ANÁLISIS DE LA SIMILITUD EN EL CÓDIGO FUENTE DE DOS PROGRAMAS DE COMPUTADOR USANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL

MAURICIO LÓPEZ BENÍTEZ (Autor Trabajo de Grado)

Nota de Aceptación

Certificamos que el presente Trabajo de Grado
Satisface, en alcances y calidad, todos los requisitos
Que demanda un Trabajo de Grado de Maestría.

GLORIA INÉS ÁLVAREZ
Director

LUISA FERNANDA RINCÓN
Jurado

GERARDO SARRIA
Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en Ingeniería énfasis Sistemas y Computación.

HERNAN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTINEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Santiago de Cali, 18 de febrero de 2021

Maestría en Ingeniería
Facultad de Ingeniería y Ciencias

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, 23 de febrero de 2021

Autor: MAURICIO LÓPEZ BENÍTEZ

Título del Trabajo de Grado: "ANÁLISIS DE LA SIMILITUD EN EL CÓDIGO FUENTE DE DOS PROGRAMAS DE COMPUTADOR USANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL"

Director: GLORIA INÉS ÁLVAREZ

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Gloria Inés Álvarez V.

Firma del Director del Trabajo de Grado

Pontificia Universidad JAVERIANA Cali

Res. 2333 del 2012

Santiago de Cali, 14 de Diciembre de 2020

Ingeniero:
JUAN CARLOS MARTINEZ ARIAS
Director Posgrados de Ingeniería
Facultad de Ingeniería
Pontificia Universidad Javeriana Cali

Cumplido los requisitos establecidos en los artículos 5.6 y 5.7 de las Directrices para Trabajo de Grado de Maestría, solicitamos se autorice la sustentación del Trabajo de Grado denominado "ANÁLISIS DE LA SIMILITUD EN EL CÓDIGO FUENTE DE DOS PROGRAMAS DE COMPUTADOR USANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL", realizado por el (la) estudiante MAURICIO LÓPEZ BENÍTEZ con código 0069273 perteneciente al énfasis en Ingeniería de Sistemas, bajo la dirección de la profesora GLORIA INES ALVAREZ.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer su sustentación ante el Tribunal que para el efecto se designe, toda vez que ha revisado meticulosamente el documento y avala que el Trabajo de Grado ya se encuentra listo para ser evaluado oficialmente.

Atentamente,

MAURICIO LÓPEZ BENÍTEZ
C.C. 94.393.536 de Tuluá

GLORIA INES ÁLVAREZ VARGAS
C.C. 30.306.105 de Manizales

Documentación anexa:
Dos copias anilladas del documento de Trabajo de Grado, con impresión por lado y lado y paginación completa. El resumen del Trabajo de Grado en formato electrónico (máximo 1 página).

DATOS DEL ESTUDIANTE

Nombres y Apellidos: MAURICIO LÓPEZ BENÍTEZ

Dirección: Calle 32 A N # 2 B – 63 Ap 602 C

Teléfono: 378 28 46

Celular: 301 201 0079

Correo electrónico: mauricio.lopez.b@gmail.com

Profesión: Ingeniero de Sistemas

Universidad: Universidad del Valle

Empresa: Universidad del Valle

Cargo: Profesor

Pontificia Universidad JAVERIANA Cali

Maestría en Ingeniería
Facultad de Ingeniería y Ciencias

FICHA RESUMEN
TRABAJO DE GRADO DE MAESTRÍA

TITULO: **"ANÁLISIS DE LA SIMILITUD EN EL CÓDIGO FUENTE DE DOS PROGRAMAS DE COMPUTADOR USANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL"**

1. ÉNFASIS: **Ingeniería de Sistemas y Computación**
2. ÁREA DE INVESTIGACIÓN: **Inteligencia Artificial**
3. ESTUDIANTE: **Mauricio López Benítez**
4. CORREO ELECTRÓNICO: **mauricio.lopez.b@gmail.com**
5. DIRECTOR: **Gloria Inés Álvarez Vargas, Ph.D.**
6. CO-DIRECTOR(ES:
7. GRUPO QUE LO AVALA:
8. OTROS GRUPOS:
9. PALABRAS CLAVE: **Code Plagiarism, Artificial Intelligence, Machine Learning**
10. CÓDIGOS UNESCO CIENCIA Y TECNOLOGÍA: **1203.04**, **1203.09**
11. FECHA DE INICIO: **30** de **06** de **2019** DURACIÓN ESTIMADA: 8 Meses
12. RESUMEN (máximo una página).

El presente proyecto aborda el problema del plagio en el código fuente del software, una situación que se presenta con frecuencia tanto en los entornos académicos como en el ámbito profesional; por esta razón, se ha propuesto como objetivo general desarrollar una herramienta computacional para analizar la similitud del código fuente de dos programas de computador usando técnicas de Inteligencia Artificial; para lograrlo se plantearon cinco objetivos específicos que abarcan la representación de los programas objeto de análisis, la preparación de los datos para el entrenamiento, la selección de las técnicas de Inteligencia Artificial a utilizar, la implementación del modelo y la evaluación de resultados. Como resultado del proyecto se espera que el modelo implementado en la herramienta pueda ser utilizado en el desarrollo de aplicaciones de detección de fraude.

RESUMEN

El plagio es la acción de usar el trabajo de otros, o una parte de este, sin el reconocimiento a los autores. Esta práctica ocurre en diferentes campos como la educación, un caso se presenta en los cursos de lenguajes de programación donde los estudiantes presentan proyectos en los que hay reuso de código fuente, en los que hay una similitud implícita.

Para determinar la similitud en un par de códigos fuente se debe considerar los diferentes niveles de modificación que pueden encontrarse, los cuales van desde el cambio en el nombre de variables, hasta el reemplazo de expresiones por expresiones equivalentes.

En este trabajo se presenta un método para identificar la similitud de código fuente de dos programas de computador usando técnicas de aprendizaje supervisado. Se inicia con una estrategia de transformación del código fuente como secuencia de tokens y trigramas, hasta obtener los vectores Tf-Idf para representar cada programa.

En el siguiente paso se usaron tres técnicas para determinar la similitud: la similitud coseno, Naive-Bayes y redes neuronales recurrentes LSTM. Los resultados muestran que el desbalance en el dataset afecta significativamente al modelo Naive-Bayes, en otro caso, las redes LSTM no se vieron afectadas.

Los métodos propuestos fueron entrenados y evaluados usando un corpus conformado con el repositorio SOCO (International competition on detection of SOurce COde reuse), que se compone de 259 programas en lenguaje java con diferentes tipos de reuso de código. Los resultados obtenidos usando las redes neuronales LSTM fueron mejores que aquellos obtenidos con los otros dos modelos y los reportados en el estado del arte.

Palabras clave: Machine Learning, code plagiarism, similitud de código, clasificador binario, redes LSTM.

ABTRACT

Plagiarism is the act of using the work of others, or a part of it, without the acknowledging authors. This practice occurs in different fields such as education, a case is presented in programming language courses where students present projects in which there is a reuse of source code, in which there is an implicit similarity.

To determine similarity in a pair of source codes, it's necessary to consider the different levels of modification that can be found, which range from changing the variable´s name to replacing expressions with equivalent expressions.

This work presents a method to identify the source code similarity of two computer programs using supervised learning techniques. It starts with a strategy of transforming the source code into a sequence of tokens and trigrams until obtaining the Tf-Idf vectors to represent each program.

In the next step, three techniques were used to determine similarity: cosine similarity, Naive-Bayes, and LSTM recurrent neural networks. The results show that the imbalance in the dataset significantly affects the Naive-Bayes model, otherwise, LSTM networks were not affected.

The proposed methods were trained and evaluated using corpora formed with the SOCO (International competition on detection of SOurce COde reuse) repository, which consists of 259 java language programs with different types of code reuse. The results obtained using LSTM neural networks were better than those obtained with the other two models and those reported in the state of the art.

Keywords: Machine Learning, code plagiarism, code similarity, binary classifier, LSTM networks.

Pontificia Universidad JAVERIANA Cali
Vigilada Mineducación
Res. 2333 del 2012

ANÁLISIS DE LA SIMILITUD EN EL CÓDIGO FUENTE DE DOS PROGRAMAS DE COMPUTADOR USANDO TÉCNICAS DE INTELIGENCIA ARTIFICIAL

MAURICIO LÓPEZ BENÍTEZ

Código 0069273

Trabajo de grado para optar al título de

Magister en Ingeniería – énfasis en Ingeniería de Sistemas

Directora

GLORIA INÉS ALVAREZ, PhD

FACULTAD DE INGENIERÍA Y CIENCIAS

MAESTRÍA EN INGENIERÍA - ÉNFASIS SISTEMAS Y COMPUTACIÓN

SANTIAGO DE CALI, DICIEMBRE DE 2020

TABLA DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

Con la rápida evolución de las Tecnologías de la Información y la Comunicación, cada vez las personas tienen mayores posibilidades de acceder a todo tipo de contenidos: desde entretenimiento musical y de videojuegos, pasando por las redes sociales, comercio y mercadeo electrónico, hasta el acceso a fuentes de información disciplinar; adicionalmente, más allá de los sistemas interconectados, como la internet, los usuarios de la tecnología tienen la posibilidad de acceder a información y compartirla por medio de los dispositivos de almacenamiento; sin embargo, la tecnología así como es una poderosa herramienta que brinda la posibilidad de generar conocimiento y comunicarlo, a veces es utilizada de forma inapropiada por quienes cometen plagio y, por ende, sin dar el crédito de propiedad intelectual o derechos de autor de quienes publican o comparten el resultado de su producción intelectual.

Con cierta frecuencia los medios de comunicación informan de casos de plagio que se presentan en diferentes ámbitos: en las artes, las investigaciones científicas, e incluso el software; estas conductas son todavía más comunes al interior de los claustros universitarios en los que algunos estudiantes, de manera consciente o no, presentan como propio parte de documentos tanto de autores reconocidos, como usuarios anónimos de las redes e incluso de sus mismos compañeros, por tal razón, este trabajo de grado aborda el problema del plagio en el software, considerando el caso de la identificación de similitud del código fuente de dos programas de computador, cuyo planteamiento se detalla en la sección 1.1.

Si bien es cierto que algunos investigadores han abordado este problema, como se presenta en la revisión de trabajos relacionados de la sección 2.2, los enfoques de solución se han desarrollado en diferentes vías y no se puede considerar un problema completamente resuelto o explorado; por lo que en las secciones 1.3 y 1.4 que corresponden a los objetivos, se plantea el desarrollo de una herramienta que use técnicas de Inteligencia Artificial para solucionar el problema identificado.

Dado que para llegar a la solución es necesario recoger conceptos de compiladores y de Inteligencia Artificial, en la sección 2.1 correspondiente a las bases teóricas, se hace una descripción de los conceptos asociados con estas áreas y una revisión de algunos algoritmos de aprendizaje automático que pueden ser considerados dentro del desarrollo del proyecto.

En el capítulo 3 se describe todo el proceso de ingeniería del proyecto en el que se plantean 3 enfoques de solución junto con las consideraciones para su parametrización y las decisiones tomadas de acuerdo con los experimentos iniciales.

El capítulo 4 se hace un análisis de los resultados obtenidos con los tres modelos implementados, en principio de manera individual y terminando con un análisis comparativo el cual conduce a la formulación de las conclusiones planteadas en el capítulo5.

Al final del documento se presentan los anexos en los que se exhibe el diagrama de componentes de la herramienta implementada para abordar la investigación y se indica la ubicación del repositorio en línea en el cual puede ser consultado el código fuente de los archivos que componen la solución.

Finalmente, se presenta la relación de referencias bibliográficas que fueron seleccionadas en el proceso de revisión sistemática de la literatura con el fin de dar solidez a la propuesta que aquí se presenta.

1.1 PLANTEAMIENTO DEL PROBLEMA

El plagio es un tema que ocupa la atención en el entorno académico, particularmente en la educación superior, de acuerdo a lo mencionado en [1], es una práctica frecuente entre los estudiantes; esto sumado al creciente acceso a la tecnología hace que cada vez sea más sencillo para un estudiante acceder a diversas fuentes de información facilitando que, de forma intencional o no, un estudiante pueda presentar como propio un trabajo (o parte de él) que no es el resultado de su producción intelectual y en el mejor de los casos limitándose a cambiar el formato.

Al hablar de plagio se viene a nuestro imaginario su relación directa con la escritura de documentos, sin embargo, éste fenómeno también está presente en las asignaturas del área de programación en las que, por su naturaleza teórico-práctica, basan su evaluación en el desarrollo de proyectos académicos más que en la presentación de trabajos escritos; con la asignación de estos proyectos se pretende que los estudiantes adquieran competencias profesionales en dicha área. Estos proyectos están enfocados en el diseño e implementación de una solución computacional usando un lenguaje de programación para su construcción, teniendo como resultado final un producto de software que debe satisfacer los requerimientos planteados en el proyecto.

La revisión de un proyecto de programación por parte de los docentes sugiere un conjunto de elementos que deben ser tenidos en cuenta: la correctitud de la solución, la lógica y el nivel de optimización de la misma, y una característica relevante en el proceso de evaluación: la originalidad de la solución implementada.

El problema abordado en este proyecto está relacionado con la última característica de evaluación mencionada: la originalidad de los proyectos de programación, dado que con frecuencia se presentan casos de plagio en el código que implementa la solución del problema relativo al proyecto y, aunque encontramos al alcance de la mano herramientas como Turnitin, PlagScan, Decode, entre otros que permiten detectar posibles situaciones de plagio

en los trabajos escritos, para el caso del análisis del código fuente de los proyectos de programación el panorama es diferente.

Actualmente, la alternativa para garantizar la originalidad del proyecto presentado por un estudiante se reduce a la simple observación que hace el docente, apelando a su capacidad de retentiva, lo cual puede llegar a ser no solamente tedioso y desgastante si no poco fructífero, incluso reduciendo el foco de observación al conjunto de proyectos entregados dentro de un grupo.

En este sentido, el problema se planteó en términos de determinar el nivel de semejanza en el código fuente de dos proyectos de programación utilizando técnicas de Inteligencia Artificial y minería de textos.

1.2 FORMULACIÓN DEL PROBLEMA

De acuerdo con el planteamiento descrito anteriormente, se llegó a formular la siguiente pregunta: ¿Cómo medir el nivel de similitud en dos códigos de programación usando técnicas de Inteligencia Artificial?

Lo anterior implica responder los siguientes interrogantes: ¿Qué características considerar como relevantes para estimar el grado de similitud al comparar dos códigos fuente?, ¿Cuáles técnicas de Inteligencia Artificial deben ser seleccionadas de tal forma que aporten adecuadamente a la solución del problema?, ¿De qué manera incorporar estrategias de análisis de textos en el proceso de análisis para obtener mejores resultados?, ¿Cómo seleccionar los datos requeridos para el análisis? y ¿Cómo valorar el nivel de desempeño de la herramienta desarrollada de acuerdo con los resultados obtenidos?

1.3 OBJETIVO GENERAL

Desarrollar una herramienta para determinar la similitud del código fuente de dos programas de computador usando técnicas de Inteligencia Artificial.

1.4 OBJETIVOS ESPECÍFICOS

1. Seleccionar las técnicas de Inteligencia Artificial que serán utilizadas.
2. Definir un modelo de representación de los programas que se van a analizar.
3. Preparar los datos para el entrenamiento y prueba de los modelos.
4. Implementar un modelo de análisis de similitud del código fuente de dos programas de computador.
5. Evaluar el desempeño de la herramienta desarrollada.

1.5 JUSTIFICACIÓN

Como se ha descrito en el planteamiento del problema, el tema del plagio en la sociedad actual y, particularmente en el entorno académico universitario, cobra una especial relevancia debido al uso inadecuado de las herramientas TIC por parte de un porcentaje de estudiantes de educación superior; esta afirmación está en concordancia con lo expresado en [2], donde se pone de manifiesto que las tecnologías de la información han generado transformaciones positivas y, al mismo tiempo, una serie de “problemáticas éticas” con respecto al uso de las mismas.

De acuerdo a lo anterior, y teniendo presente que el problema del plagio del código fuente se ha abordado en diferentes proyectos de investigación, con muy variados enfoques, se consideró necesario hacer la exploración de alternativas tendientes a proponer modelos de análisis de similitud de códigos que puedan ser aprovechadas para el desarrollo de herramientas para la detección de plagio.

Si bien es cierto que el acto deshonesto de apropiarse del trabajo de otro no debería presentarse, y menos en el entorno académico donde se están formando profesionales que deben aportar a la sociedad con un estricto sentido ético, es necesario reconocer que esto es prácticamente una utopía, por esta razón, se hace necesario implementar mecanismos de verificación que apunten a minimizar tales actos, de hecho, como se menciona en [3] "la manera en que un individuo percibe su ambiente determina en gran parte sus actuaciones". Con base en estos argumentos, se puede concluir que el proyecto que se presenta en este documento es relevante en el entorno social y académico pues plantea la implementación de estrategias encaminadas a la construcción de bases para la detección de plagio académico en los cursos de programación.

Desde el punto de vista del aporte teórico, y con base en la revisión inicial de la literatura, el proyecto es justificable debido a que las experiencias previas que han sido revisadas, reportan resultados con diferentes niveles de precisión dependiendo de las técnicas utilizadas para el análisis de similitud en sus casos de prueba y proponiendo trabajos futuros, como se menciona en [4]; esto indica que la investigación en el área no está completamente terminada y, por el contrario, pueden ser consideradas estrategias de Inteligencia Artificial que permitan el mejoramiento de los resultados experimentales. Tales estrategias se enfocarán en el uso de métodos de aprendizaje automático para la identificación de patrones, lo que facilita la toma de decisiones acerca del nivel de similitud entre un par de programas de computador.

Finalmente, en términos prácticos, se puede señalar que el proyecto se justifica debido a la vigencia que tiene el problema seleccionado, particularmente en el contexto de formación académica dentro del cual se enmarca la investigación. De este modo, la solución que se busca construir, contribuirá tanto con la practicidad que puede representar para un docente en el momento que deba hacer revisiones de los trabajos entregados por sus estudiantes, al poder identificar el nivel de similitud de tales entregas, permitiendo tanto reducir los tiempos de revisión, como la objetividad de las mismas en cuanto a su autenticidad.

2. MARCO TEÓRICO DE REFERENCIA

2.1 BASES TEÓRICAS

De acuerdo con la revisión sistemática de la literatura, que ha permitido hacer una exploración sobre las experiencias y proyectos desarrollados alrededor del problema objeto de estudio en este trabajo de grado, se considera pertinente presentar un referente teórico para el desarrollo del proyecto. A continuación, se describen estos elementos considerados relevantes por su relación directa con el problema objeto de estudio y que dan elementos

2.1.1 Compiladores.

La computación está presente en prácticamente todo tipo de dispositivos, los cuales usan programas que operacionalizan sus propiedades. Estos programas implementados en algún lenguaje de programación deben ser traducidos, mediante un compilador, en una serie de instrucciones que el microprocesador del dispositivo pueda interpretar.

El proceso de compilación puede resumirse en la figura 1, donde se pueden visualizar los diferentes componentes.

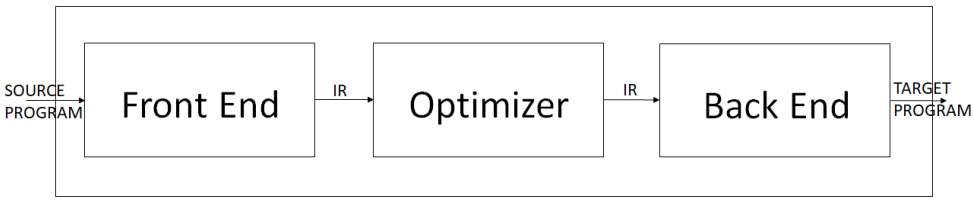


Figura 1: Componentes de un compilador.

El Front End es un analizador del código fuente, el cual convierte este código en una representación intermedia (IR) o lenguaje de bajo nivel similar al lenguaje ensamblador; de esta manera el Front End se asegura que el código está bien formado [5]. El optimizador se encarga de analizar la representación intermedia con el fin de generar una versión más eficiente de la misma. Finalmente, el Back End se encarga de generar el código de máquina

a partir del contenido de la IR, razón por la cual se asume que en este punto se han superado los errores sintácticos y semánticos.

Entrando un poco más en detalle, el Front End está compuesto por el analizador léxico, el analizador sintáctico, el analizador semántico y el generador de código intermedio; mientras que el Back End lo conforman el generador del código objeto y el optimizador del código objeto.

El analizador léxico (scanner) transforma el código fuente en tokens, eliminando caracteres innecesarios para el proceso de compilación como los espacios en blanco, comentarios o tabulaciones; estos tokens son recibidos por el analizador sintáctico (parser) que, por medio de la construcción de un árbol sintáctico de derivación (o árbol de sintaxis abstracta), valida la construcción gramatical del conjunto de instrucciones del código que se está compilando. En la última etapa del Front End, antes de la generación del código intermedio, el analizador semántico comprueba que los operadores son usados correctamente y que los operandos son compatibles con los operadores. [6]

- **Algoritmos de análisis sintáctico**

Como se mencionó anteriormente, un analizador sintáctico recibe los token generados por el analizador léxico y los transforma en una estructura, normalmente tipo árbol, con el fin de determinar si la entrada puede ser derivada desde el símbolo inicial por medio de las reglas de una gramática formal. Los algoritmos de análisis sintáctico son de diferentes enfoques, siendo los más comunes el analizador sintáctico ascendente (Bottom-Up-Parser) y analizador sintáctico descendente (Top-Down-Parser).

El análisis sintáctico ascendente busca construir el árbol sintáctico desde los tokens hacia el símbolo inicial, con lo cual se pretende disminuir la cantidad de reglas mal aplicadas. El proceso consiste en aplicar operaciones sobre un conjunto de token e ir pasando de una configuración a otra hasta que se hayan consumido todos los token y se obtenga el árbol

sintáctico completo. Los algoritmos de este tipo pueden clasificarse como analizador sintáctico ascendente con retroceso o como analizador sintáctico ascendente LR.

El análisis sintáctico descendente busca derivar la entrada a partir del símbolo inicial por medio de las producciones de la gramática. El proceso que sigue este tipo de analizador sintáctico es: examinar los símbolos terminales en el orden en que aparecen en la cadena de tokens, selecciona las reglas gramaticales aplicables y obtiene el árbol de análisis sintáctico si la entrada es correcta; esto quiere decir que el árbol se construye desde la raíz y de izquierda a derecha. Los algoritmos de análisis sintáctico descendente pueden clasificarse en descendente con retroceso, descendente con predicción o descendente LL1.

La principal diferencia entre los analizadores sintácticos ascendentes y descendentes radica en que el primero utiliza menos reglas, por lo que disminuye el número de reglas mal aplicadas

2.1.2 Similitud Coseno

La similitud coseno es una medida basada en el cálculo del coseno del ángulo entre dos objetos con una representación vectorial, de tal forma que, si dos vectores tienen la misma dirección, su ángulo es 0^o, por lo que el coseno de dicho ángulo será 1; en el mismo sentido dos vectores con direcciones distintas estarán orientados a 90^o, por lo que el coseno de dicho ángulo será 0. En este orden de ideas, dos vectores que tienen la misma orientación tendrán una similitud de 1, mientras que en el caso contrario la similitud tiende a ser 0.

Esta medida de similitud es ampliamente usada en el análisis de textos para determinar cuán parecidos son un conjunto de documentos entre sí, a partir de la representación vectorial de los mismos que es formada con diferentes estrategias tales como el cálculo de la frecuencia de términos, o la frecuencia inversa de términos.

2.1.3 Machine Learning

Esta rama de la Inteligencia Artificial, que en español es conocida como “Aprendizaje Automático” o “Aprendizaje de Máquina”, cuyo objetivo inicial era el reconocimiento de

patrones, actualmente tiene aplicaciones en diferentes ámbitos como la recuperación de información o el razonamiento probabilístico. [7]

Diferentes definiciones de esta disciplina se enuncian a continuación: “La ciencia de hacer que las computadoras actúen sin estar explícitamente programadas” Andrew Ng, Profesor de la Universidad de Stanford; “ el campo de aprendizaje automático o Machine Learning busca responder a la pregunta ¿cómo podemos construir sistemas informáticos que mejoren automáticamente con la experiencia, y cuáles son las leyes fundamentales que rigen todos los procesos de aprendizaje?” Tom Mitchell, Profesor de la Universidad Carnegie Mellon. De estos conceptos se podría deducir entonces que el aprendizaje automático implica el diseño y utilización de algoritmos que al ser ejecutados sobre un conjunto de datos puedan revelar información útil sobre los mismos.

Básicamente, el aprendizaje automático puede clasificarse en dos tipos: aprendizaje supervisado y aprendizaje no supervisado [8]. En el primer caso, se parte de un conjunto de datos que están previamente etiquetados, es decir de un conjunto de problemas que ya están resueltos y que sirven como base de conocimiento para encontrar las relaciones entre el problema (conjunto de atributos) y su solución. Este tipo de aprendizaje puede ser de regresión, cuando se pretende predecir sobre un dato continuo, o de clasificación cuando lo que se pretende es predecir sobre un resultado discreto (una clase).

En el caso del aprendizaje no supervisado, el conjunto de datos de entrada no tiene asociados los valores de sus salidas, por lo que el aprendizaje debe darse gradualmente sin que hayan sido suministrados ejemplos etiquetados para tal efecto. Una de las técnicas más utilizadas en este tipo de aprendizaje es la de clustering, donde los datos son agrupados de acuerdo a un criterio en particular.

Algunos algoritmos de aprendizaje automático son descritos a continuación: [9]

- **Árboles de decisión:**

Es un método de aprendizaje supervisado, particularmente implementa una clasificación supervisada, en el que las instancias u objetos son descritos por un conjunto de propiedades. Como su nombre lo indica, la clasificación se realiza mediante la implementación de una estructura tipo árbol en el que cada nodo interno evalúa una cierta propiedad, el resultado de esta evaluación determina el camino a seguir en el proceso hasta llegar a una hoja, donde se determina la decisión, sin embargo, cuando la hoja determina la etiqueta de una clase, se considera que el método es un árbol de clasificación.

Cada camino que conduce desde la raíz hasta una hoja puede ser considerada una regla, de modo que los conjuntos de las reglas obtenidas a través del árbol proveen el aprendizaje completo. La Figura 2 muestra el ejemplo de un árbol de decisión en el que se evalúa una serie de características climáticas para determinar si alguien decide jugar tenis o no.

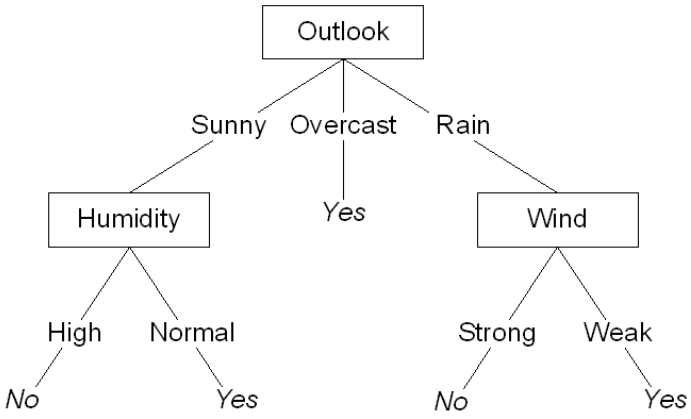


Figura 2: Árbol de decisión para predecir el juego de tenis

De éste árbol se puede derivar una regla, como una disyunción de conjunciones, que permite predecir cuándo una persona jugará tenis:

(Outlook = Sunny And Humidity = Normal)

Or (Outlook = Overcast)

Or (Outlook = Rain And Wind = Weak)

- **Método Naive-Bayes**

Este modelo, basado en el teorema de Bayes, constituye un modelo simple de clasificación que resulta ser de gran utilidad en el caso de trabajar con conjuntos de datos considerablemente grandes.

Se asume que esta estrategia es ingenua dado que considera cada característica, del conjunto de observaciones, como independiente de las demás; en general, la fórmula derivada del teorema de Bayes se expresa a continuación: [9]

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

En esta fórmula, los componentes son:

P(h): Probabilidad que la hipótesis sea cierta

P(D): Probabilidad independiente de la hipótesis

P(h|D): Probabilidad de la hipótesis h, dado los datos D

P(D|h): Probabilidad de D, dada la hipótesis h

Lo anterior conlleva a resaltar que, si bien este método permite tener predicciones rápidas y que el hecho de asumir la independencia conduce a mejores clasificaciones que otros métodos, por otra parte, su mejor rendimiento se da en el caso de tener características categóricas, además que en algunos casos se podrían tener probabilidades 0 cuando alguna característica no haya sido observada.

- **Algoritmo K-NN**

Este método, también conocido como el método de “los k vecinos más cercanos” permite clasificar una instancia encontrando, entre los datos de entrenamiento, las *k* instancias más cercanas a esta. Para lograrlo, es necesario definir una métrica de distancia entre instancias, por ejemplo, la distancia euclidiana, y a continuación se usa una medida estadística, como la mediana o la moda, para predecir la clasificación.

Un factor importante para el éxito de la técnica está en la elección del valor *k*, el cual depende fundamentalmente de la distribución de los datos en el espacio característico, por lo cual una buena elección puede marcar la diferencia entre la reducción del efecto ruido en el resultado de la clasificación o, por el contrario, hacer diferencias entre clases que son parecidas o entre instancias que corresponden a una misma clase.

- **Máquinas de vectores de soporte**

Es una técnica de aprendizaje supervisado, muy utilizada en aplicaciones de reconocimiento de lenguaje natural, también son conocidas como máquinas kernel debido a que pertenecen a un conjunto de algoritmos de Machine Learning denominados métodos kernel.

La técnica consiste en formar una superficie de decisión a partir de la construcción de un hiperplano óptimo, de tal manera que el soporte para la ubicación óptima de la superficie de decisión son los vectores de soporte, los cuales referencian un subconjunto de las observaciones de entrenamiento. [10]

En el entrenamiento se ejecutan dos fases: la primera corresponde a la transformación de los datos de entrada, o predictores, en un espacio de características dimensional; y en la segunda fase se resuelve un problema de optimización que se ajuste a un hiperplano óptimo para clasificar las características transformadas en dos clases

- **Redes Neuronales tipo perceptrón multicapa**

Uno de los objetivos en los que la Inteligencia Artificial se ha enfocado ha sido el de emular la forma en que el cerebro procesa información a través de redes de neuronas, por esta razón las creaciones de redes neuronales artificiales han tenido un amplio desarrollo a través de los años en los que se han propuesto estructuras de redes neuronales como las redes con alimentación hacia delante y las redes recurrentes [11].

En el caso particular de las redes con alimentación hacia delante se organizan en capas, de esta forma cada unidad (nodo) sólo recibe información de las unidades de la capa que la precede. Dentro de este tipo de estructura se encuentran las redes tipo perceptrón, o red neuronal de una sola capa, en la que las entradas están conectadas directamente a las salidas.

Por otra parte, están las redes neuronales tipo perceptrón multicapa en las que hay una, o varias, unidades ocultas.

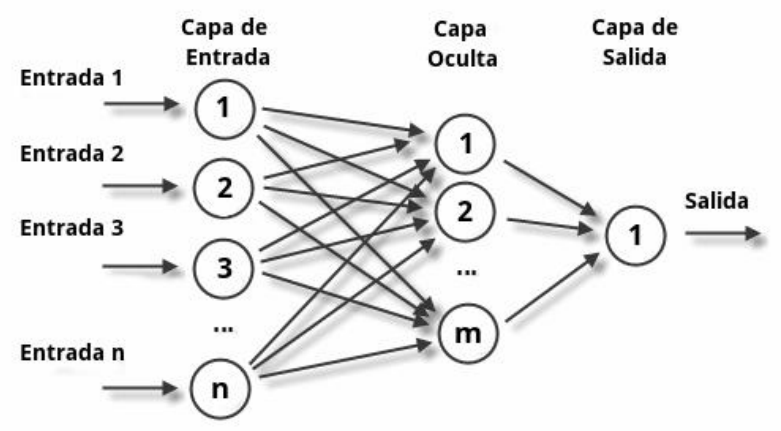


Figura 3: Red Neuronal Perceptrón Multicapa

La estructura de una red perceptrón multicapa se compone de una capa de entrada, capas ocultas y capa de salida, como se ve en la figura 3; éstas están compuestas de neuronas que reciben datos, los procesan mediante funciones matemáticas, y envían la información procesada hacia otras neuronas [12]. La ventaja de usar una red neuronal multicapa es la posibilidad de resolver problemas de clasificación de patrones no separables linealmente, debido a que el espectro de hipótesis que puede representar esta red es más amplio que en el caso de las redes de una sola capa.

- **Redes Profundas**

Las redes profundas, también conocidas como aprendizaje profundo, son redes neuronales con más de dos capas diseñadas con el fin de procesar datos de manera compleja [13]; cada capa realiza tipos específicos de clasificación y ordenación, en lo que se convierte uno de los principales usos de este tipo de redes al hacer aprendizaje automático tratando con datos no etiquetados o no estructurados.

Según la arquitectura de las redes neuronales profundas, unas importantes características suyas aprenden sin supervisión, dado que sólo necesitan datos y ella las características con las que debe etiquetarlos.

- **Redes Neuronales Recurrentes LSTM**

Las redes neuronales recurrentes (RNN) son un tipo especial de red neuronal que, a diferencia de las redes neuronales convolucionales, permiten recordar valores previos debido a que incluyen conexiones que permiten retroalimentación entre las neuronas en las capas; de esta forma las funciones de activación pueden actuar tanto hacia adelante como hacia atrás.

Estas redes son supremamente útiles en los casos en los que se deba procesar datos secuenciales debido a que esta memoria interna, llamada Memory Cell, o neurona recurrente, se encarga de preservar un estado a través del tiempo. [14]

Las redes Long-Short Term Memory (LSTM) son un tipo especial de las RNN que se caracterizan por garantizar que la memoria esté disponible por un mayor período de tiempo, en otras palabras, pueden capturar dependencias a largo plazo en una secuencia manteniendo actualizada su memoria debido a la posibilidad de realizar tareas de lectura, escritura y eliminación de información [15]. En este sentido, las unidades de memoria funcionan como válvulas que se abren o se cierran según se identifica que una característica en la secuencia debe ser recordado o no.

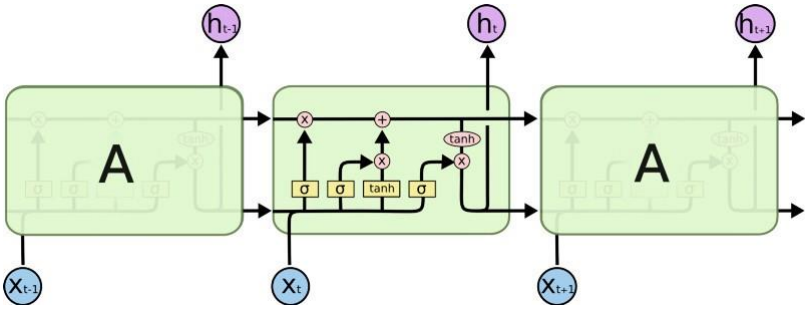


Figura 4. Arquitectura de una Red LSTM

Como se observa en la figura 4, la arquitectura LSTM está organizada en celdas, con una variable de estado interna que es modificada al pasar de una celda a otra. Cada celda recibe una entrada y produce una salida, de tal forma que la red aprende qué parte de la salida

anterior y de la entrada debe mantener y qué componente del estado interno debe enviar a la salida.

2.1.4 Corpus de datos

La conformación de un corpus de datos es una tarea fundamental a la hora de analizar datos en un proceso de investigación. El concepto de corpus de datos puede definirse como un conjunto de ejemplos reales, que deben corresponder a una muestra representativa de los diferentes casos o situaciones que están en el contexto del objeto de estudio; de esta manera se puede considerar que un corpus es representativo cuando contiene información suficiente con respecto a la variabilidad y las propiedades en el ámbito del problema tratado.

La obtención del corpus puede darse por diferentes alternativas como la generación automática o por la elección a partir de un repositorio, usando aleatoriedad o muestreo probabilístico; y su representación puede hacerse por un modelo como el booleano, el de frecuencias o TD-IDF (Frecuencia de término- Frecuencia inversa del documento). [16]

Una vez conformado el corpus, se debe estructurar, lo que implica tener una representación de los datos (o documentos) que contenidos en él con el fin de poder analizarlos.

2.1.5 Medida del desempeño de los clasificadores

- **Matriz de Confusión**

La matriz de confusión es un instrumento utilizado para analizar los resultados de un sistema de clasificación en el que se compara, para un cierto conjunto de datos de entrenamiento, la predicción hecha por el clasificador versus la clase a la que realmente estos datos pertenecen. En la figura 5 se presenta la estructura de esta matriz, en la que ***w*** representa los verdaderos positivos, ***x*** representa la cantidad de falsos negativos, ***y*** corresponde a la cantidad de falsos positivos y ***z*** indica la cantidad de verdaderos negativos.

		Predicción	
		Positivo	Negativo
Valor real	Positivo	*w*	*x*
	Negativo	*y*	*z*

Figura 5: Estructura de la matriz de confusión

De la matriz de confusión pueden obtenerse otros datos que resultan importantes para el análisis como:

- Precisión (P): proporción de predicciones correctas con respecto al total.
  P= $(w + z) / (w + x + y + z)$ (En inglés: Accuracy - ACC)
- Precisión Positiva (PP): proporción de ejemplos clasificados correctamente como positivos. PP= $w / (w + x)$ (En inglés: True Positive Rate - TPR o Sensitivity)
- Precisión Negativa (PN): proporción de ejemplos clasificados correctamente como negativos. PN = $z / (y + z)$ (En inglés: True Negative Rate - TNR o Specificity)
- Asertividad Positiva (AP): proporción de buena predicción para positivos.
  AP = $w / (w + y)$ (En inglés: Positive Predictive Value - PPV)
- Asertividad Negativa (AN): proporción de buena predicción para negativos.
  AN = $z / (x + z)$ (En inglés: Negative Predictive Value - NPV)

- **Análisis ROC**

El análisis ROC, o también curva ROC, es una herramienta que se utiliza para evaluar el nivel de precisión de un clasificador binario; aunque su utilidad ha estado basada en los sistemas de detección de señales, y más recientemente en la medicina para determinar la calidad de un procedimiento diagnóstico, en la actualidad es un mecanismo adoptado por los científicos de datos para analizar sistemas de decisión y hacer comparaciones entre diferentes clasificadores.

La curva ROC se construye a partir de la relación entre la razón de los verdaderos positivos (Sensibilidad) y la razón de los falsos positivos (Especificidad), esta relación permite tomar decisiones sobre modelos que puedan ser considerados óptimos en la medida en que esta curva se aproxime a 1. En la Figura 5 se presenta el ejemplo de una curva ROC. [17]

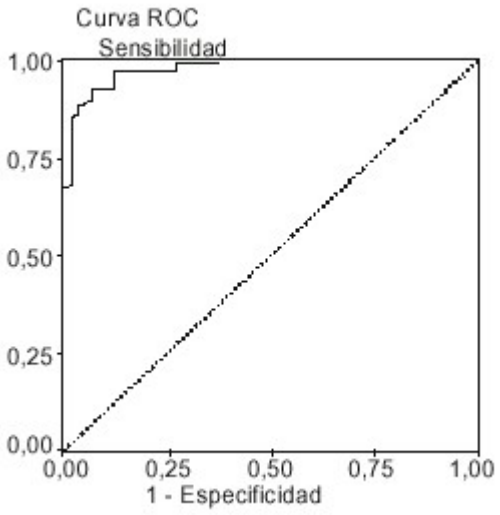


Figura 6: Ejemplo de curva ROC

2.2 TRABAJOS RELACIONADOS

En la literatura es posible encontrar amplia información acerca del plagio académico y de las estrategias implementadas por diferentes investigadores que han abordado el problema de la detección de fraude en documentos de texto, por ejemplo, en [18] enfocan la identificación del plagio en un texto a partir de dos etapas: la identificación de fuentes potenciales y en la determinación de la cantidad de texto coincidente en cada fuente.

De acuerdo con lo anterior, diferentes empresas han desarrollado herramientas para la detección de plagio en documentos de texto. De acuerdo al trabajo presentado en [19], en muchos casos estas herramientas prometen efectividad en los resultados más allá de lo que

realmente pueden cumplir, teniendo una cantidad considerable de falsos positivos y de falsos negativos debido a diversas razones, como por ejemplo la imposibilidad de revisar absolutamente todas las fuentes.

En el caso particular de la detección de plagio en el software, la literatura y las experiencias son menos amplias, bien en contenido o en la variedad de técnicas desarrolladas; en muchos casos, las estrategias empleadas para resolver este problema toman elementos de la detección de plagio en documentos de texto, en este sentido es preciso hacer referencia al algoritmo Winnowing, una estrategia presentada en [20], que se fundamenta en la creación de huellas dactilares de documentos para el análisis de similitud de textos; en este algoritmo de detección de copia, los documentos son representados por medio de n-gramas, y se hace uso de hashing para su almacenamiento, lo que garantiza la identificación de coincidencias entre pares de documentos con un costo controlado de memoria. Las características que se preservan son: insensibilidad al espacio en blanco, en el caso del software debe haber insensibilidad a los nombres de variables; supresión de ruido, que implica obviar términos comunes; y la independencia de la posición de bloques de funciones. En la primera característica se considera útil reemplazar los nombres propios (como nombres de variables y de funciones) por un identificador "V". Finalmente, la determinación de la similitud de dos códigos es determinada por la densidad de un algoritmo de huella dactilar que corresponde a la fracción esperada de huellas dactilares seleccionadas de entre todos los valores de hash calculados.

Otra alternativa es la planteada en [21], en la que se hace una revisión sistemática de la literatura asociada con el diseño de marcas de nacimiento de software y de las técnicas desarrolladas para comparar dichas marcas con el fin de identificar software plagiado o pirateado. Los resultados presentados dan cuenta de la necesidad de desarrollar tales técnicas para facilitar su utilización en la construcción de herramientas de detección de plagio.

Los resultados otros trabajos en los que se ha abordado este problema, los cuales se relacionan a continuación:

- **Recuperación y clasificación instancias de plagio de código fuente**. [22]

El punto de partida en esta investigación es la representación de los documentos de código fuente por medio del árbol de sintaxis abstracta (AST), al igual que toma la idea de recuperación de información por medio de la bolsa de palabras (Bag-of-Words - BoW) y la construcción de n-gramas, sin embargo, a pesar de su eficiencia, esta estrategia de naturaleza no supervisada puede dar un significativo número de falsos positivos debido al uso de librerías o al uso común de nombres de variables (como las i,j,k … usadas en los ciclos). Para mejorar su efectividad, se decide aplicar un clasificador supervisado (entrenado previamente con características extraídas de pares de código fuente plagiado) en los documentos recuperados mejor clasificados por BoW.

La etapa experimental es hecha con el repositorio de SOCO (International competition on detection of SOurce COde re-use), que se llevó a cabo como parte del Foro de Evaluación de Recuperación de Información. Los documentos en este conjunto de datos son casos de plagio de código fuente en la vida real. Los experimentos confirmaron que los enfoques que utilizan la estructura del código fuente son más efectivos que aquellos enfoques basados en la recuperación de información por bolsa de palabras (BoW), sin embargo el uso de la clasificación supervisada, aplicada en el conjunto de candidatos arrojados por los métodos tradicionales, mejoró significativamente la precisión en la identificación de plagio cuando se probó sobre el conjunto de pares seleccionados del repositorio SOCO.

- **Detección eficiente de plagio de código fuente basado en clustering usando PIY**

El enfoque “Program it yourself” (Prográmelo usted mismo - PIY) [23] integra dos estrategias: procesamiento paralelo, usando n-gramas optimizados para arquitecturas paralelas, y agrupación de datos. Las pruebas pudieron demostrar que si bien, este enfoque mejoraba notablemente la precisión de la estrategia MOSS (Medida de similitud del

software), la cual ha sido un referente importante en la detección de plagio del código fuente; por otra parte, la eficiencia algorítmica podría considerarse óptima cuando los datasets sobre los que se ejecutaba no eran muy grandes; de este modo, se ha demostrado empíricamente que supera al MOSS en la precisión de detección. Al utilizar el procesamiento paralelo y la agrupación de datos, PIY también es capaz de mantener la precisión de detección y los tiempos de ejecución razonables, incluso cuando se utilizan depósitos de datos extremadamente grandes.

- **Detección de plagio de código fuente escalable mediante el uso de vectores de código fuente**

La estrategia para la detección de plagio en el código fuente presentada en [24] es una base de datos de vectores característicos, los cuales se forman del árbol de sintaxis abstracta de los códigos a ser analizados. En un paso siguiente, los vectores son agrupados mediante un algoritmo de clasificación con el fin de hallar coincidencias dentro de los vectores en cada grupo específico las cuales son puntuadas como una medida de similitud, de tal forma que dos fragmentos de código fuente se considerarán coincidentes si sus vectores característicos son los mismos.

A diferencia de otras experiencias en el mismo sentido de la detección de plagio de código fuente, en las que la búsqueda de plagio se hace sobre un conjunto abierto de datos, la solución que se propone en este caso se enfoca en un conjunto de datos derivados de una asignación específica almacenados en una base de datos preconstruida.

- **Comparación de tres herramientas de similitud de código fuente para detectar plagio estudiantil**

En [25] se presentan los resultados de una investigación enfocada en la comparación de tres herramientas popularmente usadas para la detección de plagio académico en el código fuente: JPLAG, SIM y MOSS, las cuales fueron probadas en los trabajos académicos desarrollados en C++ por estudiantes de tercer y cuarto año en los cursos de algoritmos y estructuras de datos de la Universidad Tecnológica de Sídney. El objetivo del trabajo de investigación

estuvo enfocado en determinar el nivel de coherencia en los resultados arrojados por las tres herramientas seleccionadas en este trabajo de investigación.

En los resultados individuales se detecta que en el caso de JPLAG, en el que las entradas son convertidas en cadenas de token, los hallazgos son determinados por la longitud de tales cadenas coincidentes y, particularmente en las pruebas realizadas, hubo un 20% de casos que no fueron analizados exitosamente.

Por otra parte, SIM convierte el código fuente de cada uno de los programas en un conjunto de token, en una tabla de referencia se almacenan los token de uno de los archivos, mientras que los token del segundo archivo son agrupados en secciones representando módulos del programa lo cuales son alineados de acuerdo con la secuencia del primer programa, y de esta manera encontrar las similitudes. Esta herramienta es robusta en cuanto a que las variaciones de nombres y orden de las funciones no afectan el resultado.

Finalmente, el algoritmo MOSS, Medida de Similitud del Software, divide el documento con el código fuente en un conjunto de subcadenas contiguas llamadas n-gramas con el fin de construir una "huella digital" del documento y, de este modo, determinar el nivel de similitud de los documentos comparados de acuerdo con sus huellas digitales. Aunque con este algoritmo el número de hallazgos fue mayor frente a los otros dos, en el análisis realizado sobre la precisión positiva se observa que el algoritmo SIM tuvo un mejor desempeño con una precisión positiva del 83% frente al 81% y 73% de los algoritmos MSS y JPLAG respectivamente.

A manera de resumen, se puede señalar que los antecedentes encontrados en trabajos de investigación que abordan el problema de la detección de fraude en el código fuente de programas de computador están enfocados a la comparación por pares, la cual apunta en dos direcciones: por un lado el uso de estrategias para la detección de plagio en documentos escritos como la representación por medio de n-gramas, y por otro lado la representación del código a partir de los árboles sintácticos y su almacenamiento en estructuras de datos, como tablas hash o vectores, para finalmente determinar el grado de similitud de los mismos.

2.3 DEFINICIÓN DE TÉRMINOS

Los términos que se definen a continuación son ampliamente utilizados en Inteligencia Artificial, particularmente en el análisis de los resultados producidos por clasificadores binarios.

- **Verdaderos Positivos:** Son los ejemplos positivos que fueron clasificados correctamente como positivos. (TP - True Positive)
- **Falsos Positivos:** Son los ejemplos que fueron clasificados como positivos cuando, en realidad, eran negativos. (FP - False Positive)
- **Verdaderos Negativos:** Corresponde a la cantidad de ejemplos negativos que fueron clasificados correctamente como negativos. (TN - True Negative)
- **Falsos Negativos:** Es la cantidad de ejemplos positivos que fueron clasificados incorrectamente como negativos. (FN - False Negative)

3. INGENIERÍA DEL PROYECTO

En el presente capítulo se presenta el proceso de desarrollo del proyecto partiendo del diseño general del sistema, el cual sirvió como marco de referencia para la toma de decisiones relacionadas con los métodos y algoritmos usados, de igual manera se hace una descripción del corpus de datos seleccionado indicando sus características y la relación de éste con los objetivos del proyecto; se hace una descripción de las tareas de preprocesamiento llevadas a cabo y del modelamiento de la solución, mencionando las diferentes estrategias utilizadas e indicando los hallazgos obtenidos.

3.1 DISEÑO GENERAL DEL SISTEMA

La figura 7 muestra el diseño del sistema implementado, en éste se identifica como entrada la pareja de documentos que van a ser evaluados y que serán procesados por el clasificador que se describe en los métodos tratados en las sección 3.5; el resultado se visualiza como un informe de resultados con la clasificación asignada en términos de una posible existencia de reuso de código (mayor nivel de similitud) en los documentos o no, así como la precisión de la clasificación.

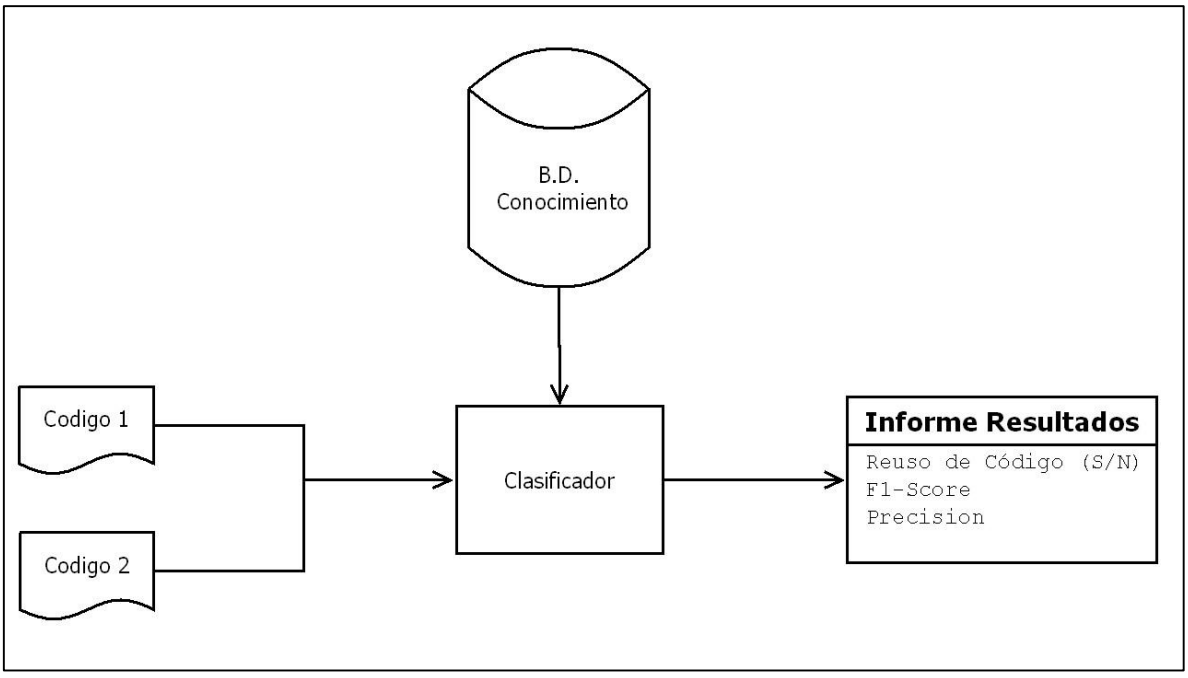


Figura 7: Diseño general del sistema

3.2 CONFORMACIÓN DEL CORPUS DE DATOS

El proceso de selección de documentos para la conformación del corpus de datos implicó una revisión de diferentes alternativas que pudieran ser consideradas para la ejecución de las tareas que posteriormente se desarrollaron y que estuvieran alineados tanto con el alcance del proyecto como con los objetivos que se propusieron. En este orden de ideas fueron considerados los siguientes criterios: contar con un conjunto de archivos de código fuente donde se garantizara la existencia de pares de ellos con diferentes niveles de código compartido (reuso de código), contar con la identificación de las parejas que tienen código compartido con el fin de llevar a cabo un proceso de aprendizaje supervisado, el contexto de los programas se limitara a un entorno académico, lo que supone unas condiciones con mayor nivel de dificultad debido a que naturalmente estos programas tendrán un nivel de similitud, y en lo posible, contar con una fuente de datos que hubiese sido usada en otros proyectos similares con el fin de realizar comparaciones en la etapa de análisis de resultados.

De acuerdo a lo anterior, se determinó que un recurso que se ajusta a las características anteriormente descritas correspondía al corpus de SOCO [26] que es una competencia centrada en la detección de reutilización de código fuente; sus primeras versiones en los años 2014 y 2015 se centraron en la detección en código fuente monolingüe y multilingüe respectivamente, sin embargo, para el desarrollo del proyecto se trabajó en los datos de la primera versión con la respectiva autorización de sus autores.[1]

Este corpus se conformó finalmente con un total de 259 archivos fuente en lenguaje java, con lo que se formaron 33411 parejas entre las que se encuentran 84 pares de archivos con algún nivel de reuso de código, haciendo claridad que algunos archivos aparecen en más de una pareja.

[1] Autorización recibida el 4 de marzo de 2020 vía correo electrónico por parte de Francisco Rangel, autores citados en [25]

3.3 PREPARACIÓN DE LOS DATOS

Con el fin de reducir la posibilidad encontrar coincidencias en archivos debido a la existencia de "*stopwords"* (palabras vacías o sin significado), se procedió a preprocesar los archivos del corpus convirtiendo todas las palabras a minúsculas, eliminando espacios en blanco, comentarios, tabulaciones, signos de final de instrucción (punto y coma); sin embargo se decide mantener otros símbolos como los marcadores de bloque (llaves), marcadores de parámetros (paréntesis) y marcadores (corchetes). La razón para mantener ciertos símbolos se justifica en la idea que estos marcadores son necesarios para mantener dentro del análisis las características estructurales de cada archivo particularmente por el sesgo que puede generarse debido a la frecuencia de palabras propias del lenguaje de programación, las cuales no pueden ser consideradas *stopwords*.

Todas las actividades relacionadas con el preprocesamiento se muestran en la figura 8

Figura 8: Actividades de preprocesamiento.

El proceso comienza con la lectura del archivo, separando las cadenas de caracteres que se encuentran en él con el objetivo de eliminar espacios en blanco y tabulaciones, en el análisis de cada cadena de caracteres se identifican los marcadores de comentarios y se ignoran, se separan los símbolos que están concatenados con palabras u otros símbolos, los números y las cadenas escritas entre comillas se representan con las palabras número y texto respectivamente dado que no se pretende hacer un análisis léxico, por lo que el valor del número o la cadena es irrelevante.

Como resultado del proceso descrito, cada archivo fuente queda representado en una lista como un flujo de palabras y símbolos que serán normalizados en la siguiente etapa.

3.4 NORMALIZAZCIÓN DE LOS DOCUMENTOS

Teniendo en cuenta que la reutilización del código fuente puede darse en diferentes niveles, por ejemplo puede limitarse a eliminar o modificar los comentarios, cambiar los nombres de las variables, intercambiar el orden de instrucciones o de bloques de función, etcétera; cada una de estas estrategias, y otras no mencionadas en este documento, revelan diferentes niveles de dificultad que deben ser tenidos en cuenta a la hora de analizar la similitud del código de dos programas de computador; estos niveles han sido abordados por varios autores y sus conclusiones se encuentran en documentos como [27] citado en [28]. En ellos se mencionan 7 niveles de modificación de código fuente que van desde el nivel 0: sin modificaciones, hasta el nivel 6: Cambios de expresiones; todos los niveles y sus características se representan en la figura 9.

Figura 9: Niveles de modificación de código.

Tomado de [27]

En el contexto de este proyecto, enfocado principalmente al análisis de similitud de código en el ámbito académico, y con base en la experiencia de un grupo de docentes consultados,

se encuentra una mayor tendencia de modificación de código en los niveles 0 a 3, un poco menos en el nivel 4 y más esporádicamente en los noveles 5 y 6. Por esta razón el proceso de normalizar documentos debe garantizar que no se pierdan características de estilo más relacionadas con los primeros niveles.

En concordancia con lo mencionado anteriormente, la definición del proceso de normalización de los documentos para este proyecto tomó como base la idea de tokenización inspirada en el concepto de árbol de análisis sintáctico, sin abarcar más allá que la conversión de cada palabra resultante del preprocesamiento en su respectivo token.

La tarea de tokenización se realizó teniendo consideraciones con las que se pretendió atenuar el impacto de algunos niveles de modificación, algunas de las cuales se relacionan a continuación:

- Todo número, independiente de su valor y del tipo particular, se representa como número.
- Todo operador lógico se representa como OP_Relacional, lo que permite reducir la sensibilidad a cambios del estilo “A > B”, por “B < A”
- Los operadores aritméticos y aritméticos con asignación se asocian con el mismo token, por ejemplo +, +=, ++ se asocian con el token “incremento”.
- En general, todas las cadenas de caracteres escritas entre comillas se asocian con el token “texto”

En la tabla 1 se presenta un ejemplo de tokenización en un fragmento de código.

Tabla 1 Ejemplo de tokenización de documento

CÓDIGO	TOKENS

String passwords=”ABC123”;
int twoChar=0;
if (twoChar < 10)
  twoChar += 1;
return 0;
	Td_String identificador Asignación Texto
	Td_num identificador Asignación Numero
	Condicional identificador OpRelacional Numero
	Identificador incremento Numero
	Retorno Numero

Como se mencionó en la sección anterior, algunos símbolos se mantienen en el proceso de tokenización, como es el caso de los marcadores de bloque (llaves) con el fin de conservar

las características estructurales de los documentos, de tal forma que en el análisis de similitud no se compararan instrucciones de bloques de función diferentes, esto garantiza el control sobre los niveles 4 y 5 de modificación del código.

En la tabla 2 se presenta una clasificación de los token de acuerdo con su naturaleza, lo que permitió dar un manejo más eficiente para la identificación de las características estructurales y estilísticas de los documentos, siendo una característica necesaria en la etapa de formulación de los modelos de análisis de similitud.

Tabla 2: Clasificación de token en categorías.

CATEGORÍA DE TOKEN	DESCRIPCIÓN
Palabras Reservadas	Conjunto de palabras propias del lenguaje como las que describen expresiones condicionales, bucles, tipos de datos, entre otras.
Operadores	Separa los símbolos que describen operaciones aritméticas, lógicas, de bit, de asignación.
Delimitadores	Corresponde al conjunto de token asociados con los marcadores de bloques de función o de estructuras, como el caso de las llaves.
Clase	Esta categoría incluye todas las cadenas identificadas como nombres de clases y nombres de métodos.
Otros	Agrupa los token identificados como literales, texto, números y, en general, todos los valores calculados o asignados a alguna variable.

La lista completa de los token agrupados de acuerdo a su categoría puede consultarse en el anexo 2.

3.5 MODELAMIENTO DE LA SOLUCIÓN

El modelamiento de la solución se divide en dos fases: la primera fase se centra en la estimación de la similitud entre dos códigos fuente; la segunda fase se utilizan dos estrategias de clasificación supervisada para realizar un análisis de los resultados obtenidos; éstas estrategias correspondieron al método Naive-Bayes y la implementación de una red neuronal recurrente LSTM. La figura 10 representa el modelo de solución.

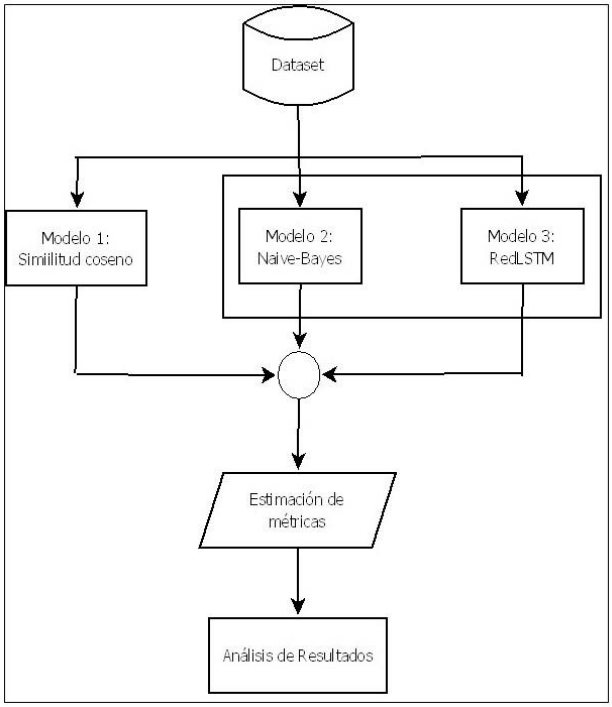


Figura 10. Modelamiento de la solución

Usando la clasificación de los niveles de similitud propuestos en [28], el modelo implementado clasifica la pareja de documentos de entrada como sospechoso o inocentes; se selecciona esta forma de clasificación debido a que existe un nivel lógico de incertidumbre por el cual no es apropiado catalogar una pareja de documentos como plagiados, de tal forma que la declaración de pares sospechosos puede brindar al usuario la posibilidad de hacer una revisión exhaustiva para tomar una decisión acerca del nivel de reuso de código en ambas entradas.

Por otra parte, no es competencia de este proyecto la identificación del documento original y del documento que reutiliza o plagia código, puesto que el alcance se enfoca en la determinación de la similitud de ambos documentos en función de indicar una posibilidad de plagio.

A continuación, se describen las fases del modelo de solución implementado:

3.5.1 Determinación de la similitud

En la revisión de la literatura para determinar la similitud de documentos, independientemente de que correspondan a documentos de texto o archivos de código fuente, es frecuente el uso de estrategias de recuperación de información (IR) y de análisis semántico latente (LSA). Partiendo de esta base teórica cuya efectividad ha sido ampliamente probada en diferentes trabajos de investigación que abordaron este mismo problema, como [29], se ejecuta el conjunto de actividades representadas en la figura 11 con lo que se busca tener una correcta estimación del nivel de similitud del código fuente en pares de documentos, teniendo presente que la ventana de observación se centra en documentos de proyectos académicos, lo que sugiere la existencia obvia de un cierto grado de similitud debido a que el escenario es controlado o limitado cuando el contenido de los documentos puede responder a un problema común propuesto por un profesor.

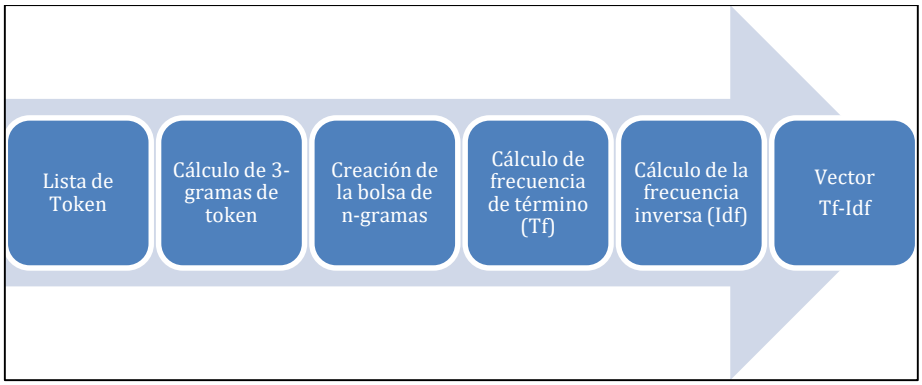


Figura 11: Preparación del espacio vectorial

Los conjuntos de actividades representadas en la figura 11 se describen a continuación:

- Cada pareja de documentos a ser comparados se normaliza de acuerdo a la estrategia planteada en la sección 3.3, lo que produce para cada uno una lista con los token identificados.
- Por cada lista de token se genera una nueva lista de 3-gramas de token; en esta tarea se realizaron estimaciones con n-gramas de diferentes tamaños, tomando como tamaño estándar el valor de n=3, partiendo de la idea que en la mayoría de los casos la sentencia del lenguaje tiene tres componentes.

- Usando una lista en Python, se conforma la bolsa de palabras (bolsa de trigramas) extrayendo los diferentes 3-gramas de token identificados en cada secuencia, obteniendo un total de 4579 3-gramas.
- En la siguiente etapa se calculan las frecuencias de término, obteniendo la cantidad de ocurrencias de cada 3-grama en cada documento.
- Dado que algunos 3-gramas pueden ser comunes en todos los documentos, se procede a calcular la frecuencia inversa de documento (idf) con el fin de minimizar el sesgo dado por términos que no permitan diferenciar entre los documentos.
- Finalmente, se calcula la frecuencia de ocurrencia de términos en la colección de documentos (tf-idf) con lo que se estima la relevancia de cada n-grama en la pareja de documentos. Esta información se interpreta como una representación vectorial de cada documento para el cálculo de la medida de similitud; dicha representación vectorial se da en un espacio vectorial de 4579 dimensiones correspondientes a cada uno de los diferentes 3-gramas obtenidos.

Como se mencionó anteriormente, el resultado del proceso descrito se interpreta como un vector de frecuencias en un espacio d-dimensional; la figura 12 muestra un fragmento del vector tf-idf para un archivo fuente, la primera columna representa la posición de cada trigrama en la bolsa de palabras y la segunda columna muestra el valor tf-idf.

```
In [4]: %run ComparaBolsa.py

Archivos: 0 - 0
0     0.000000
1     0.053319
2     0.000000
3     0.053319
4     0.053319
5     0.053319
6     0.053319
7     0.053319
8     0.053319
9     0.053319
10    0.053319
11    0.053319
12    0.000000
13    0.053319
14    0.053319
15    0.053319
```

Figura 12. Ejemplo de un vector Tf-Idf para un documento

Una vez definido el espacio vectorial, a partir de la representación de los vectores tf-idf,, se procede a la estimación de la similitud usando en primer lugar la similitud coseno representada en la figura 13

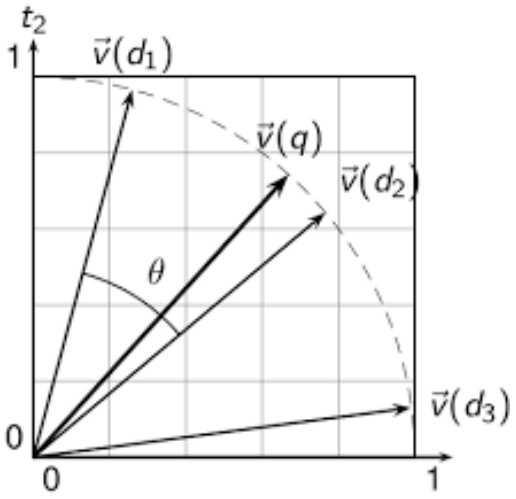


Figura 13. Representación de la similitud coseno [30]

Bajo este modelo, cada documento está representado como un vector d_i, de tal manera que cuando dos documentos son iguales, el ángulo entre ellos será 0, por el contrario, entre menos similitud haya entre los documentos, el ángulo entre los vectores que los representan será mayor aproximándose a 1; de tal forma que la similitud coseno calcula el valor del coseno del ángulo formado por los dos vectores y la similitud entre ellos se calcula como:

$$\boldsymbol{Sim(d_1, d_2) = 1 - COS(\theta)}$$

De acuerdo con lo anterior, se decide validar el resultado de la medición en los documentos que conforman el corpus, de tal forma que se pudiera encontrar un valor de similitud que permita clasificar las parejas de código identificando, con la mayor precisión posible, entre aquellas que corresponden a programas con reuso de código y aquellas que no se consideran con código reusado (o compartido), esto teniendo en cuenta la estrategia de tokenización de las instrucciones en cada documento descrita en la sección 3.4, puesto que al llevar su representación a un nivel no sensible a los primeros niveles de reutilización, sumado al espectro controlado de las soluciones, se sabía de antemano que podrían generarse vectores con dirección muy similar, desencadenando un alto número de falsos positivos.

Finalmente, con el objetivo de controlar el riesgo de aparición de falsos positivos, se decide establecer un valor de decisión (umbral) con base en la similitud coseno calculada en la etapa

de entrenamiento, de tal forma que aquellos pares de documentos que no logren superar este valor de decisión se consideraron como casos sospechosos. En la tabla 3 se especifican los cinco valores de decisión con los que se realizó el análisis sobre el corpus de datos, los cuales son representados como puntos (P1, P2, p3, P4, P5) para facilitar su interpretación.

Tabla 3. Puntos de decisión de similitud

PUNTO	Umbral de Similitud
P1	0,750
P2	0,625
P3	0,500
P4	0,375
P5	0,250

3.5.2 Análisis ROC de la estrategia de Similitud Coseno

Con el fin de determinar el valor de similitud que permite clasificar, con la mayor precisión posible, los pares de código fuente que son sospechosos de tener código compartido (reusado), se decidió hacer el análisis de la curva ROC formada por los resultados obtenidos al realizar la clasificación de los programas del corpus con los cinco puntos de decisión mencionados anteriormente; adicionalmente, con el propósito de lograr un mejor ajuste, se realiza el análisis por medio de la ejecución de 5 experimentos, de tal forma que las 67081 parejas de código fuente disponibles en el corpus son particionadas aleatoriamente en 5 bloques de 13416 parejas, los resultados obtenidos para cada uno de estos bloques se presentan a continuación:

Tabla 4. Experimento 1 - Resultados de clasificación por análisis de similitud coseno

FPR: False positive rate – TPR: True positive rate

EXPERIMENTO 1		
	FPR	TPR
P1	0,028	0,995
P2	0,015	0,981
P3	0,009	0,969
P4	0,004	0,957
P5	0,002	0,941

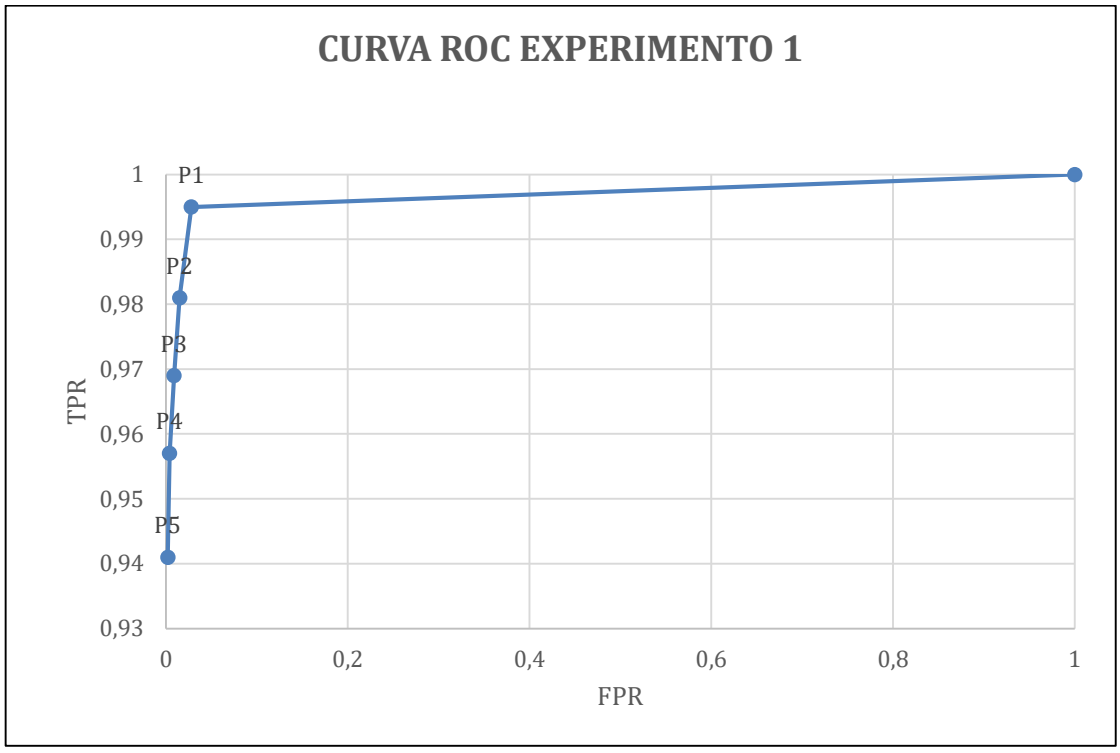


Figura 14. Curva ROC Experimento 1

Tabla 5. Experimento 2 - Resultados de clasificación por análisis de similitud coseno

EXPERIMENTO 2		
	FPR	TPR
P1	0,026	0,987
P2	0,013	0,979
P3	0,007	0,963
P4	0,003	0,955
P5	0,002	0,942

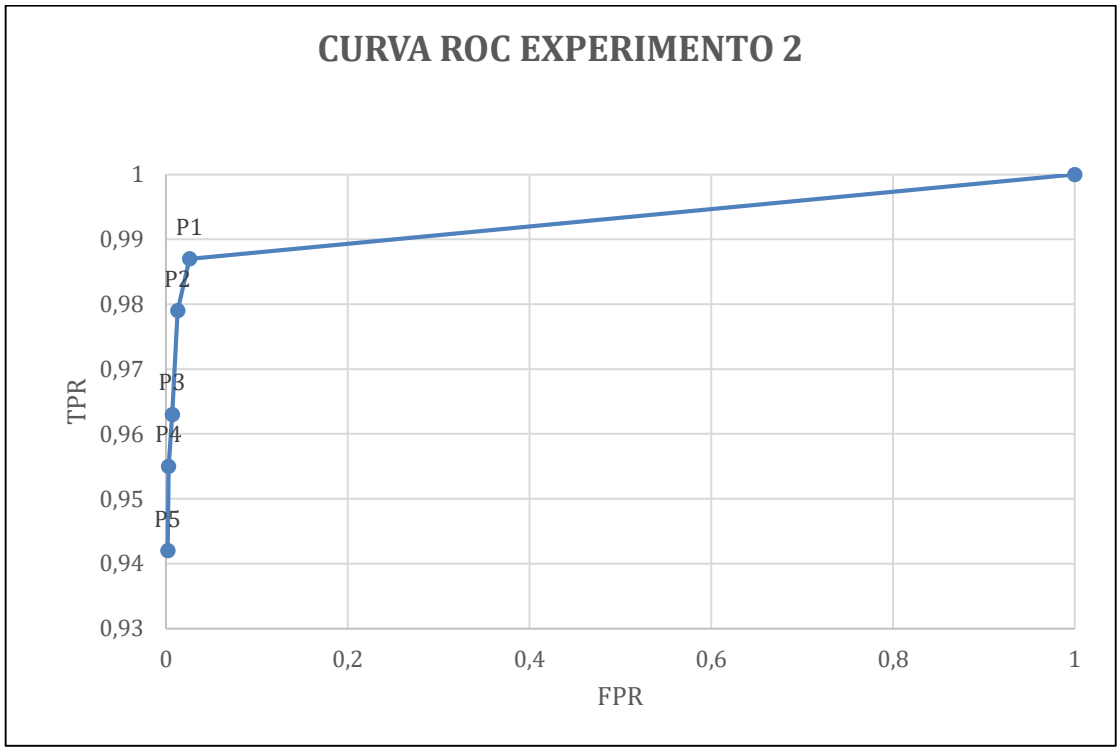


Figura 15. Curva ROC Experimento 2

Tabla 6. Experimento 3 - Resultados de clasificación por análisis de similitud coseno

EXPERIMENTO 3		
	FPR	TPR
P1	0,028	0,99
P2	0,014	0,979
P3	0,008	0,969
P4	0,003	0,961
P5	0,002	0,953

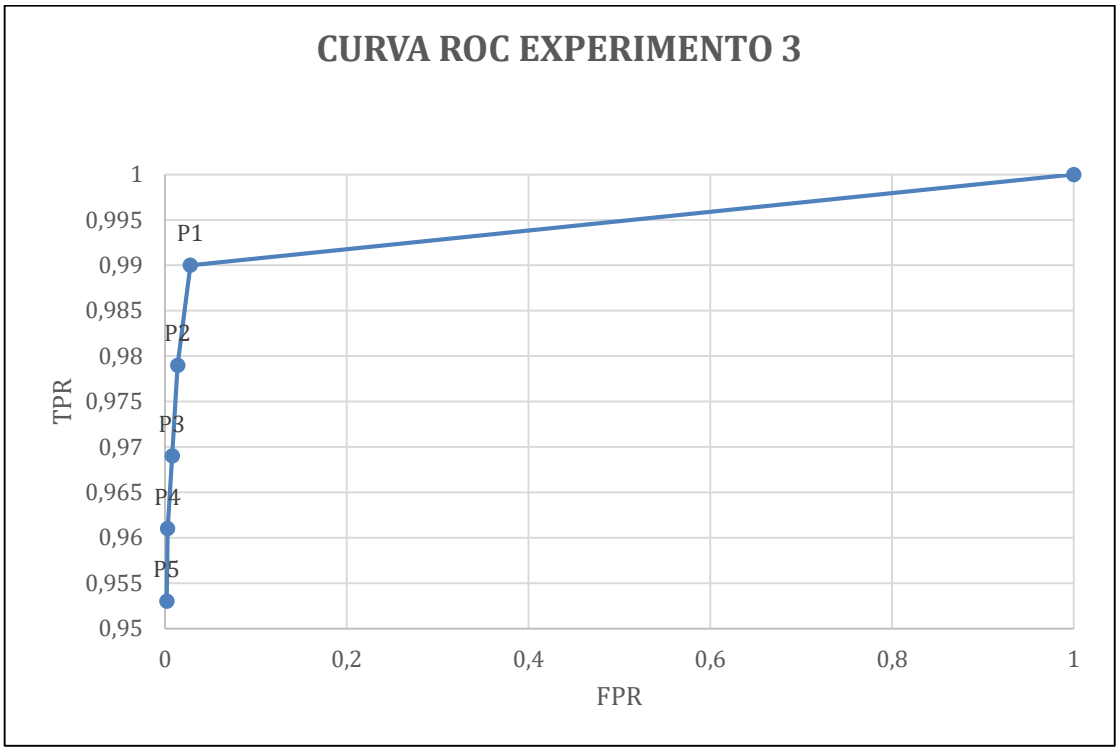


Figura 16. Curva ROC Experimento 3

Tabla 7. Experimento 4 - Resultados de clasificación por análisis de similitud coseno

EXPERIMENTO 4		
	FPR	TPR
P1	0,028	0,98
P2	0,016	0,965
P3	0,008	0,945
P4	0,003	0,935
P5	0,002	0,923

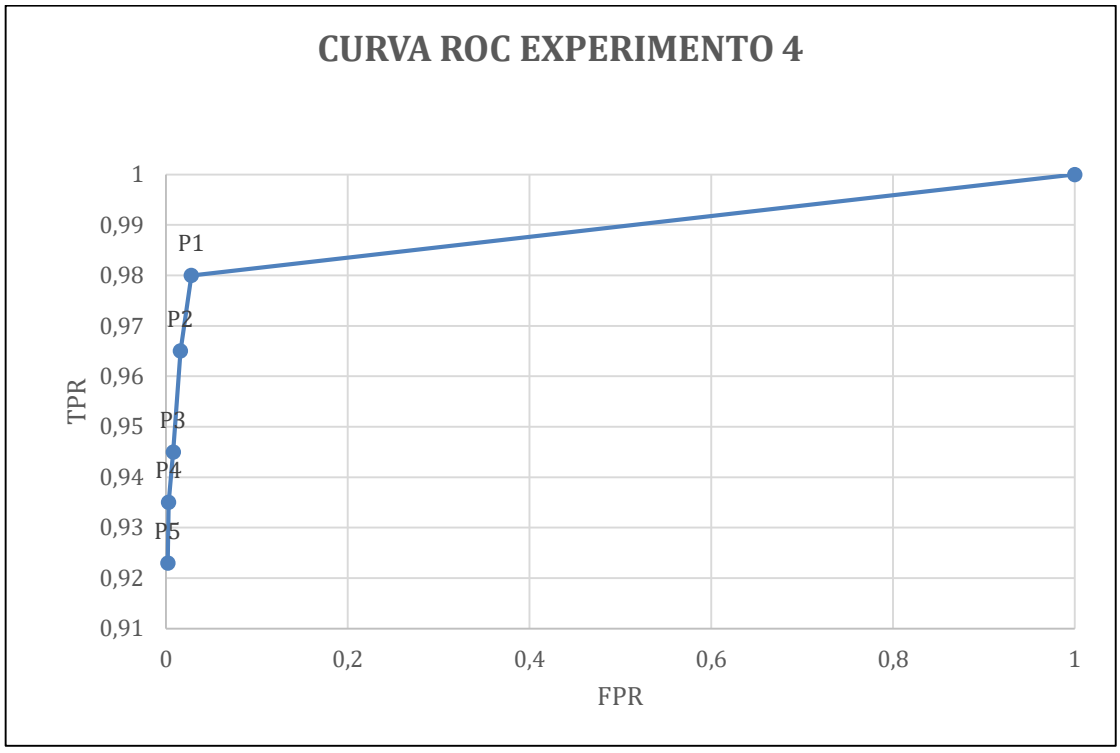


Figura 17. Curva ROC Experimento 4

Tabla 8. Experimento 5 - Resultados de clasificación por análisis de similitud coseno

EXPERIMENTO 5		
	FPR	TPR
P1	0,028	0,988
P2	0,015	0,973
P3	0,009	0,961
P4	0,004	0,949
P5	0,002	0,936

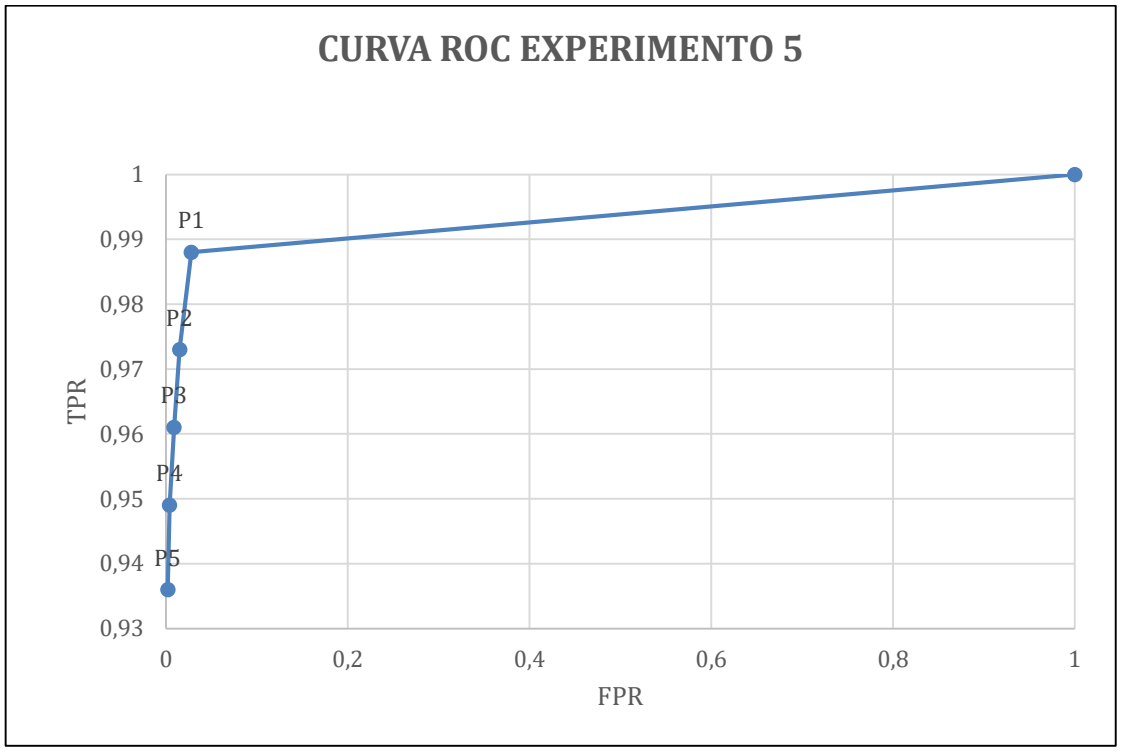


Figura 18. Curva ROC Experimento 5

Como se puede observar en los cinco experimentos, el punto de decisión que mejor clasifica las parejas de código es el punto P1 que corresponde a un umbral mínimo de similitud de 0.75; esto se explica dadas las estrategias de representación de código usadas desde el proceso de tokenización, toda vez que, al estar en un contexto controlado donde las soluciones tienden a parecerse en el sentido que abordan problemas similares, es necesario que la similitud en una pareja de códigos fuente sea suficientemente alta para ser considerada con código compartido (o reusado).

En las siguientes secciones se plantea el análisis realizado usando otras dos estrategias: clasificación por el método Naive- Bayes y clasificación basada en el entrenamiento de una Red Neuronal Recurrente LSTM (Log-Short-Term-Memory).

3.6 Estrategia usando el método de Naive-Bayes

Como se ha mencionado anteriormente, en la formulación del marco teórico, el método Naive.Bayes, o clasificador Gaussian Naive-Bayes, es un método de clasificación supervisado basado en las relaciones probabilísticas entre un conjunto de características observadas en los objetos del dataset, contando dentro de sus principales características tanto la rapidez como su eficiencia en el entrenamiento.

Con el propósito de usar diferentes enfoques para la identificación de similitud del código que permitan realizar un análisis de resultados que condujeran a conclusiones más precisas, se decidió la implementación de esta estrategia usando la librería *Sklearn* del lenguaje Python, la cual provee la clase GaussianNB lo cual facilitó las tareas de entrenamiento y pruebas, además de proveer otras herramientas como la reducción de la dimensionalidad.

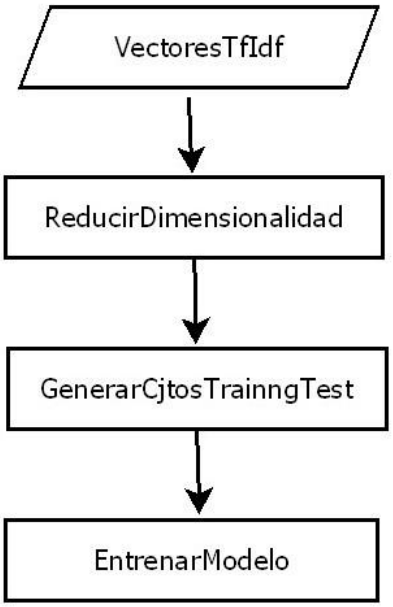


Figura 19. Diseño del Modelo Naive-Bayes

La figura 19 muestra el diseño propuesto, cuya entrada corresponde a la representación vectorial Tf-Idf de los códigos fuente, que resulta del proceso presentado en la figura 18. Ente caso el modelo recibe pares de vectores, correspondientes a los documentos a ser analizados, junto con la etiqueta que identifica para dicha pareja la clase correspondiente: 0

cuando no hay reuso de código, y 1 cuando existe reuso de código. Este formato permitió resolver de antemano el problema de la normalización de la entrada, toda vez que el valor de las diferentes características se encontraba en el rango [0,1].

Teniendo en cuenta que los vectores TfIdf de los trigramas de token que representan cada código fuente tienen una alta dimensionalidad, 4579 características de acuerdo a lo mencionado en la sección 3.5.1, significó tener igual cantidad de características para analizar con la consecuencia de enfrentarse a un problema que en Machine Learning es denominado "maldición de dimensionalidad" (o maldición de la dimensión) [31], particularmente dado que en un buen número de casos el valor de ciertas características tomaba el valor de cero a causa de los trigramas que no estaban presentes en alguno de los códigos (o en ambos).

La mitigación de los problemas de precisión derivados de la mencionada maldición de dimensionalidad implicó hacer una reducción de la dimensionalidad para lo cual se usó la técnica de selección de características usando el método selectKBest de la clase *feature_selection* contenida en la librería Sklearn de Python. Dicho método implementa el análisis de componentes principales (PCA), que corresponde a un método estadístico el cual busca simplificar espacios vectoriales multidimensionales a partir de la identificación de un conjunto más pequeño de características que permiten representar la misma información del espacio vectorial original. [32]. El entrenamiento del modelo se llevó a cabo con diferentes valores de dimensionalidad, cuyos resultados se presentan entre las secciones 3.6.1 y 3.6.3.

Finalmente, para el entrenamiento y pruebas del modelo, se particiona el dataset en 70% para el entrenamiento y el 30% restante para las pruebas, lo que equivale a 23390 y 10025 parejas de código respectivamente.

3.6.1 Reducción de la dimensionalidad a 1000 características:

En la figura 20 se puede observar que la precisión obtenida tanto en el conjunto de entrenamiento como en el conjunto de prueba es del 98%, lo que sugiere, en principio, un rendimiento notablemente eficiente en el proceso de clasificación; sin embargo, al revisar las métricas derivadas de la matriz de confusión presentadas en la tabla 9, se observa que mientras que la especificidad está en línea con la precisión total, la precisión positiva (Recall) es desfavorable, lo que condujo a generar ensayos con otros valores de dimensionalidad para observar el comportamiento del modelo.

```
In [7]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/ModeloNaiveBayes.py',
wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Precisión en el set de Entrenamiento: 0.98
Precisión en el set de Test: 0.98
[[6541  125]
 [   7   10]]
```

Figura 20. Rendimiento Naive-Bayes con 1000 características

Tabla 9. Métricas Naive-Bayes con 1000 características

MÉTRICAS	
Accuracy	0.9802
Recall	0.5882
Specificity	0.9812

3.6.2 Reducción de la dimensionalidad a 100 características.

Al entrenar el modelo reduciendo el espacio vectorial a las 100 características principales, se puede evidenciar que aun cuando la precisión total disminuye ligeramente, la proporción de predicciones positivas incorrectas disminuye considerablemente con respecto al entrenamiento anterior (ver figura 21), lo que hace que la precisión positiva mejore según lo presentado en la tabla 10.

```
In [6]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/ModeloNaiveBayes.py',
wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Precisión en el set de Entrenamiento: 0.96
Precisión en el set de Test: 0.96
[[6383  283]
 [   2   15]]

In [7]:
```

Figura 21. Rendimiento Naive-Bayes con 100 características

Tabla 10. Métricas Naive-Bayes con 100 características

MÉTRICAS	
Accuracy	0.9574
Recall	0.8824
Specificity	0.9575

3.6.3 Reducción de la dimensionalidad a 50 características.

En el tercer ejercicio de entrenamiento, reduciendo la dimensionalidad a las 50 características principales, se obtuvieron valores similares a los obtenidos en el entrenamiento anterior, por lo que se decidió continuar con el modelo entrenado con este último valor de dimensionalidad. Tanto en la figura 22 como la tabla 11, se presentan los resultados obtenidos en este entrenamiento.

```
Plag
0    9989
1      34
dtype: int64
Index(['NG144', 'NG178', 'NG236', 'NG411', 'NG493', 'NG579', 'NG764', 'NG800',
       'NG911', 'NG926', 'NG942', 'NG964', 'NG998', 'NG1038', 'NG1086',
       'NG1101', 'NG1193', 'NG1333', 'NG1574', 'NG1611', 'NG1800', 'NG1861',
       'NG1907', 'NG1925', 'NG2226', 'NG2246', 'NG2306', 'NG2569', 'NG2856',
       'NG2909', 'NG3159', 'NG3184', 'NG3218', 'NG3236', 'NG3298', 'NG3416',
       'NG3528', 'NG3817', 'NG3859', 'NG3973', 'NG4038', 'NG4043', 'NG4220',
       'NG4321', 'NG4329', 'NG4402', 'NG4477', 'NG4519', 'NG4575', 'Scos'],
      dtype='object')
Precisión en el set de Entrenamiento: 0.96
Precisión en el set de Test: 0.96
[[6414  252]
 [   2   15]]

In [6]:
```

Figura 22. Rendimiento Naive-Bayes con 50 características

Tabla 11. Métricas Naive-Bayes con 50 características

MÉTRICAS	
Accuracy	0.9620
Recall	0.8824
Specificity	0.9622

A manera de resumen, en la figura 23 se puede observar la curva ROC en la que queda evidenciado que el mejor rendimiento se obtiene cuando el espacio vectorial se reduce a 50 características.

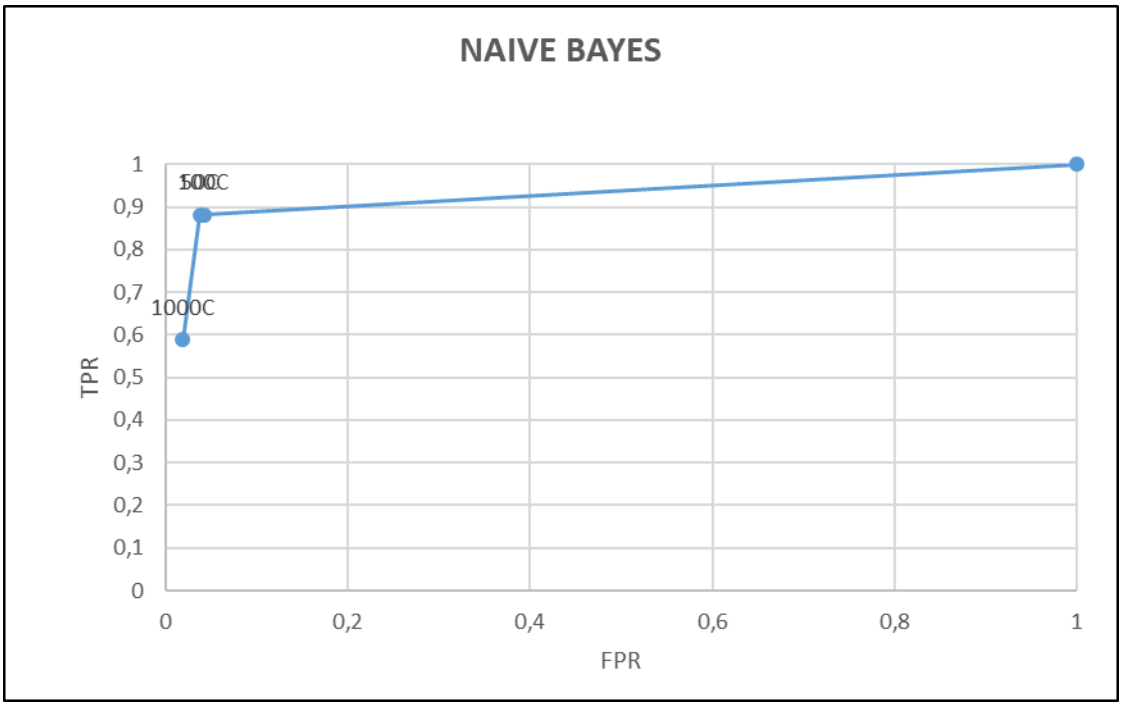


Figura 23. Curva ROC de reducción de dimensionalidad

3.6.4 Análisis de balanceo y sobreajuste.

Un factor que se tuvo en cuenta, una vez se obtuvieron los primeros resultados satisfactorios del proceso de entrenamiento de los modelos, estaba relacionado con la proporción de datos de cada clase con respecto al total de datos del dataset. En la revisión de este factor presentada en la figura 24, se puede observar que el dataset estaba desbalanceado, dado que la proporción de casos de parejas de código que no tienen reuso de código era muy alta con respecto a la proporción de las que si tenían código compartido: 99.7% versus 0.3%; de acuerdo a lo expresado en [33], un desbalanceo del dataset puede conducir a un sobreajuste (overfitting) del modelo.

```
In [29]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/ModeloNaiveBayes.py',
wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Valores del dataset:
(33411, 50)
0    33327
1       84
Name: Plag, dtype: int64
```

Figura 24. Dimensionalidad del dataset

Con el propósito de determinar si el rendimiento del modelo, cuyos resultados obtenidos hasta este punto se han presentado en secciones anteriores, podría verse afectado por un posible sobreajuste, se procedió a entrenar nuevamente el modelo aplicando previamente algoritmos de muestreo tanto de subsampling como de oversampling.

- **Muestreo con SubSampling:**
  En este caso se hizo uso de la clase NearMiss de la librería *imblearn.under_sampling*, el cual se basa en una idea similar al método de clasificación de los *k-vecinos más próximos*, en el sentido de reducir observaciones de la clase mayoritaria a partir de la identificación de las observaciones más cercanas entre sí. La figura 25 muestra el caso más extremo de reducción de observaciones de la clase mayoritaria a la misma cantidad de observaciones de la clase minoritaria para el entrenamiento, mientras que en el caso del conjunto test no se realizó algún tipo de muestreo.

```
In [2]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/
ModeloNaiveBayesSubSampling.py', wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Precisión en el set de Entrenamiento: 0.79
Precisión en el set de Test: 0.85
[[8513 1488]
 [   3   20]]
Distribución inicial de entrenamientoCounter({0: 23326, 1: 61})
Distribución finalde entrenamiento: Counter({0: 61, 1: 61})
Distribución inicial de test Counter({0: 10001, 1: 23})
Distribución final de test: Counter({0: 10001, 1: 23})
```

Figura 25. Distribución del dataset por subsampling

La tabla 12 muestra las métricas obtenidas luego del proceso de entrenamiento con reducción de dimensionalidad a 50 características y haciendo muestreo con subsampling; en este caso se observa que todos los indicadores disminuyen considerablemente con respecto a los valores obtenidos en el entrenamiento sin muestreo cuyos resultados fueron presentados en la tabla 10.

Tabla 12. Métricas Naive-Bayes con muestreo subsampling

MÉTRICAS	
Accuracy	0.8513
Recall	0.8696
Specificity	0.8512

- **Muestreo con OverSampling:**
  Para este caso se hizo uso de la clase *RandomOverSampler* de la librería *Imblearn.over_sampler*, en la que se replican aleatoriamente las observaciones pertenecientes a la clase minoritaria; nuevamente con el fin de evaluar los resultados en el caso extremo, las observaciones de la clase minoritaria fueron replicadas hasta obtener la misma cantidad de ocurrencias que la clase mayoritaria. Los resultados obtenidos con esta forma de muestreo, presentados tanto en la figura 26 como en la tabla 13, presentan una mejora en comparación con los obtenidos usando la técnica de muestreo por subsampling pero siguen siendo ligeramente inferiores al contrastarlos con los resultados obtenidos al procesar el dataset sin muestreo.

```
In [1]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/
ModeloNaiveBayesOverSampling.py', wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Precisión en el set de Entrenamiento: 0.92
Precisión en el set de Test: 0.95
[[9470  531]
 [   3   20]]
Distribución inicial de entrenamientoCounter({0: 23326, 1: 61})
Distribución finalde entrenamiento: Counter({0: 23326, 1: 23326})
Distribución inicial de test Counter({0: 10001, 1: 23})
Distribución final de test: Counter({0: 10001, 1: 23})
```

Figura 26. Distribución del dataset por oversampling

Tabla 13. Métricas Naive-Bayes con muestreo oversampling

MÉTRICAS	
Accuracy	0.9467
Recall	0.8696
Specificity	0.9469

- **Muestreo combinado con Oversampling a la clase minoritaria y Subsampling a la clase mayoritaria**:
  Finalmente, se ejecuta un último entrenamiento en el que se combinaron las dos técnicas de muestreo usadas para manejar la característica del desbalanceo entre clases. La configuración del muestreo se realizó en dos momentos: en el primero de ellos se generó un dataset de entrenamiento donde la relación entre la clase mayoritaria y la clase minoritaria fuera de 2:1; en el segundo momento, la configuración del muestreo generó una relación de 3:1 entre la clase mayoritaria y la

clase minoritaria respectivamente, como se puede observar en las figuras 26 y 27, esta proporción se pensó bajo la tesis que la clase negativa (clase mayoritaria) en condiciones reales tiene una mayor frecuencia que la clase positiva (clase minoritaria), característica que se cumple en el corpus de datos seleccionado.

```
In [11]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/
ModeloNaiveBayesSamplingSimultaneo.py', wdir='C:/Users/usuario/Documents/MAESTRIA/
APLICACION')
Precisión en el set de Entrenamiento: 0.92
Precisión en el set de Test: 0.97
[[9656  345]
 [   3   20]]
Distribución inicial de entrenamientoCounter({0: 23326, 1: 61})
Distribución finalde entrenamiento: Counter({0: 12500, 1: 6250})
Distribución inicial de test Counter({0: 10001, 1: 23})
Distribución final de test: Counter({0: 10001, 1: 23})
```

Figura 27. Distribución del dataset combinando oversamplig y subsampling 2:1

```
In [12]: runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/
ModeloNaiveBayesSamplingSimultaneo.py', wdir='C:/Users/usuario/Documents/MAESTRIA/
APLICACION')
Precisión en el set de Entrenamiento: 0.94
Precisión en el set de Test: 0.96
[[9648  353]
 [   3   20]]
Distribución inicial de entrenamientoCounter({0: 23326, 1: 61})
Distribución finalde entrenamiento: Counter({0: 15000, 1: 5000})
Distribución inicial de test Counter({0: 10001, 1: 23})
Distribución final de test: Counter({0: 10001, 1: 23})
```

Figura 28. Distribución del dataset combinando oversamplig y subsampling 3:1

Los datos, presentados en la tabla 14, comprueban que al realizar el muestreo combinado se alcanzan resultados similares a los obtenidos con el conjunto de entrenamiento original.

Tabla 14. Métricas Naive-Bayes combinando oversampling y subsampling 2:1 y 3:1

MÉTRICAS	2:1	3:1
Accuracy	0.9653	0.9645
Recall	0.8696	0.8696
Specificity	0.9655	0.9647

3.7 Estrategia usando Redes Neuronales Recurrentes LSTM

De acuerdo con lo formulado en el marco teórico, las Redes Neuronales Recurrentes (RNN), tienen como ventaja la posibilidad de creación de ciclos con lo que se puede atribuir a este tipo de redes la temporalidad necesaria para que la RNN tenga memoria; esta característica potente en dicho tipo de redes neuronales facilita el proceso de análisis de secuencias como lo puede ser el análisis de textos y, particularmente en el caso de este proyecto, fue aprovechada para el análisis de la secuencia de trigramas de token derivados de la representación del código fuente de los programas de computador.

Un tipo particular de las RNN son las redes Long-Short Term Memory (LSTM), en las que su memoria tiene la posibilidad de adaptarse para aprender sobre características relevantes que se hayan procesado anteriormente; por esta razón se decidió utilizar las redes LSTM para la implementación de la tercera estrategia, tomando como base que los elementos a procesar serán secuencias de trigramas cuyas características relevantes para la identificación de reuso de código podrían encontrarse en diferentes partes de la secuencia para cada caso.

La figura 29 muestra, de manera simplificada, la estrategia implementada desde la entrada del código hasta el entrenamiento de la red.

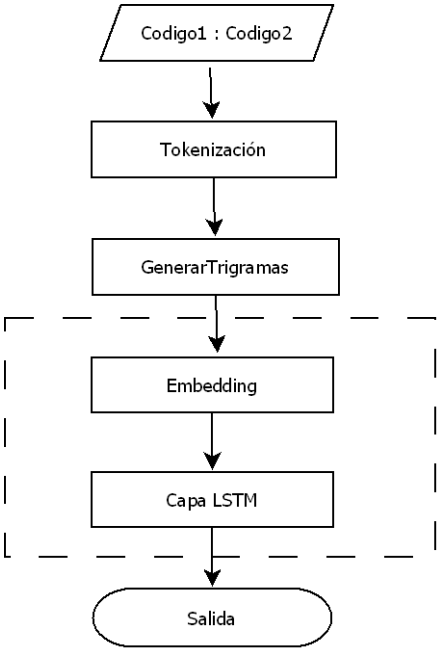


Figura 29. Modelo de la estrategia LSTM

Para la implementación de este modelo se hizo uso de la librería *Keras* del lenguaje Python, la cual provee las clases y funciones requeridas para desarrollar este tipo de red neuronal.

Como se puede apreciar en la figura 28, y al igual de lo implementado en los modelos anteriormente explicados, la entrada del código fuente de cada par se transforma en una representación de tokens, con los que luego se forman trigramas; estos trigramas de token forman una secuencia que alimenta la red para su entrenamiento.

Es claro que las secuencias de entrada son de longitud variable, sin embargo, el modelo implementado con la librería *Keras* requiere que las secuencias de entrada tengan la misma longitud, por lo cual las secuencias son fraccionadas en bloques de 500 trigramas, completando con la palabra '0' los bloques que no alcancen las 500 palabras; el modelo tiene la capacidad de identificar que esta palabra no representa información.

El modelo se configura y ajusta con la inclusión de una capa embedding con la que se usan vectores de incrustación de 32 dimensiones para representar cada palabra. La estimación de las unidades de memoria requeridas para la configuración de la capa LSTM inició con un valor experimental de 100 unidades, el cual se fue modificando para observar la relación entre precisión y costo computacional; el entrenamiento con las 100 unidades de memoria iniciales permitió tener una precisión cercana al 99.8%, pero el costo computacional evaluado en términos del tiempo de ejecución para el entrenamiento fue altamente costoso (alrededor de 18 horas) para un dataset de las dimensiones descritas en la sección 3.1. El proceso de ajuste permitió observar que en la medida en que se disminuía el número de unidades de memoria, la precisión oscilaba en valores entre el 99.6% y 99.8%, pero el tiempo que tomaba el entrenamiento disminuía.

Finalmente se logró alcanzar un alto nivel de precisión en el entrenamiento con 10 unidades de memoria en la capa LSTM a través de 3 iteraciones (epochs), dado que con un menor número de neuronas se empezaba a afectar la precisión. Por otra parte, en cuanto al control del sobreajuste (overfitting), se hizo uso del método Dropout, al cual se pasó como parámetro el valor 0.2 puesto que el número de neuronas utilizadas era bajo.

La tabla 15 resume las características de las capas LSTM y de salida:

Tabla 15. Configuración de la red LSTM

Capa Embedding	Vectores de incrustración de 32 dimensiones
Capa LSTM	Neuronas utilizadas: 10 Contenedor de la red LSTM: Modulo sequential.
Capa de salida	Dense de tamaño 2 Función de activación: Sigmoide

En la capa de salida, el "dense" de tamaño 2 se debe a que el problema que se abordó es un problema de clasificación que en conjunto con la función de activación sigmoide permitió configurar el modelo para el problema de clasificación binaria.

Finalmente, el entrenamiento se hizo con la función *RMSProp* (Método de gradiente descendente) para ajustar automáticamente la tasa de aprendizaje, usando la función predict para hacer las predicciones durante las etapas de entrenamiento y pruebas.

En las figuras 29 y 30 se puede apreciar el proceso de entrenamiento y los resultados obtenidos al final del mismo:

```
runfile('C:/Users/usuario/Documents/MAESTRIA/APLICACION/pruebaLSTM2.py',
wdir='C:/Users/usuario/Documents/MAESTRIA/APLICACION')
Reloaded modules: tmp9s4on5l5
Epoch 1/2
WARNING:tensorflow:Model was constructed with shape (None, 500) for input
Tensor("embedding_1_input:0", shape=(None, 500), dtype=float32), but it was called on an input with
incompatible shape (None, 4579).
WARNING:tensorflow:Model was constructed with shape (None, 500) for input
Tensor("embedding_1_input:0", shape=(None, 500), dtype=float32), but it was called on an input with
incompatible shape (None, 4579).
418/418 [==============================] - ETA: 0s - loss: 0.2944 - accuracy: 0.9909
WARNING:tensorflow:Model was constructed with shape (None, 500) for input
Tensor("embedding_1_input:0", shape=(None, 500), dtype=float32), but it was called on an input with
incompatible shape (None, 4579).
418/418 [==============================] - 2783s 7s/step - loss: 0.2944 - precision: 0.9909 -
val_loss: 0.1235 - val_precision: 0.9975
Epoch 2/2
418/418 [==============================] - 1318s 3s/step - loss: 0.1123 - precision 0.9975 -
val_loss: 0.0608 - val_precision: 0.9975
Model: "sequential_1"
```

Figura 30. Proceso de entrenamiento de la red LSTM

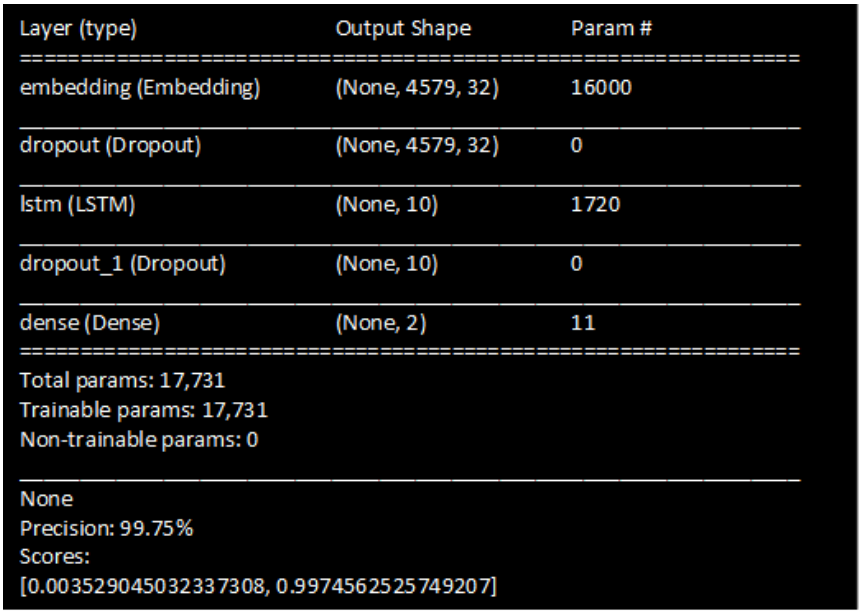


Figura 31. Resultados de entrenamiento de la red LSTM

Según se puede apreciar en la figura 30, la precisión obtenida con el modelo configurado de acuerdo con los datos mostrados en la tabla 15, es del 99.75%, lo que representa una mejora frente a los resultados obtenidos en los dos modelos implementados cuyos resultados fueron presentados anteriormente.

4. ANÁLISIS DE RESULTADOS

En este capítulo se presentan y comentan los resultados obtenidos durante el proceso de entrenamiento de los diferentes modelos implementados; es de interés particular resaltar que para la implementación y ejecución de los modelos se usó el lenguaje de programación Python en su versión 3.8.6; así como las librerías Keras, versión 2.2.4; Scikit-learn 0.23.2, Sklearn, versión 0.19.1, con los que fue posible hacer uso de los algoritmos de Machine Learning en la ejecución del proyecto y descritos en el capítulo 3.

4.1 Dataset para los experimentos

El dataset usado tanto para el entrenamiento como para las pruebas, se conformó a partir del repositorio SOCO (descrito en la sección 3.2) formando parejas de programas en las que había diferentes niveles de reuso de código. En la tabla 16 se presenta un resumen de las características del dataset conformado.

Tabla 16. Estadísticas del dataset

CARACTERÍSTICA	VALOR
Programas en el corpus	259
Parejas conformadas en el dataset	33411
Parejas etiquetadas con reuso de código	84
Número máximo de líneas de código	1664
Número mínimo de líneas de código	14
Número promedio de líneas de código	755

Para la conformación de los conjuntos de entrenamiento y test en cada uno de los modelos implementados se procedió a hacer una partición aleatoria del dataset en dos subconjuntos con el 70% y el 30% respectivamente. Sin embargo, en la sección 3.6 se explicó que dado el desbalance en la proporción de casos negativos (sin reuso de código) frente a los casos positivos (con reuso de código), se incluyeron experimentos donde se llevaron a cabo procesos tanto de oversampling como de subsampling con el fin de determinar si dicho desbalance podría conducir a un posible overfitting (sobreajuste); los resultados de este proceso se presentaron en la mencionada sección y se comentan adelante en este mismo capítulo.

4.2 Discusión de resultados individuales

Como se ha comentado en el capítulo correspondiente a la Ingeniería del Proyecto, la solución del problema planteado para el presente proyecto se abordó desde tres enfoques diferentes con el fin de hacer un análisis de los resultados obtenidos en cada uno de estos enfoques y, de esta manera, llegar a conclusiones que permitan guiar el proceso de analizar por pares el código fuente de programas y, de esta manera, determinar el posible reuso de código entre ellos.

4.2.1 Resultados de la clasificación basada en la similitud coseno

En el primer enfoque, basado en el concepto de la recuperación de información y la bolsa de palabras (Bag of Words), se transformaron los archivos fuente a través de un proceso de tokenización, formando secuencias de token que posteriormente fueron considerados para formar trigramas; de esta manera, se forma la bolsa de palabras (bolsa de trigramas) con los diferentes trigramas generados con todos los archivos del dataset y usando la similitud coseno para hacer la clasificación. Los resultados obtenidos, luego del proceso de ejecución de los 5 experimentos explicados en la sección 3.5 muestra que en todos los experimentos se obtienen los mejores resultados con un valor de similitud coseno de 0.75, este valor se obtiene calculando el coseno entre los vectores que representan cada documento; dichos resultados se presentan en la tabla 17, en la que es posible observar que con el uso de la medida de similitud coseno en 0.75 se obtiene un desempeño medido con F1-Score entre el 63% y el 71%; a pesar del desbalance en el dataset, se obtuvo un buen aprendizaje en la clase mayoritaria el cual se refleja en los resultados de Recall, sin embargo los bajos resultados obtenidos en la precisión dan cuenta de un bajo rendimiento en el aprendizaje de la clase minoritaria; este bajo rendimiento se refleja en la medida F1-Score.

Tabla 17. Resultados de los cinco experimentos en el modelo 1

EXPERIMENTO	F1-SCORE	RECALL	PRECISION	ACCURACCY
P1	0.70	0.99	0.54	0.97
P2	0.71	0.98	0.55	0.98
P3	0.71	0.99	0.55	0.98
P4	0.66	0.98	0.50	0.97
P5	0.63	0.99	0.46	0.97

De acuerdo con lo anterior, se puede deducir que el desbalance en el dataset, de alrededor de 99% frente al 1% entre las clases, es un factor determinante en este modelo para no tener el mismo desempeño en el aprendizaje en ambas clases.

4.2.2 Resultados de la clasificación basada en Naive-Bayes

En este segundo enfoque, en el que se usa el modelo Naive-Bayes para la implementación del clasificador, se tomó la decisión de reutilizar la estrategia de transformación del código fuente a través del proceso de tokenización y construcción de secuencias de 3-gramas utilizado en el método de la similitud coseno, esto con el fin de aprovechar la política para el manejo de los diferentes niveles de modificación de código comentados en la sección 3.1; adicionalmente, esto permitió hacer la comparación en los resultados entre los distintos métodos de clasificación partiendo de una base común: 3-gramas de token.

De este modo, la entrada al modelo es un conjunto de vectores en un espacio de 4570 dimensiones, que corresponden al valor tf-idf de cada 3-grama en la secuencia de trigramas que representan cada código fuente. Esta alta dimensionalidad, debida a la gran variedad de trigramas generados en todo el dataset, causó un alto número de valores cero en los vectores pues muchos trigramas no aparecían en ciertos programas, de manera que estos valores afectaban los resultados del entrenamiento pues hacían muy similares los vectores entre sí; por esta razón los experimentos iniciales se ejecutaron con una reducción de la dimensionalidad a 1000, 100 y 50 características (dimensiones) principales usando como criterio para el valor de 1000 en el experimento inicial, la estimación de la cantidad de trigramas que en promedio se generaban de los programas del corpus de datos, en los siguientes experimentos el valor se redujo, de manera experimental, observando la variación en los resultados lo que permitió identificar el número apropiado de características que debían ser considerados para el entrenamiento del modelo. En la tabla 18 se presentan las métricas obtenidas para cada uno de estos experimentos.

Tabla 18. Resultados de los experimentos con el Modelo Naive-Bayes con muestreo simple aleatorio

NÚMERO DE CARACTERÍSTICAS	F1-SCORE	RECALL	PRECISION	ACCURACCY
1000	0.13	0.59	0.07	0.98
100	0.10	0.88	0.05	0.96
50	0.11	0.88	0.06	0.96

De los resultados presentados en la tabla 18 se observa que el experimento en el que se aplica una reducción de dimensionalidad a 50 características tiene una mejor relación entre las diferentes métricas usadas con respecto a los otros dos experimentos, siendo interesante

resaltar que a pesar de obtener un valor *accuraccy* (exactitud) cuya diferencia varía entre 0.0 y 0.02, los experimentos con 100 y 50 características principales mejoran el valor *recall* (exhaustividad) correspondiente a la capacidad del modelo para identificar los casos de reuso de código; entre estos dos últimos experimentos se opta por el modelo que reduce la dimensionalidad a 50 características principales debido a que muestra un leve mejor comportamiento en la precisión y en la medida F1-Score, además de tener la mejor relación en la tasa de verdaderos positivos y la tasa de falsos positivos encontrados y que fue presentada en la curva ROC de los respectivos experimentos (figura 22 - sección 3.6.3).

Una vez dirimida la incertidumbre relacionada con la alta dimensionalidad de la representación vectorial de los códigos fuente que conforman el dataset, se ejecutan experimentos que permitieran analizar los efectos del desbalance en relación con la proporción de las clases positivas y negativas; como se ha descrito anteriormente, estos experimentos incluyeron pruebas en las que se realizó muestreo con oversampling y subsamplig, por separado, y pruebas adicionales en las que se combinó la estrategia de aplicar oversampling sobre la clase minoritaria y subsampling sobre la clase mayoritaria con proporciones de 2:1 y 3:1 de la clase negativa (mayoritaria) sobre la clase positiva (minoritaria); en todos los casos la representación vectorial usó las 50 características principales como se explicó en la sección 3.6. Las métricas observadas como resultado de estos experimentos se presentan en la tabla 19.

Tabla 19. Resultados de los experimentos con Oversampling y Subsampling

MUESTREO	F1-SCORE	RECALL	PRECISION	ACCURACCY
Oversampling	0.90	0.87	0.94	0.91
Subsampling	0.78	0.87	0.71	0.76
Combinación Subsampling-Oversampling 2:1	0.10	0.87	0.05	0.97
Combinación Subsampling-Oversampling 3:1	0.10	0.87	0.05	0.96

Como se puede observar, al aplicar la estrategia de muestreo por subsamplig desmejoran considerablemente los indicadores obtenidos con respecto a los experimentos con muestreo simple aleatorio; por otro lado, los experimentos aplicando muestreo por oversamplig conservan valores cercanos a los obtenidos en el muestreo simple aleatorio para las métricas accuraccy y recall, mientras que los valores asociados con precisión y f1-score mejoran significativamente al tener una mayor proporción de observaciones en la clase positiva. Sin embargo, la precisión disminuye considerablemente al combinar ambas técnicas de muestreo

y, por consiguiente, se disminuye la medida F1-Score, siendo similares a las medidas presentadas en la tabla 18 y que corresponden al entrenamiento sin muestreo.

Este último resultado permite inferir que el desbalance en el dataset afecta la exactitud del modelo, es decir su capacidad de detectar los casos positivos es aceptable, sin embargo, la proporción de las predicciones correctas por lo que clasifica incorrectamente un alto número de casos negativos. De acuerdo con los resultados obtenidos en los experimentos con subsamplig y oversamplig por separado, la calidad del modelo puede mejorar en la medida en que se pueda aumentar los casos reales de reuso de código en el dataset.

4.2.3 Resultados de la clasificación usando redes LSTM

El uso de las redes neuronales recurrentes LSTM, para el tercer enfoque de solución, implicó realizar experimentos con diferentes valores de unidades de memoria en la capa LSTM con el fin de observar el comportamiento del mismo en función de los resultados obtenidos y el tiempo de ejecución requerido para el entrenamiento; en la tabla 20 se presentan los datos resultantes en cada experimento:

Tabla 20. Resultados de los experimentos LSTM con diferentes unidades de memoria

UNIDADES DE MEMORIA	F1-SCORE	RECALL	PRECISION	ACCURACCY
100 neuronas	0.981	0.985	0.977	0.9976
50 neuronas	0.980	0.985	0.975	0.9973
10 neuronas	0.980	0.985	0.975	0.997
5 neuronas	0.905	0.872	0.940	0.966

Como se puede observar, la eficiencia en los modelos que se configuraron con 10 neuronas en adelante tienen una eficiencia similiar y suficientemente satisfactoria, sin embargo, el costo computacional en función del tiempo requerido para el entrenamiento varió desde un par de horas hasta 15 horas. Por otra parte, al disminuir el número de unidades de memoria por debajo de 10, se mantiene un buen comportamiento en la precisión, pero disminuye el valor recall, lo que repercute en una ligera disminución en la medida F1-Score indicando que el rendimiento general del modelo desmejora en la medida en que las unidades de memoria se reducen por debajo de 10.

4.3 Comparación de los resultados entre modelos.

En la figura 32 se puede contrastar la eficiencia de los diferentes modelos implementados, pudiendo apreciar con claridad que los resultados obtenidos con el modelo Naive Bayes con oversampling y los obtenidos con las redes LSTM son muy cercanos, sin embargo, se tiene una mayor confiabilidad en el segundo modelo puesto que en su entrenamiento no se aplicó algún tipo de muestreo para manejar el desbalance de las clases, en cuyo caso el método de Naive Bayes desmejora ostensiblemente los valores de precisión y F1-Score.

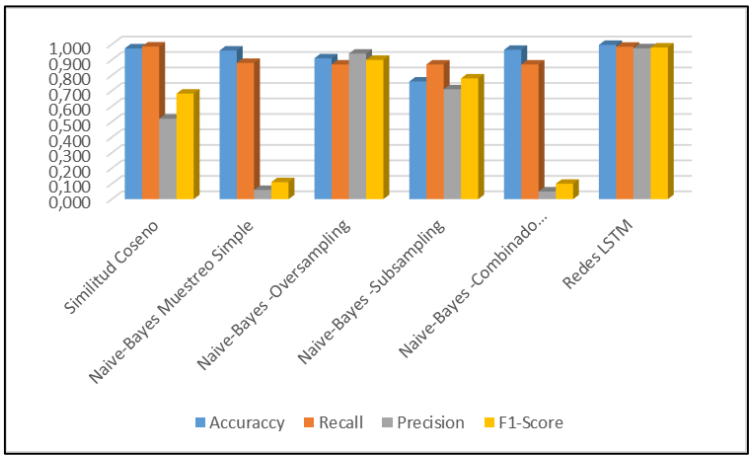


Figura 32. Comparación de métricas en los modelos implementados.

Con estos resultados se puede resaltar la capacidad de las redes neuronales recurrentes LSTM para analizar secuencias e identificar relaciones en ellas con lo que es posible clasificar correctamente tanto instancias positivas como negativas independientemente del desbalance en las clases.

5. CONCLUSIONES

El desarrollo de este trabajo de grado, de carácter investigativo-experimental, permitió abordar cada uno de los objetivos propuestos y de cuya ejecución se obtienen las conclusiones que se presentan en este capítulo, lo que se constituye en una herramienta que aporta al estado del arte referente al problema de la identificación de similitud entre pares de código fuente mediante la aplicación de técnicas de inteligencia artificial.

La primera conclusión a la que se llegó se relaciona con la importancia de la estrategia de representación del código fuente para el análisis de similitud, en función de los diferentes niveles de modificación del mismo (que van desde el simple cambio en los nombres de las variables, hasta el cambio de sentencias por instrucciones equivalentes). En este sentido se identifica que aquellas propuestas basadas en una representación del código ligadas fielmente al árbol de sintaxis abstracta pueden funcionar muy bien en la detección de reuso de código en los niveles más altos de modificación, pero llegan a ser inmunes a los niveles básicos de modificación, que son más comunes en entornos académicos.

Se concluye también que la medida de similitud coseno que debe ser usada como referencia para determinar si dos programas se consideran un caso de código compartido (o reusado) puede variar en relación con el contexto en que dichos programas son evaluados, con esto se quiere señalar que el análisis de similitud de código depende de si dichos programas se presentan en un contexto abierto o, por el contrario, son presentados en un contexto controlado donde cierto nivel de similitud estará presente sin que se trate de un caso de reuso. La expresión reuso de código, o código compartido, es utilizada en el desarrollo del proyecto puesto que no es competencia del mismo identificar plagio dado que esto implicaría detectar el código original, lo cual puede ser una motivación para un trabajo futuro.

En tercer lugar, se puede concluir que el hecho de que el dataset presente un alto nivel de desbalance, al rededor del 99% de las instancias corresponden a la clase negativa y solamente el 1% a la clase positiva, afecta la precisión en algunos modelos en los que la frecuencia de cada clase es determinante para la obtención de resultados; manejar este desbalance con estrategias de muestreo permitió mejorar ligeramente la medición de la etapa de pruebas.

De las conclusiones anteriores se deriva como cuarta conclusión que si bien el uso de modelos con diferentes enfoques para la detección de reuso de código, como los modelos basados en la similitud coseno, el modelo Naive Bayes y las redes LSTM, tuvieron resultados altos en cuanto a la capacidad de predecir correctamente los casos positivos, se encuentra que el modelo Naive Bayes clasifica incorrectamente como positivos una alta proporción de instancias frente a las que efectivamente son casos de reuso, indicando una alta afectación debida al desbalance del dataset.

Finalmente, se pudo concluir que el uso de diferentes métricas para evaluar en conjunto el desempeño de los clasificadores, como recall, accuraccy, y precisión, es necesaria para tener un análisis más preciso de la problemática del reuso de código en sus diferentes niveles y la capacidad de detectarlo teniendo en cuenta las características propias del contexto en el que los programas son desarrollados.

Como trabajos futuros se recomienda que el proyecto sea escalado a la identificación del reuso de código translingüe que, si bien ya se encuentra en la literatura algunos adelantos en este sentido, se propone tomar los elementos de representación planteados. Así como el uso de redes neuronales recurrentes LSTM descritos en el capítulo de Ingeniería del Proyecto; esto debe suponer el diseño de un mecanismo de tokenización genérico independiente del lenguaje de programación en el que esté escrito el código.

7. REFERENCIAS BIBLIOGRÁFICAS

[1] J. Coque, M.A. García., P.L. González-Torre, "El plagio entre el alumnado universitario: un caso exploratorio". 26 Congreso Universitario de Innovación Educativa en las Enseñanzas Técnicas 2018 - CUIEET, 2018.

[2] S. Jaramillo Valbuena, N.F. Rincón Belalcázar, "Los estudiantes universitarios y la sociedad de la información: una combinación que ha facilitado el plagio académico en las aulas Colombianas". *Información, Cultura y Sociedad*, no. 30, 2014.

[3] B. Hill Mayoral, "La corrupción, las empresas y la responsabilidad social corporativa", 2000. [Online]. Disponible:

http://www.coparmex.org.mx/contenidos/publicaciones/Entorno/2001/oct01/corrupci%F3n.htm

[4] Ganguly, D., Jones, G.J.F., Ramírez-de-la-Cruz, A. et al., "Retrieving and classifying instances of source code plagiarism". Springer Netherlands, 2018

[5] K. D. Cooper, L. Torczon, "Engineering a Compiler.", El Sevier, 2008

[6] J. Ruiz Catalán, "Compiladores: teoría e Implementación". Alfaomega, 2010

[7] J. D. Kelleher, B. Mac Namee, A. D'Arcy, "Fundamentals of machine learning for predictive data analytics : algorithms, worked examples, and case studies". Massachusetts: MIT Press. 2015

[8] E. Alpaydin, "Introduction to machine learning". Third edn. Cambridge, Massachusetts: MIT Press (Adaptive computation and machine learning), 2014

[9] R. S. King, "Cluster analysis and data mining : an introduction". Dulles, Virginia: Mercury Learning and Information. 2015

[10] Math Works, "Máquina de vectores de soporte (SVM)", 2019. [Online]. Disponible: https://la.mathworks.com/discovery/support-vector-machine.html%5d..html

[11] S. Russell, P. Norvig, "Inteligencia Artificial, Un Enfoque Moderno", Prentice Hall, 2007.

[12] G.A. Toro-Bayona, I.A. Lizarazo-Salcedo, "Evaluación de las Redes Neuronales Artificiales Perceptron Multicapa y Fuzzy-Artmap en la Clasificación de Imágenes Satelitales", Ingeniería, vol. 17, núm. 1, 2012

[13] S. Tahsildar, "What are deep neural networks?", BE Engineering & Computer Science, Vidyalankar Institute of Technology, 2017. [Online]. Disponible: https://www.quora.com/What-are-deep-neural-networks

[14] J. I. Bagnato. "Aprende Machine Learning". Udemy, 2019 https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-lstms-807fa6ca7f

[15] P. Neelabh. "A Guide For Time Series Prediction Using Recurrent Neural Networks (LSTMs)". https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-lstms-807fa6ca7f, 2017.

[16] J.A. Lossio-Ventura, H. Alatrista-Salas, " TUTORIAL: Minería de datos textuales utilizando Treetagger y Weka". 2019. [Online]. Disponible: https://simbig.org/SIMBig2014/TutorialSIMBig.pdf

[17] M. Nicolás-Franco, J.M. Vivo-Molina, "Análisis de Curvas Roc: principios básicos y aplicaciones", La Muralla, 2007

[18] D. Weber-Wulff, “Plagiarism Detection Software: Promises, Pitfalls, and Practices”. Handbook of Academic Integrity. Springer, Singapore, 2016

[19] D. Weber-Wulff, “Plagiarism Detection Software: Promises, Pitfalls, and Practices”, Handbook of Academic Integrity. Springer, Singapore, 2015

[20] S. Schleimer, Saul. D.Wilkerson, A. Aiken, “Winnowing: Local Algorithms for Document Fingerprinting”. Proceedings of the ACM SIGMOD International Conference on Management of Data. S, 2003

[21] S. Nazir, S. Shahzad, N. Mukhtar, “Software Birthmark Design and Estimation: A Systematic Literature Review”, Springer Berlin Heidelberg, 2019

[22] G. Debasis, J. F. Jones Gareth, A. Ramirez-de-la-Cruz, G. Ramirez-de-la-Rosa, E.Villatoro-Tello, “Retrieving and classifying instances of source code plagiarism”, Information Retrieval Journal, 2017

[23] T. Ohmann, I. Rahal, “Efficient clustering-based source code plagiarism detection using PIY”, Springer-Verlag London, 2014

[24] M. Ďuračík, E. Kršák and P. Hrkút, “Scalable Source Code Plagiarism Detection Using Source Code Vectors Clustering”, *2018 IEEE 9th* International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2018, pp. 499-502, 2018

[25] A. Ahadi, L. Mathieson, “A Comparison of Three Popular Source code Similarity Tools for Detecting Student Plagiarism”. Proceedings of the Twenty-First Australasian Computing Education Conference on - ACE ’19, 2019

[26] E. Florez, P. Rosso, L. Moreno, “On the Detection of SOurce COde Re-use. In FIRE'14 Proceedings of the Forum for Information Retrieval Evaluation”, pp 21-30, ACM Proceedings. 2014

[27] J. Faidhi, S. Robinson. “An empirical approach for detecting program similarity and plagiarism within a university programming environment”. Computers and Education, 1987.

[28] E. Flores Saez. “Reutilización de código fuente entre lenguajes de programación”, Trabajo final de máster, Universidad Politécnica de Valencia, 2012.

[29] G. Cosma, M. Joy. “An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis”. IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, NO. 3, 2012.

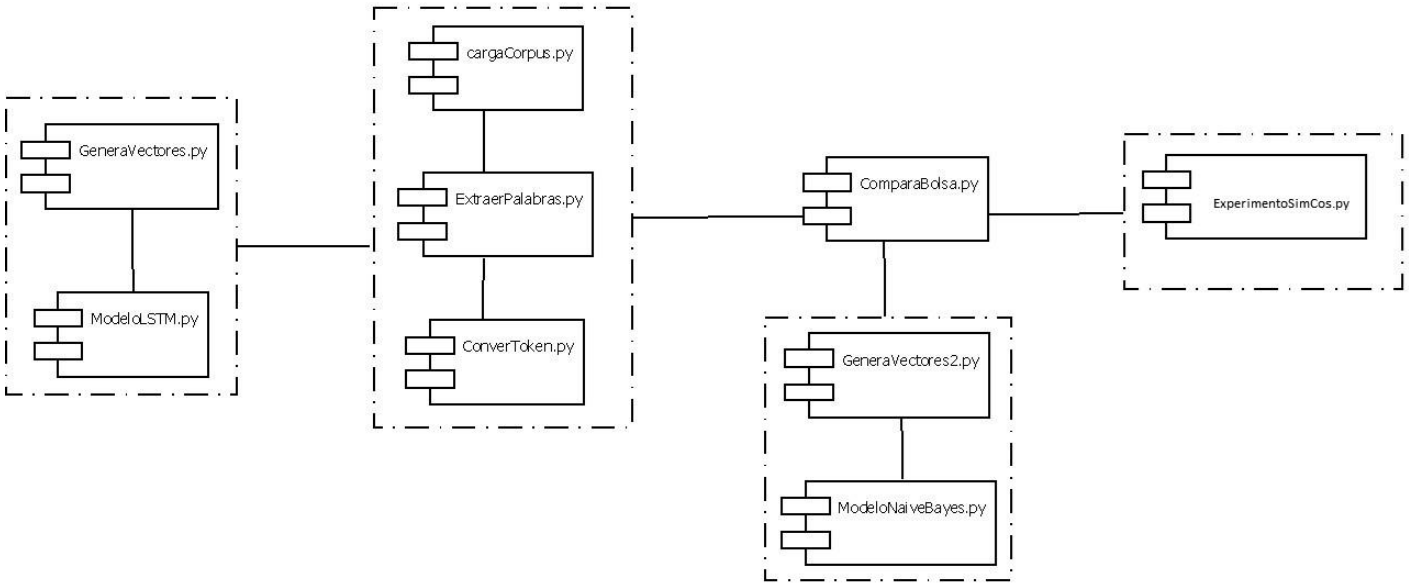
[30] F. Bravo-Marquez, G. L'Huillier, S. Rios, J.D. Velasquez, and L. Guerrero. “DOCODE-lite: A Meta-Search Engine for Document Similarity Retrieval”, In KES '10: 14th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems. Cardiff, Wales, 2010.

[31] G. Pajares Martinsanz, J.M. De la Cruz García. “Aprendizaje Automático”. Ediciones de la U, 2011.

[32] J. Amat Rodrigo. “Principal Component Analysis, PCA”. Ciencia de Datos . Net (https://www.cienciadedatos.net/documentos/35_principal_component_analysis), 2017.

[33] K. Cao, C Wei, A. Gaidon. “Learning Imbalanced Datasets with Label-Distribution-Aware-Margin Loss”. 33rd Conference on Neural Information Processing System, 2019.

ANEXO 1. DIAGRAMA DE COMPONENTES



ANEXO 2. CÓDIGO FUENTE DE LA SOLUCIÓN

Los archivos que contienen el código fuente de la solución implementada pueden ser consultados en línea en el repositorio https://github.com/mauriciolopezb/aplicacion.

La secuencia de ejecución de los archivos se describe a continuación:

- MODELO 1: ExperimentoSimCos.py --> ComparaBolsa.py --> ConverToken.py --> ExtraerPalabras.py --> cargaCorpus.py

- MODELO 2: ModeloNaiveBayes.py --> GeneraVectores2.py --> ComparaBolsa.py --> ConverToken.py --> ExtraerPalabras.py --> cargaCorpus.py

- MODELOS 3, 4, 5 : Similar al anterior, cambiando el modelo inicial por ModeloNaiveBayesOverSampling.py, ModeloNaiveBayesSubSampling.py y ModeloNaiveBayesSamplingSimultaneo.py

- MODELO 6: ModeloLSTM.py --> GeneraVectores.py --> ConverToken.py --> ExtraerPalabras.py --> cargaCorpus.py