GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

MODELOS PREDICTIVOS PARA LA ESTIMACIÓN DE TRAYECTORIAS DE RECURSOS DE VIGILANCIA Y CONTINGENCIA DE DESASTRES NATURALES

Alumna: Ortiz, Fresno, Leticia
Director: del Ser, Lorente, Javier

Curso: 2017-2018

Fecha: Martes, 17 de julio del 2018

PÁGINA EN BLANCO

Resumen trilingüe

Resumen

La mala gestión de los recursos disponibles es una de las principales causas de la rápida expansión de los incendios y también puede ocasionar víctimas. Esa es la razón por la cual es importante planificar los recursos disponibles, así como la geolocalización de cada unidad (la posición en la que cada unidad estará en cada momento). Por ello se creará una red neuronal que nos permita saber de antemano dónde estará cada unidad disponible.

Para crear la red neuronal, se usarán diferentes tipos de regresores obteniendo una precisión de cada uno de ellos. Para concluir, se pretende decidir cuál de todos los métodos utilizados genera datos más precisos a partir de las predicciones obtenidas.

Palabras clave: Recursos, Red neuronal, Regresór, Precisión, Predicción.

Laburpena

Baliabide erabilgarrien kudeaketa okerra suteen hedapen azkarraren arrazoi nagusietako bat da eta hildakoak sor ditzake. Hori dela eta, eskuragarri dauden baliabideak eta unitate bakoitzaren geolokalizazioa (unitate bakoitza une bakoitzean egon dadin) planifikatzea garrantzitsua da. Horregatik, sare neuronal bat sortuko da eskuragarri dagoen unitate bakoitzerako posizioa aldez aurretik jakin dezagun.

Sare neuronala sortzeko, erregresio mota ezberdinak erabiliko dira, haien zehaztasuna lortuz. Amaitzeko, erabilitako metodo guztien artean zeinek sortzen dituen iragarpenik zehatzenak zehaztuko da.

Hitz gakoak: Baliabideak, Sare neuronala, Erregresioa, Iragarpena, Zehaztasuna.

Abstract

The mismanagement of available resources is one of the main causes of the rapid expansion of fires and can also result in casualties. That is the reason why it is important to plan available resources, as well as the geolocation of each unit (the position where each unit will be at each moment). That is why a neural-network is going to be created, which will allow us to know in advanced where each available unit will be.

To create the neural-network, different types of regressors are going to be used obtaining precision from each of them. To conclude, it is intended to decide which of all methods used for prediction generates more accurate data from the obtained precisions.

Key words: Resources, Neuronal-network, Regressor, Prediction, Precision.

REFERENCIAS DE CONTENIDO

Ilustraciones

Tablas

Ecuaciones

Acrónimos

- CRISP — Cross Industry Standard Process for Data Mining
- RFID — Radio Frecuency Identification
- GPS — Global Positioning System
- ERP — Enterpise Resource Planning
- CRM — Customer Relationship Management
- XML — Extensible Markup Language
- RRSS — Redes Sociales
- ADN — Ácido Desoxirribonucleico
- TS — TimeStamp
- DT — Decision Tree
- MLP — Multi-Layer Perceptron
- SVR — Epsilon-Support Vector Regressor
- SK-Learn — Scikit-Learn
- SIG — Sistema de Información Geográfica
- CDC — Center of Disease Control and Prevention
- TFG — Trabajo Fin de Grado

Tabla de contenido

1 Introducción

El planeta Tierra ha sufrido diferentes tipos de desastres naturales a lo largo de la historia, entre los que destacan: el tsunami ocurrido en 1958, que generó una ola de hasta 500 metros, el huracán Katrina, que azotó Estados Unidos en 2005 y que produjo centenas de destrozos y finalmente, el incendio producido en California en Diciembre de 2017 [1].

Estos altercados generan decenas de muertes anualmente, de los cuales un porcentaje bastante elevado podrían evitarse si las tropas de salvamento llegasen antes al lugar donde se produjo el altercado.

Los desastres naturales es un tema que actualmente se hace más común a causa de su incremento en los últimos años. A pesar de que el mundo ya los ha vivido en diversas ocasiones, sigue sin estar preparado y sin darle la importancia debida para hacerles frente a los desastres futuros e incluso para evitarlos o tratar de que perdamos menos como sociedad en cuanto a vidas humanas y recursos naturales y económicos se refiere.

Las cifras de las pérdidas de los recursos naturales y económicos son alarmantes a nivel mundial. A pesar del mal generados en la sociedad, algunos de los desastres naturales ocasionados en los últimos años, como pueden ser los cientos de incendios que ocurren por todo el mundo, son generados a propósito por el ser humano para obtener beneficios propios.

Asimismo, en los incendios se podrían evitar males mayores si los bomberos supieran de antemano la trayectoria que va a seguir el fuego a extinguir. Esto podría hacer que las unidades de rescate se anticipasen al movimiento del fuego y que lo extinguiesen con mucha más facilidad. De esta manera, se lamentarían menos daños en la flora y fauna, así como evitar que las vidas humanas corrieran un riesgo innecesario como ocurrió en el incendio de Victoria (Australia) en 2009 que ocasionó 173 muertes humanas y grandes destrozos en viviendas y estructuras [14].

A parte de los incendios mencionados anteriormente, existen más casos en los que lamentar muertes, como el incendio de Julio de 2015 en Guadalajara donde 11 bomberos murieron a causa de la poca efectividad implementada a la hora de desplegar las tropas [15].

Una forma efectiva de extinguir un incendio con facilidad sería desplegar un número elevado de tropas, pero esta solución generaría un alto gasto de personal. Además, en

caso de producirse varios incendios simultáneamente por las altas temperaturas que tiene que soportar la tierra, como puede ocurrir en verano (con temperaturas de hasta 40 º en algunas localidades), el número de operarios a contratar generaría un gasto elevado innecesario de millones de euros.

Otra forma de abordar este problema, podría ser la predicción de movimientos de las tropas. Es decir, teniendo en cuenta las posiciones que han ido tomando las tropas durante un incendio, proceder a la generación de posibles futuros movimientos.

Mediante este proyecto se pretende realizar un estudio exhaustivo de posibles métodos de predicción haciendo uso de diferentes librerías para Python que soporten big data y machine Learning .

2 Contexto

Los incendios causan miles de muertes anualmente, para ser exactos, cada año se producen en España más de 11000 incendios causando una media de 4 fallecidos y 37 heridos. A parte de las muertes humanas, también hay que lamentar daños en el medio ambiente, como pueden ser el destrozo de la flora y fauna [2].

Las respuestas a los diferentes desastres naturales pueden ser muy variadas debido a diversos factores:

- La geolocalización donde se produce el incidente.
- El número de recursos disponibles en el instante del suceso.
- Los diferentes tipos de recursos disponibles (Drones, vehículos, personas...).
- La variación que puede tomar un suceso (como puede ser el cambio de dirección del fuego en un incendio).

Por todos estos factores se debe utilizar un predictor optimo que permita a las unidades saber de antemano su futura posición.

Este proyecto tiene como meta el diseño de métodos de predicción precisos que permitan un aprendizaje de máquina rápido y preciso donde la predicción se realice a partir de ciertas muestras.

Como las muestras son tomadas en un periodo corto de tiempo, se va a trabajar con un gran volumen de datos. Para ello, se va trabajar con librerías que permitan implementar Machine Learning y gestionar Big Data con gran facilidad.

Para minimizar los riesgos durante los desastres naturales, se pretende estimar las trayectorias de las unidades de salvamento mediante modelos predictivos. Para ello se tendrá en cuenta una serie de factores como pueden ser la variación del movimiento de la unidad, su posición inicial y el tipo de unidad que se trata.

Para poder utilizar un predictor correctamente se entrenará previamente con datos ya medidos (de sucesos ya solucionados o del mismo desastre).

3 Objetivos y Alcance

Por un lado, al igual que en otras iniciativas tomadas anteriormente como pueden ser [12] y [13], este proyecto tiene como objetivo principal realizar una investigación exhaustiva de diferentes modelos predictivos que permitan obtener una estimación de trayectorias de recursos de vigilancia y contingencia de desastres naturales. Además, se pretende trabajar en las áreas de big data y machine learning realizando un estudio de las diferentes librerías disponibles para la implementación de dichas áreas.

Por otro lado, se va a realizar un estudio de varios modelos predictivos obteniendo diferentes precisiones para cada caso. Para ello, se entrenarán dichos modelos de una forma similar insertando la misma cantidad de datos. Como el área de predicción es bastante amplia, este proyecto se va a centrar en la predicción de posiciones en base a la trayectoria ya seguida. Esto va a ser posible gracias a la implementación de machine learning ya mencionada anteriormente.

Además, se va a intentar demostrar que la precisión de las predicciones obtenidas será mayor en función de la distancia temporal a la que se quiera predecir. Es decir, los datos obtenidos serán mucho más precisos si son más próximos (temporalmente hablando) a las muestras que ya se tenían. Por ejemplo, en la vida real va a ser más fácil de predecir donde se va a encontrar una persona dentro de un minuto que dentro de 6 horas.

Por último, se va a hacer una implementación del filtro de Kalman para posteriormente poder realizar una comparación con los demás modelos predictivos y así poder observar que forma de predecir es más precisa.

4 Beneficios del proyecto

En este apartado se va a proceder a la exposición de los beneficios que supondría una buena implementación de este trabajo de investigación.

4.1 Económicos

La información obtenida a partir de este proyecto supondrá varias mejoras económicamente hablando.

El saber una posición más precisa, evitará desplegar tropas en exceso ante un altercado. Esto supondrá una reducción de costes en las tropas a desplegar (operarios y vehículos).

Además, sabiendo la trayectoria que va a seguir el fuego de antemano, se van a poder planificar los recursos a utilizar reduciendo así los fondos dedicados para esta sección.

A partir de las mejoras obtenidas se podrá minimizar el tiempo de extinción del incendio. Esto hará que el número de hectáreas afectadas por el fuego decremente y se tenga que invertir una cantidad menor de dinero en repoblar los bosques.

Por último, este trabajo conformará el core predictivo de un sistema de gestión de recursos que también poseerá una parte prescriptiva, en la que se tendrá en cuenta la eficiencia en coste.

4.2 Técnicos

En el caso de los beneficios técnicos, este proyecto proporciona un pequeño avance en el área de predicción así como de machine learning y big data.

En primer lugar, parte de esta investigación podrá ser usada en futuras investigaciones, facilitando el trabajo a quienes quieran implementar un sistema de prevención de incendios.

Además, esta investigación forma parte de un proyecto cuya finalidad es implementar un sistema de predicción de posiciones de tropas de salvamento ante situaciones de emergencia. La parte que engloba este TFG, pretende encontrar la manera más precisa de predecir mediante las diferentes alternativas barajadas, explicadas en apartados

posteriores. Mientras que la otra parte se encarga de la optimización del despliegue de operarios. Para ser más exactos, una vez obtenida la predicción, se decidirá que tipo de tropa desplegar en base a los datos conseguidos. Es decir, la segunda parte se dedicará a la elección del despliegue de tropas por tierra mar y aire.

4.3 Sociales

Por un lado, la implementación de este proyecto supondrá un gran avance para el planeta. Nuestra sociedad cada vez se encuentra más concienciada del impacto de los incendios en el medio ambiente así como del resto de desastres naturales. Esta investigación implica tener que lamentar un número menor de muertes por quemaduras e inhalación de humo en los incendios (ya sea en humanos o animales). Además, la extinción precoz de un incendio implicará la minimización de la deforestación en los bosques.

Por otro lado, este proyecto podrá beneficiar a mejorar el estado de los bosques. Para ello, se podrán diseñar programas exclusivos para el control de los operarios en cada incendio, indicando la predicción de las posiciones en cada instante facilitando de esta manera las labores de salvamento.

5 Estado del Arte

En esta sección se van a tratar con profundidad dos áreas mencionadas a lo largo de este proyecto, big data y machine learning.

Por un lado, el primero se podría definir como el paradigma de la captura, ingesta, almacenamiento y explotación de la información que en esta ocasión servirá para trabajar con una gran cantidad de datos.

Por otro lado, en este caso el área de machine learning hará referencia al conjunto de modelos que sean capaces de capturar patrones en los datos.

5.1 Machine Learning

5.1.1 Introducción

Según [3] se trata de una disciplina científica del ámbito de la Inteligencia Artificial. Dicha disciplina crea sistemas que aprenden automáticamente identificando patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana.

5.1.2 Tipos de Aprendizaje

En esta sección, se expondrán los diferentes tipos de machine learning. Teniendo en cuenta el modo de aprendizaje, existen tres tipos.

En primer lugar, el aprendizaje supervisado ("Supervised Learning"), utiliza una técnica para deducir las predicciones a partir de los datos de entrenamiento. Dichos datos (en general pares de vectores) son un componente con los valores de entrada y un segundo componente con los valores de salida (los valores deseados). Para ser más exactos, la maquina puede aprender a diferenciar a partir de las características introducidas de que tipo es el dato insertado. Por lo tanto, en este tipo de aprendizaje, se introducirán los datos de entrada y salida y machine learning será quien se encargue de encontrar un patrón (la estructura interna de la información).

En segundo lugar, en el aprendizaje no supervisado ("Unsupervised Learning"), los datos no son etiquetados previamente y solo se va a proceder a la introducción de datos de entrada. Es por ello que la maquina tendrá que ser capaz de estructurar los datos. Este tipo de machine learning se utiliza para la reducción de dimensionalidad de los datos, minimizando la perdida de información. Mediante esta reducción, se podrá representar gráficamente las características agrupadas por similitudes. De esta forma, se podrá obtener una mejor compresión.

Por ultimo, el aprendizaje por esfuerzo ("Reinforcement Learning") utiliza una señal de verificación. Es decir, los datos de entrada se obtendrán a través de una retroalimentación del entorno. Si la maquina actúa correctamente esta será recompensada positivamente [4]. Por el contrario, si toma una decisión errónea o perjudicial para la predicción, esta recibirá un castigo que le afectará negativamente. Mediante esta técnica, se podrá mejorar el proceso de decisión obteniendo un buen resultado final.

5.2 Big Data

5.2.1 Introducción

Mediante este término, se hace referencia al gran volumen de datos, tanto estructurados como no estructurados, que inundan los negocios cada día. Este conjunto de datos posee una complejidad bastante grande debido a su variabilidad. Además, la velocidad de crecimiento de dicho conjunto, complica su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales (BBDD). A pesar de esto, no es la cantidad de datos lo que es importante, sino lo que las organizaciones hacen con ellos. Mediante esta gran cantidad de datos, se pueden realizar análisis para obtener ideas que conduzcan a mejores decisiones y movimientos en cualquier ámbito de la sociedad.

Aunque el tamaño utilizado para determinar si un conjunto de datos determinado se considera Big Data no está firmemente definido, la mayoría de los analistas y profesionales actualmente se refieren a conjuntos de datos que van desde 30-50 Terabytes a varios Petabytes.

La naturaleza compleja del Big Data se debe principalmente a la naturaleza no estructurada de gran parte de los datos generados por las tecnologías modernas.

Algunas fuentes generadoras de información pueden ser: RFID, los sensores incorporados en dispositivos, maquinaria, los vehículos, las búsquedas en Internet, las redes sociales como Facebook, ordenadores portátiles, teléfonos móviles, dispositivos GPS y registros de centros de llamadas. En la mayoría de los casos, con el fin de utilizar eficazmente Big Data, se combina dicha información con datos estructurados (normalmente de una base de datos relacional) de una aplicación comercial más convencional, como un ERP o un CRM.

5.2.2 Importancia del Big Data

Lo que hace que este concepto sea tan útil para muchas empresas es el hecho de que proporciona respuestas a muchas preguntas que ni siquiera sabían que tenían. En otras palabras, proporciona un punto de referencia. Con una cantidad tan grande de información, los datos pueden ser moldeados o probados de cualquier manera que la empresa considere adecuada. Al hacerlo, las organizaciones son capaces de identificar los problemas de una forma más comprensible.

La recopilación de grandes cantidades de datos y la búsqueda de tendencias dentro de los datos permite un movimiento mas rápido, eficiente y eficaz, evitando una mayor cantidad de problemas. También permite eliminar las áreas problemáticas antes de que los problemas acaben con sus beneficios o su reputación.

El análisis de Big Data ayuda a las organizaciones a aprovechar sus datos y utilizarlos para identificar nuevas oportunidades. Eso, a su vez, conduce a movimientos de negocios más inteligentes, operaciones más eficientes y mayores ganancias.

5.2.3 Tipo de datos

La categorización de los datos es importante para cualquier proyecto, y en especial cuando vamos a trabajar con grandes volúmenes.

Dos de las categorizaciones más utilizadas en Big Data suelen ser las que relacionan la estructura de los datos y las que dependen del origen de los mismos.

5.2.3.1 Clasificados por Categorías

Según [5], los tipos de datos se suelen organizar en 2 categorías principales, estructurados y no estructurados.

Los datos estructurados, por un lado, pueden ser creados por el sistema de una manera predefinida (registros en tablas, ficheros XML asociados a un esquema, etc.). Por otro lado, pueden ser datos creados de manera indirecta a partir de una acción previa (valoraciones en encuestas). Estos datos también pueden ser experimentales. Por ejemplo, pueden ser generados como parte de pruebas o simulaciones que permitan validar si existe una oportunidad de negocio.

Cabe añadir la importancia de la información estructurada ya que mediante esta se pueden realizar grandes avances en el área de la investigación. Por ejemplo, una de las vías a explorar es el potencial predictivo de la información en redes sociales para la detección temprana de incendios, de igual modo que hay trabajos, incluso empresas, que son capaces de detectar brotes epidémicos analizando redes sociales. Un claro ejemplo de estas empresas, se trata de los proveedores de atención médica. Por ejemplo, los proveedores de atención médica deben estar actualizados sobre la salud pública y los brotes de enfermedades que afectan a sus comunidades para tomar decisiones correctas en el momento adecuado. Esto les ayudaría a ofrecer mejores servicios de manera eficiente. Cabe añadir que la mayoría de los proveedores de atención médica dependen del CDC para estar informados sobre brotes de enfermedades o recibir notificaciones sobre la temporada de gripe [19].

Los datos no estructurados se pueden obtener de varias formas. En primer lugar, se pueden crear a partir del comportamiento de un usuario (aplicaciones de seguimiento de actividad, posición GPS, etc.). En segundo lugar, este tipo de datos se puede generar a partir de las especificaciones de los usuarios, es decir, sus publicaciones en redes sociales, videos reproducidos, búsquedas realizadas en el explorador, etc.

Otra categoría secundaria, se trata de la mezcla de ambas, es decir los datos híbridos o multi – estructurados (por ejemplo, datos de mercados emergentes o meteorológicos).

Ilustración 1: Esquema de diferentes categorías (Fuente: [5])

5.2.3.2 Clasificados por origen

Aunque no exista un criterio único para categorizar los tipos de datos, según [5] es bastante común dividirlos en 5 categorías:

El primer grupo, está formado por los datos que provienen de las webs y redes sociales. Estos, pueden aportar información sobre los clicks en vínculos y elementos. Así como las búsquedas realizadas en Google por un usuario en concreto o las fuentes de datos de varias RRSS (Twitter, Facebook, etc).

El segundo grupo está formado por los datos generados a partir de la comunicación entre máquinas (Las señales GPS, lectura por radiofrecuencia (RFID), otros sensores como pueden ser maquinas expendedoras, cajeros o parquímetros).

El tercer grupo lo forman los datos Biométricos. Es decir, los datos generados a partir del reconocimiento facial, información genética (ADN), etc.

Los datos pertenecientes al cuarto grupo, se trata de información cuyo origen son las personas. Es decir, la información que puede generar una persona en su vida cotidiana, como pueden ser E-mails, registros médicos electrónicos, grabaciones a operadores de atención al cliente, etc.

Para finalizar, el último grupo está formado por los datos que provienen de las transacciones. Estos datos pueden ser registros de facturación (pagos con tarjeta, online) o registros de comunicaciones (llamadas realizadas, mensajes enviados, etc.).

6 Descripción de la Metodología

En esta sección se va a explicar la parte más técnica del proyecto. Dicha parte consiste en realizar el análisis de diferentes modelos de predicción que permita concluir cual de todos ellos es el más preciso.

Además, se detalla cuales han sido los pasos para realizar esta investigación y de donde se ha obtenido la información en la que esta se basa.

Finalmente, se ha seguido la metodología CRISP para el desarrollo de modelos. Se trata del modelo estándar abierto mas usado del proceso que describe los enfoques comunes utilizados para la minería de datos.

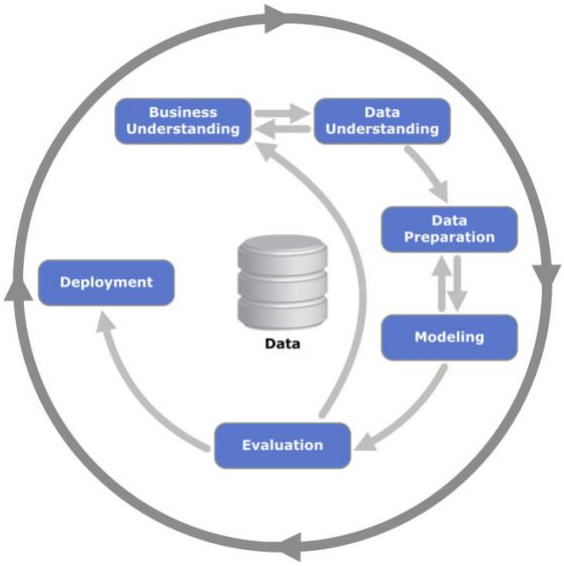

Ilustración 2: Esquema de la metodología CRISP (Fuente: [6])

6.1 Bases de la Investigación

El departamento de incendios regional de Asturias nos ha proporcionado información a partir del SIG obtenida en periodos cortos de tiempo en los incendios producidos durante un año. Estos datos han sido obtenidos a partir del rastro generado por unos dispositivos instalados en las tropas (aviones, drones, bomberos). Un total de 229 diferentes tropas tomaron parte en la recopilación de información (más de 19.000.000 de muestras). Este

proyecto se ha basado en la información proporcionada por estos datasets que proporciona la siguiente información:

- El número de tropa al que pertenece la muestra obtenida.
- El instante relativo en segundos en el que fue tomada dicha muestra.
- La latitud relativa de la tropa en dicho instante.
- La longitud relativa de dicha tropa en el momento de la medida.

1	Time Id Rel_Lat Rel_Long
2	0 N1 0.0 0.0
3	0 N2 12512.8757699 -96047.5584437
4	2 N3 2.41081288522 -59401.8968461
5	2 N4 16379.6301297 -35827.6230075
6	2 N5 12491.811293 -113266.068794
7	3 N6 32183.6147562 2204.83531494
8	5 N7 34635.0409149 -48185.5012857
9	5 N8 10446.1272884 -8203.06615803
10	7 N9 32663.7466266 38600.2975426
11	7 N10 16424.0035682 -35837.7676172
12	9 N11 18793.7430138 -141810.439183
13	9 N12 27873.365668 -32701.6351155

Ilustración 3: Datos del fichero a moldear

6.2 Gestión de la Información

Para la realización de este proyecto se a trabajado en Python ya que posee una gran variedad de librerías destinadas a la predicción. En concreto, se han utilizado las funciones proporcionadas por la librería sk-learn.

6.2.1 Modelado de la Información

En primer lugar, se va a leer el fichero relpositions-meters-112-oneyear.csv obtenido a partir de [7] que contiene la información mencionada anteriormente ordenándola por número de tropa y guardándolas en un nuevo fichero.

En segundo lugar, se va ha observar cuantas veces se repite cada ts y se plasmarán en una gráfica los resultados para facilitar el análisis. Una vez dibujados los resultados, se va a obtener el valor de ts que posee mayor numero de tropas para su posterior análisis.

Finalmente, se van a generar los ficheros que se van a utilizar para observar la aplicabilidad de cada modelo. Para ello, se van a separar los datos en dos partes.

Los datos que forman la primera han sido dedicados para la predicción . Por el contrario, los de la segunda parte han sido destinados para la optimización de los modelos. Para realizar la división de estas partes, se ha tenido en cuenta el instante de tiempo, obtenido anteriormente, con el mayor número de tropas desplegadas. Las muestras que se encuentren en ese instante o a una distancia de ± 3 horas como máximo irán a la sección de optimización. Mientras que el resto de muestras, irán a la sección utilizada para las predicciones.

6.2.2 Predicción

Una vez generados los diferentes ficheros a utilizar para la predicción, se han generado los diferentes modelos. Para ello, primero se han leído los diferentes ficheros que contengan el mismo hórizon (para su posterior uso) y se ha guardado la sección dedicada a la predicción de muestras en diccionarios. Dicha sección se encuentra dividida en dos partes formadas por las muestras de entrada (los datos de partida con los que se va a entrenar el modelo) y las de salida (el resultado que tendría que dar para cada entrada). Además, se han generado varios ficheros con estas secciones pero cada uno con diferente delta y hórizon.

Por un lado, se entiende como delta el numero de muestras a utilizar para obtener una única predicción. Por el otro, se entiende como hórizon la distancia a la que va a estar la muestra obtenida (para ver este moldeado con más detalle ir al **Anexo I**).

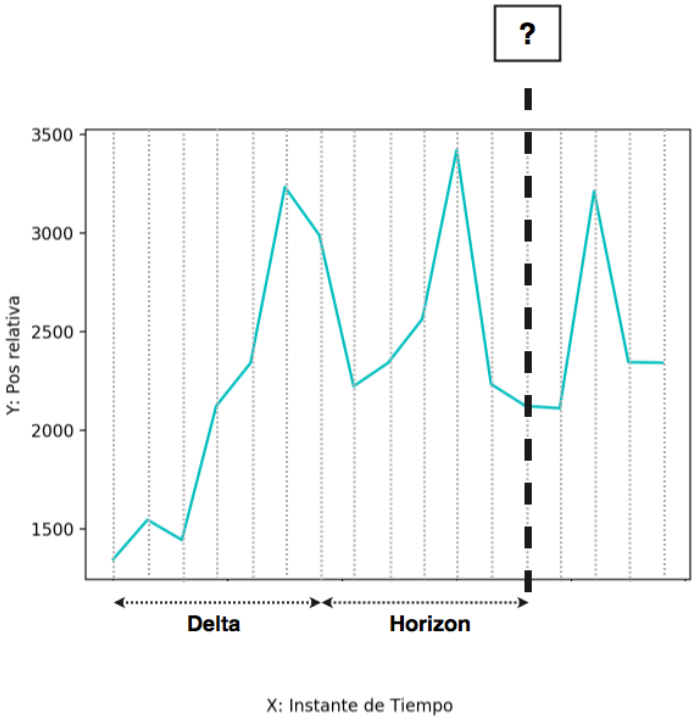

Ilustración 4: Explicación gráfica de Delta y Horizon

Después, se ha moldeado la información para que todas las muestras tengan el mismo peso a la hora de obtener las predicciones. Una vez moldeada, se ha entrenado el modelo con las muestras de la primera parte y después se han obtenido finalmente las predicciones. Para hacer las predicciones más precisas, se ha dividido toda la información en partes diferentes que permita darle cierta redundancia al modelo y así poder tener la fiabilidad de que los datos obtenidos son correctos.

Cabe añadir que el proceso de predicción se ha realizado dos veces ya que los ficheros poseen dos tipos de datos diferentes con los que trabajar, las latitudes y longitudes relativas.

6.2.3 Comparación

Una vez obtenidas todas las predicciones se ha procedido a su comparación. Para ello, se ha calculado la precisión de cada una de los muestras obtenidas mediante la media cuadrática:

$$\text{Media cuadrática} = \sqrt{\frac{(latitud_{testeo} - latitud_{predicha})^2 - (longitud_{testeo} - longitud_{predicha})^2}{nº\_de\_latitudes\_o\_longitudes\_diferentes\_predichas}}$$

Ecuación 1: Ecuación utilizada para la media cuadrática

Con la fórmula anterior, se pretende obtener la diferencia media entre el valor real y el valor predicho. En la siguiente ilustración se muestra un pequeño ejemplo de dicha diferencia representada por la línea roja:

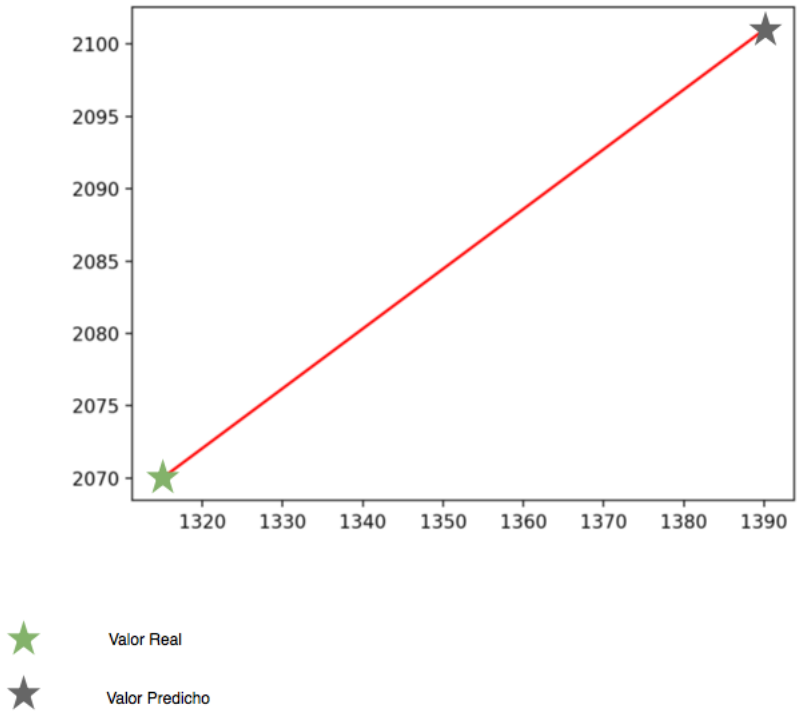

Ilustración 5: Definición gráfica de la media cuadrática

7 Análisis de Alternativas

Por un lado, en esta sección se van a analizar las alternativas de predicción propuestas para su posterior selección. Esta selección se realizará en función de la precisión ofrecida por cada una de estas. Las alternativas a estudiar han sido seleccionadas gracias a la posibilidad de introducir múltiples regresores (en este caso para predecir longitudes y latitudes relativas) [8].

Por otro lado, se van a estudiar diferentes lenguajes de programación para poder realizar los diferentes modelos.

7.1 Identificación de Alternativas

7.1.1 Identificación de modelos

7.1.1.1 Decision Tree Regressor

DT es un método de aprendizaje supervisado no paramétrico utilizado para la clasificación y regresión . El objetivo de este método es crear un modelo que prediga el valor de una variable objetivo mediante el aprendizaje de reglas simples de decisión inferidas a partir de las características de los datos.

En este proyecto, los árboles de decisión aprenderán de los datos obtenidos a partir de las posiciones de una unidad en un incendio para predecir un conjunto de reglas de decisión "if-then-else". Es decir, el algoritmo crea ramificaciones en función de los datos obtenidos. Si el nuevo dato cumple alguna de las condiciones establecida en la rama, este se dirigirá a dicha rama. En caso contrario, este analizará el dato y si es de interés, creará una nueva ramificación con nuevos valores (para futuros datos entrantes). Cuanto más profundo sea el árbol, más compleja será la decisión y más ajustado será el modelo.

Por un lado, implementar este regresór, implica ciertas ventajas:

- Es simple de entender e interpretar. Además, los árboles pueden ser visualizados.
- Mediante este método, no hace falta normalizar los datos para su posterior uso en el algoritmo. Aunque cabe añadir que en este proyecto se ha optado por la normalización de todos los datos.
- Este regresór es capaz de gestionar entradas múltiples como se puede observar en el **Anexo II**. Es decir, se podrán insertar dos tipos de datos diferentes. En el caso

de este trabajo de investigación, esta propiedad es importante ya que se va a trabajar con 2 variables diferentes, por un lado los datos de entrada que hacen referencia a la longitud relativa de una tropa y por otro lado, los que hacen referencia a la latitud relativa.

- Se utiliza un modelo de caja blanca por lo que si una situación dada es observable en un modelo, la explicación de la condición se explica fácilmente por la lógica booleana. Por el contrario, en un modelo de caja negra (por ejemplo, en una red neuronal artificial), los resultados pueden ser más difíciles de interpretar.

- Mediante pruebas estadísticas, existe la posibilidad de validar un modelo. Eso hace posible realzar la fiabilidad del modelo.

- Se desempeña bien incluso si sus suposiciones son un tanto violadas por el modelo verdadero a partir del cual se generaron los datos.

Por otro lado este regresór también posee una serie de desventajas:

- Este método puede generar árboles de decisión demasiado complejos que no generalicen bien los datos. A esta propiedad se le denomina sobreajuste. Para evitar este problema, es necesario contar con mecanismos de control que permitan gestionar el número de ramas máximo a implementar, así como poder establecer el número mínimo de muestras requeridas en un nodo hoja.

- Los árboles de decisión pueden ser inestables ya que pequeñas variaciones en los datos pueden dar como resultado la generación de un árbol completamente diferente.

- Hay conceptos que son difíciles de aprender porque los árboles de decisión no los expresan fácilmente, como los problemas de XOR, paridad o multiplexor.

- Los alumnos del árbol de decisión crean árboles sesgados si dominan algunas clases.

Por último, remarcar que para la implementación de este regresór en el proyecto, no se ha configurado ningún parámetro dentro del algoritmo por lo que su implementación será rápida y sencilla (Para más información sobre la implementación ver **Anexo II**).

7.1.1.2 Nearest Neighbors Regressor

Este tipo de regresór permite facilitar el aprendizaje basado en vecinos supervisados y no supervisados. Los vecinos más cercanos sin supervisión son la base de muchos otros métodos de aprendizaje, especialmente el aprendizaje múltiple y la agrupación espectral.

Este método se basa en la búsqueda de un número predefinido de muestras de entrenamiento más cercanas al nuevo punto para así poder predecir la etiqueta a partir de estos. El número de muestras puede ser una constante definida por el usuario (aprendizaje del vecino más cercano) o variar en función de la densidad local de puntos (aprendizaje vecino basado en el radio). Los métodos basados en vecinos se conocen como métodos de aprendizaje automático no generalizados , ya que simplemente "recuerdan" todos sus datos de entrenamiento.

Este método tiene varias ventajas remarcables:

- Se trata de un regresór simple y fácil de implementar pero los datos obtenidos suelen poseer una gran precisión. Siendo un método no paramétrico, a menudo es exitoso en situaciones de clasificación donde el límite de decisión es muy irregular.

- Esta tecnología puede manejar matrices generadas con la librería Numpy como entrada, por lo que para matrices densas, se podrá admitir una gran cantidad de posibles medidas de distancia.

Por otro lado, este regresór también posee ciertas desventajas:

- Existe la posibilidad de gestionar el número de regresores a tener en cuenta. En caso de ser un valor muy elevado, el cálculo de la predicción podría tener un tiempo bastante elevado.

- Este tipo de tecnología calcula implícitamente de manera ineficiente el límite de decisión. Esta característica puede generar predicciones bastante limitadas y generar predicciones alejadas a la realidad. Se trata de la región de un espacio problemático en el que la etiqueta de salida de un clasificador es ambigua. Esto implica que la transición de una clase en el espacio de características a otra no es discontinua, sino gradual.

Este método calcula la etiqueta de consulta asignada a un punto en concreto según la media de las etiquetas de sus vecinos más cercanos. En este caso se ha decidido tener en cuento los 7 vecinos más cercanos. Además en esta ocasión se ha decidido que los valores más cercanos posean más influencia sobre la predicción (para ello al generar el regresór se ha introducido weights = 'distance'). Por ultimo, el algoritmo utilizado para computar los vecinos cercanos es el que viene por defecto en el regresór (para más información sobre la implementación ver **Anexo III**).

7.1.1.3 Linear Regressor

Este regresór se ajusta a un modelo lineal con coeficientes $\omega = (\omega_1, \ldots, \omega_p)$ para minimizar la suma residual de cuadrados entre las respuestas observadas en el conjunto de datos, y las respuestas predichas por la aproximación lineal. Matemáticamente resolverá un problema de la forma $min_{\omega} ||X_{\omega - y}||_2{}^2$.

Por un lado, este regresór posee varias ventajas:

- Una de las ventajas que tiene el uso de este regresór es que no está afectado por los cambios en las unidades de medida.

- El valor de este regresór será más bajo cuando globalmente los errores para todas las observaciones sean pequeños, algo que resulta deseable para una recta que represente a todos los datos y que pueda utilizarse a la hora de realizar predicciones.

Por otro lado, pueden surgir varias desventajas al implementar este método:

- Las estimaciones de coeficientes para Mínimos cuadrados ordinarios se basan en la independencia de los términos del modelo. Cuando los términos están correlacionados y las columnas de la matriz de diseño X tienen una dependencia lineal aproximada, la matriz de diseño se acerca al singular y como resultado, la estimación de mínimos cuadrados se vuelve altamente sensible a los errores aleatorios en la respuesta observada, produciendo una gran varianza. A esta situación se le llama multicolinearidad.

- A pesar de tratarse de un método sencillo, los resultados obtenidos no son lo suficiente precisos. Esto podría generar un gasto innecesario a la hora de implementar las predicciones en este proyecto.

- Este método calcula la solución de mínimos cuadrados utilizando una descomposición de valor singular de X. Por lo que si X fuera una matriz de tamaño (n, p), este método podría llegar a tener un coste de O(np^2) , asumiendo que n ≥ p .

En esta ocasión, se va a utilizar un modelo bastante sencillo. Como se va a tener que calcular la intersección, la variable fit_intercept será igual a True (ver **Anexo IV** para más información).

7.1.1.4 MLP Regressor

Este Regresór hace uso de un algoritmo de aprendizaje supervisado que aprende una función f(.): $R^m \rightarrow R^o$ entrenando en un conjunto de datos, donde ***m*** es el número de dimensiones para entrada y ***o*** es el número de dimensiones para la salida. Dado un conjunto de características X=$x_1, x_2, \ldots, x_m$ y un objetivo Y , puede aprender una aproximación de una función no lineal para la regresión.

Por un lado, este regresór contiene varias ventajas:

- Posee la capacidad para aprender modelos no lineales.

- Puede aprender modelos en tiempo real usando “partial_fit”.

Por otro, la implementación de este regresór también posee varias desventajas:

- Posee una función de pérdida no convexa donde existe más de un mínimo local. Por lo tanto, diferentes inicializaciones de peso aleatorio pueden llevar a una precisión de validación diferente.

- Este método requiere ajustar varios parámetros como el número de neuronas, capas e iteraciones ocultas.

- Por ultimo, se trata de un regresór sensible al escalado de características.

Cabe destacar que para este método vamos a implementar el algoritmo "adams" internamente para obtener las predicciones. Este algoritmo trabaja bien para cantidad de datos abundantes por lo que será perfecto para este proyecto. Además, la tasa de aprendizaje será adaptativa ya que de esta manera mantendrá constante la tasa de aprendizaje en learning_rate_init. (Para más información ver **Anexo V**).

7.1.1.5 Passive Aggressive Regressor

Este regresór utiliza un algoritmo perteneciente a la familia de algoritmos de aprendizaje en línea (para clasificación y regresión) propuestos por Crammer. Este tipo de algoritmo se utiliza para el aprendizaje a gran escala. La idea es muy simple y se ha demostrado que su rendimiento es superior a muchos otros métodos alternativos.

En la configuración en línea, el algoritmo de aprendizaje observa las instancias de manera secuencial. Después de cada observación, este predecirá un resultado. Este resultado puede ser tan simple como una decisión de sí/no (+/-), como en el caso de los problemas de clasificación binaria y tan complejo como una cadena sobre un alfabeto grande. Una vez realizada la predicción por el algoritmo, este recibe retroalimentación que indica el resultado correcto. Entonces, el algoritmo en línea podrá modificar su mecanismo de predicción, mejorando las posibilidades de hacer una predicción precisa en rondas posteriores. Los algoritmos en línea son típicamente simples de implementar y su análisis a menudo proporciona límites estrictos en su desempeño.

Cabe añadir que para el diseño de este modelo, se ha decidido utilizar los valores de entrada que vienen por defecto (Para más información ir a **Anexo VI**).

7.1.1.6 Random Forest Regressor

Este regresór utiliza técnicas de perturbación y combinación diseñadas específicamente para algoritmos basados en la jerarquía en árbol. Esta metodología crea un conjunto diverso de clasificadores introduciendo aleatoriedad en la construcción del clasificador. La predicción final se obtendrá realizando el promedio de los valores obtenidos individualmente por cada uno de los clasificadores.

Para obtener las mejores predicciones, se hará uso de 2 matrices. La primera, cuyas dimensiones estarán formadas por **nº de muestras x nº de características**, contendrá las

muestras de entrenamiento. Por el contrario, la segunda matriz tendrá tamaño de **número de muestras x 1** y estará formada por los valores objetivo.

La implementación de este regresór proporciona varias ventajas:

- Los resultados obtenidos a partir de este regresór son bastante precisos.
- Esta técnica ofrece un método experimental para la detección de iteraciones en las variables.
- Ofrece un método que permite estimar los datos perdidos y así poder mantener su precisión a la hora de predecir.
- Está diseñado para actuar con gran cantidad de datos.
- Puede manejar cientos de variables de entrada sin excluir ninguna. Esta característica será de gran ayuda para este proyecto ya que se va a trabajar con dos variables diferentes (latitud relativa y longitud relativa).
- Al computar las proximidades entre los pares de casos, este método es capaz de localizar casos atípicos entre sus predicciones. Esto hará que el predictor contemple mas variedad de casos, haciendo que los valores obtenidos sean mucho más precisos.

Al igual que en los casos anteriores, también se pueden observar varias desventajas:

- Al intentar ser lo más preciso posible, este método tiende a sobreajustar los datos en algunos grupos.
- Las clasificaciones obtenidas por el regresór son difíciles de interpretar por el ser humano.

En el caso de este regresór, se ha decidido utilizar 20 decisores diferentes para que obtengan las predicciones consensuadas entre ellos (para más información ver **Anexo VII**).

7.1.1.7 Ridge

El método utilizado por Ridge obtiene predicciones mediante la combinación lineal de la variable de entrada. Además, se aborda algunos de los problemas de los mínimos cuadrados ordinarios al imponer una penalización al tamaño de los coeficientes.

Además, se trata de una forma de crear un modelo parsimonioso cuando el número de variables de predicción exceda el número de observaciones, o cuando un conjunto de datos tenga mutlicolinearidad (correlaciones entre variables de predicción).

Cabe puntualizar que existe otro método similar, en este caso se llama el método de Tikhivov. Se trata de un método que actúa igual que el Ridge pero este podrá producir soluciones bastante precisas aunque los datos posean una gran cantidad de ruido estático, es decir, que las muestras posean una variación inesperada [16].

La implementación de este regresor, implica las siguientes ventajas:

- Define cuando el número de predictores excede el número de observaciones.
- Puede diferenciar los predictores "importantes" de los "menos importantes" en un modelo, por lo que evitará sobreajustar el modelo y podrá encontrar soluciones únicas.
- Es capaz de trabajar con la multicolinearidad en los datos.

Para la implementación de este modelo se ha decidido guardar los valores obtenidos mediante la validación cruzada. Para ello, se tendrá en cuenta el valor "*store_cv_values*" poniéndolo a false. El resto de valores de entrada se quedarán con su valor por defecto al crear el regresór (Para más información ir al **Anexo VIII**).

7.1.1.8 SVR

El método de clasificación de vectores de soporte está diseñado para resolver problemas de regresión. Este método se llama Regresión vectorial de soporte.

El modelo producido por SVR depende solo de un subconjunto de los datos de entrenamiento ya que la función de costo para construir el modelo no se preocupa por los puntos de entrenamiento que están más allá del margen. Cabe añadir que el modelo

producido por SVR depende solo de un subconjunto de los datos de entrenamiento, porque la función de costo para construir el modelo ignora cualquier información de entrenamiento próxima a la predicción del modelo.

Hay tres implementaciones diferentes de Support Vector Regression: Linear SVR , SVR y NuSVR . La primera de ellas proporciona una implementación más rápida que la segunda, pero solo considera núcleos lineales, mientras que NuSVR implementa una formulación ligeramente diferente al resto.

Al igual que con las clases de clasificación, este método tomará como vectores de argumento X e Y, solo que en este caso se espera que Y tenga valores de punto flotante en lugar de valores enteros.

Para la creación se este modelo, se va a optar por que tenga un tamaño de cache bastante grande. Para ser exactos de 500MB (teniendo en cuenta que el valor por defecto es de 200). El resto de valores, serán los que vienen por defecto en el regresór (ver **Anexo IX** para más información).

7.1.1.9 Kalman Filter

Se trata de una técnica que permite el uso de modelos matemáticos a pesar de que las mediciones obtenidas contengan errores.

Es una técnica que hace uso de dos tipos de ecuaciones. Las ecuaciones del primer tipo son las ecuaciones principales. Estas relacionan las variables de estado con variables observables. Las del segundo tipo son las denominadas ecuaciones de estado y se encargan de determinar la estructura temporal de las variables de estado.

El filtro de Kalman será efectivo en caso de que las medidas obtenidas mantengan un ritmo constante. En el caso de este proyecto, las muestras han sido obtenidas cada 30 segundos.

Por ultimo, este algoritmo se ha implementado de la siguiente manera (Para más información ir a **Anexo X**):

1. Se introduce un valor en la maquina y esta predecirá el siguiente.

2. Una vez predicho, el siguiente valor a introducir será comparado con el dato obtenido anteriormente. De esta forma, la maquina irá obteniendo datos cada vez más precisos.

7.1.2 Identificación de lenguajes de programación

7.1.2.1 C

Este lenguaje fue creado para la implementación de sistemas operativos. Además, posee varia librerías de interés para este proyecto [9]:

- **Recommender**

  Las funciones de esta librería son capaces de aprender patrones y predecir los productos más adecuados para una entrada en particular mediante el análisis.

- **Neonrvm**

  Esta librería escrita en C utiliza la técnica RVM para obtener resultados mediante la regresión.

7.1.2.2 Java

Se trata de un lenguaje de programación orientado a objetos cuya sintaxis fue creada a partir de la de C++.

Según [10], Java posee varias librerías para la implementación de machine learning entre las que se encuentran:

- **Weka**

  Se trata de una librería dedicada para el aprendizaje automático la cual es ideal para la extracción y el análisis de datos, así como el modelado predictivo.

  Entre las funciones de la librería dedicada a la minería de datos se encuentran la clasificación, la agrupación, la visualización, la regresión y la selección de características.

- **MOA**

  Se trata de una tecnología de código abierto destinada a flujo de datos en tiempo real. Esta colección de algoritmos es bastante útil para la regresión y determinación de valores atípicos.

- **Mazo**

  Esta librería desarrollada por Andrew McCallum y varios estudiantes posee un código abierto y está destinada para el aprendizaje automático de texto en Java. Esta herramienta, al igual que las dos anteriores, es capaz de extraer información para su posterior moldeado.

7.1.2.3 Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se caracteriza por la facilidad de manejar listas y diccionarios. Además, implementando diferentes librerías, es capaz de manejar una gran cantidad de datos.

Para implementar Machine learning existen varias librerías [11]:

- **Keras**

  Esta librería diseñada para redes neuronales se encuentra escrita en Python. Permite crear prototipos de manera sencilla ya que intenta tener el mínimo retraso a la hora de dar resultados.

- **SK-Learn**

  SK-learn emplea algoritmos de clasificación, regresión y agrupamiento. Además, es capaz de operar simultáneamente con diferentes librerías matemáticas como NumPy.

7.2 Elección de Alternativas

Después de realizar un análisis exhaustivo, se ha decidido seleccionar varios modelos predictivos y un único lenguaje de programación.

7.2.1 Elección de modelos

7.2.1.1 Decision Tree Regressor

Aunque su capacidad de regresión sea muy limitada, este modelo ha sido seleccionado por la capacidad de proporcionar una representación directa e interpretable del conocimiento extraído mediante reglas if then else.

7.2.1.2 Random Forest Regressor

El uso de este regresór supone una mejora al anterior haciendo que no sea solo un modelo en árbol, sino varios que predicen a partir de subsets o vistas del dataset original, por lo que se podrá evitar que el modelo sufra problemas de sobreajuste.

7.2.1.3 Kalman Filter

Esta alternativa proporciona una forma de predecir un poco diferente a la anterior. Por lo que resulta bastante interesante la implementación de este filtro para comparar resultados.

7.2.2 Elección de lenguaje de programación

7.2.2.1 Python

Se ha decidido escoger esta alternativa por la gran variedad de librerías para machine learning que tiene. Para ser exactos, sk-learn proporciona una gran variedad de regresores para la predicción.

8 Planificación

En la siguiente sección se va a exponer la planificación que ha sido necesaria para la realización de este proyecto.

8.1 Ciclo de Vida del Proyecto

Es importante remarcar que ha sido necesario implementar una clasificación modular por etapas para así poder garantizar en mayor medida el éxito del proyecto. De esta manera, será más sencillo detectar y corregir errores en etapas más sencillas del proyecto. Cada fase comenzará cuando la anterior a esta finalice por completo. Este proyecto se ha dividido en 3 fases:

1. **Gestión del trabajo:**

   En esta fase, se van a decidir el director del proyecto, así como la plataforma con la que se van a gestionar todos los datos. Además, al tratarse parte de un proyecto conjunto, se van a dividir las tareas a realizar por cada componente del grupo.

2. **Gestión de las predicciones:**

   En esta parte se van a generar las diferentes alternativas de predicción con diversas librerías destinadas para ello. Una vez realizado este trabajo, se va a proceder a la selección de una o varias alternativas en función de los resultados obtenidos en la fase de pruebas. Para que todo esto sea posible, habrá que dedicar tiempo a familiarizarse con el entorno gráfico, así como con el lenguaje de programación.

3. **Realización de la documentación:**

   En esta parte se va a realizar toda la parte escrita del proyecto. En ella, se tendrá que reflejar todo el trabajo realizado a lo largo de todo el curso académico.

8.2 Diagrama de GANTT

Para identificar y organizar correctamente las tareas expuestas en el apartado anterior, se ha decidido utilizar el diagrama de GANTT. Como esta metodología ha sido diseñada al inicio del proyecto, podría estar expuesta a futuras modificaciones para implementar pequeñas mejoras.

A continuación se muestran las tareas en las que se ha dividido el proyecto, así como sus duraciones y su periodo de realización:

Tabla 1: Información sobre las tareas a realizar

EDT	NOMBRE DE TAREA	DURACIÓN	COMIENZO	FIN
PT-1	**Gestión de trabajo**	**37 días**	**Lun 02/10/17**	**Mar 21/11/17**
T-1.1	Elección de Tema	10 días	Lun 02/10/17	Vie 13/10/17
T-1.2	Definición del alcance del proyecto	15 días	Lun 16/10/17	Vie 03/10/17
T-1.3	Elección de los predictores	12 días	Sab 04/11/17	Lun 20/11/17
H-1	Inicio del proyecto	0 días	Lun 20/11/17	Lun 20/11/17
PT-2	**Gestión de los predictores**	**135 días**	**Mar 21/11/17**	**Lun 28/05/18**
T-2.1	Familiarización con el entorno gráfico y lenguaje de programación	30 días	Mar 21/11/17	Lun 01/01/18
T-2.2	Desarrollo de los predictores	75 días	Mar 02/01/18	Lun 16/04/18
T-2.3	Elección de los predictores	20 días	Mar 17/04/18	Lun 14/05/18
T-2.4	Visualización de los resultados	10 días	Mar 15/05/18	Lun 28/05/18
PT-3	**Realización de la documentación**	**35 días**	**Mar 29/05/18**	**Lun 16/07/18**

T-3.1	Elección de apartados para la memoria	5 días	Mar 29/05/18	Lun 04/06/18
T-3.2	Redacción del documento	30 días	Mar 05/06/18	Lun 16/07/18
H-2	Fin del proyecto	0 días	Lun 16/07/18	Lun 16/07/18

A continuación se reflejan en un diagrama de GANTT las fases descritas en la tabla anterior:

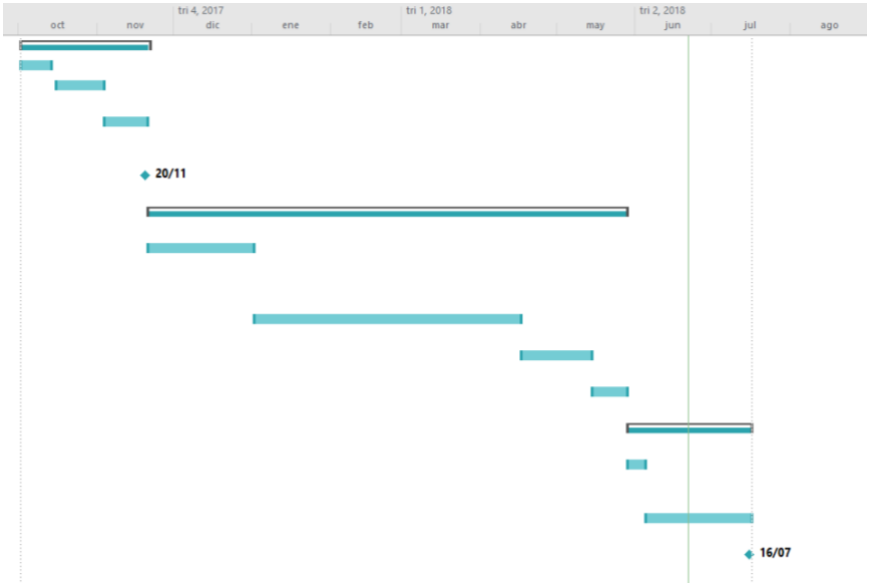

Ilustración 6: Diagrama de GANTT

8.3 Metodología de Desarrollo

Para que el proyecto sea exitoso no será suficiente con implementar la planificación mencionada anteriormente. Será necesario realizar reuniones periódicas con el director del proyecto para cerciorarse de la correcta trayectoria del proyecto. Como este trabajo forma parte de un proyecto mas extenso, las reuniones estarán formadas por los dos ingenieros técnicos que realizan el proyecto íntegramente y por sus respectivos directores.

En primer lugar, las reuniones periódicas se realizarán semanalmente para poder definir una solución lo más precisa y concreta posible. Una vez definidos el alcance y los puntos a desarrollar por cada ingeniero, dichas reuniones se realizaran una vez cada 15 días

aproximadamente para ir viendo los avances implementados por cada parte y corregir errores en caso de ser necesario. Finalmente, se volverán a realizar reuniones semanalmente para afinar la solución y analizar resultados.

9 Presupuesto y Costes

En esta sección, se va a concretar el análisis de los costes y el presupuesto del proyecto. Al tratarse de un proyecto de investigación se va a ha realizar un análisis simple, incluyendo el análisis de coste que supondría la contratación de un Ingeniero técnico en Telecomunicaciones. Para contabilizar las horas a invertir por el ingeniero, se han tenido en cuenta los créditos ECTS correspondientes a este proyecto.

9.1 Horas Internas

Para que este proyecto sea exitoso, ha hecho falta la cooperación de dos ingenieros técnicos, un subdirector y un director.

Tabla 2: Presupuesto de las horas internas

Personal	Número	Horas	Coste por Hora (€)	Coste (€)
Ingeniero Técnico	2	300	50	30000
Subdirector	1	20	50	1000
Director	1	30	55	1650
TOTAL				**32650**

9.2 Amortizaciones

Tabla 3: Presupuesto de las amortizaciones

Concepto	Coste inicial (€)	Vida Útil (h)	Tasa horaria (€/h)	Uso (h)	Coste (€)
Licencia Software de Microsoft Office	149	8760	0.017	80	1.37
Macbook Pro	1500	10100	0.15	300	45
Servidor	3000	43800	0.068	168	11.50
TOTAL					**57.87**

9.3 Gastos

Al tratarse de un lenguaje de programación que requería un aprendizaje externo, se han tenido que usar varios libros de autoayuda.

Tabla 4: Presupuesto de los gastos

Concepto	Gasto (€)
Python for Data Analysis	38.11
Material de Oficina	50
Python for Dummies	19.13
TOTAL	**107.24**

9.4 Gastos Totales

Para calcular el coste final, se va a realizar la suma de las horas internas, amortizaciones y los gastos.

Tabla 5: Presupuesto de los gastos totales

Concepto	Gasto (€)
Horas Internas	32650
Amortizaciones	57.87
Gastos	107.24
TOTAL	**32815.11**

10 Análisis de Riesgos

En la siguiente sección se van a analizar los posibles eventos que pudieran impactar negativamente en el proyecto, haciendo que este se retrase o incluso que no pudiera llegar a finalizar.

Al evaluar un riesgo, se deberá tener en cuenta la probabilidad de que este ocurra, así como el impacto que tendría en nuestro proyecto en caso de que ocurriera. Una vez identificados y evaluados, se realizará un plan de contingencia que permita minimizar el riesgo de que estos ocurran o que su impacto no sea tan grande.

10.1 Identificación de Riesgos

A continuación se nombran los diferentes riesgos que han sido identificados para este proyecto y que podrían afectarle negativamente:

- Error en el presupuesto
- Errores de diseño
- Mala organización de trabajo
- Obtención de resultados poco fiables

10.2 Evaluación de los Riesgos

En esta sección se van a evaluar profundamente los riesgos identificados en la sección anterior.

I. Error en el presupuesto

Se trata de un error bastante común a la hora de planificar un proyecto. Puede que el presupuesto se encuentre mal calculado y que no se hayan tenido en cuenta ciertos gastos, como pueden ser el servidor donde se corren los scripts generados o las licencias de los programas usados.

La probabilidad de que esto ocurra es moderado y el impacto dependerá si se ha superado el presupuesto o no, ya que si el presupuesto ha sido superado, el impacto será alto y no habrá fondos para la correcta continuación del proyecto.

II. Errores de diseño

Esta sección hace referencia a los errores de diseño de los modelos. Un modelo mal diseñado puede tener muchas consecuencias ya que en caso de no detectarlo a tiempo, este generaría predicciones erróneas creando así un retraso importante.

La probabilidad de desarrollar el software erróneamente es alta ya que es muy común realizar errores de programación casi indetectables a primera vista por el ser humano. Además, esto implica un impacto alto ya que habría que volver a realizar las simulaciones durante días.

III. Mala organización de trabajo

Puede ocurrir que la planificación inicial del proyecto sufra modificaciones. Esto es debido a que los plazos preliminares establecidos no concuerden con el tiempo real a invertir en cada tarea.

La probabilidad de que esto ocurra es bastante baja ya que la planificación del proyecto es fruto de un estudio inicial para cada etapa. Este suceso, no tendría un gran impacto en el proyecto ya que bastaría con reorganizar los plazos.

IV. Obtención de resultados poco fiables

En ocasiones, los modelos diseñados pueden generar predicciones poco precisas, a pesar de que estos se encuentren diseñados correctamente. Esto puede hacer que se ponga en duda la aplicabilidad del modelo.

La probabilidad de que esto ocurra es bastante alta y su impacto es bastante elevado ya que se podrían llegar a desechar modelos que son realmente útiles para este proyecto.

10.3 Planes de contingencia

En esta sección se va a realizar un plan de contingencia para cada uno de los riesgos especificados anteriormente. El plan de contingencia a realizar se va a centrar en la prevención y minimización del riesgo. Además, se establecerá un plan alternativo en el caso de que este ocurra.

A continuación se muestra la matriz probabilidad/ impacto para facilitar el estudio de los riesgos. En ella se analizan los riesgos para poder encontrar una solución de una forma sencilla.

Impacto\ Probabilidad	Muy bajo	Bajo	Moderado	Alto	Muy Alto
Muy Bajo	III				
Bajo					
Moderado		I			
Alto				II	
Muy Alto					IV

Ilustración 7: Matriz probabilidad impacto

I. Error en el presupuesto

Para intentar que el presupuesto no suponga un retraso en caso de no cumplirse, se va a incluir un pequeño fondo para imprevistos. De esta manera, en caso de que el presupuesto aumente durante el proyecto, esa diferencia quedará cubierta.

II. Errores de diseño

Para intentar no cometer fallos a la hora de diseñar los modelos, se van a realizar pruebas periódicas. De esta forma se podrá comprobar que se está trabajando correctamente. Una de las pruebas, podría ser, realizar predicciones con una pequeña cantidad de datos, ya que utilizar todas las muestras disponible implicaría una gran cantidad de tiempo.

III. Mala organización de trabajo

Para minimizar este riesgo, se van a realizar reuniones periódicas para ver que se cumple el plan de trabajo inicial. En caso de ser necesario, se reajustará la planificación inicial pará así intentar cumplir con los plazos de entrega establecidos o ajustarse a ellos lo máximo posible.

IV. Obtención de resultados poco fiables

Para evitar desechar modelos erróneamente, se van a realizar varias pruebas con diferentes datos (en cantidades pequeñas). De esta manera, a partir de los diferentes resultados obtenidos se podrá contrastar la fiabilidad de cada modelo.

11 Conclusiones

En este Trabajo de Fin de Grado se ha realizado una investigación exhaustiva de diferentes predictores, realizando un desarrollo software dedicado para la ocasión. Tras su ejecución, se han obtenido varias conclusiones.

En primer lugar, después de realizar una extensa investigación, mediante los resultados obtenidos se ha podido observar que en función del hórizon utilizado, las precisiones de las predicciones cambian. Cuanto más grande sea su valor, menos precisos serán los resultados obtenidos.

En segundo lugar, en la mayoría de los casos, el tamaño de delta será directamente proporcional a la precisión obtenida por cada regresór. Es decir, contra más grande sea la delta utilizada, los valores obtenidos serán más precisos.

En tercer lugar, se han obtenido resultados más precisos con Random Forest Regressor independientemente del hórizon y delta utilizados por lo que se puede concluir que se trata del mejor predictor para este caso.

Además, al trabajar en las áreas de machine learning y big data, se ha podido observar que dichas áreas actualmente se encuentran en auge ya que cada día son más utilizadas por empresas del mundo entero. Es por ello que resulta interesante familiarizarse con dichos conceptos para futuros usos en proyectos similares.

Por ultimo, se ha generado un script que permita visualizar los resultados obtenidos por cada predictor implementado a partir de la librería SK-Learn y así poder obtener las conclusiones mencionadas anteriormente (ver **Anexo XI**). Dichos resultados se pueden observar en las siguientes tablas.

Tabla 6: Resultados obtenidos con hórizon = 1

-REGRESSOR-	Delta = 2	Delta = 10
DecisionTreeRegressor	399.706	400.954
RandomForestRegressor	395.672	381.563

Tabla 7: Resultados obtenidos con hórizon = 4

-REGRESSOR-	Delta = 2	Delta = 10
DecisionTreeRegressor	574.065	576.221
RandomForestRegressor	574.24	553.035

Tabla 8: Resultados obtenidos con hórizon = 8

-REGRESSOR-	Delta = 2	Delta = 10
DecisionTreeRegressor	764.166	760.843
RandomForestRegressor	746.79	734.504

Tabla 9: Resultados obtenidos con hórizon = 12

-REGRESSOR-	Delta = 2	Delta = 10
DecisionTreeRegressor	949.044	949.045
RandomForestRegressor	913.022	897.684

12 Fuentes de Información y Bibliografía

[1] 20 minutos, *Los desastres naturales más importantes de la historia*, 2014. https://listas.20minutos.es/lista/los-desastres-naturales-mas-impactantes-de-la-historia-375009/

[2] Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente, Incendios Forestales en España, Enero 2017.

[3] González A., *¿Qué es Machine Learning?*, 2014. http://cleverdata.io/que-es-machine-learning-big-data/

[4] _Tech Talks, *Machine Learning: Tipos de Machine Learning*, 2017. https://www.raona.com/machine-learning-tipos-machine-learning/

[5] Angel M. Rayo, *Tipos de datos en Big Data: clasificación por categoría y por origen*, 2016. *https://www.bit.es/knowledge-center/tipos-de-datos-en-big-data/*

[6] Monkey Miners, *Learn data mining in a day*, 2018. https://www.monkeyminers.com/our-analytics-process/

[7] Sergio Cabrero, Roberto García, Xabiel G. García, David Melendi, 2016. https://crawdad.org/oviedo/asturies-er/20160808/

[8] http://scikit-learn.org/stable/

[9] Joseph Misiti, 2017. https://github.com/josephmisiti/awesome-machine-learning#c-general-purpose

[10] Jane Elizabeth, *Top 5 machine Learning libraries for Java*, 2017. https://jaxenter.com/top-5-machine-learning-libraries-java-132091.html

[11] Nicole Chapaval, *Cuatro librerías de Machine Learning: TensorFlow, Scikit-learn, Pytorch y Keras*, 2017. https://platzi.com/blog/librerias-de-machine-learning-tensorflow-scikit-learn-pythorch-y-keras/

[12] EFFMIS INTERREG Project, *European Forest Fire Monitoring using Information Systems*, Octubre 2013. http://www.effmis.eu/

[13] FIRESENSE European Project, Fire Detection and Management through a Multi-Sensor Network for the Protection of Cultural Heritage Areas from the Risk of Fire and Extreme Weather Conditions, *FP7-ENV-2009-1-244088-FIRESENSE*, Octubre 2013. http://www.firesense.eu/ (accessed October 2013).

[14] Victoria Police, 2013; Teague et al., 2010.

[15] El Periódico de Aragón, 2015.

[16] http://www.statisticshowto.com/ridge-regression/

[17] Wessel N. van Wieringen, *Lecture notes on RR. Retrieved*, Julio 2017. https://arxiv.org/pdf/1509.09169.pdf

[18] Dorugade y DN Kashid, *Método alternativo para elegir el parámetro Ridge para la regresión*, 2010 . http://www.m-hikari.com/ams/ams-2010/ams-9-12-2010/dorugadeAMS9-12-2010.pdf

[19] Ali Alessal y Miad Faezipour, *A review of influenza detection and prediction through social networking sites*, 2018. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5793414/

13 Anexos

13.1 Anexo I: Modelado de la información

En el presente anexo se incluye el script generado para moldear la información inicial otorgada por el departamento de incendios regional de Asturias.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np


#######################################################################
#Primera Parte
#Pasamos los datos del excel a pos.mat ordenando por el nº de tropa
#######################################################################


Datos_excel = open('relpositions-meters-112-oneyear.csv')
index = 0
pos = {}
for line in Datos_excel:
    if index>0:
        lineParsed = line.strip().split(' ')
        if lineParsed[1] not in pos.keys():
            pos[lineParsed[1]] = []

pos[lineParsed[1]].append([int(lineParsed[0]),float(lineParsed[2]),float(lineParsed[3])
])
    index = index + 1
Datos_excel.close()

from scipy.io import savemat
savemat('pos.mat',pos)


#######################################################################
#Segunda Parte
#Pasamos todos los ts a un array y lo guardamos en un fichero
#######################################################################


from scipy.io import loadmat,savemat
pos=loadmat('pos.mat')
ts=[]
for i in range(1,230):
```

```
        print i
        string = 'N'+str(i)
        tsn = [t[0] for t in pos[string]]
        ts=np.append(ts,tsn)
        ts=ts.tolist()
savemat('tsTropas.mat',{'ts':ts})


#########################################################################
#Tercera Parte
#Obtenemos cuantas veces se repite cada valor de ts
#En unique_elements se guarda el instante de tiempo
#En coounts_elements se guardan cauntas tropas hay en cada instante de tiempo
#########################################################################

from scipy.io import loadmat,savemat
veces = []
tsTropas=loadmat('tsTropas.mat')
ts=tsTropas['ts']
ts=ts.tolist()
unique_elements, counts_elements = np.unique(ts, return_counts=True)
#savemat('Tropas desplegadas.mat',{'unique_elements':
unique_elements,'counts_elements': counts_elements})

#########################################################################
#Cuarta Parte
#Pintamos los resultados obtenidos
#########################################################################


import pylab
pylab.plot(unique_elements,counts_elements,'r')
pylab.xlabel('X: Instante de Tiempo')
pylab.ylabel('Y: Tropas desplegadas')
pylab.title('Tropas desplegadas en cada instante')
pylab.show()

#########################################################################
#Quinta Parte
#Obtengo la posición del maximo de número de tropas desplegadas
#Obtengo el valor del segundo con mas número de tropas desplegadas
#########################################################################

maximo = max(counts_elements)
counts_elements = counts_elements.tolist()
indice = counts_elements.index(maximo)
indiceMAX = unique_elements[indice]
```

```
#########################################################################
#Sexta Parte
#Separamos las partes de train y test
#########################################################################

horas=6 #el número de horas podrá variar
segundos = (horas *3600)/2

from scipy.io import loadmat
pos = loadmat('pos.mat')

DELTA = 2
HORIZON = 12

#1 min->2slots
#5 min->10slots
#10 min->20slots

#Creamos los arrays a utilizar
X = np.array([])
Y = np.array([])
X_test = np.array([])
Y_test = np.array([])

userXY = []
userXY_test = []

XReshaped= np.array([])
YReshaped =  np.array([])
XReshaped_test= np.array([])
YReshaped_test =  np.array([])


for key in pos.keys():

   print(key)
   if 'N' in key and len(pos[key])>DELTA+HORIZON:

      #Parte de Train
      XReshaped =
np.array([list(pos[key][i:i+DELTA,[1,2]].reshape((1,2*DELTA))[0]) for i in
range(0,len(pos[key])-DELTA-HORIZON) if pos[key][i,0] < (indiceMAX -
segundos) or pos[key][i,0] > (indiceMAX+segundos)])
      YReshaped = np.array([list(pos[key][i+DELTA+HORIZON,[1,2]]) for i in
range(0,len(pos[key])-DELTA-HORIZON) if pos[key][i,0] < (indiceMAX -
segundos) or pos[key][i,0] > (indiceMAX+segundos)])
      X = np.vstack((X,XReshaped)) if X.size else XReshaped
      Y = np.vstack((Y,YReshaped)) if Y.size else YReshaped
```

```
        userXY.extend([int(key.strip().replace('N','')) for i in range(len(XReshaped))])

        #Parte de Test
        XReshaped_test =
np.array([list(pos[key][i:i+DELTA,[1,2]].reshape((1,2*DELTA))[0]) for i in
range(0,len(pos[key])-DELTA-HORIZON) if pos[key][i,0] >= (indiceMAX -
segundos) and pos[key][i,0] <= (indiceMAX+segundos)])
        YReshaped_test = np.array([list(pos[key][i+DELTA+HORIZON,[1,2]]) for i in
range(0,len(pos[key])-DELTA-HORIZON) if pos[key][i,0] >= (indiceMAX -
segundos) and pos[key][i,0] <= (indiceMAX+segundos)])
        X_test = np.append(X_test,XReshaped_test)
        Y_test = np.append(Y_test,YReshaped_test)
        userXY_test.extend([int(key.strip().replace('N','')) for i in
range(len(XReshaped_test))])

#Moldeamos los arrays (de 12 y 2 columnas respectivamente)
X_test = X_test.reshape((len(X_test)/(DELTA*2)),(DELTA*2))
Y_test = Y_test.reshape((len(Y_test)/2),2)

#savemat('XYUser2_12.mat',{'X':X, 'Y':Y, 'userXY':userXY,'X_test':X_test,
'Y_test':Y_test, 'userXY_test':userXY_test,'horizon':HORIZON})
#savemat('XYUser10_12.mat',{'X':X, 'Y':Y, 'userXY':userXY,'X_test':X_test,
'Y_test':Y_test, 'userXY_test':userXY_test,'horizon':HORIZON})
```

13.2 Anexo II: Código de Decision Tree Regressor

En el presente anexo se expone el código utilizado para obtener resultados mediante Decision Tree Regressor.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np
from scipy.io import loadmat, savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):

    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']

    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']

    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #X_test = myScaler.transform(X_test)

    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.tree import DecisionTreeRegressor
    regr_multirf = MultiOutputRegressor(DecisionTreeRegressor())

    kf = KFold(n_splits=10)
    scoreskf = []

    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
```

```
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)

    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf) #Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = {}
misResultados.update({'DecisionTreeRegressor' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = {}
scores.update({'DecisionTreeRegressor' : scoreskfALL})
savemat('SCORES_1.mat',scores)


#Guardamos el regresór
joblib.dump(arrayRegresores, 'DecisionTreeRegressor_1.sav')
```

13.3 Anexo III: Código de Nearest Neighbors Regressor

Para obtener predicciones mediante Nearest Neighbors Regressor se ha implementado el siguiente código.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np
from scipy.io import loadmat,savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):

    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']


    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']

    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #X_test = myScaler.transform(X_test)

    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.neighbors import KNeighborsRegressor
    regr_multirf = MultiOutputRegressor(KNeighborsRegressor(n_neighbors = 7,
weights='distance',algorithm='auto', n_jobs = -1))
    #con weights le indico que los valores vecinos le influencien más

    kf = KFold(n_splits= 10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
```

```
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)
    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = loadmat('DATOS_1.mat')
misResultados.update({'KNeighborsRegressor' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = loadmat('SCORES_1.mat')
scores.update({'KNeighborsRegressor' : scoreskfALL})
savemat('SCORES_1.mat',scores)

#Guardamos el regresór
joblib.dump(arrayRegresores, 'KNeighbordsRegressor_1.sav')
```

13.4 Anexo IV: Código de Linear Regressor

En la presente sección se expone el código utilizado para la obtención de predicciones mediante Linear Regressor.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np
from scipy.io import loadmat,savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):

    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']

    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']

    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #X_test = myScaler.transform(X_test)

    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.linear_model import LinearRegression
    regr_multirf = MultiOutputRegressor(LinearRegression(fit_intercept = True,
n_jobs = -1))

    kf = KFold(n_splits=10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
```

```
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)

    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = loadmat('DATOS_1.mat')
misResultados.update({'LinearRegression' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = loadmat('SCORES_1.mat')
scores.update({'LinearRegression' : scoreskfALL})
savemat('SCORES_1.mat',scores)
#Guardamos el regresór
joblib.dump(arrayRegresores, 'LinearRegression_1.sav')
```

13.5 Anexo V: Código de MLP Regressor

Para poder obtener resultados mediante MLP Regressor se ha implementado el código que se expone a continuación.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import numpy as np
from scipy.io import loadmat, savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):

    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']

    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']

    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #X_test = myScaler.transform(X_test)

    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.neural_network import MLPRegressor
    regr_multirf = MultiOutputRegressor( MLPRegressor(solver =
'adam',learning_rate='adaptive',verbose=2,max_iter = 200))
    #El solver que mejor funciona para datos grandes es adam


    kf = KFold(n_splits=10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
```

```
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)

    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = loadmat('DATOS_1.mat')
misResultados.update({'MLPRegressor' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = loadmat('SCORES_1.mat')
scores.update({'MLPRegressor' : scoreskfALL})
savemat('SCORES_1.mat',scores)
#Guardamos el regresór
joblib.dump(arrayRegresores, 'MLPRegressor_1.sav')
```

13.6 Anexo VI: Código de Passive Aggressive Regressor

En este apartado se muestra el código utilizado para el Passive Agressive Regressor,

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import numpy as np
from scipy.io import loadmat, savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []

for l in range (0,2):


    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']

    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']


    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)

    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.linear_model import PassiveAggressiveRegressor
    regr_multirf = MultiOutputRegressor(PassiveAggressiveRegressor(verbose = 2,
fit_intercept=True, C=1))

    kf = KFold(n_splits=10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
```

```
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)

    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = loadmat('DATOS_1.mat')
misResultados.update({'PassiveAggressiveRegressor' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = loadmat('SCORES_1.mat')
scores.update({'PassiveAggressiveRegressor' : scoreskfALL})
savemat('SCORES_1.mat',scores)
#Guardamos el regresór
joblib.dump(arrayRegresores, 'PassiveAggressiveRegressor_1.sav')
```

13.7 Anexo VII: Código de Random Forest Regressor

En el presente anexo se expone el código utilizado para predecir mediante el Random Forest Regressor.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np
from scipy.io import loadmat, savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold
resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):
    print l
    if l == 0:
        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']
    elif l == 1:
        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']
    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.ensemble import RandomForestRegressor
    regr_multirf = MultiOutputRegressor(RandomForestRegressor(n_estimators =
20,verbose=2,n_jobs=-1))
    kf = KFold(n_splits=10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)
    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas
    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
```

```
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)
misResultados = loadmat('DATOS_1.mat')
misResultados.update({'RandomForestRegressor' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)
scores = loadmat('SCORES_1.mat')
scores.update({'RandomForestRegressor' : scoreskfALL})
savemat('SCORES_1.mat',scores)

#Guardamos el regresór
joblib.dump(arrayRegresores, 'RandomForestRegressor_1.sav')
```

13.8 Anexo VIII: Código de Ridge

En este anexo se incluye el código utilizado para implementar de la forma más precisa posible el modelo Ridge.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import numpy as np
from scipy.io import loadmat,savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold
resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):
    if l == 0:
        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']
    elif l == 1:
        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']
    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #Creamos el regresór
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.linear_model import RidgeCV
    regr_multirf = MultiOutputRegressor(RidgeCV(fit_intercept = True,
store_cv_values = True))
    kf = KFold(n_splits=10) #CAMBIAR A 10
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)
    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas
    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
```

```
    arrayRegresores.append(regr_multirf)
misResultados = loadmat('DATOS_1.mat')
misResultados.update({'RidgeCV' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)
scores = loadmat('SCORES_1.mat')
scores.update({'RidgeCV' : scoreskfALL})
savemat('SCORES_1.mat',scores)

#Guardamos el regresór
joblib.dump(arrayRegresores, 'RidgeCV_1.sav')
```

13.9 Anexo IX: Código de SVR

En este anexo se expone el código utilizado para obtener resultados mediante SVR.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import numpy as np
from scipy.io import loadmat, savemat
from sklearn.externals import joblib
from sklearn.model_selection import KFold

resultadosRegresor = []
scoreskfALL = []
arrayRegresores = []
for l in range (0,2):

    if l == 0:

        myDict2 = loadmat('XYUser2_1.mat')
        X = myDict2['X']
        Y = myDict2['Y']


    elif l == 1:

        myDict10 = loadmat('XYUser10_1.mat')
        X = myDict10['X']
        Y = myDict10['Y']


    #Moldeamos los datos
    from sklearn.preprocessing import StandardScaler
    myScaler = StandardScaler()
    X = myScaler.fit_transform(X)
    #X_test = myScaler.transform(X_test)

    #Creamos el regresor
    from sklearn.multioutput import MultiOutputRegressor
    from sklearn.svm import SVR
    regr_multirf = MultiOutputRegressor(SVR(verbose = 2, gamma  = 0.1, C=1,
cache_size=500))


    kf = KFold(n_splits=10)
    scoreskf = []
    for train_index, test_index in kf.split(X):
        print("TRAIN:", train_index, "TEST:", test_index)
```

```
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        regr_multirf.fit(X_train, Y_train)
        Y_pred = regr_multirf.predict(X_test)
        scorefold=(np.sum(np.sqrt((Y_test[:,0]-Y_pred[:,0])**2 + (Y_test[:,1]-
Y_pred[:,1])**2))/len(Y_test))
        scoreskf.append(scorefold)

    scoreskfALL.append(scoreskf)# Guardamos los scoreskf de todas las deltas

    mediascoreskf = np.sum(scoreskf[:])/len(scoreskf)#Hacemos la media del barajeo
    resultadosRegresor.append(mediascoreskf)
    arrayRegresores.append(regr_multirf)

misResultados = loadmat('DATOS_1.mat')
misResultados.update({'SVR' : resultadosRegresor})
savemat('DATOS_1.mat',misResultados)

scores = loadmat('SCORES_1.mat')
scores.update({'SVR' : scoreskfALL})
savemat('SCORES_1.mat',scores)
#Guardamos el regresor
joblib.dump(arrayRegresores, 'SVR_1.sav')
```

13.10 Anexo X: Código de Kalman Filter

En el presente anexo se expone el código utilizado para obtener resultados mediante el Filtro de Kalman explicado anteriormente.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import numpy as np
from scipy.io import loadmat,savemat
from sklearn.externals import joblib

# Implementación del Filtro Kalman para una única variable.
 ########################################################################



# Definimos un Filtro de Kalman Lineal
########################################################################

class KalmanFilterLinear:
  def __init__(self,_A, _B, _H, _x, _P, _Q, _R):
    self.A = _A                  # Matriz de transición del estado
    self.B = _B                  # Matriz de Control.
    self.H = _H                  # Matriz de observación.
    self.current_state_estimate = _x # Estimación inicial del estado.
    self.current_prob_estimate = _P  # Covarianza inicial estimada
    self.Q = _Q                  # Error estimado en el proceso.
    self.R = _R                  # Error estimado en las medidas.
  def GetCurrentState(self):
    return self.current_state_estimate
  def Step(self,control_vector,vector_medida):
    #---------------------------Paso de Predicción----------------------------
    estimacion_estado_predicho = self.A * self.current_state_estimate + self.B *
control_vector
    estimacion_probabilidad_predicha = (self.A * self.current_prob_estimate) *
np.transpose(self.A) + self.Q
    #---------------------------Paso de observación----------------------------
    innovation = vector_medida - self.H*estimacion_estado_predicho
    innovation_covariance =
self.H*estimacion_probabilidad_predicha*np.transpose(self.H) + self.R
    #---------------------------Paso de Actualización--------------------------
    kalman_gain = estimacion_probabilidad_predicha * np.transpose(self.H) *
np.linalg.inv(innovation_covariance)
    self.current_state_estimate = estimacion_estado_predicho + kalman_gain *
innovation
    # Necesitamos el tamaño de la matriz para crear la matriz identidad
```

```
    size = self.current_prob_estimate.shape[0]
    # eye(n) = nxn identity matrix. -->Me crea una matriz identidad con la
    # dimensión indicada (1 en la diagonal principal y 0 el resto de matriz)
    self.current_prob_estimate = (np.eye(size)-
kalman_gain*self.H)*estimacion_probabilidad_predicha


#Inicializamos los datos a introducir en el filtro Kalman
#########################################################################


#misResultados = loadmat('DATOS.mat')
misResultados = {}

myDict10 = loadmat('XYUser10_1.mat')
x = myDict10['X']

numsteps = len(x)

A = np.matrix([1])
H = np.matrix([1])
B = np.matrix([0])
Q = np.matrix([0.00001])
R = np.matrix([0.00001])
xinit = x[0][0]
yinit = x[0][1]
xxhat = np.matrix([xinit])  #Valor Inicial x
yxhat= np.matrix([yinit])   #Valor Inicial y
P    = np.matrix([1])

kalmanx = []
kalmany = []
filtro = []

#Obtenemos el primer valor utilizando el filtro Kalman
#########################################################################

filterx = KalmanFilterLinear(A,B,H,xxhat,P,Q,R)
filtery = KalmanFilterLinear(A,B,H,yxhat,P,Q,R)

#Obtenemos  el resto de valores
#########################################################################

y_test=[]
for i in range(0,numsteps):

    fila = x[i]
    kalmanx.append(filterx.GetCurrentState()[0,0])
```

```
    kalmany.append(filtery.GetCurrentState()[0,0])

    filterx.Step(np.matrix([0]),np.matrix([fila[0]]))
    filtery.Step(np.matrix([0]),np.matrix([fila[1]]))

    filtro = np.append(filtro,kalmanx[-1])
    filtro = np.append(filtro,kalmany[-1])
    filtro = filtro.reshape((len(filtro)/2),2)

    y_test=np.append(y_test,fila[2])
    y_test=np.append(y_test,fila[3])
    y_test=y_test.reshape((len(y_test)/2),2)


media_cuad=(np.sum(np.sqrt((y_test[:,0]-filtro[:,0])**2 + (y_test[:,1]-
filtro[:,1])**2))/len(y_test))

#Guardar los datos
misResultados.update({'FiltroDeKalman' : media_cuad})
savemat('DATOS_Kalman.mat',misResultados)

KalmanFilter = [filterx, filtery]
joblib.dump(KalmanFilter, 'KalmanFilter.sav')
```

13.11 Anexo XI: Código para comparar resultados

Para obtener conclusiones de una manera más sencilla, se ha generado un script que visualice los resultados en forma de tabla. Por un lado, el numero que acompaña al nombre del fichero hace referencia al hórizon utilizado. Por otro lado, cada fichero leído, tiene el valor cuadrático medio obtenido con dos deltas diferentes y tres regresores distintos.

```
from scipy.io import loadmat
from tabulate import tabulate

for i in range(0,4):
   if i==0:
      misResultados = loadmat('DATOS_1.mat')
      print "HORIZON UTILIZADO: 1"

   if i==1:
      misResultados = loadmat('DATOS_4.mat')
      print "HORIZON UTILIZADO: 4"

   if i==2:
      misResultados = loadmat('DATOS_8.mat')
      print "HORIZON UTILIZADO: 8"

   if i==3:
      misResultados = loadmat('DATOS_12.mat')
      print "HORIZON UTILIZADO: 12"

   cabecera = ['-REGRESSOR-','Delta = 2','Delta = 10']
   tabla = []
   tabla.append(cabecera)

   for key in misResultados.keys():

      if '_' not in key:
         datos = []
         datos.append(key)
         datos.append(misResultados[key][0][0])
         datos.append(misResultados[key][0][1])
         tabla.append(datos)

   print(tabulate(tabla,headers = 'firstrow',tablefmt='fancy_grid',
               stralign='center',numalign = "center"))
```