Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de Inteligencia Artificial (IA) para la identificación del tráfico aéreo

Grado en Ingeniería Mecánica

ALUMNO: Adrián Rodríguez Carmona

DIRECTOR: Miguel Rodelgo Lacruz

CURSO ACADÉMICO: 2017-2018

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de Inteligencia Artificial (IA) para la identificación del tráfico aéreo

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

Los avances en el transporte aéreo han producido un inmenso crecimiento en el número de vuelos diarios a nivel mundial, surgiendo la necesidad de sistemas de control e identificación de los mismos. En el presente trabajo se describe el desarrollo de un sistema de inteligencia artificial (IA) que identifica a una aeronave (número de vuelo) a partir de sus posiciones (como las que se pueden obtener mediante radar), con el objetivo de realizar un mejor análisis de los datos recibidos y aumentar la seguridad en el tráfico aéreo. Su desarrollo se ha basado en NuPIC, un motor de inteligencia artificial desarrollado por Numenta. Esta inteligencia artificial emplea una innovadora forma de red neuronal artificial basada en el funcionamiento del neocortex del cerebro y utiliza conceptos de memoria temporal y jerarquía por capas. Las posiciones de las aeronaves se obtienen mediante FlightRadar24, un servicio global de seguimiento de vuelos basado en ADB-S, y posteriormente se convierten a un formato soportado por dicha inteligencia artificial. Los parámetros del modelo se han optimizado para mejorar la identificación en este entorno y se ha implementado una aplicación de identificación que analiza e identifica las aeronaves a través de los modelos aprendidos

PALABRAS CLAVE

Inteligencia Artificial (IA), Numenta, HTM, SDR, Similitud, Índice de anomalía, NuPIC, Codificador geoespacial, Red Neuronal Artificial (RNA)

AGRADECIMIENTOS

Primeramente, agradecer al tutor Don Miguel Rodelgo Lacruz por su ayuda, motivación y corrección durante el desarrollo de este trabajo.

Por otro lado, quería hacer un especial agradecimiento a mi madre Doña Mª Auxiliadora Carmona Regadío por su incansable apoyo y su ayuda. También cabe mencionar a Don Carlos Gutiérrez, Don Rafael Carrillo y Don Julio Poch por su paciencia durante los meses de trabajo.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción y motivación

Uno de los retos más deseados por la humanidad, desde su existencia, ha sido poder volar.

A lo largo de la historia, muchos son los intentos realizados con distintos mecanismos que han permitido conquistar el espacio aéreo. Dentro de todos ellos, destaca el primer avión atribuido a los hermanos Wright [1] (Wilbur y Orville) a principios del siglo XX; lo que desembocaría en una revolución del mundo del transporte, dando lugar a una innovación, desarrollo y evolución en muchos ámbitos como el turístico, el militar, el mercantil, etc.

Desde entonces, el volumen de tráfico aéreo ha crecido exponencialmente, teniendo una especial relevancia el control de los vuelos. Surgió la necesidad de la preparación de personal específico, que permitiera informar a los pilotos de las aeronaves de datos esenciales para su seguridad y trayectoria en tiempo real, tales como: condiciones de las pistas de despegue y aterrizaje, condiciones meteorológicas, trayectorias, etc. El aeródromo de Croydon (Inglaterra) estableció el primer servicio de control aéreo permanente. Hoy en día, se ha desarrollado un servicio denominado AFIS, *Aerodrome Flight Information Service*, disponible para cualquier aeronave que opere dentro de una zona de información de vuelo (*Flight Information Zone*, FIZ) que le permite volar con precisión y seguridad, conociendo los riesgos y pudiendo adoptar medidas para evitarlos de forma inmediata.

Se establecieron los primeros servicios de radionavegación y radiocomunicación en las torres de control de los propios aeropuertos, donde los controladores recibían las posiciones de los aviones a través de los servicios radio. Entonces, según las posiciones que se iban recibiendo, se iban colocando banderas de colores en un mapa con una etiqueta de la hora. Así los controladores viendo el mapa sabían de una forma aproximada dónde se encontraba el avión en ese momento, para, de esta forma, avisar al resto de posibles aviones que pudieran pasar muy cerca.

Basándose en este método de planos y pizarras que contenían las posiciones y las horas estimadas de los pilotos, se desarrolló lo que se conoce como control convencional. Esto evolucionó a las fichas de progresión de vuelos como se puede observar en la Figura 1-1, en las cuales, se apuntaba las últimas órdenes dadas a los aviones, además de los datos de cada uno de ellos [2] y [3].

Figura 1-1 Ficha de progresión en papel (Fuente: [4])

Finalmente, la revolución definitiva en el control aéreo es la invención del radar; el cual, permitía una imagen del espacio aéreo detectando todos los aviones que se movieran en el espacio aéreo que tuviese al alcance del radar, estuvieran o no identificados por los controles convencionales. Esta herramienta, no sólo era un avance para el ámbito civil, sino también para el militar. Por otra parte, la mejora de los sistemas de posicionamiento y de las comunicaciones, permitió el crecimiento del control del espacio aéreo, teniendo una mayor relevancia la identificación de aeronaves.

Con respecto al mundo militar, el avión se ha empleado, además de como transporte de personas y materiales, como una nueva arma de guerra para lanzar misiles, bombas y espionaje. Consiguiendo la capacidad de realizar ataques de una manera mucho más rápida, permitiendo la vigilancia de grandes zonas, o consiguiendo aprovisionar de forma más rápida a largas distancias. Por tanto, surgió la necesidad del control del espacio aéreo como elemento de gran importancia estratégica. Esta vigilancia constante se realiza a través de radares aéreos, que permiten detectar y controlar los contactos en el aire, obteniendo una alerta temprana. De esta forma se podrá detectar aviones no identificados por el control aéreo comercial o militar, esta tecnología cobra especial relevancia en escenarios de guerra, o conflictos bélicos, para poder detectar posibles aeronaves enemigas y prepararse para sus ataques.

Actualmente, la identificación la realiza personal formado que, por medio de la evaluación de los datos, realiza una clasificación de los contactos. Esta información la obtiene de todas las fuentes a su alcance tanto de los diferentes trasmisores que llevan las aeronaves como de las torres de control del espacio aéreo y de sus radares.

A partir de la información obtenida se buscan discrepancias entre los diferentes sensores o comportamientos extraños, o que estén fueran de los rangos preestablecidos de altura de las aerovías. Toda esta información es analizada por personas entrenadas para evaluar sus datos y así poder hacer una identificación.

Sin embargo, la evaluación del gran volumen de datos es compleja e imprecisa, en ocasiones, resultando difícil obtener evidencias precisas para evaluar si el contacto aéreo es un peligro o no. Esto puede llevar a catalogar un avión comercial como un enemigo. Uno de los casos más sonados de la historia es el vuelo 665 de Irán, el cual fue destruido por un lanzamisiles norteamericano. Una incorrecta evaluación de los datos en este escenario llevo a la conclusión de que un avión comercial no identificado era una aeronave de guerra (F-14 iraní). El comandante decidió destruir el avión con el lanzamiento de 2 misiles SM-2ER, derribando el vuelo, con la muerte de 290 personas, incluidas tripulación y pasajeros [5] y [6]. Otro accidente aéreo fue el ocurrido el 17 de julio de 2014 en un Boeing 777 que partió de Ámsterdam con dirección Malasia, desapareció de todos los sistemas de control aéreo cuando sobrevolaba la región de Ucrania, encontrando los restos de la aeronave a cuarenta kilómetros de la frontera con Rusia. En este momento Ucrania era un país que se encontraba

en guerra, aunque con las rutas aéreas supuestamente aseguradas. El derribo de esta aeronave se produjo cuando un misil del bando prorruso en la guerra de Ucrania identificó el avión comercial como un avión de guerra y lo derribó con un misil, con la muerte de las 298 a bordo [7].

Este solo es uno de los ejemplos de catástrofes aéreas producidas por una mala identificación y evaluación de los datos recibidos por los sensores, teniendo consecuencias de extrema gravedad.

Actualmente la tecnología de la inteligencia artificial ha experimentado grandes avances permitiendo el procesado de inmensas cantidades de datos con numerosas aplicaciones, como en los negocios o en Internet, entre otras. El desarrollo y aplicación de estas técnicas de control e identificación de aeronaves permitiría mejoras en los sistemas de control, basados en umbrales fijos y supervisión humana, incrementando la seguridad.

Se llega a la conclusión de que la inteligencia artificial ofrece múltiples ventajas para mejorar el procesamiento de datos, seguridad, y prevención de catástrofes. En este trabajo se propone el desarrollo de un sistema de inteligencia artificial para la identificación del tráfico aéreo.

Este sistema recibirá de una base de datos las posiciones geográficas de las aeronaves (rumbo, velocidad y altura) siendo capaz de identificar de qué tipo de vuelo se trata sin necesidad de otros datos. Actualmente este tipo de sistema podría sustituir a los sistemas convencionales de identificación basados en valores umbrales.

1.2 Objetivos

El objeto general de este trabajo es el desarrollo de una aplicación para identificación de vuelos basada en el sistema de inteligencia artificial NuPIC y estará escrita en lenguaje Python. Recorrerá las posiciones de una aeronave (como las que se pueden obtener mediante radar) e identificará de que vuelo se trata. Para la consecución de este objetivo general se dividirá en una serie de hitos específicos:

- En primer lugar, la obtención y conversión de los datos de las posiciones aéreas a un formato procesado por la inteligencia artificial.
- En segundo lugar, la optimización de los parámetros de la inteligencia artificial para las características específicas de las aeronaves, para mejorar la precisión y minimizar los errores de identificación.
- En tercer lugar, desarrollo de la aplicación utilizando los parámetros optimizados.

1.3 Organización de la memoria

A continuación, se presenta la organización de la memoria, explicando de una forma breve los contenidos de cada sección:

En la sección 1, en la que se encuentra comprendido este apartado, se introduce el tema de este trabajo, así como los objetivos que se pretende alcanzar durante el desarrollo.

En la sección 2 se realiza el desarrollo teórico de la tecnología que se va a utilizar para este proyecto, así como una introducción a sistemas similares.

En la sección 3 se explican los pasos que se han ido siguiendo para la consecución de los objetivos. Desarrollando la instalación de la tecnología usada, la obtención de los datos y su conversión para poder ser procesados por la inteligencia artificial usada, la optimización de los parámetros para los datos aéreos, y por último el desarrollo de la aplicación.

En la sección 4 se representará el análisis de los resultados de las optimizaciones realizadas en la inteligencia artificial y una descripción de las pruebas más características que se han ejecutado.

En la sección 5 se plasman las conclusiones donde se evalúa el cumplimiento de los objetivos al inicio del proyecto. Además de las posibles líneas de desarrollo de esta tecnología.

En la sección 6 se hace referencia a las fuentes bibliográficas y recursos utilizados que apoyan las bases de este trabajo.

La última sección es la de anexos donde se adjunta todo el código que se necesita para el funcionamiento de la aplicación.

2 INTRODUCCIÓN TEÓRICA Y ESTADO DEL ARTE

En esta sección se describe brevemente, el desarrollo de la inteligencia artificial y de sus diferentes aproximaciones, incluyendo las redes neuronales recurrentes con aprendizaje no supervisado, las más apropiadas para esta aplicación.

Posteriormente, se hace una descripción de los algoritmos HTM (*Hierarchical Temporal Memory*) y NuPIC (*Numenta Platform for Intelligent Computing*); respectivamente. El tipo concreto de red y el motor de inteligencia artificial empleados en este trabajo; así como de Numenta, la empresa que los ha desarrollado.

En el tercer apartado, se expone alguno de los sistemas de control del tráfico aéreo similares al que se desarrolla en este trabajo, así como una evaluación de los mismos.

Por último, se describen algunos de los sistemas de obtención de datos aéreos que han servido de referencia para la realización de la aplicación de identificación.

2.1 Inteligencia Artificial (IA)

2.1.1 Definición

El término de Inteligencia Artificial (definido como IA de aquí en adelante) está en continuo cambio según se va produciendo el avance de la tecnología. Según Nils J. Nilsson, la IA es definida como la actividad desarrollada por una máquina inteligente, entendiendo "inteligencia" como la cualidad que permite a una entidad realizar funciones de predicción sobre las variables de su entorno de una forma apropiada [8].

Otra definición, IA es una ciencia que se encarga de crear máquinas con la capacidad de hacer cosas para las cuales se requiere inteligencia, entendiendo "inteligencia" como la habilidad de aprender, entender, resolver problemas y tomar decisiones [9].

Por supuesto, no se puede hablar de la definición de IA sin nombrar a uno de los primeros autores en hacerlo: Alan Turing. Para definir la IA evitó cualquier uso de la semántica, en cambio, estableció lo que hoy se denomina, como Test de Turing [10]. Este test consiste en que un evaluador humano interactúa exclusivamente a través de la escritura con diferentes sujetos desconociendo su condición y con el objetivo de evaluar si las contestaciones recibidas son acordes con una respuesta humana; para ello, el evaluador no tiene ningún contacto previo con el sujeto y realiza las cuestiones sin saber si es humano o no. En ningún caso se evalúa si la respuesta es la solución correcta a la pregunta, sino que la respuesta obtenida sea acorde a la que daría un ser humano. Sin embargo, realmente no se busca una

abstracción y una imprecisión como la humana, sino que se busca, con la IA, llegar a utilizar el estudio del comportamiento para ayudar a la humanidad en los aspectos de su interés.

2.1.2 Desarrollo histórico

Los primeros avances en cuestión a sistemas de IA fueron llevados a cabo por Warren Mc Culloch y Walter Potss en 1943, proponiendo un primer modelo de neurona artificial siendo ésta la primera representación de la actividad cerebral [11].

Posteriormente, en 1950, aparece la figura de Alan Turing realizando un primer paso de definición de IA, ya mencionada, con la invención de un método que permite avaluar si una maquina es, o no, inteligente como un ser humano.

Sin embargo, hasta 1956, en la conferencia realizada en la Universidad de Dartmouth Colllege, ubicada en Nuevo Hampshire (Estados Unidos), no se realizaron unas primeras definiciones de IA donde se reconoció que el pensamiento podía suceder en máquinas, que lograba ser comprendido de manera científica y formal, y el modo más eficiente era utilizando computadoras digitales [10]

En 1963, se creó el programa Analogy de Tom Evas que conseguía resolver los problemas de analogía geométrica aplicados en pruebas de medición de inteligencia de un sistema.

Más tarde, en 1965 apareció el programa Dendral argumentado en la Universidad de Standford de California (EE.UU.) por Edwar Feigenbaum y Robert K. Lindsay. Con este programa lograron mapear la estructura de los productos de químicos orgánicos complejos a partir de las espectrometrías de sus masas. En la misma universidad, corriendo el año 1967, nacía el primer robot móvil inteligente llamado Shakey que se desarrolló a partir de la experiencia del programa Dendral (primer sistema experto[1]), este autómata basado en IA era capaz de recibir instrucciones simples e interpretarlas.

En la década de los setenta hasta los años ochenta, creció el uso de sistemas expertos, como el diseñado por Edward Shortliffe, denominado MYCIN. Este sistema estaba enfocado en el diagnóstico de enfermedades infecciosas, llegando a tener una tasa de acierto superior a la de un médico no especialista en este tipo de dolencias.

En 1986, MCClelland y Rumelhart publicaron el libro: “Procesamiento distribuido en paralelo: investigaciones sobre la microestructura de la cognición” donde se planteó una nueva teoría sobre el aprendizaje basada en los procesos neuronales que tienen lugar en el cerebro, siendo una obra de referencia fundamental en este tema hoy en día [11], [12] y [13].

Finalizando el siglo XX, en 1997, un ordenador construido por la empresa estadounidense IBM, llamado Deep Blue, derrotó al campeón mundial de ajedrez. Se basaba en lo que se denomina conocimiento profundo; que consiste en una serie de algoritmos ideados para el aprendizaje automático, es decir, redes neuronales en cuyo campo esta compañía es una de las empresas más innovadoras hasta el momento. En el año 2011 fue construida una supercomputadora llamada Watson por la misma compañía, y quedó primera en el famoso concurso de Jeopardy de la televisión americana derrotando a los participantes humanos más expertos del momento [14]. En la actualidad AlphaGo Zero es un programa desarrollado por Deepmind que fue capaz de derrotar al campeón del mundo en el antiguo juego chino llamado Go, utilizando un nuevo concepto de aprendizaje, denominado aprendizaje reforzado [15].

[1] Sistema Experto: software que emula el comportamiento de un experto humano en la solución de un problema. Los sistemas expertos funcionan de manera que almacenan conocimientos concretos para un campo determinado y solucionan los problemas

2.1.3 Clasificación Inteligencia Artificial (IA)

La inteligencia artificial abarca numerosas técnicas y aproximaciones, entre las que destacan las tres ramas siguientes:

- Lógica difusa
- Algoritmos[2] genéticos
- Redes neuronales artificiales

2.1.3.1 Lógica difusa

Es una de las ramas de la IA que permite a una computadora analizar datos empíricos en una escala entre lo verdadero y lo falso. Cuando los conceptos se traducen a un contexto lógico se produce una pérdida de su significado completo, lo que puede dar lugar a importantes carencias en la creación, por ejemplo, de sistemas expertos.

Un sistema experto trata de imitar el razonamiento de una persona erudita en un campo concreto de conocimiento. Los sistemas expertos más conocidos son los de diagnóstico en medicina, donde el ingeniero trata de reproducir el razonamiento que sigue el médico; sin embargo, el diagnóstico de una enfermedad y la medicación que se le receta están llenos de razonamiento difuso [11].

Los tres principios que tienen que afrontar los sistemas expertos son:

- Pereza: no es posible establecer todas las variables de un problema tanto por su naturaleza aleatoria como por su dificultad debido al número de las mismas.
- Ignorancia teórica: no existe una forma de listar todos los conceptos a los que se enfrenta un problema porque hay un dominio demasiado grande.
- Ignorancia práctica: en el caso de conocer las variables, muchas veces pueden no ser totalmente verdaderas. Siguiendo con la comparativa anterior, un buen ejemplo es el caso médico donde puede haber síntomas falsos.

Para poder atacar estos conceptos se utiliza la lógica difusa que permite la representación matemática de la incertidumbre y la vaguedad, siendo una herramienta formal para su tratamiento [16].

2.1.3.2 Algoritmos Genéticos

Parten de la teoría de evolución natural como un procedimiento para la optimización de sistemas[3], donde sus operaciones básicas proceden de los conceptos de:

- Selección
- Cruzamiento
- Mutación

Los tres fundamentos de la teoría de la evolución se pueden resumir como, que los organismos proceden de un único ancestro, donde la mutación[4] y el entrecruzamiento[5] permiten la aparición de nuevos caracteres formándose nuevos linajes evolutivos y la herencia permite que las mutaciones que se producen en dichos organismos persistan en futuras generaciones mientras que otras están destinadas a desaparecer.

Los algoritmos genéticos se basan en estos fundamentos junto con los conceptos conocidos de genética donde, en términos generales, los cromosomas se denominan en relación con la IA, como cadenas que contienen diferentes valores. Dentro de estas cadenas, mediante un algoritmo de selección

[2] Algoritmo: conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema

[3] Optimización de sistemas: proceso de seleccionar a partir de un conjunto de alternativas posibles, aquella que mejor satisfaga el o los objetivos propuestos para mejorar el rendimiento de una actividad o proceso, evitando así la pérdida de tiempo, datos o cualquier otro parámetro.

[4] Mutación: Alteración o variación en el código del individuo que produce una modificación en sus características

[5] Entrecruzamiento: Consiste en la combinación de caracteres de dos individuos.

conocido como roulette-wheel (RWS) [17] la probabilidad de elegir a una cadena es directamente proporcional a la función que se considere más importante. Por ello, lo que se obtendrá será los algoritmos que cumplan una función que se quiera optimizar.

El cruzamiento se realiza por medio de la combinación de los algoritmos anteriormente seleccionados, dando lugar a cadenas con mejores características.

En el mundo real, los individuos tienen mutaciones que proporcionan ventajas y otras que no. El mismo proceso se aplica en las cadenas de datos, produciéndose el cruzamiento de las mismas de forma pseudoaleatoria. Así se obtendrán cadenas con deformaciones aleatorias que serán seleccionadas con el algoritmo de selección si éstas favorecen a la función que se desea optimizar [11].

2.1.3.3 Redes Neuronales Artificiales

Este concepto es la base del sistema utilizado para la creación de la aplicación desarrollada en este trabajo, motivo por el cual se expone de forma más detallada en el siguiente punto.

2.1.4 Redes Neuronales Artificiales (RNA)

2.1.4.1 Definición

Las redes neuronales artificiales (en adelante RNA) son sistemas de procesamiento de información no lineales, inspirados en el funcionamiento del sistema nervioso biológico.

Se basan en la unión de unidades de procesamiento simple ligadas por conexiones de pesos[6]. Cada uno de estos procesadores es lo que se denominan neuronas artificiales [13] y [18].

Cada neurona recibe unas entradas de otros nodos[7] o del propio entorno, generando una salida simple escalar. Dicho procesamiento depende, bien de la información local de la unidad de la neurona o de la llegada a través de las conexiones con pesos.

Una red neuronal simple está compuesta por neuronas, las cuales deben entrar en un estado de activación que se obtiene como una salida de cada una de las neuronas. Las conexiones entre las diferentes neuronas, generalmente, están definidas por un peso que determina el efecto de la señal de entrada en la propia unidad de procesamiento, condicionando su estado de activación. Por lo tanto, es necesaria una función que actualice el nivel de los pesos. La mayoría de las redes neuronales están definidas como en la **¡Error! No se encuentra el origen de la referencia.**.

$$\mathrm{y} = f\left(\sum_{k} \omega_k x_k\right)$$

Ecuación 1(Fuente: [9]**)**

Tal que:

x: son las entradas bien de otros nodos o de señales externas.

ω: pesos de las neuronas.

$f(\bullet)$: función no lineal simple.

Dependiendo de la aplicación de distintas f (funciones) como, por ejemplo, la sigmoidal, tangente hiperbólica, escalón, etc. Se obtienen diferentes salidas, por lo tanto, dichas funciones proporcionarán diferentes alternativas de aprendizaje para las redes.

Las RNA se basan en una arquitectura de capas, donde la primera capa es la que recibe los datos de la fuente externa y la última capa es la salida que proporciona valores basados en el aprendizaje. El resto de capas que conforman la topología[8] de la red neuronal se denominan capas ocultas.

[6] Peso: parámetros

[7] Nodo: punto de intersección, conexión o unión de varios elementos que confluyen en el mismo lugar

El objetivo de la RNA es obtener patrones de los datos de entrada para la obtención de valores de salida adecuados, que podrán ser usados en predicciones y evaluaciones, entre otras funciones. Este proceso se denomina aprendizaje; donde la RNA, por medio de la variación de los pesos, va activando o neutralizando las diferentes neuronas para la obtención de las salidas [11].

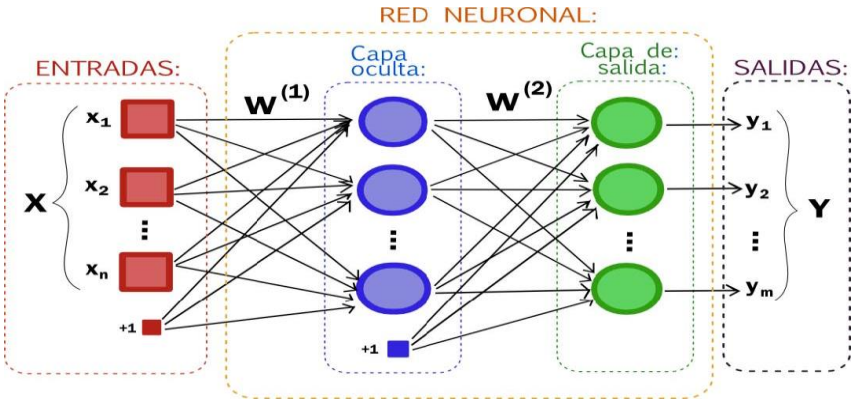


Figura 2-1 Estructura general de una red neuronal (Fuente: [19])

El proceso de aprendizaje se realiza mediante patrones de muestra o entrenamiento, que se utilizan como entradas. Por medio de estos patrones se va ejecutando la red iterativamente, cambiando los pesos de la red hasta que estos convergen a un mismo valor. Entonces, es cuando se interpretará que la red ha aprendido. Todo este proceso es denominado entrenamiento o proceso de accionamiento de la red.

2.1.4.2 Clasificación Redes Neuronales Artificiales

Se pueden distinguir tres tipos diferentes de clasificación de las redes neuronales artificiales según su arquitectura, según el flujo de datos y según el tipo de aprendizaje.

2.1.4.2.1 Según su topología

Se puede realizar una clasificación según el número de capas por las que está compuesta una red neuronal, siempre haciendo referencia a las capas ocultas de la misma. Por tanto, se clasifican en:

- Redes Monocapa: son RNA compuestas por una única capa
- Redes Multicapa: está conformadas por dos o más capas.

2.1.4.2.2 Según el flujo de datos

El flujo de datos de entrada a cada neurona puede circular de forma lineal de una neurona a otra o incluso alimentando a neuronas de capas anteriores [18]. Basado en ello, se puede distinguir:

- Redes Unidireccionales: el flujo de información circula en un único sentido.
- Redes Recurrentes o Realimentadas: la información circula entre las distintas capas e incluso la propia salida de la red alimenta de nuevo a la entrada. Esto permite que, en un momento dado, dependa de las entradas anteriores; es decir, la RNA tiene memoria, lo que es de gran utilidad para el análisis de datos temporales, como es el caso de las trayectorias, que serán el objeto de análisis de este trabajo.

2.1.4.2.3 Según el Mecanismo de Aprendizaje

Uno de los parámetros más relevantes en el aprendizaje de las RNA es la forma de modificar los valores de los pesos.

2.1.4.2.3.1 Aprendizaje Supervisado

[8] Topología: número de elementos de procesado que forman la red y las interconexiones existentes entre ellos

Se caracteriza porque dicho aprendizaje está controlado por un agente externo y experto en la materia. Su función es determinar la veracidad de las respuestas obtenidas por la propia red neuronal a partir de las entradas. En el caso de que no coincida con la salida deseada, se procederá a cambiar los pesos de las conexiones hasta obtener los resultados más adecuados [13] y [20].

Entre los que encontramos tres tipos principales:

- Aprendizaje por corrección de error: según el error que se ha cometido entre el valor deseado de salida y el valor obtenido, se modifica el peso.
- Aprendizaje por refuerzo: en este tipo de aprendizaje supervisado no se tiene una idea exacta de la respuesta que debería dar la red, por lo tanto, se valorará la respuesta asignándole un valor refuerzo en caso de que se opine que ha sido adecuada la salida. En función de estos valores de refuerzo y mediante una función probabilística se ajustan los pesos de la red neuronal.
- Aprendizaje estocástico: se realizan unos cambios aleatorios en la distribución de pesos de la red neuronal y se analizan sus resultados, en función de los objetivos deseados y sus distribuciones de probabilidad.

El principal inconveniente de este método es que requieren datos de entrada etiquetados con las salidas esperadas, lo que, en ocasiones, es complejo de obtener.

2.1.4.2.3.2 Aprendizaje No Supervisado

Las redes no supervisadas o autosupervisadas no requieren de una fuente externa que controle el ajuste de pesos entre sus conexiones y sus salidas, es decir, la RNA no recibe información por parte del entorno de si su salida es correcta.

Este tipo de aprendizaje encuentra correlaciones, características o regularidades que se pueden establecer entre los datos de entrada. Según el algoritmo de aprendizaje no supervisado, las salidas se pueden interpretar como el grado de similitud o anomalía entre los datos de entrada y los datos que se le habían mostrado a la red. Por otro lado, se pueden obtener categorías que ha obtenido la RNA a partir de los datos recibidos y la similitud entre los mismos, estableciendo así diferentes categorías e indicando a cual pertenecen los datos de entrada [20] y [13].

Los algoritmos principales son:

- Aprendizaje Hebbiano: este algoritmo permite saber la familiaridad de los datos que se le proporcionan a la red comparándolos con los anteriormente aprendidos. Siendo este algoritmo el más utilizado para aplicaciones de detección de anomalías.
- Aprendizaje competitivo y comparativo: la red va conformando una clasificación de los datos de entrada, así como clasificando en las diferentes categorizaciones que se van realizando. Por tanto, cuando reconoce una similitud con algunas de las categorías ya establecidas, lo clasifica dentro de la adecuada. En cambio, si la red detecta un nuevo patrón que no se puede establecer en ninguna categoría, forma una nueva clase ajustando los pesos y la estructura de la propia RNA.

La principal ventaja de estos métodos es que utilizan datos de entrada sin etiquetar, mucho más sencillos de obtener, y así pueden continuar aprendiendo durante su funcionamiento normal sin intervención humana en el proceso.

En la aplicación, objeto de este trabajo, se empleará este sistema de aprendizaje no supervisado generando modelos diferentes para etiqueta (número de vuelo), en un esquema de aprendizaje competitivo y comparativo, pero con categorías proporcionadas externamente. De esta manera, el sistema puede clasificar los vuelos en función de su número de vuelos, y al mismo tiempo continuar aprendiendo, para adaptarse a posibles cambios de aerovías, retrasos, rutas, tipos de aeronaves, etc.

2.2 Numenta y HTM (*Hierarchical Temporal Memory*)

Numenta es una empresa compuesta por científicos e ingenieros que trabajan conjuntamente en la investigación de los principios del funcionamiento del cerebro y la aplicación de dichos principios al ámbito de la IA. Su base está situada en California. Ha sido fundada en el años 2005 por Jeff Hawkings, Donna Dubinsk y Dileep George [21].

Numenta explica que la evolución de su empresa se ha compuesto de tres fases [22]:

- Primeramente, han examinado las teorías de la Memoria Jerárquica Temporal (HTM) y desarrollado los primeros algoritmos
- Posteriormente desarrollaron una segunda generación de algoritmos de Memoria Jerárquica Temporal (HTM) con un mayor fundamento biológico y estableciendo nuevos objetivos. Crearon una base de datos pública a la que se puede contribuir.
- La tercera fase y en la que actualmente se encuentran, consiste en realizar una tercera generación de algoritmos, con especial interés en la implementación de la función sensorial motora.

Numenta ejecuta su trabajo realizando ingeniería inversa sobre el funcionamiento del cerebro principalmente en el neocortex, para así crear programas que tengan estos mecanismos como base de funcionamiento [22].

Las investigaciones realizadas por Numenta se encuentran en fuentes abiertas donde se permite el acceso al código de sus proyectos de manera libre, para así contribuir a avanzar en la investigación y desarrollo científico.

2.2.1 HTM (Hierarchical Temporal Memory)

Es una tecnología de aprendizaje automático o de aprendizaje de máquinas que utiliza redes neuronales recurrentes con aprendizaje no supervisado como principios básicos de las teorías de topología y funcionamiento del neocortex.

Este método es la base que se ha escogido como más eficiente para el desarrollo de la optimización y aplicación expuestas en este trabajo.

Este concepto de red HTM se basa, principalmente en la memoria. Consiste en someter a entrenamiento grandes cantidades de datos de los que se obtiene y guarda patrones. En las redes HTM, la organización de la memoria es diferente al funcionamiento convencional en los ordenadores. El uso que hace HTM de la memoria tiene una arquitectura jerárquica, está basada en el concepto del tiempo, y la información se guarda de manera distribuida.

Una red HTM está formada por diferentes regiones, organizadas con una jerarquía[9].

La región es lo que constituye la unidad principal de memoria. A su vez, estas regiones tienen una estructura jerárquica, estas diferentes jerarquías son dominadas niveles.

La jerarquía proporciona eficiencia, debido a que, lo que se aprende en niveles inferiores de la red se reutiliza en niveles superiores. Esto quiere decir, que las regiones inferiores de memoria se utilizan para patrones más generalistas y según se asciende en la red se va especificando para conceptos más definidos.

[9] Jerarquía: forma de organización que se le asignará a diversos elementos de un mismo sistema, ascendente o descendente, por criterios de clase, categoría o cualquier otro de tipo parámetro

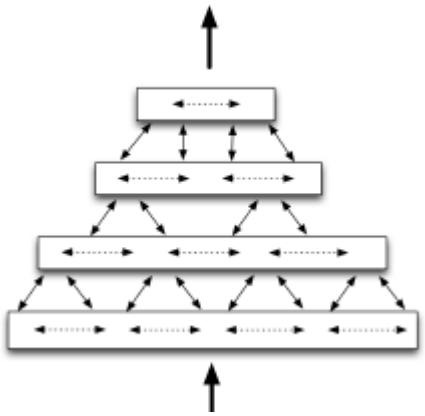


Figura 2-2 Diagrama simplificado de cuatro regiones distribuidas en cuatro niveles de jerarquía (Fuente: [23])

Es importante reseñar que, la cantidad de memoria destinada en cada nivel repercutirá en una representación más o menos compleja de ese nivel. La explicación más sencilla es mediante la simbología con palabras, donde los niveles más bajos representan las letras, el siguiente nivel, las sílabas, etc. y así hasta llegar a las palabras. El concepto de las regiones distribuidas en niveles repercute en un uso más eficiente de la memoria y un entrenamiento más rápido gracias a la generalización.

Este tipo de organización a través de regiones jerarquizadas en niveles de abstracción está basado en el funcionamiento del neocortex, donde las neuronas se encuentran distribuidas en seis capas con una jerarquía similar.

Las redes distribuidas dispersas o SDR (*Spare Distributed Representations*) es otro concepto importante en este tipo de redes. El neocortex está formado por una inmensa cantidad de neuronas donde sólo un pequeño porcentaje de las mismas se encuentran en un estado activo al procesar la información. Cuando se aplica este método a las neuronas artificiales únicamente una parte de ellas estarán activas. Este tipo de representación se denomina representación distribuida dispersa.

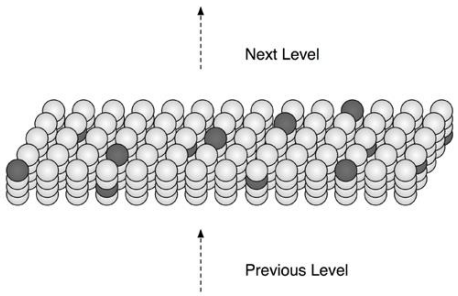


Figura 2-3 Región HTM con escasas células neuronales activas (Fuente: [23])

El concepto de SDR es lo que utilizará la red HTM para poder representar cualquier dato. El funcionamiento consiste en la representación como un pequeño porcentaje de unos (“1”) dentro de una matriz de ceros (“0”). Las entradas se representarán como un porcentaje de bits activos “1” dentro de la matriz. Estos “1” a su vez tendrán posiciones aleatorias en la propia matriz, así se podrá representar cualquier dato como una matriz de ceros y unos. El porcentaje de “1” será lo que se denomina peso y no debería ser superior al 2% del número de elementos de la matriz. Para la representación de distintos datos de entrada con el mismo peso, lo único que se realiza es el cambio de posición de los “1” dentro de la matriz.

El tiempo es un concepto de especial relevancia, ya que cada patrón de la red se forma teniendo en cuenta los patrones previos gracias a la recurrencia de la red. De hecho, la principal actividad del

algoritmo de la red HTM es aprender las secuencias que tienen una variación temporal con respecto a una serie de diferentes entradas.

El aprendizaje de este tipo de regiones se produce mediante la representación de las entradas en SDR (redes distribuidas dispersas) que contienen la información sensorial. En primer lugar, se busca combinaciones de entrada que se produzcan normalmente juntas; esto es lo que se denomina patrones espaciales. Posteriormente, busca en qué secuencia se producen estos patrones espaciales en el tiempo; pasando a llamarse patrones temporales o secuencias.

La red HTM no tiene por qué aprender en una fase de entrenamiento explícito como otro tipo de redes neuronales artificiales (RNA), sino que lo pueden hacer en tiempo real. Cuando ha aprendido la estructura básica de las entradas a la que se le somete durante el aprendizaje, al recibir una entrada nueva que no ha procesado anteriormente, la red tendrá un mayor problema para aprenderla. Cuando una red HTM ha aprendido los patrones de su entorno, cada vez que le introduzcamos una nueva entrada, ésta será modelada como uno de los patrones espaciales y secuenciales anteriormente formados. Esto se denomina inferencia.

La inferencia es el proceso de relacionar un patrón nuevo, en principio desconocido, con otro ya conocido. Durante la inferencia la red forma las representaciones distribuidas dispersas (SDR) con la peculiaridad que no tienen que ser exactamente iguales a las guardadas en las regiones, para que coincida con uno de los patrones guardados.

Gracias al aprendizaje obtienen la posibilidad de predecir el comportamiento del entorno en el que se ha puesto a funcionar el algoritmo HTM. Una vez que las regiones han obtenido los patrones básicos, mapean las nuevas entradas como se describe en el párrafo anterior, y la región forma una predicción de la secuencia que debería venir a continuación.

La mayoría de la memoria en una red HTM se dedica a la memoria secuencial, guardando una cronología de los patrones espaciales. Esto proporciona una serie de características especiales a las redes HTM que se describen a continuación:

- La predicción es continua y se produce en cada nivel del HTM.
- Las predicciones son sensibles al contexto. La memoria le permite tener las secuencias anteriores.
- La predicción proporciona estabilidad porque se realiza en todas las regiones. La predicción puede ser totalmente diferente en el nivel posterior a la entrada, sin embargo, al final la salida puede ser la misma, haciendo mucho más estables las salidas. Ejemplo: las canciones son totalmente diferentes (entrada), la melodía sigue acordes similares utilizando las notas musicales comunes adquiriendo estabilidad (salida).
- Permite convertir el algoritmo HTM en un importante sistema detector de anomalías simplemente comparando la predicción con la entrada.
- La predicción permite ser muy resistente a ruidos[10] ya que, aunque la información no fuese completa se podría corregir para las siguientes entradas.

Según los fundadores de Numenta los datos tienen que cumplir una serie de criterios para que HTM trabaje correctamente:

- Mejor el uso de datos online que información por lotes.
- Los datos deberían tener un patrón temporal.
- Mucha información de diferentes fuentes donde realizar modelos de forma manual es inviable.
- Utilizar datos con patrones que no pueden ser inferidos por el ser humano.

[10] Ruido: datos no relevantes que cambian el resultado final produciendo errores en el mismo.

- Datos que no pueden ser analizados con simples umbrales debido a los falsos positivos y los falsos negativos.

2.2.2 NuPIC (Numenta Platform for Intelligent Computing)

NuPIC es la plataforma utilizada por Numenta para poner a libre disposición el código que implementa los algoritmos de HTM. Contiene el código de los algoritmos que desarrollan aplicaciones de anomalías y predicción. Además, cuenta con un foro de discusión, tutoriales y documentación relevante.

El desarrollo de las aplicaciones basadas en HTM se realiza en numerosos lenguajes de programación (Java, C, C++, etc.) Sin embargo, en la plataforma NuPIC los algoritmos implementados, están en lenguaje Python [24].

En esta plataforma se encontrarán los diferentes algoritmos necesarios para poder llevar a cabo una aplicación de anomalías y predicción basada en HTM. Se puede dividir en fases:

1. Codificar: Es el proceso por el cual los datos sensoriales se transformarán en una representación distribuida dispersa con la que HTM pueda trabajar.
2. Formación de la representación distribuida dispersa (SDR) espacial y temporal: Estas son las representaciones de los datos de entrada en una SDR donde se tiene en cuenta el contexto, es decir, se representa conforme a los patrones anteriormente aprendidos por la red y con los que se puede hacer la predicción temporal.
3. Algoritmo clasificador de aprendizaje cortical: Está es la herramienta cuyo propósito es interpretar la SDR temporal de salida, produciendo una distribución de probabilidad. Aprende una función de la SDR temporal obteniendo una distribución de probabilidad sobre el campo de predicción como se puede observar en la **¡Error! No se encuentra el origen de la referencia.**, dando lugar a las predicciones.

$$f\,(SDR_t) \rightarrow P(PF_{k\pm t})$$

Ecuación 2 (fuente: [25])

4. Grado de anomalía de las entradas: Se expresa el valor de anomalía en un intervalo entre 0 y 1, donde "1" representa una entrada totalmente anómala y "0" como no anómala. Se calcula con la Ecuación **3**.

$$AnomalyScore = \frac{|A_t - (P_{t-1} \cap A_t)|}{|A_t|}$$

Ecuación 3 (fuente: [25]**)**

A_t = Columnas activas durante t

P_{t-1} = Columnas predichas en t

El valor P_{t-1} será utilizado por la aplicación para determinar la probabilidad de que esos datos introducidos se correspondan con el valor aprendido.

2.2.3 Codificador Geoespacial

El codificador geoespacial permite representar coordenadas geoespaciales en tres dimensiones (3D) con representaciones distribuidas dispersas (SDR) para alimentar el algoritmo HTM. Este codificador permite utilizar los algoritmos para detectar anomalías en las rutas, sin necesidad de tener que establecer umbrales como las aplicaciones convencionales de posicionamiento [26].

Las características de este codificador son:

- Las posiciones cercanas tienen un gran número de bits superpuestos.

- La resolución utilizada es dependiente de la velocidad que proporciona la escala de la SDR, cuyo concepto será explicado a continuación:
    - Primeramente, dividimos el espacio en cuadrículas. El tamaño de los cuadros en los que se divide el mapa es lo que se denomina escala. Se puede observar en Figura 2-4.
    - En segundo lugar, para codificar una posición se cogerá una caja que contenga un número de cuadrados. El tamaño de la caja viene definido por lo que denominará radio, el cual es dependiente de la velocidad.
    - Una vez se tiene una caja que contiene un numero de cuadrículas, a cada una de ellas les asignamos un número definido por una *función hash determinista* entre el intervalo 0 y 1. A su vez se utiliza una segunda *función hash determinista* para ordenar los números de bits que se quieran activos en la SDR, obteniendo un vector en forma de una representación SDR como se observa en la Figura 2-4.

La velocidad es uno de los parámetros más relevantes ya que esto permitirá utilizar un radio adecuado, es decir, si tiene una velocidad muy grande se utilizará un cuadrado con un radio mayor. Así se podrá tener bits solapados a pesar de tener dos codificaciones, a priori, separadas en mucha distancia, pudiendo aprender el algoritmo HTM.

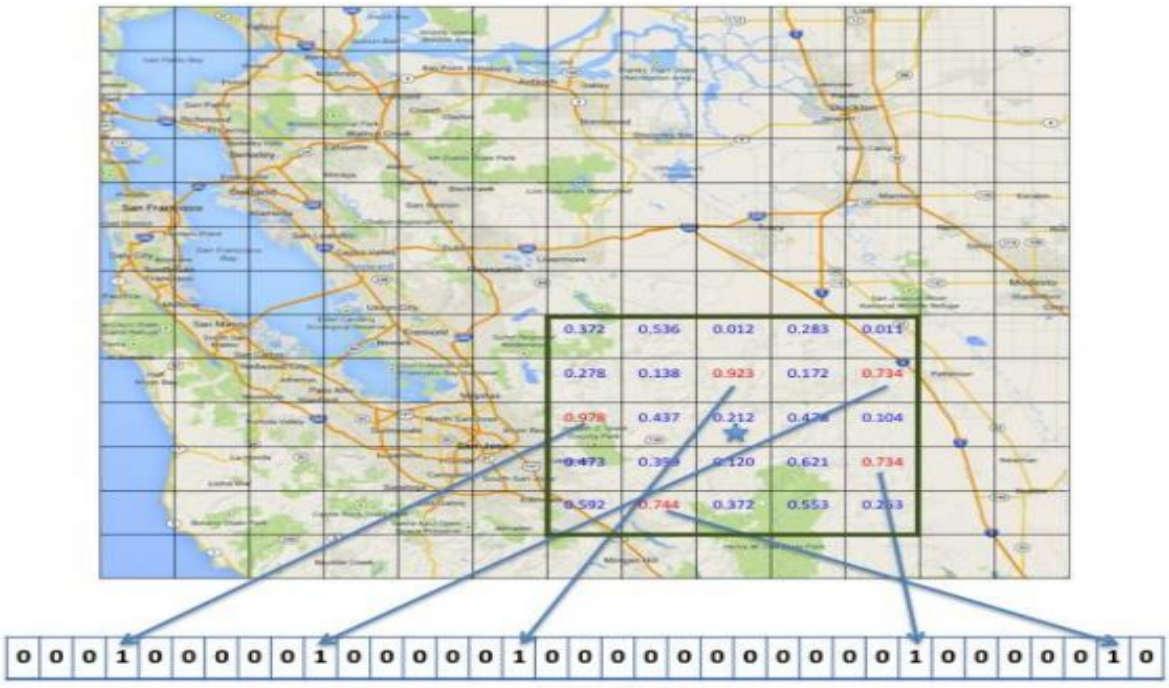


Figura 2-4 Codificación Geoespacial (Fuente [27])

2.3 Aplicaciones para Control del Tráfico Aéreo

En este apartado se describen los sistemas digitales actuales para el control, vigilancia y establecimiento de rutas aéreas tanto en el espacio aéreo como en los propios aeropuertos.

Los sistemas de automatización de las tareas de control aéreo han sido muy debatidos y se ha demostrado que son de especial utilidad cuando hay un gran flujo de aviones; otro de los aspectos relevantes que tienen, es la capacidad de análisis de gran cantidad de datos al mismo tiempo, donde la retención humana es sobrepasada. Sin embargo, todavía no se ha conseguido automatizar por completo debido a la imposibilidad de los sistemas de tomar decisiones, donde la tecnología todavía está muy por detrás de la capacidad humana [28].

A continuación, se presentan los sistemas más destacados en la aviación civil así como algunas aplicaciones en desarrollo [28].

2.3.1 Air Traffic Control Automation System

Es un sistema desarrollado por la empresa Indra que cumple con todos los estándares requeridos por la aviación civil. La misión de este sistema es garantizar la seguridad de los vuelos, representando toda la información aérea procedente de los sensores, así como sus planes de vuelos. Cuenta con un sistema de predicción del tráfico aéreo para que el controlador prevenga posibles retrasos.

Está basado en un sistema operativo Unix/Linux que está programado en lenguaje C++. El sistema dispone de múltiples funcionalidades, ya que recibe datos de todos los sensores-

Las funciones con más relevancia para este trabajo de esta aplicación son [29]:

- Análisis de los planes de vuelo de las aeronaves.
- Predicción.
- Detección de posibles errores de altitud en los planes de vuelo o proximidad entre aeronaves.

2.3.2 SACTA (Sistema Automatizado de Control de Tránsito Aéreo)

El sistema integra todos los centros de control, aproximación de ruta y aeródromos españoles, gestionando toda la información de forma conjunta. Este sistema basado en el sistema operativo Unix con el que obtiene la información de todos los sensores de diferentes aeropuertos y los comparte. Las torres de control son las usuarias de este sistema de aviso y control de aeronaves.

Este sistema proporciona información funcional a los controladores aéreos con las posiciones actualizadas, velocidades y tiempo estimado de llegada. También dispone de un sistema de alerta en caso de desviación del tráfico de la ruta. Sin embargo, no es un diseño de IA donde el sistema vaya aprendiendo de los vuelos y así poder hacer mejores predicciones. De todas formas, este sistema ha conseguido aumentar en gran medida el tráfico aéreo y unificar todos los sistemas de control de aeronaves en un mismo sistema de automatización [30] y [31].

2.3.3 ATAC (Air Traffic Anomaly Detection)

Este sistema experimental, desarrollado en una competición de Numenta, utiliza el algoritmo HTM-MoClu (*Hierarchical Temporal Memory – Model Cluster*), el cual difiere de HTM, en que permite la escalabilidad horizontal gracias al uso de servidores. La finalidad de esta aplicación es la detección de anomalías relacionadas con el posicionamiento de las aeronaves.

Hace uso de dos servidores. El primero de ellos es a partir del cual obtiene todos los datos de los vuelos, mientras que el otro, utiliza un servidor web para guardar los resultados. La representación (Figura 2-5) se realiza con una implementación de Google Maps.

Las dos aplicaciones previstas por el desarrollador son para que los clientes puedan ver en cuales aeropuertos y en qué rutas aéreas se producen una mayor cantidad de anomalías. Esta aplicación está diseñada con el objetivo de informar de las anomalías en tiempo real. Sin embargo, hace referencia a la complejidad de realizar el procesamiento de todos estos datos con los servidores, mostrando como una de sus debilidades: la dependencia de los servidores descritos anteriormente.

Esta aplicación sólo fue desarrollada para el premio de Numenta HTM Talento, galardonada con la primera posición [32] y [33].

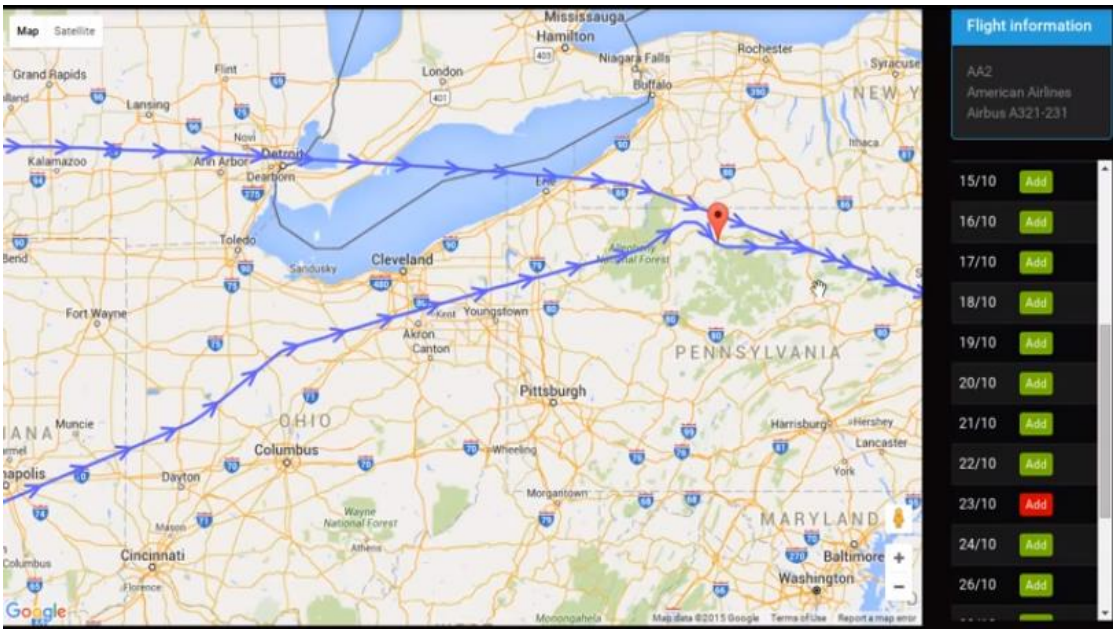


Figura 2-5 Representación gráfica ATAD (Fuente: [32])

2.3.4 ICC (Integrated Command and Control)

Es una herramienta utilizada por la OTAN (Organización del Tratado del Atlántico Norte) que proporciona información de las aeronaves militares pertenecientes a la OTAN, así como todos los vuelos comerciales. Además, se integran todas las trazas aéreas proporcionadas por los sistemas radares de la plataforma en la que esté instalado el sistema.

El ICC permite un reconocimiento de los aviones gracias a la información de las rutas aéreas y la información de los vuelos que van a realizar las aeronaves militares. Este sistema está integrado con el IFF (*Identification Friend or Foe*), el cual, mediante el envío de señales electromagnéticas, identifica aviones tanto militares como civiles. También proporciona información adicional: altura, rumbo, etc. Todo ello depende de los diferentes modos de trasmisión que tengan activados. Dicha información es recopilada por el ICC para poder tener un mayor reconocimiento de los datos aéreos [34] y [35].

2.3.5 Otros trabajos relacionados

Anteriormente se han realizado dos trabajos fin de grado utilizando la tecnología NuPIC como inteligencia artificial. El primero de ellos “Desarrollo de un sistema de inteligencia artificial para la supervisión y detección de anomalías en rutas marítimas” [36] donde realiza un análisis de las anomalías utilizando la tecnología desarrollada por Numenta.

Por otro lado se ha realizó otro trabajo “Estudio y optimización de un sistema de inteligencia artificial para el análisis del tráfico marítimo y aéreo empleando AIS y ADS-B” [37], utilizando la tecnología Nupic, para analizar mediante un servicio online de AIS el tráfico marítimo. Por otra parte, analiza las trazas aéreas obteniendo sus anomalías por medio de un receptor ADS-B sin profundizar en su análisis.

2.4 Sistemas para la obtención de datos aéreos

Existen numerosos sistemas para la obtención de datos.

En este apartado se van a explicar los utilizados por la ejecución del desarrollo de la aplicación, objeto de este proyecto.

Normalmente se utiliza una combinación de numerosos sistemas para la obtención de datos, aunque destacan: IFF (*Identification Friend or Foe*), ADS-B (*Automatic Dependant Surveillance-Broadcast*) y Radar.

Para obtener el posicionamiento mediante sensores se utiliza una técnica que se denomina multilateración o MLAT (multilateration), la cual mide la diferencia de tiempo de llegada de la señal al receptor teniendo en cuenta que las distancias entre las diferentes estaciones son conocidas. Mediante geometría se obtiene la precisión en el posicionamiento o la situación de la aeronave [38].

Existen diferentes servidores que ofrecen servicios de presentación de vuelos online a las aerolíneas para que éstas tengan un control de sus aeronaves. Todas ellas se nutren de los sistemas que se especifican en este apartado para posicionar los vuelos en tiempo real. Entre las páginas web que más destacan en este ámbito son: Flightaware [39], Flightradar24 [40], Planefinder [41], etc. Dichas páginas son evaluadas escogiendo la que proporcione el servicio más adecuado para este trabajo.

2.4.1 IFF (Identification Friend or Foe)

Es un sistema de identificación criptográfica que utiliza el transpondedor de la aeronave. Éste es un equipo que recibe la información en una frecuencia de 1030MHz, y emite en otra distinta de 1090 MHz. Creado en 1940 en Alemania con el objeto de poder distinguir aeronaves o vehículos amigos de los enemigos.

Todavía sigue en uso además de utilizarse para otra información. Estos son las diferentes modos que se encuentran en uso [42]:

- Modo 1 (militar): código de dos cifras las cuales identifican la misión y el tipo de aeronave militar.
- Modo 2 (militar): código de cuatro cifras el cual identifica el número de cola del avión.
- Modo 3/A (militar/civil): este código representa lo que se denomina “squawk”, un número asignado por el controlador aéreo.
- Modo 4 (militar): código cifrado que identifica aviones amigos de enemigos en tiempo de guerra.
- Modo C (civil y militar): código de cuatro cifras que trasmite la altura barométrica de las aeronaves.
- Modo S (civil y militar): código en el que se le pregunta a una aeronave de forma selectiva proporcionando los datos de altura e identificación.
- Modo 5 (militar): código similar al modo S que está encriptado y proporciona posicionamiento.

2.4.2 Automatic Dependant Surveillance- Broadcast (ADS-B)

El ADS-B un sistema donde los aviones emiten su posición periódicamente. Es autónomo, ya que no requiere de un interrogador; es dependiente de la recepción de la posición; sirve para la vigilancia por los datos que proporciona y, por último, de difusión para las estaciones en tierra. Está siendo el sistema con mayor crecimiento como medio de control de aeronaves comerciales [43].

Los aviones obtienen sus posiciones mediante la constelación de satélites GPS (*Global Position System*). El transpondedor de la aeronave transmite una señal que contiene datos de posición, rumbo, velocidad, altura, entre otros; los receptores de tierra reciben la señal, cuyos datos servirán a otras aeronaves y a los controladores. En la imagen esquemática se puede observar el proceso en conjunto Figura 2-6.

Esta tecnología está en desarrollo y no se encuentra instalada en todas las aeronaves activas mundialmente. Se estima que el 80 % de las aeronaves europeas y el 60 % de las norteamericanas están equipadas con estos transpondedores. Para el resto de los países es menor al 20% la implementación de este sistema en sus aviones. Sin embargo, se estima que para el 2020 que será

ampliamente utilizado por la mayoría de las aeronaves. Este transpondedor trabaja en 1090 megahercios de frecuencia lo que le permite alcanzar unas distancias de entre 250-450 km en condiciones normales de propagación [40].

Figura 2-6 Descripción gráfica ADS-B (Fuente: [40])

3 DESARROLLO DEL TFG

En primer lugar, se describe y argumenta la elección de los sistemas utilizados y como se ejecuta su instalación. Posteriormente, se plasma el procedimiento seguido para obtener los datos con las posiciones de los vuelos y se explica cómo se realiza la conversión a un formato soportado por el motor de inteligencia artificial. El siguiente paso, consiste en crear los modelos[11] y optimizar sus parámetros de modo que se obtengan las mejores representaciones a la hora de analizar e identificar los vuelos.

El último apartado describe la aplicación desarrollada que recibe los datos de un vuelo desconocido y lo compara con los modelos previamente aprendidos y configurados con los parámetros óptimos para determinar el grado de semejanza con cada uno de ellos e identificar el más similar.

3.1 Estudio y selección de los sistemas a utilizar

Para el desarrollo de la aplicación se ha seleccionado el algoritmo HTM. Se tratan de un tipo de red neuronal recurrente con aprendizaje no supervisado que, es la más adecuada para la realización de este proyecto tal y como se ha explicado en la sección anterior.

Además, este tipo de IA se adapta, especialmente, a los datos de las aeronaves debido a sus características, tales como:

- Los vuelos tienen un patrón temporal marcado con una duración constante del vuelo en la mayoría de los casos. Además, tienen una periodicidad semanal, con las rutas a las mismas horas y días de la semana.
- La identificación y posicionamiento de las aeronaves se produce en tiempo real. Esta IA es especialmente adecuada para el aprendizaje online.
- Debido a la enorme cantidad de posiciones recibidas, las técnicas simples basadas en umbrales o la inferencia humana son impracticables para el reconocimiento de los vuelos. HTM este diseñado para trabajar con una alta tasa de datos infiriendo los patrones necesarios para la identificación de las aeronaves.

3.1.1 Sistema Operativo

Los desarrolladores de los algoritmos HTM recomiendan el uso del sistema operativo Linux, tanto por ser un sistema operativo libre, como porque la mayor parte de las aplicaciones basadas en este de

[11] Modelo: una representación de los conceptos, las relaciones entre ellos, restricciones, reglas y operaciones que les son aplicables en un dominio específico

tipo IA han sido desarrolladas para este sistema. Por otro lado, la información de NuPIC sobre el desarrollo, tutoriales e instalación está principalmente enfocada a los sistemas Linux.

Se ha instalado el sistema operativo Ubuntu 16.04.3 LTS mediante una partición del disco duro del ordenador. Esto proporciona una forma más cómoda de trabajo, así como una mayor potencia de procesado que empleando la virtualización.

3.2 Instalación del sistema de inteligencia artificial HTM

Previamente a la instalación de los paquetes de HTM se realiza la actualización de los sistemas.

Cabe mencionar que para la instalación del sistema utilizan comandos en la consola de Ubuntu. Para la actualización del sistema una vez ha sido instalado se utilizan los comandos [44]:

```
sudo apt-get install
sudo apt-get upgrade
```

Una vez realizada la actualización e instalación de los paquetes nos dirigimos al repositorio[12] de NuPIC en GitHub ([24]), donde se especifica las dependencias[13] que son necesarias para la instalación:

- Python 2.7
- pip>=8.1.2
- Setuptools>=25.2.0
- Wheel>=0.29.9
- Numpy
- C++ 11 compiler como gcc (4.8+) o Clang

El sistema operativo Ubuntu tiene todas estas dependencias instaladas, como se había señalado excepto el programa "pip", una poderosa herramienta para instalar paquetes en Python que es de utilidad a la hora de instalar las dependencias necesarias para la ejecución de la aplicación. Entonces se procede a la instalación de "pip" mediante el comando [45]:

```
sudo apt-get install python-pip
```

Después se procede a la instalación de la aplicación "git" que nos permite descargar y clonar los repositorios donde se encuentra el código de NuPIC [46].

```
sudo apt-get install git
```

Tras esto, se realiza la instalación de la dependencia "nupic.bindings", añadiendo el. *flag user*.

```
pip install nupic.bingings --user
```

Se comprueba que su instalación ha sido realizada correctamente, con el comando:

```
nupic-bindings-check
```

Comprobando que están todas las dependencias, se realiza la instalación de NuPIC.

```
pip install nupic --user
```

Siguiendo las instrucciones detalladas en el repositorio de NuPIC [24], se ejecutan los tests de comprobación y se utiliza la aplicación "git", anteriormente descargada, para clonar el directorio NuPIC de Github en el equipo. Comando utilizado:

```
git clone https://github.com/numenta/nupic
```

[12] Repositorio: es un espacio que se utiliza para almacenar bases de datos digitales y a sistemas informáticos

[13] Dependencia: es una aplicación o una biblioteca requerida por otro programa informático para poder funcionar correctamente

Una vez descargado, hay que dirigirse al directorio donde se han guardado los archivos de NuPIC y desde allí comprobar que la instalación de todos los paquetes es correcta con el comando:

```
py.test tests/unit
```

Comprobado que todo está correcto y los tests no dan ningún error, se procede a la instalación del repositorio de la aplicación específica que se va a utilizar en este proyecto. Al igual que con los otros directorios se vuelve a hacer uso de la herramienta "git" para la descarga del mismo siguiendo las instrucciones de instalación de NuPIC Geospatial en la propia página web [47].

```
git clone https://github.com/numenta/nupic.geospatial
```

Una vez descargado, se instalan las últimas dependencias del "Pyton" para el funcionamiento de la aplicación en el equipo.

```
pip install -r requirements.txt –user
```

3.3 Obtención y conversión de los datos

Para la realización de este trabajo se han obtenido los datos de los vuelos a través del servicio Flightradar24 [40], que requiere una suscripción. Es uno de los servidores más importantes de monitorización del tráfico aéreo en tiempo real, ya que cuenta con una de las redes más importante de receptores ADS-B. Además, esta compañía trabaja con numerosas aerolíneas como Airbus, Boeing, etc. Lo que le proporciona una gran fiabilidad y experiencia en este campo.

En la toma de decisión de la obtención de datos se estudiaron diferentes alternativas, pero todas ellas fueran descartadas por su poca fiabilidad o por el alto coste del acceso al histórico de datos. Las características por las que se decidió coger este servicio fue:

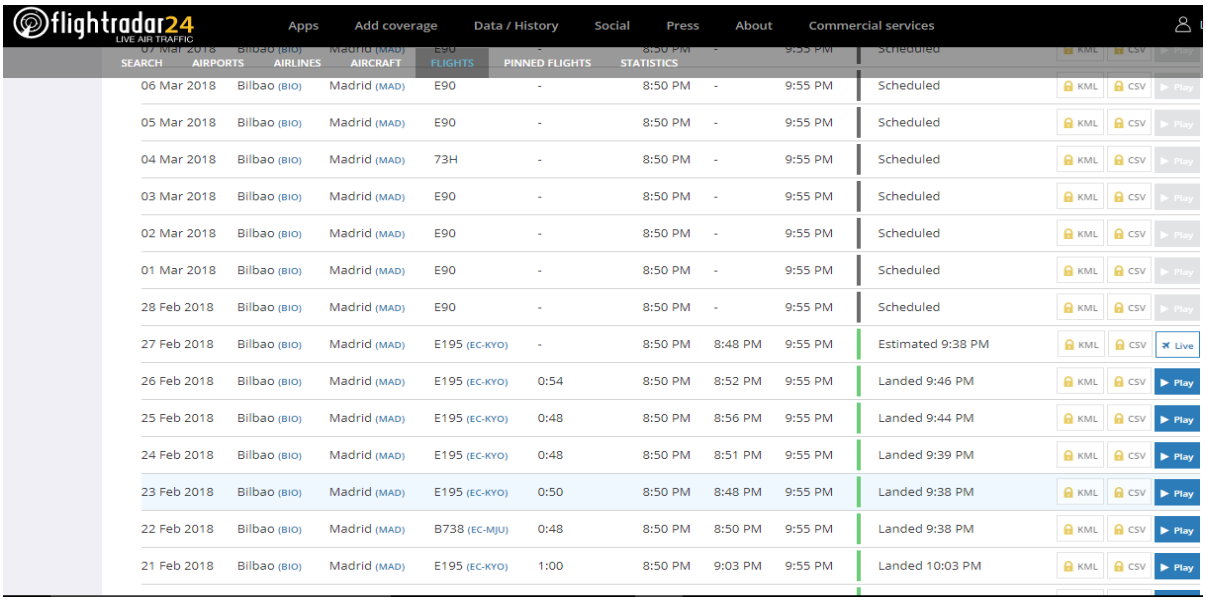
- Proporciona información sobre: vuelo, hora, posición de la aeronave cada cinco segundos, rumbo y velocidad.
- Proporciona el histórico de todos los vuelos.
- El formato del archivo de datos es un CSV[14], soportado por NuPIC.
- El precio de las suscripciones era el más rentable con respecto a los servicios que ofrecen.
- Permite trabajar con los datos de las aeronaves en tiempo real.

El principal inconveniente es que la descarga de los archivos se tiene que realizar individualmente para cada vuelo, elección por la que se opta, ya que la solución que ofrece la página para solventar este problema es enviarle un correo con los vuelos que se necesitan.

Otro de los factores a tener en cuenta es la necesidad de un acceso a Internet para la descarga de los datos del servidor.

La descarga de los datos se realiza de manera intuitiva a través de un buscador en la página web, donde se puede buscar por: número de vuelo, aeropuerto o por aerolínea. En Figura 3-1 se puede observar la página de descarga.

[14] CSV: (*comma-separated values*) son documentos en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.



06 Mar 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
05 Mar 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
04 Mar 2018	Bilbao (BIO)	Madrid (MAD)	73H	-	8:50 PM	-	9:55 PM	Scheduled
03 Mar 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
02 Mar 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
01 Mar 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
28 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E90	-	8:50 PM	-	9:55 PM	Scheduled
27 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	-	8:50 PM	8:48 PM	9:55 PM	Estimated 9:38 PM
26 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	0:54	8:50 PM	8:52 PM	9:55 PM	Landed 9:46 PM
25 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	0:48	8:50 PM	8:56 PM	9:55 PM	Landed 9:44 PM
24 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	0:48	8:50 PM	8:51 PM	9:55 PM	Landed 9:39 PM
23 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	0:50	8:50 PM	8:48 PM	9:55 PM	Landed 9:38 PM
22 Feb 2018	Bilbao (BIO)	Madrid (MAD)	B738 (EC-MJU)	0:48	8:50 PM	8:50 PM	9:55 PM	Landed 9:38 PM
21 Feb 2018	Bilbao (BIO)	Madrid (MAD)	E195 (EC-KYO)	1:00	8:50 PM	9:03 PM	9:55 PM	Landed 10:03 PM

Figura 3-1 Visualización de la Web de descarga de los vuelos (Fuente: [40])

3.3.1 Conversión de los vuelos

La descarga de datos de los diferentes vuelos que ha realizado un avión se tiene que hacer uno a uno para cada vuelo. Se obtendrá un archivo CSV con los datos de: posición, etiqueta de la fecha y hora, número de vuelo (formato hexadecimal), rumbo y velocidad. Por lo tanto, se tiene que realizar una conversión de los datos a un formato más adecuado, así como la unión de cada uno de los archivos en uno único.

Mediante la ejecución de un *script* en Python, denominado "conversor.py" (Para visualizar el código Anexo VIII: conversor.py) se obtiene un archivo CSV de salida como se observa en Figura 3-2 con los datos colocados en el orden adecuado para NuPIC, además de unir todos los diferentes vuelos en un único archivo que se denominará "flight". Para ello, se toma los datos de posición y se dividen en: latitud y longitud; se le añade el número de vuelo del nombre del archivo y se ordena por fecha de vuelo. El conversor se ejecuta mediante el comando:

```
python conversor.py
```

La transformación del archivo como se observa en la Figura 3-2.

etiqueta de tiempo	fecha y hora	Callsign	posición	altura	velocidad	rumbo
1517608157	2018-02-02T21:49:17Z	VLG1707	42.222355,-8.631549	0	14	191
1517608171	2018-02-02T21:49:31Z	VLG1707	42.222095,-8.631618	0	13	191

Número de vuelo	etiqueta de tiempo	latitud	longitud	altura	velocidad	rumbo
0x10547ef0	1517767102	40.632.397	-3.668.763	6800	0	321
0x10547ef0	1517767119	40.639.771	-3.674.346	6974	250	331

Figura 3-2 Modificación de los datos del vuelo.

3.4 Optimización de los parámetros y cálculo del error

Al crear un modelo en NuPIC, es necesario establecer una serie de parámetros para configurar la red y el codificador geoespacial; además de determinar qué datos y codificadores se van a añadir. En este apartado se describe la optimización de estos parámetros para mejorar las prestaciones de identificación de vuelos por NuPIC.

La optimización de los modelos se basa en el valor de la anomalía que proporciona NuPIC. Como se explicó en el apartado 2.2.2, el índice de anomalía determina el porcentaje de parecido entre la predicción de NuPIC y el valor real. Si se realiza una media de las anomalías de todas las posiciones de un vuelo, se podrá determinar cuan parecido es el vuelo real al aprendido anteriormente por la red.

En primer lugar, se establece codificadores adicionales al codificador geoespacial y las variables que se van a optimizar, para mejorar las prestaciones:

- Escala: se variarán sus valores para buscar el valor óptimo para esta aplicación.
- Codificador temporal: se utilizará un codificador horario para que la red aprenda a qué hora del día se produce el vuelo. Para ello se optimizarán dos de los valores del codificador; por una parte el peso del codificador temporal en la SDR y por otra, el radio temporal[15].
- Rumbo: se añadirá un codificador para que la red aprenda de los valores del rumbo que suele tener el vuelo durante sus trayectos.

La optimización del parámetro de escala permite encontrar el mejor valor para este entorno. La componente temporal de un vuelo es esencial para saber si corresponde a uno o a otro, según la hora a la que se haya realizado. El rumbo es un parámetro determinante en la aviación, ya que una vez despegan las aeronaves, siguen el rumbo de la ruta aérea que suele corresponder con un rumbo directo al destino del vuelo.

Una vez establecidas las variables que se van a estudiar, el siguiente paso es definir el método de optimización. Para ello se van a utilizar el *script* de aprendizaje (run_new.py) y el *script* de evaluación (run_load.py). El propósito es realizar el aprendizaje de un número significativo de vuelos de una aeronave concreta y con una ruta específica. Una vez NuPIC haya aprendido de los vuelos, se procederá a la evaluación de un número de vuelos correctos, es decir, vuelos que se sabe que pertenecen a esa ruta y que no deberían presentar anomalías. Por otra parte, se realizará el mismo estudio con vuelos incorrectos, estos son vuelos que no corresponde con el vuelo aprendido para ese trayecto, por tanto, deben ser totalmente anómalos.

El análisis se basa en el cálculo del error, donde se podrán diferenciar dos tipos de errores:

1. Un primer tipo de error está asociado a los vuelos que se han denominado correctos. Dado que estos vuelos se corresponden con el modelo aprendido su anomalía debería ser cero. Por tanto, se determinará el error como la anomalía proporcionada por NuPIC como se puede ver en la Ecuación 4.

$$Error_{vuelos\ correctos}\ (E_{vC}) = |\sum Anomalyscore|$$

Ecuación 4

2. El segundo tipo de error se obtiene de los vuelos incorrectos analizados, ya que deberían ser totalmente anómalos. El error será el porcentaje que le falta para ser totalmente anómalo modelado por la Ecuación 5:

$$Error_{vuelos\ incorrectos}\ (E_{vi}) = |\ 1 - \sum Anomalyscore\ |$$

Ecuación 5

Se realizará el análisis de los parámetros uno por uno, exponiendo sus resultados. Para determinar un buen escenario de aprendizaje, se persigue que un vuelo cruce una zona de numerosas rutas aéreas de modo que comprometa al sistema NuPIC con posiciones muy similares entre vuelos en ciertas zonas del tránsito aéreo. Primero, se procede a una observación cualitativa del mapa aéreo en tiempo real (como se ve en la Figura 3-3). Se puede observar que el punto con mayor número de vuelos es Madrid, debido a que es la capital de España y se encuentra en el centro geográfico del país. Por ello,

[15] Radio temporal: Es el rango numérico por el cual la red toma un valor temporal.

se decide seleccionar un vuelo que cruce de un punto a otro de España. Además, ha de ser un vuelo no demasiado largo para ser procesado en las pruebas en poco tiempo. Por otro lado, es importante que haya aeropuertos cercanos con vuelos que hagan rutas similares para que NuPIC tenga posiciones comprometidas al analizar vuelos incorrectos.

Figura 3-3 Imagen del tráfico aéreo (fuente: [40])

Por todas estas características se decide tomar el vuelo Alicante – Santiago (FR8538) mostrado en la Figura 3-4, ya que cumple todo lo descrito en el párrafo anterior. Este vuelo cruza España de extremo a extremo, sobrevolando Madrid y se encuentra cerca de otros aeropuertos como el de Palma de Mallorca – Santiago, que realiza un recorrido muy similar.

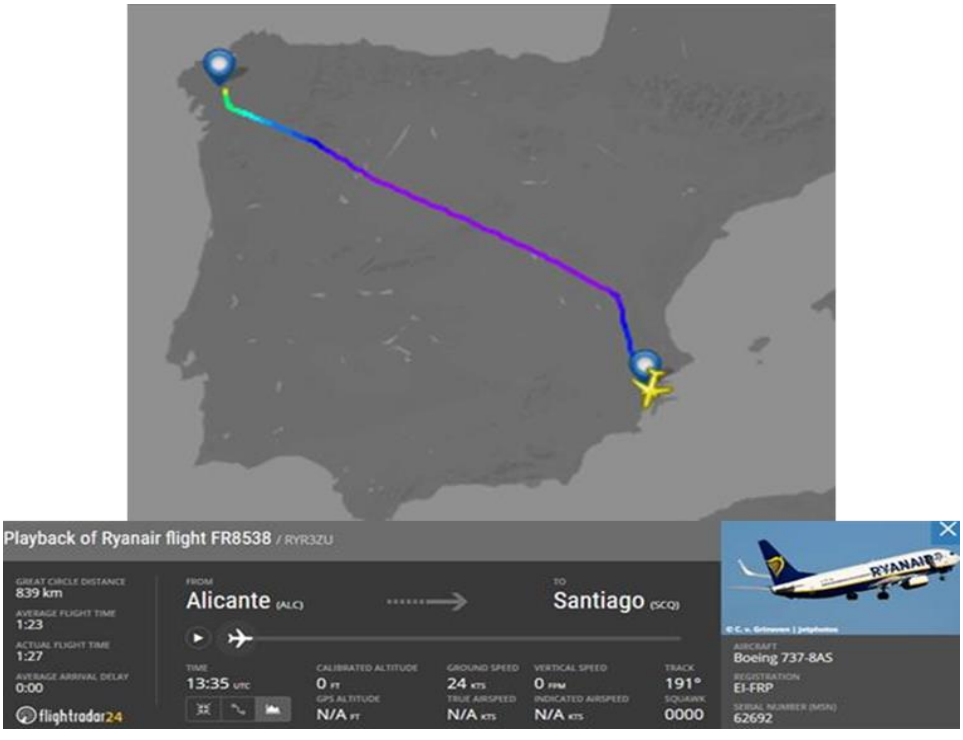


Figura 3-4 Vuelo FR8538 (Fuente: [40])

Tras la selección del vuelo con el que optimizar los modelos, se decide realizar un aprendizaje de 150 vuelos que proporcionan 22861 posiciones ya que son suficientes para estabilizar el aprendizaje.

El análisis de los diferentes modelos aprendidos, se realiza con 50 vuelos de cada tipo, tanto de "vuelos correctos" como de "vuelos incorrectos". En la siguiente Tabla 3-1 se observa los vuelos que se han tomado para analizar el E_{vi}.

Vuelos	Origen	Destino	Duración media
FR723	Palma de Mallorca	Santiago	1h 40min
FR5317	Madrid	Santiago	55 min
FR8538	Santiago	Alicante	1h 23 min
IB8390	Madrid	Alicante	45 min
UX7300	Madrid	Vigo	55min

Tabla 3-1 Datos de "vuelos incorrectos"

3.4.1 Escala

El primer parámetro que se decide optimizar es la escala. Se evalúan valores de 1000 metros (m), 2000 m, 3000 m, 4000 m, y 5000 m. En la Figura 3-5 se representa como desciende el índice de anomalía para cada valor de la escala a medida que aumenta el número de muestras de los vuelos Alicante-Santiago. Se puede observar que el índice de anomalía se estabiliza antes de alcanzar el número de vuelos total.

Por otra parte, se observar que a medida que aumenta la escala aumenta la velocidad de aprendizaje y desciende el valor de anomalía ya que las posiciones se mapean en una cuadrícula más grande, de modo que los vuelos se parecen más entre sí y el patrón secuencial es más sencillo. No obstante, si la escala es demasiado grande, la aplicación no va a poder distinguir vuelos similares de modo que cuando se mueva la aeronave en la zona del aeropuerto le va a costar diferenciar entre unos vuelos y otros.

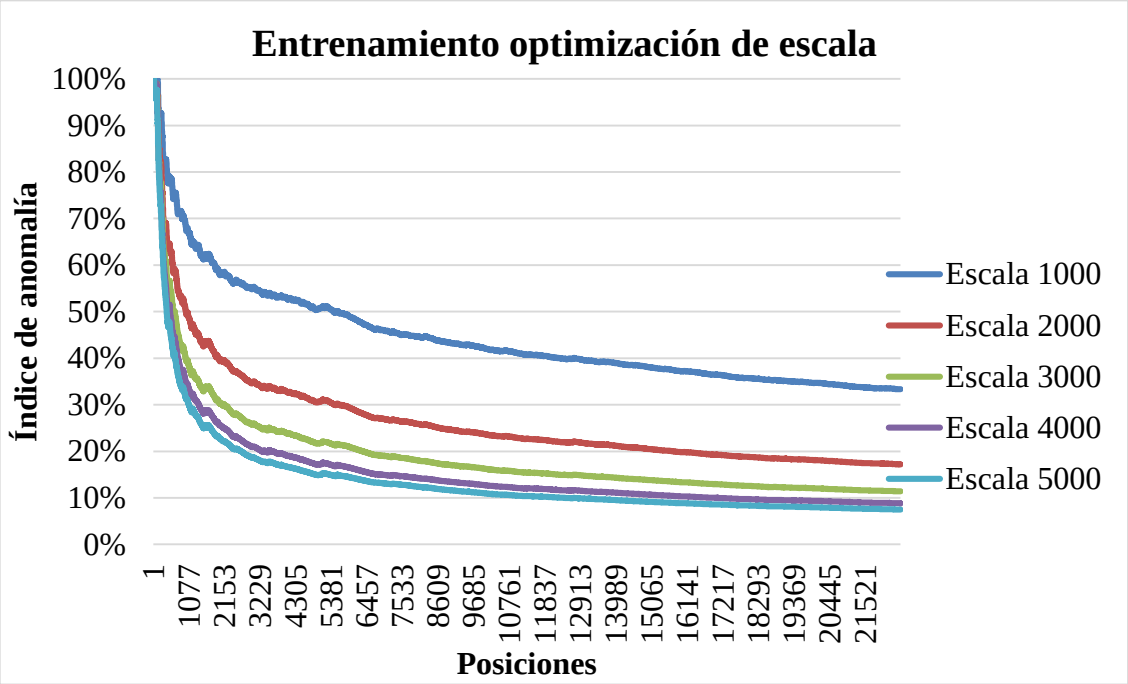


Figura 3-5 Gráfica del aprendizaje de la escala

Después de ejecutar NuPIC con los parámetros deseados y obtenido el modelo se procede a ejecutar el análisis. Primero se analiza los "vuelos correctos" para obtener el E_{vc} y se representa el siguiente gráfico Figura 3-6 para un mejor análisis. Se observa un despunte de los valores al inicio de la gráfica, esto se debe a que los vuelos tienen una gran anomalía en sus primeras posiciones hasta que NuPIC puede empezar a aprender los parámetros.

Del gráfico, se puede inferir que la escala que presenta un mayor error es la de 1000 ya que la resolución es demasiado elevada. Las escalas de 3000, 4000 y 5000 tienen un error menor al 6%. La escala de 2000 tiene un error de un 10%. Entonces se puede deducir que a partir de una escala superior a 3000 el error que se comete es pequeño, y hay que aumentar mucho la escala para disminuir el error.

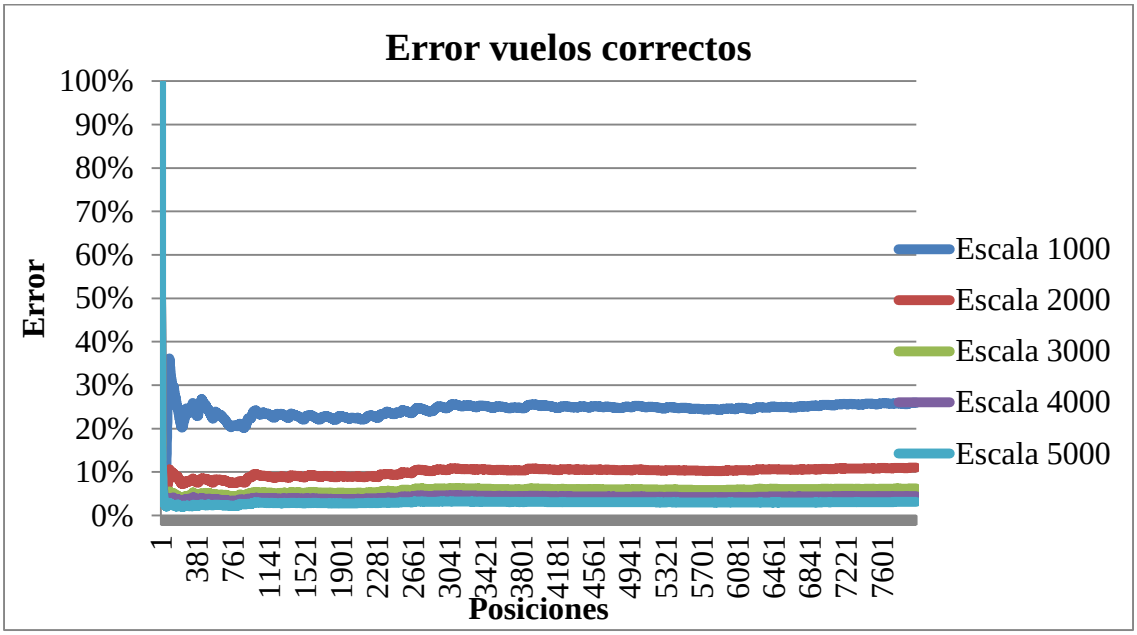


Figura 3-6 Error vuelos correctos con parámetros de escala

En el análisis de los "vuelos incorrectos" será el que determine la escala que se elegirá como más apropiada para la identificación. Se puede observar en la Figura 3-7 que el error es mucho más elevado que en los "vuelos correctos". Esto es debido a que los vuelos analizados son muy similares al del modelo aprendido.

Como era de esperar el error más bajo se obtiene para la escala de 1000 que detecta más fácilmente los vuelos anómalos. Sin embargo, se puede ver que la escala de 2000 tiene un error relativamente bajo con respecto al resto de valores. En 3000 tiene un error únicamente un 5% superior a 2000. El resto se quedan fuera de rango por su error tan elevado.

Finalmente se escoge el valor de escala de 2000 porque es el valor más apropiado con respecto a los datos y que además tiene un índice de error no muy elevado en los "vuelos correctos" y se comete el menor E_{vi}. Además, hay que tener en cuenta que cuanto menor sea la escala se tendrá una mejor resolución del mapeado.

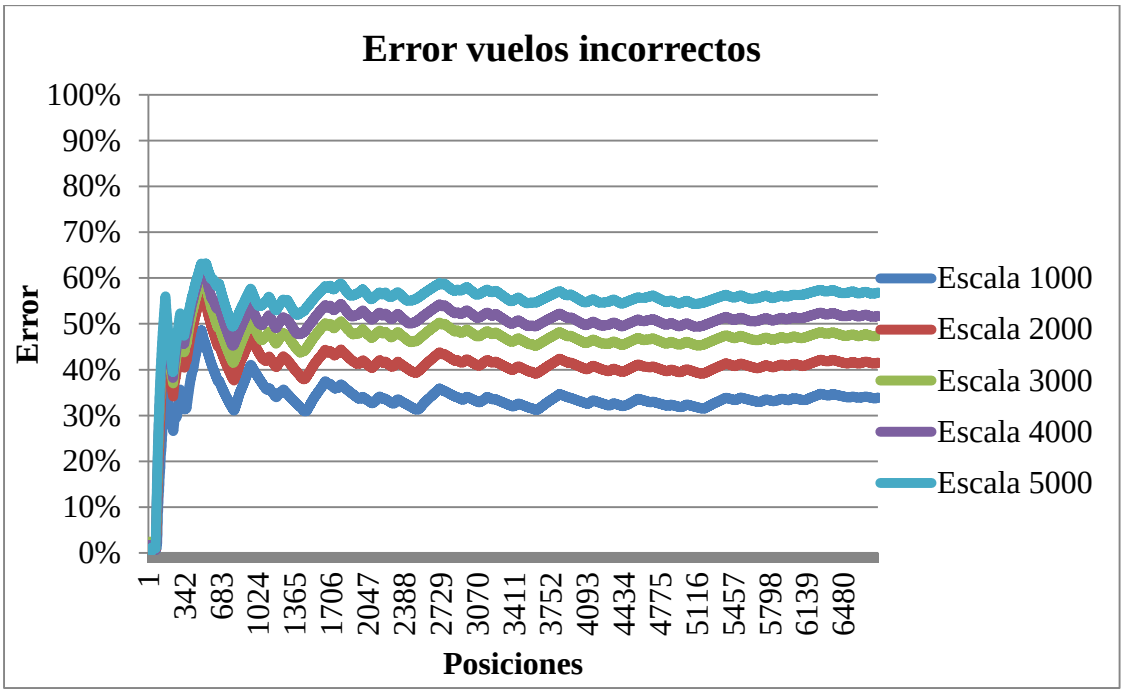


Figura 3-7 Error vuelos incorrectos con parámetros de escala

3.4.2 Peso del codificador temporal

Este parámetro hace referencia al número de bits que se asignará dentro de la SDR al codificador temporal. Este peso es proporcional a la importancia que se le dará a este parámetro dentro de la red. Así que, se intentará buscar el parámetro que minimice los errores (E_{vi} y E_{vc}).

Primeramente, se realiza el aprendizaje con la escala previamente optimizada y se le añade el f*lag "–t"* a la ejecución del *script* de aprendizaje para implementar el codificador temporal. Previamente a la ejecución se realiza la modificación del código para que calcule tres modelos diferentes con respecto a los pesos (w) con valores entre 11, 25 y 51. Es importante recordar que los pesos siempre han de ser valores impares por temas de funcionamiento de código. Tras la obtención de los modelos se analizan los errores cometidos.

Como se puede observar en la Figura 3-8 con los "vuelos correctos", el error menor es para un mayor peso ya que identifica y predice mejor el comportamiento de los vuelos (incluyendo el tiempo que es fácil de predecir). Cuando disminuimos a la mitad (w=25) o a un cuarto (w=11) el error aumenta ya que reconoce en menor medida los patrones temporales.

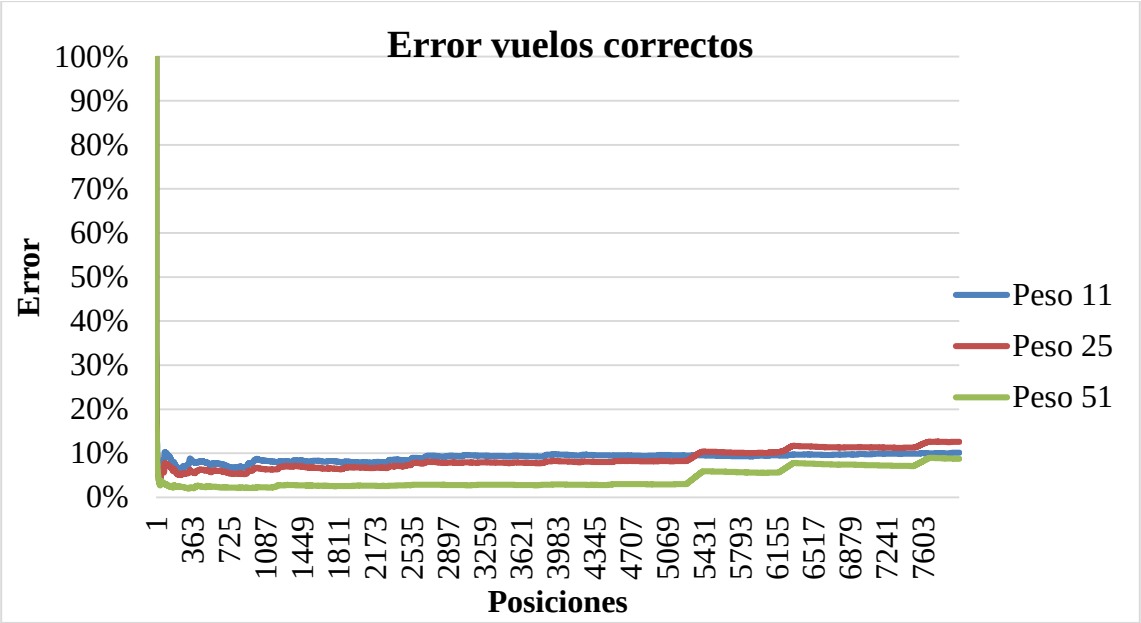


Figura 3-8 Error vuelos correctos con parámetros de peso en el codificador temporal.

Sin embargo, en la Figura 3-9 se puede observar que para un peso de 51 tendremos un error bastante elevado detectando vuelos que van a la misma hora aunque vayan por rutas diferentes. Por otro lado, se puede observar que el que error menor se comete para un peso de 25.

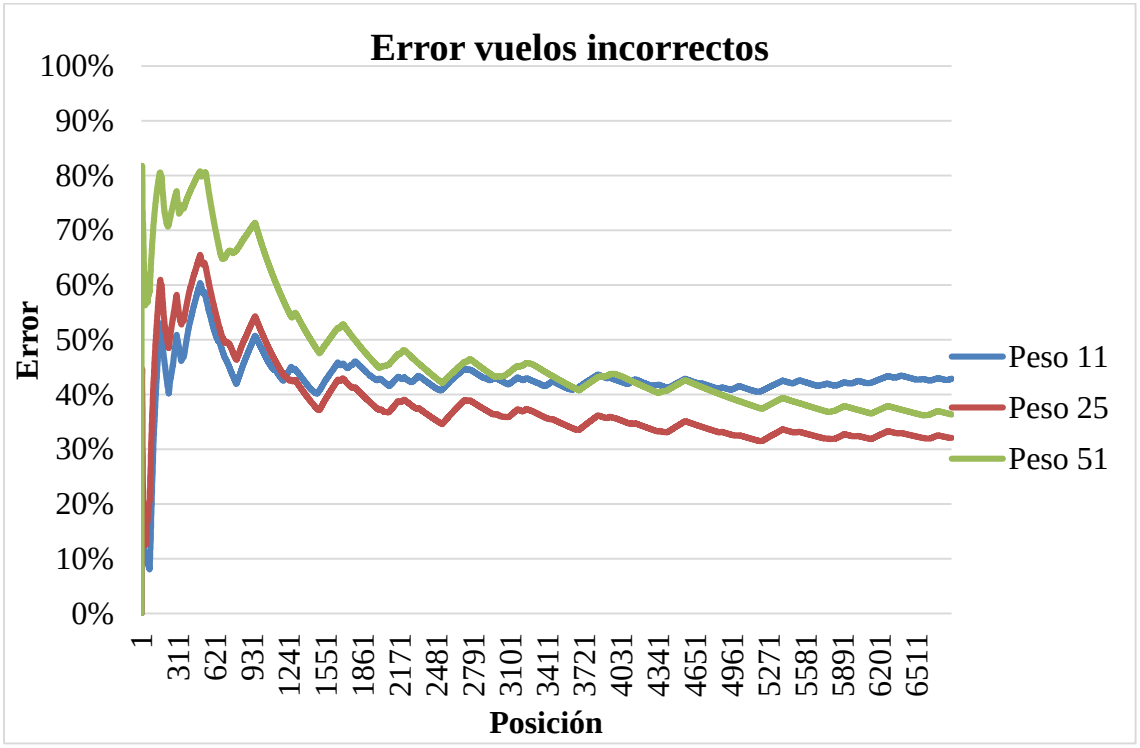


Figura 3-9 Error vuelos incorrectos con parámetros de peso en el codificador temporal.

Con las conclusiones obtenidas de las gráficas se decide tomar el valor de un peso de 25 para el codificador temporal ya que es el que minimiza los errores en menor medida.

3.4.3 Radio del codificador temporal

El radio temporal es la resolución temporal con la que NuPIC representará los datos temporales en la SDR. Para su estudio se variará entre los valores de 9.5 que es el valor por defecto en los algoritmos de NuPIC geospatial [47], y un valor inferior para obtener una mayor resolución temporal, dichos

valores serán 9.5, 4 y 1. Para ello se crearán los tres modelos y se analizarán los vuelos como en los anteriores casos.

En la Figura 3-10 se puede observar que con los "vuelos correctos" el radio mayor es el que produce un menor error ya que la anomalía del sistema será menor. Esto se debe a que NuPIC aprenderá más rápido cuando más grande sea el radio ya que la predicción de la red será más amplia. Sin embargo, se observa que para un radio de 4 se produce un peor reconocimiento de los vuelos correctos dando un mayor error. El radio de 1 es el parámetro más estable, cuya dispersión es mucho menor como se deduce de la gráfica.

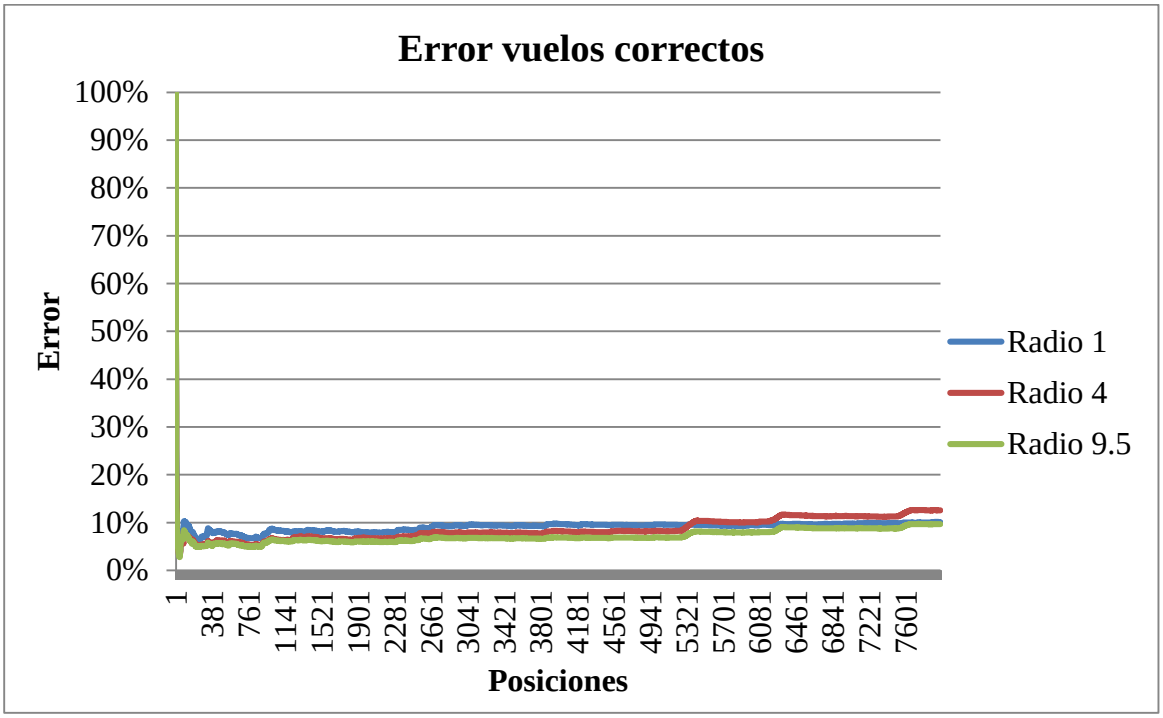


Figura 3-10 Error vuelos correctos con parámetros de radio en el codificador temporal.

En esta otra Figura 3-11 por el contrario se puede observar que un radio demasiado bajo identifica más vuelos que no son correctos como correctos. Por otra parte, si el radio es demasiado elevado comenzará también a aumentar el error.

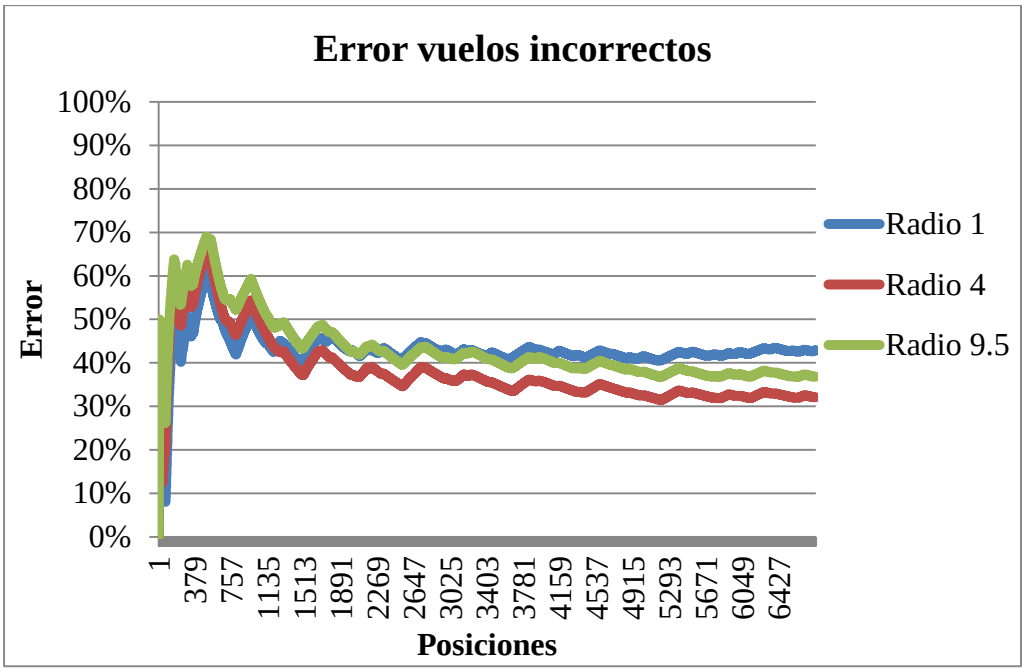


Figura 3-11 Error vuelos incorrectos con parámetros de radio en el codificador temporal.

Atendiendo a las conclusiones descritas se va a tomar el valor de radio 4, ya que es el que disminuye en mayor porcentaje el valor del E_{vi}. Así, se tendrá una mayor resolución temporal, pero sin aumentar tanto el error.

3.4.4 Codificador de Rumbo

Por último, el rumbo que no es un parámetro que haya que optimizar sino un dato que puede incorporarse para mejorar los resultados de aprendizaje e identificación de NuPIC. Para ello se añadirá un codificador escalar que codifica los valores de rumbo suministrados y los incorpora a la SDR. Esto mejorará las predicciones y como consecuencia disminuirá la anomalía reduciendo del mismo modo el error en la identificación.

Por tanto, añadimos el codificador escalar al *script* para así tener en cuenta este parámetro en el aprendizaje y funcionamiento del modelo, aumentando la precisión. En la Figura 3-12 se puede observar como disminuye el porcentaje de anomalía gracias a la implementación del codificador. El porcentaje de mejora, según los datos obtenidos con el codificador escalar, es de una disminución del 2% el índice de anomalía.

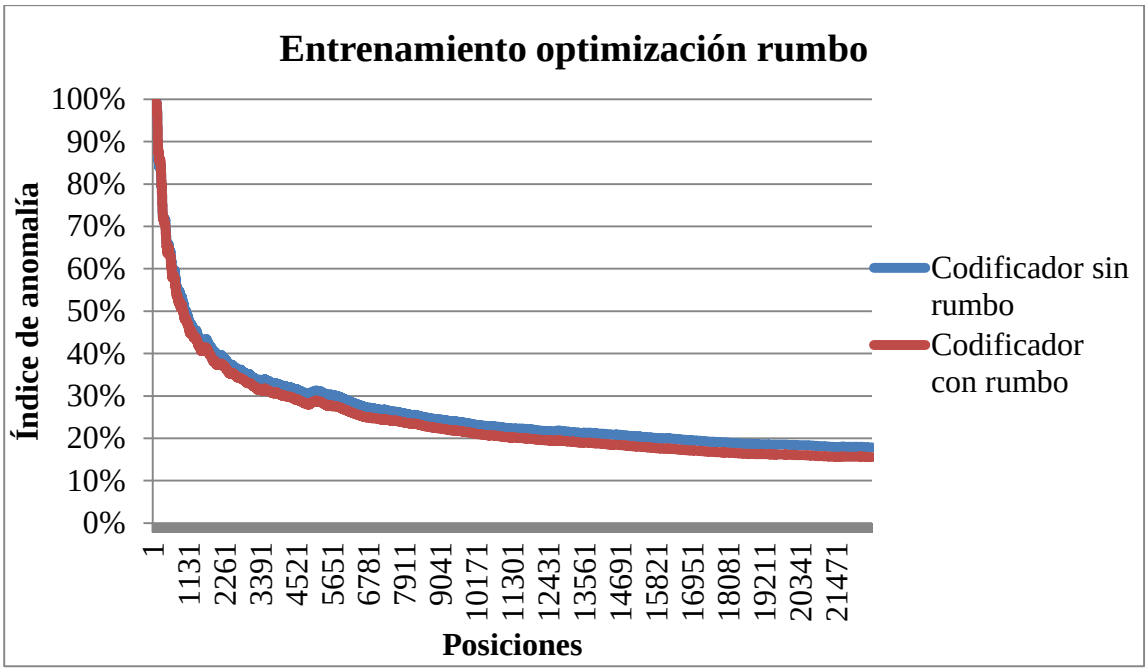


Figura 3-12 Aprendizaje con codificador escalar del rumbo

3.5 Desarrollo de prototipo

En este apartado se describe la aplicación desarrollada para entrenar y optimizar los modelos y la aplicación de identificación definitiva que emplea los modelos optimizados.

3.5.1 Aplicación para el aprendizaje

En este apartado se describe el funcionamiento de la aplicación desarrollada para entrenar los vuelos. El diagrama de flujo en la Figura 3-13 muestra el funcionamiento del *script* denominado run-_new.py. Al ejecutarlo, nos permite añadir una serie de *flags*[16] que permiten configurar ciertas características de la ejecución. Esos *flags* son los siguientes:

- **-v/ - - verbose:** esta opción permite visualizar como se ejecuta el código, imprimiendo en pantalla las anomalías de cada posición, así como las posiciones que han sido descartadas.
- **-s / --scale:** al añadirlo se define el tamaño de las cuadrículas con las que se procesará NuPIC Geospatial las posiciones geográficas tal y como se explicó en la sección 2.2.3. En caso de no establecer ninguna el valor por defecto será 1000.
- **-t /--time-encoder:** establecer un codificador temporal para que NuPIC tenga en cuenta los datos temporales en su aprendizaje.
- **-m /--manual-secuence:** establecer las particiones de datos, en este caso los vuelos, para que se realicen de forma manual. La partición en secuencias se realiza para cada vuelo automáticamente. Este parámetro no se utiliza en este caso
- **-o/ --output-dir:** se utilizará en el caso que se quiere establecer un directorio diferente al asignado por defecto.
- **-h /--help:** imprime por pantalla los anteriores *flags* e indica como se ha de ejecutar el comando.

La aplicación se ejecuta mediante el siguiente comando en el directorio del *script*, añadiendo los *flags* mencionados anteriormente que se quieran establecer, además se ha de añadir el archivo de entrada CSV que tendrá los vuelos (flight), en este caso:

[16] *Flags*: Es un valor binario que se le añade a la estructura de un programa para asignarle a un valor o función específicos.

```
python run_new.py flight
```

Hecho esto, comenzará a correr el código de NuPIC donde la función “run_new” es la que llama a las necesarias para procesar los datos de entrada.

Primeramente, toma el fichero “flight” y lo procesa llamando a la función de “preprocess”, que se encuentra definida en el *script* “preprocess_data.py”. Esta función recorre todo el archivo flight y comprueba que las posiciones tengan como mínimo un retardo temporal de 5 segundos, descartando todas aquellas posiciones que no cumplan con esta condición. Esta función devuelve un archivo CSV denominado preprocess_data.csv.

A partir del archivo devuelto por la función “preprocess”, se llama a la función geospatial_anomaly que es importada del *script* “geospatial_anomaly_new.py”. En primer lugar, lo que realiza este *script* es preguntar el nombre que se le va dar al modelo que aprende. Para la creación del modelo se basa en la función “model_params” importada de “model_params.py”, esta función crea el modelo con los parámetros definidos en el mismo *script*. Volviendo a la función “geospatial anomaly” se definen los diferentes codificadores que se utilizarán para convertir los parámetros, la activación del codificador temporal mediante el uso de (-t/--time-coding) y la activación del codificador geoespacial con el uso de (-s/--scale). Después, se procesan los datos de entrada creando el modelo, recorriendo el archivo “preprocess_data.csv” con los datos del aprendizaje de los mismos. Está función devuelve un fichero CSV, llamado “anomaly_score.csv” con el índice de anomalía. Además, se calcula otro parámetro de salida con la media de todas las posiciones procesadas hasta el momento; este parámetro es de especial relevancia ya que será utilizado posteriormente como medida del error. Por último, la función guarda el modelo con sus pesos, después del aprendizaje, con el nombre solicitado al comienzo del procesamiento de “gespatial.anomaly”, y genera un segundo archivo que guarda los nombres del vuelo aprendido y la media de su anomalía.

NuPIC se ha modificado para que automáticamente detecte una nueva secuencia al cambiar el número de vuelo. Aprendiendo de cada secuencia en el caso de que se quiera cambiar el orden de las secuencias solamente habría que agregar el *flag* (-m/--manual).

La última función realizada por “run_new.py” es la de “postprocces” que crea un archivo de salida el cual, puede ser procesado por server.py. Este último *script* convierte los datos de salida en un formato legible en Java Script y crea un servidor virtual para su visualización en una página web.

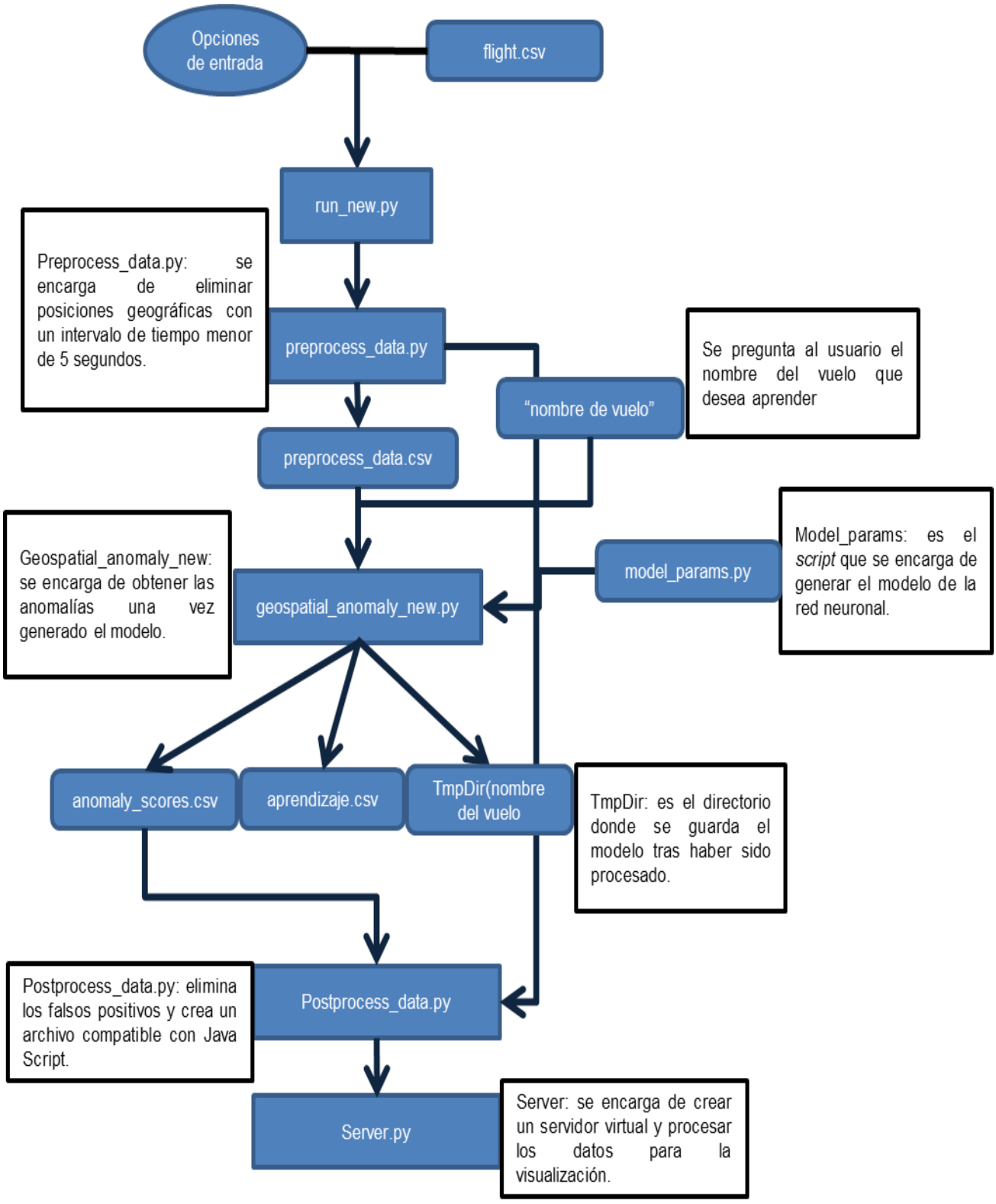


Figura 3-13 Diagrama de flujo run_new.py

Posteriormente, se desarrolla un segundo *script* similar al anterior con la diferencia de que se le desactiva el aprendizaje. Además, este *script* carga modelos de los vuelos que hayamos guardado previamente con la ejecución del "run_new.py". A este *script* no se le puede añadir ningún parámetro, ni cambiar parámetros porque los carga automáticamente en el modelo guardado. Su propósito es la evaluación de los vuelos para identificar los errores que se cometen y conocer el nivel de identificación a través del índice de anomalía.

Por tanto, al igual que en el anterior *script*, "run_load.py" llama a la función "preprocess_data" (visualizar el código en el Anexo VI: preprocess_data.py), que nos devuelve los datos preprocesados conforme a lo explicado anteriormente. Tras esto, se llama a la función "geospatial_anomaly" importada del *script* "geospatial_anomaly_load.py" donde ya no es necesario definir parámetros, sino que ahora se buscarán y cargarán los modelos guardados. Además, se desactiva el aprendizaje ya que solamente se quiere evaluar los vuelos. Al igual que en el *script* de aprendizaje, devolverá un archivo CSV (score_anomaly), el cual servirá para la identificación de los vuelos y su parecido con los modelos. En el diagrama de flujo de la Figura 3-14, se puede observar su funcionamiento.

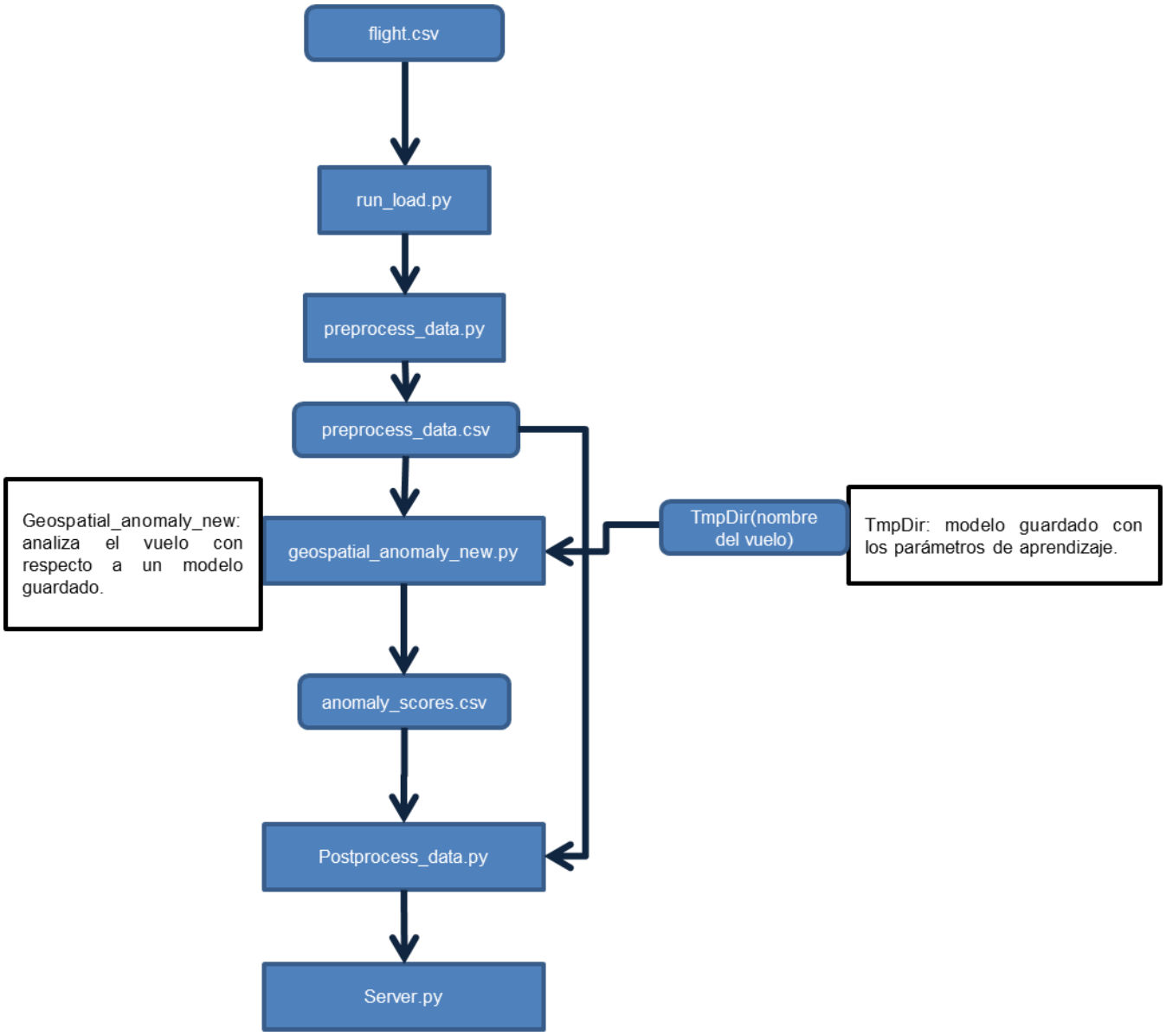


Figura 3-14 Diagrama de flujo de run_load.py

3.5.2 Desarrollo de la aplicación de identificación

En este apartado se va a describir el funcionamiento de la aplicación desarrollada a partir de los *scripts* de NuPIC para identificación de vuelos.

Para ello, se han de guardar los modelos ejecutando el *script* "run_new.py" (visualizar el código en el Anexo I: run_new.py) indicando un archivo CSV con un nuevo vuelo. Como se muestra en la Figura 3-15, la aplicación pregunta al usuario qué nombre le quiere dar al modelo. Puede ser el número de vuelo, de la aeronave, o el nombre que quiera designar al usuario, y así será reconocido posteriormente por la aplicación.

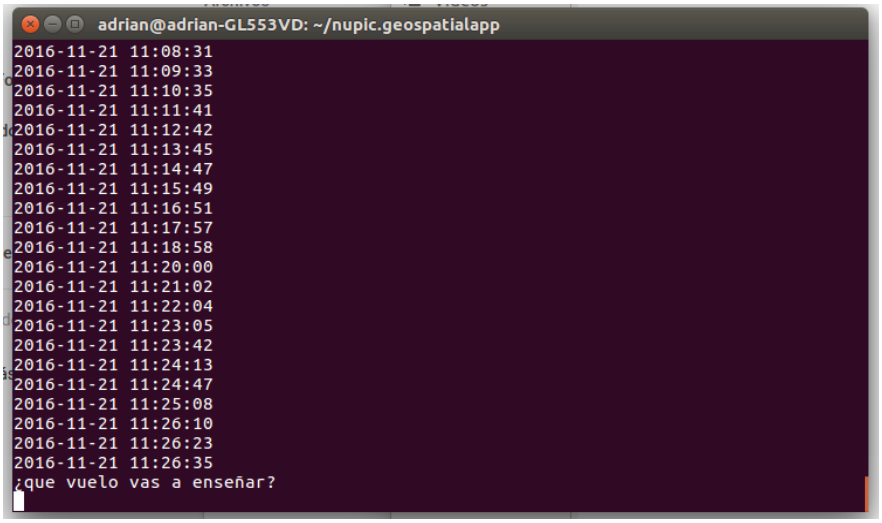


Figura 3-15 Impresión de pantalla de run_new.py

Una vez se dispone de una base de datos mínima se puede ejecutar el *script* con la aplicación propiamente dicha, descrita en el diagrama de flujo de la Figura 3-19. Este *script* denominado "identificacion.py" (para visualizar el código de éste módulo del Anexo II: identificacion.py al Anexo VII: server.py), se ejecuta en la consola, al igual que el *script* "run_load.py" descrito en la Figura 3-14 en apartados anteriores. Es importante tener en cuenta que solo se podrá añadir el *flag* "–t" o "–v", ya que el resto de parámetros se predefinen en "identificación.py".

Al comenzar, se realiza el preprocesado de los vuelos con "proprocess_data.py", tras esto se llamará a la función "geospatial_anomaly_identificacion" (para visualizar el código Anexo IV: geospatial_anomaly_identificacion.py) . Esta función, primeramente, listará los modelos guardados, entonces leerá el archivo (preprocess_data.csv) y lo analizará sacando el error del vuelo para cada uno de los diferentes modelos como se observa en la Figura 3-16. Estos errores se listarán en otro documento, indicando nombre del modelo al que pertenecen, documento CSV denominado (flightanomaly). Por otra parte, se creará el archivo (anomaly_score.csv) de cada documento, el cual se utilizará para la representación del vuelo.

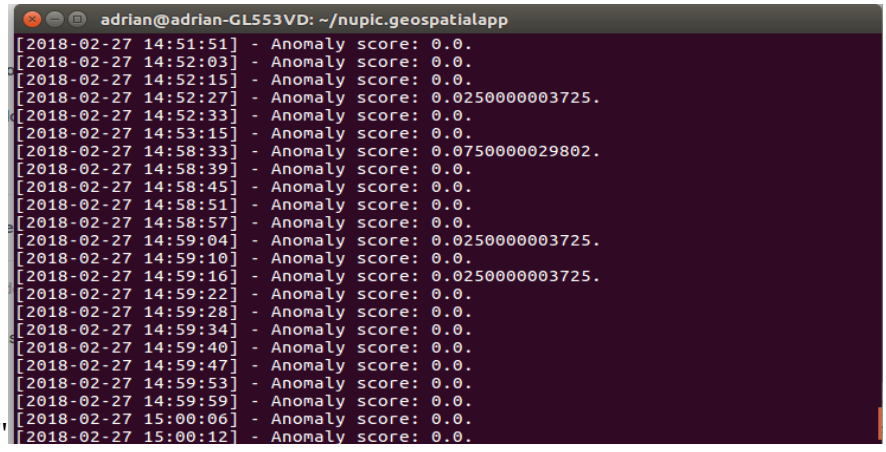


Figura 3-16 Impresión por pantalla de AnomalyScore

Tras realizar el análisis del vuelo con los diferentes modelos aprendidos se representa por pantalla el vuelo con mayor similitud mediante un porcentaje, calculado como se indica en la Ecuación 6.

$$Similitud = \left(1 - \overline{AnomalyScore}\right) * 100$$

Ecuación 6

Después, se le pregunta al usuario por pantalla si quiere entrenar al modelo con mayor similitud, con este nuevo vuelo, como se puede observar en la Figura 3-17 . En caso afirmativo se llama a la función “run_load” que a su vez llamará a “geopatial_anomaly_reload” (para visualizar el código Anexo V: geopatial_anomaly_reload.py) que procesará el vuelo de nuevo con la diferencia de que esta vez aprenderá del mismo.

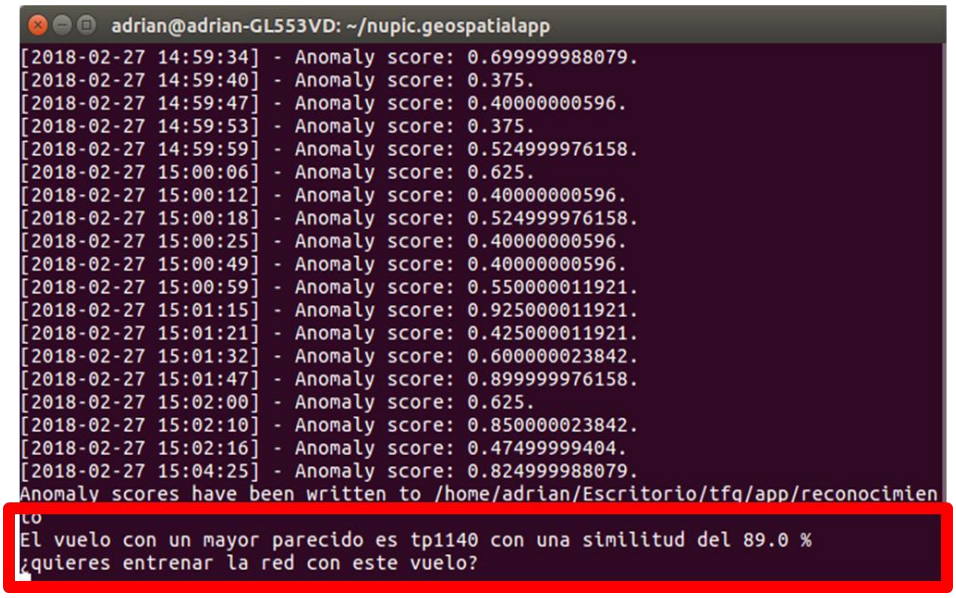


Figura 3-17 Impresión de pantalla similitud del vuelo TP1140

Por último, la aplicación preguntará si quiere realizar la representación del vuelo. En caso afirmativo se realizará el postprocesado (para visualizar el código Anexo III: postprocess_data.py) para poder representar los datos con la aplicación web. Tras lo cual, se ejecuta el *script* ‘‘serverload.py” (para visualizar el código Anexo VII:) y muestra la representación gráfica. En ésta se representa, mediante una escala de color, donde el rojo se toma como más anómalo y verde como no anómalo. Esto permite, una vez identificado, visualizar en que parte del tramo se ha producido la anomalía. En la Figura 3-18 se puede observar un vuelo con los diferentes colores según el índice de anomalía.

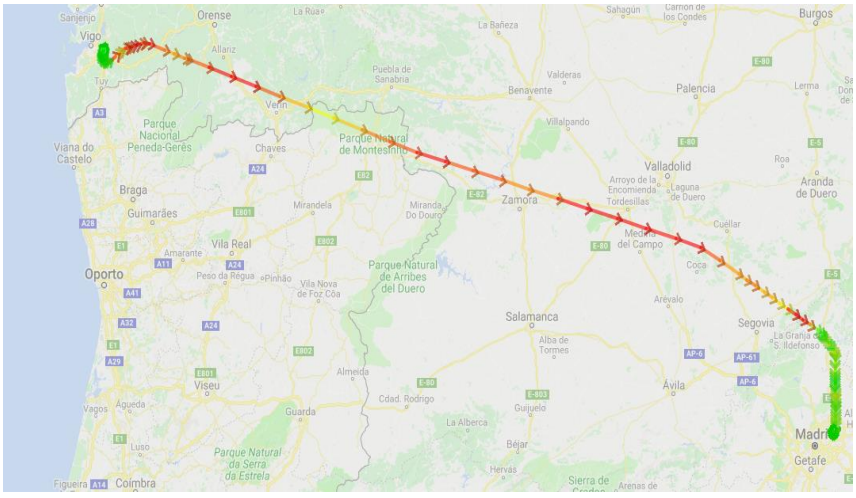


Figura 3-18 Representación de un vuelo con anomalías en escala de colores

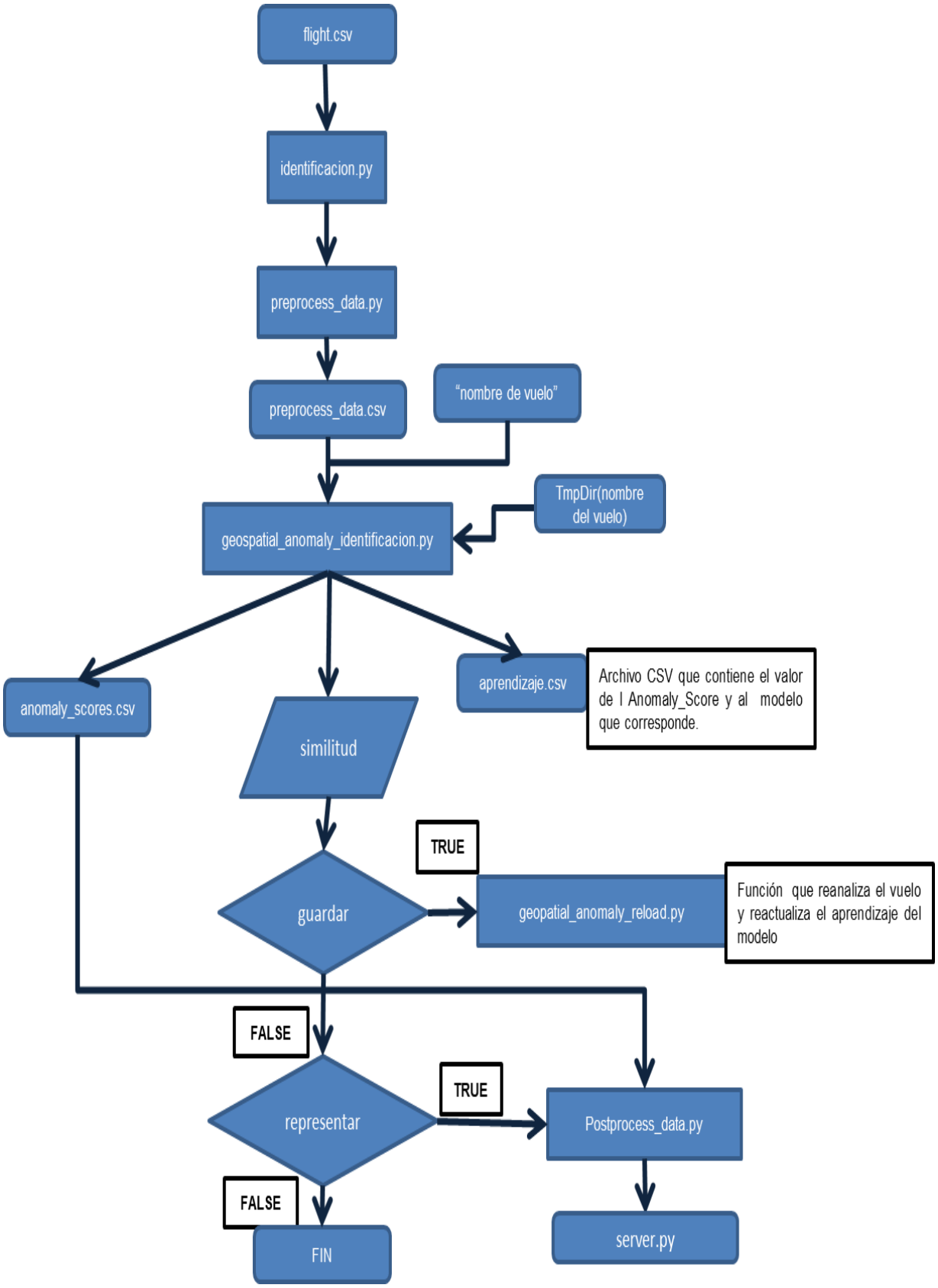


Figura 3-19 Diagrama de Flujo de la Identificación.py

4 RESULTADOS Y PRUEBAS

En esta sección se recogen las pruebas y los resultados para validar el funcionamiento de la aplicación. En primer lugar, se muestra los resultados de la optimización del modelo en apartados anteriores. Posteriormente, se describen las pruebas realizadas con la aplicación completa para validar su correcto funcionamiento y evaluar sus capacidades para la identificación de los vuelos.

4.1 Resultado de la optimización del modelo.

En este apartado se representan los resultados finales del conjunto de optimizaciones realizadas y se expone una breve explicación de los mismos.

Para evaluar las diferentes optimizaciones consideradas, se realiza primeramente el procesamiento con los parámetros por defecto, que se pueden ver en Tabla 4-1, al conjunto de vuelos descritos en el apartado 3.4, y se analizan los errores cometidos. Después, se realiza de nuevo el análisis de los vuelos, empleando los parámetros optimizados mostrados en la Tabla 4-1.

	Escala	**Peso codificador temporal**	**Radio codificador temporal**	**Codificador escalar (Rumbo)**
Sin Optimizar	1000	No	No	No
Optimizado	2000	25	4	Si

Tabla 4-1 Parámetros resultados finales

Inicialmente, se analiza la diferencia de velocidad de aprendizaje entre uno y otro modelo, graficando el índice de anomalía medio en función de las posiciones suministradas al modelo, como se muestra en la Figura 4-1. Se observa que el modelo optimizado aprende más rápidamente y además tiende a un índice de anomalía inferior, modelando mejor los datos suministrados (teniendo en cuenta que son datos del mismo vuelo, el índice de anomalía debería tender a 0).

Esto se debe a que el modelo sin optimizar tiene un menor número de parámetros que proporcionan información útil para modelar los patrones existentes en los datos de entrada. Aunque podría averiguar indirectamente el rumbo a partir de las posiciones, al añadir directamente el codificador escalar con el rumbo aprenderá con mayor rapidez como varía este parámetro.

Por tanto, se puede deducir de esta primera gráfica (Figura 4-1) que el modelo optimizado aprende mejor y más rápido, lo que será de gran utilidad para la identificación. Con un menor número de

vuelos la red tendrá un modelo con un aprendizaje suficiente para poder obtener los patrones que identifican mejor a los vuelos.

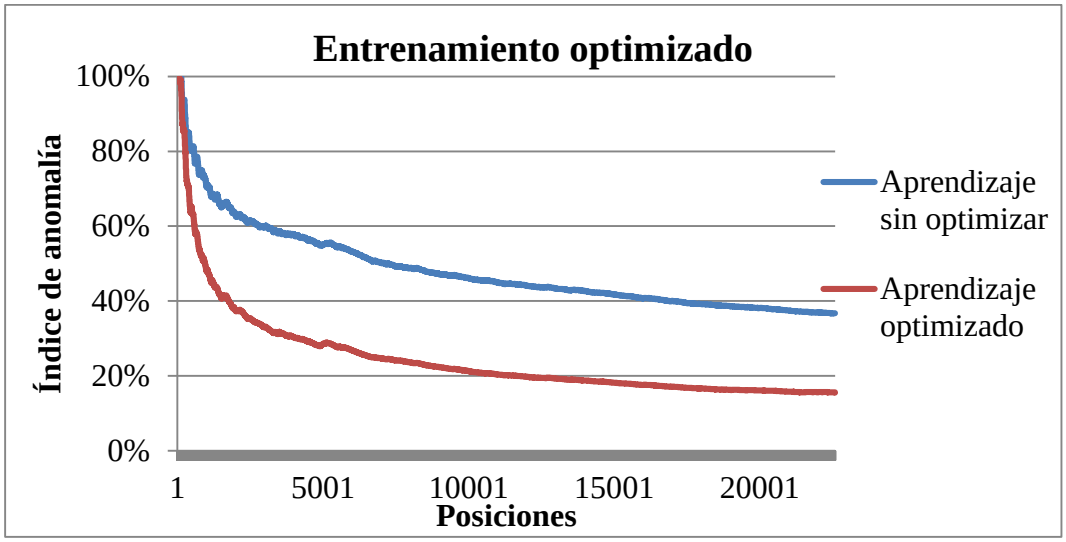


Figura 4-1 Gráfica del resultado final del aprendizaje optimizado

En segundo lugar, se realiza el análisis de los 50 "vuelos correctos" para poder obtener el Error de identificación comentado en el apartado 3.4. Se grafican los resultados como se puede observar en la Figura 4-2. Se aprecia que el valor del E_{vc} es mucho menor en el modelo optimizado; por tanto, se puede concluir que realiza una mejor identificación de los vuelos. Estos resultados son coherentes ya que el modelo optimizado ha aprendido con mayor rapidez de los datos, por lo que sus predicciones serán más correctas, disminuyendo su índice de anomalía y como consecuencia el E_{vc}.

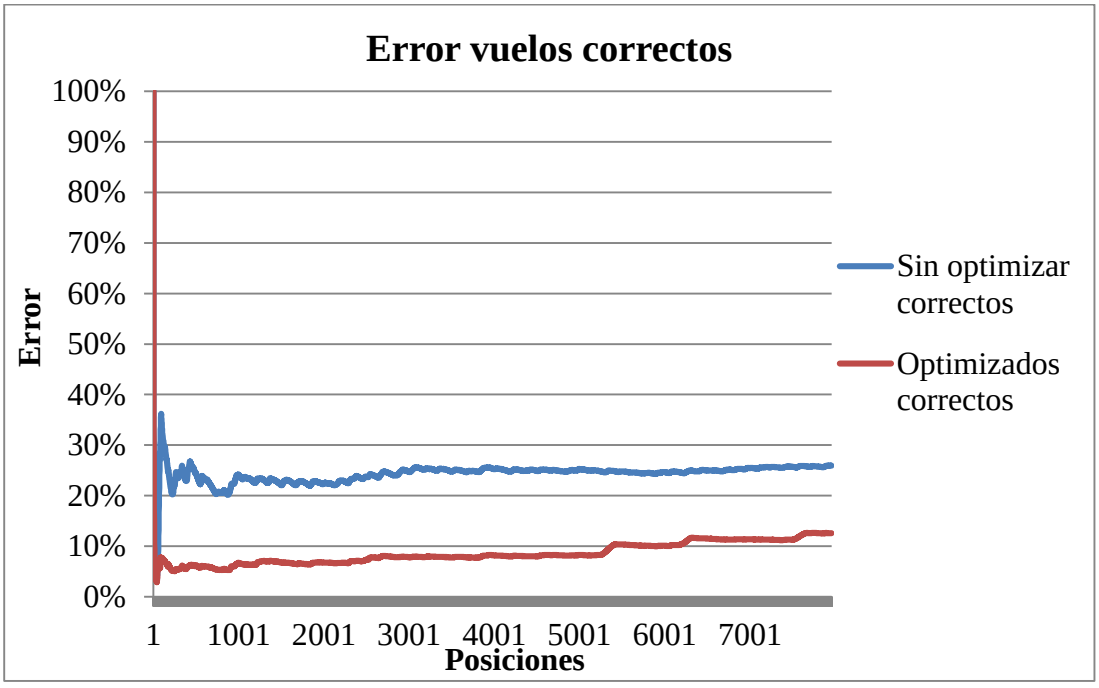


Figura 4-2 Gráfica del resultado final de errores de los vuelos correctos

Por último, se realiza el análisis de los "vuelos incorrectos" cuyos valores de error son más elevados. Se realiza el procesamiento de los datos y su correspondiente gráfica mostrado en la Figura 4-3. Se puede observar que en las primeras posiciones los vuelos tienen un error muy alto, siendo superior el modelo optimizado. Esto se debe a que el modelo sin optimizar detecta con más facilidad las anomalías ya que no ha aprendido suficiente. Sin embargo, se observa que en el análisis del

conjunto de vuelos el error es menor. Es decir, que el modelo optimizado no solo identifica mejor los datos pertenecientes al propio vuelo, sino que detecta mejor si los datos introducidos se corresponden con otro vuelo (índice de anomalía más alto).

Los picos que se aprecian en la gráfica son las posiciones con una mayor coincidencia respecto al modelo aprendido. Como se expuso en 3.4 "vuelos incorrectos" pasan por Madrid siguiendo la misma ruta. Por tanto, hay numerosas posiciones que NuPIC no es capaz de diferenciar con exactitud.

En conclusión, el error E_{vi} que tiene más impacto debido a sus valores más elevados, se reduce significativamente con la optimización.

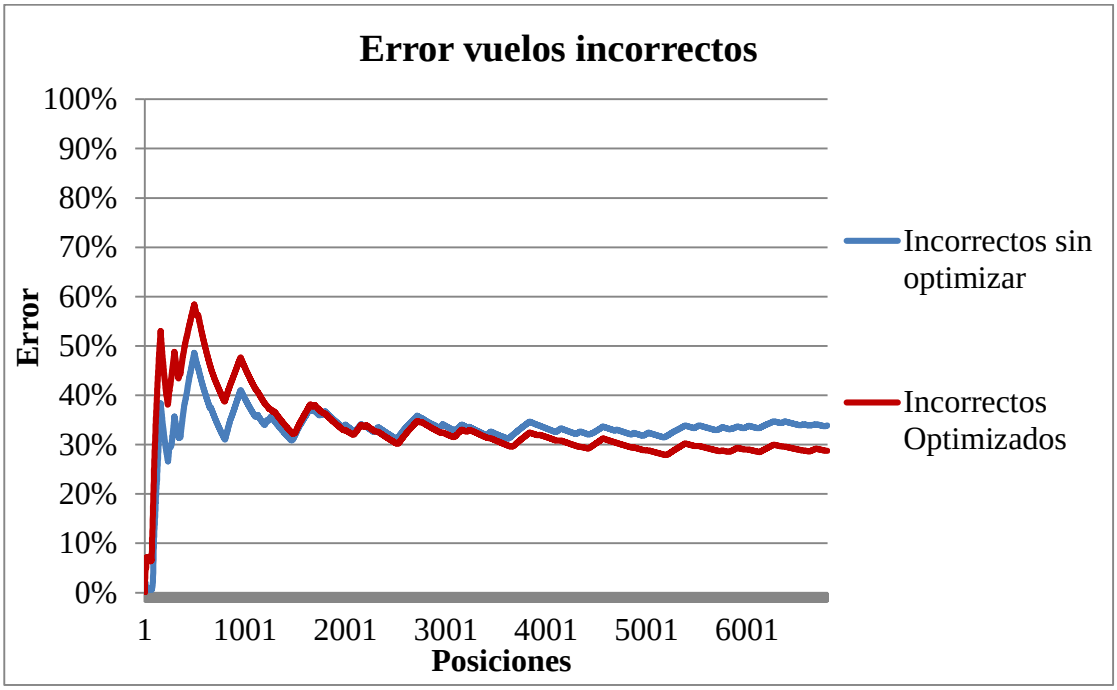


Figura 4-3 Gráfica de resultado final de errores de los vuelos incorrectos

Un resumen de los datos obtenidos se presenta en la Tabla 3-1, donde se puede observar que el modelo obtiene una mejora del 21,13% en el índice de anomalía con las mismas posiciones. Realizando la media de la optimización, se obtiene una mejora del modelo del 13,26% con respecto al modelo por defecto de NuPIC Geopatial.

	Sin optimizar	**Optimizado**	**Mejora de la Optimización[17]**
Índice de anomalía	36,68%	15,55%	21,13%
E_{vc}	25,95%	12,55%	13,40%
E_{vi}	33,99%	28,74%	5,25%

Tabla 4-2 Resultados finales de Aprendizaje y Errores para ambos modelos

4.2 Pruebas de la aplicación

En este apartado se van a presentar las pruebas realizadas, sus respectivos resultados y su correspondiente evaluación.

[17] Mejora de la Optimización = (sin optimizar) – (optimizado)

Estas pruebas consisten en evaluar si la aplicación es capaz de identificar el destino al que se dirige la aeronave a partir de sus posiciones.

En este caso, en el que sólo se está interesado en el destino de las aeronaves, se quiere que los vuelos que se repiten a lo largo del día, con rutas idénticas y horarios diferentes, se asocien al mismo modelo. Por tanto, a la hora de ejecutar los modelos no se activará el *flag* "–t", que como se explicó en el apartado 3.5.1 sirve para activar el codificador temporal.

Debido a la cantidad de vuelos diarios que se dan en la actualidad, después de un estudio de varios aeropuertos peninsulares, se decide tomar como referencia un aeropuerto pequeño para identificar los vuelos que salen del mismo. Conforme a ese criterio se toma el aeropuerto de Peinador de la ciudad de Vigo, ya que como se puede observar en la Figura 4-4, únicamente salen vuelos directos a los aeropuertos de las ciudades de Lisboa, Madrid, Barcelona y Bilbao. Por tanto, todas las pruebas se realizarán en este marco geográfico.

Figura 4-4 Rutas que realizan los aviones desde Vigo (Fuente: [40])

Para su entrenamiento se utilizarán los vuelos que se reseñan en la Tabla 3-1 tomando 50 vuelos para cada uno de ellos. Así se obtendrán siete modelos con los que se podrá evaluar cualquier vuelo que despegue o aterrice en el aeropuerto de Vigo. Cada modelo se guardará con el nombre de Origen-Destino, para obtener como resultado de donde sale o a donde se dirige. En la Tabla 4-3 también se puede observar el nivel de similitud media de los vuelos es dependiente de cuan parecidos sean los 50 vuelos unos a otros, aprendiendo con más facilidad el sistema cuanto más se parezcan.

Número de vuelo	Origen	Destino	Similitud
VY1703	Vigo	Barcelona	76,39%
VY1704	Barcelona	Vigo	78,08%
TP1140	Lisboa	Vigo	71,40%
TP1141	Vigo	Lisboa	82,20%
UX7317	Vigo	Madrid	81,45%
UX7300	Madrid	Vigo	80,30%
IB8963	Vigo	Bilbao	67,10%
IB8962	Bilbao	Vigo	66,32%

Tabla 4-3 Vuelos de entrenamiento de la red.

Según los aprendizajes que se han obtenido, el vuelo que más cuesta identificar es el de Bilbao. El motivo se clarifica en la Figura 4-5, en la que se pueden observar que los vuelos son muy diferentes unos con respectos a otros, lo que explica la dificultad de modelarlo. El resto de vuelos tienen un índice de anomalía mucho menor habiendo aprendido con más facilidad y rapidez los patrones de vuelo, lo que dará porcentajes más altos de similitud.

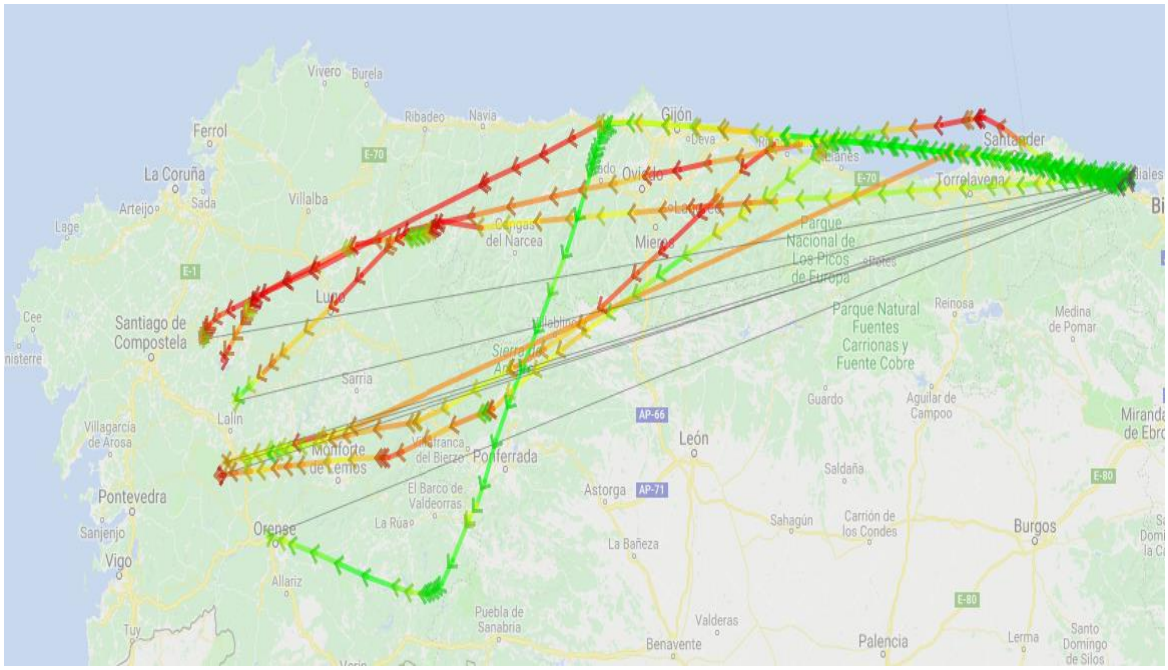


Figura 4-5 Representación de una parte de los vuelos Bilbao-Vigo

Una vez se tiene la base de datos se comienza con el análisis de los vuelos, tomando una muestra de 30 vuelos. Los vuelos son todos de salida y entrada en el aeropuerto de Vigo, escogidos aleatoriamente.

Para un mejor análisis se grafican los resultados de similitud obtenidos en la Figura 4-6. El porcentaje de acierto en la identificación de los vuelos ha sido de un 100% para este experimento, ya que en todos los casos la aplicación asignó al modelo correcto el mayor valor de similitud. Se puede observar que todos los rangos de similitud superan el 70% excepto en uno de los vuelos Bilbao - Vigo cuyo porcentaje de similitud es del 28%. Esto se debe a que es el modelo de Bilbao cambia a menudo de ruta como se mostró anteriormente en la Figura 4-5. A pesar de esto, el porcentaje de similitud del

resto de modelos era todavía menor, por lo que aun así se identificó correctamente que se trataba del vuelo Bilbao-Vigo.

De estos resultados se puede concluir que la aplicación identifica con acierto del 100% de las rutas, donde llega a alcanzar porcentajes de similitud de hasta un 95%. Sin embargo, se observan problemas en vuelos con rutas muy cambiantes que podrían causar errores en la identificación.

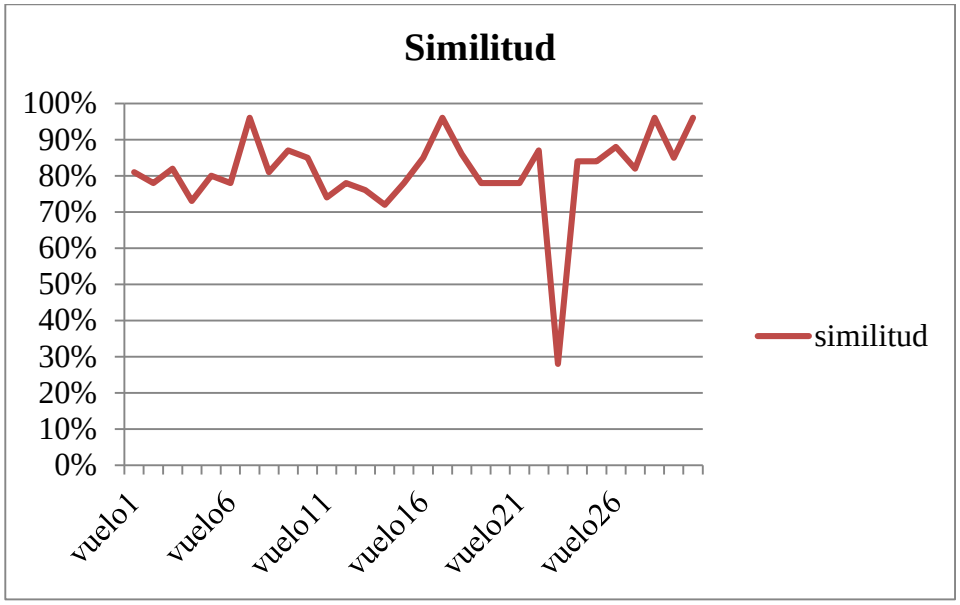


Figura 4-6 Similitud de los vuelos de entrada y salida de Vigo.

Tras esta primera prueba se observan resultados muy satisfactorios, decidiéndose realizar una segunda prueba que consiste en la identificación de los vuelos de llegada los sábados al aeropuerto de Vigo.

En primer lugar, se construye una base de datos con la que la aplicación pueda identificar los vuelos. Para ello, se descargan un número aleatorio de vuelos (siempre superior a 50) de cada uno de los vuelos que llega los sábados al aeropuerto de Vigo. Una vez realizada la descarga, se ejecuta el aprendizaje obteniendo los diferentes modelos y su porcentaje de similitud. En este experimento, tiene especial relevancia el tiempo por ello se ejecuta el *flag* "–t" para añadir el codificador temporal.

De este modo, se recogen en la Tabla 4-4 los vuelos modelados junto con su porcentaje de similitud. Se observa que los porcentajes de similitud son superiores a los de la prueba anterior, esto se debe al hecho de haber utilizado el codificador temporal.

Número de vuelo	Origen	Destino	Similitud	Salida	Llegada
FR245	Barcelona	Vigo	82,21%	1415	1610
IB530	Madrid	Vigo	82,49%	0730	0840
IB532	Madrid	Vigo	81,93%	1145	1300
IB538	Madrid	Vigo	82,30%	1940	2055
TP1140	Lisboa	Vigo	83,26%	1230	1455
UX7300	Madrid	Vigo	75,43%	1445	1555
UX7302	Madrid	Vigo	72,57%	1040	1150
UX7306	Madrid	Vigo	80,09%	1840	1950
UX7308	Madrid	Vigo	76,43%	0645	0755
VY1702	Barcelona	Vigo	77,81%	0730	0920

Tabla 4-4 Vuelos de llegada al aeropuerto de Vigo los sábados

En esta ocasión, para el análisis detallado de la similitud sólo se escogen 20 muestras de vuelos que lleguen los sábados al aeropuerto de Vigo, descritos en la Tabla 4-4.

La similitud de los vuelos presentada en la Figura 4-7 es mucho más alta con respecto al experimento anterior. Sin embargo, llega a cometer un fallo en la identificación concretamente en el vuelo UX7308 que fue identificado como IB530. El vuelo correcto (UX7308) obtuvo el segundo valor más alto de similitud. Es preciso destacar que ambos vuelos comparten el mismo destino (Madrid) y que ambos tienen un horario similar. Además, el vuelo seleccionado estaba retrasado, saliendo en el mismo horario que el IB530, haciendo imposible diferenciar entre uno. El resto de vuelos fueron identificados correctamente, resultando una estadística de acierto del 95%.

Se puede observar en la gráfica que la mayor parte de las predicciones rondan el 90% de similitud con el vuelo real lo que refleja que al utilizar el codificador temporal se aumenta en un gran porcentaje la tasa de acierto.

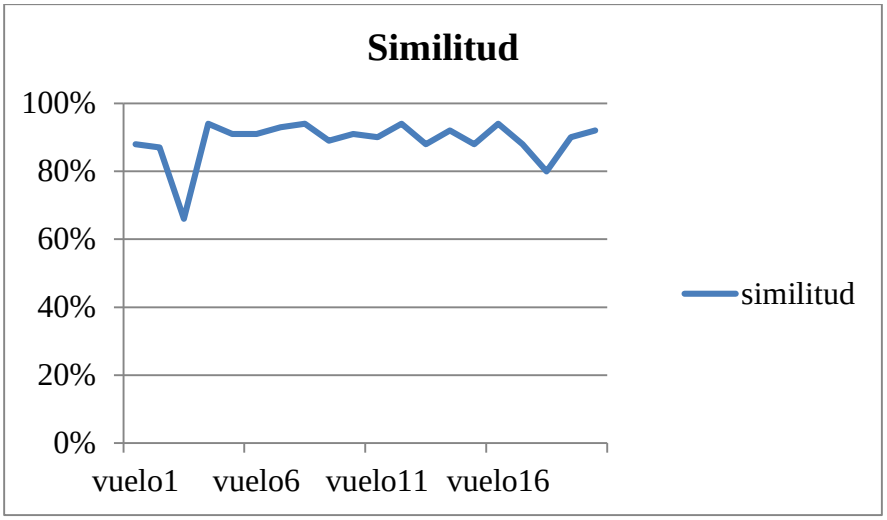


Figura 4-7 Gráfica de la similitud de los vuelos del aeropuerto de Vigo de los sábados

De ambos experimentos se puede destacar que la tasa de acierto de identificación es del 98% de los 50 vuelos analizados en condiciones diferentes. Los resultados de las pruebas realizados se consideran muy positivos debido a la alta tasa de acierto de la aplicación.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

En este apartado se revisan los objetivos específicos planteados al inicio de este trabajo y se analiza su grado de cumplimiento considerando los resultados y las pruebas realizadas.

Primeramente, se han analizado diversos servicios de datos aéreos y se ha escogido el más adecuado. Por otra parte, se ha desarrollado un *script* para convertir el formato y generar un único archivo CSV compatible con la aplicación desarrollada, como se describe en el apartado 3.3. Se puede concluir que el primer objetivo ha sido cumplido.

En segundo lugar, se han estudiado los parámetros relevantes para la identificación de los vuelos, para realizar una optimización de los mismos mediante la selección de los valores que disminuyan en mayor medida el error. Los resultados muestran que el índice de anomalía se ha mejorado un 21,13%, 13,40% para los vuelos correctos y 5,25% para los vuelos incorrectos. Por tanto, se considera cumplido el objetivo de optimización y selección de parámetros.

Por último, se desarrolló la aplicación, que a partir de las posiciones de las aeronaves es capaz de identificar los vuelos. Al analizar los datos la aplicación muestra el vuelo más similar de los previamente aprendidos con una medida de similitud. Además, permite incorporar los datos del nuevo vuelo al entrenamiento para mejorar dichas identificaciones y adaptarse a posibles cambios (rutas, retrasos, etc.). Se realizan diversas pruebas para comprobar su eficacia obteniéndose un resultado de un 98% de tasa acierto. Por ello este objetivo puede darse por alcanzando.

Atendiendo al cumplimiento de los objetivos específicos se puede concluir que se ha conseguido desarrollar un sistema basado en inteligencia artificial para la identificación de vuelos.

5.2 Líneas futuras

En cuestión a líneas de mejora de este sistema hay muchas y muy numerosas, empezando por la tecnología NuPIC que como se ha explicado en este trabajo está en continuo desarrollo. La aplicación conforme lo vayan haciendo los algoritmos de aprendizaje realizados por Numenta ya que se encuentran en continuo desarrollo. Otro posible enfoque de la aplicación que no se estudia en este trabajo sería la predicción de las posiciones futuras de las aeronaves mediante los algoritmos de predicción de NuPIC, lo que podría derivar en que la aplicación no fuese únicamente capaz de identificar los vuelos, sino que pudiese predecir posibles colisiones entre aeronaves

Por otra parte, sería de especial interés elaborar una base de datos que contenga los modelos por aeropuertos para probar esta tecnología a una mayor escala y estudiar los tiempos de procesado. Podría

mejorarse el procesado de la aplicación mediante el uso de mecanismos de concurrencia y así poder utilizarla a gran escala con los numerosos vuelos simultáneamente.

Otra vía de mejora de la aplicación sería mediante una optimización del parámetro de escala en mayor profundidad. En este trabajo se utiliza un parámetro de escala constante. No obstante, se ha observado que las rutas que siguen las aeronaves se parecen más entre sí cuanto menor es la velocidad (aterrizando y despegando). Por este motivo una escala, variable con la velocidad que aumente la resolución cuando las velocidades son bajas podría mejorar las capacidades de identificación.

Finalmente podría desarrollarse una interfaz gráfica para evitar introducir los datos y analizar los resultados utilizando la consola de comandos.

6 BIBLIOGRAFÍA

[1] P. Senge, «Gerencia Estratégica,» Julio 2012. [En línea]. Available: https://www.uac.edu.co/images/stories/publicaciones/revistas_cientificas/dimension-empresarial/volumen-12-no-2/articulo08.pdf. [Último acceso: 20 Enero 2018].

[2] Usca, «Usca,» [En línea]. Available: https://www.usca.es/profesion/historia-del-control-aereo/. [Último acceso: 2018 Enero 21].

[3] M. Rolfe, «Nats,» [En línea]. Available: https://www.nats.aero/discover/. [Último acceso: 20 Enero 2018].

[4] Aeropuertos en Red, «Aeropuertosenred,» 20 Julio 2012. [En línea]. Available: http://www.aeropuertosenred.com/noticias/aeropuerto-malaga/nuevas-fichas-de-progresion-de-vuelo-electronica/. [Último acceso: 3 Febrero 2018].

[5] F. Escartí, «El secreto de los pajaros,» 25 Agosto 2014. [En línea]. Available: https://elsecretodelospajaros.net/2014/08/25/historias-en-el-aire-el-iran-air-655-y-los-misiles-del-vincennes/. [Último acceso: 21 Enero 2018].

[6] 2orillas, «Las 2orillas,» 29 Noviembre 2018. [En línea]. Available: https://www.las2orillas.co/los-peores-accidentes-aereos-de-la-historia/. [Último acceso: 21 Enerno 2018].

[7] BBC Mundo Redacción, «BBC,» 18 Julio 2014. [En línea]. Available: http://www.bbc.com/mundo/noticias/2014/07/140717_datos_avion_malasia_ucrania_ac. [Último acceso: 25 Enero 2018].

[8] N. J. Nilsson, «Stanford University,» 13 Septiembre 2009. [En línea]. Available: https://ai.stanford.edu/~nilsson/QAI/qai.pdf. [Último acceso: 5 Febrero 2018].

[9] M. Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, Edinburgh: Pearson Education, 2005.

[10] D. Poole, A. Mackworth y R. Goebel, «ubc,» 02 septiembre 2002. [En línea]. Available: http://people.cs.ubc.ca/~poole/ci/ch1.pdf. [Último acceso: 02 05 2018].

[11] D. P. Cruz Ponce, Inteligencia artificial con aplicaciones a la ingeniería, México: Alfaomega, 2010, pp. 33-193.

[12] S. Puerto Andrade, «Calameo,» Marzo 2011. [En línea]. Available: http://es.calameo.com/read/00287981054d48962be99. [Último acceso: 8 Febrero 2018].

[13] J. J. Romero, C. Dafonte y F. J. Penousal, Inteligencia Artificial y Computación avanzada, Santiago de Compostela: Fundación Alfredo Brañas, 2007.

[14] J. Rozo y A. L. López, «Epistemología estratégica: introducción a un modelo conexionista de aprendizaje asociativo,» *cuatrimestral de psicología,* vol. I, nº 16, pp. 197-210, 1998.

[15] H. Demis, «deepmind,» 18 Octubre 2017. [En línea]. Available: https://deepmind.com/blog/alphago-zero-learning-scratch/. [Último acceso: 13 Febrero 2018].

[16] G. M. Carlos, «Escuela Superior de Informática,» 2011. [En línea]. Available: http://www.esi.uclm.es/www/cglez/downloads/docencia/2011_Softcomputing/LogicaDifusa.pdf. [Último acceso: 14 Febrero 2018].

[17] R. Lahoz-Beltrá, Bioinformática, simulación, vida artificial e inteligencia artificial, Madrid: Díaz De Santos, 2004.

[18] P. Larrañaga, I. Inza y M. Abdelmalik, «Universidad del País Vasco,» 28 Marzo 2007. [En línea]. Available: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf. [Último acceso: 2018 Febrero 18].

[19] V. G. García Vilchez, «Universidad de Granada,» Septiembre 2010. [En línea]. Available: http://ceres.ugr.es/~alumnos/esclas/. [Último acceso: 24 Febrero 2018].

[20] C. A. Ruíz y M. S. Basualdo, «Universidad de Granada,» Marzo 2001. [En línea]. Available: ftp://decsai.ugr.es/pub/usuarios/castro/Material-Redes-Neuronales/Libros/matich-redesneuronales.pdf. [Último acceso: 2018 02 18].

[21] J. Hawkins y S. Blakeslee, On Intelligence, Nueva York: Henry Holt and Company, 2005.

[22] J. Hawkins, «Numenta,» [En línea]. Available: https://numenta.com. [Último acceso: 18 02 2018].

[23] J. Hawkins, «Numenta,» 12 Septiembre 2011. [En línea]. Available: https://numenta.com/resources/papers-videos-and-more/resources/hierarchical-temporal-memory-white-paper/. [Último acceso: 22 Febrero 2018].

[24] Numenta, «NuPIC,» 6 Enero 2018. [En línea]. Available: https://github.com/numenta/nupic. [Último acceso: 14 Enero 2018].

[25] J. Hawkins, «Numenta,» 2017. [En línea]. Available: http://nupic.docs.numenta.org/prerelease/guides/anomaly-detection.html. [Último acceso: 20 Febrero 2018].

[26] J. Hawkins, «Numenta,» [En línea]. Available: https://numenta.com/assets/pdf/whitepapers/Geospatial%20Tracking%20White%20Paper.pdf. [Último acceso: 20 Febrero 2018].

[27] C. Sunpur, «Chetansunpur,» 2014. [En línea]. Available: http://chetansurpur.com/slides/2014/8/5/geospatial-encoder.html#7. [Último acceso: 21 Febrero 2018].

[28] M. Ulla y P. Raja, «sagepub,» 2001. [En línea]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=5AA7F44DDC3998F15C2154F9D375B2C7?doi=10.1.1.668.7765&rep=rep1&type=pdf. [Último acceso: 2018 Febrero 21].

[29] Indra, «Indracompany,» 02 Enero 2013. [En línea]. Available: https://www.indracompany.com/sites/default/files/indra-air-traffic-control-automation-system.pdf. [Último acceso: 21 Febrero 2018].

[30] T. Moreno Casas y J. M. Gómez-Pastrana López, «sict.fail,» 08 Junio 2018. [En línea]. Available: http://sict.fail/Master%20Enginyeria%20Aeronautica/Sistema%20SACTA.pdf. [Último acceso: 21 Febrero 2018].

[31] Enaire, «Enaire,» [En línea]. Available: https://www.enaire.es/servicios/_atm/sistemas_de_gestion_del_transito_aereo_atm/sacta. [Último acceso: 21 Febrero 2018].

[32] Numenta, «HTM-Moclu,» 20 Noviembre 2015. [En línea]. Available: https://github.com/antidata/htm-moclu. [Último acceso: 21 Febrero 2018].

[33] M. Luchetta y A. Pagadizabal, «Devpost,» 9 Noviembre 2015. [En línea]. Available: https://devpost.com/software/air-traffic-anomaly-detector. [Último acceso: 21 Febrero 2018].

[34] NATO Comunication and Information Agency, «NATO,» [En línea]. Available: https://www.ncia.nato.int/NPC/Pages/products/Integrated-Command-and-Control-Software-for-Air-Operations-ICC.aspx. [Último acceso: 22 Febrero 2018].

[35] Global Security, «GlobalSecurity,» 31 Mayo 2015. [En línea]. Available: https://www.globalsecurity.org/military/systems/aircraft/systems/iff.htm. [Último acceso: 22 Febrero 2018].

[36] C. A. Pérez-Seone, «Cud uvigo,» 2016. [En línea]. Available: http://calderon.cud.uvigo.es/handle/11621/82. [Último acceso: 28 Enero 2018].

[37] S. Martín Marco, «Cud uvigo,» 2017. [En línea]. Available: http://calderon.cud.uvigo.es/handle/11621/192. [Último acceso: 30 Enero 2018].

[38] Sra International, Multilateration & ADS-B, 2009.

[39] Flightaware, «Flightaware,» [En línea]. Available: https://es.flightaware.com/. [Último acceso: 4 Febrero 2018].

[40] FlightRadar24, «FlightRadar24,» [En línea]. Available: https://www.flightradar24.com/how-it-works. [Último acceso: 22 Febrero 2018].

[41] Planefinder, «Planefinder,» [En línea]. Available: https://planefinder.net/. [Último acceso: 3 Febrero 2018].

[42] J. M. Rebés, «Aire,» 24 Noviembre 2013. [En línea]. Available: http://www.aire.org/radares/. [Último acceso: 22 Febrero 2018].

[43] Airservice Australia, «Airservice Australia,» 18 Septiembre 2015. [En línea]. Available: http://www.airservicesaustralia.com/projects/ads-b/how-ads-b-works/. [Último acceso: 22 Febrero 2018].

[44] Juan, «Ubuntu-guia,» 20 Abril 2014. [En línea]. Available: http://www.ubuntu-guia.com/2011/01/comando-apt-get-en-ubuntu.html. [Último acceso: 10 Enero 2018].

[45] Python Software Foundation, «Python,» 2018. [En línea]. Available: https://pypi.python.org/pypi/pip. [Último acceso: 14 Enero 2018].

[46] B. Straub, «Git,» 2018. [En línea]. Available: https://git-scm.com/book/es/v1/Empezando-Instalando-Git. [Último acceso: 14 Enero 2018].

[47] M. Taylor, «NuPIC Geospatial,» 9 Agosto 2017. [En línea]. Available: https://github.com/numenta/nupic.geospatial. [Último acceso: 15 Enero 2018].

ANEXO I: RUN_NEW.PY

```
1. #!/usr/bin/env python
2. # ----------------------------------------------------------------------
3. # Numenta Platform for Intelligent Computing (NuPIC)
4. # Copyright (C) 2013, Numenta, Inc.  Unless you have an agreement
5. # with Numenta, Inc., for a separate license for this software code, the
6. # following terms and conditions apply:
7. #
8. # This program is free software: you can redistribute it and/or modify
9. # it under the terms of the GNU Affero Public License version 3 as
10.      # published by the Free Software Foundation.
11.      #
12.      # This program is distributed in the hope that it will be useful,
13.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
14.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15.      # See the GNU Affero Public License for more details.
16.      #
17.      # You should have received a copy of the GNU Affero Public License
18.      # along with this program.  If not, see http://www.gnu.org/licenses.
19.      #
20.      # http://numenta.org/licenses/
21.      # ----------------------------------------------------------------------
   --
22.      import os
23.      import sys
24.      from optparse import OptionParser
25.
26.      from tools.preprocess_data import preprocess
27.      from tools.anomaly_to_js_data import postprocess
28.      from model.geospatial_anomaly_new import runGeospatialAnomaly
29.
30.      DEFAULT_OUTPUT_DIR = "output"
31.      verbose = False
32.      scriptDir = os.path.dirname(os.path.realpath(__file__))
33.
34.
35.      parser = OptionParser(
36.        usage="%prog <path/to/input/file> [options]\n\nRun NuPIC on
   specified "
37.              "location file, which should already be in the proper format "
38.              "(downloaded from the simulator)."
39.      )
40.
41.      parser.add_option(
42.        "-m",
43.        "--manual-sequence",
44.        action="store_true",
45.        default=False,
46.        dest="manualSequence",
47.        help="Automatically breaks into sequences based upon time gaps."
48.      )
49.      parser.add_option(
50.        "-t",
51.        "--time-encoders",
52.        action="store_true",
53.        default=False,
54.        dest="useTimeEncoders",
55.        help="Adds time of day encoder to model params."
56.      )
```

```
57.       parser.add_option(
58.         "-v",
59.         "--verbose",
60.         action="store_true",
61.         default=False,
62.         dest="verbose",
63.         help="Print debugging statements."
64.       )
65.       parser.add_option(
66.         "-o",
67.         "--output-dir",
68.         default=DEFAULT_OUTPUT_DIR,
69.         dest="outputDir",
70.         help="Where to write the output file."
71.       )
72.       parser.add_option(
73.         "-s",
74.         "--scale",
75.         default=False,
76.         dest="scale",
77.         help="Meter resolution for Geospatial Coordinate Encoder (default
    5m)."
78.       )
79.
80.       def run(inputPath, outputDir, useTimeEncoders, scale, autoSequence):
81.
82.         outputPath = os.path.abspath(outputDir)
83.         if not os.path.exists(outputPath):
84.           os.makedirs(outputPath)
85.
86.         preProcessedOutputPath = os.path.join(outputPath, "preprocessed_data
    .csv")
87.         if verbose: print "Pre-processing %s..." % inputPath
88.         preprocess(inputPath, preProcessedOutputPath, verbose=verbose)
89.
90.         anomalyOutputPath = os.path.join(outputPath, "anomaly_scores.csv")
91.         if verbose: print "Running NuPIC on %s..." % preProcessedOutputPath
92.         runGeospatialAnomaly(preProcessedOutputPath,
93.                              anomalyOutputPath,
94.                              scale=scale,
95.                              autoSequence=autoSequence,
96.                              useTimeEncoders=useTimeEncoders,
97.                              verbose=verbose)
98.
99.         visualizationOutputPath = os.path.join(scriptDir, "static/js/data.js
    ")
100.        if verbose: print "Creating visualization at %s..." %
    visualizationOutputPath
101.        postprocess(anomalyOutputPath, visualizationOutputPath)
102.
103.
104.     if __name__ == "__main__":
105.       (options, args) = parser.parse_args(sys.argv[1:])
106.       try:
107.         input_path = args.pop(0)
108.       except IndexError:
109.         parser.print_help(sys.stderr)
110.         sys.exit()
111.
112.       verbose = options.verbose
113.     scale=2000
114.     outputDir = "/home/adrian/nupic.geospatialapp/output/"
115.     run(input_path,
```

```
116.      outputDir,
117.      options.useTimeEncoders,
118.      scale,
119.      not options.manualSequence)
```

ANEXO II: IDENTIFICACION.PY

```
1. #!/usr/bin/env python
2. #-*- coding: utf-8 -*-
3. # ----------------------------------------------------------------------
4. # Numenta Platform for Intelligent Computing (NuPIC)
5. # Copyright (C) 2013, Numenta, Inc.  Unless you have an agreement
6. # with Numenta, Inc., for a separate license for this software code, the
7. # following terms and conditions apply:
8. #
9. # This program is free software: you can redistribute it and/or modify
10.      # it under the terms of the GNU Affero Public License version 3 as
11.      # published by the Free Software Foundation.
12.      #
13.      # This program is distributed in the hope that it will be useful,
14.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
15.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
16.      # See the GNU Affero Public License for more details.
17.      #
18.      # You should have received a copy of the GNU Affero Public License
19.      # along with this program.  If not, see http://www.gnu.org/licenses.
20.      #
21.      # http://numenta.org/licenses/
22.      # ----------------------------------------------------------------------
   --
23.      import os
24.      import sys
25.      import csv
26.      from optparse import OptionParser
27.
28.      from tools.preprocess_data import preprocess
29.      from tools.postprocess import postprocess
30.      from model.geospatial_anomaly_identificacion import runGeospatialAnoma
   ly
31.      from model.geospatial_anomaly_recargar import runGeospatialAnomalycarg
   ar
32.
33.
34.      DEFAULT_OUTPUT_DIR = "output"
35.      verbose = False
36.      scriptDir = os.path.dirname(os.path.realpath(__file__))
37.
38.
39.      parser = OptionParser(
40.        usage="%prog <path/to/input/file> [options]\n\nRun NuPIC on
   specified "
41.              "location file, which should already be in the proper format "
42.              "(downloaded from the simulator)."
43.      )
44.
45.      parser.add_option(
46.        "-m",
47.        "--manual-sequence",
48.        action="store_true",
49.        default=False,
50.        dest="manualSequence",
51.        help="Automatically breaks into sequences based upon time gaps."
52.      )
53.      parser.add_option(
```

```
54.         "-t",
55.         "--time-encoders",
56.         action="store_true",
57.         default=False,
58.         dest="useTimeEncoders",
59.         help="Adds time of day encoder to model params."
60.       )
61.       parser.add option(
62.         "-v",
63.         "--verbose",
64.         action="store_true",
65.         default=False,
66.         dest="verbose",
67.         help="Print debugging statements."
68.       )
69.       parser.add_option(
70.         "-o",
71.         "--output-dir",
72.         default=DEFAULT_OUTPUT_DIR,
73.         dest="outputDir",
74.         help="Where to write the output file."
75.       )
76.       parser.add_option(
77.         "-s",
78.         "--scale",
79.         default=False,
80.         dest="scale",
81.         help="Meter resolution for Geospatial Coordinate Encoder (default
    5m)."
82.       )
83.
84.
85.       def run(inputPath, outputDir, useTimeEncoders, scale, autoSequence):
86.
87.         outputPath = os.path.abspath(outputDir)
88.         if not os.path.exists(outputPath):
89.           os.makedirs(outputPath)
90.
91.         preProcessedOutputPath = os.path.join(outputPath, "preprocessed_data
    .csv")
92.         if verbose: print "Pre-processing %s..." % inputPath
93.         preprocess(inputPath, preProcessedOutputPath, verbose=verbose)
94.
95.         # anomalyOutputPath = os.path.join(outputPath, "anomaly_scores.csv")
96.         if verbose: print "Running NuPIC on %s..." % preProcessedOutputPath
97.         runGeospatialAnomaly(preProcessedOutputPath,
98.                              outputPath,
99.                              scale=scale,
100.                             autoSequence=autoSequence,
101.                             useTimeEncoders=useTimeEncoders,
102.                             verbose=verbose)
103.
104.        # visualizationOutputPath = os.path.join(scriptDir,
    "static/js/data.js")
105.        # if verbose: print "Creating visualization at %s..." %
    visualizationOutputPath
106.        # postprocess(anomalyOutputPath, visualizationOutputPath)
107.
108.      def runcargar(inputPath, outputDir, useTimeEncoders, scale, autoSequen
    ce):
109.
110.        outputPath = os.path.abspath(outputDir)
111.        if not os.path.exists(outputPath):
```

```
112.         os.makedirs(outputPath)
113.
114.       preProcessedOutputPath = os.path.join(outputPath, "preprocessed_data
   .csv")
115.       # if verbose: print "Pre-processing %s..." % inputPath
116.       # preprocess(inputPath, preProcessedOutputPath, verbose=verbose)
117.       outputPath = os.path.abspath(outputDir)
118.       anomalyOutputPath = os.path.join(outputPath, "anomaly scores" +list[
   0]+".csv")
119.       if verbose: print "Running NuPIC on %s..." % preProcessedOutputPath
120.       runGeospatialAnomalycargar(preProcessedOutputPath,
121.                                  anomalyOutputPath,
122.                                  scale=scale,
123.                                  autoSequence=autoSequence,
124.                                  useTimeEncoders=useTimeEncoders,
125.                                  verbose=verbose)
126.
127.       # visualizationOutputPath = os.path.join(scriptDir,
   "static/js/data.js")
128.       # if verbose: print "Creating visualization at %s..." %
   visualizationOutputPath
129.       # postprocess(anomalyOutputPath, visualizationOutputPath)
130.
131.
132.     if __name__ == "__main__":
133.       (options, args) = parser.parse_args(sys.argv[1:])
134.       try:
135.         input_path = args.pop(0)
136.       except IndexError:
137.         parser.print_help(sys.stderr)
138.         sys.exit()
139.
140.       verbose = options.verbose
141.     scale=2000
142.     outputDir = "/home/adrian/Escritorio/tfg/app/reconocimiento"
143.     run(
144.         input_path,
145.         outputDir,
146.         options.useTimeEncoders,
147.         scale,
148.         not options.manualSequence)
149.     with open ("flightanomaly", 'r') as infile:
150.       lista=[]
151.       list=[]
152.       mejorporcentaje = csv.reader(infile)
153.       for row in mejorporcentaje:
154.         list += [str(row[1])]
155.         lista += [float(row[0])]
156.
157.     respuesta= str()
158.     print "El vuelo con un mayor parecido es", list[0], "con una similitud
   del",(round((1-lista[0])*100)),"%"
159.
160.     while respuesta != 'si' and respuesta != 'no':
161.       print "¿quieres entrenar la red con este vuelo?"
162.       respuesta = str(raw_input(""))
163.
164.     if respuesta == 'si':
165.      runcargar(
166.         input_path,
167.         outputDir,
168.         options.useTimeEncoders,
169.         scale,
```

```
170.          not options.manualSequence)
171.       print 'Modelo cargado'
172.      respuesta = str()
173.      while respuesta != 'si' and respuesta != 'no':
174.        print "¿quieres representar este vuelo?"
175.        respuesta = str(raw_input(""))
176.      if respuesta == 'si':
177.        outputPath = os.path.abspath(outputDir)
178.        anomalyOutputPath = os.path.join(outputPath, "anomaly_scores" +list[
    0]+".csv")
179.        visualizationOutputPath = os.path.join(scriptDir, "static/js/data.js
    ")
180.        if verbose: print "Creating visualization at %s..." %
    visualizationOutputPath
181.        postprocess(anomalyOutputPath, visualizationOutputPath)
```

ANEXO III: POSTPROCESS_DATA.PY

```
1. #!/usr/bin/env python
2. # ----------------------------------------------------------------------
3. # Numenta Platform for Intelligent Computing (NuPIC)
4. # Copyright (C) 2014, Numenta, Inc.  Unless you have purchased from
5. # Numenta, Inc. a separate commercial license for this software code, the
6. # following terms and conditions apply:
7. #
8. # This program is free software: you can redistribute it and/or modify
9. # it under the terms of the GNU Affero Public License version 3 as
10.      # published by the Free Software Foundation.
11.      #
12.      # This program is distributed in the hope that it will be useful,
13.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
14.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15.      # See the GNU Affero Public License for more details.
16.      #
17.      # You should have received a copy of the GNU Affero Public License
18.      # along with this program.  If not, see http://www.gnu.org/licenses.
19.      #
20.      # http://numenta.org/licenses/
21.      # ----------------------------------------------------------------------
   --
22.
23.      import csv
24.      import sys
25.
26.
27.
28.      def postprocess(dataPath, outPath):
29.        with open(dataPath) as csvfile:
30.          reader = csv.reader(csvfile)
31.
32.          with open(outPath, 'w') as out:
33.            out.write("DATA = [\n")
34.            reader.next()  # header
35.
36.            for row in reader:
37.              timestamp = row[0]
38.              longitude = float(row[1])
39.              latitude = float(row[2])
40.              speed = float(row[3])
41.              anomalyScore = float(row[4])
42.              newSequence = "true" if int(row[5]) else "false"
43.              out.write("[new Date(\"{0}\"), {1}, {2}, {3}, {4},
   {5}],\n".format(
44.                timestamp, longitude, latitude, speed, anomalyScore, newSequ
   ence))
45.
46.            out.write("]\n")
47.
48.
49.
50.      if __name__ == "__main__":
51.        if len(sys.argv) < 2:
52.          print ("Usage: {0} "
53.                 "/path/to/data.csv "
54.                 "/path/to/outfile.csv").format(sys.argv[0])
55.          sys.exit(0)
56.
```

```
57.         dataPath = sys.argv[1]
58.         outPath  = sys.argv[2]
59.         postprocess(dataPath, outPath)
```

ANEXO IV: GEOSPATIAL_ANOMALY_IDENTIFICACION.PY

```
1. #!/usr/bin/env python
2. # ----------------------------------------------------------------------
3. # Numenta Platform for Intelligent Computing (NuPIC)
4. # Copyright (C) 2014, Numenta, Inc.  Unless you have an agreement
5. # with Numenta, Inc., for a separate license for this software code, the
6. # following terms and conditions apply:
7. #
8. # This program is free software: you can redistribute it and/or modify
9. # it under the terms of the GNU Affero Public License version 3 as
10.      # published by the Free Software Foundation.
11.      #
12.      # This program is distributed in the hope that it will be useful,
13.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
14.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15.      # See the GNU Affero Public License for more details.
16.      #
17.      # You should have received a copy of the GNU Affero Public License
18.      # along with this program.  If not, see http://www.gnu.org/licenses.
19.      #
20.      # http://numenta.org/licenses/
21.      # ----------------------------------------------------------------------
   --
22.
23.      """
24.      A simple client to create a CLA anomaly detection model for geospatial
   data.
25.      """
26.
27.      import csv
28.      import datetime
29.      import sys
30.      import os
31.
32.      from nupic.frameworks.opf.model_factory import ModelFactory
33.      import model_params
34.
35.
36.
37.      DEFAULT_DATA_PATH = "data/commute.csv"
38.      DEFAULT_OUTPUT_PATH = "anomaly_scores.csv"
39.
40.      def runGeospatialAnomaly(dataPath, outputPath,
41.                               scale=False,
42.                               autoSequence=True,
43.                               useTimeEncoders=False,
44.                               verbose=False):
45.        listaaprendizaje = os.listdir("/home/adrian/nupic.geospatialapp/apre
   ndizaje")
46.        d={}
47.        for i in listaaprendizaje:
48.                clave=str(i.split("tmpDir")[1])
49.
50.                savePath = "/home/adrian/nupic.geospatialapp/aprendizaje/tmp
   Dir"+ str(clave)
51.
52.                model = ModelFactory.loadFromCheckpoint(savePath)
53.                anomalyOutputPath = os.path.join(outputPath, "anomaly_scores
   " + str(clave)+".csv")
```

```
54.
55.                 model.disableLearning()
56.
57.                 with open (dataPath) as fin:
58.                   reader = csv.reader(fin)
59.                   csvWriter = csv.writer(open(anomalyOutputPath,"wb"))
60.                   csvWriter.writerow(["timestamp",
61.                                       "longitude",
62.                                       "latitude",
63.                                       "speed",
64.                                       "anomaly_score",
65.                                        "new_sequence",
66.                                        "media",
67.                                        "rumbo"])
68.
69.                   reader.next()
70.                   reader.next()
71.                   reader.next()
72.
73.                   contador = 0
74.                   media = 0
75.                   anomaliatotal = 0
76.
77.                   lastTimestamp = None
78.                   lastTrackName = None
79.                   outputFormat = "%Y-%m-%dT%H:%M:%S"
80.
81.                   for _, record in enumerate(reader, start=1):
82.                     trackName = record[0]
83.                     timestamp = datetime.datetime.fromtimestamp(int(record[1
   ]))
84.                     longitude = float(record[3])
85.                     latitude = float(record[2])
86.                     speed = float(record[5])
87.                     altitude = float(record[4])
88.                     rumbo = float(record[6])
89.
90.
91.
92.                     if autoSequence:
93.                       if trackName != lastTrackName:
94.                                 newSequence = True
95.                       else:
96.                                  newSequence = False
97.                     lastTimestamp = timestamp
98.                     lastTrackName = trackName
99.
100.                    if newSequence:
101.                      if verbose:
102.                        print "Starting new sequence..."
103.                      model.resetSequenceStates()
104.
105.                    modelInput = {
106.                      "vector": (speed, longitude, latitude, altitude)
107.                     }
108.
109.                    modelInput["course"] = rumbo
110.
111.                    if useTimeEncoders:
112.                      modelInput["timestamp"] = timestamp
113.
114.                    result = model.run(modelInput)
```

```
115.                    anomalyScore = result.inferences["anomalyScore"] # Esto
   nos da el nivel de anomalia comparando con el resultado
116.                    contador = contador + 1
117.                    anomaliatotal = anomaliatotal + anomalyScore
118.                    media = (anomaliatotal) / contador
119.                    csvWriter.writerow([timestamp.strftime(outputFormat),
120.                                        longitude,
121.                                        latitude,
122.                                        speed,
123.                                        anomalyScore,
124.                                        1 if newSequence else 0,
125.                                        media])
126.
127.                    if verbose:
128.                      print "[{0}] - Anomaly score: {1}.".format(timestamp,
129.              anomalyScore)
130.
131.                print "Anomaly scores have been written to
   {0}".format(outputPath)
132.                d[media]=clave
133.        first = 0
134.        for i in sorted(d.keys()):
135.          if first <1:
136.            first= first+1
137.            csvWriter = csv.writer(open('flightanomaly',"w"))
138.            csvWriter.writerow([i,
139.                                d[i]])
140.
141.          else:
142.            csvWriter = csv.writer(open('flightanomaly',"a"))
143.            csvWriter.writerow([i, d[i]])
144.
145.
146.
147.      if __name__ == "__main__":
148.        dataPath = DEFAULT_DATA_PATH
149.        outputPath = DEFAULT_OUTPUT_PATH
150.
151.        if len(sys.argv) > 1:
152.          dataPath = sys.argv[1]
153.
154.        if len(sys.argv) > 2:
155.          outputPath = sys.argv[2]
156.
157.        runGeospatialAnomaly(dataPath, outputPath)
```

ANEXO V: GEOPATIAL_ANOMALY_RELOAD.PY

```
1. #!/usr/bin/env python
2. # ----------------------------------------------------------------------
3. # Numenta Platform for Intelligent Computing (NuPIC)
4. # Copyright (C) 2014, Numenta, Inc.  Unless you have an agreement
5. # with Numenta, Inc., for a separate license for this software code, the
6. # following terms and conditions apply:
7. #
8. # This program is free software: you can redistribute it and/or modify
9. # it under the terms of the GNU Affero Public License version 3 as
10.      # published by the Free Software Foundation.
11.      #
12.      # This program is distributed in the hope that it will be useful,
13.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
14.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15.      # See the GNU Affero Public License for more details.
16.      #
17.      # You should have received a copy of the GNU Affero Public License
18.      # along with this program.  If not, see http://www.gnu.org/licenses.
19.      #
20.      # http://numenta.org/licenses/
21.      # ----------------------------------------------------------------------
   --
22.
23.      """
24.      A simple client to create a CLA anomaly detection model for geospatial
   data.
25.      """
26.      import csv
27.      import datetime
28.      import sys
29.      import os
30.
31.      from nupic.frameworks.opf.model_factory import ModelFactory
32.      import model_params
33.
34.
35.      DEFAULT_DATA_PATH = "data/commute.csv"
36.      DEFAULT_OUTPUT_PATH = "anomaly_scores.csv"
37.
38.      def runGeospatialAnomalycargar(dataPath, outputPath,
39.                                     scale=False,
40.                                     autoSequence=True,
41.                                     useTimeEncoders=False,
42.                                     verbose=False):
43.        listaaprendizaje = os.listdir("/home/adrian/nupic.geospatialapp/apre
   ndizaje")
44.        with open ("flightanomaly", 'r') as infile:
45.          list=[]
46.          mejorporcentaje = csv.reader(infile)
47.          for row in mejorporcentaje:
48.            list += [str(row[1])]
49.
50.        savePath = "/home/adrian/nupic.geospatialapp/aprendizaje/tmpDir"+ st
   r(list[0])
51.        model = ModelFactory.loadFromCheckpoint(savePath)
52.
53.        with open (dataPath) as fin:
54.          reader = csv.reader(fin)
```

```
55.
56.
57.           lastTimestamp = None
58.           lastTrackName = None
59.           outputFormat = "%Y-%m-%dT%H:%M:%S"
60.
61.           for _, record in enumerate(reader, start=1):
62.             trackName = record[0]
63.             timestamp = datetime.datetime.fromtimestamp(int(record[1]))
64.             longitude = float(record[3])
65.             latitude = float(record[2])
66.             speed = float(record[5])
67.             altitude = float(record[4])
68.             rumbo = float(record[6])
69.
70.
71.
72.             if autoSequence:
73.               if trackName != lastTrackName:
74.                 newSequence = True
75.               else:
76.                 newSequence = False
77.                 lastTimestamp = timestamp
78.                 lastTrackName = trackName
79.
80.             if newSequence:
81.                   if verbose:
82.                     print "Starting new sequence..."
83.                     model.resetSequenceStates()
84.
85.                   modelInput = {
86.                     "vector": (speed, longitude, latitude, altitude)
87.                     }
88.
89.                   modelInput["course"] = rumbo
90.
91.             if useTimeEncoders:
92.               modelInput["timestamp"] = timestamp
93.
94.             result = model.run(modelInput)
95.             anomalyScore = result.inferences["anomalyScore"] # Esto nos da
    el nivel de anomalia comparando con el resultado
96.
97.             if verbose:
98.               print "[{0}] - Anomaly score: {1}.".format(timestamp,
99.                                                           anomalyScore)
100.
101.       print "Anomaly scores have been written to {0}".format(outputPath)
102.       model.save(savePath)
103.
104.
105.     if __name__ == "__main__":
106.       dataPath = DEFAULT_DATA_PATH
107.       outputPath = DEFAULT_OUTPUT_PATH
108.
109.       if len(sys.argv) > 1:
110.         dataPath = sys.argv[1]
111.
112.       if len(sys.argv) > 2:
113.         outputPath = sys.argv[2]
114.
115.       runGeospatialAnomaly(dataPath, outputPath)
```

ANEXO VI: PREPROCESS_DATA.PY

```
1. #!/usr/bin/env python
2. # ----------------------------------------------------------------------
3. # Numenta Platform for Intelligent Computing (NuPIC)
4. # Copyright (C) 2014, Numenta, Inc.  Unless you have purchased from
5. # Numenta, Inc. a separate commercial license for this software code, the
6. # following terms and conditions apply:
7. #
8. # This program is free software: you can redistribute it and/or modify
9. # it under the terms of the GNU Affero Public License version 3 as
10.      # published by the Free Software Foundation.
11.      #
12.      # This program is distributed in the hope that it will be useful,
13.      # but WITHOUT ANY WARRANTY; without even the implied warranty of
14.      # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15.      # See the GNU Affero Public License for more details.
16.      #
17.      # You should have received a copy of the GNU Affero Public License
18.      # along with this program.  If not, see http://www.gnu.org/licenses.
19.      #
20.      # http://numenta.org/licenses/
21.      # ----------------------------------------------------------------------
   --
22.      """
23.      Takes in a datafile generated by Grok Hound app.
24.
25.      - Strips out non-commutes
26.      - Sub-samples readings to a minimum resolution of 10-seconds
27.      """
28.
29.      import csv
30.      import datetime
31.      import sys
32.
33.
34.
35.      def preprocess(dataPath, outPath, verbose=False):
36.        with open(dataPath) as csvfile:
37.          reader = csv.reader(csvfile)
38.          writer = csv.writer(open(outPath, "wb"))
39.
40.          lastTimestamp = None
41.          lastTimestampKept = None
42.          numStationary = 0
43.
44.          for row in reader:
45.            timestamp = datetime.datetime.fromtimestamp(int(row[1]))
46.            speed = float(row[5])
47.            print timestamp
48.
49.            keep = True
50.
51.
52.
53.            if (lastTimestampKept and
54.                (timestamp - lastTimestampKept).total_seconds() < 5):
55.              keep = False
56.            if speed <= 0.5:
57.              numStationary += 1
58.            else:
```

```
59.               numStationary = 0
60.
61.             if numStationary > 30:
62.               keep = False
63.               print "velocidad"
64.
65.             lastTimestamp = timestamp
66.
67.             if keep:
68.               if verbose:
69.                 print "Keeping row:\t{0}".format(row)
70.               lastTimestampKept = timestamp
71.               writer.writerow(row)
72.
73.             else:
74.               if verbose:
75.                 print "Discarding row:\t{0}".format(row)
76.
77.
78.
79.       if __name__ == "__main__":
80.         if len(sys.argv) < 3:
81.           print ("Usage: {0} "
82.                  "/path/to/data.csv
    /path/to/outfile.csv").format(sys.argv[0])
83.           sys.exit(0)
84.
85.         dataPath = sys.argv[1]
86.         outPath = sys.argv[2]
87.         preprocess(dataPath, outPath
```

ANEXO VII: SERVER.PY

```
#! /usr/bin/env python
# ----------------------------------------------------------------------
# Numenta Platform for Intelligent Computing (NuPIC)
# Copyright (C) 2014, Numenta, Inc.  Unless you have an agreement
# with Numenta, Inc., for a separate license for this software code, the
# following terms and conditions apply:
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero Public License version 3 as
    # published by the Free Software Foundation.
    #
    # This program is distributed in the hope that it will be useful,
    # but WITHOUT ANY WARRANTY; without even the implied warranty of
    # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
    # See the GNU Affero Public License for more details.
    #
    # You should have received a copy of the GNU Affero Public License
    # along with this program.  If not, see http://www.gnu.org/licenses.
    #
    # http://numenta.org/licenses/
    # ----------------------------------------------------------------------
  --
    import os

    from flask import Flask, request

    from tools.preprocess_data import preprocess
    from model.geospatial_anomaly_load import runGeospatialAnomaly
    from tools.anomaly_to_js_data import postprocess



    app = Flask(__name__)

    DIR_OUTPUT           = "output"
    FILE_DATA            = "data.csv"
    FILE_PROCESSED_DATA  = "processed_data.csv"
    FILE_MODEL_OUTPUT    = "model_output.csv"

    DIR_STATIC_JS        = os.path.join("static", "js")
    FILE_JS_DATA         = "data.js"



    @app.route('/')
    def visualize():
      return app.send_static_file('visualize.html')


    @app.route('/simulate')
    def simulate():
      return app.send_static_file('simulate.html')


    @app.route('/process', methods=['POST'])
    def process():
      dataFile = os.path.join(DIR_OUTPUT, FILE_DATA)
      processedDataFile = os.path.join(DIR_OUTPUT, FILE_PROCESSED_DATA)
```

```
58.         modelOutputFile = os.path.join(DIR_OUTPUT, FILE_MODEL_OUTPUT)
59.         jsDataFile = os.path.join(DIR_STATIC_JS, FILE_JS_DATA)
60.
61.         with open(dataFile, 'w') as f:
62.           f.write(request.data)
63.
64.         preprocess(dataFile, processedDataFile)
65.         runGeospatialAnomaly(processedDataFile, modelOutputFile, verbose=Tru
    e)
66.         postprocess(modelOutputFile, jsDataFile)
67.
68.         return "Done."
69.
70.
71.       @app.route('/js/<path:path>')
72.       def js(path):
73.         return app.send_static_file(os.path.join('js', path))
74.
75.
76.       @app.route('/css/<path:path>')
77.       def css(path):
78.         return app.send_static_file(os.path.join('css', path))
79.
80.
81.       @app.route('/img/<path:path>')
82.       def img(path):
83.         return app.send_static_file(os.path.join('img', path))
84.
85.
86.       # Disable cache
87.       @app.after_request
88.       def add_header(response):
89.         response.cache_control.max_age = 0
90.         return response
91.
92.
93.       if __name__ == "__main__":
94.         if not os.path.exists(DIR_OUTPUT):
95.           os.makedirs(DIR_OUTPUT)
96.
97.         app.run(debug=True, host="0.0.0.0")
```

ANEXO VIII: CONVERSOR.PY

```
#!/usr/bin/env python

import csv
import datetime
import sys
import os
import time

os.chdir('/home/adrian/Descargas/vuelos/iberia_535')
      listavuelos = os.listdir(os.getcwd())

      d={}
      for i in listavuelos:
          clave=int((i.split('(')[1]).split(')')[0],16)
          d[clave]=i

      for i in sorted(d.keys()):
          with open (d[i]) as fin:
              reader = csv.reader(fin)
              reader.next()
              for _, record in enumerate(reader, start=1):
                  #print record[1]
                  #timestamp= int(time.mktime(time.strptime(record[1],"%Y-%m-%dT%H:%M:%SZ"))) para transformar en timestamp
                  #print hex(i),",",record[0],",",record[3].split(',')[0],record[3].split(',')[1],record[4],",", record[5],",", record[6]
                  csvWriter = csv.writer(open('flight',"a"))
                  csvWriter.writerow([hex(i),
                                      record[0],
                                      record[3].split(',')[0],
                                      record[3].split(',')[1],
                                      record[4],
                                      record[5],
                                      record[6]])
```