Voorspelling, aplicación de escritorio para aprendizaje de máquina supervisado

Autores:

Iván David Rey Rueda

German Francisco Diaz Figueredo

Universidad de Santander

Facultad de Ingenierías

Ingeniería de Software

Bucaramanga, Santander

2020

Voorspelling, aplicación de escritorio para aprendizaje de máquina supervisado

Autores:

Iván David Rey-Rueda

German Francisco Diaz Figueredo

Trabajo de grado presentado como requisito parcial para obtener título de Ingeniero de Software

Directora de proyecto:

Doctora Nydia Paola Rondón Villarreal

Universidad de Santander

Facultad de Ingenierías

Ingeniería de Software

Bucaramanga, Santander

2020

Índice general

Índice de imágenes

Índice de tablas

Abstract

Keywords: Supervised Machine Learning, Desktop app, Scikitlearn.

Resumen

Palabras clave: Aprendizaje de Máquina Supervisado, Aplicación de escritorio, Scikitlearn.

Introducción

Planteamiento del problema

Programar es una de las tantas habilidades que un estudiante de ingeniería de software, sistemas o computación debe dominar, junto a otras áreas como: cálculo, álgebra, estadística y física (Özmen & Altun, 2014). Tan y col., 2009 en su estudio manifiestan que aprender un lenguaje de programación no es tarea fácil, y además es un proceso largo y tedioso de por lo menos algunos meses, se requiere mucha dedicación, especialmente para los estudiantes de primer año, ya que en ese punto los programadores novatos carecen de las habilidades necesarias para resolver problemas.

Debido a que la programación es un requisito obligatorio, aún con todos los obstáculos que presenta para los estudiantes de carreras afines, autores como McCracken y col., 2001 se formulan la pregunta: ¿Los estudiantes cumplen realmente con los conocimientos necesarios?. ¿Cuál fue el resultado de su estudio? los estudiantes obtuvieron en promedio 22.9 de 110 puntos con un desviación estándar de 25.2, mucho más bajo de lo que esperaban en un principio, por diferentes motivos tales como: los estudiantes no evalúan todas las rutas del programa, el código no es legible, malas prácticas, falta de pruebas de software y problemas relacionados a pobre capacidad matemática. De igual forma Krpan y col., 2015 manifiestan que aprender a programar al nivel universitario es todo un reto para los estudiantes, especialmente para aquellos sin ninguna experiencia previa en algún lenguaje, y por otro lado, también hay que agregar la dificultad añadida de las primeras materias que cursa todo estudiante en su ciclo básico. En este periodo Krpan expresa que se encuentra el mayor rango de deserción, siendo una de las principales razones la dificultad con el pensamiento abstracto. Por esas y demás razones, los docentes y profesionales tienen la tarea de diseñar o implementar formas de enseñanza alternativas que mejoren significativamente las habilidades de programación de los estudiantes, y de igual forma incorporar nuevos conocimientos y conceptos que les permitan entender los lenguajes de programación como si de su primera lengua se tratase. aunque desafortunadamente aún no hay una solución definitiva, a pesar que a lo largo de los años se han intentado establecer nuevas metodologías para el aprendizaje.

Los estudiantes que se adentran dentro del mundo de la programación, en particular en un

lenguaje como lo es Python al nivel universitario encuentran una serie de desafíos, desde problemas simples con el código, hasta problemas debido a conceptos mal establecidos (Piwek & Savage, 2020). Es esta falta de habilidades y conocimientos en un tema específico, como lo es en este caso el aprendizaje de máquina lo que lleva a los estudiantes a (1) renunciar, (2) cambiar completamente su enfoque profesional, o si en definitiva continúan por el camino de la inteligencia artificial, (3) tener que toparse con aplicaciones que no satisfacen sus necesidades de aprendizaje, y que en muchos casos no son asequibles. Por lo anterior, estudiantes y entusiastas en sus primeros años de formación académica que desean desarrollar sus propios modelos de aprendizaje de máquina y que además no precisan de las habilidades necesarias para escribir su propio código, se ven en la necesidad de recurrir a software de ese tipo para ejecutar algoritmos de inteligencia artificial, pero la gran mayoría de aplicaciones disponibles son pagas o en algunos casos con pruebas gratuitas muy limitadas, por lo que después de unos pocos usos se hace necesaria una suscripción, a causa de que es necesario como mínimo generar los ingresos para mantener en servicio la aplicación, situación que es más evidente en el caso de las aplicaciones web que en las de escritorio.

Justificación

En el transcurso de los últimos 10 años diferentes estrategias de aprendizaje como: videojuegos educativos, aplicaciones web, simulaciones, técnicas de visualización y programación en parejas han sido implementadas para obtener la atención del estudiante y desarrollar el pensamiento creativo como elemento para prepáralos a ser futuros desarrolladores, no sólo simples consumidores de tecnología (Salleh y col., 2013). Tal es el caso de Weka que inicia como aplicación para el análisis de datos, pero que más adelante implementa por primera vez un algoritmo de aprendizaje de máquina automático (Thornton y col., 2013). A pesar de ser novedoso en su tiempo, otras aplicaciones con fines similares, pero esta vez en ambientes web se incorporan al conjunto de aplicaciones disponibles para el uso del público. Sin embargo, estas aplicaciones no son concebidas con el fin de ser utilizadas por estudiantes, sino que son creadas para ambientes empresariales donde realmente se requiere de grandes cantidades de recursos físicos para ejecutar esta clase de algoritmos. Por lo que el uso para usuarios diferentes a los inicialmente pensados por parte de estas aplicaciones produce como consecuencia que se vean obligados a utilizar esos medios hasta que se agote la prueba gratuita o tener que recurrir a medios tradicionales como crear código fuente para entrenar los modelos, opción poco factible debido a la falta de conocimientos que presentan los estudiantes en sus primeros años de formación profesional.

El desarrollo de aplicaciones para aprendizaje de máquina en la última década se debe a que la programación está fuertemente ligada a los ambientes de desarrollo, el lenguaje y las herramientas, siendo las herramientas las que juegan un papel de mayor relevancia al facilitar el aprendizaje de un tema en específico. Es por esto que autores como Francis y col., 1983; Salleh y col., 2013 mencionan que las herramientas para facilitar el aprendizaje son elementos esenciales debido a que están relacionadas con el uso de ambientes de desarrollo requeridos para crear aplicaciones con las cuales los estudiantes pueden resolver problemas, analizarlos, evaluarlos y expresar sus ideas con mayor claridad, caso opuesto cuando se expone a los estudiantes a interactuar directamente con el código sin tener las bases suficientes. No obstante, las aplicaciones disponibles actualmente en la

web como Google Cloud, MLJar, H20.ai y Auger.ai presentan 4 claras desventajas, las cuales son: (1) la integridad de los datos no se garantiza, ya que la información abandona el equipo del usuario debido al intercambio entre cliente-servidor. Esta situación se presenta en el caso de los conjuntos de datos (*datasets*), los cuales pueden contener datos sensibles como información personal. (2) Los tiempos de ejecución, debido a que las aplicaciones de acceso gratuito ofrecen solo una fracción de sus recursos disponibles. (3) Las aplicaciones están construidas con el fin de ser usadas en contextos empresariales, por lo que su uso en ambientes académicos no es significativo. (4) Los altos costes de ejecutar entrenamiento y predicciones de aprendizaje de máquina supervisado en ambientes web, ya sea utilizando arquitecturas *serverless* o basadas en micro-servicios, se requiere obligatoriamente del uso de recursos físicos en los servidores donde se aloja la aplicación, lo que supone costos monetarios por mantener disponible el servicio, más el propio uso de la aplicación por parte de sus clientes.

Objetivo

Objetivo general

Desarrollar una aplicación de escritorio para aprendizaje de máquina supervisado en el sistema operativo *Windows 10* x64, que permita a los usuarios la selección del modelo, hiper-parámetros, entrenamiento y predicción con base al conjunto de datos suministrado.

Objetivos específicos

- Establecer los requerimientos funcionales y no funcionales de la aplicación de aprendizaje de máquina supervisado.
- Diseñar la interfaz gráfica y el *back-end* de la aplicación, con referencia a patrones y buenas prácticas para su construcción.
- Desarrollar el código fuente de la aplicación con base al diseño establecido.
- Ejecutar pruebas de validación, integración y regresión, garantizando el funcionamiento y calidad de la aplicación.
- Generar la versión ejecutable dada la completitud de las fases de requerimientos, diseño, código, documentación y pruebas.

Marco teórico

Diseño metodológico

En el presente proyecto se plantea desarrollar una aplicación de aprendizaje de máquina supervisado de escritorio para el sistema operativo *Windows* 10 x64 en el transcurso de 8 meses a partir de Agosto del 2020. Esta aplicación tiene como finalidad permitir a estudiantes y entusiastas en su primeros años de formación académica en carreras como ingeniería de software, sistemas, informática y computación, la selección del modelo, hiper-parámetros, entrenamiento y predicción con base al conjunto de datos suministrado, por medio de una interfaz de usuario sencilla de comprender y documentación detallada de las funciones incorporadas. Para esto se tendrán en cuenta las siguientes etapas: Inicio de proyecto, Diseño y planificación, Código e implementación, Evaluación y pruebas, y por último Conclusión del proyecto.

Inicio de proyecto

La primera etapa para el desarrollo del proyecto está compuesta de 5 actividades relacionadas al levantamiento de requerimientos y definición del alcance, actividades que tienen un componente directamente relacionado con reuniones de equipo y validación. Esta etapa se realizará a partir de las actividades descritas a continuación:

- Actividad 1, Reunión inicial de requerimientos: Esta actividad consiste en la ejecución de la primera reunión del equipo, utilizando Zoom como medio para reunir a los integrantes de forma virtual, debido a la situación global del año 2020.

- Activad 2, Planteamiento de la metodología de desarrollo ágil: A partir de las metodologías de desarrollo ágil existentes, y las necesidades establecidas con anterioridad, se define la metodología a trabajar utilizando reuniones de equipo. Esta actividad es paralela a la reunión inicial de requerimientos y puede extenderse hasta la validación de los requerimientos al final de la etapa.

- Actividad 3, Definición de tecnologías, lenguajes y *frameworks*: Teniendo en cuenta los

requerimientos de la aplicación que se va a construir, se establecen el lenguaje de programación, los entornos de desarrollo, librerías, *plugins* y *frameworks* de acuerdo a los conocimientos actuales de los integrantes y de búsquedas en la web para posibles soluciones al problema planteado.

- Actividad 4, Validación de requerimientos establecidos: Antes de continuar con las siguientes fases del ciclo de vida del software, se valida con todo el equipo a través de reuniones virtuales, los requerimientos, metodología, tecnologías, lenguajes y *frameworks* que se utilizaran a lo largo del desarrollo del proyecto.

- Actividad 5, Creación del reporte de requerimientos: Una vez validado los requerimientos, se diligencia la documentación de los requerimientos con una versión simplificada de la norma técnica Colombiana para la documentación de requerimientos de software. Adicionalmente, una vez el reporte se encuentre terminado, este debe ser firmado por el director de proyecto para su validación.

Diseño y planificación

Esta etapa está conformada por 7 actividades relacionadas a la planeación, el diseño, documentación y validación. Por otro lado, esta etapa es la que mayor duración tiene a comparación de las otras, por lo que al menos se espera invertir un 30 % del tiempo en el diseño. Lo anteriormente mencionado se realizara a partir de las siguientes actividades:

- Actividad 6, Planteamiento cronograma de actividades: Con base a los requerimientos, así como todas las funcionalidades del software, se establece el cronograma de actividades para un periodo de 8 meses a partir de Agosto del 2020.

- Actividad 7, Construcción del primer borrador: Elaboración del primer *wireframe*, así como el posible flujo de la aplicación entre cada formulario. Este borrador a diferencia de un *Mockup* no es diseñado en herramientas de prototipado como *Figma*, sino que es un boceto creado a mano.

- Actividad 8: Diseño de interfaz preliminar: Primer diseño de la interfaz de usuario creada en *Figma*, teniendo en cuenta la ISO 11581-10:2010 e 9241-210:2019 para la creación de interfaces de usuario. Este diseño debe de ser validado constantemente durante su desarrollo hasta ser aceptado para su posterior aceptación como interfaz final, ya que no se tiene planeado regresar a etapas anteriores una vez se avance lo suficiente en el desarrollo de la aplicación.
- Actividad 9, Validación de interfaz de usuario: A través de reuniones de equipo en Zoom, se aprueba el diseño presentado como la interfaz final que se implementara en *QT Designer* para su posterior integración con el código fuente.
- Actividad 10, Diseño del *back-end*: Teniendo en cuenta las entradas y salidas de la aplicación, así como el diseño de interfaz de usuario aceptado hasta el momento, se diseña el código fuente del *back-end* utilizando diagramas UML de clases, y de igual forma se debe tener en cuenta patrones de diseño para su creación.
- Actividad 11, Validación final del diseño: una vez terminada las actividades relacionadas al diseño, se realiza la validación final para su posterior puesta en producción por parte del equipo de trabajo, a través de reuniones en Zoom. La evidencia de esas reuniones debe de ser entregada al final de cada etapa, al igual como sucede con otras etapas.
- Actividad 12, Construcción del reporte de diseño: Con base a los diagramas UML de casos de uso y clase, así como la documentación generada a partir de los documentos creados a partir de la norma técnica Colombiana, se genera un reporte con los documentos anteriormente mencionados que debe de ser firmado y validado por el director de proyecto.

Código e implementación

Esta etapa consiste en 4 actividades, siendo estas en su mayoría la creación de código fuente, tanto para el *front-end* como para el *back-end*, pero aun así se deben realizar labores de documentación y revisiones de código, con el fin de producir código de calidad. Esta etapa la cual tiene una duración de tiempo menor al diseño se conforma de las actividades a continuación:

- Actividad 13, Construcción de pruebas y código fuente: Con base al diseño previamente establecido, se genera el código fuente de la aplicación al mismo tiempo que se crean pruebas de validación e integración para su posterior ejecución. Este código debe de seguir las normas de estilo PEP8 para *Python*, el cual es el principal lenguaje que se utiliza en el proyecto.

- Actividad 14, Revisión del código desarrollado: A través de revisiones con el equipo utilizando Zoom y las herramientas proporcionadas en *GitHub*, se analiza línea a línea el código generado con el fin de evitar cambios futuros debido a malas prácticas.

- Actividad 15, Documentación del código fuente: Después de haberse realizado las actividades relacionadas a la creación del código fuente y sus revisiones, se realiza la documentación detallada para cada función, método y clase, con el objetivo de presentar código fuente más legible, y que además pueda ser entendido con mayor facilidad por los demás miembros del equipo

- Actividad 16, Integración *front-end* y *back-end*: Después de disponer del código fuente tanto del *front-end* como del *back-end* funcionando por separado, se realiza la integración de ambos en un solo ambiente. Adicionalmente deben escribirse pruebas de software relacionadas a la integración del sistema para garantizar el funcionamiento de la aplicación.

Evaluación y pruebas

En el transcurso de esta etapa la cual cuenta con 4 actividades, se centran los esfuerzos del equipo en ejecutar pruebas de software, corregir los errores producto de la integración entre el *front-end* y el *back-end*, y finalmente documentar los resultados. Está penúltima etapa del desarrollo del proyecto está conformada por las siguientes actividades:

- Actividad 17, Ejecución pruebas de software: Después de haber integrado el sistema y haber realizado la debida documentación, se ejecutan nuevamente todas las pruebas creadas hasta el momento, pero adicionalmente se realiza pruebas de regresión, así como pruebas de caja negra donde se pruebe la aplicación paso a paso por cada una de las rutas existentes.

- Actividad 18, Corrección de errores: Esta actividad consiste en la corrección del código fuente con base a los resultados obtenidos en la ejecución de pruebas de software. La mayoría de errores deberán de haberse generado por la integración del *front-end* y *back-end* en un solo entorno, por lo que la corrección debe de estar centrada en mayor medida en ese componente.

- Actividad 19, Validación de los cambios: A través de revisiones de código con el equipo, se revisan los cambios generados con base a los resultados de las pruebas de software.

- Actividad 20, Documentación de las pruebas: Con base al código generado en anteriores actividades y los resultados de las pruebas antes y después de correcciones, se genera la documentación de las pruebas a partir de una versión simplificada de la norma técnica Colombiana para pruebas de software.

Conclusión del proyecto

La última etapa del presente proyecto se conforma de 4 actividades, que se desarrollaran en el último mes antes de cumplir con la fecha límite. Esta etapa es la de menor duración, pero aún así es posible que tome lugar alrededor de todo el último mes en caso de haber errores inesperados que afecten la experiencia de usuario. Las actividades son:

- Actividad 21, Compilación del ejecutable: una vez que la aplicación cumpla con los requerimientos establecidos y se hayan obtenido resultados positivos de las pruebas de software, se inicia el proceso de compilar la aplicación en un archivo ejecutable para el sistema operativo *Windows* 10 x64.

- Actividad 22, Validación final de funcionalidades: Utilizando máquinas virtuales y equipos ajenos utilizados en el desarrollo, la aplicación se pone bajo pruebas de usuario final para validar su correcto funcionamiento. En caso de existir errores inesperados se debe de volver a realizar la compilación del ejecutable con los cambios pertinentes, hasta que los inconvenientes sean menores a la cantidad mínima requerida para su validación final.

- Actividad 23, Creación del manual de usuario: Considerando las funcionalidades de la aplicación, al igual que la rutas posibles que se ofrecen, se crea un manual de usuario que contenga la guía de instalación del software, los requerimientos mínimos del sistema y también los procedimientos paso a paso para el uso correcto de la aplicación.
- Actividad 24, Despliegue de la aplicación: Dada la completitud de todas las etapas y sus actividades, se libera la aplicación para el uso del público en el repositorio de *GitHub* bajo la licencia BSD 3. Adicionalmente se entrega la documentación generada, el código fuente y el instalador de la aplicación a la institución de educación superior donde se desarrolla el proyecto.

Conclusión

Referencias

Francis, G. K., Abelson, H. & DiSessa, A. (1983). *Turtle Geometry. The Computer as a Medium for Exploring Mathematics.* (Vol. 90). https://doi.org/10.2307/2975591

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125-180. https://doi.org/10.1145/572139.572181

Tan, P. H., Ting, C. Y. & Ling, S. W. (2009). Learning difficulties in programming courses: Undergraduates' perspective and perception. *ICCTD 2009 International Conference on Computer Technology and Development*, *1*(May 2019), 1-5. https://doi.org/10.1109/ICCTD.2009.188

Salleh, S. M., Shukur, Z. & Judi, H. M. (2013). Analysis of Research in Programming Teaching Tools: An Initial Review. *Procedia - Social and Behavioral Sciences*, *103*(November), 127-135. https://doi.org/10.1016/j.sbspro.2013.10.317

Thornton, C., Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2013). Auto-WEKA, 1-9. https://doi.org/10.1145/2487575.2487629

Özmen, B. & Altun, A. (2014). Undergraduate Students' Experiences in Programming: Difficulties and Obstacles. *Turkish Online Journal of Qualitative Inquiry*, *5*(3). https://doi.org/10.17569/tojqi.20328

Krpan, D., Mladenović, S. & Rosić, M. (2015). Undergraduate Programming Courses, Students' Perception and Success. *Procedia - Social and Behavioral Sciences*, *174*(June 2014), 3868-3872. https://doi.org/10.1016/j.sbspro.2015.01.1126

Piwek, P. & Savage, S. (2020). Challenges with learning to program and problem solve: An analysis of student online discussions. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 1-6. https://doi.org/10.1145/3328778.3366838