáles eran las razones por las que no pueden resolver un ejercicio que se les dejó como actividad. En la Figura 1.1 se muestran los porcentajes de cada respuesta y se observa que la razón principal con 39.4% es que no saben cómo programarlo, aunque comprendan el problema y sepan cómo resolverlo.

A fin de mitigar las dificultades de los estudiantes al codificar, se han desarrollado herramientas de apoyo que abordan el problema desde diferentes

enfoques, por un lado, las que proponen reparar el código y por otro, las que hacen a los mensajes de error del compilador más entendibles.

Figura 1.1 Problemas al resolver un ejercicio de programación

En el año 2015, en la Escuela Politécnica Nacional de Quito Ecuador se realizó un estudio comparativo de las herramientas de apoyo en el proceso de enseñanza-aprendizaje de programación. En la Tabla 1.2, se presenta un concentrado del tipo de herramienta y la retroalimentación que ofrecen (Guerrero, Guaman, & Icaza, 2015).

Tabla 1.2 Clasificación de las herramientas de apoyo y su retroalimentación

Herramienta	**Retroalimentación**
Sistemas de calificación automática	Automatización del proceso de calificación y generación de ejercicios de programación
Herramientas multimedia	Utilizan recursos como textos, imágenes, videos, entre otros, que ofrecen explicaciones y revisión de conceptos
Sistemas inteligentes de tutoría	Herramientas complejas, orientadas al soporte de los estudiantes durante la escritura de sus programas, explicando los errores del compilador
Herramientas de aprendizaje visual	Orientadas a representar gráficamente el algoritmo de un programa y su ejecución
Entornos de Desarrollo Integrado (IDE)	Coloreado de sintaxis permitiendo una mejor distinción de palabras reservadas, identificadores, variables etc.

Algunas otras herramientas de apoyo que se han realizado son los siguientes:

JECA (Java Error Correcting Algorithm). Algoritmo de corrección de errores para el sistema de tutoría inteligente de Java. Se trata de un algoritmo práctico que compara las palabras del código fuente con palabras reservadas del lenguaje para posteriormente corregir las palabras mediante inserciones, eliminaciones y

reemplazos. Por ejemplo, al encontrar "Int" en alguna línea, se producirá un mensaje de tipo "encontré un Int ¿debería reemplazarlo por int? (s / n)", de esta manera, el usuario del sistema es consciente de todos los cambios que se realizan en el código enviado (Sykes & Franek, 2004).

TRACER (Targeted Repair of Compilation Errors). Es un sistema para realizar reparaciones en errores de compilación, dirigido a programadores principiantes. El sistema utiliza redes neuronales para reparar errores de compilación prediciendo secuencias de tokens (Ahmed, Kumar, Karkare, Kar, & Gulwani, 2018).
(Yasunaga & Liang, 2020) realizaron un sistema que corrige problemas de sintaxis en el código mediante técnicas de redes neuronales, basándose en los mensajes de error del compilador. Su objetivo es localizar una línea errónea en el programa y generar una línea reparada.

En el presente trabajo se propone la creación de un modelo predictivo que, a partir de los mensajes de error generados por el compilador, determine la retroalimentación que se le debe proporcionar a un estudiante para ayudarle a corregir el error. El modelo formará parte de una herramienta de apoyo en la enseñanza-aprendizaje de la programación en lenguaje C. La retroalimentación consiste en una explicación sobre el error, sugerencias de por qué puede estar ocurriendo, y una recomendación de temas a estudiar. El modelo predictivo será desarrollado aplicando técnicas de minería de textos.

1.2 Planteamiento del problema

Es común que en una clase de programación el docente imparta los conocimientos, muestre un ejemplo y posteriormente pida a los estudiantes que realicen un ejercicio, lo cual es una tarea compleja dado que involucra solucionar un problema y familiarizarse con el lenguaje de programación con el que se va a solucionar.

De hecho, (Dann, Copper, & Pausch, 2006) enumeran cuatro factores que dificultan el aprendizaje de los fundamentos de programación: el uso de la sintaxis; la incapacidad para ver el resultado de los cálculos a la par cuando un programa se ejecuta; la falta de motivación para programar, y la dificultad de la comprensión de la lógica compuesta.

De acuerdo con lo anterior, cuando el estudiante realiza el ejercicio por su cuenta, puede ocurrir que, después de diseñar la solución al problema, su idea sea correcta, pero al momento de programarla el compilador le envíe mensajes sobre errores (deficiencia en el conocimiento del lenguaje) que al no poder corregir le impide probar su solución. Es posible que tales mensajes de error del compilador puedan resolverse rápidamente, como puede ser el caso de una palabra clave mal escrita, pero hay errores que son difíciles de detectar como una llave mal puesta.

En la encuesta realizada en el Centro Universitario UAEM Valle de México, se hicieron dos preguntas relacionadas con los errores de compilación, las respuestas se observan en la Figura 1.2 y Figura 1.3.

Figura 1.2 Solución de los errores de compilación

En la Figura 1.2 se puede observar que solo el 10.4 % de los estudiantes encuestados entienden y pueden resolver los problemas del compilador sin dificultad, es decir casi el 90% de los estudiantes tienen alguna dificultad en resolver los mensajes de error.

De manera similar en la Figura 1.3 se observa que solo el 45.55 % de los estudiantes frecuentemente o siempre saben a qué se refieren los errores que envía el compilador y lo relacionan con algún tema en particular, mientras que el 54.39% algunas veces o nunca los entienden.

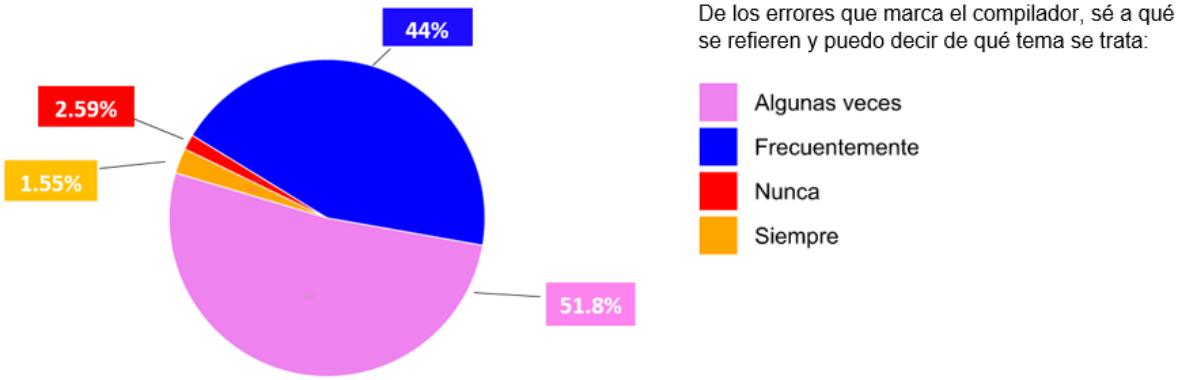


Figura 1.3 Relación de los errores de compilación con los temas asociados

La función del compilador es detectar los errores léxicos, sintácticos y algunos errores semánticos, como los que pueden ocurrir al realizar la comprobación de tipos. Que un estudiante no pueda corregir este tipo de errores, implica un desconocimiento del lenguaje, así como la falta de interpretación del mensaje para

establecer la relación entre el error y el tema que lo causa. Además, se ha identificado que un porcentaje importante del grupo se apoya en el docente para corregir los errores de compilación, pero debido al número de estudiantes, el docente no puede dar una atención personalizada en el tiempo que tiene.

El problema del tiempo de atención, se maximiza si son varios estudiantes con problemas en la compilación, por ejemplo, si son 5 estudiantes con errores de sintaxis sencillos que el docente puede resolver en máximo un minuto incluyendo la retroalimentación que se le da al estudiante, el tiempo máximo de espera es de 5 minutos, pero si el error es más complejo de descubrir por el docente, que tarde de 3 a 5 minutos para resolver, el estudiante puede llegar a esperar hasta 20 minutos para que su programa se ejecute correctamente.

Lo anterior, puede afectar la planificación de una clase y ocasionar frustración en los estudiantes en su aprendizaje de la programación. En la Figura 1.4 se observa que el 91.19% de los estudiantes está de acuerdo o parcialmente de acuerdo en que, si el profesor les dedicara más tiempo, aprenderían más.

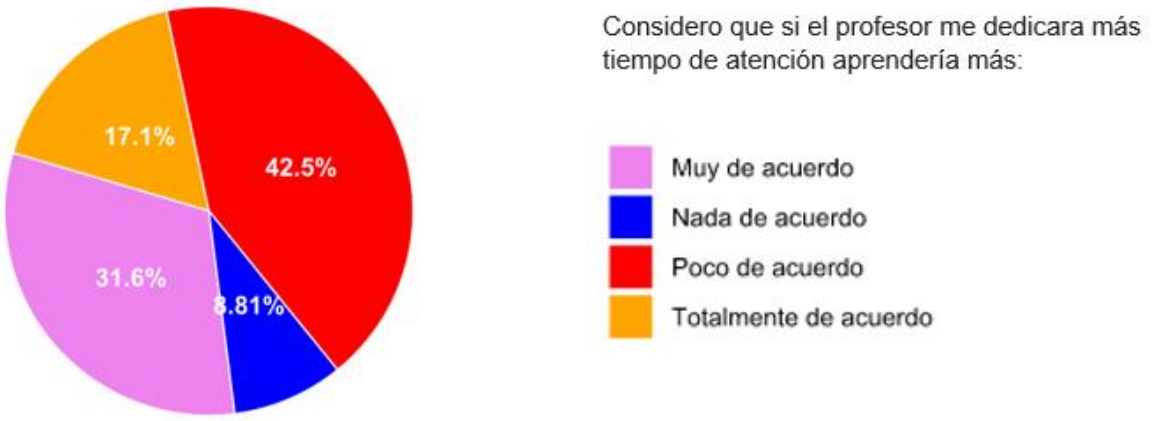


Figura 1.4 Relación entre el aprendizaje y el tiempo que le dedica el profesor

En este sentido, contar con una herramienta de apoyo en la enseñanza-aprendizaje de la programación en lenguaje C, como la propuesta en este trabajo, no solo ayudaría al estudiante sino también al profesor.

1.3 Objetivos

En el presente trabajo se formulan los siguientes objetivos:

1.3.1 Objetivo General

Desarrollar un modelo predictivo de apoyo al proceso de enseñanza-aprendizaje de programación en lenguaje C basado en técnicas de minería de textos y aprendizaje automático que, durante la codificación de programas, retroalimente a los estudiantes a partir de los mensajes de error del compilador.

1.3.2 Objetivos Específicos

- Generar un banco de datos de errores de compilación.
- Categorizar los mensajes de error.
- Extraer las características de los mensajes de error para generar una representación computacionalmente adecuada para su procesamiento.
- Generar el modelo predictivo utilizando algoritmos de aprendizaje automático.
- Desarrollar un prototipo de sistema para el uso del modelo predictivo

1.4 Delimitación o alcances de la investigación

El modelo reconocerá errores relativos al uso de sintaxis y temas de estudio del lenguaje de programación C. Se descartarán las recomendaciones relacionadas con la lógica de programación y errores semánticos. El modelo estará diseñado para alumnos que se encuentren en un curso de programación básica en el nivel superior.

1.5 Hipótesis

Mediante técnicas de minería de textos y aprendizaje automático es posible generar un modelo predictivo que clasifique los errores sintácticos y pueda integrarse a un sistema de apoyo al estudiante en su aprendizaje de programación en lenguaje C, el cual, a partir de la clasificación de los mensajes de error generados por el compilador, dé una retroalimentación y recomiende temas de estudio.

1.6 Justificación

La enseñanza de la programación tiene como principal objetivo lograr que los alumnos adquirieran competencias para crear programas que resuelvan problemas reales, aunque como se ha mencionado, diversos estudios demuestran que programar no es una tarea fácil, lo que puede comprobarse por los altos niveles de reprobación y deserción. Sin embargo, la utilización de herramientas de apoyo en el proceso de enseñanza-aprendizaje puede beneficiar tanto a los docentes como a los alumnos. En este sentido, si el docente cuenta con una herramienta de apoyo en la interpretación de los mensajes de error del compilador podría disminuir el

tiempo de atención de dudas de los estudiantes sobre estos mensajes y los estudiantes contarían con una herramienta que les ofrezca una interpretación de los mensajes de error, les proponga alternativas de por qué pueden estar ocurriendo y les recomiende temas de estudio, permitiéndoles aprender a partir de la identificación de sus errores.

1.7 Fundamentación Inicial

En este apartado se describen los conceptos fundamentales para el desarrollo del presente proyecto, tales como el aprendizaje basado el error, las técnicas de minería de textos y los principales algoritmos de aprendizaje máquina para minería de textos.

1.7.1 Aprendizaje basado en el error

Durante el proceso de enseñanza-aprendizaje de la programación, es común que surjan errores de compilación cuando el alumno o el docente está codificando la solución a un problema. Hay errores que pueden ser resueltos fácilmente como puede ser el caso de un punto y coma, pero algunos se pueden complicar debido a que no siempre el mensaje de error del compilador trata el error que realmente se encuentra en el código. Es posible aprender de estos errores de programación si se les utiliza para discutir y/o argumentar con los estudiantes la causa de estos, algunas alternativas para corregirlos, además de documentarlos para futuras referencias. De acuerdo con (Álvarez, 2019) en el aprendizaje basado en el error se recurre a esperar que, de forma casual e involuntaria, éste aparezca y en lugar de considerarlos como algo negativo, se pueden crear situaciones de aprendizaje en las que los estudiantes sean parte de su gestión y corrección con el fin de obtener un mejor aprendizaje.

En el estudio realizado por (Heemsoth & Heinze, 2016) se considera que el estudiante puede aprender efectivamente de sus errores, para ello es necesario que reflexione acerca de ellos con el fin de mejorar la adquisición de conocimientos. En la pedagogía del error, el error es un indicador de las áreas de oportunidad tanto del alumno como del profesor y hace eficiente el proceso de enseñanza aprendizaje ya que aprovecha todos los recursos que van surgiendo durante el mismo. Además, los errores se suelen utilizar para evaluar el aprendizaje esperado de los estudiantes y permite replantearse objetivos.

En el caso del aprendizaje de un lenguaje de programación, el compilador es el que se encarga de enviar mensajes de error incluyendo la línea, la posición del error y el error en sí. Sin embargo, la información dada por tales mensajes puede resultar insuficiente para que el estudiante logre corregirlos por su cuenta. Debido a esto, en este trabajo se considera desarrollar una herramienta que ayude al estudiante a comprender el error con el fin de que aprenda de los mismos. Dado

que los mensajes de error del compilador no siempre siguen un patrón definido al generarse de distintos programas, la herramienta será desarrollada utilizando minería de textos.

1.7.2 Minería de Textos

Actualmente las Tecnologías de la Información son utilizadas ampliamente en áreas como la educación, negocios, economía, comunicaciones etc., en las que se generan grandes cantidades de datos. Los cuales pueden analizarse con diversas técnicas de minería de datos con el objetivo de obtener información útil para fines específicos.

La Minería de Datos es la extracción de patrones novedosos y potencialmente útiles a partir de datos. Cuando los datos se refieren a documentos, a texto, se conoce como Minería de Textos. Este proceso consiste en diversas etapas (ver Figura 1.5) que van desde determinar el propósito del estudio hasta la presentación de resultados, pasando por las fases de recuperación, extracción y procesamiento del texto para posteriormente aplicar los métodos de minería que se aplicarán dependiendo del objetivo del estudio (Contreras Barrera, 2014).

Figura 1.5 Proceso de Minería de Texto
(Contreras Barrera, 2014)

En el presente trabajo se utiliza la minería de textos para realizar un modelo predictivo basado en los mensajes de error del compilador, por lo que se explica cómo es su construcción.

1.7.3 Construcción del modelo predictivo

De acuerdo con (Mariñelarena, Errecalde, & Castro, 2017), las tareas predictivas en la minería de textos están basadas en la construcción de un clasificador automático mediante un sistema de aprendizaje supervisado compuesto por las etapas de etiquetado, extracción de características, entrenamiento, evaluación y uso. En la

etapa de entrenamiento se utilizan algoritmos de aprendizaje automático, entre los que se encuentran:

Support Vector Machine. Es un algoritmo que inicialmente separa los datos en dos clases linealmente separables por medio de diversos hiperplanos, para posteriormente encontrar el hiperplano óptimo que maximiza el margen entre los puntos más próximos a las clases. Estos puntos más próximos son los vectores de soporte. El algoritmo no considera los puntos restantes para determinar el hiperplano (Wanjun & Xiaoguang, 2010).

Decision Tree. El árbol de decisión es un clasificador con estructura similar a un árbol, donde cada nodo representa una condición a probarse; las ramas representan la salida de la prueba, es decir si se cumplió la condición y los nodos finales representan la clasificación (Wanjun & Xiaoguang, 2010). Cuando el documento llega al final del árbol ya ha cumplido ciertas condiciones por lo que se le asigna una etiqueta (Feldman & Sanger, 2007).

K-Nearest Neighbors. Es un algoritmo de clasificación que realiza predicciones a una muestra basándose en las distancias más cortas a las clases dadas. La letra K representa el número de vecinos o puntos a agrupar para posteriormente calcular las distancias (IBM, 2015).

Red Neuronal Artificial. De acuerdo con (Tablada & Torres, 2009), son modelos matemáticos basado cuyo funcionamiento se basa en las neuronas del cerebro humano. Actualmente pueden llevar a cabo tareas como el procesamiento de información, reconocimiento de patrones, clasificación entre otras.

Random Forest. Algoritmo propuesto por Leo Breiman de la Universidad de California en el año 2001 y está compuesto por un conjunto de árboles de decisión. Cada árbol de decisión clasifica un conjunto de entrenamiento seleccionado de manera aleatoria. El resultado final es un proceso de votación del conjunto de árboles de decisión (Parmar, Katariya, & Patel, 2018).

1.8 Metodología

La metodología del presente trabajo consta de los siguientes pasos, los cuales se describen a continuación (ver Figura 1.6):

1) Construcción del modelo predictivo
    - Generar un banco de mensajes de errores de compilación. Debido a que no se cuenta con los documentos, estos se generarán mediante la inyección automática de errores a programas correctos para posteriormente compilarlos y recopilar los mensajes de error.
    - Categorizar los mensajes de error (etiquetado). Categorizar los mensajes de error de acuerdo con el tipo de error de compilación.

- Limpieza de documentos. Se eliminan caracteres que no aportan a la clasificación del documento.
- Extraer las características de los mensajes de error para generar una representación adecuada para ser utilizada por el clasificador.
- Construir el modelo clasificador que permita predecir una categoría dado el mensaje de error.

2) Desarrollar un prototipo de sistema para el uso del modelo predictivo

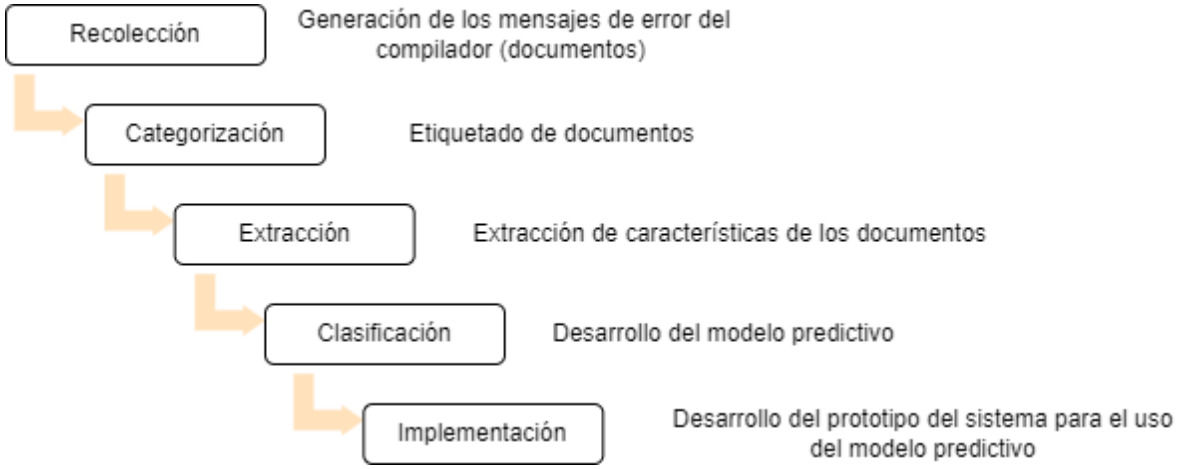


Figura 1.6 Metodología

1.9 Publicaciones derivadas del trabajo de investigación

“Análisis del comportamiento de estudiantes de programación durante su proceso de aprendizaje utilizando árboles de decisión”, publicado en la revista Research in Computing Science Vol. 151 No. 8: Advances in Artificial Intelligence (pp. 75-86), agosto 2022, ISSN: 1870-4069

Además de realizó la ponencia de este, en el XIV Congreso Mexicano de Inteligencia Artificial (COMIA 2022).

Martínez, V.G.R., López, M.Q., Chau, A.L., Moreno, V.M.L. (2024). “Using Compiler Errors Messages to Feedback High School Students Through Machine Learning Methods”, In: Calvo, H., Martínez-Villaseñor, L., Ponce, H. (eds) Advances in Computational Intelligence. MICAI 2023. Lecture Notes in Computer Science(), vol 14391. Springer, Cham. https://doi.org/10.1007/978-3-031-47765-2_22

Se realizó la ponencia de este en el 22nd Mexican International Conference on Artificial Intelligence 2023 • IIMAS, UNAM (UAEY) - UADY, Mérida, Yucatán, México.

1.10 Organización del capitulado

El contenido de este trabajo de tesis es descrito a continuación:
En el capítulo 2, se aborda el marco teórico de minería de textos, aprendizaje automático, inteligencia artificial, algoritmos de clasificación y el estado del arte sobre modelos predictivos aplicados en la enseñanza de la programación.

Se describe, en el capítulo 3, la generación, limpieza y transformación de documentos, el desarrollo del modelo predictivo a partir de los mensajes de error del compilador, así como las métricas de evaluación, matrices de confusión para cada uno de los algoritmos de clasificación, los experimentos realizados y resultados obtenidos en términos de la precisión alcanzada por el modelo predictivo. En el capítulo 4 se presenta el desarrollo del prototipo para la implementación del modelo y por último, las conclusiones y trabajo futuro se describen en el capítulo 5.

Capítulo 2. Marco Teórico y Estado del Arte

En este capítulo se expondrán los conceptos fundamentales del análisis de datos, la inteligencia artificial y el aprendizaje automático con el fin de comprender su funcionamiento y utilización en los procesos de minería de datos, minería de textos y específicamente en la clasificación de Textos. Además, se presenta el Estado del arte sobre modelos predictivos aplicados a la enseñanza de la programación.

2.1 Análisis de datos

El análisis de datos es un conjunto de técnicas computacionales en el cual dado un conjunto de datos se identifican patrones e interrelaciones entre ellos, además se extrae información que posteriormente puede utilizarse para la toma de decisiones. Para lograr lo anterior, convergen diversos campos de estudio (ver Figura 2.1), siendo semejantes en su objetivo, el cual es extraer información útil de los datos (Villén, 2023).

Figura 2.1 Campos del análisis de datos
(Villén, 2023)

A continuación, se describirán los campos de estudio del análisis de datos más relevantes para la presente investigación, específicamente inteligencia artificial, aprendizaje automático y minería de datos.

2.2 Inteligencia Artificial

En 1950, el científico Alan Turing, en su artículo Computing Machinery and Intelligence, se planteó la pregunta: ¿Las máquinas pueden pensar? (Turing, 1950),

posteriormente John McCarthy en 1956 acuño el término de Inteligencia Artificial como la ciencia e ingenio para hacer máquinas inteligentes (Barrera, 2012). Otras definiciones de inteligencia artificial son las siguientes:

> “La inteligencia artificial es el desarrollo e implementación de algoritmos que, integrados a máquinas o sistemas, muestran capacidades similares a la mente humana para resolver problemas” (Iberdrola, 2023).

Para Pineda (Cordova, Flores, García, & Salvador, 2023) del Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas de la UNAM, la Inteligencia Artificial es una disciplina científica y tecnológica cuyo objetivo es construir un programa de cómputo que modele los procesos mentales de un individuo. Estos procesos pueden ser desde identificar personas, objetos, hasta clasificarlos y tomar decisiones.

2.3 Aprendizaje Automático

El aprendizaje automático consiste en extraer información útil de un conjunto de datos. Algunas definiciones de aprendizaje automático son:

> “El aprendizaje automático (ML por sus siglas en inglés, Machine Learning) es una subcategoría de inteligencia artificial que se refiere al proceso por el cual los PC desarrollan el reconocimiento de patrones o la capacidad de aprender continuamente y realizar predicciones basadas en datos, tras lo cual realizan ajustes sin haber sido programados específicamente para ello” (Hewlett Packard Enterprises, 2023).

> “Conjunto de métodos que automáticamente detectan patrones en los datos, y los usan para predecir el futuro o realizar otros tipos de decisiones bajo incertidumbre” (Murphy, 2012).

> “Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y una medida de desempeño P, si su desempeño en las tareas en T, medido por P, mejora con la experiencia E” (Mitchell, 1997).

> “El aprendizaje automático es una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones (análisis predictivo). Este aprendizaje permite a los computadores realizar tareas específicas de forma autónoma, es decir, sin necesidad de ser programados” (Iberdrola, 2023).

Con el aprendizaje automático se pueden realizar tareas de clasificación (asignar etiquetas a los datos), regresión (determinar relaciones entre variables y agrupamiento (agrupar los datos por similitud).

Los algoritmos de Aprendizaje Automático se dividen en las categorías de Aprendizaje Supervisado, Aprendizaje no Supervisado y Aprendizaje por Refuerzo, (Rouhiainen, 2018), las cuales se describen a continuación (ver Figura 2.2). En el aprendizaje supervisado, los algoritmos se entrenan con datos previamente etiquetados, su objetivo es que una vez entrenado el algoritmo se pueda determinar la etiqueta de nuevos datos. Por otro lado, en el aprendizaje no supervisado, los datos no están etiquetados, por lo que el objetivo del algoritmo es agruparlos por características similares. Por último, en el aprendizaje por refuerzo. Los algoritmos aprenden a través de un sistema de recompensa o de prueba y error. Cada vez que aciertan sobre el grupo al que corresponde un tipo de datos se asigna un punto positivo, de manera que van mejorando mediante la experiencia.

Figura 2.2 Tipos de Aprendizaje Automático

Existen varios algoritmos para llevar a cabo los aprendizajes mencionados, entre estos se encuentran: Decision Tree, Support Vector Machine, Random Forest, Artificial Neural Networks y K-Nearest Neighbors. Posteriormente en este capítulo, se explicarán a detalle, los algoritmos de aprendizaje automático empleados en este trabajo, específicamente, aquellos relacionados con la clasificación de textos.

2.4 Minería de Datos

En esta sección se describirán los fundamentos de la minería de datos.

2.4.1 Introducción

De acuerdo con (Molina, 2000), la minería de datos es el proceso de extraer conocimiento de bases de datos, con el fin descubrir anomalías, tendencias, patrones y secuencias de datos.

La mayoría de las técnicas de minería de datos están basadas en el aprendizaje inductivo, con el que se construye un modelo a partir de cierto número de datos los cuales deben considerarse suficientes para que el modelo aprenda y pueda ser utilizado.

Algunas tareas de la minería de datos son la clasificación cuyo objetivo es construir el modelo para predecir la clase de una nueva instancia y la predicción cuando el modelo construido asigna un valor a la nueva instancia. Por otro lado, los modelos descriptivos identifican patrones en los datos con el fin de describirlos, explicarlos o resumirlos. Las principales tareas que realizan los modelos descriptivos son la asociación y el agrupamiento. En la asociación, se identifican relaciones entre los datos, y en el agrupamiento se revisa cómo los datos se agrupan de manera natural usando su grado de similitud.

Los algoritmos utilizados por la minería de datos se muestran en la Figura 2.3, para la predicción y clasificación: redes neuronales artificiales, árboles de decisión, redes bayesianas, k-vecinos más cercanos, random forest y máquinas de soporte vectorial, estos son algoritmos de aprendizaje supervisado, mientras que para la asociación: el algoritmo a priori y para la el agrupamiento: k-means, agrupamiento jerárquico y basado en densidad, estos son algoritmos de aprendizaje no supervisado (Nabeel, y otros, 2021).

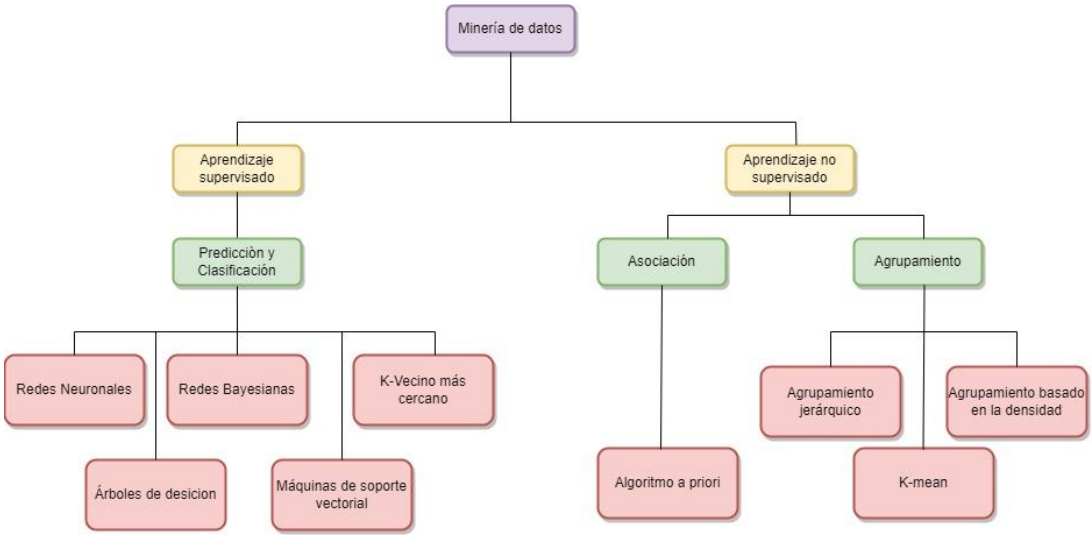


Figura 2.3 Taxonomía de la minería de datos
(Nabeel, y otros, 2021)

2.4.2 Proceso de Descubrimiento de Conocimiento (KDD)

De acuerdo con Fayyad (2002), el proceso de descubrimiento del conocimiento (KDD por las siglas en inglés de Knowledge Discovery and Data Mining) es “el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles

y comprensibles a partir de los datos". Para llevar a cabo el objetivo de identificar información útil, se llevan a cabo los pasos mostrados en la Figura 2.4 y se describen a continuación considerando que previamente al proceso, ya se cuenta con un conjunto significativo de datos (Fayyad, Piatetsky-Shapiro, & Smyth, 1996).

En la etapa de selección, se determina el objetivo de la aplicación con el propósito de recolectar un conjunto inicial de datos que sirva posteriormente para elegir los más relevantes. Luego en el preprocesamiento/limpieza, se obtienen atributos útiles de los datos eliminando todas las impurezas o ruido que pueda afectar la certeza de los resultados de la investigación. A continuación, en la etapa de transformación los datos se transforman de manera que puedan ser procesados eficientemente por los métodos de minería de datos. Después, en la minería de datos se crean los modelos predictivos o modelos descriptivos según sea el caso, seleccionando y aplicando los algoritmos más convenientes. Por último, se interpreta la información obtenida como patrones identificados, relaciones entre variables o predicciones. Para lo anterior se generan reportes escritos, gráficos o visualizaciones.

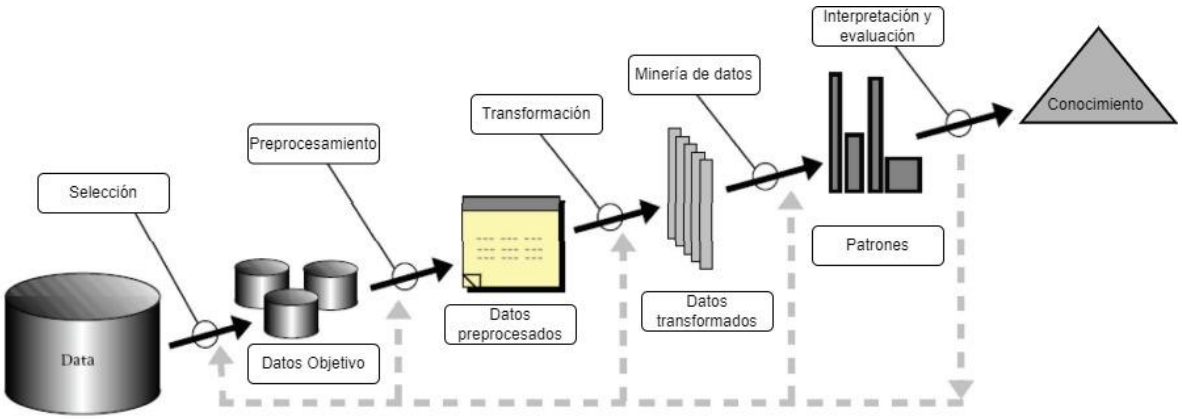


Figura 2.4 Proceso KDD
(Fayyad, Piatetsky-Shapiro, & Smyth, 1996)

2.5 Minería de Textos

En esta sección, se abordará la minería de textos y su metodología como una aplicación particular de la minería de datos.

2.5.1 Introducción

La minería de textos se define como el proceso de descubrir patrones en los datos, cuando los datos son documentos de texto; a continuación, se presentan algunas definiciones:

“La minería de textos es la extracción automática de datos de diferentes recursos escritos con el fin de encontrar en ellos relaciones, patrones o tendencias” (Hearst, 2003).

“La minería de textos busca extraer información útil e importante de formatos de documentos heterogéneos, tales como páginas web, correos electrónicos, medios sociales, artículos de revistas, etc. Esto se hace mediante la identificación de patrones dentro de los textos, tales como tendencias en el uso de palabras, estructura sintáctica, etc.” (Arévalo-Alonso, 2018).

De acuerdo con (Miner, y otros, 2012), la minería de textos es un área multidisciplinaria, basada en la minería de datos, y que utiliza técnicas de inteligencia artificial, lingüística computacional, bases de datos, aprendizaje automático y estadística. En la Figura 2.5 se observan las técnicas de la minería de textos y su interacción con otras áreas.

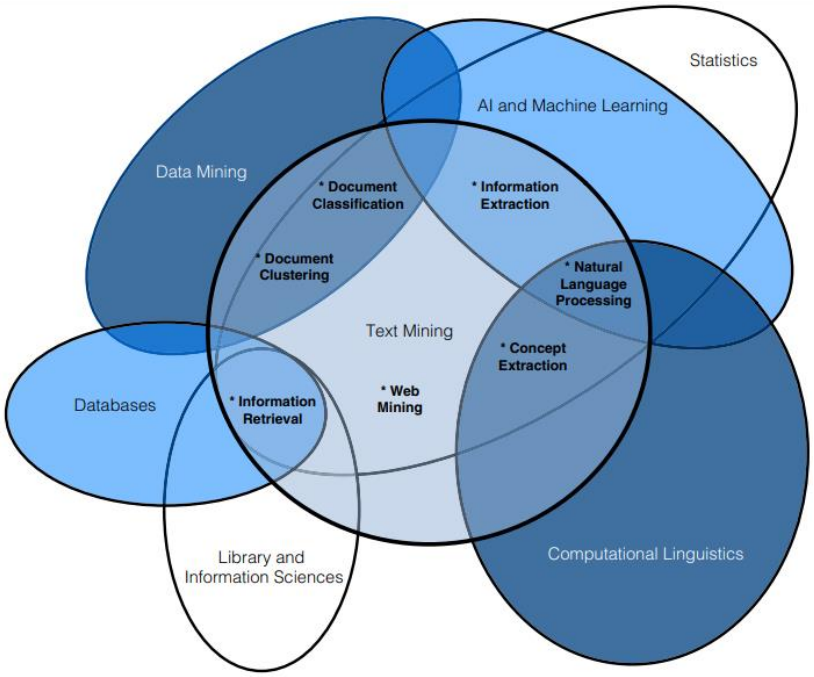


Figura 2.5 Minería de textos y su interacción con otras áreas
(Miner, y otros, 2012)

En los procesos de minería de textos, se trata con texto en lenguaje natural el cual puede estar almacenado en formato estructurado, no estructurado o semiestructurado (Joyanes, 2016).

Los datos estructurados son datos que están en un formato estandarizado, y tienen una estructura bien definida, por ejemplo, las hojas de cálculo con campos fijos. Generalmente este tipo de datos se pueden gestionar fácilmente mediante

bases de datos relacionales. Los datos semiestructurados no tienen un formato definido, pero pueden contener etiquetas que ayudan a distinguir unos de otros. Un ejemplo de este tipo de datos sería el código fuente de una página web, ya que contiene etiquetas y marcadores. Por último, los datos no estructurados son datos no organizados bajo ningún esquema, patrón determinado o tipo predefinido. Ejemplos de estos datos se pueden observar en correos electrónicos, archivos de texto, una imagen, un objeto, archivos de audio, blogs, mensajes de correo de voz, mensajes instantáneos, páginas web entre otros.

El objetivo de la minería de textos es entonces extraer información de texto no estructurado (Pérez y Cardoso, 2010), o más aún, de acuerdo con (Hearst, 2003) es extraer patrones y conocimiento significativo de fuentes de datos de texto convirtiendo datos no estructurados en datos estructurados.

Las técnicas de minería de textos utilizan el procesamiento del lenguaje natural con objetivo de enseñar a la computadora cómo analizar, interpretar y visualizar texto. De acuerdo con (Gaikwad, Chaugule, & Patil, 2014), estas técnicas son las siguientes:

Extracción de información. Tarea de obtener información sistematizada de datos no estructurados o semiestructurados. Esta información se almacena en una base de datos para su uso posterior. La información puede tratarse de nombres de personas, correos electrónicos, patrones, relaciones entre documentos etc.

Clasificación: donde se utilizan documentos predefinidos para clasificar automáticamente nuevos documentos. Algunas técnicas de clasificación son el vecino más cercano, el árbol de decisión y las máquinas de soporte vectorial.

Agrupamiento: En este método se utiliza un algoritmo para agrupar documentos con características comunes. El algoritmo crea un vector de temas para cada documento y determina qué tan bien encaja el documento en cada grupo. Las diferentes técnicas de agrupación son: jerárquica, distribución, densidad, centroide y k-medias.

Resumen de Texto: Consiste en reducir la longitud de un documento conservando los puntos más importantes y el significado general. Es útil para determinar si un documento extenso satisface o no las necesidades del usuario y si vale la pena leerlo para obtener más información, por lo que el resumen puede reemplazar un conjunto de documentos.

Visualización: Su objetivo es representar la información contenida en un documento o conjunto de documentos basándose en jerarquías de los textos y utilizando fuentes de distintos tamaños o densidades de color.

2.5.2 Metodología para la Minería de Textos

De acuerdo con (Maheswari & Sathiaseelan, 2017), la metodología de la minería de textos comienza con la recopilación de documentos, posteriormente el procesamiento de datos, la transformación de datos, el análisis de datos y por último la evaluación de los resultados. Estas etapas se pueden observar en la Figura 2.6 y se describen a continuación:

1. **Recopilación de datos.** Recopilación de documentos en forma de datos no estructurados.

2. **Preprocesamiento de datos.** Se preprocesan los datos recopilados para eliminar inconsistencias, redundancias, palabras vacías y derivaciones. Particularmente, los procesos realizados en esta etapa son:

    - **División del texto (tokenization).** Es el proceso de dividir el texto en tokens, es decir, palabras, símbolos o elementos significativos. Por ejemplo, al tokenizar el texto “hola mundo”, se tendrían las dos palabras separadas: “hola”, “mundo”.

    - **Derivación (stemming).** En la recopilación de información, la derivación es el proceso utilizado para reducir las palabras derivadas a su raíz o forma original con el fin de facilitar la búsqueda de palabras. Por ejemplo, al aplicar derivación a las palabras “programado”, “programado” y “programa”, se reducirán a “progra”.

    - **Eliminación de palabras vacías (stop word removal).** Se eliminan palabras vacías, es decir aquellas que no juegan un papel determinante en la clasificación de documentos o recuperación de información, por ejemplo: artículos, preposiciones o conjunciones. Por ejemplo en la oración: “El mensaje de error del compilador”, al eliminar las palabras vacías se obtendría: Mensaje, error, compilador.

3. **Transformación de datos.** En esta etapa se realiza la vectorización del documento de texto, así como la extracción y selección de características.

    - **Representación vectorial de documentos (vectorization)**. Los documentos se transforman en vectores numéricos utilizando alguna técnica de vectorización (se detalla más adelante).

    - **Extracción de características (feature selection).** Se extraen las palabras que son útiles para la investigación (Forman, 2003).

    - **Selección de características (transformation)**. En la selección de características, se eligen las palabras relevantes. El objetivo es disminuir

la dimensionalidad del conjunto de datos eliminando las características que no contribuyen a la tarea a realizar (Forman, 2003).

4. **Análisis de información.** A la información transformada se le pueden aplicar métodos de minería de textos como recuperación de información, categorización, clasificación y resumen, para así obtener información relevante para la toma de decisiones.

5. **Evaluación.** Por último, se evalúa la precisión exactitud de los resultados obtenidos.

Debido a que en el presente trabajo se utilizará el método de clasificación usando los mensajes de error del compilador, se profundiza en este tema en el siguiente apartado.

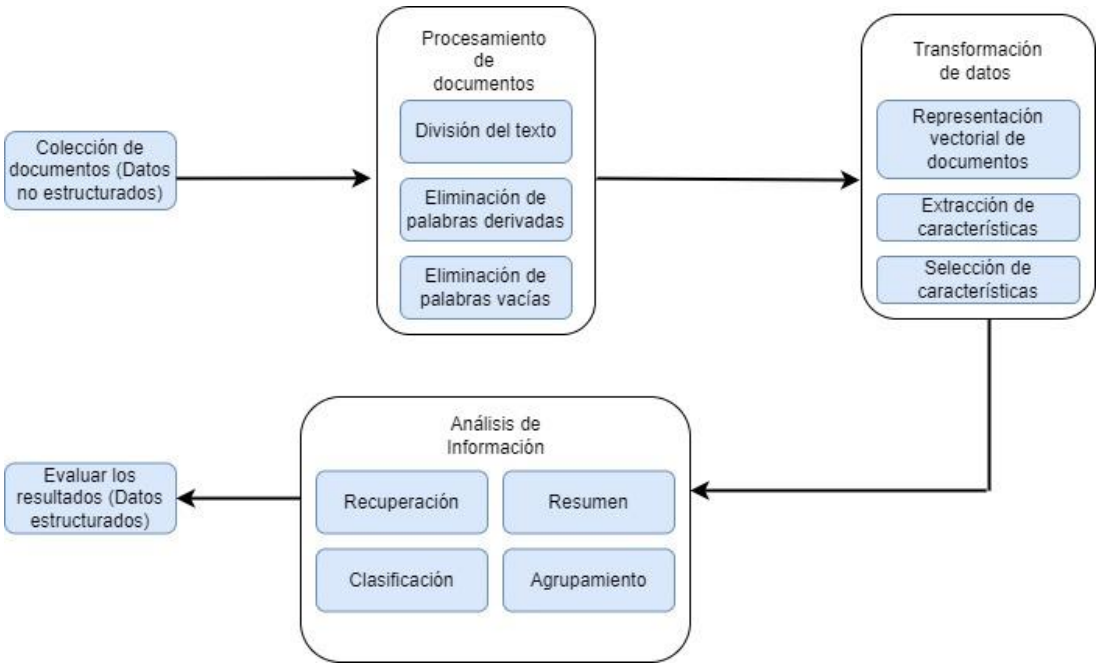


Figura 2.6 Metodología de minería de textos
(Maheswari & Sathiaseelan, 2017)

2.6 Clasificación de textos

La clasificación de textos se refiere al proceso de etiquetar documentos escritos en lenguaje natural con categorías temáticas de un conjunto previamente organizado, es decir, a cada documento (secuencia de palabras) se le asigna una categoría (Dasari & K, 2012). Para que esta asignación sea automática, se pueden utilizar algoritmos de aprendizaje automático, los cuales consisten en sistemas que aprenden de los datos para tomar decisiones con cierto grado de exactitud.

De acuerdo con (Mariñelarena, Errecalde, & Castro, 2017), las tareas predictivas en la minería de textos están basadas en la construcción de un clasificador automático mediante un sistema de aprendizaje supervisado, es decir con documentos etiquetados. Para lo anterior, se comienza por dividir el conjunto de datos de manera aleatoria en dos subconjuntos, un subconjunto de entrenamiento y un subconjunto de prueba, generalmente del 80% y del 20% de los datos respectivamente. Posteriormente en la extracción de características los documentos se vectorizan mediante alguna técnica de vectorización cono bolsa de palabras o el algoritmo TF-IDF. Finalmente, con los datos vectorizados, se entrena el clasificador mediante un proceso inductivo como el aprendizaje bayesiano, las redes neuronales, los árboles de decisión o las máquinas de soporte vectorial, entre otros. Una vez entrenado el clasificador, este puede clasificar documentos no observados previamente (ver Figura 2.7).

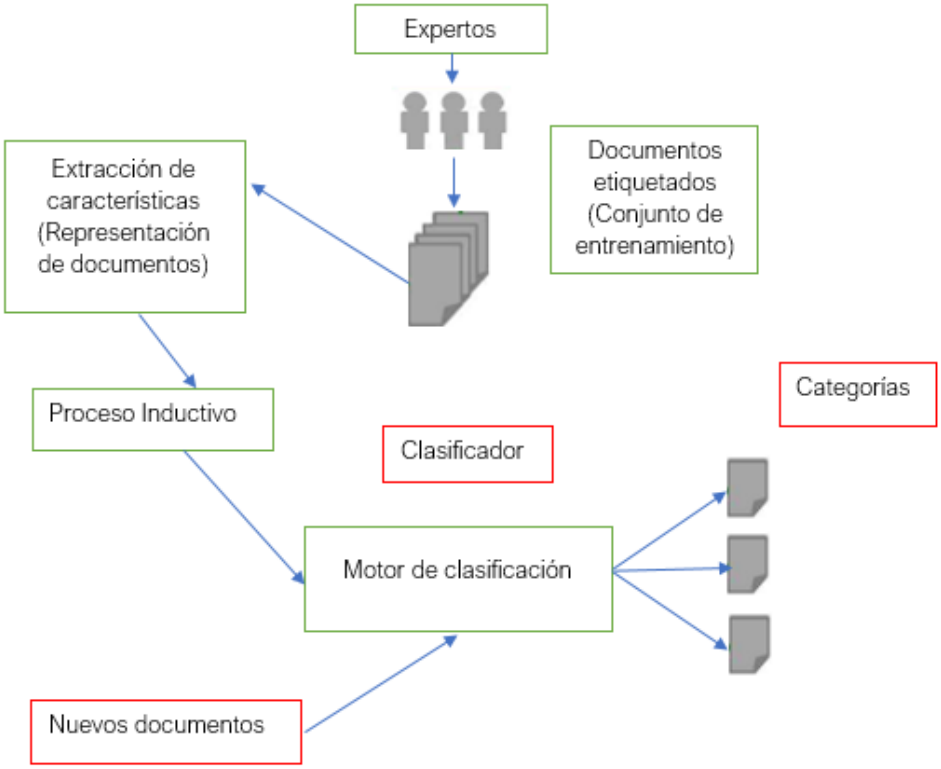


Figura 2.7 Proceso de construcción de un clasificador automático
(Mariñelarena, Errecalde, & Castro, 2017)

Para evaluar el desempeño de clasificador, se utiliza el conjunto de prueba. Si el desempeño del modelo es aceptable, este se puede utilizar para predecir la clase de nuevos documentos, de lo contrario se tiene que ajustar (ver Figura 2.8).

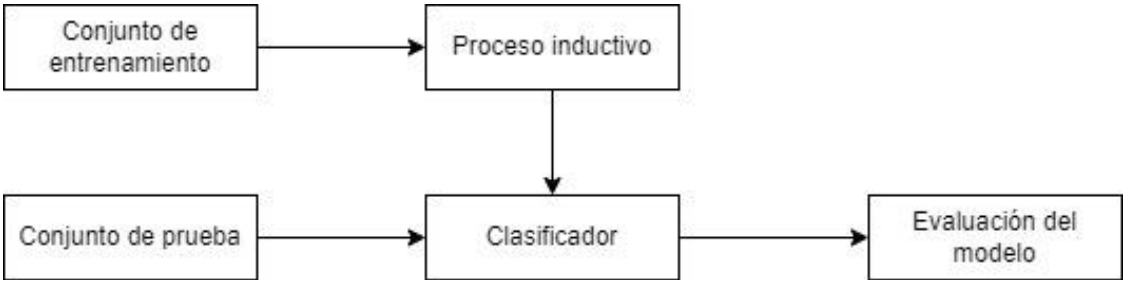


Figura 2.8 Entrenamiento y evaluación de un clasificador
(Liu, 2011)

2.6.1 Vectorización

Para clasificar los textos, es necesario realizar la vectorización, entre los algoritmos ampliamente utilizados se encuentran la bolsa de palabras y el TDF-ID, los cuales se explican a continuación.

2.6.1.1 Bolsa Palabras

El modelo Bolsa de Palabras (Bag of Words) utilizado en el procesamiento del lenguaje natural, representa un documento de texto como un vector numérico, solo teniendo en cuenta si una palabra en específico aparece o no en el documento y descartando el orden de las palabras y su gramática (Soumya & Joseph, 2014) por ejemplo:

Dados dos documentos D1 y D2:

D1: “Este es un documento de texto".
D2: "Cada documento de texto contiene palabras".

De acuerdo con los documentos anteriores, se elabora un vocabulario V:

V = {Este, es, un, documento, de, texto, cada, contiene, palabras}

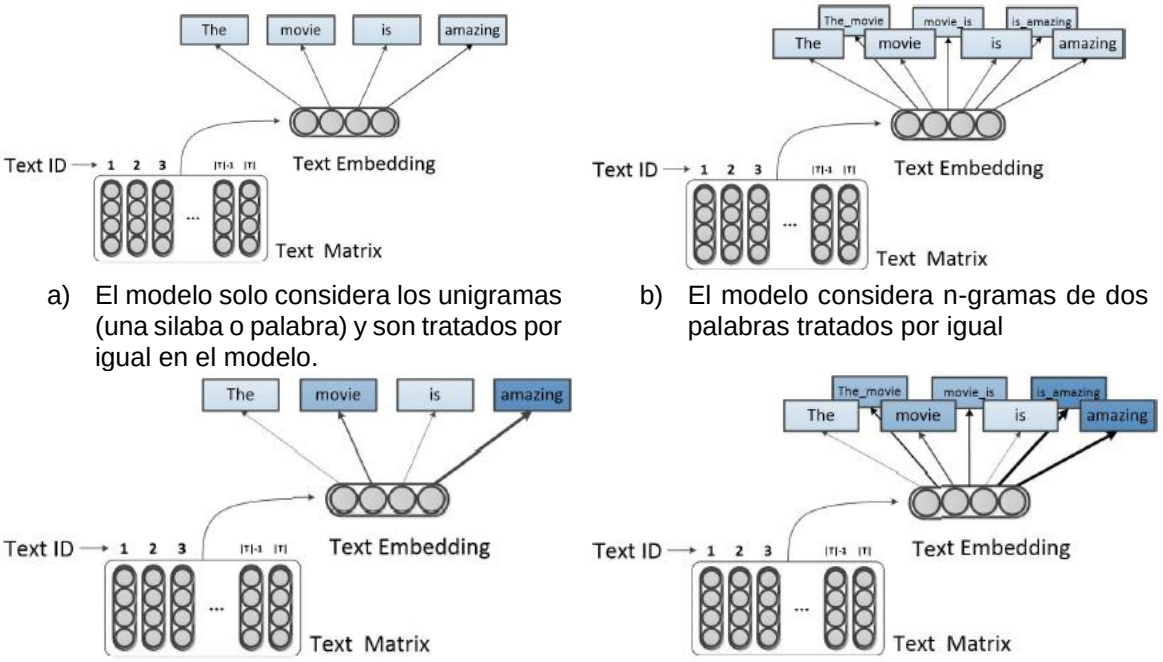
En la Tabla 2.1 se observa el vector construido para cada documento con donde la dimensionalidad *d* corresponde a la cantidad de palabras diferentes en el vocabulario ($d = |V|$). En la tabla mencionada también se observa el conjunto de palabras diferentes en el encabezado, cada fila corresponde a un documento, si la palabra se encuentra en el documento se coloca un 1, si no está se coloca un cero (Raschka, 2017).

Tabla 2.1 Ejemplo de bolsa de palabras para dos documentos

	Este	Es	Un	documento	De	Texto	cada	contiene	palabras
D1	1	1	1	1	1	1	0	0	0
D2	0	0	0	1	1	1	1	1	1
Σ	1	1	1	2	2	2	1	1	1

Para dividir el texto se emplea un proceso llamado “tokenización”, en el que las palabras se dividen en tokens. Los tokens pueden consistir en unigramas, es decir una sola palabra, n-gramas formados por dos palabras en adelante y así sucesivamente (Raschka, 2017) (ver Figura 2.9).

Para que el desempeño del modelo en la clasificación de textos mejore, a los tokens más distintivos de cada documento se les puede otorgar más peso, como se observa en la Figura 2.9 (Bofang, Zhe, Tao, Puwei, & Xiaoyong, 2016).



a) El modelo solo considera los unigramas (una silaba o palabra) y son tratados por igual en el modelo.

b) El modelo considera n-gramas de dos palabras tratados por igual

c) Ponderación de las palabras importantes durante el proceso de entrenamiento, la palabra importante se resalta en azul.

d) Combinación de n-gramas y técnicas de ponderación. EL n-grama importante se resalta en azul.

Figura 2.9 Ponderación en el modelo de Bolsa de Palabras
(Bofang, Zhe, Tao, Puwei, & Xiaoyong, 2016)

2.6.1.2 TF-IDF

TF-IDF es un estadístico que determina las palabras clave asociadas a un documento particular con el fin de asignarle una categoría. Está compuesto por dos conceptos, la frecuencia del término (TF por sus siglas en inglés Term Frequency) y la frecuencia inversa del documento (IDF por sus siglas en inglés Inverse Document Frequency) (Qaiser & Ramsha, 2018).

La frecuencia de término TF se determina mediante la Ecuación 2.1, y se utiliza para medir cuántas veces está presente un término en un documento

$$TF = \frac{Np}{Tp} \tag{2.1}$$

Donde Np es el número de veces que aparece una palabra especifica en el documento y Tp es el número de palabras en el documento.

TF trata las palabras vacías como “El”, o “la” como palabras clave, de las cuales puede haber un gran número en el documento, pero que no ayudan a clasificarlo. Para evitarlo, la frecuencia inversa (IDF) asigna menor importancia a las palabras muy frecuentes y mayor importancia a las palabras menos frecuentes en un conjunto de documentos para ello se utiliza la Ecuación 2.2.

$$IDF = \ln\left(\frac{Nd}{Ndp}\right) \tag{2.2}$$

Donde Nd es el número de documentos y Ndp es el número de documentos que contienen la palabra. Aunque se puede utilizar el logaritmo de base 10, el logaritmo natural se utiliza para evitar que las frecuencias de término que sean muy altas no afecten los resultados. La librería scikit-learn utilizada en el presente proyecto utiliza el logaritmo natural para realizar los cálculos respectivos, aunque hay autores que utilizan el logaritmo de base 10.

Para terminar, $TF - IDF$ se calcula como:

$$TF - IDF = TF * IDF \tag{2.3}$$

destacando que entre mayor sea este valor, más rara es la palabra, ayudando más a la clasificación del documento, esto se puede observar en la Figura 2.10.

Figura 2.10 Relevancia de las palabras bajo la vectorización TF-IDF
(Hamdaoui, 2019)

Por ejemplo, si en un documento hay 100 palabras y la palabra “Moda” se encuentra 10 veces, entonces:

$$TF = \frac{10}{100} = 0.1$$

Ahora bien, si el total de documentos es 10, y la palabra “moda” está presente en 2 de ellos, sin importar el número de veces que la palabra aparece en el documento, la frecuencia inversa sería $IDF = \ln\left(\frac{10}{2}\right) = 0.6987$. Finalmente, $TF - IDF\ es\ 0.1 * 0.6987 = 0.069$

2.6.2 Algoritmos para la clasificación de textos

Entre los algoritmos de aprendizaje automático más utilizados para la clasificación, de textos se encuentran: Decision Tree, Support Vector Machine, Random Forest, Multi-layer Perceptron, y K-Nearest Neighbors, los cuales son algoritmos de aprendizaje supervisado.

En todos los algoritmos se parte de un conjunto de entrenamiento E, $E = \{(d_1, c_1), \dots, (d_n, c_m)\}$ donde se tienen n documentos ($d_1 \dots d_n$) etiquetados, cada uno con la clase que le corresponde, de entre las m clases, $C = \{c_1, c_2, \dots, c_m\}$, para posteriormente generar un clasificador que pueda mapear un nuevo documento con una de las clases predefinidas. El objetivo de la clasificación es entonces tomar una

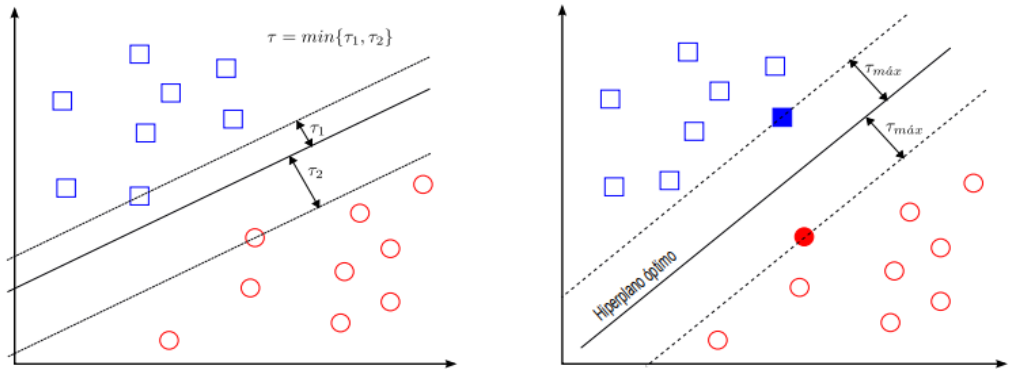
entrada d con sus respectivos atributos $x_1, x_2, x_3 \ldots x_n$ y predecir la clase $c \in C$ que le corresponde.

2.6.2.1 Support Vector Machine

Es un algoritmo de aprendizaje supervisado utilizado para realizar tareas de clasificación y regresión (Mathworks, 2023). A continuación, se describen los aspectos fundamentales del algoritmo.

El propósito del algoritmo es determinar un hiperplano que separe de manera óptima dos o más clases de un conjunto de instancias. Para determinar el hiperplano óptimo se introduce el concepto de margen de separación representado por τ, el cuál es la mínima distancia entre el hiperplano y la instancia más cercana de cada clase, el margen debe tener la misma longitud entre las instancias de las clases.

Estas instancias se denominan vectores de soporte, ya que, si estas observaciones se modifican, habrá cambios en el hiperplano óptimo. Un hiperplano de separación es óptimo si su margen de separación es máximo, si el hiperplano no es óptimo puede afectar la clasificación de instancias cercanas a él. (ver Figura 2.11). Adicionalmente el hiperparámetro C controla el número de observaciones que se pueden salir del margen, si C es infinito no se permite ninguna violación al margen y si C se acerca a cero se van permitiendo más violaciones al margen, es decir se tiene un control sobre el sesgo y la varianza (Amat, 2020).



a) b)

Figura 2.11 Márgenes de hiperplano de separación τ. a) No óptimo, b) Óptimo (Carmona, 2014)

Support Vector Machine utiliza el método de núcleo o Kernel para resolver problemas de regresión y clasificación, tomando el conjunto de datos y transformándolo en uno de mayor dimensión.

El método de kernel es un algoritmo que utilizan los clasificadores lineales para resolver problemas no lineales, es decir transforman datos linealmente no

separables en datos linealmente separables. En la Figura 2.12 se observa a la izquierda un conjunto de datos en 2D, el cual a simple vista no es linealmente separable, el conjunto puede hacerse linealmente separable si se lleva a un espacio 3D (Amat, 2020).

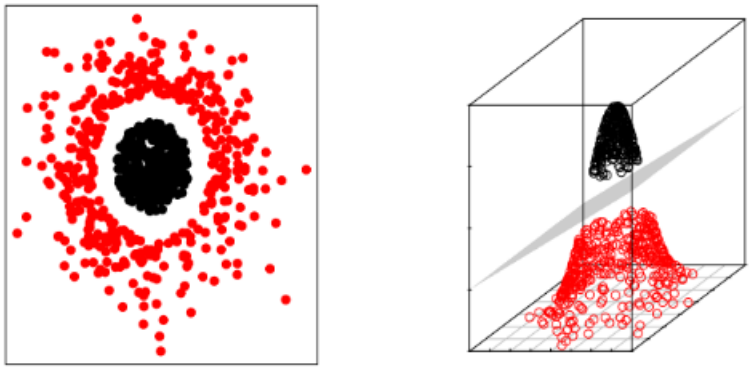


Figura 2.12 El método Kernel
(Amat, 2020)

Se han desarrollado diferentes funciones de Kernel, las cuales se pueden adaptar en mayor o menor medida al conjunto de observaciones. Las funciones kernel más utilizadas son la lineal, radial basis function (RFB), polinomial y sigmoide (ver Figura 2.13)

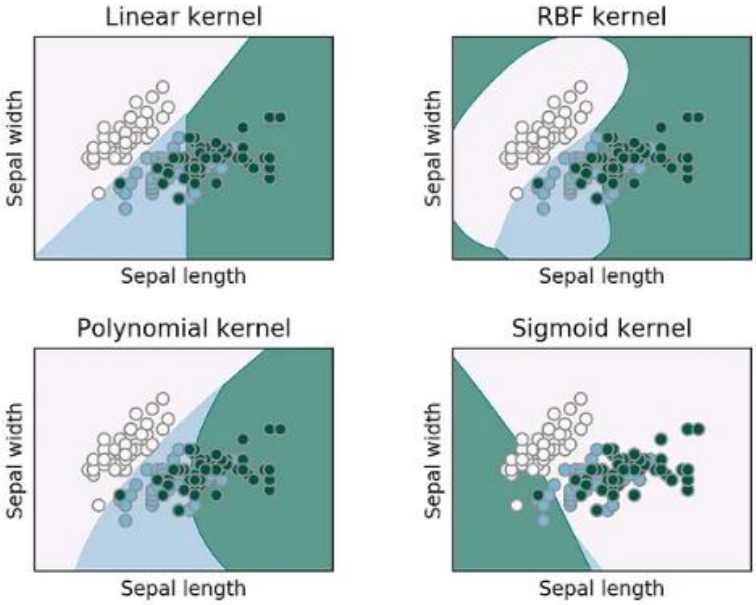


Figura 2.13 Funciones Kernel
(Hucker, 2020)

2.6.2.2 Decision Tree

Un árbol de decisión (Decision Tree) es una estructura que representa la información necesaria para clasificar una instancia, para ello, el árbol debe recorrerse de la raíz a las hojas. En cada nodo, se realiza una prueba sobre un atributo, y se elige la rama correspondiente al valor que presenta la instancia para ese atributo particular. Finalmente, la hoja en la que termine tendrá la clase que le corresponde (Mitchell, 1997). En la Figura 2.14 se ilustra la estructura general de un árbol de decisión.

En general, los árboles de decisión representan una disyunción de conjunciones de restricciones sobre los atributos de las instancias. Cada camino desde la raíz del árbol hasta una hoja corresponde a una conjunción de pruebas de atributos y que tiene una clase dada.

Las características de un árbol de decisión son las siguientes (Mitchell, 1997):

- Las instancias se describen mediante un conjunto fijo de atributos y sus valores
- La función objetivo tiene valores de salida discretos
- Los árboles de decisión pueden aprender de expresiones disyuntivas
- Los atributos de los datos de entrenamiento pueden tener valores desconocidos
- Del árbol final pueden extraerse un conjunto de reglas
- Es recursivo y utiliza la entropía

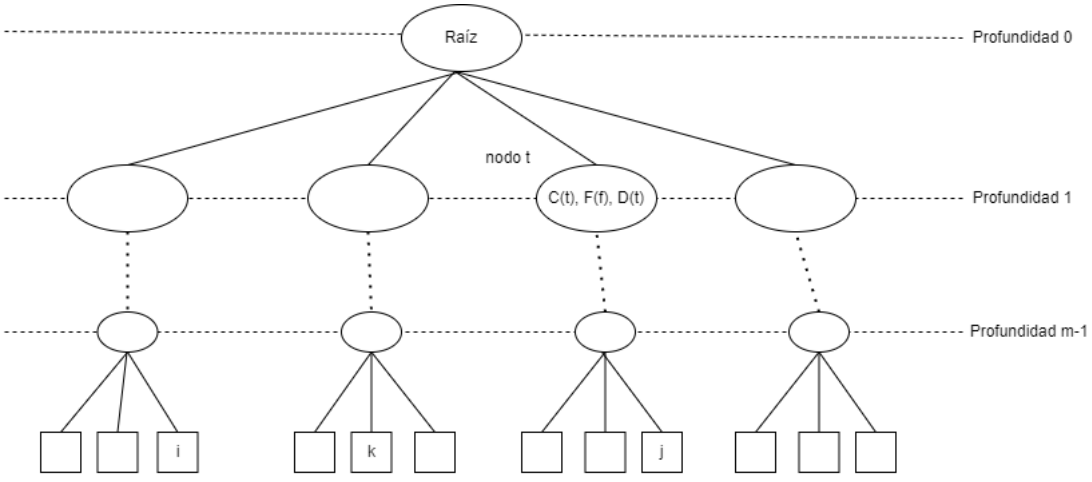


Figura 2.14 Estructura general de un árbol de decisión
(Safavian & Landgrebe, 1991)

El algoritmo ID3 desarrollado por J. Ross Quinlan en 1983, construye árboles de decisión de arriba hacia abajo, seleccionando en cada nodo el atributo que mejor clasifica las instancias de entrenamiento. Este proceso se realiza hasta que el árbol clasifique todas las instancias de entrenamiento (Mitchell, 1997).

Para seleccionar el mejor atributo, el algoritmo ID3 utiliza los términos entropía y la ganancia de información. La entropía, la cual define la impureza de un conjunto de instancias, para n conjuntos de instancias, se define como:

$$Entropía(S) = \sum_{1}^{n} -p_i log_2 p_i \qquad (2.4)$$

En caso de que todas las instancias sean de una misma clase, la entropía seria cero, y cuando las instancias tengan el mismo número de elementos, la entropía es 1. Para medir la efectividad de un atributo para clasificar un conjunto de instancias se utiliza la ganancia definida como la reducción de la entropía después de una clasificación dada por un atributo. La ganancia de un atributo A con respecto a un conjunto de instancias se define como:

$$Ganancia(S, A) = Entropia(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{S} Entropía(s_v) \qquad (2.5)$$

Donde $Valores(A)$ es el conjunto de valores del atributo(A), S_v son los subconjuntos particionados por el atributo A.

2.2.6.3 K-Nearest Neighbors

El algoritmo k vecinos más cercanos o KNN (por sus siglas en inglés K- Nearest Neighbors) es un clasificador de aprendizaje automático, que se basa en el cálculo de la distancia de la instancia a clasificar a las instancias ya clasificadas (IBM, 2015). Para realizar la clasificación, primero se define K cómo el número de vecinos a utilizar, generalmente se recomienda un número impar de vecinos. Posteriormente, se calculan las distancias y se eligen las K más cercanas. Finalmente, se elige la clase mayoritaria de esos K vecinos y se le asigna a la nueva instancia, este ejemplo se puede observar en la Figura 2.15 (IBM, 2015). La elección del número de vecinos determina la exactitud del modelo, así por ejemplo si se elige un numero grande de vecinos, esto puede disminuir la varianza causada por patrones atípicos, pero puede no tomar en cuenta patrones pequeños, por el contrario, si se elige un solo vecino, puede ignorar patrones atípicos, por lo que la elección de los vecinos debe ser un valor no muy grande ni muy pequeño.

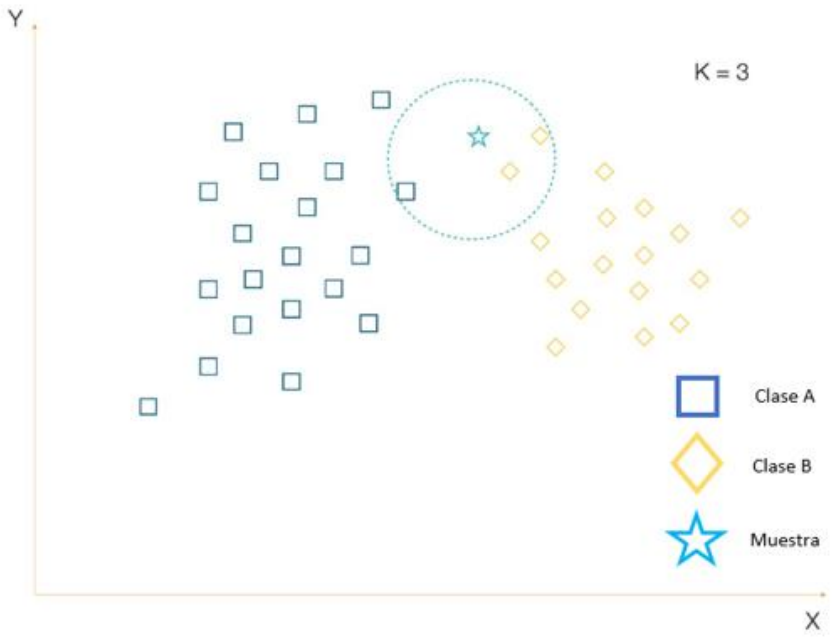


Figura 2.15 Clasificación de una muestra con el algoritmo KNN
(González L. , 2019)

Algunas ventajas de utilizar este algoritmo son principalmente que es no paramétrico, es decir que no requiere ninguna suposición sobre la distribución de datos, además el algoritmo se ajusta fácilmente al agregar nuevos datos y como hiperparámetros solo requiere el número de vecinos con el que se va a trabajar. Entre las desventajas, requiere una gran cantidad de memoria y no funciona muy bien con datos de alta dimensionalidad (IBM, 2015).

Entre las distancias más utilizadas para calcular que puntos están más cerca de un punto de consulta y otros puntos de datos, se encuentra el conjunto de distancias de Minkowski, las cuales son la distancia Euclidiana, Manhattan y Chebyshev (ver Figura 2.16). La distancia euclidiana (p=2) mide en línea recta la distancia entre el punto de consulta y el punto que se mide (distancia más corta), la distancia manhattan (p=1) lo hace través de una cuadrícula como bloques de una ciudad, y la distancia Chebishev (p=infinito) donde la distancia entre dos puntos representados por vectores, es la mayor de sus diferencias sobre el espacio bidimensional (Fu & Yang, 2021).

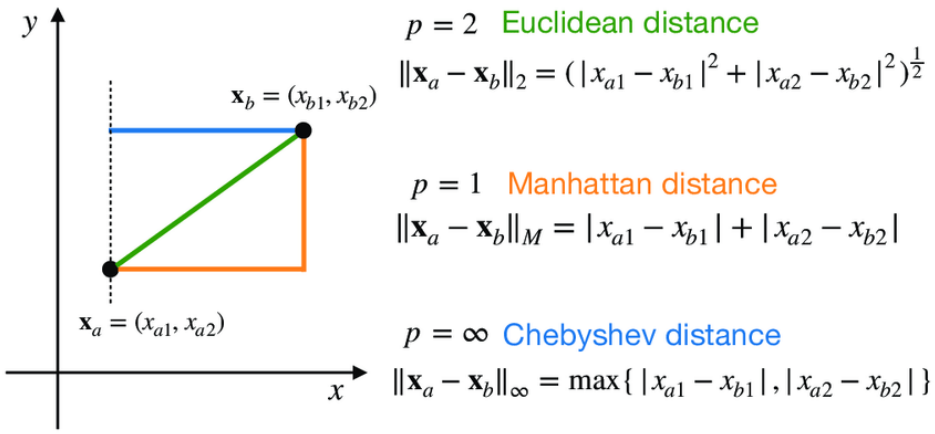


Figura 2.16 Distancias Minkowski, euclidiana, Manhattan y de Chebyshev
(Fu & Yang, 2021)

2.6.2.4 Random Forest

El algoritmo de clasificación Random Forest, está integrado por un conjunto de clasificadores de árboles decisión independientes, generados a partir de subconjuntos aleatorios del conjunto de entrenamiento. Para clasificar un nuevo documento se elige la predicción más común de cada árbol de decisión, es decir un proceso de votación (Parmar, Katariya, & Patel, 2018). La Figura 2.17 describe el proceso anterior. Para realizar una selección de atributos en los árboles de decisión, se utiliza el criterio de ganancia de información (Quinlan 1993) y el índice de Gini (Breiman et al. 1984).

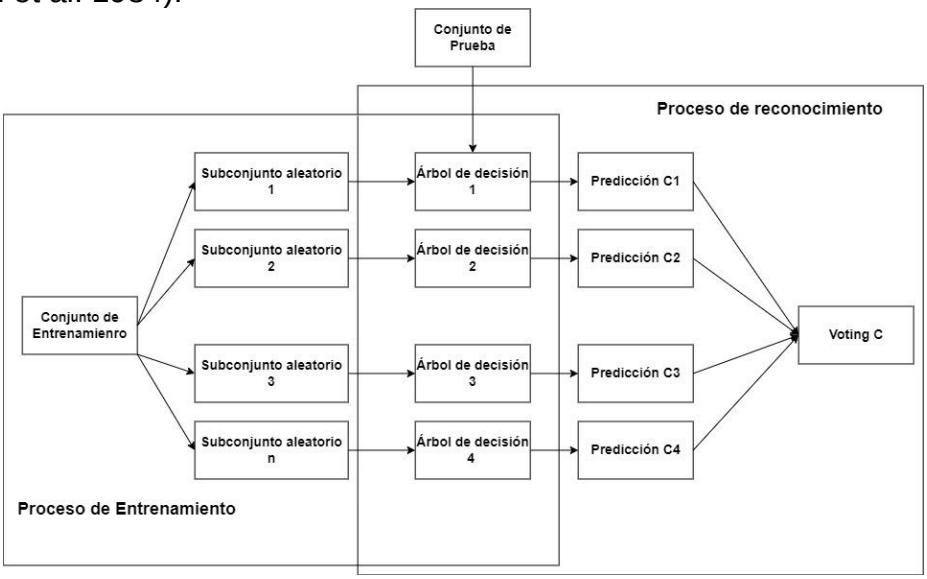


Figura 2.17 Marco conceptual del clasificador de bosque aleatorio
(Parmar, Katariya, & Patel, 2018)

2.6.2.5 Redes Neuronales Artificiales

De acuerdo con (Tablada & Torres, 2009), una red neuronal artificial es un modelo matemático basado en el funcionamiento de las neuronas del cerebro humano, la cual puede realizar tareas como el procesamiento de información, reconocimiento de patrones, clasificación entre otras. Una red neuronal artificial consta de los siguientes elementos (ver Figura 2.18): Entradas x e y, que representan la entrada de datos, los pesos $w1$ y $w2$ de cada entrada, el término aditivo b, la función de activación f, seleccionada de acuerdo con la tarea a realizar, la salida z, representa la salida de la función y está dada por $z = f(w_1 x + w_2 y + b)$.

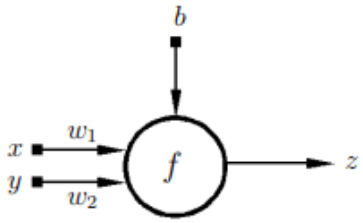


Figura 2.18 Modelo de McCulloch-Pitts para una neurona artificial
(Tablada & Torres, 2009)

Un perceptrón es un modelo neuronal utilizado para la clasificación binaria y cuyos elementos se muestran en la Figura 2.19.

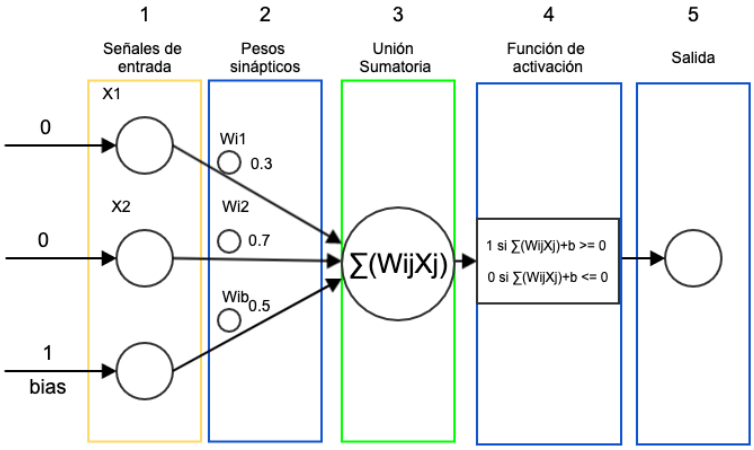


Figura 2.19 Perceptrón
(García U. , 2020)

Una red neuronal artificial resulta de la agrupación en capas de un conjunto de neuronas, de manera que la salida de una neurona es la entrada para la siguiente, a excepción de la capa de entrada y de salida. A una red neuronal también se le llama perceptrón multicapa (Multi-layer perceptron) (ver Figura 2.20) y el cual es utilizado para problemas de clasificación multiclase.

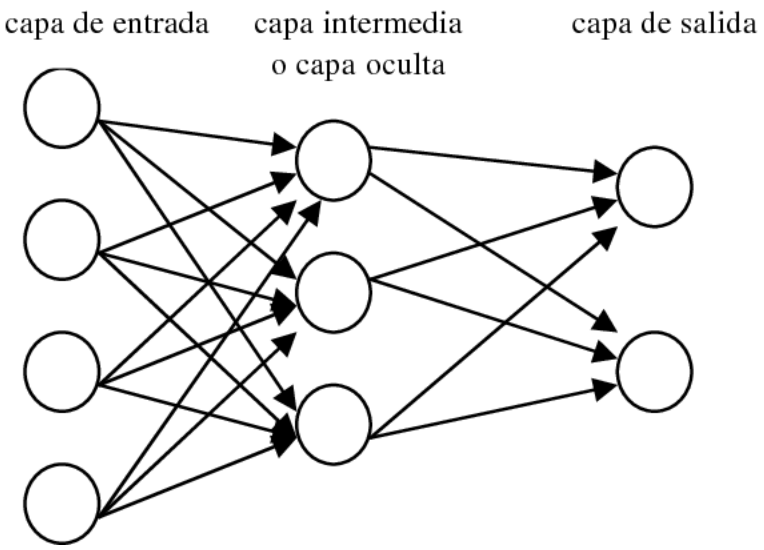


Figura 2.20 Multilayer perceptron

2.7 Métricas de evaluación

Para evaluar el desempeño de un modelo predictivo se utilizan métricas de evaluación tales como: Accuracy, Precision, Recall y F1 (Campos Mocholi, 2021). Tales métricas consideran los términos verdadero positivo (VP), en el que la clase real y la predicha son positivas; verdadero negativo (VN), donde la clase real y la predicha son negativas; falso positivo (FP), cuando la clase real es negativa y la predicha es positiva y falso negativo (FN), cuando la clase real es positiva y la predicha es negativa.

2.7.1 Exactitud

La exactitud o accuracy es la proporción de predicciones clasificadas correctamente. Se calcula dividiendo el número de predicciones correctas entre el total de predicciones realizadas. Es útil cuando las clases del conjunto de datos está balanceado.

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

2.7.2 Precisión

Es la proporción de verdaderos positivos para una clase.

$$Presición = \frac{VP}{VP + FP}$$

2.7.3 Recall

El recall es la proporción de verdaderos positivos entre el número de muestras relevantes positivas, es decir, los verdaderos y falsos negativos.

$$Recall = \frac{VP}{VN + FN}$$

2.7.4 F1 Score

El valor F1 combina la precisión y el recall en un sólo valor.

$$F1 = 2 \cdot \frac{precisión \cdot recall}{precisión + recall}$$

2.7.5 Matriz de confusión

La matriz de confusión sirve para evaluar el desempeño del modelo de clasificación mostrando el número de predicciones correctas e incorrectas para cada clase. Además, se puede observar con claridad que clase es confundida con otra (Campos Mocholi, 2021).

En la matriz de confusión el eje horizontal representa la clase real, el vertical representa la predicción del modelo, los valores de la diagonal son los verdaderos positivos, los valores fuera de la diagonal los falsos negativos y positivos (ver Figura 2.21).

	Clase predecida	
	Positive	Negative
Clase actual	VP	FN
	FP	VN

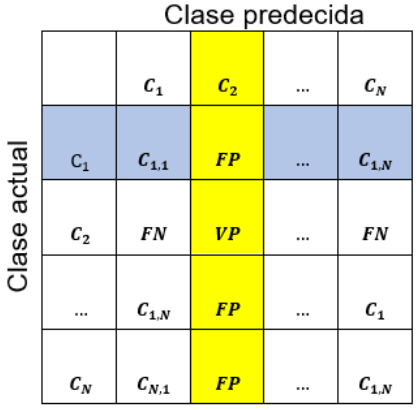


Figura 2.21 Ejemplos de matrices de confusión. (a) Matriz de confusión para una clasificación binaria. (b) Matriz de confusión para la clasificación multiclase.
(Markoulidakis, y otros, 2021)

2.8 Estado del Arte

Han sido numerosos los trabajos que abordan como objeto de estudio los mensajes de error del compilador en diversos lenguajes. Por un lado, existen herramientas que reparan o corrigen los errores como TRACER, Dr Repair, MultiFix o Synshine que utilizan redes neuronales artificiales y los mensajes de error del compilador para reparar errores de programas en lenguaje C o C++. Por otro lado, existen herramientas que brindan retroalimentación relacionada con los errores, como GrammarGuru, TEGCER, DeepDelta y LEARNSKELL. Estas herramientas se presentan a continuación.

TRACER (Targeted Repair of Compilation Errors), es un sistema que repara errores de compilación en lenguaje C. A partir de una línea de código errónea predice una nueva línea correcta utilizando redes neuronales. Dirigido a programadores principiantes (Ahmed, Kumar, Karkare, Kar, & Gulwani, 2018). Los autores consideran que los errores en el tiempo de compilación representan un obstáculo importante para el aprendizaje de los estudiantes de cursos de introducción a la programación. En la Figura 2.22 se muestra un ejemplo de corrección de error. Los mensajes de error del compilador, aunque precisos, están dirigidos a programadores experimentados y en menor medida para principiantes.

En una evaluación de 4500 programas erróneos escritos por estudiantes de un curso de programación de primer año, TRACER recomienda una reparación de errores que coincide exactamente con la esperada por el estudiante para el 68% de los casos, y en el 79.27% de los casos, produce una reparación funcional.

```
1  #include<stdio.h>
2  int main(){
3     int x,x1,d;
4     // ...
5     d=(x-x1)(x-x1);
6     return d;
7  }
```

```
1  #include<stdio.h>
2  int main(){
3     int x,x1,d;
4     // ...
5     d=(x-x1)*(x-x1);
6     return d;
7  }
```

Figura 2.22 Detección y reparación de un error
(Ahmed, Kumar, Karkare, Kar, & Gulwani, 2018)

El sistema Dr. Repair localiza líneas erróneas en programas codificados en lenguaje Java y posteriormente repara. Para ello el sistema aprende a reparar mediante una red neuronal gráfica que toma como datos los símbolos de las líneas erróneas y los mensajes de error del compilador, logrando una tasa de reparación correcta del 68.2% (Yasunaga & Liang, 2020). Este procedimiento se muestra en la Figura 2.23 Y Figura 2.24

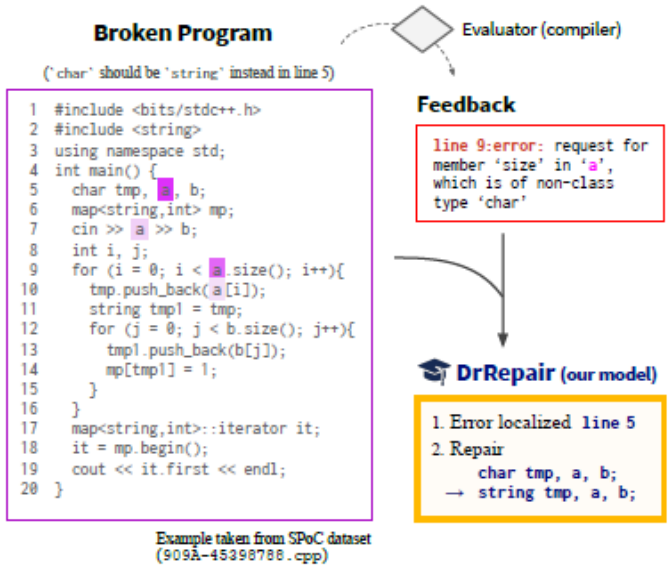


Figura 2.23 Localización y reparación de una línea de código
(Yasunaga & Liang, 2020)

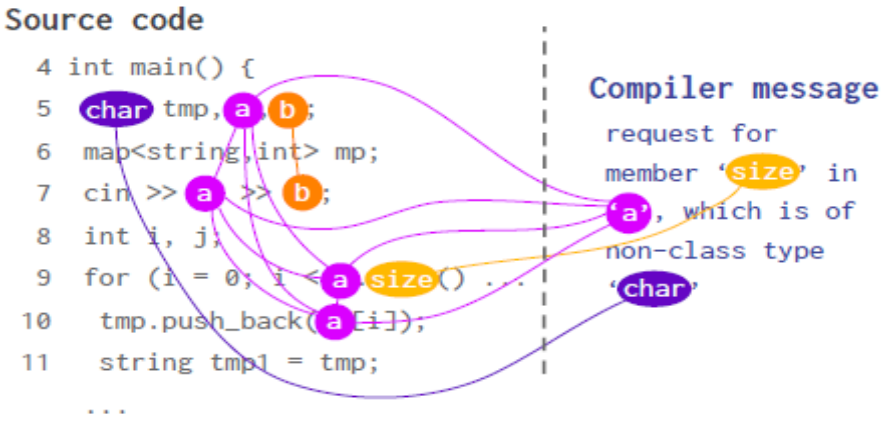


Figura 2.24 Ilustración del gráfico de retroalimentación del programa.
(Yasunaga & Liang, 2020)

MultiFix es un modelo basado en una red neuronal profunda que permite corregir múltiples errores en un programa (Seo, Han, & Ko, 2021). El modelo es entrenado con programas erróneos con su respectiva solución logrando una tasa de reparación correcta del 74.6% (ver Figura 2.25).

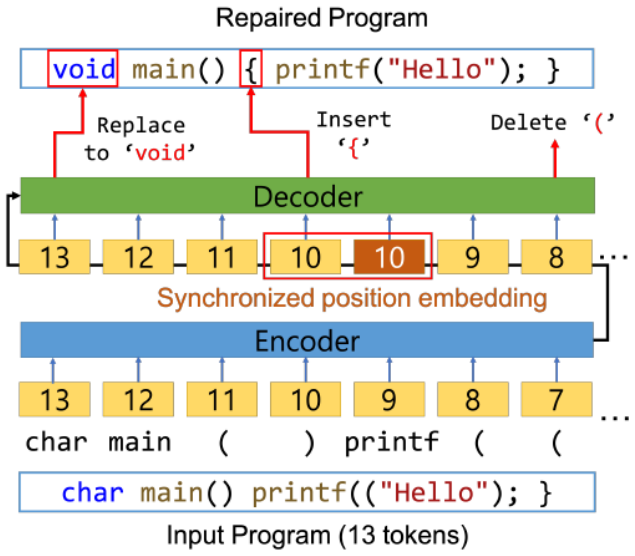


Figura 2.25 Modelo de reparación de programas (MultiFix)
(Seo, Han, & Ko, 2021)

Synshine es una herramienta que utiliza las redes neuronales y los mensajes de error del compilador para reparar errores de sintaxis de programas en lenguaje java (Ahmed, Ledesma, & Devanbu, 2022). En la Figura 2.26 se observa la arquitectura de la herramienta Synshine, la cual recibe un código con error para posteriormente detectar el error de sintaxis, la línea donde se encuentra y finalmente realizar la reparación. La tasa de reparación correcta lograda por Synshine es del 74.89%.

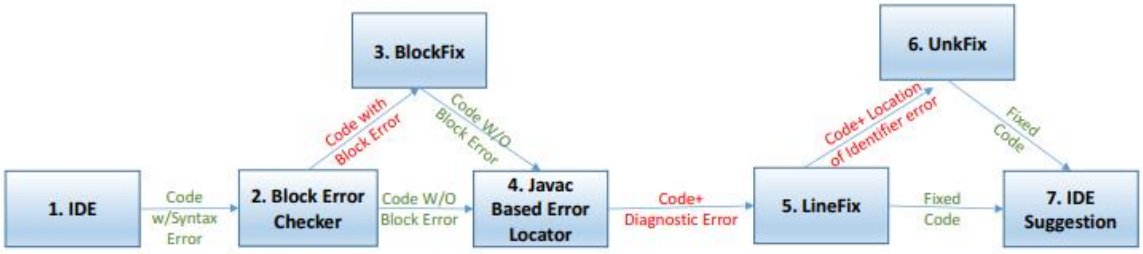


Figura 2.26 Arquitectura de la herramienta Synshine
(Ahmed, Ledesma, & Devanbu, 2022)

CLACER (Classification of Compilation Errors) es un modelo basado en una red neuronal para predecir los errores de compilación proponiendo una clasificación basada en tokens (Wang, y otros, 2022). Para ello utiliza los mensajes de error del compilador (CEMs) y redes neuronales para la clasificación de texto (TextCNN). En la Figura 2.27 se observa el diagrama del modelo, en el que se predice el error en un nuevo programa. El modelo fue entrenado con 29,573 programas fuente, se abstrae la línea y el token con error para posteriormente clasificarlo en base a un conjunto de tipos de error que los autores proponen, alcanzando una tasa de predicción correcta arriba del 90%.

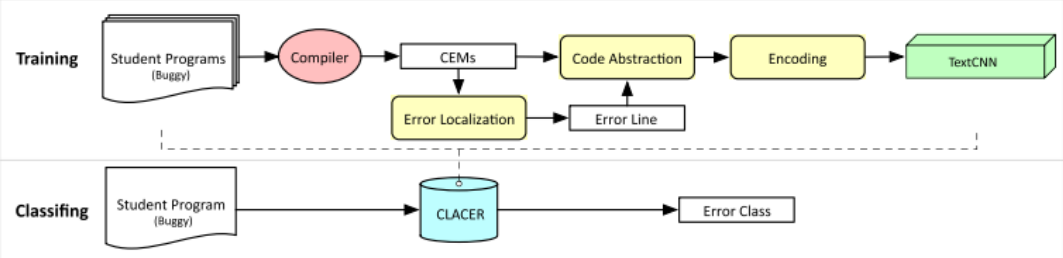


Figura 2.27 Diagrama del modelo CLACER
(Wang, y otros, 2022)

GrammarGuru es una herramienta desarrollada en 2017 basada en una red neuronal de memoria a corto plazo (LSTM short term memory), la cual identifica con precisión un error de sintaxis, su ubicación y produce una sugerencia de solución en JavaScript. Para entrenar la red neuronal se utilizaron repositorios de código JavaScript. Esta herramienta propone una forma de detectar errores y sugerir correcciones comparando el código del usuario con el código correcto que está almacenado en el sistema, alcanzando una tasa de reparación correcta del 35.5% (Santos, Campbell, Hindle, & Amaral, 2017). En la

Figura **2.28** se observa un ejemplo de entrada y salida de la herramienta.

```
1 if (name)) {
2   $form.addClass('highlight');
3   return;
4 }
```

```
insertion.js:5:14: try removing ')'
    if (name)) {
               -
               )
```

Figura 2.28 Entrada y salida de la herramienta GrammarGuru
(Santos, Campbell, Hindle, & Amaral, 2017)

TEGCER es una herramienta de retroalimentación automatizada para estudiantes de programación, que compara los errores del compilador con los prexistentes generados anteriormente por otros estudiantes en lenguaje c (Ahmed, Sindhgatta, Srivastava, & Karkare, 2019). Para lograr lo anterior se entrenó a la red neuronal con al menos 15000 códigos de ejemplo. El objetivo del modelo es proporcionar una retroalimentación alternativa para los estudiantes que encuentran errores de compilación en su programa, logrando una tasa de predicción del 97.68%. En la Figura 2.29 se muestra la interfaz de usuario de TEGCER, del lado derecho el código original y el código del error, y del lado izquierdo la retroalimentación. Para tal retroalimentación se utilizaron correcciones ya realizadas por otros estudiantes.

Figura 2.29 Interfaz de usuario de la herramienta TEGCER.
(Ahmed, Sindhgatta, Srivastava, & Karkare, 2019)

DeepDelta es un sistema que utiliza una red neuronal profunda que sugiere reparaciones para los errores compilación en lenguaje java. Para el entrenamiento se recopilan los errores de compilación junto a su respectiva corrección y con esta información alimenta la red neuronal que genera la traducción automática, alcanzando una tasa de predicción correcta del 90% (Mesbah, Rice, Aftandilian, Johnston, & Glorioso, 2019). En el estudio se detectó que las dos clases más frecuentes de error de compilación son símbolos faltantes y métodos que no coinciden, para los cuales la red neuronal artificial funciona correctamente, sin embargo, no detecta los errores menos frecuentes.

Learnskell (Wu, Campora III, & Chen, 2017), es un sistema depurador de errores desarrollado en Python para el lenguaje Haskell que mediante aprendizaje automático genera mensajes de error del compilador de alta calidad a nivel expresión. Para ello utiliza mensajes de error que contienen texto no estructurado. Para entrenar el sistema se utilizaron 1604 programas con errores generados por estudiantes, alcanzando una precisión de 86%. En la Figura 2.30 se observa el diagrama de flujo para el desarrollo del sistema.

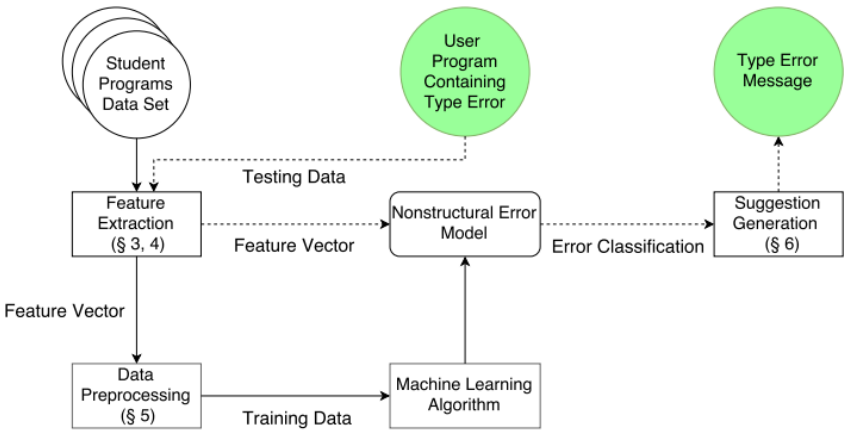


Figura 2.30 Diagrama de flujo del sistema Learnskell
(Wu, Campora III, & Chen, 2017)

De esta forma, para un programa con error, mostrado a la izquierda de la

Program with error	Learnskell Message

`groepeer :: Int -> [Int] -> [[Int]]`
`groepeer n [] = []`
`groepeer n x  = take n x ++ groepeer n drop n x`
	Add an open parenthesis before *drop* and a closing parenthesis after *x* And needs some further changes

Figura 2.31, el sistema Learnskell produce la salida mostrada a la derecha.

Program with error	Learnskell Message

`groepeer :: Int -> [Int] -> [[Int]]`
`groepeer n [] = []`
`groepeer n x  = take n x ++ groepeer n drop n x`
	Add an open parenthesis before *drop* and a closing parenthesis after *x* And needs some further changes

Figura 2.31 Mensaje mejorado como salida del sistema Learnskell
(Wu, Campora III, & Chen, 2017)

Capítulo 3. Desarrollo del modelo

Una de las partes importantes de este trabajo es el desarrollo del modelo predictivo, ya que, de acuerdo con la salida del modelo será la retroalimentación que se dará al estudiante. Para el desarrollo del modelo se utilizó la metodología de minería de textos que consta de cuatro etapas, las cuales son en primer lugar la generación del corpus, el proceso inductivo utilizando algoritmos de aprendizaje automático, la evaluación del modelo a través de la experimentación y finalmente el uso del modelo desarrollado. Estas etapas se describen a continuación (ver Figura 3.1).

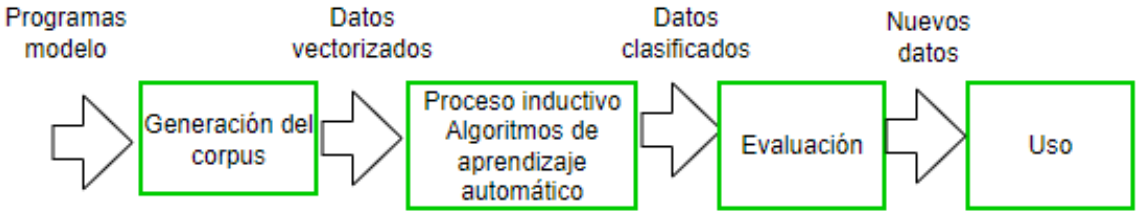


Figura 3.1 Metodología de Minería de Textos Aplicada

3.1 Generación del corpus

El corpus es el conjunto de documentos que se utiliza para entrenar el modelo predictivo, usualmente los documentos son recolectados de fuentes como el archivo de la organización o de internet. En este trabajo, el corpus es generado mediante un proceso de inyección de errores en programas modelo, lo que además permite etiquetar los datos al mismo tiempo que se inyecta el error al programa (ver Figura 3.2).

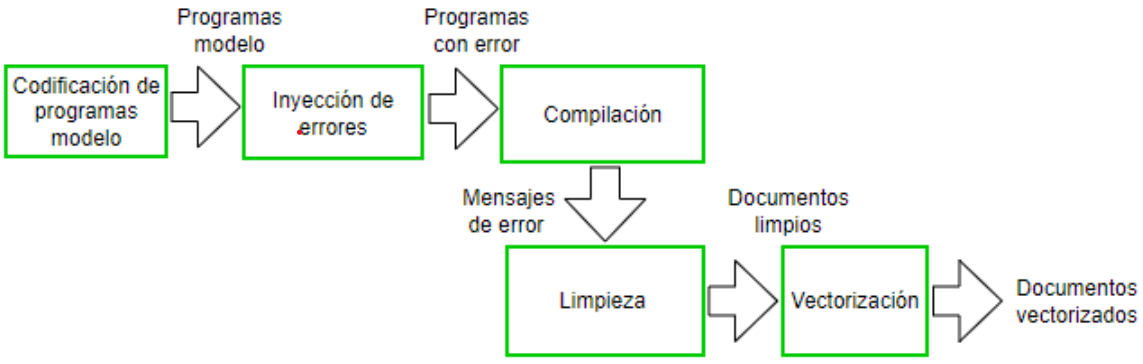


Figura 3.2 Generación del corpus

Como primer paso, se codificaron 35 programas modelo que cumplen con la estructura básica de un programa C (ver Figura 3.3).

```
#include<stdio.h>            Incluye informacion acerca de la biblioteca estándar

main()                       Define la función main que no recibe valores de argumentos

{                            Las proposiciones de main están encerradas entre llaves

printf("hola,mundo\n");      main llama a la función de la biblioteca printf para escribir

}                            esta secuencia de caracteres; \n representa el carácter
                             de la última línea
```

Figura 3.3 Estructura básica de un programa
(Kernighan & Ritchie, 1991)

Los programas se realizaron considerando los conocimientos que el estudiante debe adquirir en la asignatura de fundamentos de programación de la carrera de ingeniería en sistemas y comunicaciones e ingeniería en computación del Centro Universitario UAEM Valle de México, teniendo elementos como las directivas del procesador, la función principal, declaración de variables, funciones de entrada y salida, estructuras de selección y ciclos (ver Figura 3.4).

```
//Arreglos
//Guarda 5 numeros enteros en un arreglo y los imprime
#include <stdio.h>
int main()
{
     int arreglo[5];
     int n;
    for(n=1; n<=5; n++)
    {
        scanf("%i", &arreglo[n]);
    }
    for(n=1; n<=5; n++)
    {
        printf(" i", arreglo[n]);
    }
return 0;
}
```

Figura 3.4 Ejemplo de programa modelo

Como siguiente paso, para la generación de documentos (mensajes de error del compilador), se llevó a cabo el proceso de inyección de errores a los programas modelo, el cual consiste en eliminar intencionalmente elementos específicos que provocan errores de compilación, como son: punto y coma, dos puntos, paréntesis,

variables, numeral(#), símbolos de menor qué y mayor que, llaves, corchetes y letras de palabras clave. El resultado es un conjunto de programas, cada uno de los cuales contiene un error, por ejemplo, para el programa modelo anterior se removieron las comillas de la función `printf` (ver Figura 3.5).

```
#include <stdio.h>
int main()
{
     int arreglo[5];
     int n;
    for(n=1; n<=5; n++)
    {
        scanf("%i", &arreglo[n]);
    }
    for(n=1; n<=5; n++)
    {
        printf(" i , arreglo[n]);
    }
return 0;
}
```

Figura 3.5 Programa con error

La salida del paso anterior son los programas con errores que son compilados para generar los documentos con mensajes de error (ver Figura 3.6).

```
main.cpp:12:16: error: missing terminating " character
   12 |         printf(" i , arreglo[n]);
      |                ^~~~~~~~~~~~~~~~~~
main.cpp: In function ‘int main()’:
main.cpp:13:5: error: expected primary-expression before ‘}’ token
   13 |     }
      |     ^
```

Figura 3.6 Documento generado (Mensaje de error del compilador)

Al inyectar un error a un programa modelo, se genera un documento con el mensaje de error del compilador, el cual se etiqueta de manera simultánea. Tal etiqueta está compuesta por el tipo de error el cual puede ser una función, estructura de selección, ciclos, punto y coma, llaves y el elemento removido. Un ejemplo de esto se puede observar en la Figura 3.7.

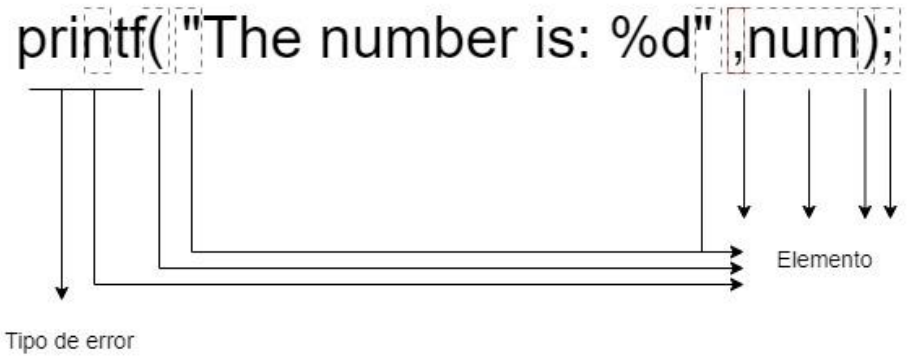


Figura 3.7 Elementos eliminados de la función printf

En el ejemplo mostrado, si se elimina una coma en la función printf la etiqueta sería printFunction_co, es decir, estaría compuesta por la palabra `printf` y colon.

La Tabla 3.1 muestra los elementos que fueron removidos de cada uno de los programas modelo y el número de errores generados, en total 2653. Por ejemplo, para el `ciclo for` se remueve el paréntesis izquierdo, el paréntesis derecho, el contenido entre paréntesis, una letra de la palabra clave "for" señalada por la columna letter, la variable de la expresión del ciclo (inicialización, comprobación, incremento o decremento) señalada por la columna "va" y en la última columna el número de documentos generados, en este caso 107. Para la función `main` se removieron ambos paréntesis y la letra de la palabra clave `main`. En el caso de la función `printf` se removieron ambos paréntesis, las comillas simples, la coma, la variable de salida y una letra de la palabra clave `printf`. De la misma forma para las demás estructuras.

Tabla 3.1 Elementos removidos de los programas modelo

Clase/Elementos	,	(	)	:	#	"	[	]	.	>	<	{	}	;	va	letter	Data
Loop for		•	•											•	•	•	107
Main		•	•													•	108
Printf	•	•	•			•									•	•	653
Scanf	•	•	•			•									•	•	357
Processor directives					•				•	•	•					•	228
Semicolon														•			300
Braces												•	•				316
Switch		•	•	•											•	•	197
Variable declaration						•	•	•									95
Logic expression		•	•												•		191
if structure		•	•												•		24
while cicle															•	•	42
Return sentence															•	•	35
Total																	**2653**

El total de las clases formadas por tipo de error y elemento es 40 y se muestran en la Tabla 3.2. En la primera columna se presenta el nombre de la clase, en la segunda la descripción de los elementos removidos y en la tercera columna el número de documentos generados.

Tabla 3.2 Clases formadas

Clase	Descripción de los elementos eliminados	Número de Documentos
curlyBracket_{_	Llave que abre	158
curlyBracket_}_	Llave que cierra	158
expresion_cf	Contenido entre paréntesis de la función scanf	51
expresion_if	Contenido entre paréntesis del selector if	12
expresion_pf	Contenido entre paréntesis de la función printf	114
expresion_wh	Contenido entre paréntesis del ciclo chile	14
forCicle_pr	Paréntesis del bucle for(abre o cierra)	32
forCicle_sc	Punto y como del bucle for	32
forCicle_va	Variable del ciclo for	43
ifStructure_pr	Paréntesis del selector if(abre o cierra)	24
mainFunction_ln	Letra n de la función main	36
mainFunction_pr	Paréntesis de la función main(abre o cierra)	72
printfFunction_co	Coma de la función printf	42
printfFunction_cs	Comillas de la función printf	228
printfFunction_ln	Letra n de la función printf	114
printfFunction_pr	Paréntesis de la función printf(abre o cierra)	228
printfFunction_va	Variable de la función printf	41
processorDirectives_le	Letra e de la palabra reservada include	76
processorDirectives_li	Línea de la librería	38
processorDirectives_lt	Símbolo Mayor qué o menor que de la librería include	76
processorDirectives_po	Punto de la declaración de la librería include	38
returnSentence_ln	Letra n de la palabra reservada return	35
scanfFunction_co	Coma de la función scanf	51
scanfFunction_cs	Comillas de la función scanf	102
scanfFunction_lf	Letra f de la función scanf	51
scanfFunction_pr	Paréntesis de la función scanf(abre o cierra)	102
scanfFunction_va	Variable de la función scanf	51
semiColon_sc	Punto y coma	300
switchStructure_cn	Dos puntos de la sentencia case de la estructura switch	53
switchStructure_va	Expresión de la estructura switch	10
switchStructure_le	Letra e de la sentencia case	53
switchStructure_lh	Letra h de la estructura switch	10
switchStructure_li	Línea donde aparece la palabra reservada switch	10
switchStructure_lk	Letra k de la palabra reservada break	41
switchStructure_pr	Paréntesis de la estructura switch	20
variableDeclaration_br	Corchetes en la declaración de un arreglo	22
variableDeclaration_cs	Comillas en la declaración de un string	16
variableDeclaration_li	Línea donde se declara una variable	57
whileCicle_le	Letra e de la palabra reservada while	14
whileCicle_pr	Paréntesis del bucle while	28

3.1.2 Limpieza de datos

Después de generar los mensajes de error del compilador, en esta etapa se limpiaron los datos o documentos eliminando caracteres irrelevantes para el modelo, por ejemplo, la palabra error la cual aparece casi en la totalidad de los mensajes. También se eliminó la aparición de nombres de variables ya que son parte del contexto específico del programa modelo y por último la aparición de cadenas como el nombre de programa o contenido dentro paréntesis. Lo anterior se muestra en la Figura 3.8.

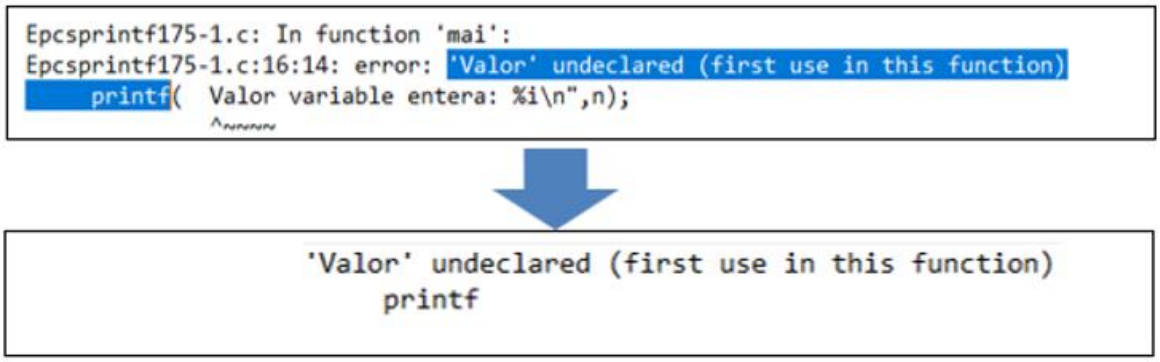


Figura 3.8 Ejemplo de limpieza de datos

3.1.3 Vectorización

Una vez que los documentos se limpiaron, para su vectorización se utilizó el estadístico TF-IDF de la biblioteca sklearn que determina qué tan relevante es una palabra para un documento en una colección, comparando el número de veces que la palabra aparece en el documento con el número de documentos de la colección en los que la palabra aparece.

Los pasos para realizar la vectorización o transformación del texto son los siguientes:

Calcular la frecuencia del término (TF). Por ejemplo, para el documento *"expected expression before '}' token"* se tienen cuatro palabras, "*expected, expression, before* y *token",* como cada una aparece una vez en el documento, la frecuencia del término es 1/4 = 0.25. Cabe aclarar que la biblioteca sklearn para la vectorización TF-IDF no considera caracteres especiales como '}', '!', entre otros. Tampoco se incluyeron estos caracteres en el documento debido a que podían favorecerlo en el proceso de clasificación que se describe más adelante.

Posteriormente se calcula la frecuencia inversa del documento (IDF) utilizando la fórmula:

$$idf(t) = \ln \frac{n}{df(t)} + 1$$

Donde *n* es la cantidad de documentos y *df(t)* es la cantidad de documentos que contienen el término. En el caso del término token que aparece en 736 documentos de 2653 que integran el corpus, el IDF quedaría:

$$IDF = \ln \frac{2653}{736} + 1$$

$$IDF = 2.2822$$

Así al multiplicar por TF que es 0.25, su el TF -IDF para este término seria:

$$TF - IDF_{token} = 2.2953 * 0.25$$
$$TF - IDF_{token} = 0.5705$$

La Tabla 3.3 muestra el cálculo del estadístico TF-IDF para el resto del documento.

Tabla 3.3 Cálculo del estadístico TF-IDF de un documento inicial

	Apariciones en el documento	**TF**	**IDF**	**TF-IDF**
expected	1554	0.25	1.53485882	0.38371471
expression	291	0.25	3.21012309	0.80253077
before	1405	0.25	1.63565377	0.40891344
token	736	0.25	2.28221624	0.57055406

Después de calcular el término TF-IDF, se normaliza para así evitar inconsistencias entre los términos y el número de documentos. Para ello se utilizará la fórmula:

$$v_{norm} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}$$
$$Donde\ v\ es\ el\ estadístico\ TF - IDF\ para\ cada\ documento$$

La Tabla 3.4 muestra el resultado de la normalización.

Tabla 3.4 Normalización del vector TF-IDF

	TF-IDF	v^2	$\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$	Norm
expected	0.38371471	0.14723698	1.137493859	0.33733343
expression	0.80253077	0.64405564	1.137493859	0.70552537
before	0.40891344	0.1672102	1.137493859	0.35948629
token	0.57055406	0.32553193	1.137493859	0.50158869

De esta forma, la transformación del mensaje de error se observa en la Tabla 3.5.

Tabla 3.5 Transformación del mensaje de error

Documento	expected	expression	before	token
expected expression before '}' token	0.33733343	0.70552537	0.35948629	0.50158869

Una vez llevado a cabo el proceso de vectorización de los documentos, se conformó una matriz de 2653 vectores con 147 características, teniendo así el corpus que se utilizará para entrenar y evaluar el modelo.

3.2 Proceso Inductivo

Luego de obtener el corpus, se llevó a cabo el entrenamiento de los clasificadores o modelos predictivos utilizando el 80% de los documentos (2122 vectores). Para ello se utilizaron clasificadores con la biblioteca gratuita de Python SkitLearn, los cuales fueron: Decision Tree, Support Vector Machine, Random Forest, Multi-layer Perceptron y K-Nearest Neighbors.

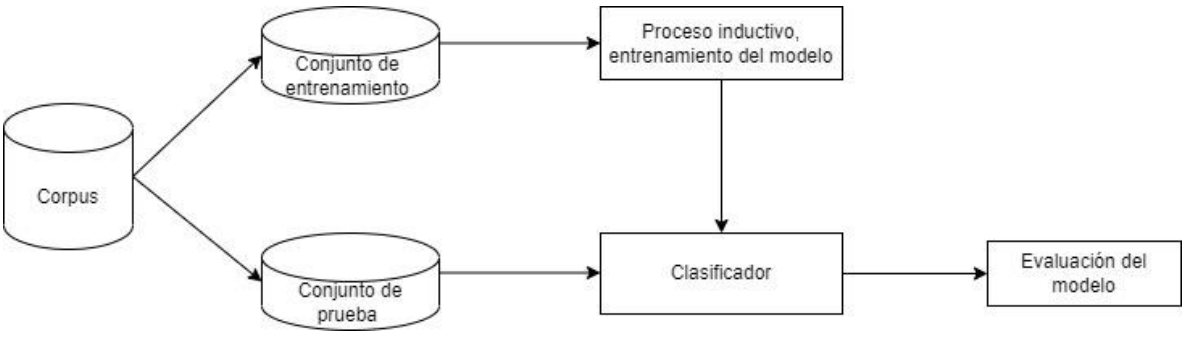


Figura 3.9 Proceso inductivo

Para el entrenamiento de los clasificadores, se utilizó, por un lado, la configuración predeterminada de hiperparámetros (variables de configuración externa para administrar el entrenamiento del modelo), y por otro lado se configuraron sus hiperparametros usando la implementación Random Search de la biblioteca scikit-learn.

Una vez entrenados, los clasificadores se evaluaron con el conjunto de prueba integrado con el 20% de los documentos (531 vectores), ver Figura 3.9. Con el fin de realizar una comparación justa, los clasificadores se entrenaron y evaluaron con el mismo conjunto de documentos.

Algunos de los hiperparámetros configurados para cada clasificador se presentan a continuación.

Decision Tree:

- **criterion** {gini, entropy, log_loss}, predeterminado=gini. Es la función que mide la calidad de una división y puede admitir el criterio para la impureza de gini, log_loss y entropía para la ganancia de información.
- **máx_deph**, predeterminado = Ninguno. Define la profundidad máxima del árbol. Si se elige ninguno, los nodos se expanden hasta que todas las hojas sean puras.
- **min_samples_leaf** {int o flotante, predeterminado = 1}. El número mínimo de muestras necesarias para estar en un nodo de hoja.

Random Forest:

- **min_samples_leaf** {int o flotante, predeterminado = 1}. Establece el número mínimo de muestras necesarias para estar en un nodo de hoja.
- **n_estimators** {int, predeterminado=100}. Define el número de árboles en el bosque.
- **max_features**{“sqrt”, “log2”, ninguno, int o float}, predeterminado=” sqrt”. Establece la cantidad de características a considerar al buscar la mejor división. Si es por ejemplo sqrt, las máximas características serán definidas por la raíz cuadrada del número de características del corpus.

Support vector machine:

- **c** {float, predeterminado=1.0}. Controla las violaciones al margen de separación del hiperplano. Entre más cercano a cero se defina se permiten más violaciones al margen.

- **kernel** {lineal, poly, rbf, sigmoide}, predeterminado=rbf. Función que especifica el tipo de núcleo que se utilizará en el algoritmo para transformar la dimensión del espacio.
- **gamma** {scale, auto, float}, predeterminado=’scale. Controla el comportamiento del kernel, si se reduce el valor se puede obtener el kernel lineal, si por el contrario se aumenta, el modelo se hace más flexible.

Multilayer perceptrón.

- **hidden_layer_sizes,** predeterminado=100. Define el número de neuronas en las capas ocultas.
- **activation** {identity, logistic, tanh, relu}, predeterminado=relu. Establece la función de activación para la capa oculta.
- **solver** {lbfgs, sgd, adam}, predeterminado=adam. Determina el optimizador para calcular la función de coste.
- **learning_rate** {constant, invscaling, adaptive}, predeterminado=constant. Establece la tasa de aprendizaje para la actualización de los pesos.

K-Nearest Neighbors.

- **n_neighbors,** int, predeterminado=5. Número de vecinos a utilizar para las consultas.
- **weights** {uniform, distance}, predeterminado=uniform. Función de peso utilizada para la predicción, si es uniforme los puntos se ponderan por igual, si se elige distancia, los puntos más cercanos al punto de consulta tendrán mayor influencia que los más alejados.
- **metric,** {Euclidean, minkowsky}, predeterminado=minkowski. Métrica utilizada para calcular la distancia entre dos puntos.

3.3 Evaluación del modelo

En esta sección se presentan los resultados de la evaluación de los modelos con el conjunto de prueba. Dado que se obtuvo mejor exactitud con la configuración de hiperparámetros en comparación con la configuración predefinida, para los modelos entrenados de esta forma, se obtuvieron las métricas de desempeño, las matrices de confusión y se realizaron dos experimentos. En el primer experimento se evaluó el modelo utilizando el conjunto de prueba, y en el segundo experimento utilizando nuevos datos, es decir, utilizando programas con características similares, pero no contenidos en el corpus original, esto con el fin de comparar las predicciones del modelo con otros datos. Además, en ambos experimentos se consideró la votación

por mayoría de los clasificadores (Voting), es decir la predicción más común del conjunto de clasificadores. Después de que los modelos son entrenados (clasificadores), cada modelo predice una clase para cada documento del conjunto de prueba, y finalmente se realiza la votación por mayoría (voting). Así si por ejemplo se obtiene que tres clasificadores predijeron la clase A y dos predijeron una clase B, la clase final o resultante sería la clase A (ver Figura 3.10).

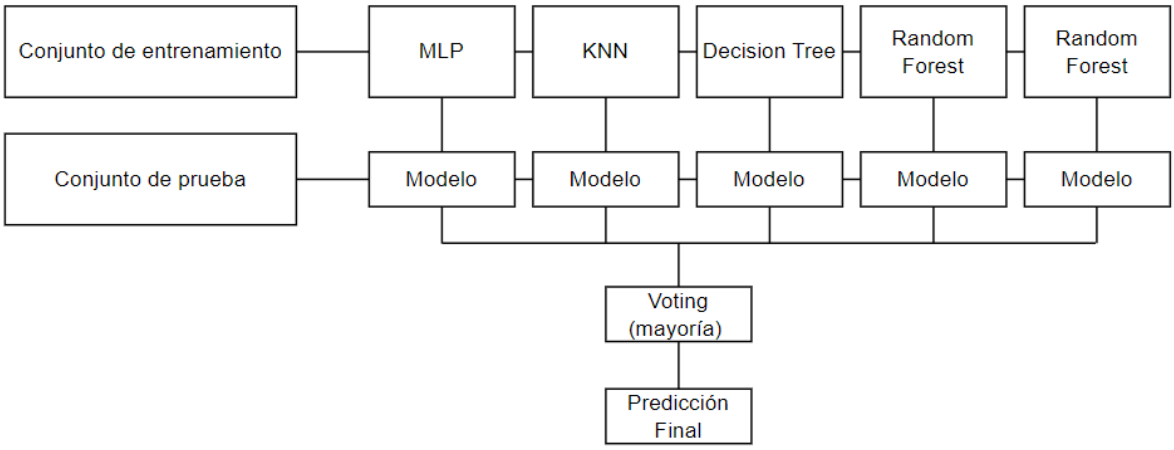


Figura 3.10 Arquitectura de la clasificación por votación (Voting)
(Awan, Saleem, Roberto, & Crespi, 2020)

Los resultados de las métricas de evaluación utilizando el conjunto de prueba y la configuración predefinida de hiperparámetros son mostrados en la Tabla 3.6 y en la cual se observa que los algoritmos Decisión Tree, Random Forest y Multilayer Perceptron alcanzan una exactitud del 90%, Support Vector Machine 87% y K-Nearest Neighbors 83%. Cabe destacar que para tener una mejor apreciación de la exactitud se utilizaron porcentajes con dos números decimales.

Tabla 3.6 Métricas de evaluación de la configuración de parámetros predefinida

	Precision	Recall	F1 Score	Accuracy
Decision Tree	0.94	0.90	0.91	0.90
Support Vector Machine	0.92	0.87	0.89	0.87
K-Nearest Neighbors	0.92	0.83	0.84	0.83
Random Forest	0.95	0.90	0.92	0.90
Multilayer Perceptron	0.94	0.90	0.91	0.90

En la sección 3.2 Proceso Inductivo se describieron los hiperparámetros a utilizar, en esta sección se describen su mejor configuración a utilizar, la cual se muestra en la Tabla 3.7.

Para el algoritmo Decision Tree, se determinó que el número óptimo de muestras requeridas en cada nodo (min_samples_leaf) es dos, la máxima profundidad del árbol (max_depth) óptima es 20, y se encontró que la entropía es el mejor criterio de selección.

Para el algoritmo Support Vector Machine, se determinó que el núcleo (kernel) lineal es el óptimo, el parámetro C que define el tamaño del margen en 1000 y el parámetro gamma que define el margen de decisión entre las observaciones en 1.

El número de vecinos óptimo que se encontró para el algoritmo KNN fue de 15, los pesos de los puntos se establecen de acuerdo con la distancia euclidiana con respecto a un punto de búsqueda.

El número de árboles (n_estimators) para el algoritmo Random Forest se estableció en 150, la entropía como el criterio de selección, la cantidad de hojas (max_leaf_nodes) ilimitada, el número máximo de características se determinó por sqrt (max_features=sqrt(n_features)), la máxima profundidad del árbol hasta que las hojas sean puras y la aleatoriedad del arranque de las muestras utilizadas al construir el árbol (random_state) se estableció en 45.

Para el algoritmo Multi-layer perceptron, el número de neuronas en la capa oculta se determinó en 100, lbfgs (Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm) como el algoritmo de optimización para aprender los pesos y bias, adaptive para el porcentaje de cambio con el que se actualizan los pesos en cada iteración y la función de activación se estableció como logística es decir f(x) = 1 / (1 + exp(-x)).

Tabla 3.7 Configuración de Hyperparámetros

Algorithms	**Best Params**
Decision Tree	min_samples_leaf=2, max_depth=20,criterion=entropy
Support Vector Machine	kernel=linear, gamma=1, C=1000
K-Nearest Neighbors	weights=distance, n_nighbors=15, metric=euclidean
Random Forest	n_estimators=150, max_leaf_nodes=None, max_features=sqrt,
Multi-layer Perceptron	activation=logistic, hidden_layer_sizes=100, learning_rate=adaptiva, solver=lbfgs

3.4 Métricas de evaluación

Las métricas de evaluación utilizadas fueron la exactitud, precisión, recall y F1 Score como se comentó anteriormente en la sección 2.7 Métricas de evaluación. En los resultados de tales métricas se puede observar que los algoritmos Random Forest, Decisión Tree y Multi-layer Perceptrón alcanzaron el mejor desempeño el cual fue de 90% (ver Tabla 3.8).

Tabla 3.8 Resultado de las Métricas de Evaluación

	Precision	**Recall**	**F1 Score**	**Accuracy**
Decision Tree	0.94	0.90	0.90	0.90
Support Vector Machine	0.94	0.89	0.89	0.89
K-Nearest Neighbors	0.89	0.90	0.89	0.90
Random Forest	0.94	0.90	0.90	0.90
Multilayer Perceptron	0.94	0.90	0.90	0.90

Asimismo, se generó la matriz de confusión para cada algoritmo con las 40 clases obtenidas, que muestra en la diagonal las predicciones correctas, y fuera de la diagonal las predicciones incorrectas.

En la Tabla 3.9 se muestra la matriz de confusión del algoritmo Random Forest. En la diagonal de la matriz se observan 480 instancias correctamente clasificadas y en otros casos las 51 instancias incorrectamente clasificadas. Por ejemplo, se puede observar que 10 instancias de la clase mainFunction_pr se clasifican incorrectamente como processorDirectives_li, 14 instancias de la clase scanf_cs incorrectamente clasificadas como scanf_va ,cuatro instancias de la clase printf_cs incorrectamente clasificadas como printf_va, tres instancias de la clase forCicle_pr, y tres de la clase ifStructure_pr incorrectamente clasificadas como semiColon, una instancia de la clase expresión_pf, incorrectamente clasificada como printfFunction_va, una instancia de la clase printfFunction_ln clasificada incorrectamente como switchStructure_le, una instancia de la clase printfFunction_cs, clasificada incorrectamente como variableDeclaration_li, tres instancias de la clase variableDeclaration_br, clasificadas incorrectamente como processorDirectives_li, tres instancias de la clase variableDeclaration_cs incorrectamente clasificadas como processorDirectives_lt, una instancia de la clase switchStructure_lh y una de la clase whileCicle_pr, clasificadas incorrectamente como semiColon_sc.

Tabla 3.9 Matriz de confusión del clasificador Random Forest

	curlyBracket_{_	curlyBracket_}_	expresion_cf	expresion_if	expresion_pf	expresion_wh	forCicle_pr	forCicle_sc	forCicle_va	ifStructure_pr	mainFunction_ln	mainFunction_pr	printfFunction_co	printfFunction_cs	printfFunction_ln	printfFunction_pr	printfFunction_va	processorDirectives_le	processorDirectives_li	processorDirectives_lt	processorDirectives_po	returnSentence_ln	scanfFunction_co	scanfFunction_cs	scanfFunction_lf	scanfFunction_pr	scanfFunction_va	semiColon_sc	switchStructure_cn	switchStructure_va	switchStructure_le	switchStructure_lh	switchStructure_li	switchStructure_lk	switchStructure_pr	variableDeclaration_br	variableDeclaration_cs	variableDeclaration_li	whileCicle_le	whileCicle_pr
curlyBracket_{_	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
curlyBracket_}_	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_cf	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_if	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_pf	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_wh	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_pr	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_sc	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_va	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ifStructure_pr	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_ln	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_pr	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
printfFunction_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
printfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_lt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_po	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
returnSentence_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_lf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
semiColon_sc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	1	0	0	0	0	0	0	0	0	0	0	0	3
switchStructure_cn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0
switchStructure_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
switchStructure_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
switchStructure_lh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0
switchStructure_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
switchStructure_lk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
switchStructure_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
variableDeclaration_br	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
variableDeclaration_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
variableDeclaration_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
whileCicle_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
whileCicle_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	6

En la Tabla 3.10 se muestra la matriz de confusión del algoritmo Decisión Tree con 478 instancias clasificadas correctamente y 53 clasificadas de manera incorrecta. Por ejemplo, se puede observar que 10 instancias de la clase mainFunction_pr se clasifican incorrectamente como processorDirectives_li, 14 instancias de la clase scanf_cs incorrectamente clasificadas como scanf_va ,cuatro instancias de la clase printf_cs incorrectamente clasificadas como printf_va, tres instancias de la clase forCicle_pr, y tres de la clase ifStructure_pr incorrectamente clasificadas como semiColon, una instancia de la clase expresión_pf, incorrectamente clasificada como printfFunction_va y otra de la misma clase incorrectamente clasificada como printfFunction_cs, una instancia de la clase printfFunction_cs, clasificada incorrectamente como variableDeclaration_li, tres instancias de la clase variableDeclaration_br, clasificadas incorrectamente como processorDirectives_li, tres instancias de la clase variableDeclaration_cs incorrectamente clasificadas como processorDirectives_lt, una instancia de la clase forCicle_pr incorrectamente clasificada como forCicle_sc, una instancia de la clase

switchStructure_lh y una de la clase whileCicle_pr, clasificadas incorrectamente como semiColon_sc.

Tabla 3.10 Matriz de confusión del clasificador Decision Tree

	curlyBracket_{_	curlyBracket_}_	expresion_cf	expresion_if	expresion_pf	expresion_wh	forCicle_pr	forCicle_sc	forCicle_va	ifStructure_pr	mainFunction_ln	mainFunction_pr	printfFunction_co	printfFunction_cs	printfFunction_ln	printfFunction_pr	printfFunction_va	processorDirectives_le	processorDirectives_li	processorDirectives_lt	processorDirectives_po	returnSentence_ln	scanfFunction_co	scanfFunction_cs	scanfFunction_lf	scanfFunction_pr	scanfFunction_va	semiColon_sc	switchStructure_cn	switchStructure_va	switchStructure_le	switchStructure_lh	switchStructure_li	switchStructure_lk	switchStructure_pr	variableDeclaration_br	variableDeclaration_cs	variableDeclaration_li	whileCicle_le	whileCicle_pr
curlyBracket_{_	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
curlyBracket_}_	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_cf	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_if	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_pf	0	0	0	0	29	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_wh	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_pr	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_sc	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_va	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ifStructure_pr	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_ln	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_pr	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
printfFunction_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_lt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_po	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
returnSentence_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_lf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0
semiColon_sc	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53	2	0	0	0	0	0	0	0	0	0	0	3
switchStructure_cn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0
switchStructure_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
switchStructure_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
switchStructure_lh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0
switchStructure_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
switchStructure_lk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
switchStructure_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0
variableDeclaration_br	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
variableDeclaration_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
variableDeclaration_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0
whileCicle_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
whileCicle_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	6

En la Tabla 3.11 se muestra la matriz de confusión del algoritmo Multi-layer Perceptrón con 477 instancias clasificadas correctamente y 54 clasificadas incorrectamente. Por ejemplo, se puede observar que 10 instancias de la clase mainFunction_pr se clasifican incorrectamente como processorDirectives_li, 14 instancias de la clase scanf_cs incorrectamente clasificadas como scanf_va ,cuatro instancias de la clase printf_cs incorrectamente clasificadas como printf_va, tres instancias de la clase forCicle_pr, y tres de la clase ifStructure_pr incorrectamente clasificadas como semiColon, una instancia de la clase expresión_pf, incorrectamente clasificada como printfFunction_va, cuatro instancias de la clase printfFunction_cs clasificada incorrectamente como printfFunction_va, una instancia de la clase printfFunction_cs, clasificada incorrectamente como variableDeclaration_li, tres instancias de la clase variableDeclaration_br, clasificadas incorrectamente como processorDirectives_li, tres instancias de la clase variableDeclaration_cs incorrectamente clasificadas como

processorDirectives_lt, una instancia de la clase switchStructure_lh y una de la clase whileCicle_pr, clasificadas incorrectamente como semiColon_sc, tres instancias de la clase semiColon_sc clasificadas incorrectamente como whileCicle_pr y una como switchStructure_lh.

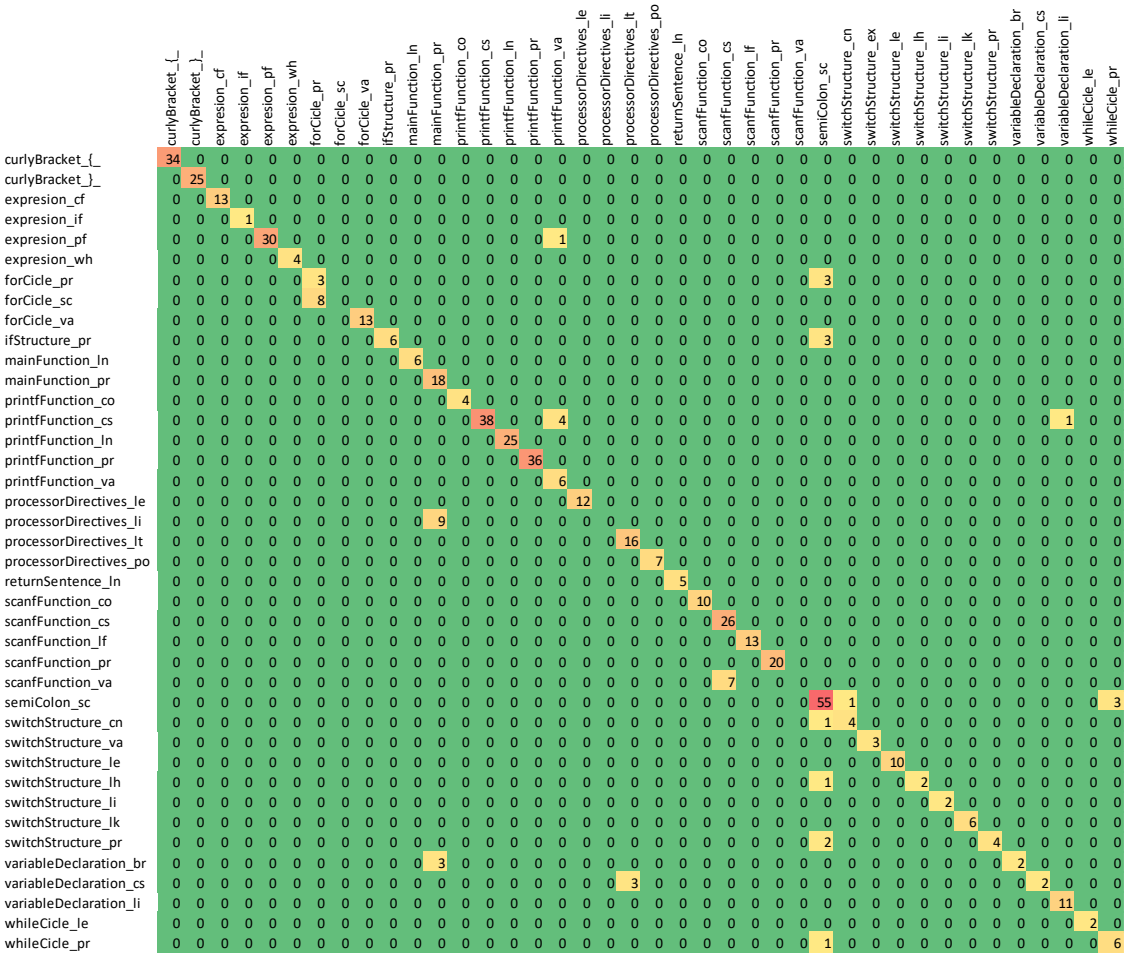
Tabla 3.11 Matriz de confusión del clasificador de Multi-layer perceptron

	curlyBracket_{_	curlyBracket_}_	expresion_cf	expresion_if	expresion_pf	expresion_wh	forCicle_pr	forCicle_sc	forCicle_va	ifStructure_pr	mainFunction_ln	mainFunction_pr	printfFunction_co	printfFunction_cs	printfFunction_ln	printfFunction_pr	printfFunction_va	processorDirectives_le	processorDirectives_li	processorDirectives_lt	processorDirectives_po	returnSentence_ln	scanfFunction_co	scanfFunction_cs	scanfFunction_lf	scanfFunction_pr	scanfFunction_va	semiColon_sc	switchStructure_cn	switchStructure_ex	switchStructure_le	switchStructure_lh	switchStructure_li	switchStructure_lk	switchStructure_pr	variableDeclaration_br	variableDeclaration_cs	variableDeclaration_li	whileCicle_le	whileCicle_pr
curlyBracket_{_	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
curlyBracket_}_	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_cf	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_if	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_pf	0	0	0	1	29	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_wh	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_pr	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_sc	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_va	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ifStructure_pr	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_ln	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_pr	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
printfFunction_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_lt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_po	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
returnSentence_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_lf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0
semiColon_sc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54	1	0	0	1	0	0	0	0	0	0	0	3
switchStructure_cn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0
switchStructure_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
switchStructure_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
switchStructure_lh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0
switchStructure_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
switchStructure_lk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
switchStructure_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0	0	0	0
variableDeclaration_br	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
variableDeclaration_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
variableDeclaration_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0
whileCicle_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
whileCicle_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	6

En la Tabla 3.12 se muestra la matriz de confusión del algoritmo K-Nearest Neighbors con 480 instancias clasificadas incorrectamente y 51 instancias clasificadas correctamente. Se pueden observar que nueve instancias de la clase processorDirectives_li se clasifican incorrectamente como mainFunction_pr, siete instancias de la clase scanfFunction_va se clasifican incorrectamente como scanfFunction_cs, ocho instancias de la clase semiColon son incorrectamente clasificadas como forCicle_pr, cuatro instancias de clase printfFunction_cs clasificadas incorrectamente como printfFunction_va, tres instancias de la clase variableDeclaration_br como mainFunction_pr, tres instancias de la clase variableDeclaration_cs como processorDirectives_lt, tres instancias de la clase forCicle_pr, y tres de la clase ifStructure_pr incorrectamente clasificadas como

semiColon, dos instancias de la clase switchStructure_pr clasificadas incorrectamente como semiColon_sc.

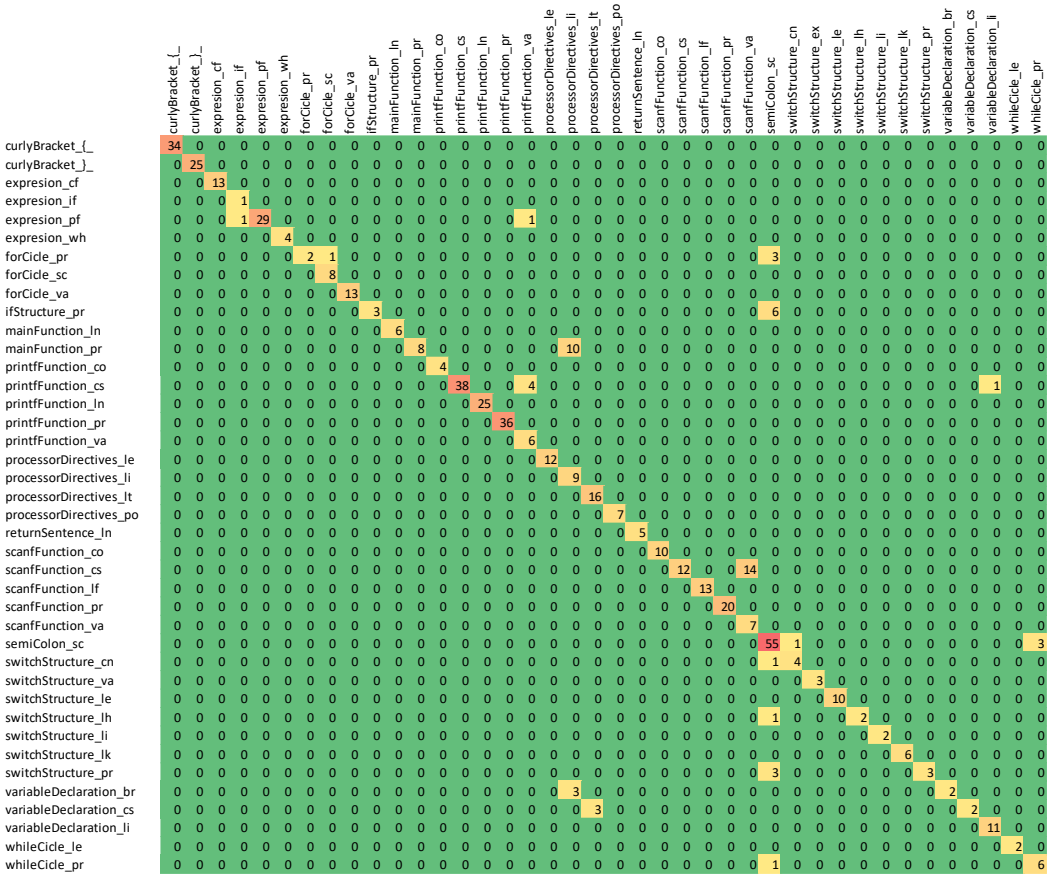
Tabla 3.12 Matriz de confusión para el clasificador K-Nearest Neighbors



	curlyBracket_{_	curlyBracket_}_	expresion_cf	expresion_if	expresion_pf	expresion_wh	forCicle_pr	forCicle_sc	forCicle_va	ifStructure_pr	mainFunction_ln	mainFunction_pr	printfFunction_co	printfFunction_cs	printfFunction_ln	printfFunction_pr	printfFunction_va	processorDirectives_le	processorDirectives_li	processorDirectives_lt	processorDirectives_po	returnSentence_ln	scanfFunction_co	scanfFunction_cs	scanfFunction_lf	scanfFunction_pr	scanfFunction_va	semiColon_sc	switchStructure_cn	switchStructure_ex	switchStructure_le	switchStructure_lh	switchStructure_li	switchStructure_lk	switchStructure_pr	variableDeclaration_br	variableDeclaration_cs	variableDeclaration_li	whileCicle_le	whileCicle_pr
curlyBracket_{_	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
curlyBracket_}_	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_cf	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_if	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_pf	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
expresion_wh	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_pr	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
forCicle_sc	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
forCicle_va	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ifStructure_pr	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
mainFunction_ln	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mainFunction_pr	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_co	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
printfFunction_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
printfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_li	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_lt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
processorDirectives_po	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
returnSentence_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_co	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_lf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scanfFunction_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
semiColon_sc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	1	0	0	0	0	0	0	0	0	0	0	0	3
switchStructure_cn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0
switchStructure_va	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
switchStructure_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
switchStructure_lh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0
switchStructure_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
switchStructure_lk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
switchStructure_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0	0	0	0	0
variableDeclaration_br	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
variableDeclaration_cs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
variableDeclaration_li	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
whileCicle_le	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
whileCicle_pr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	6

En la Tabla 3.13 se muestra la matriz de confusión del algoritmo Support Vector Machine con 474 instancias clasificadas correctamente y 57 instancias clasificadas incorrectamente. Se observa que una instancia de la clase expresión_pr (expresión de la función printf) se clasifica incorrectamente como expresión_if(expresión de la estructura if), seis instancias de la clase ifStructure_pr (paréntesis de la estructura if) se clasifican incorrectamente como semiColon (punto y coma), una instancia de la clase forCicle_pr(paréntesis del ciclo for) es incorrectamente clasificadas como forCicle_sc(punto y coma del ciclo for), una instancia de la clase printfFunction_cs clasificada incorrectamente como variableDeclaration_li(declaración de variable).

Tabla 3.13 Matriz de confusión del clasificador Support Vector Machine



Los resultados del primer experimento son mostrados en la Tabla 3.14, en los que se observa que los clasificadores K-Nearest Neighbors y Random Forest obtuvieron el mayor porcentaje de acierto en las predicciones con 90.39%.

Tabla 3.14 Desempeño de clasificadores con el conjunto de prueba

Clasificador	Conjunto de prueba	
	# Errores	% Aciertos
Decision Tree	53	90.01%
Support Vector Machi	57	89.26%
K-Nearest Neighbors	51	90.39%
Random Forest	51	90.39%
Multilayer Perceptron	54	89.83%
Voting	53	90.01%

En el segundo experimento se evaluaron los clasificadores con 267 nuevos datos, es decir datos que no pertenecían al corpus original, pero con una estructura similar

concluyendo que el algoritmo que tuvo mejor desempeño en este caso fue el Random Forest alcanzando un 91.39% de desempeño (ver Tabla 3.15).

Tabla 3.15 Desempeño de clasificadores con el nuevo conjunto de datos

Clasificador	Nuevo corpus	
	# Errores	% Aciertos
Decision Tree	26	90.26%
Support Vector Machi	26	90.26%
K-Nearest Neighbors	27	89.89%
Random Forest	23	91.39%
Multilayer Perceptron	26	90.26%
Voting	26	90.26%

Debido a que el algoritmo Random Forest es de los que tuvo mejor desempeño en los experimentos, es el que se consideró para el modelo predictivo del sistema final. En la Tabla 3.16 y Tabla 3.17 se muestra un extracto de los documentos con su clase real y la predicción del clasificador Random Forest y el Voting, ambos para el conjunto de prueba. Asimismo, en la Tabla 3.18 y Tabla 3.19 se muestra un extracto de mismos clasificadores para el nuevo conjunto de datos. Las predicciones sombreadas son las que resultaron incorrectas o diferentes a la clase real en cada clasificador.

Tabla 3.16 Predicciones del clasificador Random Forest para el conjunto de prueba

Documento	Clase Real	Predicción
a label can only be part of a statement and a declaration is not a statement	printfFunction_ln	printfFunction_ln
unknown type name 'sca' sca f	scanfFunction_lf	scanfFunction_lf
invalid operands to binary & (have 'char *' and 'int') scanf	scanfFunction_co	scanfFunction_co
too few arguments to function 'printf' printf	expresion_pf	expresion_pf
expected ';' before 'return' return	semiColon_sc	semiColon_sc
expected ':' or '...' before 'printf' case 1 printf	switchStructure_cn	switchStructure_cn
expected ';' before string constant scanf	scanfFunction_pr	scanfFunction_pr
expected ',' or ';' before 'char'	semiColon_sc	semiColon_sc
expected identifier or '(' before 'else'	curlyBracket_{_	curlyBracket_{_
too few arguments to function 'scanf' scanf	expresion_cf	expresion_cf
'retur' undeclared (first use in this function)	returnSentence_ln	returnSentence_ln
expected ')' before '{' token {	whileCicle_pr	semiColon_sc
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected '=', ',', ';', 'asm' or '__attribute__' before ']' token	variableDeclaration_br	processorDirectives_li
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
unknown type name 'pri' pri	printfFunction_ln	switchStructure_le
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
missing terminating " character scanf	scanfFunction_cs	scanfFunction_cs
expected expression before ')' token scanf	scanfFunction_va	scanfFunction_va
a label can only be part of a statement and a declaration is not a statement	printfFunction_ln	printfFunction_ln
too few arguments to function 'printf' printf	expresion_pf	expresion_pf
expected ')' before ';' token scanf	scanfFunction_pr	scanfFunction_pr
expected '=', ',', ';', 'asm' or '__attribute__' before 'char'	semiColon_sc	semiColon_sc

Tabla 3.17 Predicciones del Voting para el conjunto de prueba

Documento	Clase Real	Predicción
a label can only be part of a statement and a declaration is not a statement	printfFunction_ln	printfFunction_ln
unknown type name 'sca' sca f	scanfFunction_lf	scanfFunction_lf
invalid operands to binary & (have 'char *' and 'int') scanf	scanfFunction_co	scanfFunction_co
too few arguments to function 'printf' printf	expresion_pf	expresion_pf
expected ';' before 'return' return	semiColon_sc	semiColon_sc
expected ':' or '...' before 'printf' case 1 printf	switchStructure_cn	switchStructure_cn
expected ';' before string constant scanf	scanfFunction_pr	scanfFunction_pr
expected ',' or ';' before 'char'	semiColon_sc	semiColon_sc
expected ')' before '{' token {	whileCicle_pr	semiColon_sc
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected '=', ',', ';', 'asm' or '__attribute__' before ']' token	variableDeclaration_br	processorDirectives_li
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected '(' before 'opcion' switch switch	switchStructure_pr	semiColon_sc
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected ')' before ';' token printf	printfFunction_pr	printfFunction_pr
expected ';' before 'm' for	forCicle_sc	forCicle_sc
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va

Tabla 3.18 Predicciones del clasificador Random Forest para el nuevo conjunto de datos

Documento	Clase Real	Predicción
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected ')' before 'n' printf	printfFunction_co	printfFunction_co
expected ')' before 'n' printf	printfFunction_co	printfFunction_co
expected ')' before numeric constant printf	printfFunction_co	printfFunction_co
'Introduce' undeclared (first use in this function) printf	printfFunction_cs	variableDeclaration_li
'Introduce' undeclared (first use in this function) printf	printfFunction_cs	variableDeclaration_li
'Las' undeclared (first use in this function) printf	printfFunction_cs	variableDeclaration_li
invalid operands to binary & (have 'char *' and 'int') scanf	scanfFunction_co	scanfFunction_co
invalid operands to binary & (have 'char *' and 'int') scanf	scanfFunction_co	scanfFunction_co
invalid operands to binary & (have 'char *' and 'int') scanf	scanfFunction_co	scanfFunction_co
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected expression before '%' token scanf	scanfFunction_cs	scanfFunction_va
expected ';' before 'break' break;	semiColon_sc	semiColon_sc
expected ';' before '}' token }	semiColon_sc	semiColon_sc
expected ';' before 'scanf' scanf	semiColon_sc	semiColon_sc
expected ';' before 'scanf' scanf	semiColon_sc	semiColon_sc
expected ';' before 'else'	semiColon_sc	curlyBracket_}_
expected ';' before 'else'	semiColon_sc	curlyBracket_}_

Tabla 3.19 Predicciones del Voting para el nuevo conjunto de datos

Documento	Clase Real	Predicción
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
expected ')' before 'printf' printf	ifStructure_pr	semiColon_sc
ld returned 1 exit status	mainFunction_ln	mainFunction_ln
ld returned 1 exit status	mainFunction_ln	mainFunction_ln
ld returned 1 exit status	mainFunction_ln	mainFunction_ln
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected '=', ',', ';', 'asm' or '__attribute__' before ')' token	mainFunction_pr	processorDirectives_li
expected declaration specifiers or '...' before '{' token	mainFunction_pr	mainFunction_pr
expected declaration specifiers or '...' before '{' token	mainFunction_pr	mainFunction_pr
expected declaration specifiers or '...' before '{' token	mainFunction_pr	mainFunction_pr
expected ')' before 'n' printf	printfFunction_co	printfFunction_co
expected ')' before 'n' printf	printfFunction_co	printfFunction_co
expected ')' before numeric constant printf	printfFunction_co	printfFunction_co
expected ')' before numeric constant printf	printfFunction_co	printfFunction_co
expected ')' before 'n1' printf	printfFunction_co	printfFunction_co
expected ')' before 'n2' printf	printfFunction_co	printfFunction_co
expected ')' before 'necesidadesCom' printf	printfFunction_co	printfFunction_co
'la' undeclared (first use in this function) printf	printfFunction_cs	printfFunction_cs
'Ptas' undeclared (first use in this function) printf	printfFunction_cs	variableDeclaration_li
expected expression before '%' token printf	printfFunction_cs	printfFunction_va
expected expression before '%' token printf	printfFunction_cs	printfFunction_va
stray '\303' in program def	printfFunction_cs	printfFunction_cs

Capítulo 4. Desarrollo del prototipo del sistema

En este capítulo se presenta el desarrollo del prototipo del sistema para el uso del modelo predictivo.

4.1 Descripción del sistema

La arquitectura de software en el desarrollo de un sistema, implica elementos que lo componen tales como: el servidor encargado de ofrecer servicios; el cliente que realiza peticiones al servidor para obtener dichos servicios; el diagrama de diseño como representación gráfica para el desarrollo del sistema; la interfaz de usuario como el medio que permite a un usuario de un sistema informático comunicarse con el mismo; la reusabilidad que implica la posibilidad de que un elemento de software pueda ser usado nuevamente sin necesidad de ser reimplementado, simplificando y agilizando el desarrollo de aplicaciones y el usuario, que es la persona o personas que operan o interactúan directamente con el producto (Sommerville, 2011).

La interfaz de usuario permite la interacción del modelo predictivo con el usuario, a fin de mostrar retroalimentación sobre sus errores, para ello el modelo debe estar conformado por los componentes que se ilustran en la Figura 4.1 y se explican a continuación.

Usuario. Estudiante de fundamentos de programación en lenguaje C de nivel superior.

Interfaz de usuario. Canal de comunicación entre el usuario y el modelo cuyas funciones serán las siguientes:

- Iniciar el sistema.
- Ingresar el programa a revisar.
- Visualizar las recomendaciones sobre el error.
- Enviar la retroalimentación sobre la herramienta.

Módulo de conocimiento. Almacenará datos del modelo tales como mensajes de error del compilador y recomendaciones.

Motor de clasificación. Modelo predictivo generado, el cual a cada mensaje de error asigna una categoría particular y con base en ella se provee la retroalimentación.

Retroalimentación. Se trata de la salida o el objetivo del modelo. Una vez que el estudiante ingrese el error de su interés y este haya pasado por el motor de clasificación.

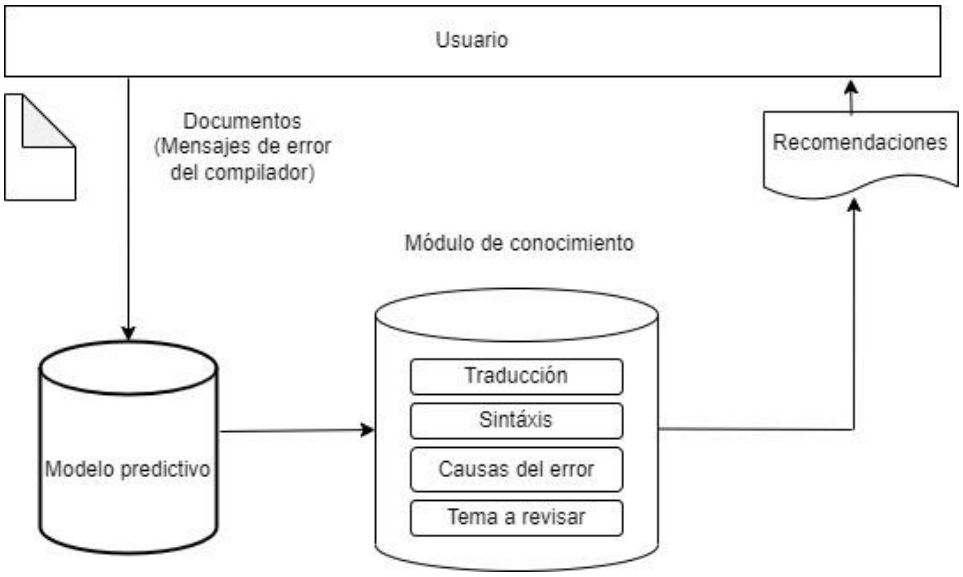


Figura 4.1 Arquitectura del sistema

4.2 Funciones del sistema

La interfaz de usuario tiene la función de cargar el programa con error, ejecutar el modelo predictivo y mostrar la retroalimentación asociada al error. También contiene una encuesta de satisfacción que podrá contestar.

4.3 Características de los usuarios

Los usuarios, son estudiantes con al menos un semestre estudiando programación y lenguajes en el nivel medio superior o superior.

4.4 Suposiciones y dependencia

Dado que el sistema es una aplicación web podrá funcionar en cualquier sistema operativo reciente. Otra ventaja es que es que no dependerá de una conexión a internet ya que se ejecutará en un servidor local, permitiéndole ser un sistema más estable, robusto, menor tiempo de desarrollo y de carga de datos.

4.5 Requisitos Futuros

Es posible que en un futuro al sistema se le agreguen nuevas características como que el usuario pueda elegir mediante un menú el lenguaje de programación a trabajar, personalizar el tipo retroalimentación, mejorar la interfaz de usuario.

4.6 Especificación de Requerimientos

El sistema funcionará como una aplicación web local permitiendo el ingreso de los mensajes de error del compilador por parte del estudiante que seleccionará con un botón de carga. Cuando el sistema capta el programa que contiene error de sintaxis, lo compila, limpia el mensaje de error, lo vectoriza y este nuevo documento es utilizado por el modelo para predecir la clase a la que pertenece, es decir el tipo de error del que se trata. Este proceso se ilustra en la Figura 4.2.

Figura 4.2 Proceso de recomendación

4.7 Interfaz de usuario

La interfaz consta de consta de tres apartados principales, en el primero el usuario cargará el archivo con el programa que tiene error y ejecutará el sistema, en el segundo el sistema mostrará la retroalimentación generada por el modelo predictivo y módulo de conocimiento, y en el tercero aparecerá una encuesta de satisfacción a ser contestada por el usuario. Para la programación de la interfaz se utilizó el lenguaje de programación Python. Los estudiantes elaborarán sus programas en el software libre CODE-BLOCKS o cualquier ambiente de programación en C.

En el apartado izquierdo de la interfaz de usuario se encuentra un botón para cargar el programa con error, un botón para ejecutar el sistema y un botón para limpiar la pantalla y eliminar los datos generados. En el apartado central se mostrará la información generada por el modelo predictivo y en el tercer apartado una encuesta de satisfacción (ver Figura 4.3).

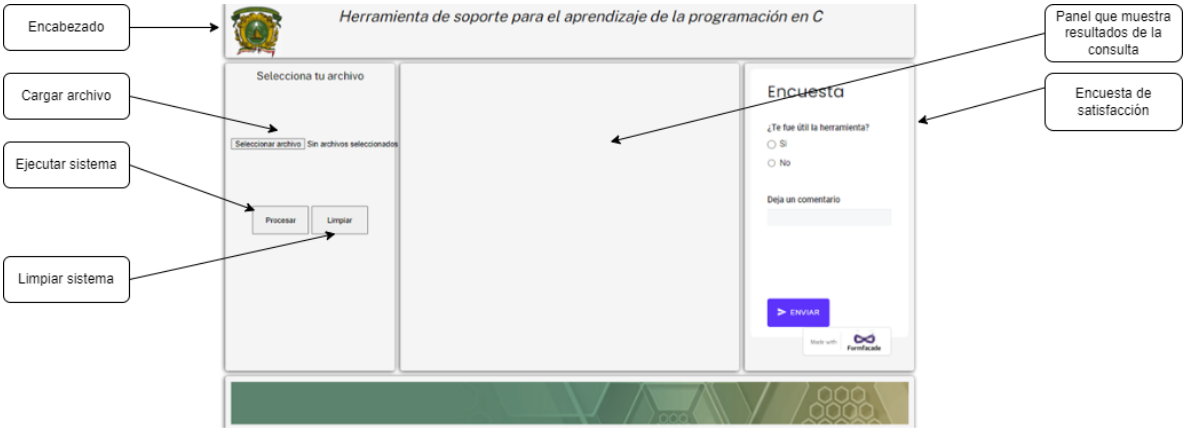


Figura 4.3 Interfaz de Usuario

4.8 Requerimientos funcionales

El sistema deberá poder ejecutarse en cualquier sistema operativo actualizado, en el que el usuario podrá iniciar el sistema e ingresar los mensajes de error del compilador en la ventana indicada. El tiempo de respuesta al ingresar al sistema debe ser inferior a 5 segundos, permitir acceder al menos a 100 usuarios el mismo tiempo sin perjudicar la continuidad y eficacia del servicio. El sistema fue desarrollado como una aplicación web y el equipo en el que se va a ejecutar deberá tener instalado Python 3 y MinGW. Una vez cargado el archivo con el programa con error y se ejecuta el sistema, este debe mostrar la recomendación mostrada en la Figura 4.4.

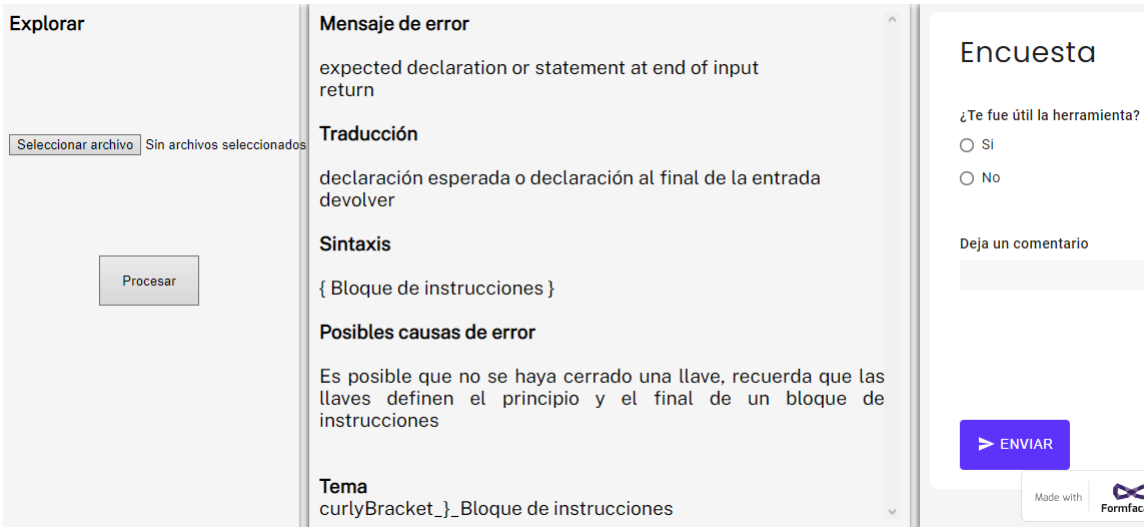


Figura 4.4 Recomendaciones

4.9 Diagrama de casos de uso

El diagrama de casos de uso (ver Figura 4.5) muestra las operaciones que el usuario puede realizar con el sistema, en este caso iniciar el sistema, cargar el programa con error, ejecutar el sistema, contestar y enviar encuesta y cerrar el sistema. Estas operaciones también son mostradas en el diagrama de secuencia

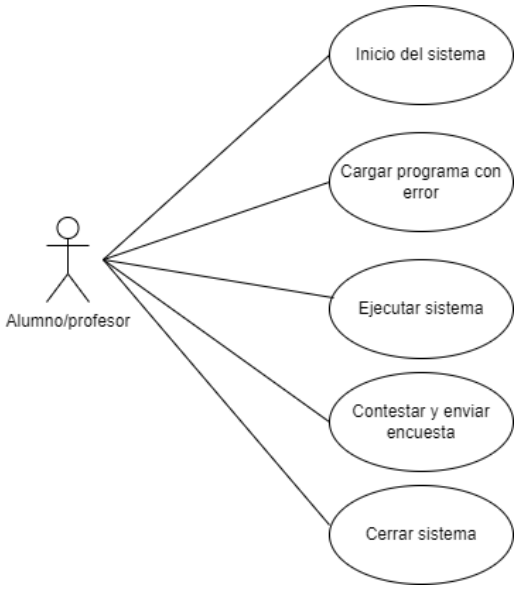


Figura 4.5 Diagrama de casos de uso

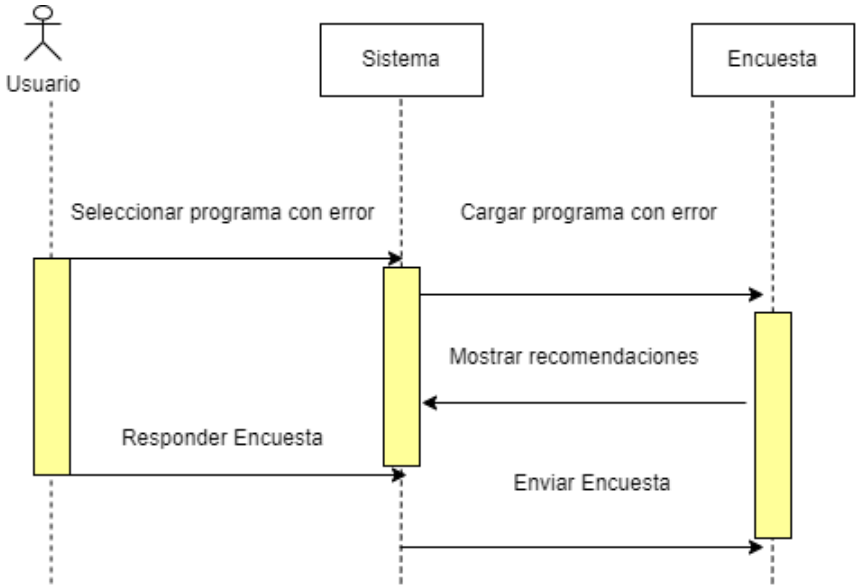


Figura 4.6 Diagrama de Secuencia

4.10 Acceso al sistema

Acceder a símbolo de sistema

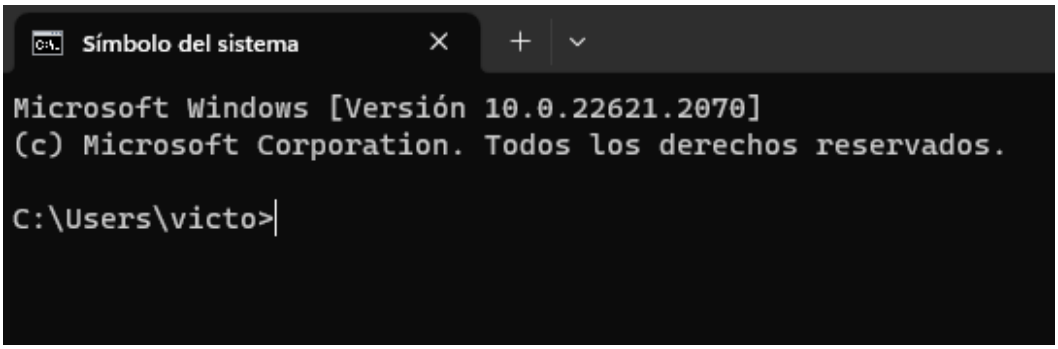


Figura 4.7 Símbolo del sistema

Cambiar la dirección a la carpeta del proyecto, en este caso analizador y escribir el comando python predict.py

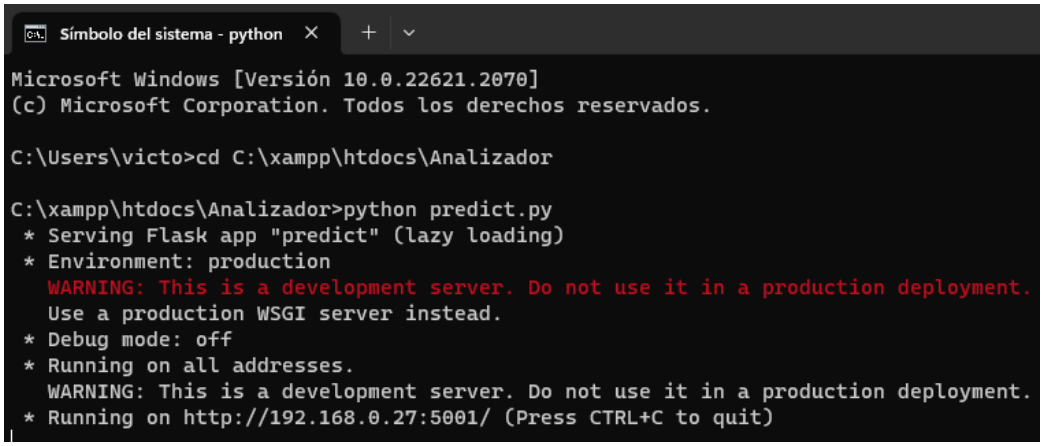


Figura 4.8 Ejecución del sistema

Ingresar en el navegador la dirección http://192.168.0.27:5001/. Para cerrar el sistema presionar CTRL+C

Capítulo 5. Conclusiones y trabajo futuro

Desde hace años se han presentado dificultades en la interpretación de los mensajes de error del compilador durante la codificación de programas por parte de alumnos y docentes. Han sido numerosos los trabajos que abordan esta problemática desarrollando sistemas, herramientas y modelos basados en inteligencia artificial ya sea para detectar el error en el programa y corregirlo, hacer sugerencias para su solución o presentar el mensaje de error de forma más legible para el estudiante. Además de los trabajos mencionados en el estado del arte, en la presente investigación se propuso que el estudiante pueda aprender de los errores que comete al codificar programas, mediante un modelo predictivo basado en técnicas de minería de textos y aprendizaje automático, el cual clasifica el mensaje de error de compilación de acuerdo con los tipos de error comúnmente cometidos por los estudiantes. Posteriormente se integró un sistema con el modelo predictivo, un módulo de conocimiento y una interfaz de usuario para que a partir de la clasificación realizada por el modelo predictivo, genere una retroalimentación al mensaje de error de compilación. Tal retroalimentación consiste en la traducción del error, la sintaxis, las causas que lo provocan y el tema de estudio relacionado con el error con el objetivo de que el estudiante aprenda de su error.

Para la generación del corpus se crearon 35 programas modelo en lenguaje C tomando como referencia los programas de estudio de un curso inicial de programación. Después de un proceso de inyección de errores a los programas modelo, se obtuvo un total de 2653 documentos (mensajes de error). Posteriormente se limpiaron los datos y se vectorizaron con el estadístico TF-IDF. Una vez obtenido el corpus, se utilizó el 80% para entrenar los modelos y el 20% para prueba. Para el desarrollo de los modelos se utilizaron los algoritmos Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors, Multi-layer perceptron y finalmente un proceso de votación de estos algoritmos, es decir, después de que cada algoritmo haya generado su predicción, se eligió el más común entre ellos. Después de ajustar los hiperparámetros para cada modelo y evaluarlos con el conjunto de prueba integrado por 531 documentos, todos tuvieron una exactitud del 90%, excepto por el Support Vector Machine, el cual tuvo una exactitud del 89%.

Una vez que el modelo fue desarrollado, se realizaron dos experimentos para verificar el comportamiento del modelo, en el primero se utilizó el conjunto de prueba, del cual el modelo Random Forest y K-Nearest Neighbors tuvieron una exactitud del 90.39% siendo la más alta y en el segundo experimento se utilizó un nuevo conjunto de datos, es decir, no contenido en el corpus original, pero con características similares, cuya exactitud más alta la obtuvo el modelo Random Forest con 91.39%. Por lo anterior, se consideró este clasificador para el modelo predictivo del sistema.

Para el trabajo futuro se utilizará la matriz de confusión para generar recomendaciones alternativas a partir de las instancias clasificadas incorrectamente, ampliar el tipo de error de compilación a lógicos y semánticos, realizar modelos para otros lenguajes de programación y finalmente, implementar el modelo en un escenario del mundo real, es decir con grupos de alumnos con los que se pueda realizar un análisis estadístico de los errores que comenten y conocer si hay algún tipo de error que no se está contemplando.

Referencias

Ahmed, T., Ledesma, N. R., & Devanbu, P. (2022). SYNSHINE: improved fixing of Syntax Errors. *IEEE Transactions on Software Engineering, 49*(4), 2169-2181.

Ahmed, U., Kumar, P., Karkare, A., Kar, P., & Gulwani, S. (2018). Compilation Error Repair: For the Student Programs, From the Student Programs. *IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, (pp. 78-87). Gothenburg, Sweden.

Ahmed, U., Sindhgatta, R., Srivastava, N., & Karkare, A. (2019). Targeted Example Generation for Compilation Errors. *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 327-338). IEEE.

Álvarez, J. F. (2019). El error como estrategia pedagógica para generar un aprendizaje eficaz. *Conference Proceedings CIVINEDU* (p. 166). España: REDINE.

Amat, J. (2020, Diciembre). *Máquinas de Vector Soporte (SVM) con Python*. Retrieved 03 26, 2023, from https://www.cienciadedatos.net/documentos/py24-svm-python.html

Arévalo-Alonso, J. (2018, Febrero 22). *Universo Abierto*. Retrieved 02 2023, 28, from ¿Qué es la minería de textos, cómo funciona y por qué es útil?: https://universoabierto.org/2018/02/22/que-es-la-mineria-de-textos-como-funciona-y-por-que-es-util/

Argente, E., Sapena, O., Botti, V., Serra, J., Chica, A., & Corma, A. (2001). Aplicación de una red neuronal para la predicción de la reacción catalítica isomerización del n-Octano. *Actas de la IX Conferencia de la Asociación*

Española para la Inteligencia Artificial, IV Jornadas de Transferencia Tecnológica de Inteligencia Artificial. Gijón.

Barrera, L. (2012). FUNDAMENTOS HISTÓRICOS Y FILOSÓFICOS DE LA INTELIGENCIA ARTIFICIAL UCV-HACER. *Revista de Investigación y Cultura, 1*(1), 87-92.

Becker, B. (2016). An effective approach to enhancing compiler error messages. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, (pp. 126-131).

Bofang, L., Zhe, Z., Tao, L., Puwei, W., & Xiaoyong, D. (2016). Weighted Neural Bag-of-n-grams Model: New Baselines for Text Classification. *In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* , (pp. 1591-1600).

Bordes, M. (2021). *adaitw.org*. Retrieved from La programación como herramienta indispensable de la actualidad: https://adaitw.org/novedades/la-programacion-como-herramienta-indispensable-de-la-actualidad/

Campañ, P., Satorre, R., Llorens, F., & Molina, R. (2015). Enseñando a programar: un camino directo para desarrollar el pensamiento computacional. *Revista de EducacIón a Distancia.* Retrieved from http://www.um.es/ead/red/46

Campos Mocholi, M. (2021). Clasificación de Textos Basada en Redes Neuronales. *Doctoral dissertation, Universitat Politècnica de València*.

Carmona, E. (2014). *Tutorial sobre Máquinas de Vectores Soporte.* Madrid: ETS de Ingeniería Informática, Universidad Nacional de Educación a Distancia.

Centro de Innovación Industrial en Inteligencia Artificial. (2021, 05 8). *Aprendizaje Supervisado*. Retrieved from CII.IA: https://www.ciiia.mx/noticiasciiia/aprendizaje-supervisado

Charnelli, M. (2019). Sistemas Recomendadores aplicados a la Educación. [Trabajo de Especialización]. Universidad Nacional de la Plata. Retrieved from http://sedici.unlp.edu.ar/handle/10915/85850

Contreras Barrera, M. (2014). Minería de Texto: Una visión actual. *Biblioteca Univeritaria, 17*(2), 129-138. Retrieved from https://doi.org/10.22201/dgb.0187750xp.2014.2.72

Cordova, D. G., Flores, E. N., García, R. R., & Salvador, J. C. (2023, 03 21). *Inteligencia artificial, la herencia de Alan Turing*. Retrieved from Ciencia UNAM: https://ciencia.unam.mx/leer/631/inteligencia-artificial-la-herencia-de-alan-turing-

Coull, N. J. (2008). *SNOOPIE: development of a learning support tool for novice programmers within a conceptual framework (Doctoral dissertation, University of St Andrews).* (Doctoral dissertation, University of St Andrews).

Dann, W., Copper, S., & Pausch, R. (2006). Learning to program with Alice. Upper Saddle River, NJ: Prentice Hall.

Dasari, D. B., & K, V. G. (2012). Text Categorization and Machine Learning Methods: Current State of the Art. *Global Journal of Computer Science and Technology Software & Data Engineering*.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). Data Mining to Knowledge Discovery in Databases. *AI Magazine, 17*(3). doi:https://doi.org/10.1609/aimag.v17i3.1230

Feldman, R., & Sanger, J. (2007). *The text mining handbook: advanced approaches in analyzing unstructured data.* Cambridge: Cambridge university press.

Forman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification . *Journal of Machine Learning Research*, 289-1305.

Fu, C., & Yang, J. (2021). Granular Classification for Imbalanced Datasets: A Minkowski Distance-Based Method. *Algorithms*.

Fuentes, J., & Moo, M. (2017). Dificultades de aprender a programar. *Educación en Ingeniería, 12*(24), 76-82. Retrieved from https://doi.org/10.26507/rei.v12n24.728

Gaikwad, S., Chaugule, A., & Patil, P. (2014). Text Mining Methods and Techniques. *International Journal of Computer Applications (0975 – 8887)*.

García, J. (2023). *Tecnology Evaluation Centera.* Retrieved 03 19, 2023, from Informe de TEC sobre la industria del software inteligencia artificial en manufactura: Habilitando la cuarta revolución industrial - Part 2: https://www3.technologyevaluation.com/es/publications/27047/artificial-intelligence-in-manufacturing-enabling-the-fourth-industrial-revolution/part-2

García, U. (2020, 9 7). *Introducción a las Redes Neuronales Pt. II Implementación de un perceptrón en Python*. Retrieved 06 17, 2023, from https://medium.com/futurelabmx/introducci%C3%B3n-a-las-redes-neuronales-53abbc3abbf

González, A. (2017). *Microsoft*. Retrieved from La importancia en la vida diaria de aprender a programar: https://news.microsoft.com/es-xl/features/la-importancia-la-vida-diaria-aprender-programar/

González, L. (2019, 07 19). *Aprendeia*. Retrieved from https://aprendeia.com/algoritmo-k-vecinos-mas-cercanos-teoria-machine-learning/

Guerrero, M., Guaman, D., & Icaza, J. (2015). Revisión de Herramientas de Apoyo en el Proceso de Enseñanza-Aprendizaje de Programación. *Revista Politécnica.,* *31*(1), 84. Retrieved from https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/430

Hamdaoui, Y. (2019, Diciembre 9). *TF(Term Frequency)-IDF(Inverse Document Frequency) from scratch in python.* Retrieved from Towards Data Science: https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

Hearst, M. (2003). *What is Text MIning, SIMS, UC Berkeley*. Retrieved from https://people.ischool.berkeley.edu/~hearst/text-mining.html

Heemsoth, T., & Heinze, A. (2016). Secondary School Students Learning From Reflections on the Rationale Behind Self-Made Errors: A Field Experiment. *The Journal of Experimental Education*.

Hewlett Packard Enterprises. (2023). *Aprendizaje Automático*. Retrieved 03 12, 2023, from https://www.hpe.com/mx/es/what-is/machine-learning.html

Hucker, M. (2020, 06 9). *Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions*. Retrieved 03 26, 2023, from Towards Data Science: https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02

Iberdrola. (2023). *¿Qué es el Machine Learning?* Retrieved 03 18, 2023, from https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

Iberdrola. (2023). *¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?* Retrieved 03 18, 2023, from https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial

IBM. (2015, 08 15). *¿Qué es el algoritmo de k vecinos más cercanos?* Retrieved from https://www.ibm.com/mx

IBM. (2023). *¿Qué es la minería de texto?* Retrieved 8 24, 2023

Joyanes, L. (2016). *Big Data, Análisis de grandes volúmenes de datos en organizaciones.* México: Alfaomega Grupo Editor.

Kernighan, B., & Ritchie, D. (1991). *El lenguaje de Programación C.* Pearson Education.

Liu, B. (2011). *Web Data Mining.* Chicago: Springer.

Maheswari, U., & Sathiaseelan, D. J. (2017). Text Mining: Survey on Techniques and Applications. *International Journal of Science and Research (IJSR), 6*(6).

Malagón, L. C. (2013, Mayo 14). *Clasificadores bayesianos. El algoritmo Naïve Bayes [En línea]*. Retrieved from https://www.nebrija.es/~cmalagon/inco/Apuntes/bayesian_learning.pdf

Manzano, M. C. (2015). *Minería de datos para el diagnóstico de deterioro neuropsicológico a individuos expuestos a pesticidas organofosforados.* Universidad Técnica Federico Santa María.

Mariñelarena, L., Errecalde, M., & Castro, A. (2017). Extracción de conocimiento con técnicas de minería de textos aplicadas a la psicología. *Revista Argentina de Ciencias del Comportamiento, 9*(2), pp. 65-76. Retrieved from https://www.redalyc.org/journal/3334/333452119006/html/

Markoulidakis, I., Rallis, I., Georgoulas, I., Kopsiaftis, G., Doulamis, A., & Doulamis, N. (2021). Multiclass Confusion Matrix Reduction Method and Its Application on Net Promoter Score Classification Problem. *Technologies*. doi:https://doi.org/10.3390/technologies9040081

Mathworks. (2023). *Support Vector Machine (SVM)*. Retrieved 03 26, 2023, from https://la.mathworks.com/discovery/support-vector-machine.html

Mesbah, A., Rice, A., Aftandilian, E., Johnston, E., & Glorioso, N. (2019). DeepDelta: Learning to Repair Compilation Errors. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (pp. 925-936).

Miner, G., Elder, J., Fast, A., Hill, T., Nisbet, R., & Delen, D. (2012). *Practical Text Mining and Statistical Analysis for Non-Structured Text Data Applications.* Oxford,UK: Elsevier Inc.

Mitchell, T. (1997). *Machine Learning.* Nueva Delhi: McGraw Hill Education India.

Molina, L. C. (2000). *Torturando los Datos hasta que confiesen.* España: Departamento de Lenguajes y Sistemas Informáticos, Universidad de Cataluña.

Murphy, K. P. (2012). *Machine Learning: Una perspectiva probabilistica.* London: Cambridge: The MIT Press.

Nabeel, M., Majeed, S., Awan, Mazhar, M.-u.-D., Wasique, M., & Nasir, R. (2021). Review on Effective Disease Prediction through Data Mining Techniques. *International Journal on Electrical Engineering and Informatics, 13*(3).

Noriega, R., Mendoza, A., Robledo, I., Acosta, A., & Esquivel, C. (2016). Análisis de resultados del examen departamental: caso de estudio departamental de Fundamentos de Programación. *Universidad Autónoma de CIudad Juarez*, 26. Retrieved from https://erevistas.uacj.mx/ojs/index.php/culcyt/article/view/1495

Oracle. (2023). *¿Qué es el aprendizaje automático?* Retrieved from https://www.oracle.com/mx/artificial-intelligence/machine-learning/what-is-machine-learning/

Parmar, A., Katariya, R., & Patel, V. (2018). A Review on Random Forest: An Ensemble Classifier. *An Ensemble Classifier. Lecture Notes on Data Engineering and Communications Technologies*, 758–763. doi: doi:10.1007/978-3-030-03146-6_86

Qader, A., Ameen, M., & Ahmed, B. (2019). An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges. *Fifth*

International Engineering Conference on Developments in Civil & Computer Engineering Applications, (pp. 200-204). Erbil.

Qaiser, S., & Ramsha, A. (2018). Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. *International Journal of Computer Applications (0975 – 8887)*.

Raschka, S. (2017, 02 14). *NaiveBayes and Text Classification I Introduction and Theory.* Retrieved 03 27, 2023, from arXiv:1410.5329v4: http://chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/1410.5329.pdf

Rouhiainen, L. (2018). *Inteligencia Artificial 101 cosas que debes saber hoy sobre nuestro futuro.* Barcelona: Planeta.

Safavian, R., & Landgrebe, D. (1991). A Survey of Decision Tree Classifier Methodology. *IEEE transactions on systems, man, and cybernetics, 21(3)*, 660-674.

Santos, E. A., Campbell, C. J., Hindle, A., & Amaral, J. N. (2017). Finding and correcting syntax errors using recurrent neural networks. *PeerJ PrePrints, e3123v1*.

Seo, H.-T., Han, Y.-S., & Ko, S.-K. (2021). MultiFix: Learning to Repair Multiple Errors by Optimal Alignment Learning. *Findings of the Association for Computational Linguistics: EMNLP 2021* , (pp. 4850-4855).

Sommerville, I. (2011). *Software engineering 9th Edition ISBN-10: 0-13-703515-2.* Pearson.

Sun, W., & Sun, X. (2011). Teaching computer programming skills to engineering and technology students with a modular programming strategy. *ASEE Annual Conference & Exposition 22-1378*, (pp. 22-1378). Vancouver, BC. doi:10.18260/1-2--18625

Suresh, A. (2021, 12 17). *What is a Confusion Matrix?* Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

Sykes, E., & Franek, F. (2004). Presenting JECA: A Java Error Correcting Algorithm for the Java Intelligent. *IASTED International Conference on Advances in Computer Science and Technology.* St. Thomas, Virgin Islands, USA.

Tablada, C., & Torres, G. (2009). Redes neuronales artificiales. *Revista de educación matemática, 24*(30), 22-30.

Tejera, F., Aguilera, D., & Miguel, V. J. (2021). Lenguajes de programación y desarrollo de competencias clave. Revisión sistemática. *Revista electrónica de investigación educativa*.

Turing, A. (1950). Computing Machinery and Intelligence. Mind 49: 433-460.

Villén, M. (2023). *Ingeniero de Caminos, Canales y Puertos (1982). Universidad Politécnica de Madrid*. Retrieved 03 21, 2023, from Big Data Analytics y la inteligencia Artificial: https://www.caminosmadrid.es/9938-2

Viveros , J., López , M., & Villareal , Y. (2016). Estrategias para Reducir el Indice de Reprobación en Fundamentos de Programación de Sistemas Computacionales del I.T. Mexicali. *Revista de gestón empresarial y sustemtabilidad*, (págs. 25-41).

Wang, H., Henyuan, L., Li, Z., Liu, Y., Sun, F., & Chen, X. (2022). A Token-based Compilation Error Categorization and Its Applications. *Journal of Software: Evolution and Process, 35*(2).

Wu, B., Campora III, J. P., & Chen, S. (2017). Learning User Friendly Type-Error Messages. *Proc. ACM Program. Lang. 1, OOPSLA, Article 106*. doi:10.1145/3133930

Yasunaga, M., & Liang, P. (2020). *Graph-based, Self-Supervised Program Repair from Diagnostic Feedback,International Conference on Machine Learning*, (pp. 10799-10808).

Yasunaga, M., & Liang, P. (2020). Graph-based self-supervised program repair from diagnostic feedback. *International Conference on Machine Learning*, (pp. 10799-10808).

Zhou, Z., Wang, S., & Qian, Y. (2021). Learning From Errors: Exploring the Effectiveness of Enhanced Error Messages in Learning to Program. *Frontiers in Psychology 12:768962*.