



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFM del Máster Universitario en
Ingeniería Informática**

**Modelo predictivo de
resultados de carreras de
Fórmula 1**



Presentado por Francisco Martín Vargas
en Universidad de Burgos — 5 de julio de 2023
Tutor: Daniel Urda Muñoz



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Daniel Urda Muñoz, profesor del departamento de Digitalización, área de Ciencia de la Computación e Inteligencia Artificial.

Expone:

Que el alumno D. Francisco Martín Vargas, con DNI 71704736M, ha realizado el Trabajo final de Máster Universitario en Ingeniería Informática titulado Modelo predictivo de resultados de carreras de Fórmula 1.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 5 de julio de 2023

Vº. Bº. del Tutor:

D. Daniel Urda Muñoz

Resumen

La *Inteligencia Artificial* o *IA* ha cambiado significativamente nuestras vidas y muchas facetas de nuestra sociedad. Gracias a la eficacia y la precisión que demuestran, han conseguido mejorar la comodidad de nuestras actividades diarias mediante la automatización de tareas repetitivas. Además, se han creado nuevas posibilidades en la industria del transporte, la salud, la educación, e incluso en la competición del motor, impulsando su desarrollo y elevando su nivel. Su gran capacidad de analizar y procesar información en aspectos sobre el rendimiento del vehículo, o el comportamiento del conductor y las estrategias de carrera han proporcionado a los equipos de competición gran cantidad de diferentes perspectivas para abordar la toma de decisiones de los equipos.

Un modelo de predicción de resultados resume a la perfección la fusión de la inteligencia artificial y la competición automovilística, poniendo al alcance de todo el mundo las tecnologías que se usan en lo más alto del mundo del motor. Con ese propósito se desarrollará un modelo para pronosticar los resultados de las carreras.

El presente documento muestra el empleo de técnicas de *aprendizaje automático* para realizar predicciones sobre los resultados en la competición automovilística *Fórmula 1*. Se pretender llevar a cabo este proyecto a través del lenguaje de programación *Python*, así como proporcionar una aplicación desde la cual podremos gestionar dichas predicciones. Para llevarlo a cabo, se han utilizado el entorno de desarrollo *Visual Studio Code*.

El resultado de este modelo de predicción se encuentra en el siguiente repositorio de GitHub: [F1RacePredictor](#).

Descriptores

Inteligencia Artificial, Modelo predictivo, Aprendizaje automático, Fórmula 1, Python.

Abstract

Artificial Intelligence (AI) has significantly changed our lives and many facets of our society. Thanks to the efficiency and precision they demonstrate, they have managed to improve the convenience of our daily activities by automating repetitive tasks. In addition, new possibilities have been created in the transport industry, healthcare, education, and even in motor racing, driving their development and raising their level. Its strong ability to analyse and process information on aspects of vehicle performance, or driver behaviour and race strategies has provided racing teams with a wealth of different perspectives to approach team decision making.

A performance prediction model perfectly encapsulates the fusion of artificial intelligence and motor racing, making the technologies used at the top of the motor racing world available to everyone. To this end, a model for predicting race results will be developed.

This document shows the use of machine learning techniques to make predictions about results in Formula 1 motor racing. The aim is to carry out this project using the Python programming language, as well as to provide an application from which we will be able to manage these predictions. To carry it out, the development environment *Visual Studio Code* has been used.

The result of this prediction model can be found in the following GitHub repository: [F1RacePredictor](#).

Keywords

Artificial Intelligence, Predictive Modeling, Machine Learning, Formula 1, Python.

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
Introducción	1
1.1. Estructura de la memoria	2
1.2. Estructura de los anexos	2
Objetivos del proyecto	5
2.1. Objetivos generales	5
2.2. Objetivos técnicos	5
2.3. Objetivos personales	6
Conceptos teóricos	7
3.1. Fórmula 1	7
3.2. IA - Inteligencia Artificial	7
3.3. Modelo predictivo	8
3.4. Aprendizaje automático	8
3.5. Python	11
Técnicas y herramientas	13
4.1. Herramientas de control de versiones	13
4.2. Herramientas de gestión de proyectos	14
4.3. Metodologías	14
4.4. Patrones de diseño	15

4.5. Herramientas de documentación	15
4.6. Entornos de desarrollo integrado (IDE)	15
4.7. Lenguajes de programación	16
4.8. Librerías	16
Aspectos relevantes del desarrollo del proyecto	19
5.1. Elección del tema	19
5.2. Comienzo del proyecto	20
5.3. Recolección de datos	20
5.4. Preparación y limpieza de datos	22
5.5. Selección de características y codificación de los datos	28
5.6. Entrenamiento	32
5.7. Optimización del modelo	40
5.8. Aplicación para interacción	45
Trabajos relacionados	55
6.1. A machine learning approach to predict the winner of the next F1 Grand Prix - Veronica Nigro	55
6.2. Formula 1 Race Prediction	55
6.3. Ventajas y debilidades respecto a la competencia	56
Conclusiones y Líneas de trabajo futuras	59
7.1. Conclusiones	59
7.2. Líneas de trabajo futuras	60
Bibliografía	61

Índice de figuras

3.1. lenguajes de programación mas usados desde 2010 a 2022 [4]. Basado en el porcentaje de preguntas en <i>Stack Overflow</i> , uno de los portales de preguntas y respuestas sobre programación más importantes de todo el mundo.	12
5.1. Extracto de los cambios de nombre en la historia de algunas de las escuderías[12].	24
5.2. Promedio de errores de los pilotos según la edad	25
5.3. Porcentaje de victorias cuando el piloto sale en la primera posición.	26
5.4. Porcentaje de victorias cuando el piloto sale en las tres primeras posiciones.	27
5.5. Fiabilidad de los coches entre 2015 y 2018	31
5.6. Eficiencia de los modelos para predecir la posición final del carrera.	35
5.7. Tiempos de ejecución de los algoritmos para predecir la posición final del carrera.	36
5.8. Eficiencia de los modelos para predecir el ganador de la carrera.	37
5.9. Tiempos de ejecución de los algoritmos para predecir el ganador de la carrera.	38
5.10. Eficiencia de los modelos para predecir el ganador de la <i>pole</i>	39
5.11. Tiempos de ejecución de los algoritmos para predecir el ganador de la <i>pole</i>	40
5.12. Valores anteriores y valores conseguidos con el ajuste de los hiperparámetros.	45
5.13. Primer boceto de la interfaz de usuario.	47
5.14. Boceto de la interfaz de usuario con desplegables horizontales y con imagen de fondo.	47
5.15. Boceto de la interfaz de usuario realizado con la librería <i>wxPython</i>	48
5.16. Diseño de la interfaz de usuario de la aplicación con WX.	48

5.17. Diseño de la pantalla dónde escogemos el circuito.	49
5.18. Diseño de la pantalla dónde escogemos el clima.	49
5.19. Diseño de la pantalla dónde hacemos predicciones de el resultado total.	50
5.20. Diseño de la pantalla dónde hacemos predicciones del ganador. .	50
5.21. Diseño de la pantalla dónde escogemos las posiciones de salida del piloto.	51
5.22. Diseño de la pantalla dónde escogemos las posiciones disponibles de salida del piloto.	51
5.23. Diseño de la pantalla de inicio de la aplicación.	52
5.24. Predicción del resultado total.	53
6.1. Interfaz de Formula 1 Race Prediction.	56

Índice de tablas

5.1.	Estados de finalización de un piloto en una carrera.	23
5.2.	Estados de finalización simplificados.	23
5.3.	Puntuación de cada piloto en función de la posición final de carrera.	26
6.1.	Comparativa de características de los proyectos.	56

Introducción

Al igual que muchos otros aspectos de nuestras vidas, el mundo de la Fórmula 1 también se ha visto significativamente alterado por la inteligencia artificial (IA). Las IA se han abierto camino para alcanzar su máximo potencial en este emocionante deporte, donde la innovación, la velocidad y la estrategia son cruciales.

La Fórmula 1 es un evento automovilístico de primer nivel que combina ingeniería de vanguardia, capacidad de conducción extrema y tácticas de equipo sofisticadas. Cada milisegundo, cada elección y cada pequeño detalle importan en este entorno intensamente competitivo. Una décima de segundo puede decidir una victoria o una derrota.

El uso de la inteligencia artificial en la Fórmula 1 ha creado nuevas posibilidades para mejorar la eficiencia de los equipos y los pilotos. La IA puede revelar patrones ocultos y ofrecer información importante en la toma de decisiones estratégicas. Todo esto gracias al análisis de grandes conjuntos de datos obtenidos durante las carreras, como pueden ser los datos de telemetría o las condiciones climáticas.

Uno de los aspectos más emocionantes de la inteligencia artificial en la Fórmula 1 es la predicción del rendimiento y los resultados. Los modelos de aprendizaje automático pueden observar datos de todas las carreras, evaluar el rendimiento del piloto y del equipo, y señalar los factores que tienen el mayor impacto en el resultado de una carrera. Gracias a ello los equipos toman decisiones inteligentes con el objetivo de maximizar su rendimiento durante el gran premio, desde los ajustes en la configuración del coche hasta las estrategias de parada en boxes.

Las IA también han contribuido significativamente al diseño y la simulación de los coches de Fórmula 1. Para encontrar la configuración que

maximiza la velocidad con la estabilidad del coche, los algoritmos de optimización pueden examinar gran cantidad de combinaciones diferentes de ajustes mecánicos y configuraciones en la aerodinámica.

Finalmente, la inteligencia artificial ha logrado revolucionar la Fórmula 1, brindando importantes herramientas para mejorar su desempeño y la capacidad en la toma de decisiones. Ha cambiado permanentemente los deportes de élite, desde la predicción de resultados hasta la optimización del diseño de automóviles. Se deben tener en cuenta los potenciales usos de la inteligencia artificial y los modelos de aprendizaje automático en la Fórmula 1 y cómo se pueden aplicar a la vida diaria.

1.1. Estructura de la memoria

La memoria está organizada de la siguiente manera:

- **Introducción:** Describe el proyecto y da una visión general. También se puede acceder a la organización de la memoria, sus anexos y sus materiales adjuntos.
- **Objetivos del proyecto:** objetivos que se tratará de cumplir al terminar el proyecto.
- **Conceptos teóricos:** breve explicación de las principales ideas teóricas necesarias para comprender el desarrollo del proyecto.
- **Técnicas y herramientas:** colección de herramientas y técnicas metodológicas que se utilizaron para desarrollar el proyecto.
- **Aspectos relevantes del desarrollo del proyecto:** presentación de los aspectos más importantes del proyecto.
- **Trabajos relacionados:** presentación y comparación con algunos trabajos relacionados.
- **Conclusiones y Líneas de trabajo futuras:** conclusiones extraídas de la finalización del proyecto, así como sugerencias para posibles avances en el desarrollo del proyecto en el futuro.

1.2. Estructura de los anexos

Los anexos se estructuran de la siguiente manera:

- **Plan del proyecto Software:** desarrollo de un estudio de viabilidad del proyecto y de la planificación temporal.
- **Especificación de Requisitos:** requisitos resultantes de los objetivos del proyecto.
- **Especificación de diseño:** descripción de la arquitectura del sistema y los diagramas resultantes.
- **Documentación técnica de programación:** descripción de las herramientas (entornos de desarrollo, lenguajes, etc.) necesarias para trabajar en el proyecto.
- **Documentación de usuario:** guía en la que se expone como se debe usar el producto final, con vistas al usuario final.

Enlaces de los materiales del proyecto

- [Repositorio del proyecto](#).
- [Vídeo demostración](#) (enlace a YouTube).

Objetivos del proyecto

El objetivo principal de este proyecto consiste en desarrollar un modelo predictivo para carreras de Fórmula 1. Por otro lado, se van a comparar dos sistemas de selección de características, además de ajustar un modelo en el campo de la inteligencia artificial con el fin de identificar el más adecuado y optimizar al máximo su desempeño. Además, se pretende crear una aplicación que permita la interacción con el modelo final, ofreciendo una herramienta sencilla y accesible para los usuarios finales.

2.1. Objetivos generales

- Desarrollar un modelo para la predicción de resultados en las carreras de Fórmula 1.
- Comparar una selección mediante algoritmos contra una selección manual.
- Ajustar un modelo para mejorar sus predicciones.
- Desarrollar una aplicación para la interacción con el modelo final.

2.2. Objetivos técnicos

- Desarrollar un modelo de predicción con el lenguaje Python que gestione toda la complejidad.
- Desarrollar una aplicación en el entorno Python para la predicción de resultados del modelo.
- Utilizar Git como sistema de control de versiones distribuido junto con la plataforma GitHub y ZenHub para una gestión más ágil.

2.3. Objetivos personales

- Demostrar la utilidad de los conocimientos adquiridos en la universidad.
- Adquirir las habilidades necesarias para utilizar enfoques y herramientas de vanguardia en el lugar de trabajo.
- Perfeccionar mis habilidades de desarrollo de aplicaciones.
- Ampliar los conceptos de Python que aprendí durante mis estudios universitarios.

Conceptos teóricos

Para comprender el marco teórico del desarrollo de este proyecto, es fundamental contar con un conocimiento previo de los conceptos en los que se basa.

3.1. Fórmula 1

La **Fórmula 1** es una competición automovilística que dio inicio en 1950. En la actualidad es considerada la más prestigiosa del mundo y también es la serie deportiva anual con mayor popularidad. Este campeonato está regulado por la la FIA (Federación Internacional del Automóvil) y actualmente está compuesta por 10 equipos con dos coches cada uno. Dicho certamen se lleva a cabo desde marzo hasta noviembre, con un total de 23 carreras disputadas en 20 países distribuidos en cuatro continentes [1].

3.2. IA - Inteligencia Artificial

La Inteligencia Artificial o IA se podría definir como la capacidad de las máquinas para tomar decisiones tal como lo hace el ser humano. Para ello se vale de diferentes algoritmos y en el aprendizaje sobre una gran cantidad de datos. Algunos sostienen que la inteligencia radica en las propiedades de los procesos internos de pensamiento y razonamiento, mientras que otros la definen en términos de comportamiento inteligente, una caracterización externa [16].

3.3. Modelo predictivo

Un modelo predictivo es un modelo de datos, basado en estadísticas inferenciales, que se utiliza para predecir resultados. Consiste en un conjunto de herramientas y técnicas estadísticas que sirven para pronosticar y predecir el comportamiento ante un evento.

Aplicado al campo de la inteligencia artificial, podríamos decir que un modelo de predicción no es más que un modelo de caja negra para hacer predicciones [5], lo que significa que no se requiere un conocimiento detallado de su funcionamiento interno para utilizarlo. Los modelos predictivos utilizan técnicas basadas en el aprendizaje automático para hacer predicciones basadas en datos históricos o patrones identificados en los datos de entrenamiento. Dichos modelos aprenden automáticamente de los datos de entrenamiento y generan predicciones basadas en su capacidad para identificar patrones ocultos en los datos de entrada.

3.4. Aprendizaje automático

El aprendizaje automático o machine learning es un subcampo de la inteligencia artificial que, a su vez es una técnica que mejora el rendimiento de un sistema mediante el uso de la computación a través de la experiencia. Los datos son la forma principal de experiencia en los sistemas informáticos, y el objetivo principal del aprendizaje automático es crear algoritmos de aprendizaje que crean modelos a partir de datos. Al proporcionar al algoritmo de aprendizaje datos de experiencia, podemos crear un modelo que puede pronosticar los resultados de futuras observaciones. Si consideramos la informática como el tema de los algoritmos, entonces el aprendizaje automático es el tema de los algoritmos de aprendizaje [18].

En este proyecto vamos a emplear algoritmos de aprendizaje automático diferenciados en dos grupos: algoritmos de regresión y de clasificación.

Algoritmos de regresión:

- Linear Regression: este modelo tiene como objetivo crear una relación lineal entre la variable objetivo y las demás variables. Se aplica a problemas de regresión donde el objetivo es predecir valores numéricos. Además este modelo es sumamente eficiente en cuanto a computación.
- Decision Tree Regressor: es un modelo que se basa en árboles de decisión. Crea una estructura de árbol tratando de que las ramas

representen posibles respuestas a las preguntas que forman los nodos con respecto a las características. Cada hoja del árbol recibe un valor de salida tras la división recursiva en subconjuntos del conjunto inicial de datos. Se aplica en problemas que requieran identificar relaciones no lineales.

- Random Forest Regressor: el propósito del modelo es hacer predicciones a través de la combinación de varios árboles de decisión. El resultado final se obtienen promediando las predicciones de cada uno de los árboles, los cuales se han entrenado con una muestra aleatoria diferente a las demás. Además, cuenta con la ventaja de reducir el sobreajuste y aumentar la eficiencia. Es aplicado a problemas con características de alta dimensión y puede gestionar un gran volumen de datos.
- Support Vector Regression: modelo que predice utilizando vectores de soporte. Los puntos de los vectores de soporte se encuentran en el borde del límite de decisión o en la zona que afectará en mayor medida cómo se define la función de regresión. El modelo controla una tolerancia de error, buscando una función de regresión que se ajuste mejor a los datos y reduzca la discrepancia entre la predicción y el valor real. Cuando nuestros datos contienen características no lineales o incluso con valores atípicos, este modelo es sumamente útil en problemas de regresión.

Algoritmos de clasificación:

- Logistic Regression: es utilizado para resolver problemas donde las variables objetivo son binarias o representan categorías. Se usa una función logística para calcular la probabilidad con la que una instancia dada pertenezca a una clase en particular. Se sustenta en la relación lineal encontrada entre características y variable objetivo. Además, convierte el resultado en una probabilidad gracias a una función de activación. Utilizado sobre todo para problemas de clasificación binaria, pero se puede expandir a problemas con variables objetivo categóricas.
- Decision Tree Classifier: funciona igual que el Decision Tree Regressor pero con la particularidad de que en este cada hoja del árbol recibe una etiqueta de clase tras la división recursiva del conjunto de datos y no un valor.

- Random Forest Classifier: al igual que en la regresión, realiza las predicciones a través de la combinación de varios árboles de decisión. El resultado se obtiene promediando la clasificación total de los árboles.
- Support Vector Classification: los vectores utilizados en este modelo se usan para dividir instancias en clases. La diferencia con el SVR es que ahora buscamos el mejor hiper plano en el cual haya la mayor distancia de mayor entre clases. Permite el uso de datos tanto linealmente separables como no separables gracias a funciones kernel. Gran elección en problemas de clasificación de variables categóricas o binarias.
- KNeighbors Classifier: usa el algoritmo K-Nearest Neighbors. Asigna etiquetas de los k vecinos cercanos a las instancias. Se vale de la noción de que las instancias parecidas pertenecen a la misma clase con frecuencia. Es muy eficiente y sencillo para problemas de clasificación siempre y cuando la elección del número de k vecinos no sea demasiado alta.
- Gaussian NB (Naive Bayes): este modelo se basa en el teorema de Bayes, además de asumir que los datos forman una distribución gaussiana. Utiliza la regla de Bayes para determinar la probabilidad de pertenecer a una clase en particular. Se asume la independencia entre características, algo ingenuo, de ahí su nombre. Para problemas de clasificación con variables continuas y distribuidas en forma normal es muy eficiente.

Además, estos algoritmos pueden usarse en combinación con diferentes estrategias para realizar una selección de características, como se ha utilizado para este desarrollo. Entre ellas encontramos la estrategia de eliminación hacia atrás que comienza con todas las características y elimina una a una en cada iteración. Esto es, en cada iteración de esta técnica se entrena y evalúa un modelo con las variables disponibles. La característica menos significativa es la eliminada, tomando como criterio el peso de la misma en esa iteración. Esto se repite hasta llegar al criterio de parada. Pero para ello debemos usar técnicas como el algoritmo RFE o *Recursive Feature Elimination*, que es un tipo de eliminación hacia atrás, pero tiene la ventaja de ser compatible con varios algoritmos de aprendizaje automático, lo que lo convierte en una herramienta flexible y adaptable. Se puede combinar con algoritmos de regresión, clasificación u otras estrategias de modelado, brindándonos la flexibilidad de manejar varios problemas y conjuntos de datos. Además, proporciona una medida de importancia o relevancia para

cada característica en función de cómo contribuye al modelo final. Dado que nos permite identificar los rasgos que tienen mayor influencia en la toma de decisiones del modelo, será particularmente útil para nuestro propósito.

3.5. Python

Python [7] es un lenguaje de programación orientado a objetos, interpretado e interactivo que posee características como la inclusión de módulos, excepciones, tipado dinámico, tipos de datos de alto nivel y clases. Debido a su gran cantidad de librerías y programadores, es muy popular y ampliamente utilizado en diferentes campos de la programación, lo que permite realizar diversos propósitos. Además, se le considera un lenguaje de alto nivel que ofrece una sintaxis clara y sencilla de comprender.

Fue creado a principios de la década de 1990 por Guido van Rossum en Stichting Mathematisch Centrum (CWI) en los Países Bajos como sucesor de un idioma llamado ABC [6].

Además, este lenguaje de programación es muy popular y ampliamente utilizado en el campo de la inteligencia artificial, ofreciendo una serie de librerías especializadas como pueden ser TensorFlow, Keras, PyTorch, y Scikit-learn, que facilitan la implementación de algoritmos de aprendizaje automático y procesamiento de datos.

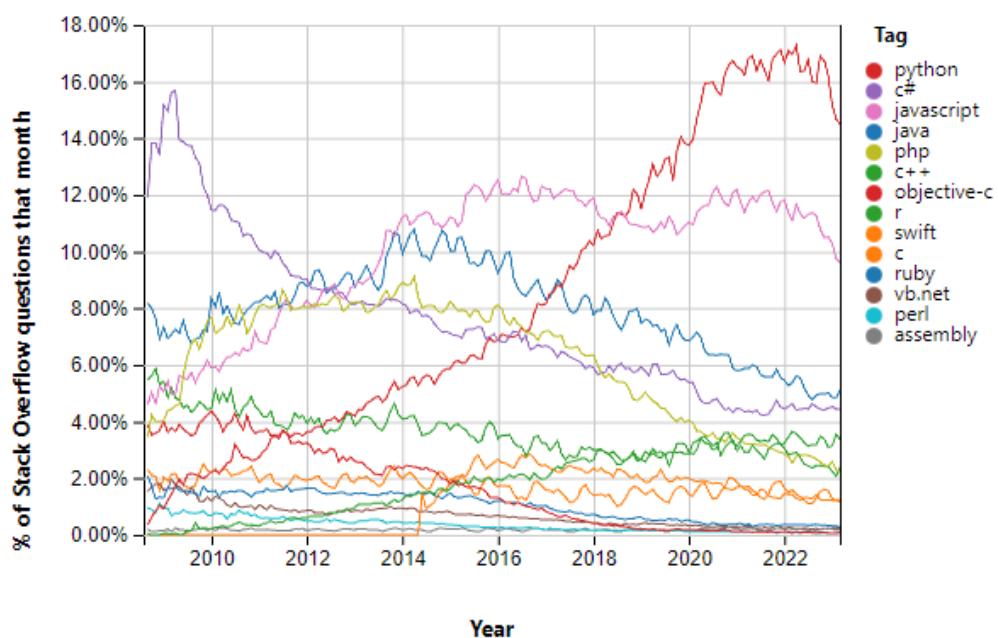


Figura 3.1: lenguajes de programación mas usados desde 2010 a 2022 [4]. Basado en el porcentaje de preguntas en *Stack Overflow*, uno de los portales de preguntas y respuestas sobre programación más importantes de todo el mundo.

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

4.1. Herramientas de control de versiones

Github

GitHub es una plataforma que nos permite gestionar y organizar nuestros proyectos y está basada en la nube, incorporando las funciones de control de versiones que proporciona Git. Esta herramienta posibilita a los desarrolladores poder almacenar y administrar su código, realizar un registro y control de los cambios sobre el código almacenado. Es una de las herramientas más populares entre los desarrolladores, cuenta con más de 100 millones de repositorios, y la mayoría de ellos son de código abierto.

He utilizado GitHub para el alojamiento de mi proyecto en el repositorio [F1RacePredictor](#).

Github aplicación escritorio

Esta aplicación nos permite interactuar con repositorios de GitHub a través de una interfaz gráfica muy intuitiva que sustituye el uso de Git mediante comandos. Nos facilita la tarea de gestionar nuestro repositorio, la sincronización de los cambios, la revisión y/o creación de solicitudes de extracción de datos, además de la colaboración con otros usuarios de nuestros o sus proyectos.

Git LFS

Git LFS es una herramienta que nos proporciona GitHub para poder alojar en nuestro repositorio archivos de más de 100 MB (como es el caso de uno de los modelos entrenados), estos archivos se guardarán en un repositorio de GitHub LFS y en nuestro repositorio habrá un archivo con un enlace al fichero original.

4.2. Herramientas de gestión de proyectos

ZenHub

ZenHub es una plataforma para la gestión ágil de proyectos que se integra con github, funcionando como aplicación nativa en su interfaz. Te ayuda a planificar tu proyecto dentro de GitHub, automatiza el flujo de trabajo. Más del 75 % de los desarrolladores que usan ZenHub en sus proyectos dicen que ZenHub mejora su enfoque y ayuda en el envío de un mejor software en un menor periodo de tiempo, y el 65 % informan de proyectos con un mejor alcance.

ZenHub me ha permitido gestionar mi proyecto, ayudándome con la planificación del mismo, para cumplir los tiempos de entrega del mismo.

4.3. Metodologías

Sprints

Es una metodología ágil, cuya base es la división del trabajo de breves ciclos, los sprints. Se deben llevar a cabo tareas tanto de planificación, como de desarrollo o revisión y retrospectivas en cada uno de los sprints. Los sprints tiene un período de tiempo establecido. Gracias a esta metodología fomentaremos la colaboración, la entrega de resultados de forma incremental, y resulta en una rápida adaptación a los cambios.

4.4. Patrones de diseño

Patrón Modelo-Vista-Controlador o MVC

MVC es un patrón de diseño arquitectónico que diferencia el sistema en tres capas, los datos, la lógica y la interfaz de usuario. En este patrón el modelo está representado en la capa de datos, la vista referencia a la interfaz de usuario y el controlador implica la lógica de la aplicación.

4.5. Herramientas de documentación

L^AT_EX

L^AT_EX es un sistema de software libre de composición tipográfica de alta calidad, orientado a la producción de documentación técnica y científica. L^AT_EX es el estándar de facto para la comunicación y publicación de documentos científicos gracias a sus características, posibilidades y calidad profesional.

He usado L^AT_EX para el desarrollo de tanto este documento como de los anexos.

Overleaf

Es una web que permite a los usuarios colaborar en la creación y edición de documentos LaTeX. Tiene la gran ventaja de poder usarse sin la necesidad de instalar software adicional, gracias a su entorno de desarrollo integrado (IDE), simplificando la escritura y compilación de documentos LaTeX.

Draw.io

Draw.io es una herramienta de diagramación que nos permite realizar cualquier tipo de diagramas (diagramas de flujo, de secuencia, de red, UML, etc). Es muy interesante ya que proporciona elementos UML, muy necesarios en proyectos software.

4.6. Entornos de desarrollo integrado (IDE)

Visual Studio Code

Usaremos esta interfaz de desarrollo por sus grandes ventajas como puede ser su integración nativa con git, el soporte multiplataforma y multilenguaje,

la alta personalización con gran cantidad de extensiones, además de ser gratuito y de código abierto.

4.7. Lenguajes de programación

Python

Se ha escogido python como lenguaje en el desarrollo del modelo porque es uno de los lenguajes más utilizados en el campo de la Inteligencia Artificial. Además de tener una sintaxis clara y simple, existe una amplia comunidad de desarrolladores, y bibliotecas especializadas en IA. Asimismo, es muy flexible y fácil de usar, lo que permite una fácil integración con otros lenguajes.

4.8. Librerías

Pandas

Esta librería está enfocada en el análisis de datos en python. Gracias a sus estructuras podemos manipular los datos de forma fácil y eficiente. Sus estructuras más conocidas son los DataFrames y las Series, con las cuales podemos trabajar con datos tanto tabulares como series temporales. Es una de las librerías más usadas gracias a su gran cantidad de funcionalidad en procesamiento y análisis de datos. Es perfecta para el tratamiento, preparación y limpieza de datos en modelado.

beautifulsoup4

Biblioteca de python diseñada para la extracción de datos de documentos del tipo HTML o XML. Nos hace posible navegar entre los datos de manera muy eficiente. Es utilizada sobre todo para webscrapping, recopilación de información de sitios web.

Requests

Requests es un librería de python que tiene el fin de proporcionarnos una manera muy simple de hacer peticiones HTTP. Además, permite el manejo de autenticación, cookies y redireccionamiento automáticamente. Sobre todo usada para la interacción con APIs web.

Scikit-learn

Una de las bibliotecas más usadas en python para el aprendizaje automático es Scikit-learn, ya que nos ofrece una gran cantidad de algoritmos y herramientas para el procesado de datos. Cuenta con soluciones de clasificación, regresión, reducción de dimensionalidad, etc. Además tiene una curva de aprendizaje muy corta para aprendizaje automático.

WXpython

Es un librería que proporciona una interfaz gráfica de usuario (GUI) para el desarrollo de aplicaciones multiplataforma. Ofrece una gran selección de herramientas y widgets para crear interfaces interactivas con posibilidad de modificación.

PyQt5

Es un librería que proporciona enlaces de python para Qt v5. Qt es una colección de herramientas de interfaz de usuario multiplataforma y es altamente personalizable. Gracias a esta librería aprovechamos la versatilidad y la capacidad de Qt para desarrollar aplicaciones robustas en el entorno python.

Matplotlib

Dentro de python esta librería nos ofrece una gran cantidad de opciones en el trazado de gráficos de alta calidad. Además es una de las librerías más populares en el ámbito de las ciencias para la visualización de datos.

Aspectos relevantes del desarrollo del proyecto

Esta sección tiene como objetivo recopilar las facetas más fascinantes de la evolución del proyecto, desde la exposición del ciclo de vida empleado hasta los detalles más cruciales de las fases de análisis, diseño e implementación.

5.1. Elección del tema

Desde mi infancia he tenido la suerte de poder interactuar con el apasionante mundo del deporte y más en concreto de la Fórmula 1. Puedo recordar claramente haber visto carreras en la televisión con mi familia y maravillarme con la velocidad, la habilidad y la emoción que caracterizan a este deporte de élite. También, esa época coincidió con el éxito de nuestro representante en este deporte, Fernando Alonso. Viví sus dos mejores años, 2005 y 2006. Jamás olvidaré sus luchas con Michael Schumacher, considerado uno de los mejores pilotos de la historia sino el mejor.

Con el tiempo mi interés por la Fórmula 1 se hizo más fuerte. Comencé por investigar a los pilotos más laureados y a los equipos míticos que han dejado una larga huella en la historia de este deporte como pueden ser Ferrari o McLaren. Más adelante continué por interesarme en la evolución de los monoplazas como resultado de los avances tecnológicos y viendo como cada vez se volvían más rápidos, precisos y efectivos.

Es por todo esto que decidí realizar mi Trabajo de Fin de Máster sobre este increíble deporte que es la Fórmula 1. Además, en él es extremadamente importante el uso de modelos predictivos, tanto para predecir el ritmo del

coche y el de sus rivales, como para controlar el desgaste de los neumáticos o la potencial pérdida de tiempo en boxes durante una carrera.

5.2. Comienzo del proyecto

Una vez escogido el tema, tocaba planificar el desarrollo del proyecto. Para ello decidí llevar a cabo una planificación por sprints. Comenzando por la compresión del problema y la recolección de datos, siguiendo por la creación, entrenamiento y ajuste de los modelos, y acabando por el desarrollo de una aplicación para la interacción con los mismos.

Para llevar a cabo este proyecto decidí escoger GitHub como herramienta de control de versiones, ya que es uno de los requisitos de este proyecto, además de estar ampliamente familiarizado con su tecnología, y Zenhub como herramienta de desarrollo ágil, la cual ya he usado con anterioridad. Es aquí dónde encontré mi primer problema en este propósito. Zenhub dispone de un programa de estudiantes que permite utilizar su herramienta de forma gratuita, y así empecé planificando el primer sprint y las primeras tareas. Fue en la preparación del segundo sprint dónde tuve el problema, ese programa gratuito se saturó, por lo que lo inhabilitaron, quedando mi cuenta fuera del mismo. Todo esto hizo que se me complicara la gestión de tareas del proyecto.

5.3. Recolección de datos

Para la recolección de datos inicialmente pasé por contactar con varias páginas web que albergan gran cantidad de información sobre la Fórmula 1. Contacté con 4 de las páginas más relevantes de este deporte: *statsf1*, *racing-reference*, *f1-fansite*, y *motorsportstats*. Incluso contacté con la página oficial de la *Fórmula 1*. Algunas de las páginas no me contestaron y otras simplemente me respondieron que no sería posible cederme datos, que no es la política de empresa.

Tras ver que no fue posible obtener los datos de alguna empresa, tomé la inevitable decisión de recopilarlos yo mismo. Para ello existe de una API llamada *Ergast Developer API*.

El término *API* (Application Programming Interface o Interfaz de programación de aplicaciones) se refiere a un conjunto de pautas y protocolos que nos permiten la comunicación entre varios componentes de software. Es

una interfaz que especifica cómo podemos hacer uso e interactuar con las funcionalidades de un sistema, biblioteca, framework o servicio.

En el caso de Ergast Developer API es un servicio web experimental que proporciona un registro histórico de datos de carreras de coches para fines no comerciales[2]. Este servicio nos proporciona datos del deporte de la Fórmula 1, desde el inicio del campeonato en 1950 hasta el día de hoy.

Después de conocer la fuente inicié la recopilación de datos, realizado en varias fases:

1. Inicialmente se obtuvieron los datos de todos los pilotos, constructores o equipos y circuitos de la Fórmula 1. Para obtener datos de los circuitos quise obtener la altitud en cuanto a la posición geográfica, ya que afecta notablemente al rendimiento del motor. Para ello me creé una cuenta en la herramienta para desarrolladores de Google, la Google Cloud Platform (GCP). La GCP es una plataforma de servicios de computación en la nube de Google. Se creó para ayudar a las empresas y desarrolladores a crear, implementar y administrar sus aplicaciones y servicios en la nube. Además, nos ofrece una amplia gama de servicios y herramientas. Más concretamente nos proporciona una API llamada *Maps Elevation API*, la cual nos permite a partir de un punto de latitud y longitud, la elevación del terreno entre otros datos. A partir de la información que nos da Ergast (latitud y longitud) he obtenido la altitud geográfica de los circuitos.
2. Más adelante se obtuvieron todos los datos de los resultados de las carreras y las clasificaciones. En este momento nos surge otro problema, sólo obtenemos datos desde el año 2003 de las clasificaciones. Decidí sacar los datos de la página oficial de la Fórmula 1 con técnicas de *webscrapping*. A continuación tuve que crear varios diccionarios para poder asociar a qué circuito y piloto corresponden dichos datos, ya que no cuentan el mismo nombre exacto en la página de la Formúla 1 que en la API de Ergast. Pero al recopilar y hacer el cruce de todos los datos me encontré con que no están todos. Faltan la mayor parte de los datos de clasificación y sólo disponemos de parte de los datos de posición de salida. Por ello usaremos únicamente la información obtenida en la API de Ergast, la cual dispone de los datos de posición de salida para todas las carreras.
3. En tercer y último lugar quise añadir el dato del tiempo meteórológico de las carreras, que es un factor muy influyente en las carreras.

Cuando una carrera es bajo lluvia la influencia del rendimiento del coche baja considerablemente y el desempeño del piloto es de mayor importancia. Es por ello que es un factor muy relevante en las carreras y por lo tanto debe tenerse en cuenta. Estos datos no se encuentran en ninguna de las webs de las carreras mencionadas anteriormente y en la API de Ergast tampoco. En un primer lugar consideré a través de la posición geográfica de los circuitos y de la fecha de realización de las carreras, consultar en alguna API de empresas o servicios de meteorología. Tras hacer varias pruebas con la API archive-api.open-meteo.com y la API de wunderground (api.weather.com), no se pudo determinar el tiempo exacto durante la carrera ya que podría haber llovido ese día pero no durante la realización del gran premio. Después de unos días reflexionando sobre cómo abordar este problema, encontré que en wikipedia se muestra cierta información sobre la temperatura del asfalto y sobre si llovió o no. Es por esto que mediante técnicas de *webscrapping* y la librería *BeautifulSoup*, creando un diccionario que detecta si hay palabras que indiquen que ha llovido, se ha establecido si llovió (representado con el valor *wet*) o no (*dry*) durante el gran premio.

5.4. Preparación y limpieza de datos

Tras la recopilación de los datos se procedió a prepararlos y limpiarlos para el modelo.

Fase 1, estados

En este momento decidí comenzar con el estado de finalización de las carreras. En la API de Ergast además de la posición de finalización de una carrera tenemos el estado en que se terminó esta, a continuación mostramos los estados más comunes en orden descendentes en la siguiente tabla 5.1.

Había demasiados estados finales, por lo que los reduje a tres estados para simplificarlo. Podemos verlos en la tabla 5.2.

Datos del piloto, circuitos y constructores

Más adelante crucé el *dataset* con los datos del piloto, de los circuitos y de los equipos, los cuales había obtenido con anterioridad. Aquí se contemplan datos como por ejemplo la nacionalidad del piloto, el país donde se ubica el circuito o la nacionalidad del equipo. A la hora de obtener la nacionalidad y

Estados	Representación del estado
Finished	Terminó la carrera sin problemas
+1 Lap	Terminó la carrera con una vuelta de retraso
Engine	Problemas en el motor del vehículo
Accident	Estuvo involucrado en un accidente
Collision	Estuvo involucrado en una colisión
Spun off	Se salió de pista durante la carrera
Gearbox	Problemas en la caja de cambios
Did not qualify	No logró calificar para la carrera
Suspension	Problemas en la suspensión del vehículo
Electrical	Problemas eléctricos en el vehículo
Transmission	Problemas en la transmisión del vehículo
Brakes	Problemas en los frenos del vehículo
Clutch	Problemas en el embrague del vehículo

Tabla 5.1: Estados de finalización de un piloto en una carrera.

Estados	Representación del estado
Finished	Terminó la carrera sin problemas
Driver mistake	No terminó por un error de pilotaje
Mechanical failure	No terminó por problemas mecánicos del vehículo
Engine failure	No terminó por problemas en el motor del vehículo

Tabla 5.2: Estados de finalización simplificados.

el país del equipo o circuito, observé que para el modelo sería interesante que ese dato fuera igual en los tres casos. Es por ello que modifiqué la nacionalidad de los pilotos para que aparezca en forma país y no de nacionalidad. Por otra parte, como amante del deporte que soy, tuve en cuenta que por temas de patrocinios muchos equipos cambian constantemente de nombre durante su historia, y es muy imperante que se tenga esto en cuenta para que los datos de un mismo equipo no se traten de forma separada. Con el fin de unificar los nombres del mismo equipo, busqué información sobre la historia de todos los constructores, la cual podemos ver la siguiente imagen 5.1, para modificar este dato.

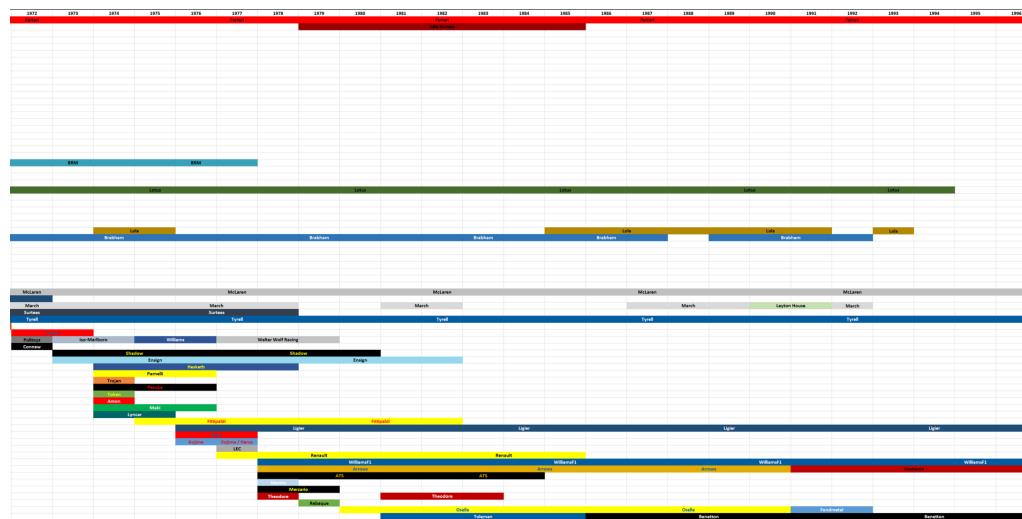


Figura 5.1: Extracto de los cambios de nombre en la historia de algunas de las escuderías[12].

Además, utilicé la fecha de realización del gran premio y la fecha de nacimiento del piloto para calcular la edad del piloto durante la carrera. Esto es importante, ya que con el tiempo los pilotos están más experimentados, por lo cual son más rápidos y cometan menos errores como podemos ver en la figura 5.2.

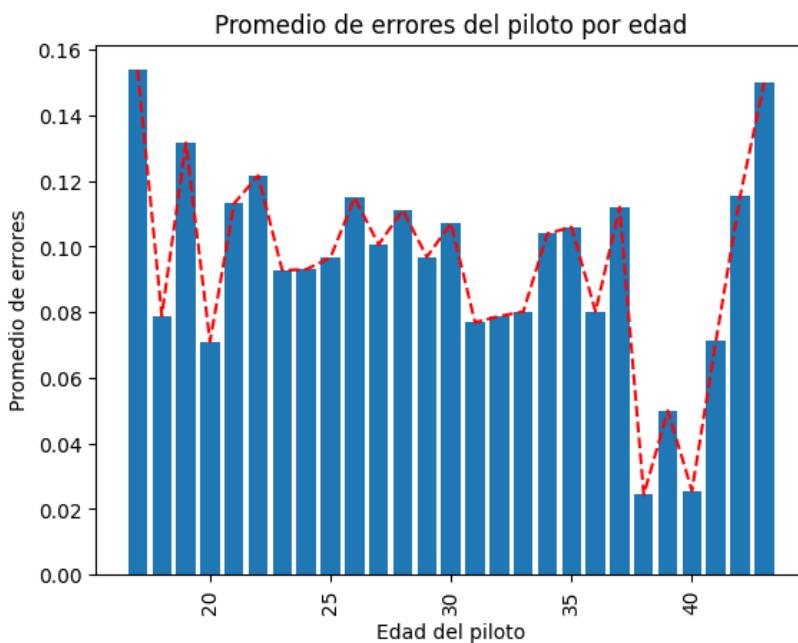


Figura 5.2: Promedio de errores de los pilotos según la edad

Por último se obtuvieron las puntuaciones de cada año del campeando tanto en constructores como en pilotos. Estos datos se pueden obtener directamente de la API de Eargast. Pero como la puntuación en función del resultado en carrera ha sido modificado con el tiempo, se va a proceder a adjudicar los puntos en función de la posición final de cada carrera con el sistema de puntuación actual. En este sistema sólo puntúan los 10 primeros pilotos, tal como podemos ver en la tabla 5.3.

Fase 3, Ganadores, *polemans*, fiabilidad de coches y consistencia de pilotos

Con los datos de las posiciones de salida obtendremos los *polemans*. La *pole* es para el piloto que consigue hacer la vuelta más rápida en clasificación y por ello sale en primer lugar en la carrera. El *poleman* es el piloto que consigue la *pole*. Es interesante conocer este dato porque en muchos circuitos en los cuales es muy difícil adelantar y salir por delante es de gran importancia como vemos en el gráfico de la figura 5.3, el 40 % de las victorias es saliendo desde la primera posición.

Posición	Puntuación
1°	25
2°	18
3°	15
4°	12
5°	10
6°	8
7°	6
8°	4
9°	2
10°	1

Tabla 5.3: Puntuación de cada piloto en función de la posición final de carrera.



Figura 5.3: Porcentaje de victorias cuando el piloto sale en la primera posición.

Y si comparamos las victorias saliendo desde la tercera posición o mejor el dato es aún más relevante (figura 5.4), un 80 %.

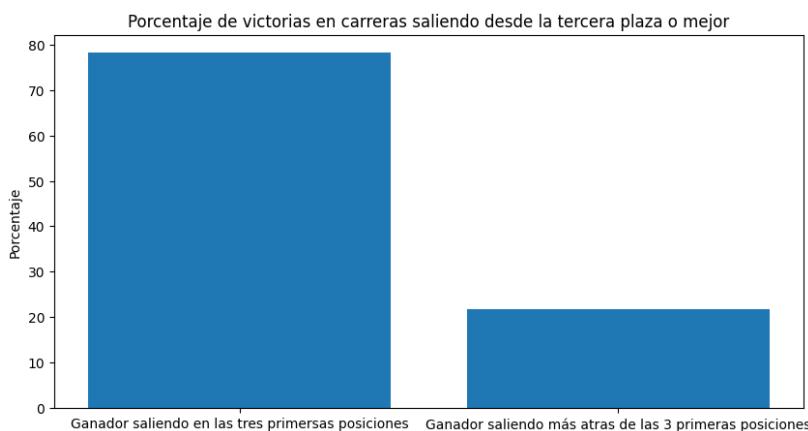


Figura 5.4: Porcentaje de victorias cuando el piloto sale en las tres primeras posiciones.

Además, vamos a obtener el ganador del gran premio a través de las posiciones de salida, para más adelante poder predecir este dato.

De igual modo es importante conocer la fiabilidad de los coches y la consistencia de los pilotos. En este caso usaremos la información de finalización de carrera (tabla 5.2), en función del total de carreras. Para la consistencia del piloto se usará el estado *driver mistake* y para la fiabilidad de los coches los estados *mechanical failure* y *engine failure*.

Limpieza de datos

Más adelante se procedió con la limpieza de datos, ya que debemos comprobar si hay algún dato que sea nulo, ya que los valores nulos afectarán negativamente a la eficiencia de los modelos [3]. Al comprobar los datos inexistentes, se eliminará la siguiente información:

- Tiempos de carrera: no contamos ni con el 50 % de los tiempos.
- Vuelta rápida de piloto en carrera: al igual que con los tiempos de carrera falta más de la mitad de los datos, y por ello la velocidad media de esa vuelta también será eliminada.
- Código de piloto y número de piloto: estos números no aportan información y además no hay prácticamente registros de ello.

Datos descargados que se eliminaran porque no aportan ninguna información sobre las carreras:

- Urls tanto de pilotos como de circuitos, carreras, o constructores.
- Nombres y/o apellidos de pilotos y constructores, ya que contamos con el id o alias de cada uno de ellos.
- Nacionalidad de piloto o equipo, y nombre de la localidad del circuito. Como ya hemos convertido la nacionalidad a nombre del país, ya no es necesario conocer dicha información.

5.5. Selección de características y codificación de los datos

En este punto se deben escoger las características que son más importantes para el entrenamiento del modelo. Para lograr esto, contrastaremos el uso de un algoritmo de selección de características frente a la selección manual de los rasgos que, en mi opinión, son más cruciales según mis muchos años de experiencia siguiendo este deporte.

Codificación de los datos

Es necesario tener los datos codificados para realizar la selección automática de características, ya que los modelos trabajan con datos numéricos.

En el aprendizaje automático existen dos grandes codificadores de características en función del tipo de variable.

1. Para variables categóricas con diferentes categorías se suele utilizar la codificación *One-Hot*, en la cual se crea una columna binaria para cada categoría.
2. En variables con un orden jerárquico se usan codificaciones ordinales, las cuales asignan a cada valor diferente de la variable un valor numérico normalmente ascendente o descendente.

Para nuestros datos se han escogido las siguientes codificaciones para cada uno de las variables:

- Año, posición final en carrera, posición de salida, ronda o serie, vueltas terminadas en carrera, puntos obtenidos tras la carrera, latitud o longitud de la posición geográfica del circuito, altitud del circuito respecto del nivel del mar, edad del piloto, consistencia del piloto

y fiabilidad del coche: estos datos ya son numéricos así que no es necesario codificarlos.

- Fecha del gran premio: para este dato hemos pasado la fecha a segundos con la referencia de la fecha mas antigua.
- Id de circuito, piloto y equipo: para esta variable se ha usado la codificación ordinal para asignar a cada piloto, circuito y equipo un número diferente de cada variable.
- Estados de finalización de carrera (estado final y estado simple final): para los estados se ha escogido también una codificación ordinal, asignando un número a cada variable.
- Clima: como tenemos dos tipos de climas se ha decidido codificarlos utilizar el método *One-Hot*, por lo que pasamos a tener dos variables en vez de una. Cada variable indica si llovió o no durante la carrera.
- Fecha de nacimiento del piloto: para este dato hemos utilizado el mismo criterio que con la fecha del gran premio, hemos pasado ese día a segundos con la referencia de la fecha mas antigua de nacimiento.
- País de procedencia de piloto, equipo y circuito: se ha considerado que este dato debía ser codificado con *One-Hot* debido a que el algoritmo debe poder detectar cuando un piloto o equipo corre en el gran premio de su país de procedencia de una mejor forma.
- Ganador del gran premio: no se ha codificado ya que ya es un dato numérico que representa un 1 si el piloto queda en primer lugar tras la carrera o un 0 si no.
- *Poleman* del gran premio: de igual modo que el ganador del gran premio no se ha necesitado codificar.

Selección de características manual

En primer lugar hice la selección manual. Gracias a este enfoque logro tener un mayor control sobre qué características se incluyen o excluyen en el modelo final, lo que me brinda la oportunidad de aplicar mi conocimiento y experiencia en este campo. Estas son las características elegidas:

- Año: el año es importante ya que tenemos que conocer este dato para distinguir entre carreras de un año y otro.

- Id de circuito, piloto y equipo: valores necesarios para diferenciar entre pilotos y equipos en cada circuito.
- Posición de salida del piloto: esta variable es de suma importancia en circuitos de complejidad de adelantamiento.
- Posición final del piloto en carrera: este dato es una de las variables objetivo del proyecto.
- Clima en carrera: uno de los valores más importantes en mi opinión, en mojado los rendimiento de los coches no son demasiado importantes y cobra mayor relevancia la destreza del piloto.
- País de procedencia de piloto, equipo y circuito: los pilotos de carreras suelen tener mayor motivación cuando corren en casa, al igual que cuando corren en el país de procedencia del equipo.
- Altitud sobre el nivel del mar donde se encuentra el circuito: los motores de los coches de Fórmula 1 se ven afectados por este dato, esto es por la cantidad de oxígeno en el aire. A mayor altitud menor concentración de oxígeno y mayor exigencia para los motores, ya que son motores de combustión y el oxígeno es de vital importancia en la explosión del combustible.
- Edad del piloto durante el gran premio: la edad es importante, ya que cuando un piloto es joven es menos experimentado y suele arriesgar más y por tanto comete más errores. Pero también cuando los pilotos son muy mayores suelen perder reflejos y algunos pierden la motivación para correr. Esto lo pudimos ver anteriormente en la figura 5.2.
- Ganador de la carrera: variable objetivo del proyecto.
- Poleman de la carrera: variable objetivo del proyecto.
- Consistencia del piloto: esta variable nos indica la consistencia del piloto a la hora de cometer errores, esto es importante porque cuantos más errores cometan más fácil es que no acaben la carrera.
- Fiabilidad del coche: es muy importante conocer este dato, en caso de un coche con poca fiabilidad es posible que el coche sufra un fallo en carrera y abandone. Podemos recordar la época de Fernando Alonso en McLaren con el motor honda entre los años 2015 y 2018 con la fiabilidad de los coches ese año, ilustrado en la figura 5.5. En dos de cada 10 carreras fallaban los dos coches abandonando por fiabilidad.

Podemos decir sin lugar a dudas que fue el peor coche en cuanto a fiabilidad de esa época.

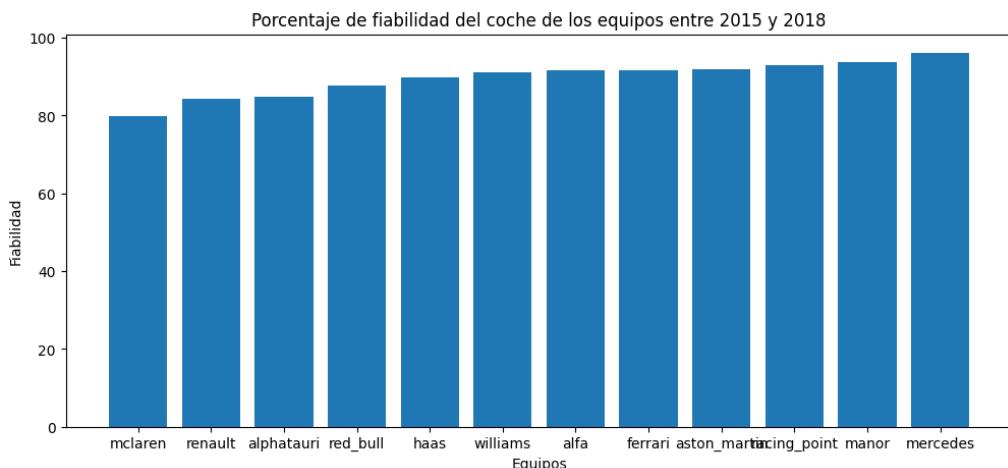


Figura 5.5: Fiabilidad de los coches entre 2015 y 2018

Selección de características automática

Para esta tarea contemplamos dos métodos muy efectivos, la eliminación recursiva o *Recursive Feature Elimination* y la eliminación hacia atrás o *Backward Elimination*, que ya fueron mencionados en la sección 3.4.

Este algoritmo se ha combinado con varios modelos para hacer una selección más óptima. Estos modelos son: *Decision Tree Regressor* y *Linear Regression*. Los dos son modelos de regresión ya que queremos una elección lineal de características. Por un lado los árboles de decisión son modelos muy versátiles que nos van a permitir capturar las relaciones no lineales y podremos detectar interacciones entre las diferentes características. Y por otra parte el algoritmo de regresión lineal tiene la ventaja de ser más interpretable y computacionalmente más eficiente.

Tras la ejecución del algoritmo se han eliminado las siguientes características:

Para el modelo con variable objetivo resultado de pilotos en carrera

1. Nacionalidades de constructores de Hong Kong, Nueva Zelanda, Sudáfrica, México, Rusia.
2. Posición geográfica en latitud y longitud.

3. Fecha de carrera.

Modelo con variable objetivo poleman de la clasificación

1. Nacionalidades de pilotos de Venezuela y Zimbabwe.
2. Posición geográfica en latitud y longitud.
3. Fecha del gran premio.
4. Ronda o serie.
5. Año.
6. Fecha nacimiento.

Para el modelo con variable objetivo ganador de la carrera.

1. Edad del piloto.
2. Posición geográfica en latitud y longitud.
3. Fecha del gran premio.
4. Ronda o serie.
5. Año.
6. Fecha nacimiento.

5.6. Entrenamiento

Para llevar a cabo el entrenamiento se ha escogido la librería *scikit-learn*, la cual cuenta con una gran cantidad de algoritmos y modelos de aprendizaje automático.

Se ha decidido hacer una división del 80 % para entrenamiento de datos y un 20 % para evaluación, para darle al modelo muchos ejemplos permitiendo descubrir y comprender patrones subyacentes en los datos. Como resultado, el modelo puede volverse más preciso y general. Además, reservar ese 20 % final de los datos para la evaluación le permite evaluar el rendimiento en datos a los que no estuvo expuesto durante el entrenamiento. Esto le da una indicación concreta de qué tan bien puede extraer datos a partir de nuevos datos y hacer predicciones. Inicialmente se pensó en la función `train_test_split`

de *scikit-learn* para llevar a cabo esta división. La cual nos permite dividir matrices o listas en subconjuntos aleatorios de entrenamiento y prueba [15].

Finalmente se utilizará la validación cruzada para encontrar la mejor división de datos posible, manteniendo el 20 % de datos de prueba. La validación cruzada o cross-validation es una técnica que consiste en dividir el conjunto de datos en subconjuntos de entrenamiento y prueba, y evaluar cada subconjunto para encontrar el más óptimo. Para ello se utiliza la librería *scikit-learn* y su algoritmo KFold, el cual realiza divisiones del conjunto de datos en K partes iguales, utilizando K - 1 partes para el entrenamiento y la parte restante para validación en cada una de las iteraciones. Asignando a K el valor de 5 para obtener ese porcentaje del 80 % de entrenamiento frente al 20 % de prueba. Con esto se logra maximizar la utilización de datos, gracias a permitir el uso de todo el conjunto tanto para entrenamiento como para validación.

Modelos del entrenamiento

Para el cálculo de las posiciones de carrera, se van a utilizar los siguientes modelos de regresión ya mencionados en la sección 3.4:

1. Linear Regression.
2. Decision Tree Regressor.
3. Random Forest Regressor.
4. Support Vector Regression.

Y para el cálculo del *poleman* y ganador, problemas de clasificación, se usarán los algoritmos de clasificación especificados en la sección 3.4:

- Logistic Regression.
- Decision Tree Classifier.
- Random Forest Classifier.
- Support Vector Classifier.
- KNeighbors Classifier.
- Gaussian NB (Naive Bayes).

Comparación de las soluciones y evaluación de los diferentes modelos utilizados

Vamos a comparar los algoritmo en función de la variable a predecir y el tipo de selección utilizado. La evaluación se ha llevado a cabo con la métrica R2 score, la cual cuenta con valores en el intervalo $(-\infty, 1]$, que implican las siguientes opciones:

- Número negativo: el modelo es peor que predecir con la media del valor.
- 0: el modelo es igual que predecir con la media del valor.
- Número positivo en el intervalo $(0, 1)$: el modelo es mejor que predecir con la media del valor.
- 1: Ajuste perfecto, todas las predicciones coinciden con la realidad.

Variable objetivo: posición final del carrera

En la figura 5.6 podemos apreciar que tanto el algoritmo RFE como la selección manual consiguen una similar puntuación. También se observa que tanto el modelo SVR como el Decision Tree Regressor obtienen una puntuación demasiado baja, esto se puede deber a que el conjunto datos no tiene muchas relaciones lineales. Sin embargo, el modelo Random Forest Regressor nos da valores más aceptables. Esto se puede deber a que uno de sus puntos fuertes es trabajar con problemas que tienen un conjunto de datos muy grande y de alta dimensión, como es nuestro caso.

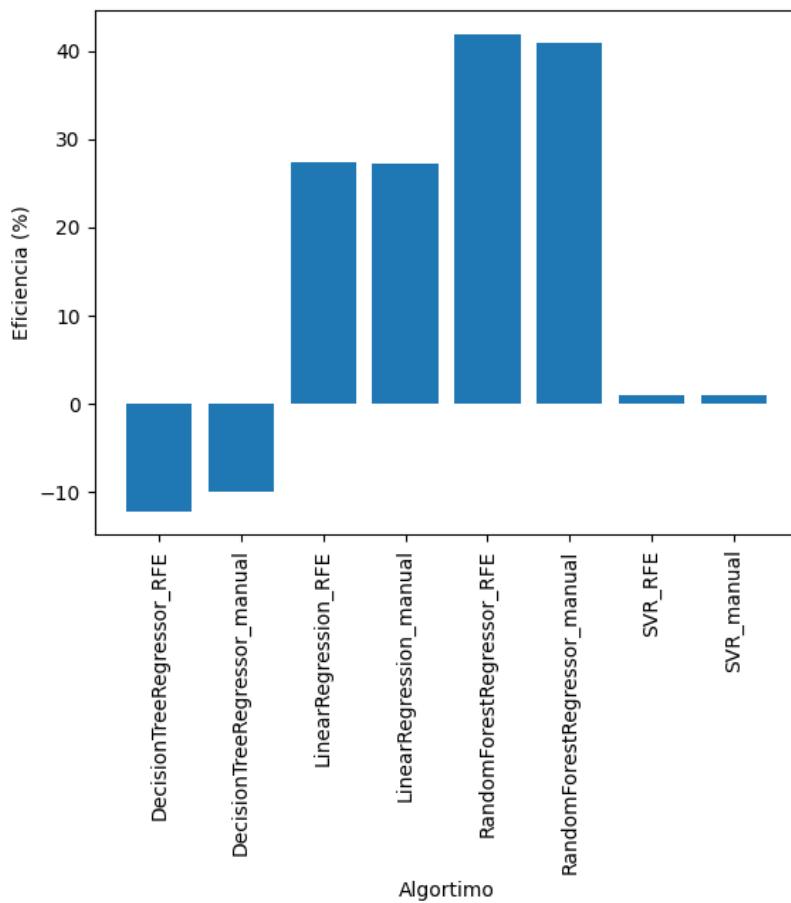


Figura 5.6: Eficiencia de los modelos para predecir la posición final del carrera.

Sin embargo, en la imagen 5.7 podemos apreciar que el conjunto de datos elegido miedicante el uso del algoritmo RFE reduce en hasta un 50 % de media los tiempos de ejecución. Además, podemos concluir que a pesar de que el modelo SVR tarde casi 80 veces más en entrenarse que el Random Forest Regressor no alcanza ni un cuarto de su eficiencia.

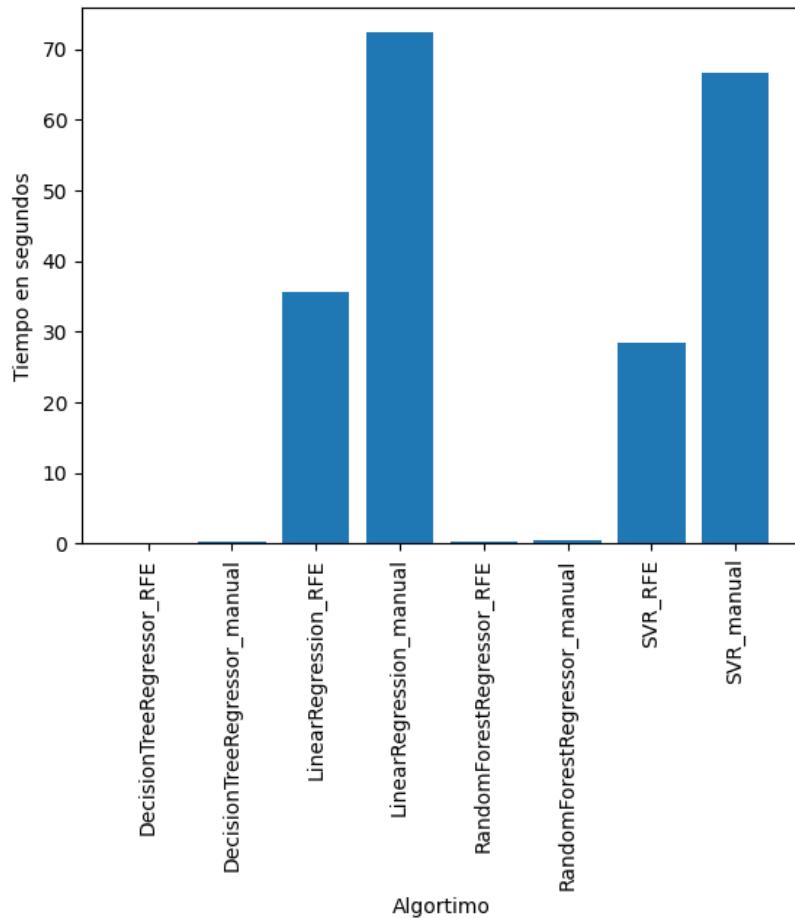


Figura 5.7: Tiempos de ejecución de los algoritmos para predecir la posición final del carrera.

Variable objetivo: ganador de la carrera

En este caso la variable a predecir es binaria. La eficiencia de los algoritmos para este problema es sustancialmente mayor que en el caso anterior. Hemos conseguido una eficiencia cercana al 100 %, algo muy útil para predecir en las carreras. El único modelo que no logra tal eficiencia es el modelo de Gaussian Naive Bayes, el cual no logra ni alcanzar el 40 % de los aciertos. Pero en este caso no se nota demasiado el uso de una selección u otra.

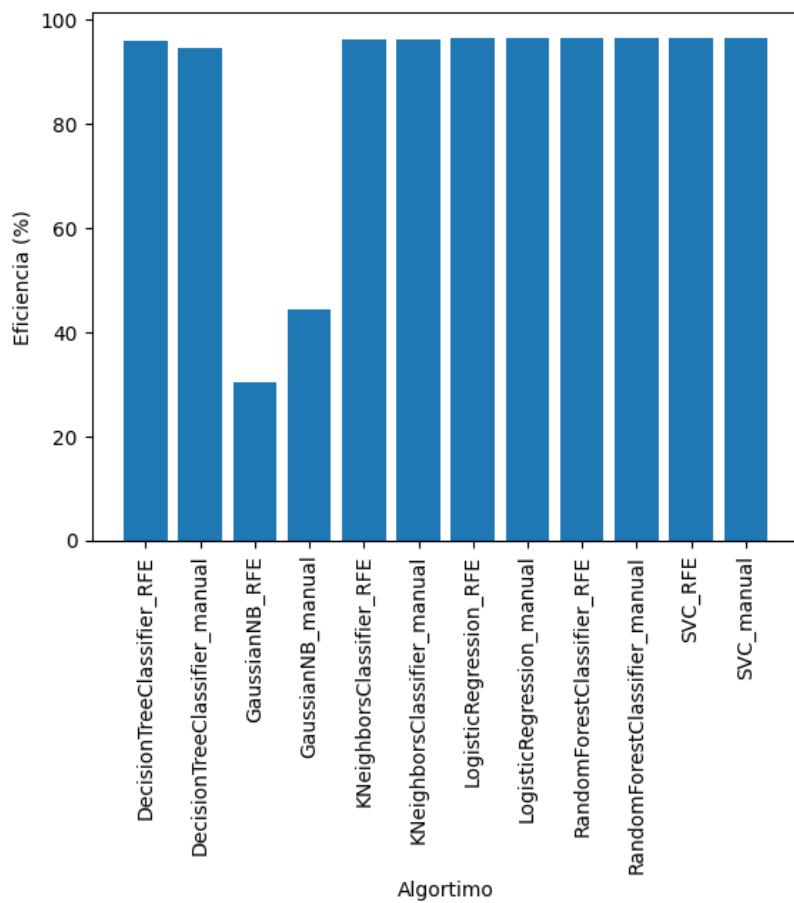


Figura 5.8: Eficiencia de los modelos para predecir el ganador de la carrera.

Los tiempos de ejecución (figura 5.9) marcan el mismo patrón que en el caso anterior, el algoritmo RFE es un 50 % más rápido (exceptuando el modelo de árboles de decisión y el de regresión logística), lo que supone una gran ventaja de computación. Además, podemos ver que estos algoritmos son muy rápidos, apenas tardan unos segundos en ejecutarse.

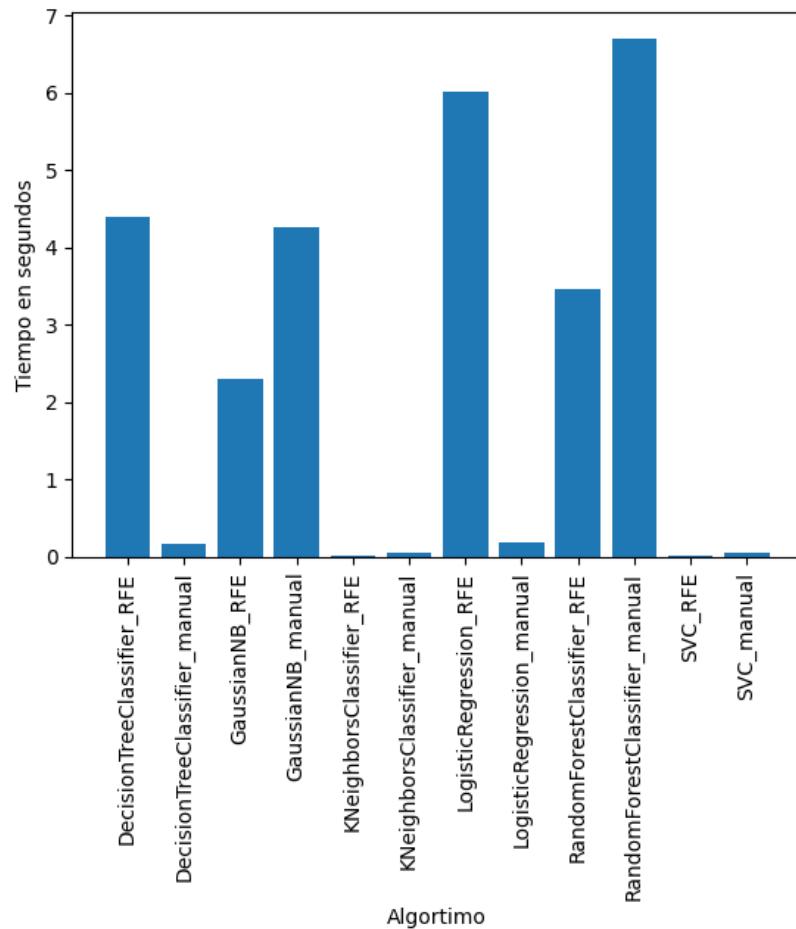


Figura 5.9: Tiempos de ejecución de los algoritmos para predecir el ganador de la carrera.

Variable objetivo: ganador de la pole

Como en el caso anterior hemos conseguido un muy alto porcentaje de acierto gracias a los algoritmos de clasificación, excepto con el modelo de Gaussian Naive Bayes.

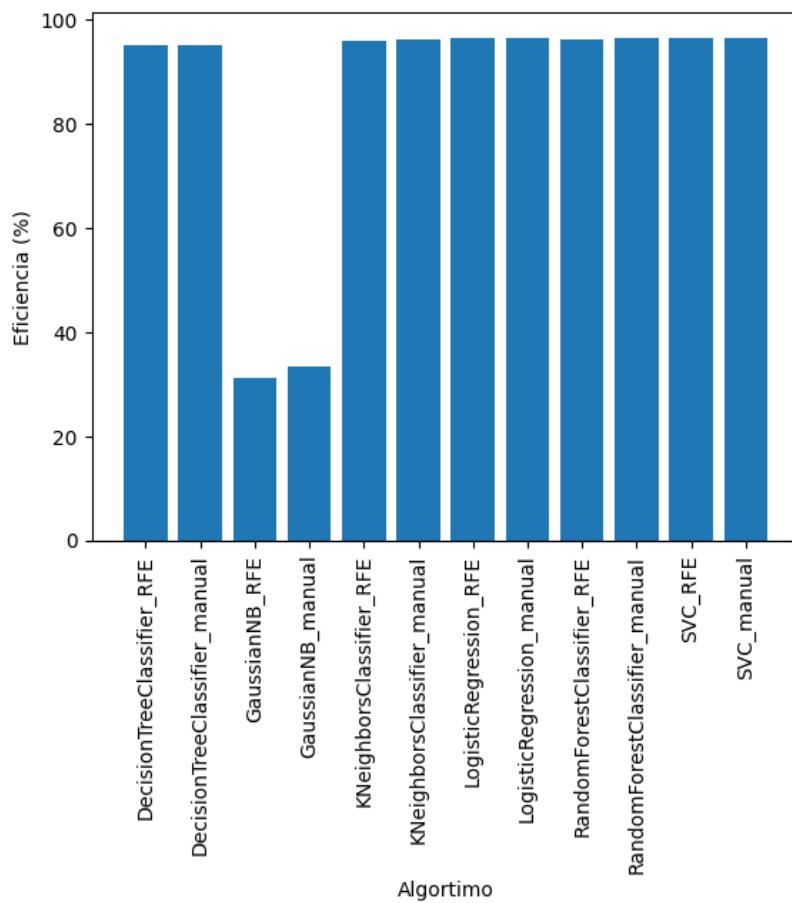


Figura 5.10: Eficiencia de los modelos para predecir el ganador de la *pole*.

También es importante ver cómo los tiempos de ejecución son muy bajos y se observa el patrón de mejores tiempos para el entrenamiento con selección utilizando el RFE (de igual modo a excepción del modelo de árboles de decisión y del de regresión logística).

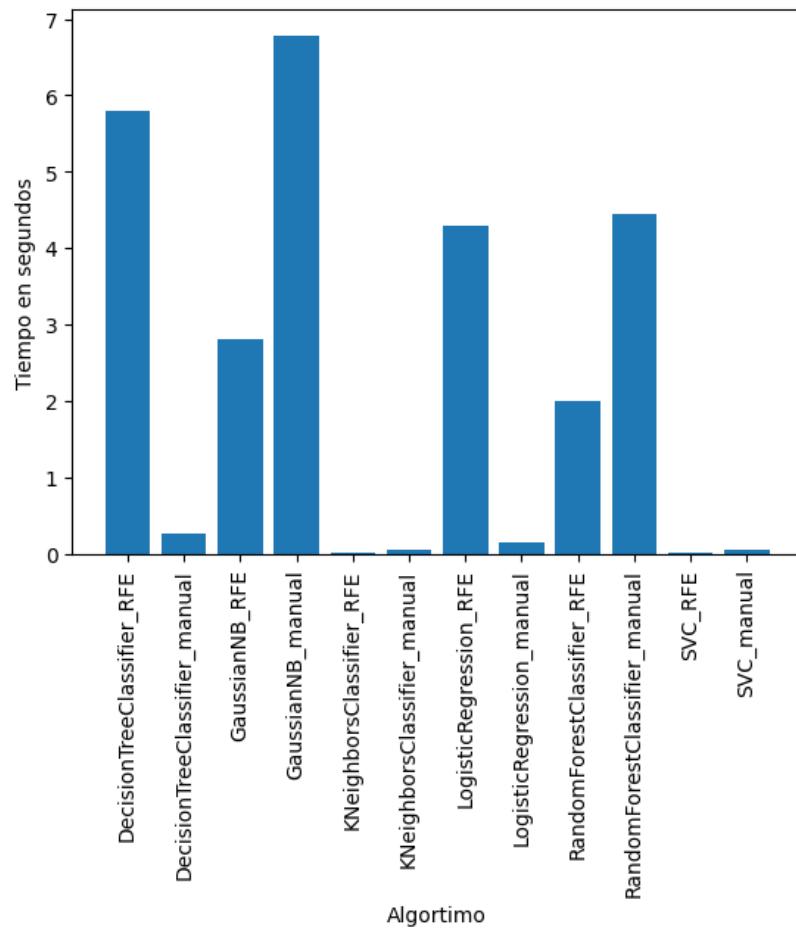


Figura 5.11: Tiempos de ejecución de los algoritmos para predecir el ganador de la *pole*.

5.7. Optimización del modelo

Tal como hemos observado en la imagen 5.6 el algoritmo Random Forest Regressor ha conseguido la mayor de las puntuaciones, y el Decision Tree Regressor la peor puntuación, y por ello vamos a escoger estos dos modelos con el conjunto de datos seleccionado por el algoritmo RFE para el ajuste (ya que es más eficiente computacionalmente hablando). No se han escogido los problemas de clasificación porque ya eran bastante precisos y veremos una mayor mejora en las demás opciones.

Random Forest Regressor

Para la optimización de este modelo se ha creado un diccionario que contiene los parámetros más importantes para el ajuste del modelo y se han generado todas las combinaciones posibles para obtener el conjunto de hiper parámetros que mayor eficiencia sea capaz de conseguir. En este caso el modelo con los parámetros iniciales suele ser bastante bueno, veamos si es posible ajustarlo más para mejorar la solución.

Los hiperparámetros que se van a combinar en sus diferentes posibilidades son [14]:

- ***bootstrap***: indica si se usan muestras bootstrap para construir los árboles. Si no se usa todo el conjunto. Si le indicamos que sí, que es como lo vamos a hacer, tenemos otros parámetros a elegir:
 - ***max_samples***: elige el número de muestras a extraer de X para entrenar cada estimador base. Generando con la función *linspace* de la librería *numpy*, la cual nos permite generar x números entre un número inicial y un final, una lista como la siguiente: *[500, 1000, 1500, 2000]*.
 - ***oob_score***: nos permite indicar si utilizaremos muestras fuera de la bolsa para estimar la puntuación de generalización. Se pondrá en *true*.
 - ***random_state***: sirve para controlar la aleatoriedad de las muestras bootstrap utilizadas al construir los árboles, además del muestreo de las características a considerar al buscar la mejor división en cada nodo (si el número máximo de características escogido es menor que el número de características totales). Utilizaremos los siguientes valores: 5, 10 y 15.
- ***max_depth***: profundidad máxima del árbol. Usaremos los valores: 5, 16, 27, 38, 50.
- ***n_estimators***: número de árboles. Se ha generado esta lista de valores a probar: 500, 1000, 1500 y 2000.
- ***criterion***: función utilizada para determinar la calidad de cada división. Tenemos 4 opciones:
 - ***squared_error*** utiliza el error cuadrático medio para minimizar la pérdida de L2 mediante el uso de la media de cada nodo terminal.

- **friedman_mse** usa el error cuadrático medio con la puntuación de mejora de *Friedman* para escoger las posibles divisiones.
- **absolute_error** mediante el error absoluto medio minimiza la pérdida L1 gracial al uso de la mediana de cada nodo terminal. El problema es que esta función es significativamente más lenta que "**squared_error**", y por ello es la única que no se probará.
- **poisson** divide los datos gracias a la reducción de la desviación de Poisson.

- **max_features**: número máximo de características s considerar en la búsqueda de la mejor división. Crearemos una lista con 10 números del tipo *float*, desde 0.1 hasta 1. Si le pasamos un número decimal entre 0 y 1 multiplicará este por el número total de características que tenemos, 109, obteniendo así el número que se utilizará en el parámetro. Hemos generado con *linspace* la siguiente lista: [0.1, 0.2, 0.3000000000000004, 0.4, 0.5, 0.6, 0.7000000000000001, 0.8, 0.9, 1.0]. Además podemos indicarle dos funciones para este parámetro: *sqrt* para que el valor del parámetro sea la raíz cuadrada del total de características y *log2* para el logartimo en base dos.

- **warm_start**: indica si se reutilizan o no las soluciones de anteriores para añadir y ajustar más estimadores al conjunto. Si es falso, se usa un bosque completamente nuevo. Probaremos indicando que se use y que no se use por separado.

- **ccp_alpha**: este parámetro define la complejidad de costes mínima para la poda, es decir, se elige el subárbol que cuente con la mayor complejidad de costes que sea menor que el valor indicado. Se comprobarán valores entre 0 y 0.1.

- **min_samples_split**: número mínimo de muestras para hacer la división de los nodos internos. Probando con 2, 5 y 10.

- **min_samples_leaf**: número mínimo de muestras para estar en un nodo hoja. Se probarán los siguientes valores: 1, 2 y 4.

- **verbose**: este valor controlará la verbosidad al ajustar y predecir los datos. Utilizaremos los siguientes valores: 0, 2 y 5.

- **n_jobs**: número de trabajos en paralelo durante el entrenamiento. Se establecerá en -1 para que se escoja el número máximo de procesadores disponibles.

Para hacer las pruebas con todas las combinaciones se usará la librería *itertools* con su función *product*, la cual nos permite a partir de varias listas de valores generar una lista con todas las combinaciones posibles, que son 2808. Tras 5 horas de ejecución y la gran cantidad de parámetros y combinaciones utilizadas, no se ha conseguido mejorar la puntuación de los valores por defecto, ya que este algoritmo es bastante bueno en esa configuración.

Decision Tree Regressor

En este caso es más probable que consigamos una mejora, pero ¿seremos capaces de generar una solución mejor que para el caso anterior? Para ello los hiperparámetros que se van a combinar en sus diferentes posibilidades son [13]:

- ***max_depth***: profundidad máxima del árbol. Vamos a utilizar la función *linspace* de la librería *numpy*. Le hemos dado 5 y 50 como rango y obtenemos las siguientes: 5, 16, 27, 38, 50.
- ***criterion***: función utilizada para determinar la calidad de cada división. Tenemos 4 opciones:
 - ***squared_error*** utiliza el error cuadrático medio para minimizar la pérdida de L2 mediante el uso de la media de cada nodo terminal.
 - ***friedman_mse*** usa el error cuadrático medio con la puntuación de mejora de *Friedman* para escoger las posibles divisiones.
 - ***absolute_error*** mediante el error absoluto medio minimiza la pérdida L1 gracias al uso de la mediana de cada nodo terminal. El problema es que esta función es significativamente más lenta que "*squared_error*", por ello va a ser la única en no utilizarse.
 - ***poisson*** divide los datos gracias a la reducción de la desviación de Poisson.
- ***max_features***: número máximo de características a considerar en la búsqueda de la mejor división. Usaremos la misma lista escogida en el caso anterior. Además podemos indicarle dos funciones para este parámetro: *sqrt* para que el número máximode carcterísticas sea la raíz cuadrada del total y *log2* para usar el logaritmo en base dos.
- ***splitter***: estrategia que se usa para la división de los nodos del árbol. Tenemos dos opciones disponibles:

- **best**: escoge la mejor división.
 - **random**: elige una división aleatoria.
- **ccp_alpha**: este parámetro define la complejidad de costes mínima para la poda, es decir, se elige el subárbol que cuente con la mayor complejidad de costes que sea menor que el valor indicado. Se utilizará el valor por defecto *0.0* ya que queremos la mayor complejidad.
 - **min_samples_split**: número mínimo de muestras para hacer la división de los nodos internos. Los valores probados son: 2, 5 y 10.
 - **min_samples_leaf**: número mínimo de muestras para estar en un nodo hoja. Se probarán los siguientes valores: 1, 2 y 4.

Se han conseguido probar 3510 combinaciones de parámetros generadas como en el caso anterior con la función *product* de la librería *itertools*. Este modelo es sustancialmente más rápido y hemos logrado ejecutar todas las posibilidades en cuestión de 15 minutos.

Comparación de valores

En la figura 5.12 podemos apreciar los valores conseguidos sin ajuste, con ajuste y con el conjunto de datos escogido a mano.

Se observa que en el modelo Decision Tree Regressor se ha conseguido gracias a ajustar los hiperparámetros obtener un acierto del 45 % respecto del modelo por defecto. Sin embargo con el modelo Random Forest Regressor hemos logrado únicamente 5.5 % más de rendimiento. El ajuste de hiperparámetros por defecto en este modelo es una solución bastante buena, pero se puede ajustar para ganar algo más de rendimiento. En cuanto al modelo de árboles de decisión hemos tenido que ajustarlo para poder exprimir su máximo rendimiento, algo que queda lejos de la solución con los parámetros por defecto.

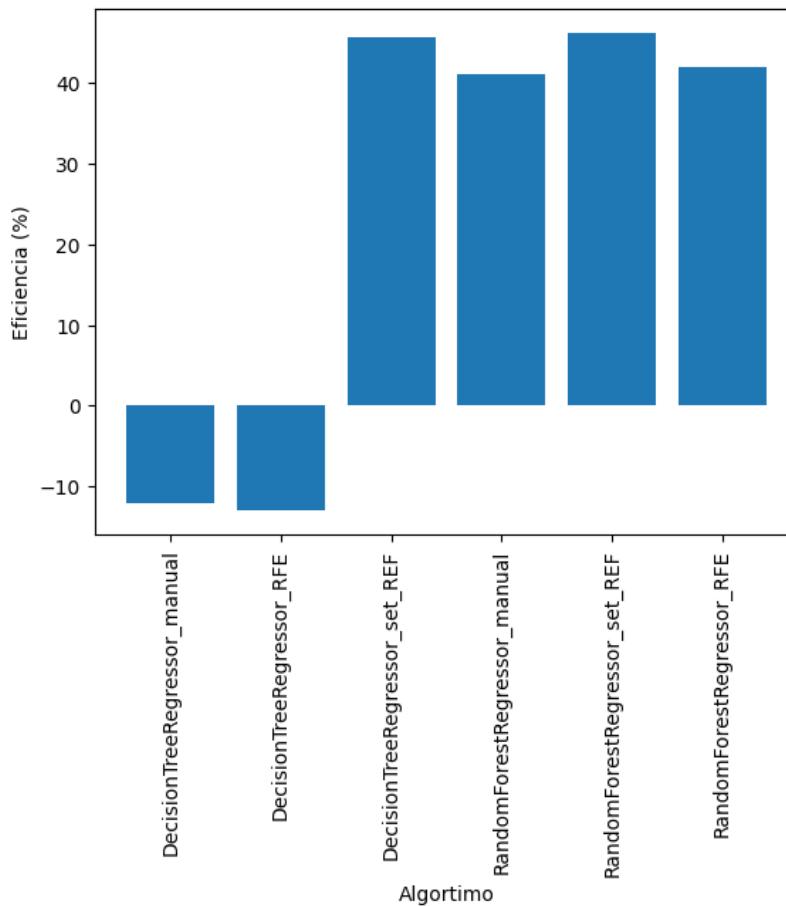


Figura 5.12: Valores anteriores y valores conseguidos con el ajuste de los hiperparámetros.

5.8. Aplicación para interacción

Tras haber realizado los entrenamientos y ajustes, llegamos a la parte final del proyecto, el desarrollo de la aplicación que nos va a permitir hacer predicciones. Comenzaremos con la arquitectura, más adelante se diseñará la interfaz de usuario y por último la lógica e implementación del modelo.

Arquitectura de la aplicación

En esta fase tenía que diseñar la interfaz de usuario para la aplicación y toda la lógica que conlleva, pero antes debía escoger el entorno en el que desarrollarla.

Entorno de desarrollo

En este momento tuve que decidir que entorno escoger para el desarrollo de la aplicación. Contaba con diferentes opciones:

1. Desarrollar una aplicación WinForms de .NET que es la tecnología con la que actualmente trabajo y estoy muy familiarizado con ella. Acabé descartando esta opción porque al ser una aplicación no demasiado compleja, esta tecnología requiere mayor esfuerzo. Además sólo se podría lanzar en Windows y no quería limitar al usuario final.
2. Hacer una aplicación para Android y que pudiera ejecutarse en cualquier móvil. También descarte esta opción porque de nuevo requería mayor esfuerzo de la complejidad de la propia app. Aunque pudiera ejecutarse sólo en un móvil Android, hoy en día la mayor parte de la gente cuenta con uno.
3. Python: al haber utilizado este mismo entorno y lenguaje para hacer el modelo, eliminaría la complejidad de unir dos sistemas diferentes. Escogí esta opción ya que además de minimizar esfuerzos me permitía desarrollar una aplicación de una forma más simple y para cualquier dispositivo que ejecutara python, el mismo lenguaje con el que he construido el modelo.

Diseño de la interfaz de usuario de la aplicación

Para hacer un primer boceto quise contar ya con la librería que usaría para poder desarrollar la interfaz, y python cuenta con muchas librerías con las que se pueden desarrollar aplicaciones y de todas ellas las más conocidas son:

- Tkinter: interfaz Tk, interfaz por defecto de Python para el kit de herramientas de GUI Tk [8].
- PyQt: es un conjunto de enlaces de Python para el framework Qt [10]. Nos proporciona una gran cantidad de widgets, y es muy conocida por su capacidad para crear interfaces muy personalizables y atractivas para el usuario.
- PySide: es otro conjunto de enlaces de python para Qt.
- Kivy: es un framework de python de código abierto para desarrollar aplicaciones multiplataforma. Está más enfocado a dispositivos de uso táctil, por lo que queda descartada.

- wxPython: conjunto de herramientas GUI (interfaz gráfica) multiplataforma y de código abierto. Su punto fuerte es su facilidad de uso y capacidad para crear interfaces gráficas robustas y altamente funcionales [17].

Inicialmente escogí *Tkinter* por su simpleza, su curva de aprendizaje es muy corta. Con esta librería realicé el primer boceto, con unos pocos desplegables en formato vertical para seleccionar el piloto y demás datos, el cual podemos ver en la imagen 5.13:

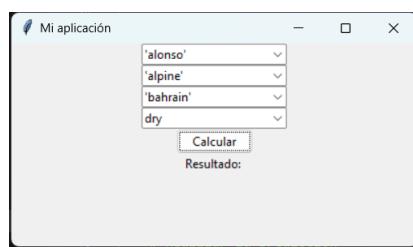


Figura 5.13: Primer boceto de la interfaz de usuario.

La disposición vertical de los desplegables no me convencía así que decidí ponerlos en horizontal y añadir una imagen de fondo (5.14).

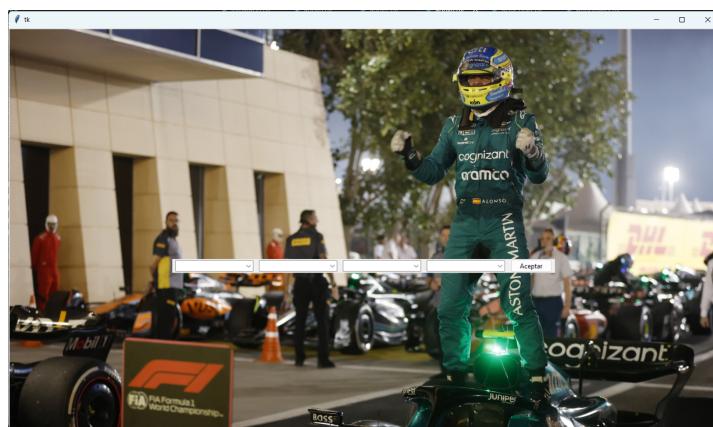


Figura 5.14: Boceto de la interfaz de usuario con desplegables horizontales y con imagen de fondo.

Cuando intenté poner alguna etiqueta para poner los datos del desplegable y algún botón me di cuenta de que con la librería *Tkinter* tenía lagunas limitaciones, por lo que decidí probar con la librería *wxPython*. Con esta otra librería en poco tiempo conseguí poner los desplegables con etiquetas

de texto y un cuadro de texto a modo de log para ver que pasos seguía la aplicación mientras trataba los datos antes de la predicción (figura 5.15).

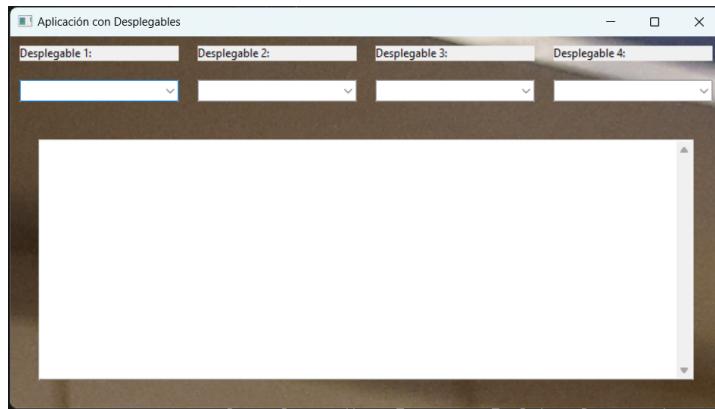


Figura 5.15: Boceto de la interfaz de usuario realizado con la librería *wxPython*.

Tras añadirle los botones para predecir los resultado, como vemos en la figura 5.16, seguía sin convencerme el diseño que podíamos alcanzar con esta librería.

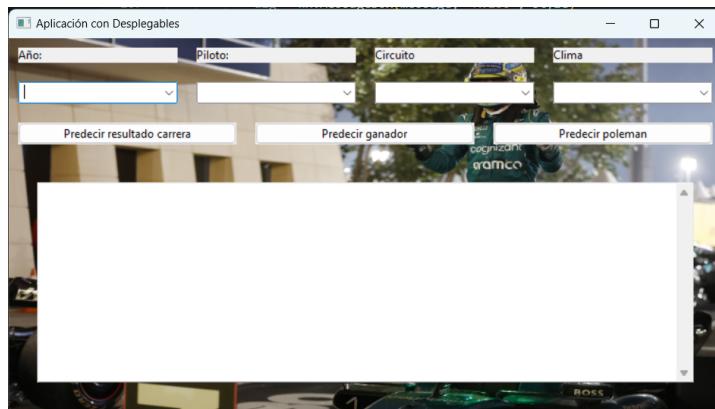


Figura 5.16: Diseño de la interfaz de usuario de la aplicación con WX.

Por último probé con PyQt pese a que tiene una curva de aprendizaje mayor, nos proporciona una gran cantidad de posibilidades. Una vez empecé con esta librería se me ocurrió en vez de usar desplegables, mostrar todos los circuitos con imágenes y que estas sean el propio botón de selección. De ahí salió la primera pestaña funcional, escoger el circuito (figura 5.17).



Figura 5.17: Diseño de la pantalla dónde escogemos el circuito.

Después se debe escoger el cima en carrera, lo cual decidí utilizar los neumáticos para escoger clima seco o mojado, visto en la imagen 5.18.



Figura 5.18: Diseño de la pantalla dónde escogemos el clima.

Por último entramos en la pestaña dónde podemos hacer las predicciones, contamos con 3 botones para predecir y uno para ir al inicio. En la imagen 5.24 podemos ver una predicción de los resultados del GP de Mónaco. Para predecir el ganador o poleman se verá como ilustra la figura:

Predecir posiciones		Predecir ganador		Predecir poleman		Ir a inicio	
1ST	VERSTAPPEN			2ND	MAGNUSEN		
3RD	TSUNODA			4TH	ALBON		
5TH	STROLL			6TH	NORRIS		
7TH	BOTTAS			8TH	HAMILTON		
9TH	PEREZ			10TH	ALONSO		
11TH	DEVRIES			12TH	LECLERC		
13TH	OCON			14TH	SAINZ		
15TH	RUSSELL			16TH	GASLY		
17TH	ZHOU			18TH	PIASTRI		
19TH	HULKENBERG			20TH	SARGEANT		

Figura 5.19: Diseño de la pantalla donde hacemos predicciones de el resultado total.

Predecir posiciones		Predecir ganador		Predecir poleman		Ir a inicio	
1ST	VERSTAPPEN						

Figura 5.20: Diseño de la pantalla donde hacemos predicciones del ganador.

En el caso de que el circuito escogido aún no haya concluido la clasificación y no se conozcan las posiciones de salida, debemos escoger una a una las posiciones de salida de los pilotos que deseemos para hacer la predicción. Se muestra esta pestaña para elegir la posición de cada piloto [5.21](#), las cuales se eliminan una vez seleccionadas para un piloto, como vemos en la figura [5.22](#)



Figura 5.21: Diseño de la pantalla dónde escogemos las posiciones de salida del piloto.

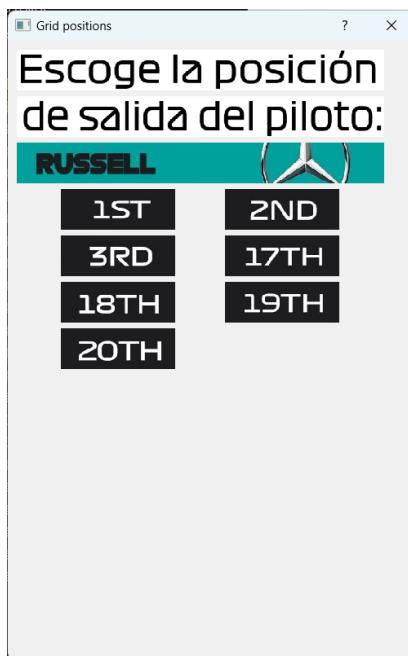


Figura 5.22: Diseño de la pantalla dónde escogemos las posiciones disponibles de salida del piloto.

Además se ha diseñado una imagen principal al inicio de la aplicación:



Figura 5.23: Diseño de la pantalla de inicio de la aplicación.

Funcionalidad de la aplicación

Para poder interactuar con el modelo se guardaron los modelos con la librería *joblib* en el entrenamiento y cargaron en la app. *Joblib* nos provee de varias funciones, entre ellas *load* y *dumb*. Estas funciones nos permiten guardar y cargar objetos de python en archivos binarios, o lo que es lo mismo una manera muy efectiva de serializar y deserializar objetos de python, algo muy útil para procesamiento de datos y aprendizaje automático [9].

Para poder ejecutar las predicciones se obtendrán los datos de la propia API de Ergast y más adelante se codificarán con los mismos codificadores que se han utilizado en la codificación para el entrenamiento, también gracias a las funciones de *joblib* para guardarlos y luego cargarlos en la app.

Como hemos visto en la arquitectura se han creado 5 archivos para llevar a cabo la lógica:

- **AppCircuits:** para obtener los datos de los circuitos de la temporada escogida.
- **AppConstructors:** para obtener los datos de los equipos de la temporada escogida.

- **AppDrivers:** para obtener los datos de los pilotos de la temporada escogida.
- **AppResults:** para obtener los datos de los resultados de la clasificación de la carrera escogida para poder hacer la predicción de la posición de carrera o si ha ganado o no la carrera.
- **AppCoder:** para combinar y codificar todos los datos obtenidos.
- **DialogSelectGrid:** para la ventana emergente de elección de posiciones de salida.

Primero se seleccionará el circuito que desencadenará la descarga de los datos necesarios. Y más adelante se podrá escoger entre clima seco y mojado. Después ya solo debemos hacer click en la predicción deseada.



Figura 5.24: Predicción del resultado total.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

6.1. A machine learning approach to predict the winner of the next F1 Grand Prix - Veronica Nigro

Este trabajo es un proyecto desarrollado por Veronica Nigro. Este proyecto se encuentra documentado en un [artículo](#) de [Towards Data Science](#), una plataforma en línea dedicada a compartir información y recursos sobre ciencia de datos, inteligencia artificial y aprendizaje automático. Este proyecto ha construido un algoritmo para predecir el ganador en las 21 carreras de Fórmula 1 de la temporada 2019 [11]. Podemos encontrar el repositorio que alberga el proyecto en el siguiente [enlace](#) o en el propio artículo.

6.2. Formula 1 Race Prediction

Formula 1 Race Prediction es un proyecto de aprendizaje automático utilizando el módulo Scikit-learn en Python para predecir el resultado de las carreras de Fórmula 1. El objetivo de este predictor es predecir si el piloto gana (primera posición), entra en los puntos (entre el segundo y el décimo piloto) o se queda fuera de los puntos (undécimo o peor), además cuenta con una interfaz gráfica que proporciona esta información (podemos verlo

en la figura 6.1). El trabajo se encuentra alojado en el repositorio [Formula 1 Race Prediction](#).



Figura 6.1: Interfaz de Formula 1 Race Prediction.

6.3. Ventajas y debilidades respecto a la competencia

Características	AppFran	F1 Grand Prix	Veronica Nigro	Formula 1 Race Prediction
Predicción del ganador	✓		✓	✓
Interfaz de usuario	✓		✗	✓
Gratis	✓		✓	✓
Predicción resultado carrera	✓		✗	✗
Predicción del ganador de la pole	✓		✗	✗

Tabla 6.1: Comparativa de características de los proyectos.

Las principales fortalezas del proyecto son:

- **Funcionalidades:** Contamos con diferentes tipos de predicciones que permiten al usuario final elegir entre las diferentes opciones.
- **Interfaz de usuario:** la interfaz de usuario de la aplicación es muy intuitiva, sencilla y con una curva de aprendizaje rápida. Además, es accesible a todo el mundo, incluso sin conocimiento previo del tema.
- **Aplicación gratuita.**

Las principales debilidades son:

6.3. VENTAJAS Y DEBILIDADES RESPECTO A LA COMPETENCIA

- Cuenta con una menor probabilidad de acierto en el caso de la predicción del resultado total de la carrera.
- Menor dimensionalidad en los datos.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Procedo a mostrar las conclusiones que se derivan del desarrollo del presente proyecto:

- El objetivo general del proyecto se ha cumplido con éxito. Se ha desarrollado un modelo de predicción de resultados de Fórmula 1, el cual puede utilizarse de una manera muy simple gracias a la interfaz de usuario por cualquier persona.
- También se ha completado el ajuste de los modelos con éxito consiguiendo mejoras notables en alguno de ellos.
- Por otra parte he comprobado que aunque la selección manual de características puede ser un buen enfoque, la selección mediante algoritmos como la eliminación recursiva de características son más eficientes computacionalmente en la mayor parte de las ocasiones.
- Gracias a la realización del proyecto he profundizado en el conocimiento sobre inteligencia artificial, modelos predictivos y machine learning.
- Este proyecto ha abarcado una buena parte de los conocimientos que he obtenido durante mi estancia en la universidad. Pero no sólo he hecho uso de ellos, si no que también he estudiado y adquirido nuevos conocimientos requeridos para la realización del proyecto. Entre ellos se encuentran: Webscrapping, Scikit-learn, L^AT_EX, pyqt, etc.
- He aprendido a hacer búsquedas bibliográficas rápidas y efectivas como resultado de los requerimientos de investigación del proyecto.

- El período de desarrollo de este proyecto ha supuesto el uso de numerosas tecnologías y herramientas. Todas ellas han contribuido a elevar su calidad. Sin embargo, algunas de ellas han requerido una sobrecarga de trabajo considerable. A pesar de esto, la información aprendida será muy útil para futuros emprendimientos.

7.2. Líneas de trabajo futuras

A continuación, discutiremos cada extensión o mejora potencial que sería aplicable al trabajo actual.

Aumentar la dimensionalidad de los datos

Un punto de mejora en este proyecto sería lograr mayor cantidad de datos de clasificación, recopilar datos de los pit stops o de las vueltas rápidas, adelantamientos, etc.

Añadir funciones a la aplicación

Se podría añadir un apartado en la aplicación para añadir datos al conjunto y que se vuelvan a entrenar los modelos automáticamente, y si los datos mejoran, adoptar esos nuevos modelos.

Interpretación de resultados

Podrían desarrollarse métodos para comprender y justificar las opciones del modelo a medida que interpreta las decisiones. El usuario final puede beneficiarse de esto al sentirse más seguros de las predicciones y al poder comprender las variables que afectan a los resultados.

Aplicación en dominios particulares

Una buena aplicación de este proyecto podría ser su uso en dominios particulares como pueden ser análisis deportivo, simulaciones o videojuegos.

Bibliografía

- [1] Formula 1®. Formula 1 - the pinnacle of motorsport, 2023. [Internet; descargado 18-abril-2023].
- [2] Ergast Developer API. Api documentation, 2023. [Internet; descargado 12-junio-2023].
- [3] S. Chen, C. A. Dembia, C. R. Landis, M. J. Mckenna, C. J. Deemer, and I. Allison. The impact of missing data on machine learning models: a comparative study. 2019.
- [4] Datademia. ¿qué lenguajes de programación deberías conocer en 2022?, 2023. [Internet; accedido 19-abril-2023].
- [5] Emmert-Streib F, Yang Z, Feng H, Tripathi S, and Dehmer M. An introductory review of deep learning for prediction models with big data. *Front. Artif. Intell.*, 3:4, 2020.
- [6] Python Software Foundation. Historia y licencia, 2023. [Internet; descargado 19-abril-2023].
- [7] Python Software Foundation. Preguntas frecuentes generales sobre python, 2023. [Internet; descargado 19-abril-2023].
- [8] Python Software Foundation. tkinter — interface de python para tcl/tk, 2023. [Internet; descargado 26-junio-2023].
- [9] Joblib. Joblib: running python functions as pipeline jobs, 2023. [Internet; descargado 25-junio-2023].
- [10] Riverbank Computing Limited. What is pyqt?, 2023. [Internet; descargado 26-junio-2023].

- [11] Veronica Nigro. Formula 1 race predictor - a machine learning approach to predict the winner of the next f1 grand prix, 2020. [Internet; descargado 27-junio-2023].
- [12] User of Reddit. History of f1 team ownership in one graphic, 2023. [Internet; descargado 13-junio-2023].
- [13] Scikit-learn. sklearn.ensemble.decisiontreeregressor, 2023. [Internet; descargado 22-junio-2023].
- [14] Scikit-learn. sklearn.ensemble.randomforestrgressor, 2023. [Internet; descargado 22-junio-2023].
- [15] Scikit-learn. sklearn.modelselection.taintestsplit, 2023. [Internet; descargado 26-junio-2023].
- [16] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach, Fourth Edition*. IEEE Intelligent Systems, 2021.
- [17] The wxPython Team. Overview of wxpython, 2023. [Internet; descargado 26-junio-2023].
- [18] Zhi-Hua Zhou. *Machine Learning*. Springer, 2021.