Voorspelling, Aplicación de Escritorio para Aprendizaje de Máquina Supervisado

Autores:

Iván David Rey Rueda y

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 y

German Francisco Diaz Figueredo

Anteproyecto 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 figuras

Índice de cuadros

Abstract

The lack of skills and knowledge in a specific subject, as it is in the case of Machine Learning, leads students to change their learning path in most cases, or in the best case scenario if students keep studying Artifical Intelligence, the biggest obstacle on the way are applications that do not meet their academic needs and have high cost prices. It is for such reasons that students are forced to use these means until the free trial runs out, otherwise they have to resort to traditional means such as creating source code to train models, which is not a doable option due to the lack of knowledge that students have in their first years of professional training. It is mainly for these reasons that in the present project it is planned to develop a desktop application for the Windows 10 x64 operative system in the course of eight months starting from August 2020, with the purpose of allowing model selection, hyperparameters, training and prediction based on the supplied dataset to students in their first years of academic courses in careers such as Software Engineering, Systems Engineering and Computer Science, throughout and easy-to-understand user interface and detailed documentation of the built-in functions.

Keywords: Supervised Machine Learning, Desktop app, Python.

Resumen

La falta de habilidades y conocimientos en un tema específico, como lo es en este caso el Aprendizaje de Máquina lleva a los estudiantes a cambiar su ruta de aprendizaje en la mayoría de los casos, o si en definitiva se continua por el camino de la Inteligencia Artificial, el mayor obstáculo en el camino son las aplicaciones con costos elevados que no satisfacen las necesidades de aprendizaje. Por tales motivos, los estudiantes se ven obligados a utilizar esos medios hasta agotar 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. Por lo tanto, en el presente proyecto se plantea desarrollar una aplicación de escritorio para el sistema operativo Windows 10 x64 en el transcurso de 8 meses a partir de Agosto del 2020, con el fin de permitir a estudiantes en su primeros años de formación académica en carreras como Ingeniería de Software, Sistemas, Informática y Computación, la selección del modelo, hiperparámetros, 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.

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

Introducción

En toda enseñanza el uso de herramientas juega un papel fundamental. A partir de esa idea se considera desarrollar el proyecto Voorspelling como una alternativa a las aplicaciones ya existentes, con el fin de facilitar la formación de estudiantes en Aprendizaje de Máquina Supervisado. Este proyecto busca enseñar el procedimiento necesario para crear un modelo de Aprendizaje de Máquina Supervisado, otorgando la posibilidad de realizar el proceso paso a paso o de forma automática a partir de las opciones disponibles para construir un modelo de Inteligencia Artificial. Los estudiantes dispuestos a recorrer la formación en Aprendizaje de Máquina suelen enfrentarse a más problemas de los que pueden prever en un inicio. Esto se debe a que los estudiantes en los primeros semestres de formación profesional en carreras como Ingeniería de Software, Sistemas, Informática y Computación, carecen de los conocimientos y la guía necesaria para la comprensión de temas relacionados a la Inteligencia Artificial. A pesar de que la información existe, es poco accesible en algunos casos y en otros la documentación se encuentra desactualizada, lo cual obstaculiza el proceso de aprendizaje y lleva a los estudiantes al punto de desistir.

Voorspelling es desarrollado por un equipo conformado por dos desarrolladores y un director de proyecto, utilizando la metodología Scrumban y un ciclo de desarrollo a partir de etapas compuestas por actividades estrechamente relacionadas con los objetivos específicos del proyecto. Esta aplicación es desarrollada para el sistema operativo Windows 10 de 64 bits, utilizando el lenguaje Python tanto para el *front-end* como para el *back-end*, el entorno de desarrollo integrado empleado es PyCharm y el *framework* para crear la interfaz de usuario es Qt Designer. Con Voorspelling se espera que estudiantes universitarios puedan tener mejor compresión en la creación de modelos de Aprendizaje Máquina Supervisado, creen sus propios modelos, puedan adentrarse en la solución de problemas y los utilicen en futuros campos de trabajo. Adicionalmente, esta aplicación aprovecha los recursos del equipo donde es ejecutado, puede crearse modelos que utilicen conjuntos de datos a nivel empresarial si se considera necesario, y por último, carece de las desventajas que presentan las aplicaciones web que ofrecen servicios afines.

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) en su estudio titulado "*A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*" se formulan la pregunta: ¿Los estudiantes cumplen realmente con los conocimientos necesarios?. En el estudio previamente mencionado, los estudiantes obtuvieron en promedio 22.9 de 110 puntos con una 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 con 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. Desafortunadamente aún no hay una solución definitiva, a pesar de que a lo largo de los años se han intentado establecer nuevas metodologías para el aprendizaje.

Los estudiantes que se adentran al mundo de la programación, en particular en un lenguaje como 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 en este caso el Aprendizaje de Máquina, la razón que conduce a los estudiantes en sus primeros años de formación profesional a afrontar tres situaciones particulares (a) renunciar (b) cambiar completamente el enfoque profesional (c) toparse con aplicaciones que no satisfacen las necesidades de aprendizaje, y que en muchos casos no son asequibles. Por las razones anteriormente mencionadas, los estudiantes de carreras profesionales como Ingeniería de Software, Sistemas, Informática y Computación, 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 las aplicaciones disponibles en la web. Sin embargo, la gran mayoría de las aplicaciones 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 prepararlos a ser futuros desarrolladores, no sólo simples consumidores de tecnología (Salleh y col., 2013). Tal es el caso de Weka (Hall y col., 2009) 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, 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 tal razón, el uso de 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 está fuertemente ligada a la programación, los ambientes de desarrollo y las herramientas, siendo estas ultimas las que juegan un papel de mayor relevancia al facilitar el aprendizaje de un tema en específico. Por tal razón, autores como Francis y col. (1983) y 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, H20.ai y Auger.ai presentan cuatro claras desventajas, las cuales son (a) 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, los cuales pueden contener datos sensibles como información personal; (b) los tiempos de ejecución, debido a que las aplicaciones de acceso gratuito ofrecen solo una fracción de sus recursos disponibles; (c) las aplicaciones están construidas con el fin de ser usadas en contextos empresariales, por lo que su uso en ambientes académicos no es significativo; (d) los altos costos de ejecutar entrenamiento y predicciones de Aprendizaje de Máquina en ambientes web, ya sea utilizando arquitecturas *serverless* o basadas en micro-servicios.

Las desventajas mencionadas anteriormente están relacionadas con el inconveniente de requerir obligatoriamente del uso de recursos físicos en los servidores donde se aloja la aplicación. Este uso de recursos supone costos monetarios por mantener disponible el servicio, más el propio uso de la aplicación por parte de sus clientes. Por lo tanto, ofrecer precios razonables y servicios que se ejecuten con la mayor velocidad posible no son factibles de comercializar, dado que llevan a las organizaciones a tener pérdidas millonarias si no se efectúa el cobro correspondiente a sus usuarios. Por tales motivos, el enfoque de este proyecto se dirige por una ruta diferente a las aplicaciones web, dado que una aplicación de escritorio no presenta tales inconvenientes.

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, hiperparámetros, 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

Python

Definición

Según la documentación de Python ("Python 3.7 documentation", 2018) este lenguaje es la combinación de velocidad y una sintaxis fácil de comprender, tiene múltiples librerías, puede ejecutarse en diferentes sistemas operativos y es extensible a otros lenguaje como C y C++. Python (Van Rossum & Drake, 2009) es un lenguaje de programación de alto nivel interpretado multiparadigma, es decir, no hay nada que impida a los desarrolladores programar utilizando otros paradigmas, tales como lo son la programación procedural y funcional.

Por lo general, este lenguaje es utilizado en aplicaciones de consola, dado que no cuenta con la facilidad de otros lenguajes como PHP y JavaScript para tener acceso a lenguajes de hipertexto. En esos casos específicos donde Python es el lenguaje utilizado en el *back-end*, se emplean *frameworks* como Electron o librerías como EEl para comunicar las entradas del usuario a Python y luego regresar una salida que pueda ser comprendida en un ambiente con interfaz de usuario que utilice HTML y CSS para el *front-end*. No obstante, cuando los requerimientos de una aplicación necesitan de un ambiente nativo de escritorio y no es necesario utilizar tecnologías relacionadas a ambientes web, es entonces cuando otros *framework* que utilizan Python tanto para el *front-end* como para el *back-end* son una opción más óptima, dado que no se necesitan de otros lenguajes que aumentan la complejidad del problema.

Librerías

Conforme a lo mencionado en la documentación de Python ("Python 3.7 documentation", 2018), la librería estándar de este lenguaje contiene módulos originalmente construidos en C que aportan funcionalidades como el acceso a las rutas del sistema, que de otro modo no serían posible en este lenguaje. De igual forma, la librería estándar también tiene módulos desarrollados originalmente en Python, los cuales aportan soluciones para los problemas regulares que se enfrentan los desarrolladores en este lenguaje.

Algunos de los módulos por defecto en Python proveen varias interfaces específicas del

lenguaje, mientras que otros son del sistema operativo o módulos relacionados a ambientes web. Aunque gran parte de las librerías se encuentran disponibles directamente en el lenguaje, hay otras que necesitan primero ser descargadas e instaladas en el sistema, ya sea en un entorno virtual o en la instalación base de la distribución elegida a través de pip, el cual es el paquete estándar para administrar los módulos en Python.

Los módulos, paquetes y librerías que se utilizan en mayor medida en este proyecto de Aprendizaje de Máquina Supervisado son: numpy, pandas, scikitlearn, mljar-supervised, xgboost, tensorflow, seaborn, Pyinstaller, PyQt5, abc, random y json.

- Numpy: se utiliza para trabajar con vectores y matrices propias del álgebra linear en Python, en vez de utilizar las listas y tuplas que ofrece por defecto el lenguaje.
- Pandas: está diseñado para manipular estructuras de datos tabulares y multidimensionales, así como la información que contienen archivos csv y tsv.
- Scikitlearn: es una librería de acceso libre que utiliza principalmente Numpy y pandas. Esta librería proporciona una implementación de varios algoritmos conocidos de Aprendizaje de Máquina, al mismo tiempo que una interfaz fácil de utilizar integrada en Python.
- MLjar-supervised: es un paquete de Aprendizaje de Máquina Automático que trabaja principalmente con datos tabulares, es decir conjuntos de datos por lo general en formato csv. Está diseñada para ahorrar todo el tiempo posible a científicos de datos gracias a las funciones automatizadas que tiene el algoritmo, tales como búsqueda de hiperparámetros, construcción y ajuste del modelo.
- Xgboost: es una librería optimizada de gradientes estocástica multiplaforma diseñada para ser eficiente, flexible y portable.
- Tensorflow: es una librería de matemáticas desarrollada por el equipo de GoogleBrain. Esta librería es de acceso libre y es utilizada por lo general en Aprendizaje de Máquina y Redes Neuronales.
- Seaborn: es una librería para visualización de información tabular basada en matplotlib para Python, la cual brinda una interfaz de alto nivel para dibujar información estadística atractiva y

entendible para el usuario.

- PyInstaller: el principal objetivo de este módulo es ser compatible con aplicaciones de terceros sin necesidad de otras extensiones. Este módulo abstrae todos los paquetes de un proyecto y los deja listos para ser utilizados en un archivo ejecutable en diferentes sistemas como Mac Os, Windows, Solaris y AIX.
- PyQt5: Qt es un conjunto de librerías multiplataforma desarrollada en C++, mientras que PyQt5 es la librería para utilizar las funciones disponibles en Qt v5.
- Abc: este módulo provee la infraestructura para definir clases abstractas en Python.
- Random: este módulo hace parte del paquete estándar de Python. Su función es implementar generación de pseudo números aleatorios ya sea entre un rango de valores enteros o decimales.
- Json: es un formato liviano de intercambio de información inspirado en la notación clave-valor de JavaScript que hace parte del paquete estándar de Python, pero que es utilizado en todo lenguaje de programación.

La única dependencia que necesita ser obtenida por aparte para el desarrollo de la aplicación es Visual Studio Build Tools 2019, la cual es utilizada en la instalación de mljar-supervised. No obstante, las demás librerías mencionadas previamente si pueden ser instaladas sin ningún problema en un ambiente virtual utilizando pip (ver Figura 1), a través de un documento de requerimientos creado con anticipación.

Figura 1

Instalación de librerías con pip

```
PS C:\Users\folder> python -m venv envname
PS C:\Users\folder> .\envname\Scripts\activate
(envname) PS C:\Users\folder> pip install -r requirements.txt
```

Nota. Este proceso debe realizarse en la consola de comandos (cmd). No es un conjunto de instrucciones que pueda ser ejecutado en una sola línea, sino que las entradas deben ser efectuadas una después de la otra.

Ambiente de desarrollo

Un *framework* no es solo un ambiente donde se desarrolla aplicaciones con base a unas funcionalidades preestablecidas, sino que es una herramienta para facilitar la construcción de una aplicación de acuerdo con un lenguaje, que en este caso es Python. Toda aplicación necesita de un entorno de desarrollo, pero un editor de texto no es lo suficiente robusto para generar las ideas de un grupo de desarrolladores, por tal motivo, se combinan entornos de desarrollo integrado (IDE) como PyCharm con un *framework*, con el fin de generar software con el mínimo esfuerzo posible. En Python existen varios *frameworks* relacionados al desarrollo de aplicaciones de escritorio tales como Kivy, Tkinter y WxPython, pero el más cercano a las necesidades y conocimientos del equipo es Qt Designer (Nord & Chambe-Eng, 1995).

De acuerdo con la documentación de "QT Designer Manual" (2020), este *framework* es un entorno para desarrollar interfaz de usuario con base a los *widgets* disponibles de QT, los cuales pueden ser combinados con lenguajes como Python y C++, a través de la librería PyQt en sistemas operativos como Windows, Mac Os y Linux. Adicionalmente, permite diseñar a partir de arrastrar y soltar elementos como botones, campos de texto y cuadros, funcionalidad que no todos los *framework* de Python comparten, pero que si es utilizada en otros *framework* como Windows Forms de .net.

Aplicación en Aprendizaje de Máquina

Sebastian Raschka (2019) en su libro "*Python Machine Learning*" menciona que Python es considerado como el más popular en ciencias de datos e Inteligencia Artificial. Sin embargo, dado que es un lenguaje interpretado, no tiene la misma velocidad que lenguajes de bajo e intermedio nivel como Fortran y C++, aunque existen librerías como Numpy que solucionan parte del problema y le permiten al lenguaje ser competitivo en la velocidad de ejecución de los algoritmos.

En un principio todos los algoritmos relacionados a Inteligencia Artificial se encontraban disponibles principalmente en C++, pero con el transcurso del tiempo y la introducción de Python al mercado como uno de los lenguajes más enseñados a estudiantes de ciencias de la computación como primera opción por su fácil sintaxis, los algoritmos migran a este lenguaje como el principal

utilizado ya sea con Aprendizaje de Máquina Supervisado, no Supervisado, Reforzado y Redes Neuronales. Actualmente la librería más conocida y empleada en estos tipos de problemas es scikit-learn (Pedregosa y col., 2011), dado que proporciona una implementación de vanguardia y brinda una gran variedad de algoritmos de Aprendizaje de Máquina, a través de una interfaz simple de utilizar, que permite realizar comparaciones y generar resultados en el lenguaje de programación de Python.

En el presente proyecto de Aprendizaje de Máquina Supervisado, la principal librería que permite el desarrollo del back-end de la aplicación es scikit-learn, dadas las ventajas que mencionan previamente Pedregosa y col. (2011). Sin embargo, si los requerimientos y habilidades del equipo de desarrollo no estuvieran enfocadas en el desarrollo de software con Python, existen otras opciones de librería en otros lenguajes como C# y Java, aunque actualmente se encuentran en desuso y la documentación no es igual de comprensible.

Aprendizaje de Máquina

Definición

El Aprendizaje de Máquina (ML) según Ethem Alpaydin (2014) no solo está relacionado con los conjuntos de datos, sino que es una parte integra de la Inteligencia Artificial, es decir, es la construcción de una aproximación a partir de datos históricos que puede llegar a suministrar salidas correctas. Asimismo Bost y col. (2015) expresan que el Aprendizaje de Máquina se utiliza para diferentes tareas tales como detección de correo basura, reconocimiento facial y estimaciones financieras, pero de igual forma se utiliza en otras áreas como robótica y medicina, es decir, no está ligada a un solo campo del saber.

La aplicación de métodos Aprendizaje de Máquina a bases de datos se le conoce como Minería de Datos, puesto que la extracción de las materias primas extraídas en esos sitios son procesados con el fin de producir material más refinado. Este material refinado son los datos de entrada de todo modelo de Inteligencia Artificial, ya sea Aprendizaje de Máquina o Aprendizaje Profundo. Pero en vista de que esos datos representan una parte importante para la generación de modelos, es necesario garantizar la integridad y seguridad de esa información, debido a que pueden

contener información sensible de una organización o personas.

Tipos de problemas

Los tipos de problemas en Aprendizaje de Máquina se clasifican en tres grupos dependiendo del tipo de entrada y salida requerida. Según autores como Russell y Norvig (2003), Han y col. (2012) y Sebastian Raschka (2019) estos grupos son: Aprendizaje Supervisado, no Supervisado y Reforzado.

Aprendizaje Supervisado. Este tipo de aprendizaje es sinónimo de clasificación. El Aprendizaje Supervisado aporta como principal herramienta algoritmos con el fin de clasificar información etiquetada a partir de datos históricos. Por lo tanto, a partir de la información conocida se puede predecir los datos a futuro, ya sea un grupo, una cantidad numérica o una clase binaria.

Este tipo de algoritmo es el principal utilizado en el presente proyecto, no solo porque permite realizar gran parte de las tareas que todo estudiante en sus primeros años de formación académica necesita, sino que a diferencia de los demás, este es el que mayormente se utiliza en los cursos iniciales de ciencias de datos, mientras que los demás son de uso más avanzado.

Aprendizaje no Supervisado. Este tipo de aprendizaje es sinónimo de Agrupamiento. El proceso de aprendizaje se considera no supervisado dado que las entradas son datos no etiquetados. Debido a que no se ingresan datos etiquetados al algoritmo de aprendizaje, es el propio algoritmo que se encarga de clasificar los resultados. El Aprendizaje no Supervisado puede traer problemas como tener que descubrir los patrones ocultos en la información, pero sin importar ese inconveniente, puede ser la mejor elección para generar un modelo de Inteligencia Artificial.

Aprendizaje Reforzado. En este caso el modelo intenta tomar acciones viables para maximizar la recompensa en una situación en particular. Es aplicado por varios software como videojuegos para encontrar el mejor camino posible o comportamiento que se debería tomar en una situación especifica. El entrenamiento reforzado se diferencia del supervisado, dado que los datos de entrenamiento tienen sus respectivas predicciones o respuestas, mientras en el reforzado el modelo se entrena sin datos, es decir, decide que acciones realizar para cumplir con la tarea suministrada con base a la experiencia que acaba de obtener.

Tipos de predicción

La ruta general en Aprendizaje de Máquina Supervisado consiste en una serie de pasos que inicia con la obtención del conjunto de datos y termina con las métricas del modelo entrenado y probado. Conforme a lo mencionado por Oswald Campesato (2020) en su libro "*Python 3 for Machine Learning*" estos pasos son:

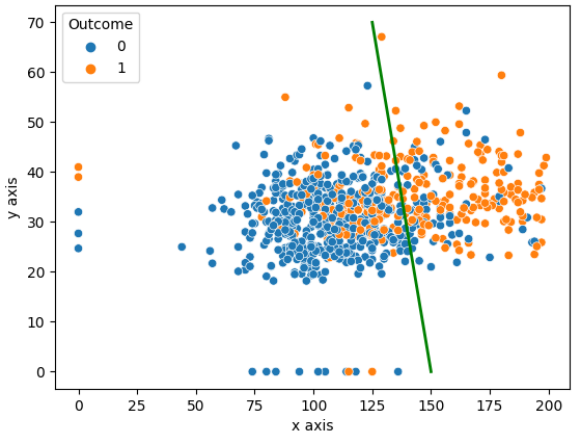
1. Obtener el conjunto de datos.
2. Limpiar el conjunto de datos.
3. Seleccionar las características y reducir la dimensionalidad.
4. Seleccionar el algoritmo de Aprendizaje de Máquina.
5. Separar el conjunto de datos en entrenamiento, validación y pruebas.
6. Entrenar el modelo.
7. Probar el modelo.
8. Optimizar el modelo y obtener el rendimiento.

Independientemente del proceso necesario para obtener las predicciones utilizando Aprendizaje de Máquina Supervisado, es necesario elegir el tipo de predicción que se va a realizar, seleccionar el modelo más óptimo para el conjunto de datos, ajustar los hiperparámetros y si es necesario reducir la dimensionalidad. Cada uno de los tipos de predicción tiene sus propios algoritmos basados en estadística y matemática, que de acuerdo con Géron (2019) y Oswald Campesato (2020) son Clasificación, Regresión y Agrupamiento, pero por simplicidad se consideran en este proyecto como cajas negras.

Clasificación. La salida obtenida es del tipo Booleano, es decir, uno, cero, verdadero, falso, si o no. La clasificación (ver Figura 2) en pocas palabras es asignar nuevas entradas al modelo para generar la predicción más probable con base a la clase, pero si hay un desbalance significativo entre las posibles salidas, lo más seguro es que el resultado no sea correcto.

Figura 2

Ejemplo de Clasificación



Nota. La recta de color verde representa el plano que separa los datos, entre aquellos que son verdadero con valor uno y falso con valor cero.

Este tipo de clasificación se utiliza para predecir una clase discreta o etiqueta que puede tener dos estados, pero si el problema lo requiere se puede utilizar clasificación multi-clase. La clasificación con múltiples clases a diferencia de la clasificación binaria, no tiene únicamente los estados esperado y no esperado, sino que las salidas son clasificadas en una de todas las clases disponibles.

Los modelos disponibles para Clasificación en Python a través de la librería scikit-learn (Buitinck y col., 2013) que se utilizan en este proyecto son:

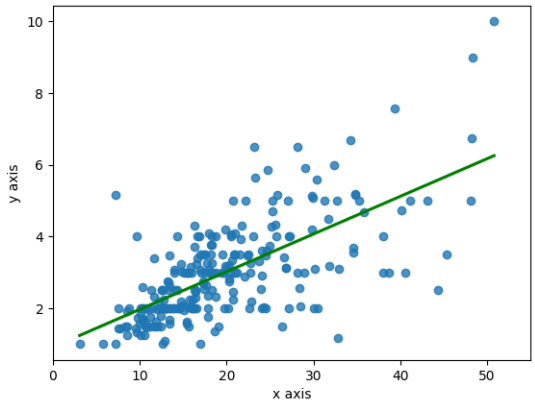
- SVC rbf: La implementación de este estimador esta basada en libsvm con kernel igual a *rbf*. El tiempo de entrenamiento se incrementa hasta el orden cuadrático cuando el número de muestras supera las diez mil unidades, por lo tanto es recomendable usar otras opciones si se presenta tal situación.
- LinearSVC: Similar a SVC con kernel igual a *linear*, pero implementado en términos de liblinear en vez de lbsvm, por ende adquiere más flexibilidad en la selección de penalizaciones y pérdidas de función, con lo que tiene la posibilidad de escalar mejor para grandes cantidades de muestras.

- KNN: *K Nearest Neighbours* Es un algoritmo que almacena los casos disponibles y realiza la clasificación de nuevos casos apoyándose en una medición de similaridad con base a la distancia *k* entre una muestra y la otra.
- Gaussian Naive Bayes: Este estimador hace parte de los métodos Naive Bayes. Estos algoritmos están basados en la implementación de teorema de Bayes con la suposición de una independencia condicional entre las características, y de igual forma el algoritmo de clasificación Naive Bayes Gaussiana supone que la probabilidad de las características es Gaussiana.

Regresión. En este tipo de predicción el resultado se calcula a partir de un modelo que minimice la función de pérdida. Dado que la salida de estos algoritmos es un número, la Regresión (ver Figura 3) puede ser empleada tanto en problemas para clasificar clases, como estimar cantidad numéricas. Por tal motivo, es utilizada para predecir todo tipo de situaciones donde la salida sean datos continuos, por ejemplo: el valor de un seguro de vida y el valor en bolsa.

Figura 3

Ejemplo de Regresión



Nota. La recta de color verde representa el comportamiento medio de los datos a partir de la función $f(x) = m \cdot x + b$ donde m es la pendiente y b el punto de corte con la abscisa.

Los modelos disponibles de Regresión en Python a través de la librería scikit-learn (Buitinck y col., 2013) que se utilizan en este proyecto son:

- Lasso: Es un tipo de regresión lineal que utiliza contracción de los valores con base a un punto central, al igual que la media. Este procedimiento fomenta modelos más simples y dispersos,

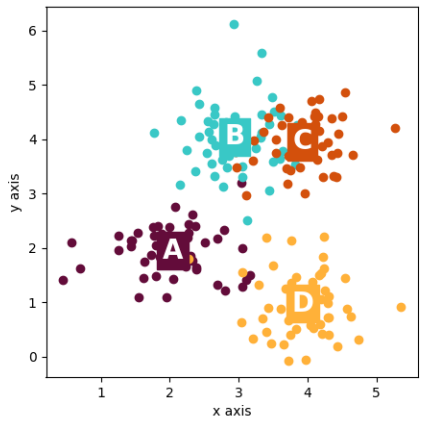
lo cual se traduce a modelos con menos parámetros.

- SGDClassifer: Este estimador implementa modelos lineales regulados, con un gradiente aleatorio de aprendizaje en descenso. El gradiente de las pérdidas se estima en cada muestra a la vez que modelo se actualiza en paraleo con la curva de aprendizaje.
- SVR rbf: Los parámetros en el modelo son *C* y *epsilon*. La implementación es basada en libsvm con kernel igual a *rbf*. El tiempo de entrenamiento supera el orden cuadrático, de manera que es difícil trabajar con este modelo cuando el número de muestras supera las diez mil unidades.
- LinearSVR: Similar a SVC con kernel igual a *linear*, pero implementado en términos de liblinear en vez de lbsvm. Permite mayor flexibilidad en la selección de penalizaciones y perdidas de función, y además escala mejor para grandes cantidades de muestras.

Agrupamiento. A este tipo de algoritmos se les conoce en inglés como *Clustering* (ver Figura 4). Su principal objetivo es entrenar un algoritmo para generar las agrupaciones deseadas, dado un conjunto de datos con sus respectivos datos históricos. La similitud de los grupos está dada por información adicional suministrada por el usuario, lo que puede convertirse en un problema de optimización de hiperparámetros, sobre todo si hay muchos atributos que deben considerarse para el entrenamiento.

Figura 4

Ejemplo de Agrupamiento



Nota. Cada *cluster* está separado según su color. En este ejemplo, los grupos que se quieren predecir son A, B, C y D.

Los modelos disponibles de Agrupamiento en Python a través de la librería scikit-learn (Buitinck y col., 2013) que se utilizan en este proyecto son:

- Affinity Propagation: crea grupos a través del envío de mensajes entre las parejas en las muestras hasta que se haya convergencia. Un conjunto de datos es luego descrito utilizando una parte de las muestras, las cuales son seleccionadas como las más representativas entre el conjunto total de muestras.
- Kmeans: Este algoritmo agrupa información tratando de separar muestras en un numero de grupos equivalente a la varianza, minimizando un criterio conocido como la inercia o suma de los cuadrados dentro de un grupo. Este algoritmo requiere un número de grupos especificados desde el inicio, pero trabaja bien para un gran número de muestras, a diferencia de otros modelos.
- Minibatch Kmeans: Este algoritmo es una variante del algoritmo KMeans, el cual utiliza minilotes para reducir el tiempo de compilación, mientras optimiza la función objetivo.
- Meanshift: Esta forma de agrupamiento tiene como fin descubrir irregularidades en una superficie de muestras. Este algoritmo está basado en datos centrales llamados centroides, los cuales funcionan convirtiendo los candidatos en la media de los puntos dentro de una determinada región. Estos candidatos son filtrados en una etapa de post-procesamiento con la finalidad de eliminar cualquier duplicado cercano del conjunto de centroides finales.

Selección de hiperparámetros

Wu y col. (2019) en su artículo "*Hyperparameter optimization for machine learning models based on Bayesian optimization*" mencionan que los hiperparámetros son de gran importancia en el Aprendizaje de Máquina, dado que se encargan de cambiar el comportamiento de los algoritmos cuando entrenan el modelo. Por lo tanto, los hiperparámetros tienen una relación directa en el rendimiento final de cualquier modelo de Aprendizaje de Máquina, y no deben ser establecidos en valores arbitrarios, sino que deben ser valores obtenidos de algoritmos tales como búsqueda Exhaustiva o Bayesiana.

Búsqueda exhaustiva. Este tipo de búsqueda es conocida como en inglés como *greedy*. La búsqueda Exhaustiva implica realizar una exploración utilizando todos los valores posibles dada

una lista de elementos S, pero en la mayoría de casos no se busca optimizar un solo hiperparámetro, sino que se requiere realizar una búsqueda para al menos dos o más listas de valores. Esto da como resultado tiempos de ejecución polinómicos, es decir $O(n^k)$ donde k es el número de listas de valores o espacios de parámetros y n la cantidad de elementos en el espacio S, si todos las listas tienen la misma longitud.

Este algoritmo se encuentra disponible en la librería scikit-learn (Buitinck y col., 2013) para Python, con dos diferentes opciones: GridSearchCV y RandomSearchCV. La opción elegida para este proyecto es GridSearchCV (ver Figura 5), dado que esta puede realizar una búsqueda exhaustiva con todo el espacio de valores, a diferencia de RandomSearchCV que escoge un subconjunto del espacio de valores con respecto a una semilla aleatoria.

Figura 5

Implementación de Búsqueda Exhaustiva

```
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
from sklearn.svm import SVC


X, y = load_iris(True)
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = SVC()

search_cv = GridSearchCV(svc, parameters)
```

Búsqueda Bayesiana. Este algoritmo está basado en el teorema de Bayes y se le conoce como búsqueda Bayesiana. A diferencia de una búsqueda Exhaustiva, es una de las estrategias más eficientes con respecto al número de evaluaciones necesarias. Esta búsqueda es utilizada comúnmente cuando las ya mencionadas evaluaciones requieren de grandes cantidades de tiempo, cuando el problema no es convexo y cuando no se tiene una expresión para la función objetivo, pero si se puede obtener la información de los eventos de la función (Brochu y col., 2010).

Este algoritmo se encuentra disponible en la librería scikit-learn (Buitinck y col., 2013) para Python con el nombre de BayesSearchCV (ver Figura 6), y aunque existan otras opciones de vanguardia, utilizar directamente la implementación de scikit-learn beneficia al proyecto en el

tiempo de desarrollo requerido para desplegar la aplicación.

Figura 6

Implementación de Búsqueda Bayesiana

```
from skopt import BayesSearchCV

from sklearn.datasets import load_iris
from sklearn.svm import SVC

X, y = load_iris(True)

search_cv = BayesSearchCV(
SVC(gamma='scale'),
search_spaces={'C': (0.01, 100.0, 'log-uniform')},
n_iter=10,
cv=3
)
```

Reducción de dimensionalidad

Existe la creencia respecto al volumen de los datos que más información es mejor, y por lo general es cierto, pero no en todos los casos esto se cumple. Un mayor número de observaciones redunda en un mejor modelo, pero un mayor número de variables no necesariamente lo hace mejor (Guerrero, 2016). Para conjuntos de datos con varias dimensiones, la reducción de dimensionalidad se ejecuta antes de aplicar los algoritmos de Aprendizaje de Máquina, con el propósito de evitar los problemas relacionados a la maldición de la dimensionalidad. Por tales motivos, autores en años anteriores como Finley y Joachims (2005) mencionan que la selección de características se convierte en la opción mas viable, especialmente en conjuntos de datos con alta dimensionalidad.

Existen diferentes tipos de métodos con los cuales se aborda el problema de la reducción de la dimensionalidad, sin embargo, de acuerdo con Finley y Joachims (2005) y Sánchez-Maroño y col. (2007) los más utilizados son los métodos de filtrado y envolventes. Estos métodos pretenden hallar un subconjunto de datos a partir del original, es decir, que el nuevo subconjunto candidato a ser la opción óptima pierde una o más características en el proceso. Esto da como resultado en el mejor de los casos un incremento en el rendimiento del clasificador, mientras que por otro lado, si el proceso de selección de características no es necesario desde el inicio, el resultado final es un clasificador con un rendimiento menor al inicial, dado que las características eliminadas eran significativas.

Métodos de filtrado. Los métodos de filtrado realizan un análisis de las características individuales para identificar su importancia relativa. Son computacionalmente eficaces e independientes del modelo, pero por otro lado, es posible que una característica no sea útil por si sola, aunque puede ser significativa cuando está asociada con otra. Esta situación representa una desventaja en los métodos de filtrado, ya que se puede perder la potencial importancia de dichas características.

Los algoritmos disponibles para métodos de filtrado en Python a través de la librería scikit-learn (Buitinck y col., 2013) son:

- VarianceThreshold: es un método para la selección de características que remueve toda aquella característica cuya varianza no supere cierto límite. Adicionalmente, por defecto elimina toda característica con varianza cero, es decir aquellas que tienen el mismo valor en todas las muestras y por ende no otorgan información relevante.
- SelectKBest: este algoritmo se encarga de remover todas las características que no pertenezcan a las k características con mayor peso. La salida puede ser un nuevo vector con las nuevas características o el peso individual de cada una de ellas.
- SelectPercentile: es un algoritmo que remueve toda característica que el usuario no seleccione entre las de mayor puntuación.
- GenericUnivariateSelect: este algoritmo utiliza una estrategia configurable para realizar una selección de características univariadas.

Métodos envolventes. Este tipo de métodos inician el proceso de evaluación de las características una a una, para luego seleccionar aquella que tiene el mejor rendimiento. Una vez terminado ese proceso se realiza la selección de las posibles combinaciones hasta que el modelo no tenga un mejor rendimiento con respecto a la última iteración.

Los algoritmos disponibles para métodos envolventes en Python a través de la librería scikit-learn (Buitinck y col., 2013) son:

- Recursive Feature Elimination: el objetivo de este método es la eliminación de características generando un conjunto de características cada vez más pequeño. Las características

menos importantes son eliminadas del conjunto de características actuales, proceso que se repite constantemente hasta llegar a la cantidad requerida de características

- SelectFromModel: este algoritmo puede ser utilizado con cualquier estimador que tenga los atributos coeficiente o importancia de característica, después de realizar el entrenamiento del modelo. Las características por defecto son consideradas no relevantes al no superar el umbral suministrado, mientras que aquellas que califican como importantes se mantienen.
- Sequential Feature Selector: es un algoritmo exhaustivo que se utiliza para reducir el espacio de características de cierta dimensionalidad a un espacio de menor dimensionalidad. Tiene como finalidad seleccionar automáticamente un subconjunto de características que son relevantes para el problema.
- Exhaustive Feature Selector: este algoritmo evalúa subconjuntos de características exhaustivamente aplicando todas las posibles combinaciones. El mejor subconjunto se selecciona a partir del rendimiento obtenido entre todas las iteraciones con el modelo seleccionado, ya sea Regresión o Clasificación.

Aprendizaje automático

Inicialmente todos los procesos de Aprendizaje de Máquina se realizaban paso a paso, pero con el desarrollo del primer algoritmo automatizado por parte de Thornton y col. (2013) en su trabajo titulado "*Auto-Weka*" el panorama cambia a uno donde es más efectivo utilizar algoritmos de aprendizaje automático para problemas de nivel empresarial. La ruta más fácil para seleccionar un estimador utilizada por estudiantes es definida a partir de la popularidad que tenga o que tan sencillo parece de utilizar, por consiguiente no se consideran las alternativas disponibles que pueden ser más eficientes. Parte de esos inconvenientes son remediados con los repositorios de acceso libre desarrollados para el uso de la comunidad, tales como: Weka (Hall y col., 2009), scikit-learn (Pedregosa y col., 2011) y mljar-supervised (Piotr y col., 2018), los cuales ofrecen paquetes de selección de características, hiperparámetros y modelo por medio de sus algoritmos de aprendizaje automático.

En trabajos posteriores Feurer y col. (2020) en su artículo "*Auto-Sklearn 2.0: The Next*

Generation" mencionan que a partir de mejoras en su algoritmo original de Auto-sklearn, obtienen mejoras significativas a partir de la implementación de una técnica de meta-aprendizaje más sencilla, la cual según los autores es el futuro para tener un Aprendizaje de Máquina automatizado cercano a la perfección y que además sea de libre acceso. No obstante, implementar esta librería actualmente no es posible dado que aún se encuentra en desarrollo, y pese a que la versión 1.0 se encuentra disponible desde el año 2016, es únicamente para sistemas operativos Linux. Por tales motivos, la librería utilizada para cubrir las funcionalidades de aprendizaje automático en el presente proyecto es mljar-supervised (ver Figura 7), la cual es desarrollada por Piotr y col. (2018) como un proyecto de software libre. Esta librería al igual que otros algoritmos de aprendizaje automatizado, tiene las funcionalidades de vanguardia necesarias para ejecutar optimización de hiperparámetros y selección de modelo, dado que si es necesario, los usuarios de *Voorspelling* pueden elegir entre un proceso paso a paso o automático; decisión que está relacionada al objetivo del proyecto.

Figura 7

Implementación de AutoML

```
from sklearn.datasets import load_iris
from supervised import AutoML


X, y = load_iris(True)
clf = AutoML(
            mode="Compete",
            explain_level=0,
            random_state=0,
            validation_strategy={
                "validation_type": "kfold",
                "k_folds": 10,
                "shuffle": False
            })
clf.fit(X,y)
```

Desarrollo de aplicaciones

Definición

El estudio o disciplina que comprende crear aplicaciones de software confiables y de calidad a través de etapas sistematizadas se conoce en el área de las ciencias de la computación como Ingeniería de Software (Sommerville y col., 2005). Una aplicación o también conocida como *app*, de acuerdo con Pressman (2002) es un tipo de software desarrollado con la función de ayudar al usuario en la realización de tareas determinadas y como la mayoría de las aplicaciones, estas son resultado de satisfacer una necesidad específica que desencadena una oportunidad de negocio. Su tamaño puede comprender desde códigos no muy extensos de funciones muy especializadas, hasta grandes obras de la ingeniería informática que toman un gran personal y muchas horas de trabajo para ser desarrolladas.

Las tareas realizadas por las aplicaciones abarcan un campo extenso que no se limita a áreas de la informática, aun así desarrollar una aplicación es labor de un programador o una compañía de Software, es decir, un software se crea por un especialista informático conocido como desarrollador, que por medio de herramientas como software y lenguajes de programación crea una aplicación la cual satisface unos requerimientos preestablecidos.

Tipos de aplicaciones

Las aplicaciones surgen de una de las ramas de los programas informáticos que se conocen como software. Aunque se pueden categorizar de muchas maneras tal como lo menciona (Pressman, 2002), se conoce que de acuerdo con sus fines prácticos se puede clasificar como:

- Software de sistema: rigen el comportamiento del sistema. Usualmente genera una separación entre el usuario y los componentes que conforman el ordenador.
- Software de aplicación: son programas informáticos hechos para desarrollar determinadas tareas. Su utilidad radica en la automatización o asistencia en procesos.
- Software de programación: este tipo de software es usado para desarrollar o modificar programas informáticos por medio de lenguajes de programación como Java, C++, C, C#, Ruby, Go, Fortran y Python.

Normas

En el desarrollo de aplicaciones las normas conforman unos estándares los cuales rigen el curso a tomar por los proyectos de software. Por tal motivo, la aplicación de las normas tiene como objetivo que el software desarrollado ofrezca una mayor confiabilidad, mantenibilidad, usabilidad, productividad y calidad.

Las normas de las cuales se abstrae los lineamientos para desarrollar software de calidad en el presente proyecto son:

- ISO/IEC 11581-10:2010: aporta una guía a los desarrollares y diseñadores para crear o usar iconos con base a los estándares de iconografía ("ISO/IEC 11581-10", 2010). Por otro lado, esta norma brinda una linea base para crear nuevas partes relacionadas a los iconos de un proyecto, dado que estos elementos no son solo símbolos, sino que son un medio para lograr un objetivo en una aplicación.
- ISO 9241-112:2017: aporta principios sobre la presentación de información en las interfaces de usuario. Al aplicar este principio se buscan interfaces más entendibles, precisas y más rápidas, debido a la reducción de esfuerzo mental y una mejor experiencia de usuario ("ISO 9241-112", 2017). En pocas palabras, es un estándar de usabilidad para no generar ambigüedades en el entendimiento de la información consecuente y extensa.
- ISO 9241-210:2019: aporta requerimientos y recomendaciones para los principios de diseño centrados en el ser humano, así como las actividades del ciclo de vida de sistemas interactivos basados en computadores ("ISO 9241-210", 2019). Esta norma está diseñada para ser utilizada por aquellos que gestionan el proceso de diseño de *software*, pero de igual forma está relacionada con las formas en que el *hardware* y *software* de sistemas interactivos pueden optimizar la interacción humano-máquina.

Por otro lado, dado que el lenguaje de programación principal de la aplicación de escritorio es Python, los estándares que utilizan los desarrolladores para generar código de calidad son: PEP8, PEP20, PEP257, PEP3131, PEP 484 y PEP 526.

- PEP8: es una guía para generar código en Python comprensible para todo desarrollador.

La guía tiene como finalidad mejorar la legibilidad del código, hacerlo consistente y más coherente (Reitz, 2001). La herramienta de autoPEP8 disponible en todo editor de texto utiliza pycodestyle para seleccionar que partes del código debe de ser reorganizados acorde al estilo de PEP8. Por lo tanto, no es necesario indagar profundamente si un archivo cumple o no con el estándar, dado que el proceso es automatizado.

- PEP20: este estándar es la guía de principios BDFL para diseño en Python en 20 aforismos. Algunos de estos son: Explicito es mejor que implícito, plano es mejor que anidado, la legibilidad cuenta, los errores nunca deben pasar desapercibidos, y si la implementación es fácil de explicar, entonces puede que sea una buena idea (Peters, 2004).
- PEP257: su objetivo es estandarizar la estructura de los comentarios de documentación en métodos, funciones y clases, es decir, qué deberían contener y cómo lo expresan (Goodger, 2001).
- PEP3131: este estándar expone que todos los identificadores en la biblioteca estándar de Python deben usar palabras en Inglés en formato ASCII (von Löwis, 2007). Adicionalmente los comentarios deben ir también en ASCII, aunque hay un par de excepciones: los casos de prueba y nombres propios.
- PEP484: este estándar tiene como objetivo establecer la sintaxis para las anotaciones, posibilitando que el código de Python sea más fácil de analizar, refactorizar y en algunos casos generar código a partir de realizar validación del tipo de variable. (Shannon, 2014).
- PEP526: en este estándar el tema principal son las anotaciones de variables. Según la documentación (Gonzalez y col., 2016) las notaciones para variables a nivel de módulo, clase, instancias y variables locales deben de tener un espaciado simple después de las comas y dos puntos, situación que es similar con los símbolos de igualdad, aunque en ese caso si hay espaciado antes y después.

Documentación

En el contexto del software, la documentación consiste en explicar cómo está compuesta y organizada una aplicación, siendo necesaria para generar el entendimiento del sistema de quienes lo vayan a usar y mantener. La documentación en muchos casos depende de los procesos de la

organización donde se desarrolla, pero aquellas que utilizan buenas prácticas seguramente respetan el sistema por medio del lenguaje de modelado unificado (UML), el cual brinda diagramas estandarizados, siendo los diagramas de caso de uso y clase los más frecuentemente aplicados, ya que aportan significativamente al entendimiento de la arquitectura del sistema (Rumbaugh y col., 2004). Aunque dependiendo del problema se pueden utilizar otros diagramas UML, tales como: secuencia, componentes, actividades y estado. En el presente proyecto los diagramas que se utilizan para describir el sistema y sus interacciones son el diagrama de casos de uso, clase y actividades.

Diagramas de caso de uso. Se utilizan para capturar el dinamismo de un sistema y busca plasmar los requisitos del sistema en un diseño de alto nivel, dado que idealizan la ruta del usuario al interactuar con el sistema. Este tipo de diagrama representa el conjunto de funcionalidades del sistema y el flujo que toma hasta llegar a un resultado. Sin embargo, los comportamientos en el diagrama de casos de uso tienden a ser de alto nivel y no se menciona la estructura lógica interna de estos.

Diagramas de clase. Estos diagramas describen los objetos en un sistema, mostrando los atributos y métodos que los componen, y adicionalmente las relaciones entre los objetos. Estos diagramas representan el sistema de una forma estática, por ende, no describe como se comportan los distintos elementos a lo largo de la ejecución, aun así son útiles, ya que son aplicables tanto para sistemas pequeños como grandes, y gracias a sus propiedades se transforma cómodamente el modelo a código fuente.

Diagramas de actividades. Descomponen las actividades y se utilizan para hacer un modelado de alto nivel de los requisitos. A este tipo de diagrama se les consideran diagramas de comportamientos ya que muestran el sistema de una manera dinámica junto a sus relaciones, por lo tanto, son similares a los diagramas de flujo (DFD), aunque tiene sutiles diferencias. El diagrama de actividades plantea un flujo de trabajo con un principio y un fin definidos, el cual puede desenvolverse como un flujo único, paralelo, concurrente o de acuerdo con la necesidad. Estos diagramas son utilizados para representar la lógica interna de los algoritmos que componen el sistema, especialmente los casos de uso.

Los formatos utilizados en la documentación del proyecto son una representación simplificada de la norma técnica colombiana, conservando sus principales características técnicas y enfocándose en la información de mayor relevancia. Los más importantes utilizados en este proyecto son el formato de levantamiento de requerimientos, casos de uso, clases y pruebas.

Formato de levantamiento de requerimientos. Este formato está conformado por tres secciones: el encabezado donde se encuentra el nombre del autor, la fecha de creación, el propósito, alcance, características del usuario, entorno operativo y requerimientos mínimos del sistema. Después del encabezado se encuentra el cuerpo, el cual alberga la lista de los requerimientos funcionales y no funcionales. Cada uno de los elementos tiene su respectivo código, descripción, prioridad y requerimientos asociados. Por último, la aprobación que contiene el control de cambios y la firma del director del proyecto.

Formato de casos de uso. Este formato está conformado por tres secciones: el encabezado donde se encuentra el nombre del autor y la fecha de creación. Después de este se encuentra el cuerpo, el cual alberga el nombre del caso de uso, código, versión, módulo, actor, objetivo, descripción de escenario, flujo, resultados esperados, precondiciones, excepciones, postcondiciones y el diagrama correspondiente a dicho caso. Por último, la aprobación que contiene el control de cambios y la firma del director del proyecto.

Formato de clases. Este formato esta conformado por tres secciones: el encabezado donde se encuentra el nombre del autor y la fecha de creación. Después se encuentra el cuerpo, el cual alberga el código asignado al diagrama, nombre del diagrama, descripción del escenario, las clases que conforman el diagrama con una casilla para cada tipo y el diagrama de clase respectivo. Por último, la sección de aprobación que contiene el control de cambios y la firma del director del proyecto.

Formato de pruebas. Este formato está conformado por tres secciones: el encabezado donde se encuentra el nombre del autor y la fecha de creación. Después de este se encuentra el cuerpo que alberga el objetivo de la prueba, las técnicas, el código involucrado, el caso de prueba (contiene un formato anidado) y observaciones. Por último la aprobación, la cual contiene el control

de cambios y la firma del director del proyecto.

Metodología de desarrollo

La metodología de acuerdo con Cambridge Dictionary (s.f.) se identifica como un conjunto de procedimientos sistemáticos, técnicas, enseñanza y estudio de un tema, que son utilizadas para el diseño, planificación y documentación de los sistemas de información. Si el tema principal es desarrollar software, la Ingeniería de Software toma el papel principal de aplicar una metodología, dividiendo el proyecto en secciones más pequeñas, planteando una serie de pasos o etapas, las actividades correspondientes del proyecto, y por último, las entradas y sus respectivas salidas (Sommerville y col., 2005). El objetivo de las metodologías es exponer un conjunto de técnicas para modelar sistemas, dado que estas permiten desarrollar un software de calidad.

Cada tipo de metodología tiene su propio enfoque, fortalezas y debilidades, haciéndolas más factibles o endebles en determinados casos. Las metodologías ágiles combinan una filosofía y unos lineamientos de desarrollo, buscan ejecutar una serie de pasos para obtener la satisfacción del cliente, buenos tiempos de entrega, sencillez al ser desarrollado el software y una comunicación constante y activa con el cliente final (Pressman, 2002). La correcta aplicación de una metodología genera un software que crece constante y rápidamente, y que además es exitoso, es decir, un software completamente operativo entregado en las fechas acordadas y que cumple con los requerimientos establecidos. Las metodologías ágiles se enfocan en diferentes aspectos del ciclo de vida del desarrollo de software y difieren una de otra por el enfoque que tienen. Algunas metodologías ágiles se enfocan en las buenas prácticas, por ejemplo, Programación Extrema (XP), mientras que otras se enfocan en la gestión de proyectos de software, como es el caso de Scrum, Kanban y Scrumban (Khan, 2014).

Scrumb. Esta metodología incorpora un conjunto de patrones del proceso que realzan las prioridades del proyecto, las unidades de trabajo agrupadas, la comunicación y la retroalimentación frecuente con el cliente. En este modelo una organización se divide en pequeños equipos auto-organizados con tamaños que van de cuatro a diez personas, donde se consta de un propietario de producto, el equipo de desarrollo, *testers* y un *Scrum Master*. Un equipo de Scrum además de

auto-organizado debe ser multifuncional y tener todas las competencias necesarias para realizar el proyecto sin la necesidad de intervención externa.

Kanban. Usando esta metodología el flujo de trabajo en el proyecto de desarrollo de software se visualiza usando un tablero llamado Kanban. El tablero Kanban tiene columnas que representan las etapas de trabajo, los hitos que se muestran como notas adhesivas y además en cada columna se puede especificar subcolumnas para tener un mayor orden de cada tarea. La metodología Kanban se basa en los principios de visualizar el flujo de trabajo del proceso, limitar el trabajo en curso y controlar el tiempo de entrega. Por tales motivos, es útil cuando se necesitan que los equipos conozcan el flujo de trabajo del proyecto, el estado actual de los hitos y que se realiza en el momento.

Scrumban. Es resultado de la combinación de los modelos de desarrollo Scrum y Kanban. Esta metodología intenta utilizar las mejores características de ambos modelos, usando la naturaleza descriptiva de Scrum para ser ágil y la mejora de procesos de Kanban para que el equipo mejore continuamente en el proceso. Una de las herramientas que adquiere principalmente de Kanban es visualizar el flujo de trabajo, debido a que el equipo puede visualizar cada una de las etapas, y por lo tanto, ayuda a conocer los encargados de las tareas y el estado actual del proyecto.

Programación Extrema. Esta metodología de desarrollo se centra en aplicar una clara comunicación y trabajo de equipo, que por lo general sucede entre dos personas. Es ligera e ideal para equipos pequeños y medianos que desarrollan software con requerimientos no muy bien definidos o que tienden a cambiar. Se distingue de otras metodologías por su confianza en la comunicación oral y como deben relacionarse las personas, interacciones sociales que según Kent Beck (2004) se consideran en XP tan importantes como las habilidades técnicas.

Aplicaciones en el mercado

Existe una gran variedad de aplicaciones y herramientas de Aprendizaje Máquina gratuitas y de pago en el mercado, de las cuales algunas están disponibles desde principio de siglo. Entre las más importantes que han sido guía para el desarrollo el presente proyecto son Orange, KNIME, WEKA, Google AI PlatformL, MLJar y Auger.ai.

- Orange (Demšar y col., 2004) es una librería originalmente de C++ que incluye algoritmos de Aprendizaje de Máquina y Minería de datos. Esta librería es una colección de módulos basados en Python que funcionan alrededor de la librería principal, lo que permite crear algoritmos que son más comprensibles en Python que en C++, siempre y cuando el tiempo de ejecución no sea un problema. Todas las funciones de Orange lo convierte en un *framework* basado en componentes, ideal para las labores relacionadas a la Inteligencia Artificial, que puede ser utilizado tanto por expertos, investigadores y estudiantes en un ambiente gráfico o en consola. A pesar de que principalmente es una librería, también se comporta como un grupo herramientas gráficas que utiliza los métodos disponibles con el fin de proveer mayor usabilidad para el usuario final.

- El desarrollo de KNIME (Berthold y col., 2007) inició en Enero del 2004 por un grupo de ingenieros de software en University of Konstanz. Este software es una aplicación de acceso libre desarrollada en Java, la cual le permite a los usuarios visualizar y crear flujos de datos, ejecutar pasos del proceso a voluntad y proporcionar inspección de resultados y modelos, por medio de las vistas y *widgets* que ofrece.

- El proyecto WEKA (Hall y col., 2009) se enfoca en proveer una colección de algoritmos de Aprendizaje de Máquina y Minería de datos, con el fin de ayudar a los investigadores y estudiantes por igual. Este software permite al usuario de realizar comparaciones rápidas entre diversos algoritmos y métodos con base a un conjunto de datos previamente suministrado. Por otra parte, el entorno de trabajo de esta aplicación incluye algoritmos de Clasificación, Regresión, Agrupamiento y selección de características, pero de igual forma es posible explorar y transformar la información previa al entrenamiento de un modelo de Aprendizaje de Máquina.

- AI Platform (Google, 2018) hace parte de los servicios ofrecidos de Google, pero enfocados en Inteligencia Artificial. Hay tres productos ofrecidos por este servicio, los cuales son AutoML Vision, AutoML Natural y AutoML Tables, los cuales varían en precios y operaciones permitidas de acuerdo con la suscripción que tenga el usuario, por ejemplo: el servicio de AutoML Tables tiene un valor de 19.32 dolares por hora más los costos de implementación de modelos,

predicción por lotes y predicción en línea.

- El enfoque de MLJar (Piotr y col., 2018) es permitir realizar el proceso de creación de modelos de Aprendizaje de Máquina con el mínimo esfuerzo posible. Este servicio puede ser accedido por medio de la API o utilizando los servicios web, los cuales si tienen un costo una vez agotados los créditos iniciales. Esta plataforma ofrece los servicios de Clasificación, Regresión, Agrupamiento y Redes Neuronales en un solo paquete, al igual que la extracción de características y obtención de datos relevantes.
- Auger ("Auger.AI", s.f.) es una aplicación web gratuita que ofrece principalmente los servicios de Clasificación y Regresión. Ofrece una búsqueda rápida y precisa basada en la optimización bayesiana patentada por Auger. Por último, esta plataforma elige el optimizador apropiado en función de las características del conjunto de datos, con la premisa que los optimizadores propietarios superan a otros abiertos.

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 en su primeros años de formación académica en carreras como Ingeniería de Software, Sistemas, Informática y Computación, la selección del modelo, hiperparámetros, 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 lograr el objetivo, la metodología de desarrollo ágil seleccionada para el proyecto es Scrumban, y además se tiene 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 cinco actividades relacionadas al levantamiento de requerimientos y definición del alcance. Estas actividades tienen un componente directamente relacionado con reuniones de equipo y validación. Las actividades son:

- 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: 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 mismos 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 siete actividades relacionadas a la planeación, el diseño, documentación y validación. Por otro lado, esta etapa es la de mayor duración a comparación de las demás, por lo que al menos un treinta por ciento del cronograma se distribuye para las actividades de diseño y planificación. Lo anteriormente mencionado se realiza a partir de las siguientes actividades:

- Actividad 6. Planteamiento del cronograma: 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, 9241-112:2019 e 9241-210:2019 para la creación de interfaces de usuario. Este diseño debe de ser validado constantemente durante su desarrollo hasta 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 para posterior implementación e 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 clase y teniendo en cuenta patrones de diseño para su creación.
- Actividad 11. Validación final del diseño: una vez terminadas 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 las demás.
- 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 cinco actividades, siendo estas en su mayoría la creación de código fuente, tanto para el *front-end* como para el *back-end*. Aun así, en esta etapa se realizan labores de documentación y revisiones de código, con el fin de producir software de calidad. Por último, esta etapa que tiene una duración menor al diseño y se conforma de las siguientes actividades:

- Actividad 13. Construcción de pruebas y código fuente: a partir del 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 los estándares PEP8, PEP20, PEP257, PEP3131, PEP 484 y PEP 526 para Python, el cual es el principal lenguaje que se utiliza en el proyecto.
- Actividad 14. Construcción de la interfaz de usuario: a partir de el *wireframe* y mockup previamente desarrollados se crea la interfaz en QT Designer.
- Actividad 15. Revisión del código desarrollado: por medio 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 16. Documentación del código fuente: después de realizarse 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 17. 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 cuatro 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. Esta penúltima etapa del proyecto está conformada por las siguientes actividades:

- Actividad 18. Ejecución pruebas de software: una vez integrado el sistema y se haya 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 19. 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 producidos deben ser el resultado de integrar el *front-end* y *back-end* en un solo entorno, por lo que la corrección debe estar centrada en mayor medida en ese componente.
- Actividad 20. 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 21. Documentación de las pruebas: de acuerdo 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 cuatro 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 correspondientes a la conclusión del proyecto de acuerdo con lo mencionado son:

- Actividad 22. 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 23. Validación final de funcionalidades: utilizando máquinas virtuales y equipos ajenos al 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 24. 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 los procedimientos paso a paso para el uso correcto de la aplicación.
- Actividad 25. 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.

Participantes del proyecto

Los integrantes del equipo (ver Cuadro 1) para el presente proyecto llamado *Voorspelling*, aplicación de escritorio para Aprendizaje de Máquina Supervisado, está conformado por dos desarrolladores y un director de proyecto.

Cuadro 1

Integrantes del proyecto

Cargo	Nivel de estudios	Nombres	Apellidos	Horas / semana
Desarrollador	Estudiante pregrado	Ivan David	Rey Rueda	24
Desarrollador	Estudiante pregrado	German Francisco	Diaz Figueredo	24
Director de proyecto	Doctorado	Nydia Paola	Rondón Villarreal	1

Cada uno de los desarrolladores son estudiantes de octavo y noveno semestre de Ingeniería de Software, con conocimientos en C#, Python, Patrones de Diseño y Aprendizaje de Máquina Supervisado. Mientras que por otro lado, el director de proyecto cuenta con Maestría en Ingeniería de Sistemas e Informática y Doctorado en Ingeniería electrónica.

Recursos disponibles

El presupuesto de acuerdo con el valor hora de cada miembro del equipo (ver Cuadro A1) a lo largo de ocho meses, da como resultado doce millones trescientos veinticinco mil ochocientos setenta y tres pesos Colombianos. Adicionalmente, el costo de los equipos de cómputo y periféricos, libros, licencias, servicios técnicos, papelería y otros (ver Cuadros A2, A3 A4, A5, A6 y A7) suman catorce millones setecientos setenta mil ochocientos treinta y cinco pesos Colombianos para un total (ver Cuadro 2) de veintisiete millones noventa y seis mil setecientos ocho pesos Colombianos.

Cuadro 2

Presupuesto total

Ítem	Rubro	UDES-Especie	Propias-Especie	Total
1	Personal	$ 12,325,873.92	$ -	$ 12,325,873.92
2	Equipos	$ -	$ 9,685,000.00	$ 9,685,000.00
3	Compra de libros	$ 1,557,029.75	$ -	$ 1,557,029.75
4	Licencia de software	$ 2,029,205.28	$ -	$ 2,029,205.28
5	Servicios técnicos	$ -	$ 720,000.00	$ 720,000.00
6	Papelería	$ -	$ 20,000.00	$ 20,000.00
7	Otros	$ -	$ 759,600.00	$ 759,600.00
Total		$ 15,912,108.95	$ 11,184,600.00	$ 27,096,708.95

Aproximadamente un cincuenta y nueve por ciento del presupuesto total hace parte de los recursos en especie de la Universidad de Santander, mientras que el restante cuarenta y uno por ciento es de los recursos en especie propios. No obstante, a pesar que gran parte de los recursos propios corresponde a la compra de equipos (ver Cuadro A2), estos ya se encuentran en posesión de los desarrolladores desde el inicio del proyecto.

Por último, el valor correspondiente a los servicios técnicos (ver Cuadro A5) está estipulado para los costos de internet y reparación del equipo de trabajo, mientras que los gastos relacionados a otros (ver Cuadro A7) son inversiones que uno de los miembros del equipo realizó en cursos web para Aprendizaje de Máquina.

Cronograma

La primera parte del cronograma (ver Cuadro 3) está planeada desde el inicio del proyecto en el mes de Agosto del 2020 hasta el final del mes de Noviembre del mismo año. Esta primera parte está compuesta de quince actividades distribuidas a lo largo del tiempo, de las cuales las últimas tres se extienden hasta la segunda parte del cronograma.

Cuadro 3

Cronograma de Agosto a Noviembre del 2020

Etapa	Actividades	Evidencia	Mes													
			Agosto		Septiembre				Octubre				Noviembre			
			Semana													
			3	4	1	2	3	4	1	2	3	4	1	2	3	4
1. Inicio de proyecto	Reunión inicial de requerimientos	Captura de pantalla														
	Planteamiento de la metodología	Issue en repositorio														
	Definición tecnologías	Issue en repositorio														
	Validacion Requeriemeintos	Reunión														
	Creación del reporte de requerimientos	Formato de requerimientos														
2. Diseño y planificación	Planteamiento del cronograma	Archivo en repositorio														
	Construcción del primer borrador	Archivo en repositorio														
	Diseño de interfaz preliminar	Archivo en repositorio														
	Validación de interfaz de usuario	Reunión														
	Diseño back-end	Archivo en repositorio														
	Validación diseño	Formato de casos de uso														
	Construcción del reporte de diseño	Reporte de diseño														
3. Código e implementación	Construcción de pruebas y código fuente	Archivo en repositorio														
	Construcción interfaz de usuario	Archivos .ui														
	Revisión del código desarrollado	Revision en Github														

Por otro lado, la segunda parte del cronograma (ver Cuadro 4) está planeada desde Diciembre del 2020 hasta Marzo del 2021, la cual es la fecha límite estipulada por la Universidad de Santander.

Cuadro 4

Cronograma de Diciembre del 2020 a Marzo del 2021

Etapa	Actividades	Evidencia	Mes															
			Diciembre				Enero				Febrero				Marzo			
			Semana															
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
3. Código e implementación	Construcción de pruebas y código fuente	Archivo en repositorio																
	Construcción interfaz de usuario	Archivos .ui																
	Revisión del código desarrollado	Revision en Github																
	Documentación del código fuente	Revision en Github																
	Integración front-end y back-end	Revision en Github																
4. Evaluación y pruebas	Ejecución pruebas de software																	
	Corrección de errores	Revision en Github																
	Validación de los cambios	Revision en Github																
	Documentación de las pruebas	Formato de pruebas																
5. Conclusión del proyecto	Compilación del ejecutable	Ejecutable																
	Validación final de funcionalidades	Formato de pruebas																
	Creación del manual de usuario	Manual de usuario																
	Despliegue de la aplicación	Recursos de la aplicación																

La segunda parte del cronograma está compuesta de trece actividades, aunque tres de estas pertenecen a la tercera etapa de la primera parte del cronograma. Por tal razón, el número total de actividades del presente proyecto son veinticinco; las cuales son las mismas planteadas en el diseño metodológico.

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

Nord, H. & Chambe-Eng, E. (1995). QT Designer. Consultado el 6 de octubre de 2020, desde https://www.qt.io/

Goodger, D. (2001, 29 de mayo). PEP 257: Docstring Conventions. Consultado el 30 de agosto de 2020, desde https://www.python.org/dev/peps/pep-0257/

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

Reitz, K. (2001, 5 de julio). PEP 8: The Style Guide for Python Code. Consultado el 30 de agosto de 2020, desde https://pep8.org/

Pressman, R. (2002). *Ingeniería del Software. Un enfoque práctico.* (7.ª ed.). McGraw-Hill.

Russell, S. & Norvig, P. (2003). *Artificial intelligence - a modern approach* (2.ª ed.).

Demšar, J., Zupan, B., Leban, G. & Curk, T. (2004). Orange: From experimental machine learning to interactive data mining. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3202*(August 2014), 537-539. https://doi.org/10.1007/978-3-540-30116-5_58

Kent Beck. (2004). *Extreme Programming Explained* (J. N. Mulcahy & K. Arney, Eds.; 2.ª ed.). John Wait.

Peters, T. (2004, 19 de agosto). PEP 3131: The Zen of Python. Consultado el 10 de septiembre de 2020, desde https://www.python.org/dev/peps/pep-0020/

Rumbaugh, J., Jacobson, I. & Booch, G. (2004). *The Unified Modeling Language Reference Manual* (2.ª ed.). Addison-Wesley Longman, Inc.

Finley, T. & Joachims, T. (2005). Supervised clustering with support vector machines. *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, 217-224. https://doi.org/10.1145/1102351.1102379

Sommerville, I., María, T., Galipienso, I. A., Botía, A., Francisco, M., Lizán, M., Pascua, J. & Jover, T. (2005). *Ingeniería del software* (M. Martín-Romo, Ed.; Séptima ed). PEARSON EDUCACIÓN, S.A.

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K. & Wiswedel, B. (2007). KNIME: The Konstanz Information Miner. *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*.

Sánchez-Maroño, N., Alonso-Betanzos, A. & Tombilla-Sanromán, M. (2007). Filter methods for feature selection - A comparative study. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4881 LNCS*, 178-187. https://doi.org/10.1007/978-3-540-77226-2_19

von Löwis, M. (2007, 1 de mayo). PEP 3131: Supporting Non-ASCII Identifiers. Consultado el 10 de septiembre de 2020, desde https://www.python.org/dev/peps/pep-3131/

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009). The WEKA data mining software. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10-18. https://doi.org/10.1145/1656274.1656278

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

Van Rossum, G. & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Brochu, E., Cora, V. M. & de Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, 1-49. http://arxiv.org/abs/1012.2599

ISO/IEC 11581-10. (2010). Consultado el 3 de octubre de 2020, desde https://www.iso.org/standard/46445.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830.

Han, J., Kamber, M. & Pei, J. (2012). *Data Mining: Concepts and Techniques*. Elsevier Inc. https://doi.org/10.1016/C2009-0-61819-5

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108-122.

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

Ethem Alpaydin. (2014). *Introduction to Machine Learning* (3.ª ed.). https://mitpress.mit.edu/books/introduction-machine-learning-third-edition

Khan, Z. (2014). Scrumban-Adaptive Agile Development Process: Using scrumban to improve software development process. (May), 122. http://urn.fi/URN:NBN:fi:amk-2014052710211

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

Shannon, M. (2014, 29 de septiembre). PEP 484: Type Hints. Consultado el 10 de septiembre de 2020, desde https://www.python.org/dev/peps/pep-0484/

Bost, R., Popa, R. A., Tu, S. & Goldwasser, S. (2015). Machine Learning Classification over Encrypted Data. (February), 1-14. https://doi.org/10.14722/ndss.2015.23241

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

Gonzalez, R., House, P., Levkivskyi, I., Roach, L. & van Rossum, G. (2016, 9 de agosto). PEP 526: Syntax for Variable Annotations. Consultado el 10 de septiembre de 2020, desde https://www.python.org/dev/peps/pep-0526/

Guerrero, J. A. (2016). El problema de la dimensionalidad. *Revista de Estadística y Sociedad*, *1*(68), 22-24. https://dialnet.unirioja.es/servlet/articulo?codigo=5711139

ISO 9241-112. (2017). Consultado el 10 de octubre de 2020, desde https://www.iso.org/standard/64840.html

Google. (2018). Google AI Platform. Consultado el 29 de octubre de 2020, desde https://cloud.google.com/ai-platform

Piotr, shahules786, spamz23, abtheo, uditswaroopa & partrit. (2018, 4 de noviembre). mljar-supervised. Consultado el 10 de octubre de 2020, desde https://github.com/mljar/mljar-supervised

Python 3.7 documentation. (2018). Consultado el 6 de octubre de 2020, desde https://docs.python.org/3.7/

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (N. Tache, Ed.; 2.ª ed.). O'Reilly Media, Inc.

ISO 9241-210. (2019). Consultado el 3 de octubre de 2020, desde https://www.iso.org/standard/77520.html

Sebastian Raschka. (2019). *Python Machine Learning* (3.ª ed.). Packt Publishing.

Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H. & Deng, S. H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of*

Electronic Science and Technology, *17*(1), 26-40. https://doi.org/10.11989/JEST.1674-862X.80904120

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M. & Hutter, F. (2020). Auto-Sklearn 2.0: The Next Generation, 1-18. http://arxiv.org/abs/2007.04074

Oswald Campesato. (2020). *Python 3 for Machine Learning*. Mercury Learning; Information.

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

QT Designer Manual. (2020). Consultado el 6 de octubre de 2020, desde https://doc.qt.io/qt-5/qtdesigner-manual.html

Auger.AI. (s.f.). Consultado el 29 de octubre de 2020, desde https://auger.ai

Cambridge Dictionary. (s.f.). Meaning of methodology. Consultado el 19 de octubre de 2020, desde https://dictionary.cambridge.org/us/dictionary/english/methodology

Apéndice

Recursos disponibles

Cuadro A1

Valor hora del personal

Ítem	Cargo	Valor Hora
1	Director Proyecto	$ 117,573.00
2	Estudiante Pregrado	$ 5,575.00
3	Auxiliar de Investigación	$ 5,575.00

Cuadro A2

Costo de equipos

Ítem	Descripción	Justificación	Costo Comercial
1	HP Pavillon laptop	Estacion de Trabajo	$ 4,000,000.00
2	Asus gl552vw	Estacion de Trabajo	$ 4,000,000.00
3	Asus vp249qgr ips 144hz	Equipo de trabajo	$ 1,200,000.00
4	Mouse Logitech g203	Perifericos de trabajo	$ 120,000.00
5	Redragon G711	Perifericos de trabajo	$ 110,000.00
6	Diadema Gamer V5000	Perifericos de trabajo	$ 75,000.00
7	RUNMUS - Auriculares	Perifericos de trabajo	$ 180,000.00
Total			$ 9,685,000.00

Cuadro A3

Costo de libros

Ítem	Descripción	Costo Comercial
1	Introduction to Machine Learning (3rd Edition)	$ 236,405.00
2	Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)	$ 169,411.00
3	Python Machine Learning (3rd edition)	$ 154,000.00
4	Filter methods forfeature selection - A comparative study	$ 115,507.00
5	Ingeniería de Software (7th Edition)	$ 157,860.00
6	The Unified Modeling Language Reference Manual (2nd Edition)	$ 269,518.00
7	Artificial Intelligence: A Modern Approach (2nd Edition)	$ 454,329.00
Total		$ 1,557,030.00

Cuadro A4

Costo de licencias

Ítem	Descripción	Justificación	Costo Comercial
1	PyCharm	2 licencias, 8 meses, entorno de desarrollo integrado	$ -
2	Microsoft office 365	2 licencias, 8 meses, conjunto de programas informáticos	$ 960,000.00
3	LucidChart	2 licencias, 8 meses	$ 293,476.00
4	Figma	1 licencias, 8 meses	$ 355,729.00
5	QT Designer	1 licencia, 8 meses	$ -
6	Windows 10 professional	2 licencias, 8 meses	$ 420,000.00
Total			$ 2,029,205.00

Cuadro A5

Costo de servicios técnicos

Ítem	Descripción	Justificación	Costo Comercial
1	Limpieza de pc	Altas temperaturas en ejecución de algoritmos	$80,000.00
2	Servicio de Internet	8 meses de internet	$640,000.00
Total			$720,000.00

Cuadro A6

Costo de papelería

Ítem	Descripción	Justificación	Costo Comercial
1	Papelería	Impresión, fotocopias y escaneos	$20,000.00
Total			$20,000.00

Cuadro A7

Costo de elementos adicionales

Ítem	Descripción	Costo Comercial
1	Curso de introducción Machine Learning	$189,900.00
2	Curso de aplicación Machine Learning con Python	$189,900.00
3	Curso Python	$189,900.00
4	Curso Ingeniería de datos	$189,900.00
Total		$759,600.00