Revista Peruana de Computación y Sistemas 2023 5(1):25-32
https://doi.org/10.15381/rpcs.v5i1.25802

ISSN Online: 2617-2003
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos

Artículo de Contribución

# Sistema de Recomendación basado en Contenido para Jueces de Programación utilizando Procesamiento de Lenguaje Natural y Aprendizaje Profundo

## Content-Based Recommendation System for Programming Judges using Natural Language Processing and Deep Learning

**Wilson Julca-Mejia[1,a], Herminio Paucar-Curasma[1,2,b]**

[1] Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática. Lima, Perú

[2] Universidad de São Paulo, Instituto de Matemáticas y Ciencias de la Computación. São Paulo, Brasil

[a] Autor de correspondencia: wilson.julca@unmsm.edu.pe, ORCID: https://orcid.org/0009-0007-2998-3919

[b] E-mail: herminiopaucar@usp.br, ORCID: https://orcid.org/0000-0001-9565-3757

## Resumen

En el ámbito de la educación y las compañías tecnológicas, los jueces en línea juegan un papel importante en el desarrollo de habilidades de programación debido a que en estas plataformas los estudiantes deben resolver desafíos utilizando lenguajes de programación específicos. Sin embargo, la gran cantidad de desafíos de programación disponibles puede ser abrumadora para los estudiantes generando frustración y pérdida de interés. Para resolver esta situación, los sistemas de recomendación pueden ser una solución eficaz. Sin embargo, los jueces de programación no han profundizado lo suficiente en esta área. Por lo tanto, esta investigación se enfocó en evaluar seis técnicas de inteligencia artificial mediante una arquitectura basada en la nube para la predicción del nivel de dificultad a partir de los planteamientos de los problemas para ser acoplado a un sistema de recomendación. Para validar los experimentos, se utilizó un juez real de programación CodeChef y los experimentos se evaluaron a través de pruebas estadísticas. Los resultados indicaron que el modelo BERT es el mejor prediciendo el nivel de dificultad de los problemas lo que ayuda al sistema de recomendación a mejorar la experiencia de aprendizaje de los estudiantes en los jueces de programación en línea.

**Palabras clave:** Jueces de Programación en línea, Sistemas de Recomendación, Procesamiento de Lenguaje Natural, Aprendizaje Profundo.

## Abstract

In the field of education and technology companies, online judges play an important role in the development of programming skills because on these platforms students must solve challenges using specific programming languages. However, the sheer number of coding challenges available can be overwhelming for students, leading to frustration and loss of interest. To resolve this situation, recommender systems can be an effective solution. However, programming judges have not delved far enough into this area. Therefore, this research focused on evaluating six artificial intelligence techniques through a cloud-based architecture for the prediction of the level of difficulty from the statements of the problems to be coupled to a recommendation system. To validate the experiments, a real CodeChef programming judge was used and the experiments were evaluated through statistical tests. The results indicated that the BERT model is the best for predicting the level of the problems, which helps the recommendation system to improve the learning experience of the students in the online programming judges.

**Keywords:** Programming Online Judges, Recommender Systems, Natural Language Processing, Deep Learning.

Recibido: 30/04/2023 - Aceptado: 28/05/2023 - Publicado: 30/06/2023

**Citar como:**
Julca-Mejia, W. & Paucar-Curasma, H. (2023). Sistema de Recomendación basado en Contenido para Jueces de Programación utilizando Procesamiento de Lenguaje Natural y Aprendizaje Profundo. Revista Peruana de Computación y Sistemas, 5(1):25-32. https://doi.org/10.15381/rpcs.v5i1.25802

## 1. Introducción

Los jueces de programación en línea (POJ) tienen una gran importancia en la industria y la academia. En la industria, permite mejorar habilidades como el razonamiento lógico, el manejo de estructuras de datos y el uso de diferentes paradigmas de algoritmos para diferentes problemas de la vida real. Todos estos temas son una parte importante de las entrevistas en Big Tech como Amazon, Google, Meta y otros [1]. En el sector educativo se ha comprobado que el uso de POJ en cursos de Informática mejora considerablemente el rendimiento de los estudiantes [2]. Actualmente, existe un número considerable de POJ con una gran cantidad de desafíos en orden secuencial [3], pero tienen poca claridad sobre la dificultad y categoría de estos [4]. Algunos ejemplos de la gran cantidad se pueden ver en el POJ UVA que tiene más de 2 mil desafíos y en el juez SPOJ que tiene más de 6 mil problemas de programación [5]. Esta sobrecarga de información genera frustración y abandono en los estudiantes si no cuentan con alguien con experiencia que los guíe en los problemas a resolver [3, 5, 6, 7]. A pesar de este problema, los POJ rara vez tienen alguna forma de recomendar el próximo desafío a resolver [6]. El método clásico de recomendación es sugerir los retos más resueltos. Este enfoque carece de personalización porque sugerirá los mismos desafíos a todos los usuarios, independientemente de los problemas que hayan resuelto [7].

Una de las formas de recomendación más utilizadas por su eficacia y facilidad son las técnicas de filtrado colaborativo (CF), que utiliza la información interactiva del usuario con los ítems [8, 9]. A pesar de los beneficios de las técnicas CF, solo se limita a las interacciones usuario-elemento y no tiene en cuenta la información semántica de los elementos, para lo cual existen enfoques de filtrado basado en contenido (CBF). Las técnicas de CBF usan metadatos de los elementos (descripción, características, etiquetas) para encontrar ítems similares que le hayan gustado al usuario en el pasado [10], lo que permite proporcionar recomendaciones más precisas y personalizadas. Además, el CBF no está limitado por la dependencia de los datos históricos de interacciones entre usuarios e ítems, lo que lo hace especialmente útil en situaciones en las que se tienen pocos datos de interacción [10].

Algunos autores [11, 12, 13] han explorado la utilización de características específicas de los problemas de programación como entrada para la generación de recomendaciones. En este sentido, se han abordado los desafíos de programación categorizando las descripciones y/o el código fuente de los problemas como features para la generación de un sistema de recomendación basado en contenido. De esta forma, se pueden utilizar técnicas de minería de datos y aprendizaje automático para analizar y categorizar estos atributos (features) y, posteriormente, hacer recomendaciones personalizadas a los programadores en función de su historial de programación y preferencias.

El presente artículo tiene como objetivo plantear un sistema de recomendación con un modelo de clasificación de texto basado en la Representación de Codificador Bidireccional de Transformadores (BERT), el cual se complementa con procesos de normalización, eliminación de stopwords, tokenización y lematización. El modelo propuesto será comparado con técnicas clásicas de clasificación de texto, tales como SVC, KNN, RandomForest y DecisionTree, utilizando las técnicas de Bag of Words y TF-IDF. Para la implementación del sistema de recomendación se utilizará una arquitectura basada en la nube, haciendo uso de componentes de Amazon Web Services (AWS) para proporcionar una solución de clasificación de texto altamente escalable, precisa y fácilmente integrable en otras aplicaciones, utilizando tecnologías de vanguardia en el campo de la inteligencia artificial y la computación en la nube.

Se elige BERT debido a su superioridad en la clasificación de texto en comparación con enfoques tradicionales de procesamiento de lenguaje natural [14, 15]. BERT es conocido por su capacidad para capturar el contexto y las relaciones semánticas en el texto, lo que lo hace especialmente adecuado para aplicarlo al escenario de los desafíos de programación de jueces en línea.

En este trabajo, la sección 2 hablará sobre el marco teórico y trabajo relacionado, la sección 3 sobre la metodología, la sección 4 discutirá los experimentos, la 5 los resultados y la 6 sobre conclusiones y trabajo futuro.

## 2. Marco Teórico y antecedentes

### *2.1. Sistemas de Recomendación*

Los sistemas de recomendación (RS) proporcionan una colección de elementos recomendados para aliviar el proceso de búsqueda en un entorno sobrecargado, tratando de predecir los productos o servicios más adecuados según las preferencias del usuario [16, 17]. Estos se pueden clasificar en varias categorías: sistemas de recomendación de filtrado basado en contenido (CBF), sistemas de recomendación de filtrado colaborativo (CF) y sistemas de recomendación de filtrado híbrido (HF) [18] para los cuales se analizan los datos recopilados. y aprenden utilizando varios métodos y técnicas de aprendizaje automático, como el agrupamiento y la categorización [16].

### *2.2. Procesamiento de Lenguaje Natural con Aprendizaje Profundo*

La evolución del Procesamiento de Lenguaje Natural (NLP) con Aprendizaje Profundo ha transitado desde el uso de técnicas como la inscrustración de palabras hasta la adopción de modelos basados en transformadores. Esto último, permitió que BERT y GPT surgieran y se desarrollaran llevando el procesamiento de lenguaje natural a un nivel superior al capturar de manera efectiva el contexto y mejorar el rendimiento en diversas tareas. NLP con Aprendizaje Profundo es ampliamente utilizado y ha demostrado ser efectivo en clasificación de texto, extracción de información,

traducción automática, generación de texto, análisis de sentimientos, preguntas y respuestas, resumen automático de texto entre otras [19, 20].

### *2.3. Jueces de Programación*

Los jueces de programación en línea (POJ) son plataformas web que tienen una lista de problemas. Estos problemas son resueltos por los usuarios que envían sus soluciones (código fuente) desarrolladas en un lenguaje de programación soportado por la plataforma. El código fuente enviado se evalúa automáticamente y el veredicto se le da al usuario como podemos ver en la Tabla 1. Los POJ se inspiraron principalmente en el concurso mundial más prestigioso conocido como ACM-ICPC, donde diferentes equipos de todo el mundo compiten en un concurso presencial. Este concurso tiene una duración de 5 horas, en las que los participantes intentan resolver entre 10 a 13 problemas de programación y los ganadores serán los que resuelvan más problemas en el menor tiempo. Los POJ en estos días son una herramienta muy efectiva para aprender a programar. Muchos de ellos se han establecido con el propósito de aprender y competir en línea [4]. Por ejemplo, a nivel mundial, tenemos el juez en línea de la Universidad de Pekín, Universidad de Valladolid, Universidad Estatal de Saratov, SPOJ y Timus [3]. En Latinoamérica, algunos jueces son Caribbean Online Judge, Beecrowd y OmegaUp. En Perú, NinjaCoding y Huahcoding se utilizan en diferentes universidades y eventos [21].

**Tabla 1**

*Veredictos Princpales de los POJ*

| Veredicto | Abreviatura | Descripción |
|---|---|---|
| Accepted | AC | La solución es correcta |
| Wrong Answer | WA | La solución no es correcta |
| Time Limited Exceeded | TLE | La solución demora más de lo establecido |

### *2.4. Sistemas de Recomendación en POJ*

En el escenario de jueces de programación, los sistemas de recomendación que se han desarrollado se basan en contenido e interacciones usuario-elemento. Esta investigación se centra en los sistemas basados en contenido en la cual las investigaciones más relevantes encontradas en nuestra revisión de la literatura son:

En la investigación [6] usan el POJ llamado CodeBench y para medir el esfuerzo que tomó resolver un problema usan el número promedio de intentos, el número promedio de líneas de código, el número promedio de variables, la complejidad algorítmica, el número de intentos y otros indicadores para cada problema. Usando la similitud del coseno como la métrica de distancia, el análisis del vecino más cercano se usa para calcular el grado de similitud entre el problema recomendado y el problema activo.

En el estudio [13], proponen 2 métodos de recomendación 1) Método basado en el enunciado (MBE) que recomienda problemas con enunciados similares al problema resuelto donde utiliza técnicas de procesamiento de lenguaje natural para representar el texto de los enunciados en vectores quantitativos para el cálculo de la similaridad de coseno 2) Método basado en el comportamiento (MBC) que recomienda problemas similares basados en el nivel de esfuerzo requerido para resolver problemas que se obtienen del juez CodeBench. Como resutado el método MBE y MBC, obtuvieron 47, 8% y 63, 3% respectivamente de evaluacionens positivas por los alumnos. Además, en MBE y MBC el 60% de la recomendaciones fueron respondidas correctamente por los alumnos frente al 25% del método aleatorio.

En la pesquisa [12] con el objetivo de extraer el tema del planteamiento de un problema de un POJ, proponen una metodología de 5 fases las cuales son 1) aumento de datos, 2) preprocesamiento, 3) representación de texto, 4) clasificación y 5) validación en el que combina varias técnicas de NLP. En esta investigación se evidenció la importancia de parafrasear en la etapa de aumento de datos. Por ejemplo, sin parafrasear, el modelo GLOVE + SVM obtuvo 86% en la métrica F1. Por el contrario, el rendimiento del modelo después de parafrasear aumentó a 94%.

En la investigación [4] se centran en poder clasificar los problemas de programación según el tema al que pertenecen, con el fin de ayudar a los programadores novatos. En esta investigación, se utilizaron los desafíos del POJ AOJ donde se utilizaron técnicas de clasificación como Multilayer Perceptron (MLP), KNN, Naive Bayes para examinar las características extraídas por Latent Dirichlet Assignment (LDA) y Non-Negative Matrix Factorization (NMF). Concluyen que las etiquetas y las palabras clave junto con los clasificadores podrían ser una herramienta de recomendación para programadores novatos.

## 3. Metodología

### *3.1. Materiales y métodos*

El conjunto de datos utilizado en esta investigación se basa en el POJ de CodeChef. Este conjunto de datos contiene información sobre problemas de programación desde el año 2009 hasta el 2016, incluyendo un total de 1474 problemas. Cada problema en el conjunto de datos está identificado por un QCode único y se acompaña de su título, nivel de dificultad y descripción. Este conjunto de datos proporciona una base sólida para el análisis y la evaluación de sistemas de recomendación basados en contenido en el campo de la programación. En la Tabla 2 se muestran algunas estadísticas del dataset mencionado.

**Tabla 2**

*CodeChef Dataset*

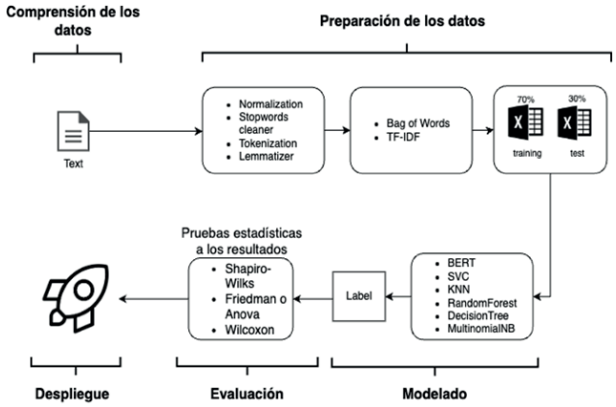
| | QCode | Title | Level | Statement |
|---|---|---|---|---|
| count | 1474 | 1474 | 1474 | 1474 |
| mean | 1474 | 1456 | 5 | 1430 |
| freq | 1 | 3 | 507 | 47 |

La metodología que se utiliza es CRISP-DM (Cross-Industry Standard Process for Data Mining), que proporciona fases y establece un conjunto de tareas y actividades para cada fase para proyectos de ciencia de datos [22].

En la Figura 1 describe cada fase llevada a cabo los cuales son:

**Fig 1**

*Metodología basado en CRISP-DM*




Fuente: Elaboración propia

*a. Comprensión de datos:* En esta fase se busca comprender los datos que se tienen disponibles, su calidad, su estructura, las relaciones entre las variables y los posibles problemas que puedan presentarse.

*b. Preparación de los datos:* Se lleva a cabo la limpieza, transformación y selección de los datos necesarios para el análisis posterior. En esta fase se aplica técnicas de Procesamiento de Lenguaje Natural como el Stopwords cleaner, que consiste en eliminar palabras que no aportan información relevante al análisis, como preposiciones o conjunciones. También se utiliza la tokenización, que implica la separación del texto en unidades significativas, como palabras o frases; Lematización, que consiste en reducir las palabras a su forma base, eliminando las terminaciones que indican género, tiempo o número. Además, utilizamos técnicas como el Bag of Words y el TF-IDF para convertir el texto en vectores numéricos.

*c. Modelado:* Se construye los modelos que permitirán hacer predicciones o identificar patrones en los datos. Los modelos seleccionados son los siguientes:

*c.1. 3.2. BERT (Bidirectional Encoder Representations from Transformers):* Es un modelo de lenguaje de última generación que utiliza redes neuronales para procesar texto. BERT utiliza el aprendizaje profundo y es capaz de procesar texto de manera bidireccional, lo que significa que tiene en cuenta el contexto de las palabras en ambas direcciones. Este modelo se ha utilizado con éxito en diversas tareas de procesamiento de lenguaje natural, incluyendo la clasificación de texto [15, 19, 23]. Para clasificar texto con BERT, se requiere el texto de entrada, etiquetas de clasificación, un modelo pre-entrenado que captura representaciones semánticas del lenguaje, y un proceso de finetuning para ajustar el modelo a la tarea de clasificación específica. Al combinar estos elementos brinda una representación profunda del texto y mejora la capacidad de clasificación [23].

*c.2. Random Forest:* Es un algoritmo de aprendizaje supervisado usado para la clasificación y regresión. Este modelo crea múltiples árboles de decisión y combina sus resultados para obtener una predicción final. RandomForest es conocido por su capacidad para manejar grandes conjuntos de datos y por su capacidad para manejar características no lineales [11].

*c.3. KNN (K-Nearest Neighbors):* Es un modelo de aprendizaje supervisado que clasifica una instancia en función de las clases de las instancias más cercanas a ella en un espacio de características. Este modelo es simple y fácil de implementar, pero puede ser computacionalmente costoso en grandes conjuntos de datos [24].

*c.4. Decision Tree (Árboles de Decisión):* Es un modelo de aprendizaje supervisado empleado para la clasificación y regresión. Este modelo crea un árbol de decisiones que se utiliza para tomar decisiones basadas en las características de una instancia. Los árboles de decisión pueden ser interpretados fácilmente por los humanos y son útiles para la clasificación de datos categóricos y numéricos [25].

*c.5.SVC (Support Vector Machine):* Es un modelo de aprendizaje supervisado aplicado para la clasificación y regresión. Este modelo busca el hiperplano que mejor separa las clases en un espacio de características. [24].

c.6. MultinomialNB (Naive Bayes): Es un modelo de aprendizaje supervisado que emplea el teorema de Bayes para predecir la clase de una instancia. MultinomialNB se utiliza comúnmente en la clasificación de texto y se basa en la suposición de que las características son independientes entre sí [24].

*d. Evaluación:* En esta fase se evalúa los modelos construidos para determinar su precisión y su capacidad para generalizar. Se utilizan técnicas estadísticas como la prueba Shapiro-Wilks para verificar la normalidad de los datos, la prueba Friedman para comparar los modelos y la prueba Wilcoxon para comparar pares de modelos.

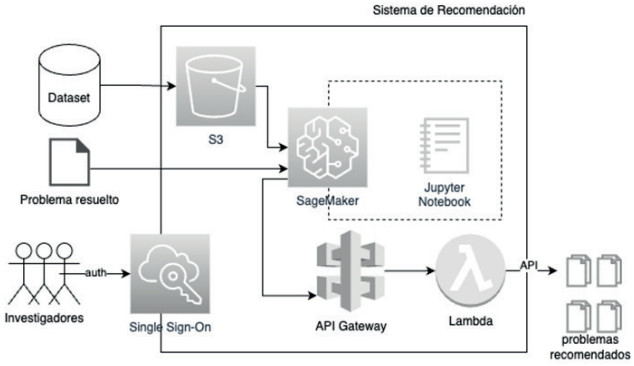
*e. Despliegue*: Se pone en marcha el modelo seleccionado y se integra en el sistema o proceso donde será utilizado. Se lleva a cabo un monitoreo continuo para asegurar que el modelo sigue siendo efectivo y se realiza el mantenimiento necesario.

### *3.2. Arquitectura*

Se propone una arquitectura basada en la nube que se puede observar en la Figura 2, haciendo uso de diversos componentes de Amazon Web Services (AWS). En particular, utilizamos servicios como Amazon SageMaker para el entrenamiento y evaluación del modelo, y se utilizarán notebooks de SageMaker para el desarrollo y pruebas del modelo. Los datos de entrenamiento y evaluación se almacenan en Amazon S3 para garantizar la seguridad y disponibilidad de los mismos. Para la implementación del modelo se utilizan servicios como AWS Lambda y Amazon API Gateway para la creación de una API RESTful que permita la integración del modelo con otras aplicaciones. Además, se utiliza Single Sign-On (SSO) para garantizar la seguridad y autenticación de los usuarios. El objetivo final es proporcionar una solución de sistema de recomendación mediante clasificación de texto altamente escalable, precisa y fácilmente integrable en otras aplicaciones, utilizando tecnologías de vanguardia en el campo de la inteligencia artificial y la computación en la nube.

**Fig 2**

*Arquitectura para Sistema de Recomendación*




Fuente: Elaboración propia

Una vez entrenado el modelo, nuestro sistema de recomendación recibirá un problema resuelto el cual llamaremos el problema objetivo para sugerir una lista de problemas similares a este. Con la ayuda de BERT, previamente entrenado, obtendremos el vector de características con el cual calcularemos la distancia de similaridad entre el vector de características del problema objetivo y los vectores de características de todos los problemas del conjunto de entrenamiento. Finalmente, ordenaremos los problemas y retornaremos una lista de problemas recomendados en orden descendente de similaridad como se observa en el algoritmo de la Figura 3.

**Fig 3**

*Algoritmo de sistema de recomendación propuesto*

```
def getListProblems(problem, allProblems, k):
    vector_features = BERT(problem)
    similarities = similarity(vector_features, allProblems)
    sorted_indices = sort(similarities)
    k_most_similar = sorted_indices[1:k+1]
    return k_most_similar
```

Fuente: Elaboración propia

## 4. Experimentación y Resultados

### *4.1. Configuración del Experimento*

Para llevar a cabo los experimentos de esta investigación, se utilizó inicialmente la plataforma Google Colaboratory con Python 3 y notebooks Jupyter, lo que permitió un proceso ágil y flexible de programación y análisis de datos. Sin embargo, para hacer que la arquitectura fuera más robusta y adecuada para un ambiente productivo, se transfirió el cuaderno de Colab a un cuaderno de SageMaker, la plataforma de aprendizaje automático de Amazon AWS. Esto permitió que el experimento se ejecutara en un ambiente en la nube con mayor capacidad de procesamiento y almacenamiento, y se pudiera hacer disponible a través de los componentes de AWS como S3, Lambda, y API Gateway. De esta forma, se pudo realizar un análisis más exhaustivo y preciso del modelo propuesto.

### *4.2. Resultados y Discusión*

En el estudio realizado se evaluó la precisión de clasificación de diferentes modelos de aprendizaje automático, tales como BERT, RandomForest, KNN, DecisionTree, SVC y MultinomialNB. Para obtener una evaluación precisa, se realizaron varias iteraciones con diferentes semillas para la generación de datos con 30% de prueba y 70% de entrenamiento. Los resultados se presentan en la Tabla 3, donde se observa que los modelos BERT y Random Forest obtuvieron una mayor precisión en comparación con los demás modelos evaluados. No obstante, es importante validar si estas diferencias son estadísticamente significativas mediante pruebas adecuadas.

Para ello, se utilizará la prueba de Shapiro-Wilks para conocer si los resultados están distribuidos de forma normal, lo que permitirá saber si es necesario utilizar una prueba paramétrica o no.

**Tabla 3**

*Resultados de los modelos en base a la métrica de Precisión.*

| S | BERT | RForest | KNN | DTree | SVC | MNBayes |
|---|---|---|---|---|---|---|
| 1 | 0.63 | 0.50 | 0.36 | 0.32 | 0.33 | 0.27 |
| 2 | 0.65 | 0.51 | 0.37 | 0.31 | 0.42 | 0.26 |
| 3 | 0.59 | 0.50 | 0.33 | 0.26 | 0.41 | 0.26 |
| 4 | 0.56 | 0.56 | 0.34 | 0.22 | 0.41 | 0.28 |
| 5 | 0.59 | 0.59 | 0.37 | 0.24 | 0.38 | 0.28 |
| 6 | 0.83 | 0.83 | 0.35 | 0.32 | 0.41 | 0.28 |
| 7 | 0.96 | 0.96 | 0.40 | 0.32 | 0.53 | 0.28 |
| 8 | 0.64 | 0.64 | 0.35 | 0.29 | 0.32 | 0.25 |
| 9 | 0.55 | 0.55 | 0.44 | 0.25 | 0.43 | 0.27 |
| 10 | 0.67 | 0.67 | 0.35 | 0.25 | 0.34 | 0.25 |
| 11 | 0.59 | 0.59 | 0.31 | 0.19 | 0.37 | 0.27 |
| 12 | 0.55 | 0.55 | 0.33 | 0.18 | 0.40 | 0.28 |
| 13 | 0.56 | 0.56 | 0.32 | 0.24 | 0.42 | 0.27 |
| 14 | 0.53 | 0.53 | 0.35 | 0.26 | 0.46 | 0.26 |
| 15 | 0.73 | 0.73 | 0.33 | 0.25 | 0.46 | 0.28 |

Según los resultados de la prueba de Shapiro-Wilks en la Tabla 4, la distribución de los datos de la Tabla 3 no está equilibrada.

**Tabla 4**

*Prueba de Shapiro-Wilks a los resultados de los modelos*

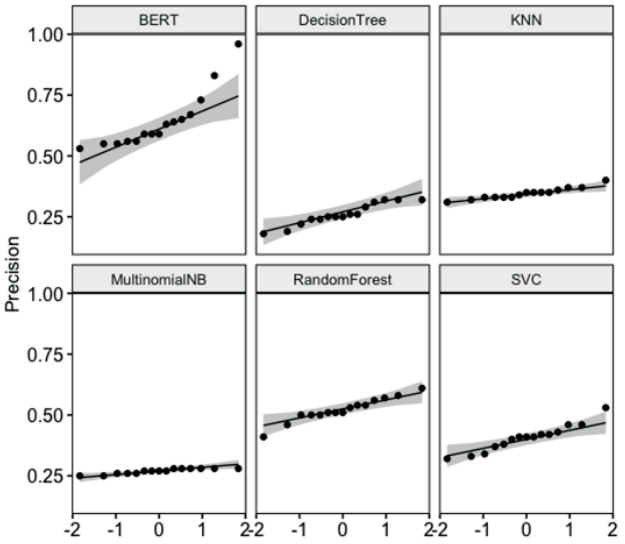
| Modelo | p-value |
|---|---|
| BERT | 0.00451 |
| Decision Tree | 0.207 |
| KNN | 0.449 |
| MultinomialNB | 0.0124 |
| RandomForest | 0.711 |
| SVC | 0.573 |

Esto es particularmente evidente en los valores obtenidos para los modelos BERT, MultinomialNB y DecisionTree que tienen valores p inferiores a 0.05, lo que sugiere que los datos de precisión para estos modelos no siguen una distribución normal. Esto indica que la distribución de estos métodos no está equilibrada, y esto está respaldado por la evidencia visual presentada en la Figura 4. Muestra claramente que los puntos de datos se desvían significativamente de la línea recta, lo que indica que la distribución de los datos para estos métodos no es normal.

Como la prueba de Shapiro-Wilks demostró que los resultados no estaban equilibrados, utilizamos una prueba no paramétrica (prueba de Friedman) para saber si existen diferencias significativas. El estadístico de Friedman es 71.4, con 5 grados de libertad y un valor p muy bajo (5.24e – 14), lo que sugiere una alta significancia estadística. El resultado indica que hay una diferencia significativa entre los grupos ($p < 0.05$) y se puede continuar con pruebas post hoc para determinar cuáles pares de modelos son significativamente diferentes entre sí. Para identificar los pares de modelos que difieren significativamente, se ejecutó una prueba post-hoc de Wilcoxon con ajuste de Bonferroni y los resultados se presentan en la Tabla 5. De acuerdo con la tabla, los pares de modelos que son significativamente diferentes son aquellos que tienen una p-valor ajustado menor que 0.05. Basado en esto, BERT es significativamente diferente a todos los modelos especialmente a DecisionTree y a SVC.

***Fig 4***

*Gráfico que muestra el comportamiento de cada modelo respecto a la normalidad*




Fuente: Elaboración propia

**Tabla 5**

*Prueba de Wilcoxon con ajuste de Bonferroni*

| Comparación de Modelos | p-value-adj | Significancia |
|---|---|---|
| BERT vs DecisionTree | 9.15 e-4 | *** |
| BERT vs KNN | 1.1 e-2 | * |
| BERT vs MultinomialNB | 4.3 e-2 | * |
| BERT vs RandomForest | 9.15 e-4 | * |
| BERT vs SVC | 1.1 e-2 | *** |
| DecisionTree vs KNN | 1 e+0 | * |
| DecisionTree vs MultinomialNB | 1.1 e-2 | ns |
| DecisionTree vs RandomForest | 1.1 e-2 | * |
| DecisionTree vs SVC | 1.1 e-2 | * |
| KNN vs | 1.1 e-2 | * |
| KNN vs RandomForest | | * |
| KNN vs SVC | 5.1 e-2 | ns |
| MultinomialNB vs RandomForest | 1.1 e-2 | * |
| MultinomialNB vs SVC | 1.1 e-2 | * |
| RandomForest vs SVC | 1.1 e-2 | * |

## 5. Conclusiones y Trabajo Futuro

Esta investigación ofrece una arquitectura respaldada por computación en la nube que admite modelos de aprendizaje automático y profundo, para recomendar problemas de programación a los jueces en línea.

Los resultados del estudio muestran que BERT es el modelo de aprendizaje automático más preciso para la clasificación de texto, especialmente cuando se combinan con técnicas de preprocesamiento de texto, como normalización, eliminación de Stopwords, tokenizacion y lematización. Los análisis estadísticos realizados para comparar la precisión de los diferentes modelos fueron fundamentales para demostrar la superioridad de BERT y para identificar las diferencias significativas entre los modelos. Además, la utilización de arquitectura de AWS en la nube para ejecutar los modelos permitió reducir el tiempo de ejecución y escalar de acuerdo a las necesidades. Por último, la clasificación de texto para evaluar el nivel de dificultad de los problemas permitió construir un sistema de recomendación para sugerir problemas similares a los ya resueltos, lo que puede ser de gran utilidad en la educación y en otros campos. En resumen, los resultados demuestran que BERT es un modelo altamente eficiente para la clasificación de texto y que la metodología estadística empleada en este estudio fue fundamental para obtener conclusiones significativas y confiables. Nuestro trabajo futuro se centrará en utilizar un enfoque de recomendación híbridos para comparar con los enfoques propuestos en este trabajo.

## Referencias


[1] G. L. McDowell, Cracking the coding interview: 189 programming questions and solutions. CareerCup, LLC, 2015.

[2] H. Wu, Y. Liu, L. Qiu, and Y. Liu, “Online judge system and its applications in c language teaching,” in 2016 International Symposium on Educational Technology (ISET). IEEE, 2016, pp. 57–60.

[3] R. Yera and L. Martínez, “A recommendation approach for program- ming online judges supported by data preprocessing techniques,” Applied Intelligence, vol. 47, no. 2, pp. 277–290, 2017.

[4] C. M. Intisar, Y. Watanobe, M. Poudel, and S. Bhalla, “Classification of programming problems based on topic modeling,” in Proceedings of the 2019 7th International Conference on Information and Education Technology, 2019, pp. 275–283.

[5] P. Fantozzi and L. Laura, “Collaborative recommendations in online judges using autoencoder neural networks,” in International Symposium on Distributed Computing and Artificial Intelligence. Springer, 2020, pp. 113–123.

[6] F. D. Pereira, H. B. Junior, L. Rodriguez, A. Toda, E. H. Oliveira, A. I. Cristea, D. B. Oliveira, L. S. Carvalho, S. C. Fonseca, A. Alamri et al., “A recommender system based on effort: Towards minimising negative affects and maximising achievement in cs1 learning,” in International Conference on Intelligent Tutoring Systems. Springer, 2021, pp. 466–480.

[7] M. Caro-Martinez and G. Jimenez-Diaz, “Similar users or similar items? comparing similarity-based approaches for recommender systems in online judges,” in International Conference on Case-Based Reasoning. Springer, 2017, pp. 92–107.

[8] M. F. Aljunid and M. Dh, “An efficient deep learning approach for collaborative filtering recommender system,” Procedia Computer Science, vol. 171, pp. 829–836, 2020.

[9] J. Bobadilla, F. Ortega, A. Gutiérrez, and S. Alonso, “Classification-based deep neural network architecture for collaborative filtering recommender systems,” 2020.

[10] M. Berbatova, “Overview on nlp techniques for content-based recommender systems for books,” in Proceedings of the Student Research Workshop Associated with RANLP 2019, 2019, pp. 55–61.

[11] F. D. Pereira, F. Pires, S. C. Fonseca, E. H. Oliveira, L. S. Carvalho, D. B. Oliveira, and A. I. Cristea, “Towards a human-ai hybrid system for categorising programming problems,” in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 2021, pp. 94–100.

[12] S. C. Fonseca, F. D. Pereira, E. H. Oliveira, D. B. Oliveira, L. S. Carvalho, and A. I. Cristea, “Automatic subject-based contextualisation of programming assignment lists.” International Educational Data Mining Society, 2020.

[13] H. B. de Freitas Júnior, F. D. Pereira, E. H. T. de Oliveira, D. B. F. de Oliveira, and L. S. G. de Carvalho, “Recomendação automática de problemas em juízes online usando processamento de linguagem natural e análise dirigida aos dados,” in Anais do XXXI simpósio brasileiro de informática na educação. SBC, 2020, pp. 1152–1161.

[14] S. González-Carvajal and E. C. Garrido-Merchán, “Comparing bert against traditional machine learning text classification,” arXiv preprint arXiv:2005.13012, 2020.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.

[16] M. Aamir and M. Bhusry, “Recommendation system: state of the art approach,” International Journal of Computer Applications, vol. 120, no. 12, 2015.

[17] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, Recommender systems: an introduction. Cambridge University Press, 2010.

[18] T. Anwar, V. Uma, M. Hussain, M. Pantula et al., “Collaborative filtering and knn based recommendation to overcome cold start and sparsity issues: A comparative analysis,” Multimedia Tools and Applications, pp. 1–19, 2022.

[19] N. C. B. Beltrán and E. C. R. Mojica, “Procesamiento del lenguaje nat- ural (pln)-gpt-3.: Aplicación en la ingeniería de software,” Tecnología Investigación y Academia, vol. 8, no. 1, pp. 18–37, 2020.

[20] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” ieee Computational intelligenCe magazine, vol. 13, no. 3, pp. 55–75, 2018.

[21] W. Julca-Mejia, H. D. Calderon-Vilca, and F. C. Cárdenas-Mariño, “Evaluation of source code in acm icpc style programming and training competitions,” in Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE 2018, 2018.

[22] F. Martínez-Plumed, L. Contreras-Ochando, C. Ferri, J. Hernández-Orallo, M. Kull, N. Lachiche, M. J. Ramírez-Quintana, and P. Flach, “Crisp-dm twenty years later: From data mining processes to data science trajectories,” IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 8, pp. 3048–3061, 2019.

[23] S. Ravichandiran, Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT. Packt Publishing Ltd, 2021.

[24] X. Luo, “Efficient english text classification using selected machine learning techniques,” Alexandria Engineering Journal, vol. 60, no. 3, pp. 3401–3409, 2021.

[25] B. Charbuty and A. Abdulazeez, “Classification based on decision tree algorithm for machine learning,” Journal of Applied Science and Technology Trends, vol. 2, no. 01, pp. 20–28, 2021.