ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

EVALUACIÓN DE UN ALGORITMO DE INTELIGENCIA ARTIFICIAL APLICADO A LA PREDICCIÓN DEL CONSUMO ELÉCTRICO

(Analysis of a power consumption forecasting method based on artificial intelligence)

Para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Autor: Marcos Romero Castaño

Octubre - 2014

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Marcos Romero Castaño
Director del PFC: Adolfo Cobo García

Título: "Evaluación de un algoritmo de inteligencia artificial aplicado a la predicción del consumo eléctrico"
Title: "Analysis of a power consumption forecasting method based on artificial intelligence"

Presentado a examen el día: 27 de Octubre del 2014

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Mañana Canteli, Mario
Secretario (Apellidos, Nombre): Cobo García, Adolfo
Vocal (Apellidos, Nombre): Lanza Calderón, Jorge

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del PFC (sólo si es distinto del Secretario)
Vº Bº del Subdirector	Proyecto Fin de Carrera Nº (a asignar por Secretaría)

Trabajo realizado en parte con los medios aportados por el proyecto SENSORES DE FIBRA ÓPTICA PARA SEGURIDAD Y PROTECCIÓN (TEC2013-47264-C2-1-R)

A Adolfo por su ayuda, su paciencia y por animarme a estudiar esta carrera.

A mis compañeros por apoyarme en todo momento.

A mi padre, por todo lo que le debo. Sin él no lo habría conseguido.

Índice

IV. EVALUACIÓN DEL ALGORITMO

V. RESUMEN, CONCLUSIONES Y LÍNEAS FUTURAS

I. INTRODUCCIÓN

I 1. Contexto

La evolución tecnológica ha sido muy importante en los últimos años. Cada vez son más los procesos que dejamos en manos de máquinas, ya no sólo en el sector industrial donde la mano de obra se ha sustituido en su totalidad por automatismos que son capaces de mejorar la producción en prácticamente todos los aspectos, sino que a día de hoy prácticamente todo está conectado a un ordenador. Estos automatismos y ordenadores son en su mayoría lo que se denominan como sistemas expertos que son sistemas informáticos o robóticos cuyo software está programado para realizar una tarea concreta de la forma más eficiente posible.

El problema de estos sistemas expertos reside en que no son capaces de tener en cuenta las diferentes variantes de circunstancias, matices o eventos que se pueden producir en el proceso o en su entorno. Es decir, que estos sistemas no son flexibles y no son capaces de imitar la actividad de un ser humano a la hora de resolver problemas por lo que son muchos los campos de la industria y de la vida en general que no pueden recurrir a estos sistemas.

La solución a esto pasa por el uso de redes neuronales, uno de los múltiples campos de la inteligencia artificial. Se trata de un sistema que intenta modelar el funcionamiento de nuestras neuronas, capaces de colaborar entre sí para producir un estímulo de salida, aprendiendo de patrones comunes y reaccionando ante eventos inesperados. Son éstas propiedades las que hacen de las redes neuronales una herramienta muy útil para la industria en el campo de la predicción, un campo que empieza a cobrar mucha importancia en la industria eléctrica.

La electricidad es indispensable y de una importancia estratégica para la economía de los países. En consecuencia las eléctricas están haciendo grandes esfuerzos para poder equilibrar la energía generada en función de la demanda con el objetivo dar un mejor servicio y ofrecer un precio competitivo. Con este propósito se recurre a la predicción para intentar estimar el consumo que se va a demandar en un futuro y así generar la energía necesaria evitando sobre costes. El problema es que la demanda eléctrica depende de muchos factores, como pueden ser el día de la semana, si es festivo o laboral, la estación del año, invierno o verano, o incluso el clima en un momento puntual como una oleada de frío en pleno agosto.

Todos estos factores hacen de la predicción una tarea compleja por lo que se hace necesaria la utilización de redes neuronales capaces de tener en cuenta y reaccionar ante los distintos eventos que puedan producirse generando una respuesta capaz de ajustarse a la realidad sea esta esperada o no.

I 2. Motivación

Como se ha mencionado es importante tener una herramienta que sea capaz de predecir la carga de consumo eléctrico que se va a generar en un futuro. Cometer errores en dicha predicción puede llevar a problemas graves de suministro ya que en algunos casos se tendrá una carencia de energía, lo que supone cortes y apagones, y en otros casos producirá un exceso de energía generada. Ambos escenarios conllevan a un encarecimiento de la energía eléctrica lo que supone una lastra para un mercado altamente competitivo. Es por ello que las

empresas invierten en el desarrollo e investigación de técnicas de predicción cada vez más eficaces con el fin de abaratar al máximo posible los gastos de electricidad.

Existen varios métodos que se han ido aplicando a lo largo de los años para realizar esta estimación como la regresión lineal, modelos ARIMA o modelos de descomposición. El problema de estos métodos es que son puramente estadísticos, hacen su análisis sobre series históricas de consumo y no son capaces de dar respuesta ante eventos inesperados. Por ello surge la necesidad de utilizar redes neuronales capaces de reaccionar ante incidencias no esperadas y de aprender nuevos patrones dando una predicción más exacta.

Podemos definir una red neuronal como un sistema que trata de modelar los detalles arquitecturales del neocórtex que es la base del pensamiento inteligente en el cerebro de los mamíferos. Existen distintos tipos de redes neuronales diferenciadas en función del marco teórico en el que se basan y en la utilidad para la cual hayan sido diseñadas. Uno de estos tipos es la memoria temporal jerárquica o HTM (del inglés *"Hierarchical Temporal Memory"*) que nace de la teoría desarrollada por Jeff Hawkins [1] en la que explica cómo el cerebro se basa en la predicción para desenvolverse en el mundo y seguir aprendiendo.

Siguiendo con esta teoría y con la intención de emularla en los ordenadores se funda Numenta [2] en el 2006, empresa destinada a ser un impulsor en el campo emergente de las máquinas inteligentes, y desarrolla Grok, un software basado en la teoría de las HTMs y que comercializa desde entonces. En junio del 2013 Numenta publica un software de código abierto derivado de su producto Grok llamado NuPIC [3] (*Numenta Platform for Intelligent Computing*) con el objetivo de crear y apoyar una comunidad interesada en la inteligencia artificial y en el aprendizaje automático basado en la teoría desarrollada por Jeff Hawkins sobre el neocórtex.

Aprovechando la disponibilidad de este software de código abierto en el presente proyecto se estudia y analiza el potencial de dicha herramienta aplicándolo al uso concreto de la predicción del consumo eléctrico.

I 3. Objetivos

El objetivo de este proyecto consiste en evaluar el software de código abierto NuPIC desarrollado por Numenta aplicándolo al uso concreto de la predicción del consumo eléctrico y a la detección de anomalías.

Para ello se instalará en un sistema operativo Linux el software NuPIC, cuyo entorno de programación está desarrollado en lenguaje Python [4], al que se le proporcionarán varios meses de datos del consumo eléctrico generado en la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad de Cantabria durante los años 2012/13 en hojas de cálculo con formato CSV [5], y se analizará su eficacia tanto en el campo de la predicción como en la detección de anomalías.

I 4. Organización del documento

Este documento se divide en las siguientes secciones:

- Descripción de los fundamentos biológicos y teóricos de las HTMs.
- Algoritmo de aprendizaje cortical o CLA (del inglés *"Cortical Learning Algorithm"*) desarrollado por Numenta, NuPIC, y su instalación.
- Evaluación del algoritmo con una serie de ejemplos de predicción del consumo eléctrico y de detección de anomalías.
- Conclusión y líneas futuras, donde se resumen los resultados obtenidos a lo largo de este trabajo y se exponen líneas futuras de investigación sobre este mismo.

II. FUNDAMENTOS DEL ALGORITMO

En este apartado se explican los fundamentos tanto teóricos como prácticos de los algoritmos que se encuentran en el software que se ha utilizado para el desarrollo de este proyecto. Prácticamente la mayoría de la información que aquí se presenta ha sido sacada del CLA White Paper [6], escrito por Numenta. Tanto la información presentada en el CLA White Paper como en este proyecto trata de dar una visión general de la estructura de la memoria temporal jerárquica o HTM necesaria para entender los fundamentos del algoritmo. Como se ha mencionado anteriormente, el algoritmo trata de emular el funcionamiento del neocórtex en nuestro cerebro, sin embargo en este proyecto no se va a profundizar en el marco teórico de la neurobiología ya que no se considera necesario. Si se desea profundizar en la teoría de las HTMs se recomienda la lectura del libro ON Intelligence [7], escrito por Jeff Hawkins y Sandra Blakeslee.

II 1. Visión general de la HTM

La memoria temporal jerárquica (HTM) es una tecnología de aprendizaje de máquina que pretende capturar las propiedades estructurales y algorítmicas del neocórtex, la base del pensamiento inteligente de nuestro cerebro. La visión de alto nivel, escuchar, tocar, mover, el lenguaje y la planificación son llevadas a cabo por el neocórtex. Contando con esta diversidad de funciones cognitivas, se podría esperar que el neocórtex implementara la misma diversidad de algoritmos neuronales. Este no es el caso. El neocórtex muestra una gran uniformidad de patrones en su circuitería neuronal. La evidencia biológica sugiere que el neocórtex implementa un grupo común de algoritmos para llevar a cabo diferentes funciones relacionadas con la inteligencia.

HTM proporciona un marco teórico para entender el neocórtex y sus múltiples capacidades. A diferencia de la programación estándar, los programas HTM son entrenados mediante la exposición a un flujo de datos sensoriales. En gran medida las capacidades del HTM son determinadas según los datos a los que ha sido expuesto.

Aunque antes hemos definido las HTM como un tipo de red neuronal no es algo del todo cierto. Las HTM modelan neuronas, a las que nos referiremos como células, las cuales son organizadas en columnas, capas, regiones y en una jerarquía. Estos detalles son importantes y hacen que las HTM sean lo que podemos denominar como una nueva forma de red neuronal.

Tal como indica su nombre, HTM es fundamentalmente un sistema basado en la memoria. Las redes HTM son entrenadas con muchos tipos de datos variados y confían en el almacenamiento de grandes grupos de patrones y secuencias. La memoria HTM tiene una organización jerárquica inherentemente basada en el concepto del tiempo. La información siempre es guarda de manera distribuida. El usuario especifica el tamaño de la jerarquía y en qué entrenar al sistema, pero el HTM controla dónde y cómo se almacena la información.

A continuación se describen con detalle las funciones clave de las HTMs: la jerarquía, el tiempo y las representaciones distribuidas dispersas.

II 1.1 Jerarquía

Una red HTM consiste en regiones organizadas en una jerarquía. La región es la unidad principal de memoria y predicción en un HTM y cada una representa un nivel en la jerarquía. A medida que se va ascendiendo en la jerarquía, la información converge. Además debido a las conexiones de retroalimentación, ésta también puede fluir hacia abajo.

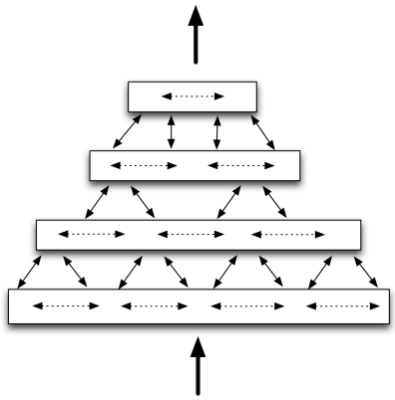

Figura II.1 – Diagrama de cuatro regiones HTM organizadas en una jerarquía de cuatro niveles.

El beneficio de la organización jerárquica es la eficiencia. Reduce significativamente el tiempo de entrenamiento y el uso de la memoria, ya que los patrones aprendidos en cada nivel de la jerarquía son reutilizados cuando se combinan de manera distinta en los niveles superiores. Esto significa que para aprender un nuevo objeto de alto nivel no hace falta reaprender sus componentes más básicas. Por ejemplo para aprender una nueva palabra no hace falta reaprender las letras, sílabas o fonemas.

Compartir representaciones en una jerarquía conlleva conseguir la generalización del comportamiento esperado. Cuando ves un nuevo animal, si ves una boca y dientes, predecirás que el animal come con su boca y que puede ser que te muerda. La jerarquía permite que un nuevo objeto en el mundo herede las propiedades conocidas de sus subcomponentes.

Existe una solución de compromiso entre cuánta memoria reservada a cada nivel y cuantos niveles son necesarios para el aprendizaje en una jerarquía HTM. Las HTM aprenden automáticamente las mejores posibles representaciones en cada nivel conociendo las estadísticas de los datos de entrada y el número de recursos reservados. Si se reserva más memoria a un nivel, ese nivel formará representaciones mayores y más complejas, lo cual indica que podrían hacer falta menos niveles jerárquicos. Si se reserva menos memoria, el nivel formará representaciones más pequeñas y simples, lo que significa que podrían ser necesarios más niveles jerárquicos.

En resumen, las jerarquías reducen el tiempo de entrenamiento, reducen el uso de memoria, e introducen una nueva forma de generalización. No obstante cabe destacar que muchos problemas simples que conllevan predicción pueden ser resueltos con una única región HTM.

II 1.2 Regiones

Las regiones HTM se componen de varias celdas (células a partir de ahora), las cuales están organizadas en una matriz bidimensional de columnas. Cada columna conecta con una parte de la entrada de datos y cada célula conecta con otras células en la región. Algunas regiones reciben la entrada de información directamente de los sensores y otras regiones sólo lo hacen después de que esta haya pasado antes por varias otras regiones. Es la conectividad entre regiones la que define la jerarquía.

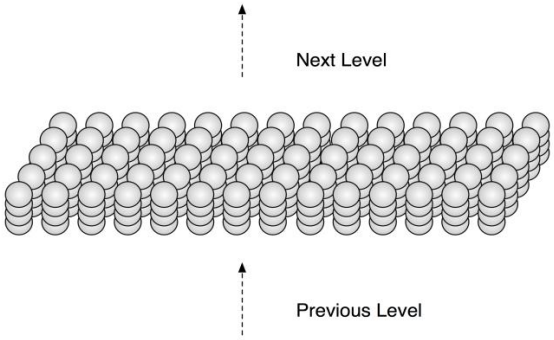

Figura II.2 – Sección de una región HTM de cuatro células por columna.

II 1.3 Representaciones distribuidas dispersas

A pesar de que las neuronas del neocórtex están altamente interconectadas, las neuronas inhibidoras garantizan que sólo un pequeño porcentaje de las neuronas estén activas en cada momento. Por lo tanto, la información en el cerebro siempre es representada por un pequeño porcentaje de neuronas activas. Éste tipo de codificación se denomina "representación distribuida dispersa". Dispersa significa que sólo un pequeño porcentaje de neuronas están activas en cada momento. Distribuida significa que las activaciones de muchas neuronas son requeridas para poder representar algo.

Las regiones HTM también utilizan las representaciones distribuidas dispersas, de hecho, los mecanismos de la memoria dentro de una región HTM dependen de su uso. La entrada a una región HTM es siempre una representación distribuida, pero puede que no sea dispersa, por lo tanto lo primero que hace una región HTM es convertir estos datos de entrada en una representación distribuida dispersa.

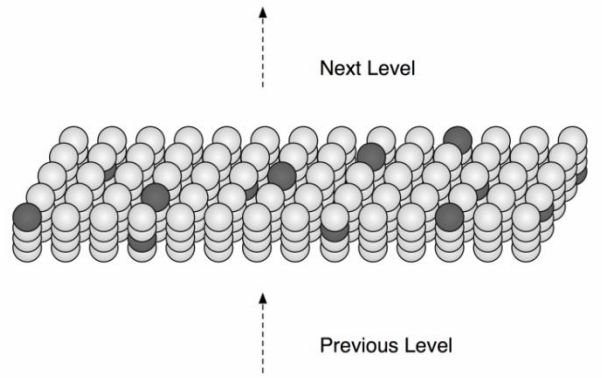

Figura II.3 – Activación celular distribuida dispersa en una región HTM.

II 1.4 El rol del tiempo

El tiempo juega un rol crucial en el aprendizaje, inferencia y predicción. Para poder aprender, todos los sistemas HTM tienen que ser expuestos a entradas que varíen en el tiempo durante el entrenamiento. Hay que hacer notar que no es suficiente con que las entradas sensoriales vayan cambiando en el tiempo. Una sucesión de patrones sensoriales sin relación generaría sólo confusión. Las entradas que cambian con el tiempo tienen que venir de una misma fuente. Si queremos entrenar un HTM para reconocer patrones de temperatura, vibración y ruido provenientes de sensores de una planta de producción de electricidad, el HTM tendrá que ser entrenado con datos de estos sensores cambiando el tiempo.

Típicamente, una red HTM necesita ser entrenada con muchos datos. El trabajo de los algoritmos HTM es aprender las secuencias temporales de una serie de datos de entrada, como por ejemplo construir un modelo para saber que patrones siguen a que patrones. El trabajo es difícil ya que puede ser que no sepa cuándo empiezan y terminan las secuencias, puede haber secuencias superpuestas ocurriendo al mismo tiempo, el aprendizaje tiene que ocurrir continuamente, y el aprendizaje tiene que ocurrir en la presencia de ruido.

Aprender y reconocer secuencias es la base de la formación de las predicciones. Una vez que un HTM aprende que patrones son los que siguen a otros patrones, puede predecir los que se supone serán los siguientes conociendo la entrada actual y las inmediatamente anteriores.

II 2. Aprendizaje

Una región HTM aprende sobre su mundo encontrando patrones y luego secuencias de patrones en la información sensorial. La región no sabe que representa la información de entrada, trabaja en un campo puramente estadístico. Busca combinaciones de entrada que ocurren juntas normalmente, lo que llamamos patrones espaciales. Entonces busca en qué secuencia aparecen estos patrones en el tiempo, lo que llamamos patrones temporales o secuencias.

Una simple región HTM tiene una capacidad de aprendizaje limitada. Una región automáticamente ajusta lo que aprende basándose en cuánta memoria tiene y la complejidad de la entrada que está recibiendo. Los patrones espaciales aprendidos por una región necesariamente se simplificarán si la memoria asignada una región ha sido reducida. Los patrones espaciales aprendidos se complicaran si la memoria asignada es incrementada.

Igual que un sistema biológico, los algoritmos de aprendizaje en una región HTM son capaces de aprender en tiempo real. No existe la necesidad de separar la fase de aprendizaje de la inferencia, a pesar de que esta última vaya mejorando con el entrenamiento. A medida que los patrones de la entrada vayan cambiando, la región HTM cambiará gradualmente también.

Después del entrenamiento inicial, un HTM puede continuar aprendiendo o puede ser desactivado. Otra opción es desactivar el aprendizaje sólo en los niveles más bajos de la jerarquía pero continuar aprendiendo en los niveles superiores. Una vez que un HTM ha aprendido la estructura estadística básica, casi todo el aprendizaje nuevo ocurre en los niveles superiores de la jerarquía. Si un HTM es expuesto a nuevos patrones que contengan nuevas estructuras en los niveles bajos, le llevará más tiempo al HTM aprender estos nuevos patrones.

II 3. Inferencia

Una vez que un HTM ha aprendido los patrones, puede desarrollar la inferencia sobre entradas nuevas. Cuando un HTM recibe una entrada, lo mapeará a patrones espaciales y temporales aprendidos con anterioridad. Mapear satisfactoriamente nuevas entradas a secuencias guardadas previamente es la esencia de la inferencia y el mapeo de patrones.

Una región HTM, en un proceso de inferencia, está constantemente mirando a flujos de entrada y mapeándolos a secuencias aprendidas previamente. Una región HTM puede encontrar mapeos desde el principio de la secuencia o desde cualquier ubicación.

En una región HTM las entradas puede que no sean exactamente las mismas jamás. Consecuentemente, al igual que en nuestro cerebro, una región HTM debe manejar nuevas entradas durante la inferencia y el entrenamiento. Una de las maneras en las que una región HTM gestiona las nuevas entradas es mediante el uso de las representaciones distribuidas dispersas. Una propiedad clave de las representaciones distribuidas dispersas es que sólo hace falta que coincide una porción del patrón para estar seguros de que el mapeo es relevante.

II 4. Predicción

Cada región HTM almacena secuencias de patrones. Mapeando secuencias almacenadas con las nuevas entradas, una región forma una predicción sobre las entradas que vendrán a continuación. Las regiones HTM guardan transiciones entre representaciones distribuidas dispersas. En algunas instancias las transiciones pueden parecerse a secuencias lineales pero en general varias posibles futuras entradas suelen ser predichas al mismo tiempo. La mayoría de la memoria en un HTM se dedica a la memoria secuencial, guardando transiciones entre patrones espaciales.

A continuación se exponen algunas propiedades importantes de la predicción en las HTM:

- **La predicción es continua.**
  A pesar de no ser conscientes, estamos prediciendo constantemente. Cuando escuchamos una canción, tratamos de predecir la siguiente nota. Cuando estás bajando unas escaleras, estas prediciendo que tu pie tocara el siguiente escalón. En una región HTM ocurre lo mismo.

- **La predicción ocurre en cada región y a cada nivel de la jerarquía.**
  Si tienes una jerarquía de regiones HTM, la predicción ocurrida a cada nivel. Las regiones harán predicciones sobre los patrones que han aprendido. En un ejemplo del lenguaje, los niveles inferiores pueden predecir los posibles siguientes fonemas, mientras que las regiones superiores pueden predecir palabras o frases.

- **Las predicciones son sensibles al contexto.**
  Las predicciones están basadas en lo que ha ocurrido en el pasado, así como en lo que está ocurriendo en este mismo momento. De esta manera, una entrada producirá diferentes predicciones basándose en el contexto previo. Una región HTM aprende a utilizar tanto contexto previo como el que le haga falta y puede mantener el contexto tanto para periodos cortos o largos de tiempo.

- **La predicción conduce a la estabilidad.**
  La salida de una región es su predicción. Una de las propiedades de las HTMs es que las salidas de las regiones cambian más despacio y aguantan más en el tiempo, es decir, se estabilizan en cuanto más altos estén en la jerarquía. Una región no predice que pasará inmediatamente después. Si puede, predecirá varios pasos en el futuro.

- **Una predicción nos dice si una nueva entrada es esperada o inesperada.**
  Ya que cada región predice que va a ocurrir a continuación, detecta cuando algo inesperado ha ocurrido. Puede que no sea capaz de predecir exactamente qué pasará a continuación, pero si la siguiente entrada no coincide con ninguna de las predicciones la región HTM sabrá que ha ocurrido una anomalía.

- **La predicción ayuda a hacer el sistema más robusto al ruido.**
  Cuando un HTM predice lo que parece que ocurrirá, la predicción puede polarizar al sistema a inferir lo predicho. Esta predicción ayuda al sistema a añadir los datos que falten, ayudando de esta manera a inferir incluso en la presencia de ruido.

II 5. Algoritmos de aprendizaje cortical HTM

Cada región HTM evalúa su entrada y busca patrones comunes y a continuación aprende secuencias de esos patrones. A partir de su memoria de secuencias, cada región hace predicciones. Para llevar a cabo este proceso el algoritmo realiza tres tareas principales:

- Formar una representación distribuida de los datos de entrada.
- Formar una representación de la entrada sobre el contexto de las entradas previas.
- Formar una predicción basada en la entrada actual y el contexto de las entradas previas.

II 5.1 Formar una representación distribuida dispersa de los datos de entrada

A la entrada a una región llegan un gran número de bits. En cualquier momento en el tiempo algunas de estas entradas están activas (valor 1) y otras inactivas (valor 0). La primera acción que acomete la región HTM es convertir esta entrada en una nueva representación que sea dispersa. Por ejemplo, la entrada podría tener el 40% de los datos de entrada en “on” pero la nueva representación tendrá sólo el 20% de sus bits en “on”. Puede parecer que este proceso genera una gran pérdida de información ya que el número posible de patrones de entrada es mucho mayor que el número de posibles representaciones en la región. De todas maneras, los dos números son increíblemente grandes. Las entradas vistas por la región serán una fracción minúscula de todas las posibles entradas. La pérdida de información teórica no tiene ningún efecto práctico.

Una región HTM está compuesta a nivel lógico por un grupo de columnas, que a su vez están compuestas por una o varias células. Cada columna dentro de una región está conectada a un subgrupo único de bits de entrada. Como resultado, diferentes patrones de entrada resultan en distintos niveles de activación de las columnas. Las columnas con la activación más fuerte inhiben, o desactivan, las columnas con las activaciones más débiles. La representación distribuida de la entrada será codificada según las columnas que están activas y las que son inactivas después de la inhibición. La función de inhibición está definida para lograr que un

porcentaje relativamente pequeño de columnas quede activa, incluso cuando el número de bits de entrada que es activo varias significativamente.

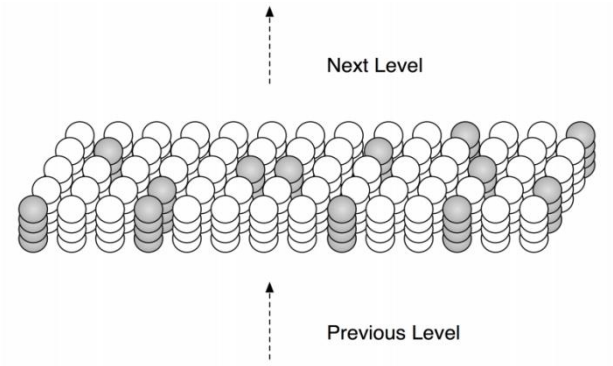

Figura II.4 – Representación distribuida dispersa después de la fase de inhibición mostrando las columnas activas en gris claro.

Aprender las conexiones para cada columna desde un subgrupo de entradas, determinar el nivel de entradas de cada columna y usar la inhibición para seleccionar un grupo disperso de columnas activas, son los pasos que conforman el "agrupador espacial". El término indica que los patrones que son espacialmente similares están agrupados, es decir, significa que comparten un gran número de bits activos y que son agrupados en una representación común.

II 5.2 Formar una representación de la entrada en el contexto de las entradas previas

La siguiente función que ejecuta la región es convertir la representación en columnas de la entrada en una nueva representación que incluya el estado, o contexto, del pasado. La nueva representación es formada activando un subgrupo de células dentro de cada columna, típicamente sólo una célula por columna. Codificar una entrada de manera diferente en contextos diferentes es una cualidad universal de la percepción y de la acción y es una de las funciones más importantes de una región HTM.

Cada columna en una región HTM está constituida por varias células. Todas las células en una columna obtienen la misma entrada de alimentación. Cada célula en cada columna puede estar activa o no. Seleccionando diferentes células activas en cada columna activa, podemos representar exactamente la misma entrada de maneras distintas en contextos distintos.

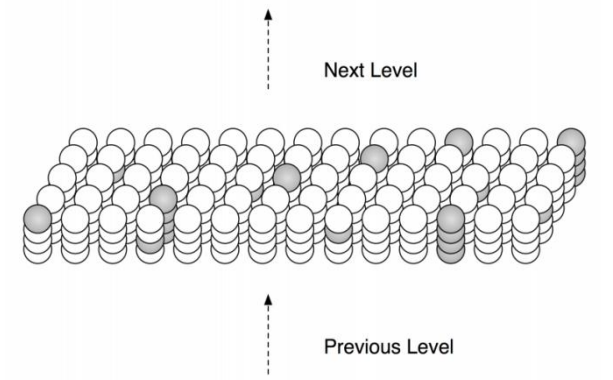

Figura II.5 – Representación distribuida dispersa con columnas con una célula activa y columnas con todas sus células activas.

La regla general utilizada por una región HTM es la siguiente. Cuando una columna se activa, mira en todas las células de la columna. Si una o más células de la columna están ya en

un estado predictivo, sólo esas células pasan a activarse. Si en la columna no había células en estado predictivo, entonces todas las células se activan. Es decir, si un patrón de entrada es esperado entonces el sistema confirma esa expectativa activando sólo las células en el estado predictivo. Si el patrón de entrada es inesperado, el sistema activa todas las células en la columna. Si no existe un estado previo, y por lo tanto ni contexto de predicción, todas las células en la columna se activaran cuando ésta se active. Si hay un estado previo pero la entrada no concuerda con lo esperado, todas las células en la columna activa se activarán.

II 5.3 Formar una predicción basada en la entrada actual y el contexto de las entradas previas

Cuando una región predice, activa todas las células que presumiblemente se activarán debido a la entrada de alimentación futura. Como todas las representaciones en una región son distribuidas, se pueden realizar múltiples predicciones al mismo tiempo.

Una región predice de la siguiente manera. Cuando los patrones de entrada cambian en el tiempo, diferentes grupos de columnas y células se activan en secuencia. Cuando una célula se activa, forma conexiones a un subgrupo de células cercanas que estaban activas en el momento inmediatamente anterior. Estas conexiones se pueden formar de manera rápida o lenta dependiendo de la velocidad de aprendizaje requerida por la aplicación. A continuación todo lo que tiene que hacer la célula es mirar a todas esas conexiones para encontrar actividad coincidente. Si las conexiones se activan, la célula puede esperar su próxima activación y entra en estado predictivo. La activación por alimentación de un grupo de células llevará a la activación predictiva de otros grupos de células.

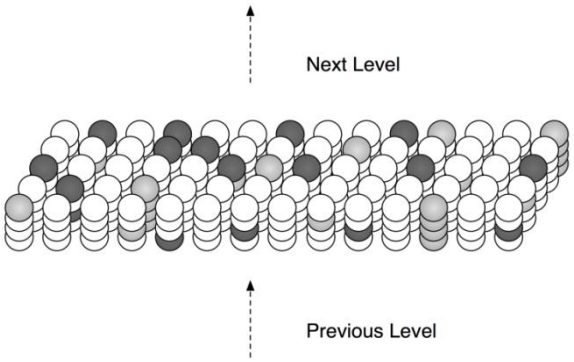

Figura II.6 – Región HTM con células activas debido a la entrada de retroalimentación positiva, en gris claro y células en estado predictivo, en gris oscuro.

Las predicciones de una región HTM pueden ser para varios pasos en el futuro. Usando melodías como ejemplo, una región HTM no sólo predice la siguiente nota, sino que predecirá las cuatro siguientes. Esta manera de funcionar proporciona una propiedad muy útil. La salida de una región, es decir la unión de todas las células activas y predictivas en la región, cambia más despacio que la entrada.

Se utiliza el término “agrupador temporal” para describir los dos pasos correspondientes a añadir contexto a la representación y a la predicción. Mediante la creación de salidas para las secuencias de patrones que cambian lentamente, en esencia estamos agrupando diferentes patrones que se siguen en el tiempo.

El aprendizaje en el agrupador espacial y temporal es similar. En ambos casos aprender significa establecer conexiones entre las células. El agrupador temporal aprende sobre las conexiones entre las células de la misma región mientras que el agrupador espacial aprende

sobre la entrada de alimentación entre los bits de entrada y las columnas. Explicamos con más detalle sus funciones.

II 6. Conceptos del agrupador espacial

La función fundamental del agrupador espacial es convertir la entrada a una región en un patrón disperso. Esta función es importante, ya que el mecanismo utilizado para aprender y predecir las secuencias requiere empezar con patrones distribuidos dispersos. Para ello el agrupador espacial trata de cumplir los siguientes objetivos.

- **Utilizar todas las columnas.**
  Una región HTM tiene un número fijo de columnas que aprenden a representar patrones comunes. Uno de los objetivos es asegurar que todas las columnas aprenden a representar algo útil independientemente del número de columnas que tengas. Por ello se mantiene un registro de cuantas veces se ha activado una columna respecto a sus vecinas. Si la actividad relativa de una columna es muy baja, estimula su nivel de actividad en la entrada hasta que empieza a tomar parte en el grupo de columnas activas.

- **Mantener la densidad deseada.**
  Una región necesita formar una representación distribuida de su entrada. Las columnas con el mayor número de entradas inhiben a sus vecinas. Dentro del radio de inhibición, sólo se permite a un porcentaje de las columnas con las entradas más activas ser las ganadoras. Las columnas restantes son deshabilitadas.

- **Evitar los patrones triviales.**
  El objetivo se alcanza estableciendo un umbral mínimo para que la columna se active, lo que garantiza un cierto nivel de complejidad del patrón que está representando.

- **Evitar conexiones extras.**
  Si una columna forma demasiadas conexiones válidas puede acabar respondiendo fuertemente a muchos patrones de entrada no relacionados. Para evitar este problema, se disminuye el valor de permanencia de cualquier conexión que no esté contribuyendo a una columna ganadora. De esta manera se garantiza que una columna representa un número limitado de patrones de entrada.

II 6.1 Detalles del agrupador espacial

A continuación enumeramos los pasos que realiza el agrupador espacial:

1. El agrupador espacial empieza con una entrada consistente en un número fijo de bits. Estos bits pueden representar datos sensoriales o pueden venir de una región inferior de la jerarquía.

2. Asigna un número fijo de columnas a la región que está recibiendo esta entrada. Cada columna tiene asociada un segmento predictivo. Cada segmento predictivo

tiene un grupo de conexiones potenciales representando un subgrupo de bits de entrada. Cada conexión potencial cuenta con un valor de permanencia. Basado en el valor de permanencia, algunas de las conexiones potenciales serán válidas.

3. Para cualquier entrada, determina cuántas conexiones válidas en cada columna están conectadas a bits de entrada activos.

4. El número de conexiones activas es multiplicado por un factor estimulador determinado dinámicamente dependiendo de cuántas veces ha estado activa la columna respecto a sus vecinas.

5. Las columnas con las activaciones más altas después de la estimulación deshabilitan todas las columnas menos un porcentaje prefijado en el radio de inhibición. El radio de inhibición es determinado dinámicamente por la dispersión de los bits de entrada.

6. Por cada una de las columnas activas, se ajustan los valores de permanencia de todas las conexiones potenciales. Los valores de permanencia de las conexiones alineadas con los bits activos de entrada son incrementadas, mientras que los valores de permanencia de las conexiones alineadas con bits inactivos de entrada son disminuidos. Los cambios en los valores de permanencia hacen que algunas conexiones pasen de ser válidas a inválidas y viceversa.

II 7. Conceptos del agrupador temporal

La función del agrupador temporal es aprender secuencias y predecir. El método básico consiste en que cuando una célula se activa, forma conexiones con otras células que estaban activas justo en el momento anterior. Las células pueden predecir cuándo se activarán mirando sus conexiones. Si todas las células lo hacen colectivamente pueden guardar y recordar secuencias, y pueden predecir qué ocurrirá a continuación. No existe un almacenamiento central para una secuencia de patrones, en su lugar la memoria está distribuida entre todas las células individuales. Como la memoria está distribuida, el sistema es muy robusto contra el ruido y el error.

II 7.1 Detalles del agrupador temporal

A continuación enumeramos los pasos que realiza el agrupador temporal. Éstos empiezan donde termino el agrupador espacial:

1. Por cada columna activa, busca las células en la columna que estén en estado predictivo, y las activa. Si no hay células en estado predictivo activa todas las células en la columna. El grupo resultante de células activas es la representación de la entrada en el contexto de la entrada anterior.

2. Por cada segmento predictivo en cada célula de la región, cuenta cuantas conexiones establecidas están conectadas a células activas. Si el número excede el umbral, ese segmento predictivo es marcado como activo. Las células con estos segmentos activos pasan al estado predictivo a no ser que estén ya activos debido a la entrada de alimentación. Las células sin segmento predictivo activo y las células no activas debido a la entrada de alimentación siguen inactivas.

3. Cuando un segmento predictivo se activa, modifica los valores de permanencia de todas las conexiones que estén asociadas a dicho segmento. Por cada conexión potencial en el segmento predictivo activo, se incrementa la permanencia de esas conexiones a las células inactivas. Estos cambios en la permanencia de las conexiones son marcadas como temporales.

4. En el momento en el que una célula pasa de estar inactiva a activa debido a la entrada de alimentación, recorremos cada conexión potencial asociada con la célula y eliminamos cualquier marca temporal. De esta manera sólo se actualiza la permanencia de las conexiones si se ha predicho correctamente la activación de la célula por la entrada de alimentación.

5. Cuando una célula cambia de cualquiera de los estados activos al inactivo, se deshacen todos los cambios de permanencia marcados como temporales por cada conexión potencial. De esta manera no se fortalece la permanencia de las conexiones que han predicho incorrectamente la activación de la célula. Sólo las células que están activas debido a alimentación propagan la actividad dentro de la región. Todas las células activas, por alimentación o las predictivas, forman la salida de la región y se propagan a la siguiente región en la jerarquía.

III. IMPLEMENTACIÓN DEL ALGORITMO

Este capítulo contiene el pseudocódigo detallado para una primera implementación de la función del agrupador espacial y del agrupador temporal. Al final de la descripción de cada pseudocódigo se incluye un apéndice con la explicación de las variables y estructuras utilizadas.

III 1. Implementación del agrupador espacial y pseudocódigo

La entrada de este código es una matriz binaria que sube desde los sensores o el nivel anterior. El código computa activeColumns(t) que es la lista de columnas que ganan debido a la entrada que sube en el momento t. Esta lista, que es la salida de la rutina del agrupador espacial, es enviada como entrada a la rutina del agrupador temporal. Se puede desconectar el aprendizaje simplemente saltando la fase 3.

El pseudocódigo está dividido en tres distintas fases que ocurren en secuencia:

Fase 1: computar la superposición con la entrada actual para cada columna.
Fase 2: computar las columnas ganadoras después de la inhibición.
Fase 3: actualizar la permanencia de la sinapsis y las variables internas.

III 1.1 Inicialización

La región es inicializada computando una lista de conexiones potenciales iniciales para cada columna antes de recibir cualquier entrada. Consiste de un grupo aleatorio de entradas seleccionadas del espacio de entradas. Cada entrada es representada por una conexión y se le asigna un valor de permanencia aleatorio. Los valores de permanencia aleatorios son elegidos con dos criterios. Primero, los valores son elegidos para que estén en un pequeño rango alrededor de connectedPerm (el mínimo valor para la permanencia para la cual la conexión se considera establecida). Esto permite que las conexiones potenciales se conecten, o desconecten, después de unas pequeñas iteraciones de entrenamiento. En segundo lugar, cada columna tiene un centro natural sobre la región de entrada y los valores de permanencia tienen una tendencia hacia este centro. Cuanto más cerca estén del centro más altos serán sus valores de permanencia.

III 1.2 Fase 1: Superposición

Dado un vector de entrada, la primera fase calcula la superposición de cada columna con ese vector. La superposición de cada columna es el número de conexiones establecidas con entradas activas, multiplicado por su respectivo estímulo. Si el valor es menor que minOverlap, establecemos el valor de superposición a cero.

```
1.      for c in columns
2.
3.              overlap(c) = 0
4.              for s in connectedSynapses(c)
```

```
5.                  overlap(c) = overlap(c) + input(t, s.sourceInput)
6.
7.                  if overlap(c) < minOverlap then
8.                          overlap(c) = 0
9.                  else
10.                         overlap(c) = overlap(c) * boost(c)
```

III 1.3 Fase 2: Inhibición

La segunda fase calcula que columnas quedan como vencedoras después de la fase de inhibición. El parámetro desiredLocalActivity controla el número de columnas que acaban ganando. Por ejemplo, si desiredLocalActivity es 10, una columna será la ganadora si su valor de superposición es mayor que el valor de la décima columna con el valor más alto en un radio de inhibición.

```
11.     for c in columns
12.
13.             minLocalActivity = kthScore(neighbors(c), desiredLocalActivity)
14.
15.             if overlap(c) > and overlap(c) ≥  minLocalActivity then
16.                     activeColumns(t).apennd(c)
17.
```

III 1.4 Fase 3: Aprendizaje

La tercera fase ejecuta el aprendizaje; actualiza los valores de la permanencia en todas las conexiones que la necesiten, así como los radios de estímulo de inhibición.

La regla de aprendizaje principal está implementada entre las líneas 20-26. Para las columnas ganadoras, si una conexión está activa, su valor de permanencia es incrementado, en otro caso es disminuido. Los valores de permanencia están limitados para estar entre el 0 y el 1.

Las líneas 28-36 implementan el estímulo. Existen dos mecanismos de estímulo separados para ayudar a las columnas a aprender las conexiones. Si una columna no gana lo suficiente (según se mide en activeDutyCycle), su valor de estímulo general será incrementado (líneas 30-32). Alternativamente, si las conexiones establecidas a una columna no encajan bien con ninguna entrada (según se mide en activeDutyCycle), sus valores de permanencia son estimulados (líneas 34-36). Finalmente, al final de la fase 3 el radio de inhibición es recalculado (líneas 38).

```
18.     for c in activeColumns(t)
19.
20.             for s in potentialSynapses(c)
21.                     if active(s) then
22.                             s.permanene += permanenceInc
23.                             s.permanene = min(1.0, s.permanence)
24.                     else
25.                             s.permanene -= permanenceInc
26.                             s.permanene = max(0.0, s.permanence)
```

```
27.
28.     for c in columns
29.
30.             minDutyCycle(c) = 0.01 * maxDutyCycle(neighbors(c))
31.             activeDutyCycle(c) = updateActiveDutyCycle(c)
32.             boost(c) = boostFunction(activeDutyCycle(c), minDutyCycle(c))
33.
34.             overlapDutyCycle(c) = updateOverlapDutyCycle(c)
35.             if overlapDutyCycle(c) < minDutyCyle(c) then
36.                     increasePermanences(c, 0.1 * connectedPerm)
37.
38.     inhibitionRadius = averageReceptiveFieldSize()
39.
```

III 1.5 Apéndice del agrupador espacial

columns	Lista de todas las columnas.
input(t,j)	La entrada a este nivel en el momento t, input(t,j) es 1 si la posición de entrada j está activada.
overlap(c)	La superposición del agrupador espacial de la columna c con un patrón de entrada particular.
activeColumns(t)	Lista de los índices de las columnas que han resultado ganadoras debido a la entrada de alimentación.
desiredLocalActivity	Un parámetro que controla el número de columnas que serán ganadoras después de la etapa de inhibición.
inhibitionRadius	Tamaño medio del campo receptivo de las columnas.
neighbors(c)	Lista de todas las columnas que están dentro del inhibitionRadius de la columna c.
minOverlap	El número mínimo de entradas que tienen que estar activas para que una columna pueda ser considerada en la etapa de inhibición.
boost(c)	El valor de estímulo de la columna c cuando está aprendiendo. Se utiliza para incrementar el valor de superposición para las columnas inactivas.
synapse	Una estructura de datos que representa una conexión. Contiene un valor de permanencia y el índice de la fuente de datos.
connectedPerm	Si el valor de permanencia para una conexión es mayor que este valor, se dirá que está establecida.

potentialSynapses(c) La lista de conexiones potenciales y sus valores de permanencia.

connectedSynapses(c) Un subgrupo de potentialSynapses(c), donde la permanencia es mayor que connectedPerm. Son las entradas de alimentación que están conectadas a la columna c en el momento actual.

permanenceInc Cantidad por la que se incrementan los valores de permanencia de las conexiones durante el aprendizaje.

permanenceDec Cantidad por la que se disminuyen los valores de permanencia de las conexiones durante el aprendizaje.

activeDutyCycle(c) Media variable que representa cuantas veces la columna c ha estado activa después de la inhibición.

overlapDutyCycle(c) Media variable que representa cuantas veces la columna c ha tenido una superposición significativa con sus entradas.

minDutyCycle(c) Variable que representa la tasa de disparo mínima deseada para una célula. Si la tasa de disparo cae por debajo de este valor, será estimulada. Este valor está calculado como el 1% de la tasa de disparo máxima de sus células vecinas.

Las siguientes rutinas de apoyo se usan en el código anterior.

kthScore(cols, k)

Recibiendo como entrada la lista de columnas, devuelve el k'esimo mayor valor de superposición.

updateActiveDutyCycle(c)

Calcula una media variable de cuantas veces la columna c ha sido activada después de la inhibición.

updateOverlapDutyCycle(c)

Calcula una media variable de cuantas veces la columna c tiene el valor de superposición mayor que minOverlap.

averageReceptiveFieldSize()

El radio medio del tamaño de campo receptivo conectado de todas las columnas. El tamaño de campo receptivo conectado de una columna incluye sólo las conexiones establecidas. Se utiliza para determinar el grado de inhibición lateral entre las columnas.

maxDutyCycle(cols)

Devuelve el máximo ciclo de servicio activo de las columnas pasándole la lista de las columnas.

increasePermanences(c, s)

Incrementa el valor de permanencia de cada conexión en la columna c por un factor de escala s.

boostFunctuion(c)

Devuelve el valor de estimulación de una columna. El valor de estimulación es un escalar >= 1. Si activeDutyCycle(c) está por encima de minDutyCycle(c), el valor de estimulación es 1. La estimulación se incrementa linealmente una vez el activeDutyCycle de la columna empieza a bajar por debajo de su minDutyCycle.

III 2. Implementación del agrupador temporal y pseudocódigo

Esta sección contiene el código detallado para una primera implementación de la función del agrupador temporal. La entrada para este código es activeColumns(t), después de ser computado por el agrupador espacial. El código computa el estado activo y predictivo para cada célula en el momento temporal actual t. El booleano OR de cada estado activo o predictivo en cada célula forma la salida del agrupador temporal para la siguiente etapa. Aunque la fase 3 es solo necesaria para el aprendizaje, las fases 1 y 2 contienen algunas partes específicas del aprendizaje.

El pseudocódigo está dividido en tres distintas fases que ocurren en secuencia:

Fase 1: computar el estado activo, activeState(t), para cada célula.
Fase 2: computar el estado predictivo, predictiveState(t), para cada célula.
Fase 3: actualizar las sinapsis.

Ya que el agrupador temporal es significativamente más complicado que el agrupador espacial, primero se expone la versión de solo inferencia, seguido de una versión que combina inferencia y aprendizaje. Al final del capítulo se ha añadido una descripción de algunos de los detalles de implementación, terminología y rutinas de soporte.

III 2.1 Pseudocódigo del agrupador temporal: Inferencia

Fase 1:

La primera fase calcula el estado activo de cada célula. Para cada columna ganadora determinamos que células deben activarse. Si la entrada de alimentación fuera predicha por cualquier célula, esto es, su predictiveState era 1 debido al segmento de secuencia en el paso del tiempo previo, entonces esas células se activan (líneas 4-9). Si la entrada de alimentación hubiera sido inesperada, cada célula en la columna se activa (líneas 11-13).

```
1.    for c in activeColumns(t)
2.
3.        buPredicted = false
4.        for i = 0 to cellsPermColunms - 1
5.            if predictiveState(c, i, t-1) == true then
6.                s = getActiveSegment(c, i, t-1, activeState)
7.                if s.sequenceSegment == true then
8.                    buPredicted = true
9.                    activeState(c, i, t) = 1
10.
11.       if buPredicted == false then
12.           for i = 0 to cellsPerColumn – 1
```

```
13.                                 activeState(c, i, t) = 1
```

Fase 2:

La segunda fase calcula el estado predictivo de cada célula. Una célula cambiará a "on" su predictiveState si cualquiera de sus segmentos se activa, esto es, si suficientes conexiones horizontales están actualmente disparando debido a la entrada de alimentación.

```
14.     for c, i in cells
15.             for s in segments(c, i)
16.                     if segmentActive(c, i, s, t) then
17.                             predictiveState(c, i, t) = 1
```

III 2.2 Pseudocódigo del agrupador temporal: Inferencia y aprendizaje

Fase 1:

La primera fase calcula el activeState para cada célula de una columna ganadora. Para esas columnas, el código selecciona una célula por columna como célula de aprendizaje (learnState). El funcionamiento es el siguiente: si la entrada de alimentación ha sido predicha por cualquier célula, esto es, su salida predictiveState es 1 debido a un segmento de secuencia, entonces esas células se activan (líneas 23-27). Si ese segmento se activa desde células con el learnState "on", esta célula es seleccionada como la célula de aprendizaje (líneas 28-30). Si la entrada de alimentación no ha sido predicha, entonces las células se activan (líneas 32-34). Adicionalmente, la célula que mejor encaje es seleccionada como la célula de aprendizaje (líneas 36-41) y un nuevo segmento es añadido a esa célula.

```
18.     for c in activeColumns(t)
19.
20.             buPredicted = false
21.             lcChosen = false
22.             for i = 0 to cellsPerColumn - 1
23.                     if predictiveState(c, i, t-1) == true then
24.                             s = getActiveSegment(c, i, t-1, activeState)
25.                             if s.sequenceSegment == true then
26.                                     buPredicted = true
27.                                     activeState(c, i, t) = 1
28.                                     if segmentActive(s, t-1, learnState) then
29.                                             lcChosen = true
30.                                             learnState(c, i, t) = 1
31.
32.             if buPredicted == false then
33.                     for i = 0 to cellsPerColumn - 1
34.                             activeState(c, i, t) = 1
35.
36.             if lcChosen == false then
37.                     l, s = getBestMatchingCell(c, t-1)
38.                     learnState(c, i, t) = 1
39.                     sUpdate = getSegmentActiveSynapses(c, i, s, t-1, true)
```

```
40.                 sUpdate.sequenceSegment = true
41.                 segmentUpdateList.add(sUpdate)
```

Fase 2:

La segunda fase calcula el estado predictivo para cada célula. Una célula activará su estado predictivo si uno de sus segmentos se activa, esto es, si suficientes de sus entradas laterales están activas debido a la entrada de alimentación. En este caso, la célula encadena los siguientes cambios: a) refuerza el segmento activo actual (líneas 47-48), y b) refuerza un segmento que podría haber predicho esta activación, esto es, un segmento que tiene un encaje débil con la actividad en el paso del tiempo anterior (líneas 50-53).

```
42.     for c, i in cells
43.         for s in segments(c, i)
44.             if segmentActive(s, t, activeState) then
45.                 predictiveState(c, i, t) = 1
46.
47.                 activeUpdate = getSegmentActiveSynapses(c, i, s, t, false)
48.                 segmentUpdateList.add(activeUpdate)
49.
50.                 predSegment = getBestMatchingSegment(c, i, t-1)
51.                 predUpdate = getSegmentActiveSynapses(
52.                                                 c, i, predSegment, t-1, true)
53.                 segmentUpdateList.add(predUpdate)
```

Fase 3:

La tercera y última fase es la que realiza el aprendizaje. En esta fase las actualizaciones de los segmentos que han sido puestos en cola son efectivamente implementadas una vez que tenemos la alimentación y la célula es elegida como la célula que realizará el aprendizaje (líneas 56-57). Al contrario, si la célula deja de predecir por cualquier razón, reforzaremos negativamente los segmentos (líneas 58-60).

```
54.     for c, i in cells
55.         if learnState(s, i, t) == 1 then
56.             adaptSegments(segmentUpdateList(c, i), true)
57.             segmentUpdateList(c, i).delete()
58.         else if predictiveState(c, i, t) == 0 and predictiveState(c, i, t-1) == 1 then
59.             adaptSegments(segmentUpdateList(c, i), false)
60.             segmentUpdateList(c, i).delete()
```

III 2.3 Detalles y terminología de la implementación

A continuación se describen algunos de los detalles de la implementación y terminología del agrupador temporal. Cada célula se indexa utilizando dos números: un indexador de columna, c, y un indexador de célula, i. Las células mantienen una lista de segmentos predictivos, donde cada segmento contiene una lista de las conexiones y el valor de la permanencia de cada una. Los cambios en las conexiones de una célula son marcados como

temporales hasta que la célula se activa por la entrada de alimentación. Estos cambios temporales son mantenidos en segmentUpdateList. Además cada segmento mantiene un señalizador booleano, sequenceSegment, indicando si el segmento predice o no la entrada de alimentación en el siguiente paso temporal.

El código además utiliza una pequeña máquina de estados para hacer el seguimiento de los estados de las células en diferentes momentos en el tiempo. Se mantienen tres estados diferentes por cada célula. Las matrices activeState y predictiveState hacen el seguimiento del estado activo y predictivo de cada célula en cada momento. La matriz learnState determina qué salidas de la célula son usadas durante el aprendizaje. Cuando una entrada es inesperada, todas las células en una columna en particular se activan en el mismo instante en el tiempo. Sólo una de esas células, la que mejor encaje con la entrada, tiene su estado learnState activado. Sólo se añaden conexiones de células que tengan el learnState en modo “on”.

III 2.4 Apéndice del agrupador temporal

Cell(c, i)	Lista de células, indexadas por i y c.
cellsPerColumn	Número de células en cada columna.
activeColumns(t)	Lista de los índices de las columnas ganadoras debido a la entrada de alimentación (salida del agrupador temporal).
activeState(c, i, t)	Un vector booleano con un número por célula. Representa el estado activo de la columna c célula i en el momento t pasandole la entrada de alimentación y el contexto temporal pasado. Si es 1, la célula tiene entrada de alimentación así como el contexto temporal adecuado.
predictiveState(c, i, t)	Vector booleano con un número por célula. Representa la predicción de la columna c célula i en el momento t, pasándole la alimentación de otras columnas y el contexto temporal pasado. Si es 1, la célula predice la entrada de alimentación en el contexto temporal actual.
learnState(c, i, t)	Un booleano indicando si la célula i en la columna c ha sido elegida como la célula para aprender.
activationThreshold	Umbral de activación para un segmento. Si el número de conexiones establecidas en un segmento es superior que activationThreshold, se dice que el segmento está activo.
learningRadius	El área alrededor de una célula del agrupador temporal desde la cual puede recibir conexiones laterales.
initialPerm	Valor de permanencia inicial para una conexión.
conectedPerm	Si el valor de permanencia para una conexión es mayor que este valor, se dice que está conectada.
minThreshold	Umbral de actividad mínima del segmento para el aprendizaje.

newSynapseCount	Número máximo de conexiones añadidas al segmento durante el aprendizaje.
permanenceInc	La cantidad por la que se incrementan los valores de la permanencia de las conexiones cuando ocurre el aprendizaje.
permanenceDec	La cantidad por la que se disminuyen los valores de la permanencia de las conexiones cuando ocurre el aprendizaje.
segmentUpdate	Estructura de datos que guarda tres tipos de información requeridos para actualizar un segmento: a) índice de segmentos, b) una lista de las conexiones activas existentes, y c) un señalizador indicando si este segmento debería estar marcado como segmento de secuencia.
segmentUpdateList	Una lista de estructuras de segmentUpdate. segmentUpdateList(c, i) es la lista de cambios para la célula i en la columna c.

Las siguientes rutinas de apoyo se usan en el código anterior.

segmentActive(s, t, state)

Esta rutina devuelve **true** si el número de conexiones establecidas en el segmento s que son activas debidas al estado en el momento t es mayor que activationThreshold. El estado del parámetro puede ser activeState o learnState.

getActiveSegment(c, i, t, state)

Para la columna c célula i, devuelve un índice de segmentos tal que segmenActive(s, t, state) es **true**. Si hay múltiples segmentos activos, los segmentos de secuencia tienen preferencia. En otro caso, los segmentos con mayor actividad tienen preferencia.

getBestMatchingSegment(c, i, t)

Para la columna c célula i en el momento t, encuentra el segmento con el número más grande de conexiones activas. El valor de permanencia de las conexiones puede estar por debajo de connectedPerm. El número de conexiones activas puestas por debajo de minThreshold. La rutina devuelve el índice de segmentos. Si no los encuentra, devuelve un índice de -1.

getBestMarchingCell(c)

Para la columna, devuelve la célula con el mejor segmento coincidente. Si no hay células con segmentos coincidentes, devuelve la célula con el menor número de segmentos.

getSegmentActiveSynapses(c, i, t, s, newSynapses = **false**)

Devuelve una estructura de datos del tipo segmentUpdate que contiene una lista con los cambios propuestos para el segmento s. activeSynapses es una lista de las conexiones activas donde las células originales en su salida activeState output = 1 en el momento t. Si newSynapses es **true**, entonces las conexiones resultantes de newSynapsesCount – count(activeSynapses) serán

añadidas a activeSynapses. Estas conexiones son elegidas aleatoriamente desde el grupo de células que tienen la salida learnState = 1 en el momento t.

adaptSegments(segmentList, positiveReinforcement)

Esta función itera sobre una lista de segmentUpdate-s, y refuerza cada segmento. Para cada elemento segmentUpdate, se realizan los siguientes cambios. Si positiveReinforcement es **true** entonces las conexiones en la lista activa ven sus contadores de permanencia incrementados por permanenceInc. El resto de las conexiones de sus contadores de permanencia disminuidos por permanenceDec. Si positiveReinforcement es **false**, entonces las conexiones en la lista activa ven sus contadores de permanencia disminuidos por permanenceDec. Después de esta etapa, cualquier conexión en segmentUpdate que todavía exista verá su contador de permanencia añadido por initialPerm.

III 3. Instalación del software NuPIC

La mayor fuente de problemas en la realización de este proyecto se corresponde con la instalación y puesta a punto del software NuPIC. Es por ello que se ha estimado oportuno exponerlo en este capítulo.

La instalación, durante el desarrollo de este proyecto, ha sido bastante tediosa. Durante el proceso aparecían múltiples errores que había que depurar y aún con todo han sido múltiples las veces en las que ha habido que reinstalar el software para su uso. Esto ha conllevado a un retraso importante en el desarrollo del proyecto.

Los problemas en la instalación vienen de dos fuentes principales. En primer lugar NuPIC es un software de código abierto (open source) lanzado por la empresa Numenta y apoyado por una comunidad de gente interesada en la inteligencia artificial. El que sea un software de código abierto quiere decir que, aparte de estar disponible de forma gratuita, cualquiera puede ser desarrollador del software y cambiar módulos para mejorarlo. Esto es un aspecto positivo en cuanto a que gente que se interesa por el proyecto aporta ideas y realiza cambios con el fin de mejorar la herramienta. Sin embargo el problema es que estos cambios no están exentos de errores y es la propia comunidad la encargada de comprobar y depurar dichos errores.

En segundo lugar, el otro problema viene de la corta edad del software. NuPIC se lanzó en junio de 2013 y tuvieron que pasar algunos meses para que la comunidad aprendiera su estructura y fuera capaz de aportar desarrollo al proyecto.

No se va a dar una guía detallada de los pasos que se deben realizar para instalación del software por dos razones. La primera y la más simple es que no lo creemos necesario ya que en la Wiki del propio NuPIC está toda la información necesaria. La segunda razón reside en que como ya se ha mencionado el programa está en constante cambio gracias a las aportaciones de la comunidad por lo que la instalación es una sección que ha ido cambiando a lo largo del desarrollo del proyecto y que sigue cambiando a día de hoy.

Lo primero que hay que hacer para instalar y trabajar con NuPIC, es familiarizarse con GitHub [8]. GitHub es una plataforma de desarrollo colaborativo que se enfoca hacia la cooperación entre desarrolladores para la difusión de software y el soporte al usuario, y que se usa para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework

Ruby on Rails por *GitHub*, *Inc*. GitHub facilita la tarea en la gestión de versiones de software y tiene sus propios comandos con los que podemos descargar el programa, actualizarlo, realizar cambios, y guardar distintas versiones del programa con el fin de que podamos recuperar el código si los últimos cambios que se han realizado no han funcionado.

NuPIC se ha preparado para que sea capaz de ejecutarse en diferentes sistemas operativos. Este proyecto se ha realizado bajo el sistema operativo Ubuntu [9] basado en Linux. Una vez instalado GitHub en nuestro sistema operativo, primero hay que instalar una lista de herramientas que necesita a modo de requerimiento el módulo de instalación del NuPIC. El problema es que estas herramientas hay que instalarlas una a una manualmente ya que si no están correctamente instaladas no se podrá seguir. Hay un problema añadido y es que, si no se han instalado correctamente alguna de estas herramientas, hay que eliminarlas completamente del sistema operativo, lo que no es trivial en Ubuntu, y proceder a su reinstalación.

Después de instalar las herramientas, se hace un clone con GitHub para copiar el repositorio de NuPIC, esto es descargarse el programa en bruto. A continuación hay que definir una serie de variables en el propio sistema operativo editando el archivo .bashrc, y hay que instalar una lista de módulos a modo de requerimientos, externos al NuPIC pero necesarios para su instalación. Con estos módulos sucede lo mismo que ocurría con las herramientas anteriormente requeridas.

Una vez que se tiene todo lo necesario ya se puede instalar el NuPIC confiando en que no se produzcan errores durante el proceso. Si aparece algún error es que alguno de los requerimientos no se instaló correctamente. Localizarlo, eliminarlo, y reinstalarlo, a veces es tan complicado que a menudo la mejor solución es reinstalar el sistema operativo, haciendo un back up si es posible, y comenzar de nuevo.

Este proceso se ha realizado infinidad de veces durante el desarrollo del proyecto debido a que una vez instalado y en funcionamiento, a la semana, por poner un ejemplo, algún desarrollador de la comunidad actualizaba el software introduciendo alguna mejora interesante viéndonos obligados a actualizar a la última versión. Al actualizar aparecían los errores que no habían sido depurados y que poco a poco se irían arreglando con el feedback de los usuarios. Con lo que lo mejor que se podía hacer cada vez que aparecía una actualización era reinstalar el programa desde cero con el inconveniente de que cada tres semanas, más o menos, se introducían cambios en el proceso de instalación con sus respectivos problemas.

Aunque a día de hoy sigue habiendo mensajes de usuarios que tienen algún problema durante el proceso de instalación hay que decir que se ha trabajado bastante este módulo haciéndolo más automático y libre de errores. Es de esperar que en unos pocos meses se cree un instalador sencillo que realice todo el proceso de instalación sin la intervención del usuario.

IV. EVALUACIÓN DEL ALGORITMO

Para evaluar el algoritmo se ha utilizado un programa que realiza predicciones en base a un modelo creado con HTM. Estos experimentos se han enfocado en la predicción del consumo eléctrico para lo cual se han utilizado series históricas de consumo generado en la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad de Cantabria durante los años 2012/13. Los experimentos se han realizado con dos tipos de datos de consumo. Con datos espaciados entre sí 15 minutos y cada una hora, y se han evaluado los resultados tanto en la predicción como en la detección de anomalías.

El programa consta de dos fases. Una fase de "swarming [10]", en la que se crea un modelo con los datos de las series históricas o datos estadísticos, y una fase de ejecución en la que se van introduciendo en el modelo creado en la primera fase los datos actuales obteniendo la predicción a la salida. En este capítulo se nombran diferentes archivos con la extensión .py. Esta es la extensión del lenguaje Python el cual utiliza NuPIC por lo que todos los archivos, estructuras y algoritmos están programados en dicho lenguaje.

IV 1. Swarming

Swarming es un proceso que determina de manera automática el modelo que produce con mayor precisión la salida deseada dada una serie de datos de entrada. Durante el proceso decide qué componentes debe utilizar en el modelo (agrupador espacial, agrupador temporal, clasificadores, etc), así como los mejores valores de los parámetros a utilizar para cada componente.

Para realizar el proceso de swarming debemos proporcionarle la siguiente información:

- Un archivo en formato .csv con la serie de datos que se quiere optimizar.
- Los campos de la serie de datos que se van a considerar entradas en el modelo y su tipo de dato.
- Cuánto se quiere combinar los datos de entrada antes de alimentar el modelo.
- La tarea que se desea realizar (por ejemplo predecir cuatro pasos en el futuro).

IV 1.1 Ejecución del swarming

Lo primero que se necesita para realizar un proceso de swarming es un archivo en formato .csv con los datos que se quieren procesar, en nuestro caso los datos de consumo eléctrico de la escuela. La particularidad de los archivos con formato .csv, a diferencia de otras hojas de cálculo, es que sus campos están separados por comas y es este formato con el cual NuPIC está preparado para trabajar.

Para la predicción del consumo eléctrico nuestro archivo .csv contiene dos campos separados en columnas. El primer campo refleja la fecha y la hora en la que se tomó el dato de consumo. Las librerías del software NuPIC son bastante restrictivas en cuanto al formato de las

fechas por lo que hay que poner especial cuidado para evitar incompatibilidades. En este proyecto se ha trabajado con el formato mes/día/año hora:minuto. El segundo campo indica el consumo eléctrico medido en un instante de tiempo. Un ejemplo de archivo .csv utilizado sería el siguiente:

```
timestamp,kw_energy_consumption
datetime,float
T,
11/01/2012 00:00 , 025.489
11/01/2012 00:15 , 025.516
11/01/2012 00:30 , 024.497
11/01/2012 00:45 , 022.964
11/01/2012 01:00 , 022.369
11/01/2012 01:15 , 022.515
11/01/2012 01:30 , 022.564
```

Cabe destacar que cuantos más datos se utilizan en el proceso de swarming mejor y más ajustado será el modelo creado. En los experimentos realizados en este proyecto se ha trabajado con archivos de 6000 y 24000 datos dependiendo del tiempo entre medidas, con lo que se ha reflejado el consumo generado durante 8 meses. Lo ideal para crear el mejor modelo sería proporcionar datos de varios años de consumo. El problema es que el swarming es un proceso muy pesado que requiere el uso de máquinas potentes lo que limita la densidad de datos a la hora de trabajar. Por poner un ejemplo, realizar un proceso de swarming de ocho meses con datos espaciados cada 15 minutos lleva unas 14 horas de proceso.

Una vez que se tiene el archivo .csv hay que configurar el tipo de swarming que se desea realizar. Para ello se le proporciona un archivo de descripción, swarm_description.py, con las siguientes secciones:

- En *includedFiles* se indican que campos del archivo .csv se van a considerar entradas en el modelo y el tipo de dato de cada campo. Por defecto, los campos que no son reflejados aquí no se incluyen en el modelo como entradas.

- *streamDef* describe dónde está el archivo de datos de entrada, es decir la ruta del archivo .csv.

- *inferenceType* describe el tipo de modelo que se desea crear. Para crear un modelo capaz de predecir varios pasos en el futuro con antelación se ha utilizado el tipo de inferencia “multiStep”. Existen otros tipos de inferencias que se utilizan para crear diferentes modelos pero éstos se salen del objetivo del proyecto.

- *inferenceArgs* para un modelo de tipo multiStep, incluye el número de pasos en el futuro que se desea predecir con antelación, así como el nombre del campo del que se quiere realizar la predicción. El número de pasos que se desea predecir con antelación tiene un significado diferente dependiendo del tipo de dato utilizado, es decir, si tenemos datos de consumo cada hora y hacemos una predicción de un paso, estaremos haciendo una predicción del consumo que habrá dentro de una hora. Si por ejemplo realizamos una predicción de cuatro pasos estaremos prediciendo cuatro horas en el futuro. Si por el contrario en vez de tener datos de consumo cada hora tenemos datos cada 15 minutos, la predicción en el futuro será en el primer caso de 15 minutos y en el segundo caso de una hora.

- El valor de *iterationCount* indica el número máximo de datos combinados para introducir en el modelo. Si se tienen 1000 datos de entrada espaciados cada 15 minutos y se especifica un intervalo de combinación de una hora, se tendrán un máximo de 250 datos disponibles para introducir en el modelo. Con el fin de conseguir el modelo con mayor ajuste en este proyecto se han utilizado todos los datos de entrada. Para indicar esto sólo hay que fijar el valor de *iterationCount* a -1.

- Por último *swarmSize* indica cómo de exhaustivo debe ser el proceso. Existen tres posibles ajustes que son "small", "medium", y "large". El ajuste "small" evalúa un único modelo posible y utiliza sólo los 100 primeros datos de entrada. Es un proceso muy rápido que sólo sirve para comprobar que no hay ningún fallo de sintaxis o con los datos de entrada. En la mayoría de los casos se suele utilizar el ajuste "medium" que cubre la mayor parte de las necesidades de los usuarios. En este proyecto se ha utilizado el ajuste "large" en todos los experimentos ya que es el que produce el mejor modelo.

Después de la ejecución nos aparecerá por consola un mensaje con la ruta de los archivos generados, el número de modelos creados y el tiempo que se ha empleado en el proceso. Como se ha mencionado anteriormente el swarming es un proceso muy pesado. Por esta razón una de las opciones que tenemos a la hora de ejecutar el swarming es indicarle cuantos núcleos tiene nuestro procesador con el fin de que la ejecución se realice en paralelo reduciendo de manera significativa el tiempo de ejecución.

El proceso de swarming genera los siguientes archivos entre los que se encuentra el modelo creado:

- **description.py**: Un archivo de descripción de tipo OPF (Online Prediction Framework) que describe los componentes del modelo y sus parámetros. Éste archivo es el modelo creado donde están definidos el tipo de jerarquía, el tamaño de las regiones, cuántas columnas y cuantas células por columna contiene el modelo, etc.

- **permutations.py**: Este archivo define el tipo de búsqueda que se ha utilizado durante el swarming. Por ejemplo, incluye el mínimo y el máximo valor permitido para cada parámetro.

- **model_0/description.py**: Un archivo de descripción de tipo OPF que sobrescribe algunos parámetros específicos del archivo description.py descrito más arriba, con los valores que encuentra durante el proceso de swarming.

- **model_0/params.csv**: Un archivo .csv que contiene los valores de los parámetros escogidos para crear el mejor modelo.

- **model_0/model_params.py**: Contiene los parámetros del modelo creado. Es este archivo el que se utilizará posteriormente en la fase de ejecución. Se puede editar manualmente por ejemplo para cambiar el tipo de inferencia.

- **search_def_Report.csv**: Se trata de una hoja de cálculo que contiene información sobre cada uno de los modelos que fueron evaluados durante el swarming. Incluye los parámetros utilizados de cada modelo así como un cálculo del error del modelo generado.

Ya se ha mencionado que el swarming es un proceso muy pesado sin embargo hay que aclarar que no es necesario crear un modelo cada vez que se quiere evaluar un conjunto de datos diferente. Una vez creado un modelo este puede realizar predicciones con cualquier conjunto de datos siempre y cuando éstos tengan relación y mantengan el mismo formato. Por poner un ejemplo podríamos evaluar los datos del año 2013, 2014, 2015 con un modelo creado con los datos del 2011, 2012 o ambos. Esto nos lleva a la conclusión de que siendo posible es preferible crear un único modelo con la mayor cantidad de datos posibles, es decir cubriendo el máximo de años, con el fin de obtener el modelo más completo.

IV 2. Ejecución del programa

Una vez creado el modelo se puede pasar a la fase de ejecución. En esta fase se inicia un nuevo modelo de NuPIC partiendo de los parámetros definidos en el archivo model_params.py que nos devolvió el proceso de swarming, y se van introduciendo línea a línea los datos que se van a analizar. Es totalmente imprescindible haber realizado el proceso de swarming para que NuPIC pueda partir de un modelo base. El resultado de la ejecución, es decir la predicción que se va obteniendo, se puede imprimir en una hoja de cálculo de formato .csv o se puede ir visualizando en una gráfica al tiempo que se van generando los datos.

Éstas opciones de salida se pueden programar y editar en el archivo nupic_output.py al que se llama desde el programa de ejecución. Se puede editar cualquier aspecto de las gráficas, tamaño, fondo, color de las líneas, etc, y especificar el formato con el que se escriben los resultados en la hoja de cálculo. Por defecto al ejecutar el programa NuPIC imprimirá el resultado en una hoja de cálculo. Si queremos que nos muestre la gráfica en tiempo real al ejecutar el programa debemos añadirle por línea de comandos “--plot”.

Los datos de consumo que se quieran evaluar deben tener el mismo formato descrito en la sección de swarming pero no tienen por qué ser los mismos. Si en el modelo creado se utilizaron datos espaciados cada 15 minutos, este modelo se puede utilizar para analizar datos espaciados una hora sin ningún problema. También puede hacerse al revés sin embargo es muy posible que el ajuste sea peor ya que los datos que se pretenden evaluar en la fase de ejecución tienen más precisión que los que se usaron para crear el modelo por lo que no es recomendable.

IV 3. Detección de anomalías

Una de las herramientas más interesantes de NuPIC es la detección de anomalías. En el campo de la predicción se entiende como anomalía cuando el resultado no era ninguno de los esperados. Esto no significa que exista anomalía cada vez que se falla en la predicción. Cuando NuPIC realiza una predicción no estima solamente un posible valor sino que abarca posibles combinaciones de las cuales finalmente se decide por la que considera más probable.

Por ejemplo cuando nos dirigimos hacia una puerta predecimos que se puede abrir. Si al intentar abrirla descubrimos que está cerrada fallamos en la predicción sin embargo esto no nos sorprende, no sería una anomalía, ya que contábamos con la posibilidad de que estuviera cerrada. Si por el contrario cuando nos acercamos descubrimos que no hay puerta sólo un hueco, o incluso un muro que no podemos atravesar entonces nos sorprendemos ya que es algo que no esperábamos. En este caso habría una anomalía.

La detección de anomalías le proporciona al CLA (Cortical Learning Algorithm) una herramienta que representa como es de predecible un dato y que se implementa en el núcleo tanto del agrupador espacial como del agrupador temporal sin necesidad de hacer ningún cambio en sus algoritmos. En este proyecto se trabaja con un modelo de anomalía temporal para la predicción del consumo eléctrico en donde cada dato predicho tendrá una puntuación de anomalía (anomaly score), entre el uno y el cero. Un cero representa un valor totalmente esperado mientras que un uno representa un valor totalmente inesperado, es decir una anomalía.

En el caso anterior de la puerta, si ésta estuviera abierta o cerrada tendríamos una puntuación de anomalía cercana al cero mientras que en el escenario de que no existiera puerta tendríamos una puntuación cercana al uno.

IV 3.1 Modelo de anomalía temporal

Para la detección de anomalías NuPIC puede trabajar tanto con modelos temporales como no temporales. En este proyecto se ha trabajado con modelos temporales ya que en todo momento se está trabajando con datos de consumo que varían en el tiempo y es por esto que sólo nos centraremos en este tipo de modelo. En este documento no abarcaremos el modelo de anomalía no temporal ya que es algo que no está en los objetivos de este proyecto.

Debemos definir como temporal el modelo con el que vamos a trabajar para poder ejecutar la detección de anomalías. Esto podemos hacerlo de dos formas. La primera es en la fase de swarming en donde podemos fijar el tipo de inferencia como “temporalAnomaly” dentro del archivo swarm_description.py. La otra forma es que si ya tenemos un modelo creado con una inferencia de tipo “multiStep” podemos editar el archivo model_params.py para cambiar el tipo de inferencia por “temporalAnomaly”.

Una vez que se ha fijado el tipo de inferencia, al ejecutar el programa el detector de anomalías utiliza el agrupador temporal para detectar nuevas secuencias de puntos. Esto detectará tanto patrones de entrada nuevos así como antiguos patrones espaciales que se producen en un nuevo contexto.

El modelo de anomalía temporal calcula el valor de anomalía basándose en el acierto de la anterior predicción. Esto se calcula como el porcentaje de columnas activas en el agrupador espacial que fueron incorrectamente predichas por el agrupado temporal.

El algoritmo del detector de anomalías sería el siguiente:

$$anomalyScore = \frac{|A_t - P_{t-1} \cap A_t|}{|A_t|}$$

En donde P_{t-1} son las columnas predichas en el tiempo t, y A_t son las columnas activas en el tiempo t.

Por lo tanto, un valor de anomalía igual a 1 significa que no hay células predichas que se vayan a activar y representa un hecho completamente anómalo. Un valor igual a 0 significa que todas las células predichas se van a activar y representa un hecho totalmente previsible.

IV 3.2 Anomaly likelihood

Como complemento a la detección de anomalías, durante la realización de este proyecto un ingeniero de Numenta añadió un módulo denominado anomalyLikelihood que nos indica con un valor entre el cero y el uno como de acertada ha sido la detección de anomalías.

Este dato nos es de gran utilidad ya que nos sirve para acotar las anomalías y detectar los falsos positivos que nos llevarían a confusión durante la ejecución del programa y la evaluación de los resultados.

IV 4. Resultados

Para evaluar el algoritmo se han realizado tres experimentos. Para cada uno de estos experimentos se ha creado un modelo diferente con el fin de obtener resultados de predicción distintos. Los datos utilizados para la creación del modelo y la ejecución del programa son los datos de consumo eléctrico proporcionados por la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad de Cantabria.

La mejor manera de analizar los resultados es mediante la observación de una gráfica en la cual aparecen los datos de consumo real y la predicción que NuPIC realiza al tiempo que se va generando. Como no es posible mostrar la ejecución del programa, en este documento se incluyen algunas capturas de las gráficas con los momentos más reseñables con el fin de evaluar los resultados para cada uno de los experimentos. Además de las gráficas el programa crea una hoja de cálculo con los datos de consumo reales y con la predicción que el algoritmo ha estimado. Para tener un dato numérico de la eficacia de este algoritmo, utilizando dicha hoja de cálculo se ha calculado tanto el error cuadrático medio como el porcentaje de error medio que se comete en la predicción antes y después del entrenamiento.

IV 4.1 Caso A: predicción de una hora

En el primer experimento realizado se ha creado un modelo para la predicción de una hora. Los datos de consumo utilizados abarcan las fechas del 1 de noviembre de 2012 hasta el 30 de junio de 2013, prácticamente los ocho meses de mayor actividad en la escuela, con un total de 5807 muestras espaciadas cada hora.

Para la creación del modelo, en el proceso de swarming se ha utilizado el tipo de inferencia “multistep”, se han utilizado todos los datos fijando el valor de “iterationCount” a -1 y se ha definido un tamaño de swarming “large”. Como se quiere realizar una predicción de una hora en el futuro y los datos están espaciados cada hora se ha fijado el paso de predicción, “predictionStep”, a 1. El tiempo de ejecución durante la creación del modelo ha sido de unos 22 minutos.

Al revisar el resultado de la predicción se puede observar que en las primeras líneas la predicción es exactamente igual al consumo real. Esto no es debido a que el programa acierte nada más empezar sino que al no tener datos anteriores con los que realizar la predicción NuPIC predice que en este instante habrá el mismo valor que justo en el instante anterior. Aunque parezca bastante aleatoria, esta forma de trabajar es la que menos errores produce. Un ejemplo claro de este comportamiento es que se acierta más veces diciendo que el tiempo que hace hoy será el mismo que mañana. De todas formas esto no tiene ningún efecto sobre el

resultado de la predicción ya que sólo ocurre en unas pocas líneas del principio, despreciables respecto al total de los datos utilizados.

A continuación se muestran algunas de las gráficas de la ejecución del programa en secuencia:

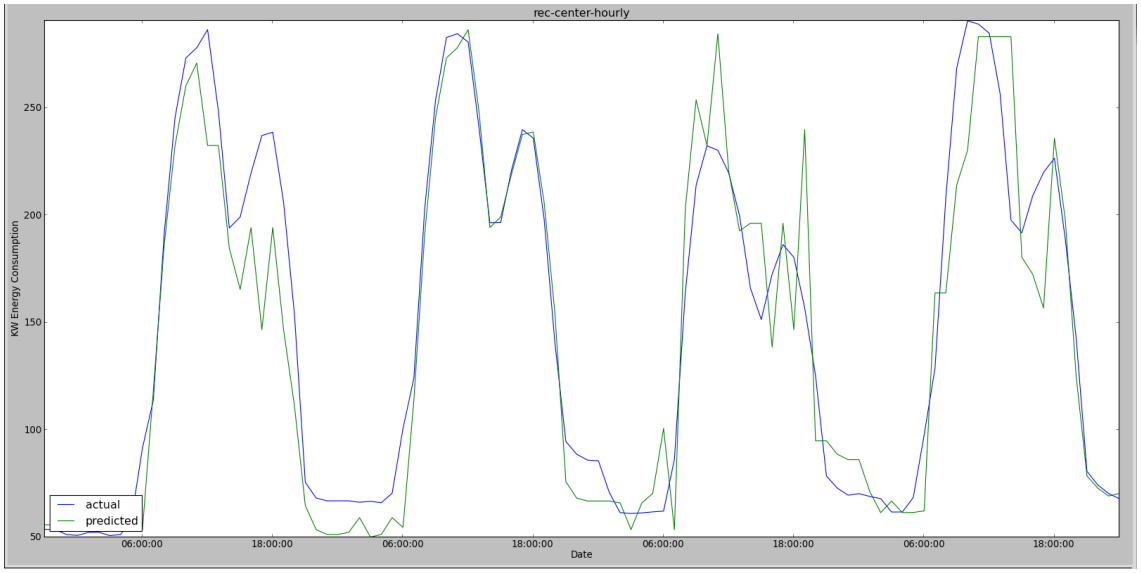

Imagen IV.1 – Caso A, 300 iteraciones

La primera gráfica corresponde al principio de ejecución del programa con 300 interacciones. La línea de color azul corresponde al consumo eléctrico real generado en la escuela mientras que la línea de color verde corresponde a la predicción que NuPIC realiza de una hora en el futuro. Se pueden observar cuatro montículos. Cada uno de estos montículos corresponden a un día de la semana, en este caso se corresponden con los días lunes, martes, miércoles y jueves. A pesar de haber empezado el entrenamiento (sólo lleva 300 interacciones), la predicción mantiene la forma debido a que parte del modelo creado durante la fase de swarming. Se observa que el martes (segundo montículo), la predicción es bastante ajustada, sin embargo el miércoles, si nos fijamos en el dato de consumo real, se generó un consumo menor de lo que le corresponde a una jornada normal de la escuela, por lo que NuPIC que esperaba un consumo de unos 280 kW rectifica su predicción dando lugar a ese diente de sierra.

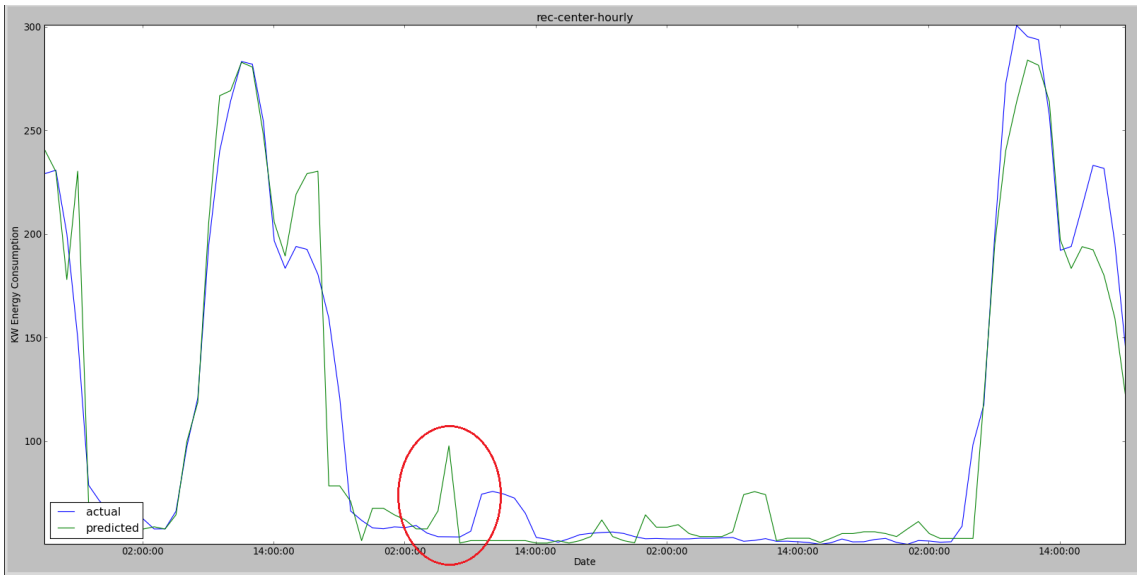

Imagen IV.2 – Caso A, 700 iteraciones

En la siguiente gráfica vemos que existe un vacío entre dos montículos. Este vacío se corresponde con los días sábado y domingo en los que como es lógico el consumo eléctrico es mucho menor debido a la falta de clases. Con 700 interacciones al programa aún le cuesta reaccionar en los fines de semana. En la zona enmarcada en rojo se puede ver como NuPIC intenta dar una predicción de consumo de un día laboral pero en el momento que comprueba que sus datos de entrada son de consumo bajo rectifica.

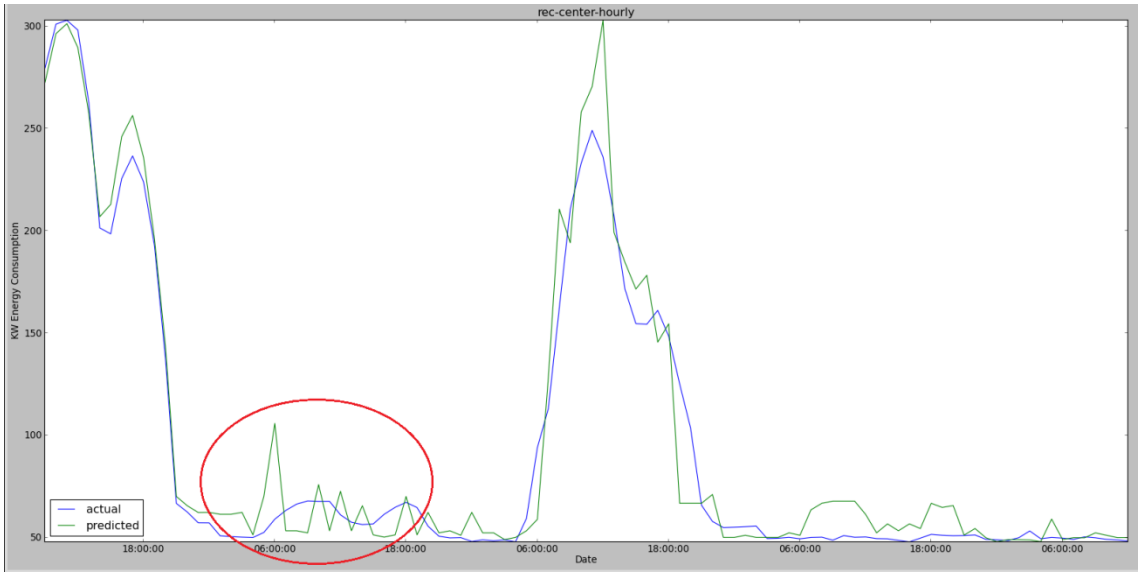

Imagen IV.3 – Caso A, 900 iteraciones

En esta captura los dos montículos grandes corresponden a los días miércoles y viernes. El espacio que hay entre los dos corresponde a un jueves festivo. Una vez más el algoritmo espera que ese jueves haya un consumo normal de un día laboral, es por eso que en la zona enmarcada en rojo se ve un pico de color verde de predicción que una vez más rectifica al ver la entrada de consumo real. Son estas rectificaciones las que hacen del software NuPIC una herramienta de gran potencial. En los métodos estadísticos que se han ido utilizando hasta ahora, algunos de los cuales se ha mencionado en la introducción de este documento, precisamente se ha señalado que este en concreto es uno de los escenarios que incrementan

el error cometido ya que estos métodos no son capaces de dar respuesta ante un evento inesperado como puede ser un día festivo.

El viernes se observa un consumo menor seguramente debido a que una gran parte del alumnado y también del profesorado decidió hacer puente ese día.

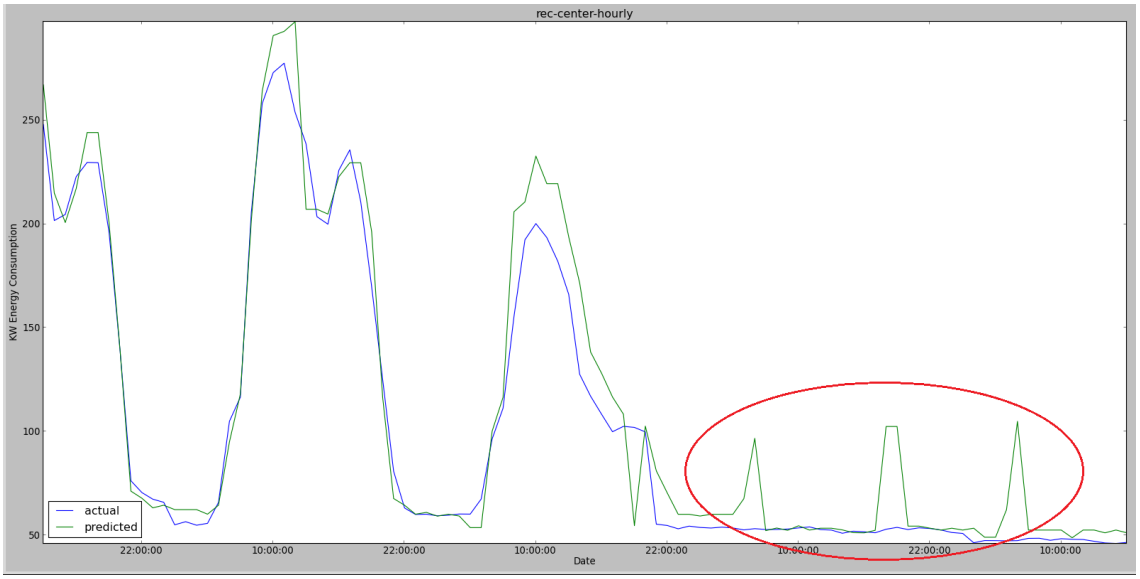

Imagen IV.4 – Caso A, 1200 iteraciones

En esta captura se aprecia una serie de errores de predicción debido a que corresponde con la entrada de las vacaciones de Navidad. En el montículo central se aprecia un consumo menor de normal seguramente por ser el último día antes de las vacaciones, un día en el que muchas de las clases son suspendidas. Los tres siguientes picos de color verde son los siguientes días en los que NuPIC espera que haya un consumo normal. El primer pico de color verde corresponde a un viernes que para el software es un día laboral. Al detectar que esto no es así rectifica, sin embargo de casos como el anterior aprende que después de un día festivo no esperado viene un día laboral de consumo normal por lo que al día siguiente vuelve a predecir que habrá un consumo normal. Después de varias interacciones el programa se adapta al período de vacaciones.

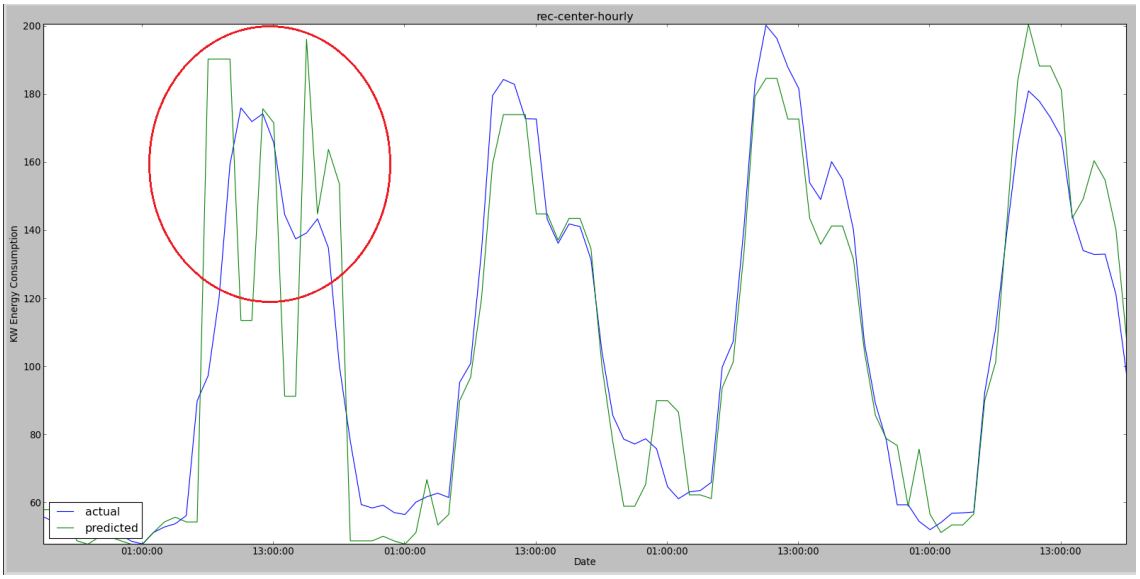

Imagen IV.5 – Caso A, 3700 iteraciones

Con 3700 interacciones el programa ya ajusta bastante bien la predicción sin embargo como se puede observar en esta gráfica en algún momento puntual aparecen situaciones en las que el modelo que se está creando no consigue dar una buena predicción. Este día corresponde a un lunes en el que por algún motivo hubo un consumo menor de lo habitual, un evento al cual al algoritmo le cuesta dar respuesta.

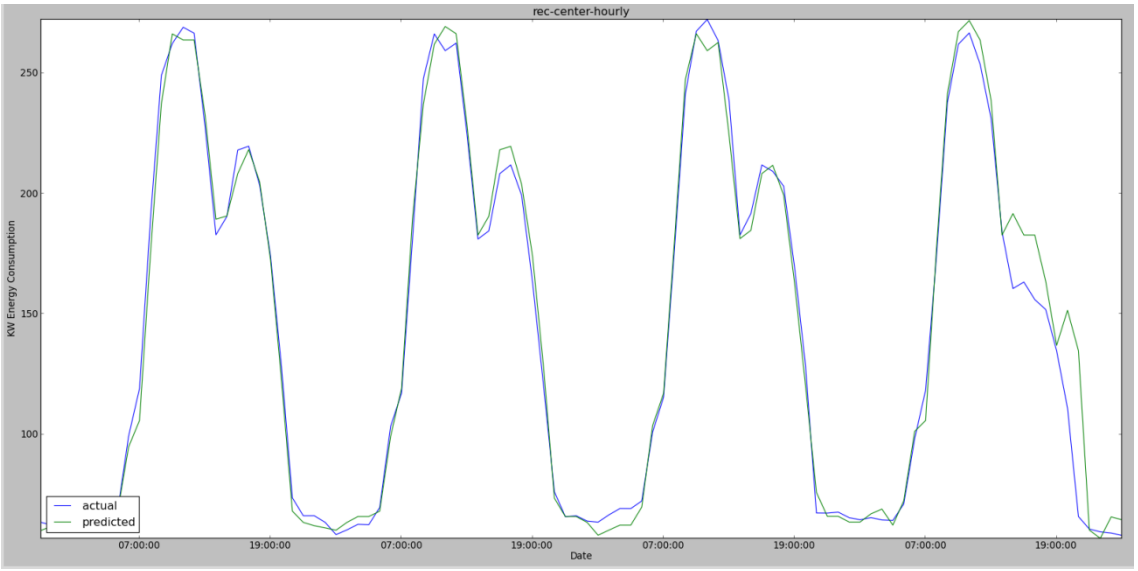

Imagen IV.6 – Caso A, 5500 iteraciones

Finalmente después de 5500 interacciones, ya casi al final del proceso, se puede observar que la predicción es bastante ajustada. Aunque no se puede apreciar observando estas capturas, durante el proceso de ejecución si se aprecia cómo se desarrolla la fase de aprendizaje del programa. Puede observarse cómo la predicción va mejorando gradualmente a medida que aumentan las interacciones.

Utilizando la hoja de cálculo que nos devuelve NuPIC después de la ejecución con los datos de predicción se ha calculado el error cuadrático medio y el porcentaje de error medio. Éste cálculo se ha realizado en dos escenarios; antes del entrenamiento, es decir utilizando los datos de predicción que se generan desde el principio de la ejecución del programa, y después del entrenamiento, con los datos de predicción generados a partir de 3000 interacciones. Cuando hablamos del cálculo después del entrenamiento nos referimos a que ha habido un entrenamiento previo, en este caso 3000 interacciones, que hacen que la predicción sea mejor. En cualquier caso no significa que el entrenamiento haya terminado ya que NuPIC seguirá aprendiendo de sus entradas nuevas hasta el final de la ejecución del programa. Recordamos que el entrenamiento puede desactivarse cuando se desee, no obstante esto sólo es interesante en el caso en el que no se quiera que los datos que se estén evaluando influyan en el modelo que se está creando.

El error cuadrático medio se calcula como:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(Y_i - X_i)^2$$

Siendo Y un vector de n predicciones y X el vector de valores reales.

El error porcentual medio se ha calculado como:

$$ERROR = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|Valor\ estimado_i - Valor\ real_i|}{Valor\ real_i}\right) * 100$$

Siendo el valor estimado un vector de n predicciones y el valor real el vector de valores de consumo reales.

ECM	ECM AT	ERROR %	ERROR % AT
747.9	686.9	14.4	13.8

Como era de esperar tanto el error cuadrático medio como el porcentaje de error medio después del entrenamiento, en este caso después de 3000 interacciones, es mejor. A medida que aumenten las interacciones los errores disminuirán hasta que se desactive el entrenamiento o hasta que el modelo se estabilice.

IV 4.2 Caso B: predicción de diez horas

Para este experimento se ha creado un modelo de predicción de diez horas. Los datos de consumo utilizados son los mismos que los utilizados en el experimento anterior, es decir, abarcan las fechas 1 de noviembre de 2012 hasta el 30 de junio de 2013, prácticamente los ocho meses de mayor actividad en la escuela, con un total de 5807 muestras espaciadas cada hora.

Para la creación del modelo sólo vamos a cambiar el número de pasos de predicción por lo que el resto de parámetros serán los mismos que en el caso anterior. El tipo de inferencia será “multistep”, se utilizarán todos los datos fijando el valor de “iterationCount” a -1 y configuramos un tamaño de swarming “large”. Como se quiere realizar una predicción de diez horas en el futuro y nuestros datos están espaciados cada hora se tiene que fijar el paso de predicción, “predictionStep”, a 10. El tiempo de ejecución durante la creación del modelo ha sido de unos 24 minutos.

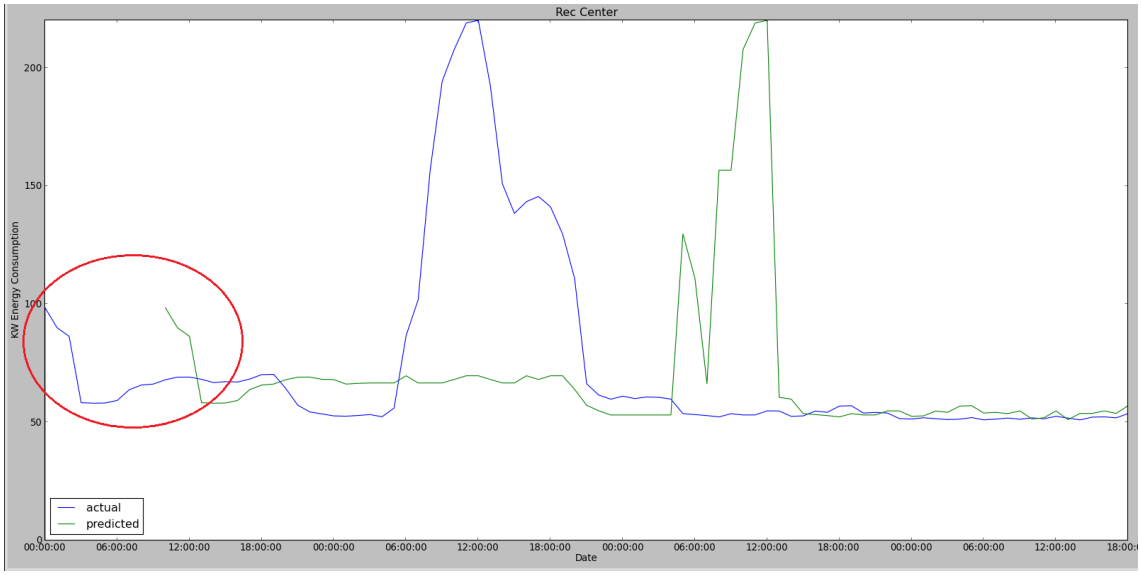

Imagen IV.7 – Caso B, 0 iteraciones

Lo primero que observamos nada más ejecutar el programa es que no se generan datos de predicción (línea verde), hasta que pasan 10 horas (10 pasos). Esto es lógico ya que al estar haciendo una predicción de 10 horas en el futuro NuPIC necesita de al menos 10 muestras de consumo anteriores para poder hacer la primera predicción. Este hecho además hace que al comienzo toda la perdición esté desfasada 10 horas. El pico de consumo corresponde a un viernes seguido del sábado y el domingo que son días de bajo consumo. El primer cambio puede apreciarse cuando NuPIC realiza la predicción del viernes, de manera errónea debido al desfase de 10 horas, y la interrumpe de manera abrupta cambiando el modelo al ver que no coincide su predicción con el consumo real actual.

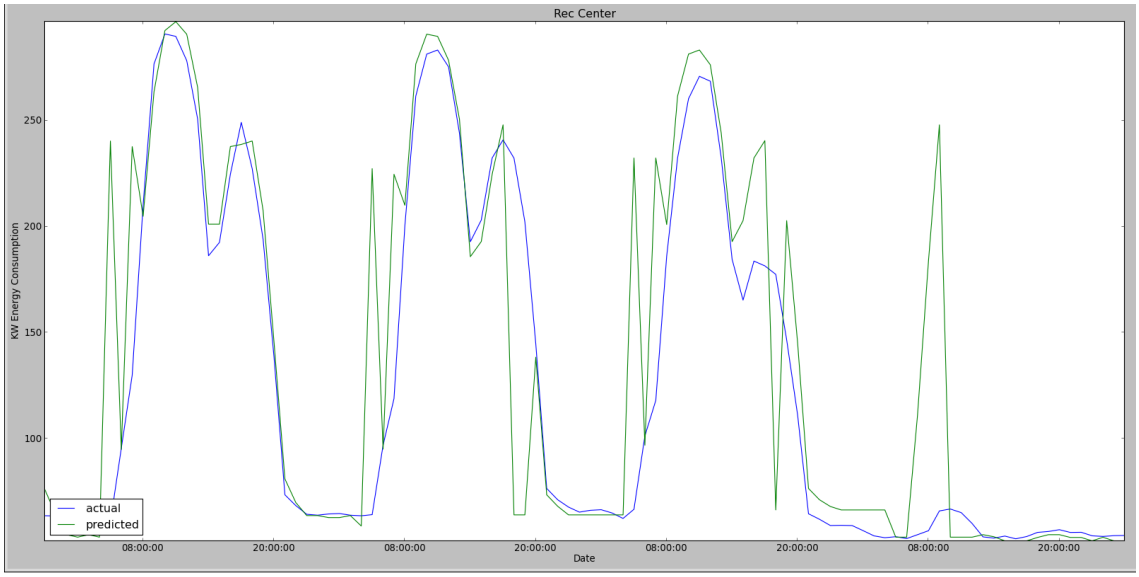

Imagen IV.8 – Caso B, 200 iteraciones

Con sólo 200 interacciones NuPIC ya consigue corregir el desfase inicial sin embargo aún no es capaz de dar una predicción estable. Tampoco responde bien a los cambios como se puede observar en el último pico de predicción que corresponde a un sábado en el que una vez más el algoritmo predice un consumo de día laboral, no obstante la velocidad a la que responde corrigiendo la predicción es una muestra del potencial que tiene NuPIC.

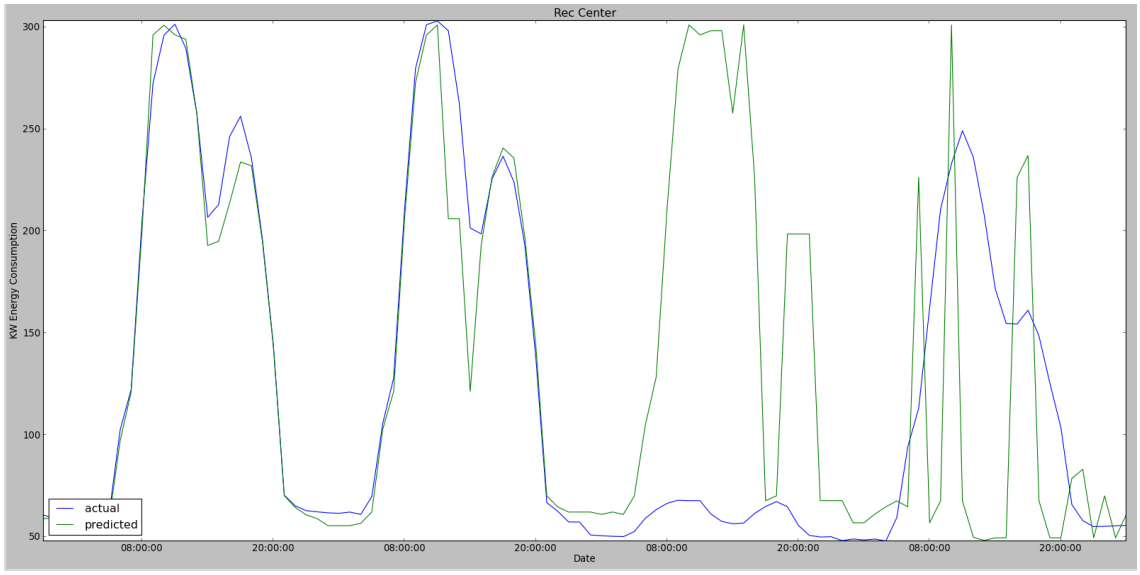

Imagen IV.9 – Caso B, 800 iteraciones

Con 800 interacciones (cerca de las 900), nos encontramos en el mismo escenario que en el experimento anterior en donde tenemos un jueves festivo. Los días que aparecen en esta gráfica por orden de izquierda a derecha corresponden al martes, miércoles, jueves y viernes. En el caso de la predicción de una hora se pudo observar como el algoritmo conseguía corregir de manera eficaz la predicción y se ajustaba al evento inesperado. No ocurre lo mismo aquí en donde podemos observar que esta vez no es capaz de dar respuesta al evento. Además es algo que hace desestabilizar el modelo generando ese diente de sierra el viernes.

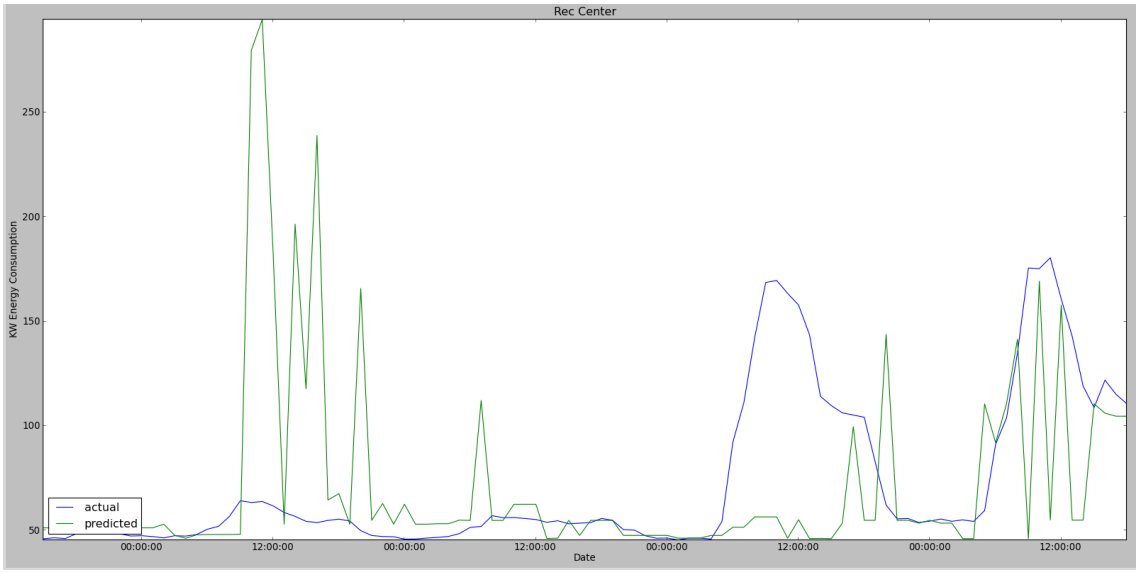

Imagen IV.10 – Caso B, 1300 iteraciones

Durante el período de vacaciones de Navidad el sistema se descompensa totalmente. No consigue reaccionar ni ajustarse de manera correcta, al contrario que en el experimento anterior, generando una predicción totalmente errónea.

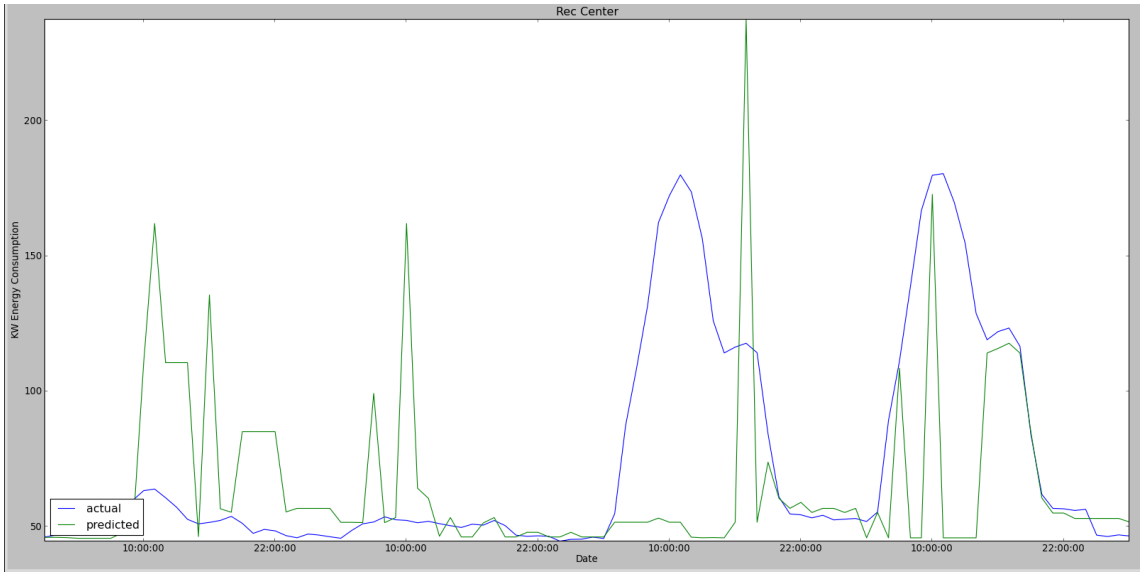

Imagen IV.11 – Caso B, 1500 iteraciones

200 interacciones más adelante el sistema sigue descompensado. El período de vacaciones de Navidad cambió el modelo dando la impresión de que NuPIC tiene que empezar desde cero para poder mejorar la perdición aunque sabemos que esto no es cierto, el programa sigue aprendiendo con cada nueva entrada que recibe.

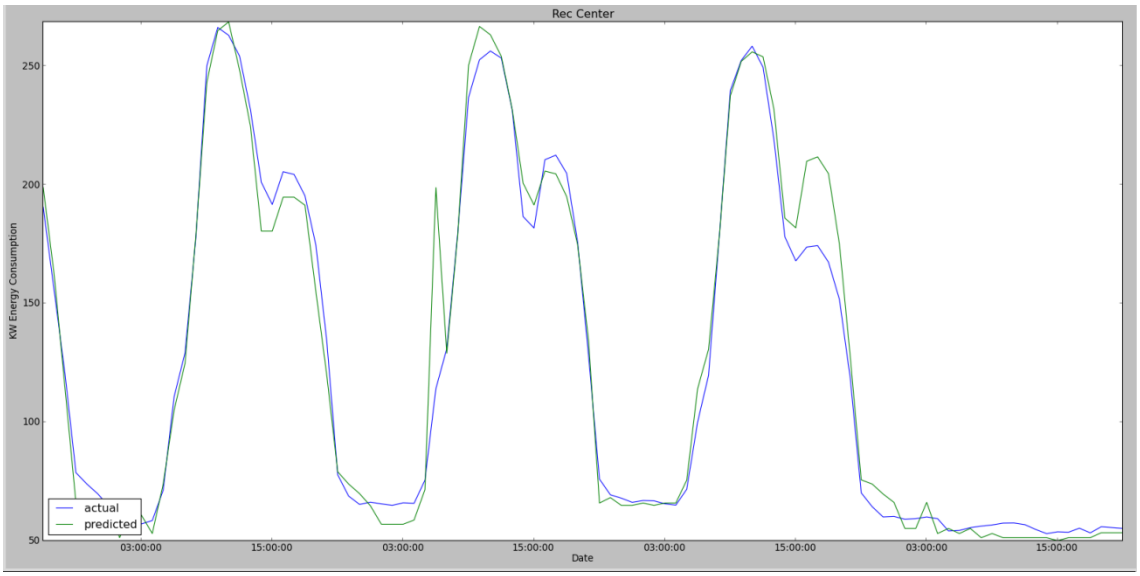

Imagen IV.12 – Caso B, 2400 iteraciones

Son necesarias 900 interacciones para que el sistema vuelva estabilizarse y genere una predicción bastante acertada. Aunque el período de vacaciones cambiara el modelo descompensando el sistema, NuPIC encuentra patrones comunes dentro del modelo que facilitan la recuperación. Este caso puede verse como un ejemplo de memoria en el que el algoritmo "recuerda" la secuencia de consumo.

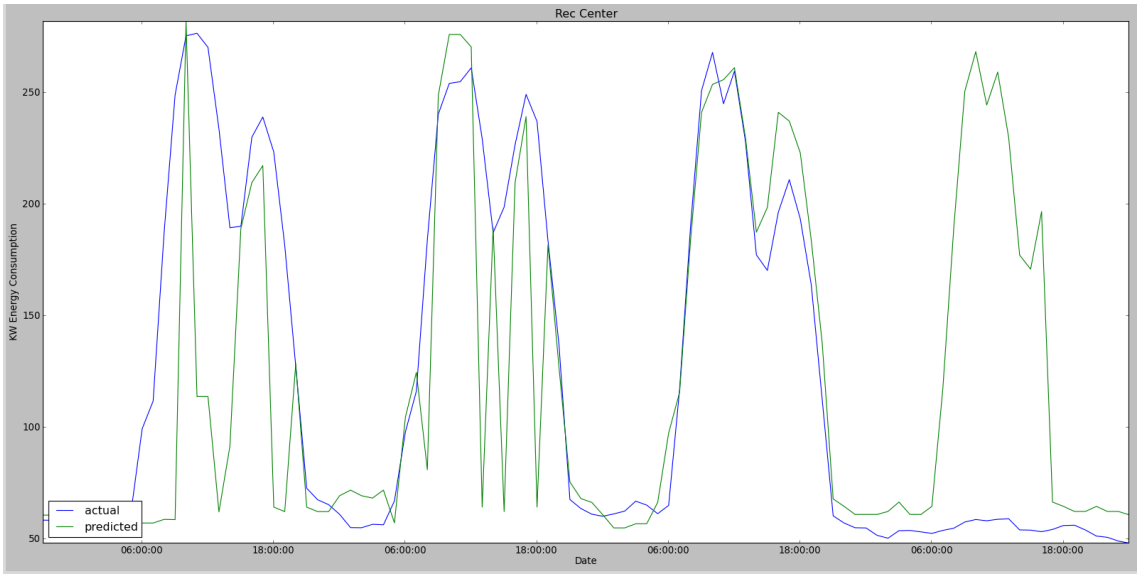

Imagen IV.13 – Caso B, 3500 iteraciones

Aunque el modelo ya se había estabilizado, en esta gráfica podemos observar que el sistema se descompensa con facilidad después de un evento inesperado, seguramente un día festivo en jornada laboral. Aunque el error es bastante acusado cabe esperar que el algoritmo se recupere de manera más rápida que en el caso anterior gracias al aprendizaje.

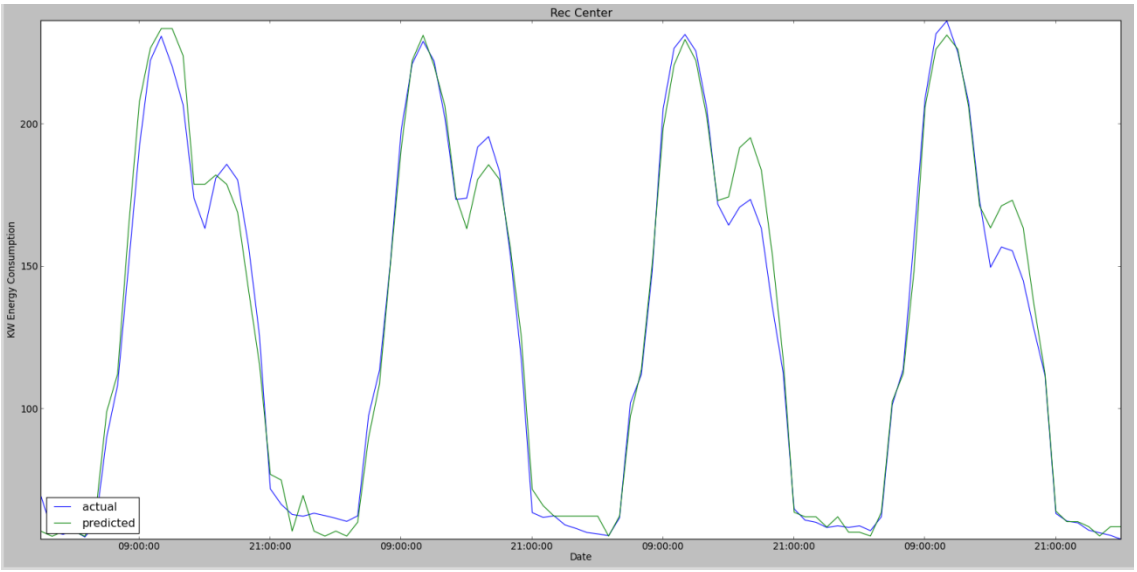

Imagen IV.14 – Caso B, 5500 iteraciones

Para terminar, ya casi al final de la ejecución del programa, con 5500 interacciones el sistema es bastante estable y genera una predicción ajustada. En este experimento se observa mejor cómo responde NuPIC y cómo influye el aprendizaje en la creación del modelo. Prueba de ello es que durante la ejecución se puede observar que aunque el sistema se descompensa con los eventos inesperados, éste se recupera de manera más rápida a medida que aumentan las iteraciones.

Para el cálculo del error se ha seguido el mismo procedimiento que en el experimento anterior. Utilizando los datos de predicción generados durante la ejecución del programa se calcula el error cuadrático medio y el error porcentual medio desde el principio de la ejecución y a partir de 3000 iteraciones.

ECM	ECM AT	ERROR %	ERROR % AT
2028.3	1666.0	19.3	17.8

Como se refleja en el cálculo del error la predicción es mucho peor que en el experimento anterior. Esto es algo lógico ya que predecir algo es mucho más difícil cuanto mayor es la distancia en el tiempo que nos separa del momento presente. Es más sencillo acertar lo que ocurrirá en el instante justamente posterior porque son pocas las cosas que pueden cambiar de un momento a otro. Sin embargo cuanto más lejos en el futuro queremos hacer la predicción son muchos más los eventos que pueden ocurrir que hacen que nuestra perdición falle.

Un ejemplo práctico es que cualquiera de nosotros puede intentar predecir el tiempo que hará mañana y acertar, pero tendríamos dificultades para predecir el tiempo que hará dentro de una semana, un mes o incluso un año.

En comparación con el experimento anterior, recordando que se han utilizado los mismos datos, cuando tratamos de predecir el consumo eléctrico 10 horas en el futuro, aunque al final se consigue un modelo estable de predicción, el sistema responde peor ante los eventos inesperados y tarda mucho más en recuperarse que en el caso anterior. En los días festivos no consigue ajustar el consumo prediciendo el valor correspondiente a una jornada laboral aumentando de manera considerable el error.

Aunque analizando las capturas pueda parecer que la predicción falla la mayor parte del tiempo hay que aclarar que sólo se ha mostrado una parte pequeña del proceso. Durante el resto el proceso la predicción es bastante ajustada. Los problemas que tiene el modelo para dar respuesta a los eventos inesperados y ajustar el consumo desaparecerían si se aumenta el tiempo de aprendizaje, es decir si aumentamos el número de iteraciones dándole más datos de consumo. Si en vez de alimentarle con ocho meses de datos de consumo trabajamos con 12, 14 o 18 meses, veríamos como el modelo mejoraría drásticamente.

IV 4.3 Caso C: predicción de 15 minutos y detección de anomalías

En este último experimento que se recoge en este documento se ha creado un modelo de predicción de 15 minutos. En este caso los datos de consumo son diferentes de los utilizados en los anteriores experimentos. El periodo de actividad sigue siendo el mismo, las muestras van desde el 1 de noviembre de 2012 hasta el 30 de junio de 2013, sin embargo en esta ocasión los datos están espaciados cada 15 minutos, con lo que el número de muestras se multiplica por cuatro contando con un total de 23220 muestras.

Esto supone un cambio más que notable en el resultado de la predicción por dos motivos. El primero es el aumento considerable de los datos de entrada lo que conlleva a que el modelo creado en el proceso de swarming sea más completo y por lo tanto se ajuste mejor a los cambios de consumo eléctrico. El segundo motivo es que los datos de consumo están espaciados cada 15 minutos. Esto significa que las entradas cambian más despacio que en los casos anteriores, concretamente cuatro veces más despacio, algo que consigue que la predicción sea un poco más sencilla.

En la parte teórica de este documento se ha mencionado que para que las HTM puedan inferir algo es necesario que las entradas cambien en el tiempo. No obstante cuanto más despacio cambien estas entradas, más fácil nos será hacer una predicción. Esto es lógico en cuanto a que nos es más sencillo seguir la evolución de una cosa cuanto más despacio cambia esta. Si por ejemplo en este instante tenemos un tiempo soleado podemos predecir que dentro de una hora seguirá estando igual. Si por el contrario a los 15 minutos vemos como empiezan aparecer unas pocas nubes, 15 minutos más tarde esas nubes se oscurecen, y 15 minutos más tarde todo el cielo está cubierto por nubes negras nuestra perdición cambiará y podremos asegurar que cuando se cumpla una hora desde que el sol brillaba en lo alto, éste no se mantendrá sino que acabará lloviendo y con bastante fuerza.

Cabe explicar que cuando decimos que las entradas cambien despacio no nos referimos únicamente al tiempo sino que además estos cambios se produzcan de una manera progresiva, como es el caso del consumo eléctrico de una escuela universitaria. Por muy despacio que cambie una entrada de nada nos servirá si está cambia de forma abrupta y no somos capaces de encontrar una relación entre el instante anterior y el instante justamente posterior.

Siguiendo con el último de los experimentos, en el proceso de swarming se ha utilizado el tipo de inferencia “multistep”. Antes de la ejecución del programa para poder mostrar la anomalía se cambió el tipo de inferencia, editando el archivo model_params.py generado en el proceso de swarming, por el de “temporalAnomaly”. Se han utilizado todos los datos fijando el valor de “iterationCount” a -1 y se ha definido un tamaño de swarming “large”. La predicción que se quiere realizar es de 15 minutos por lo que establecemos el valor de “predictionStep” a 1. El tiempo de ejecución durante la creación del modelo ha sido de 14 horas y 44 minutos.

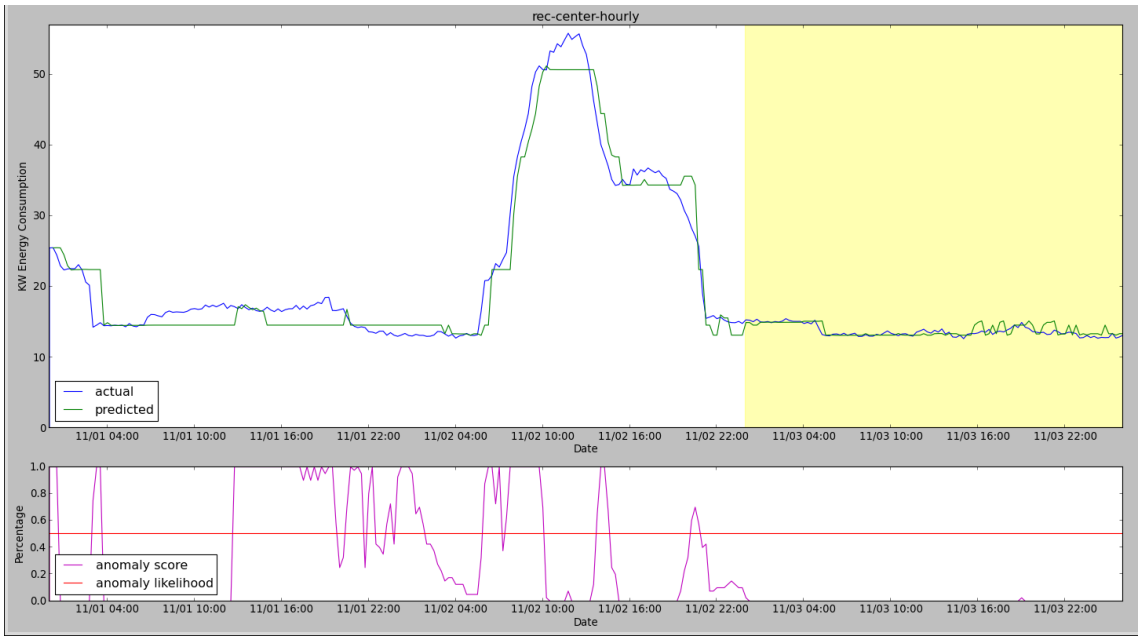

Imagen IV.15 – Caso C, 200 iteraciones

En la ventana de arriba de esta captura tenemos la predicción que se muestra de la misma manera que en los anteriores experimentos, la línea de color azul corresponde al consumo actual mientras que la línea de color verde corresponde a la predicción generada por NuPIC. Para facilitar la comprensión de los datos que se muestran en la gráfica, en este experimento se ha añadido un módulo que sombrea con el color amarillo claro en la parte de arriba los días correspondientes al sábado y al domingo, días en los que no hay prácticamente actividad y por lo tanto tienen un consumo menor.

En la ventana de abajo tenemos la detección de anomalías. En color morado tenemos el valor de anomalía (anomaly score) con un rango de valor que va del 0 al 1. Un 0 corresponde a un valor de predicción esperado mientras que 1 significa un valor de predicción inesperado. En color rojo tenemos el "anomaly likelihood" que nos proporciona una medida de la veracidad de la anomalía detectada. En este caso un valor de anomaly likelihood cercano al 0 indicará que la anomalía que se haya detectado en ese instante es un falso positivo, mientras que un valor cercano al uno significará que la anomalía detectada en ese instante es cierta.

Si nos fijamos en esta captura en el valor del anomaly likelihood (línea roja) nos damos cuenta de que se mantiene en un valor constante del 50% (0.5). Esto es debido a que el algoritmo necesita una serie de iteraciones antes de poder hacer una estimación.

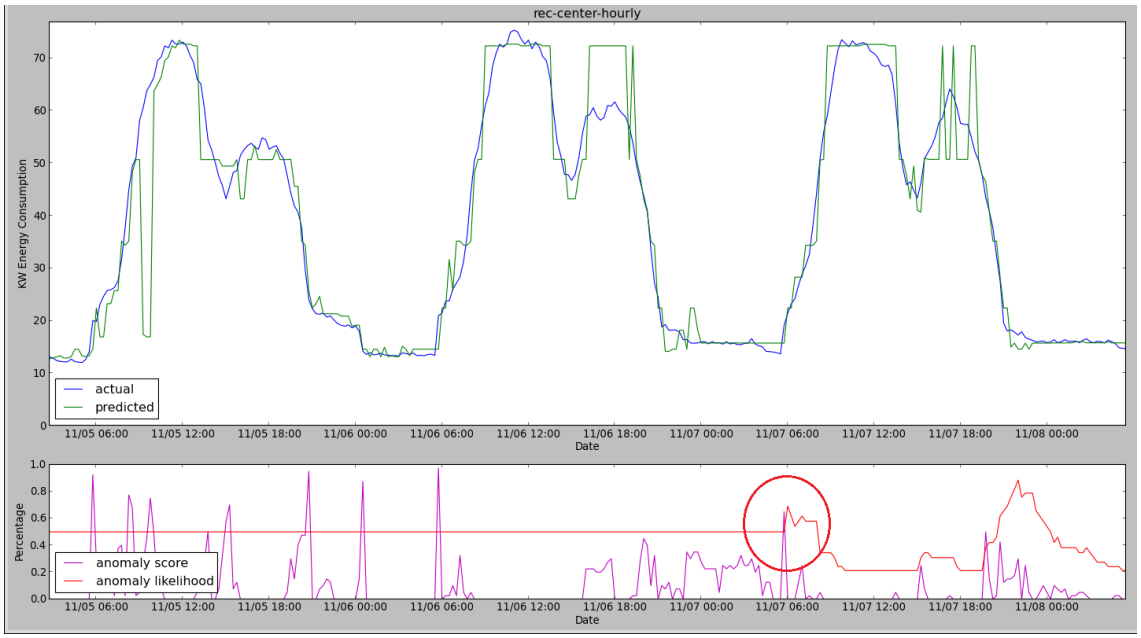

Imagen IV.16 – Caso C, 600 iteraciones

Después de 600 iteraciones el algoritmo ya nos puede dar una estimación del valor del anomaly likelihood. Ya desde el principio se observa un mejor ajuste de la predicción que en los experimentos anteriores aun cuando el programa acaba de ejecutarse. Recordemos que lleva 600 iteraciones de 23220 muestras.

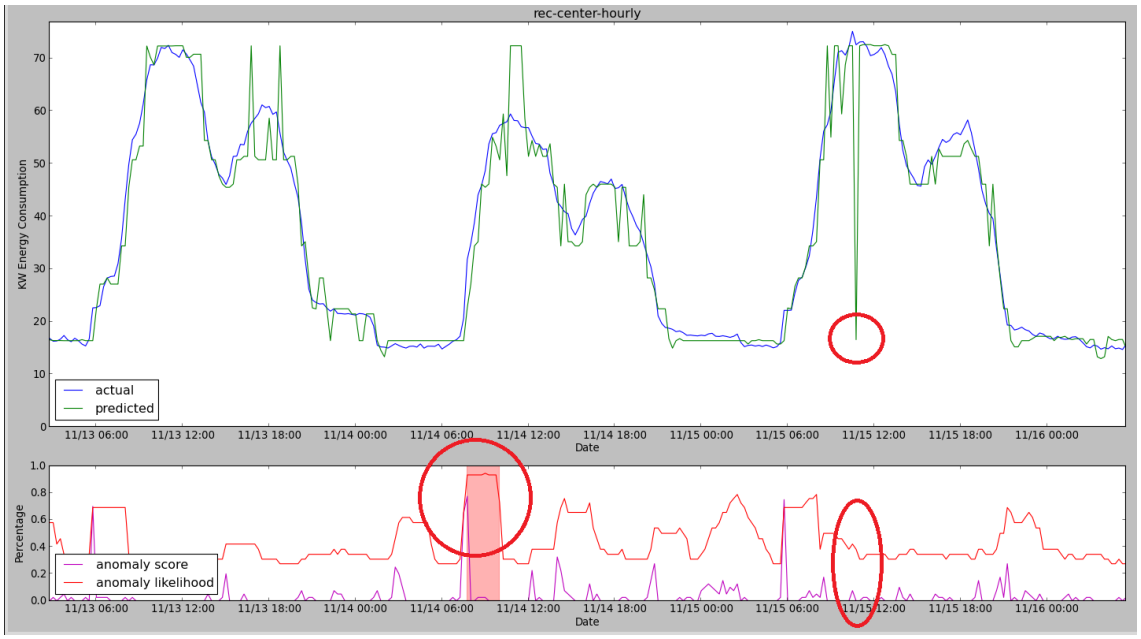

Imagen IV.17 – Caso C, 1400 iteraciones

En esta captura podemos apreciar un caso detección de anomalías consecuencia de un consumo menor de lo habitual en una jornada laboral. Esta disminución del consumo eléctrico supone un evento que NuPIC no se esperaba. En la ventana de abajo para facilitar al usuario la detección de anomalías, el mismo módulo que se encarga de sombrear de amarillo los días correspondientes al sábado y domingo, sombrea de color rojo todo periodo en el que el valor de anomaly likelihood supera el 90% (0.9), lo que indica que el valor de anomaly score es un valor veraz y debe tomarse en cuenta. En este caso se tiene un valor de anomaly score cercano al 80%.

Un poco más a la derecha, enmarcado en rojo nos encontramos con un valor de predicción que se sale totalmente de lo que debería indicar el modelo pero que sin embargo como se observa en la ventana de abajo no produce anomalía. Esto es un ejemplo que responde a lo que se explicaba más arriba de que por el hecho de que NuPIC falle la predicción no significa que se haya producido una anomalía.

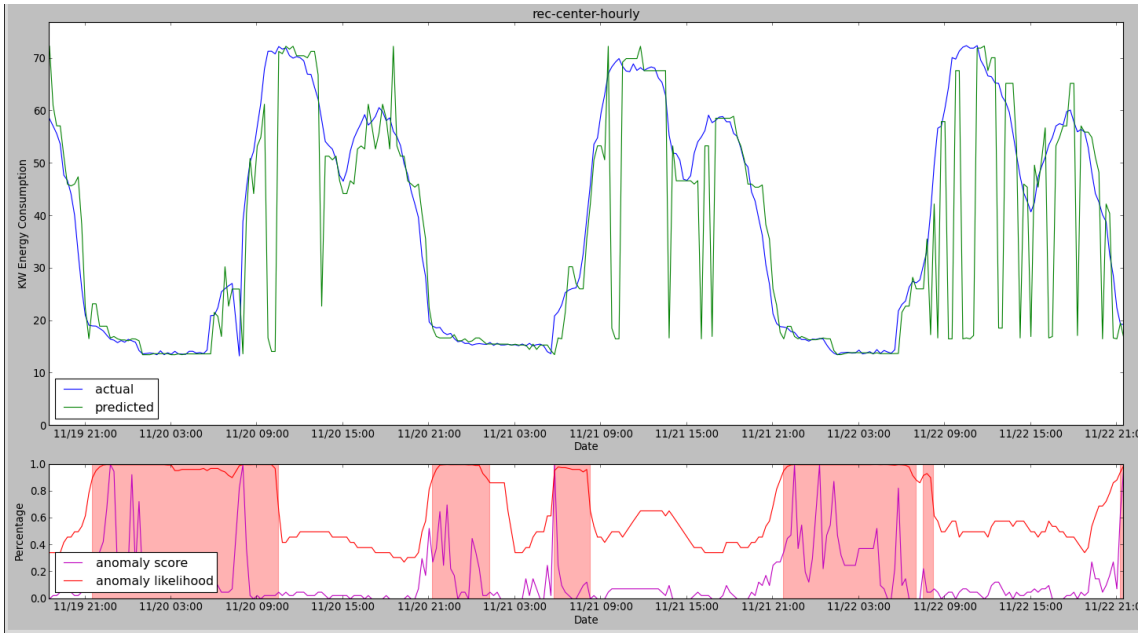

Imagen IV.18 – Caso C, 2100 iteraciones

Con 2100 iteraciones el modelo que se está creando intenta ajustarse a la predicción. Aunque se observan puntos de predicción que difieren totalmente del consumo actual esto sólo se produce en momentos puntuales y se corrige rápidamente. Si nos fijamos en la ventana de abajo aparecen indicios de anomalía. Esto sucederá con frecuencia durante la ejecución del programa sin embargo en muy pocas ocasiones puede encontrarse una relación directa entre la predicción y la de detección de la anomalía.

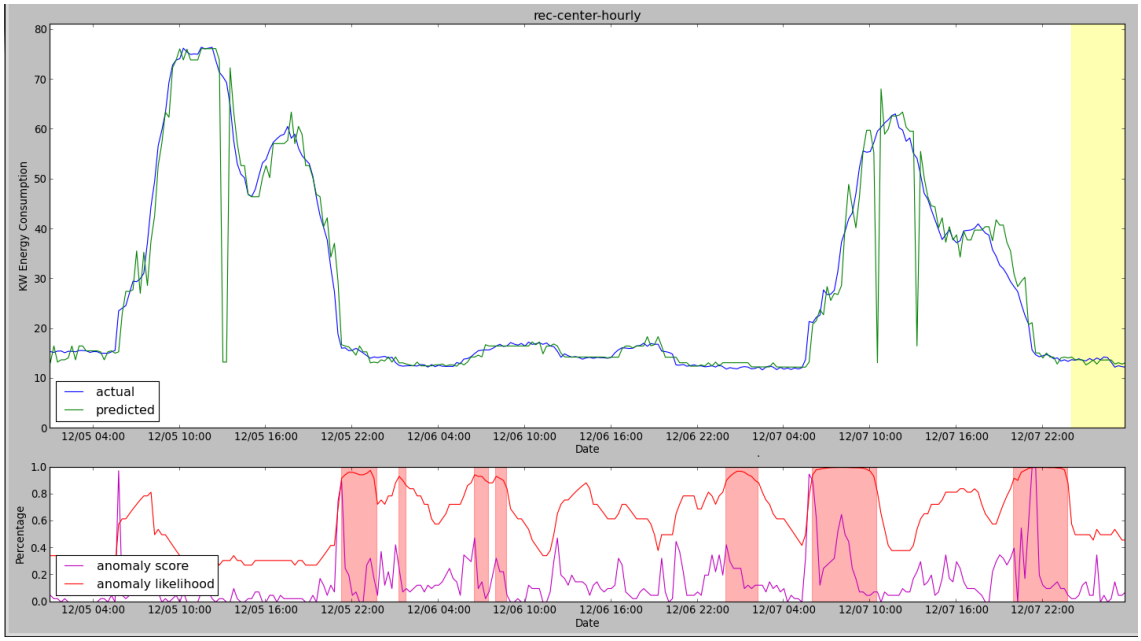

Imagen IV.19 – Caso C, 3500 iteraciones

Con 3500 iteraciones nos situamos en el mismo escenario que en los experimentos anteriores en el que tenemos un jueves festivo, que corresponde al hueco que existe entre los dos montículos. En este caso la respuesta es mucho más eficaz que en los anteriores experimentos ajustando la predicción de una manera que apenas genera errores.

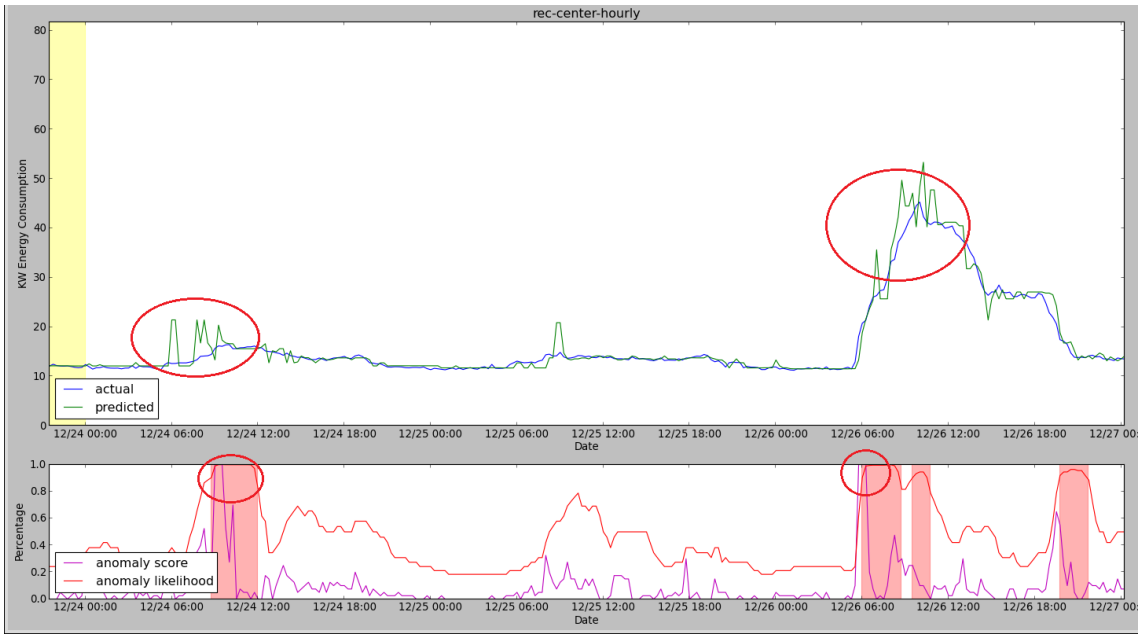

Imagen IV.20 – Caso C, 5300 iteraciones

Esta captura se sitúa en el inicio de las vacaciones de Navidad, en concreto el día 24 de diciembre de 2012. Enmarcado en rojo en la zona izquierda se puede observar como NuPIC intenta predecir un consumo normal de día laboral pero una vez más sorprende la rapidez con la que reajusta el sistema. Estos días festivos no son esperados por el algoritmo algo que sí que podemos interpretar en la detección de anomalía enmarcada en rojo de la izquierda. El día 26 en donde se genera algo de consumo NuPIC vuelve intentar predecir el consumo de un día laboral y sucede lo mismo que ha ocurrido en el día 24.

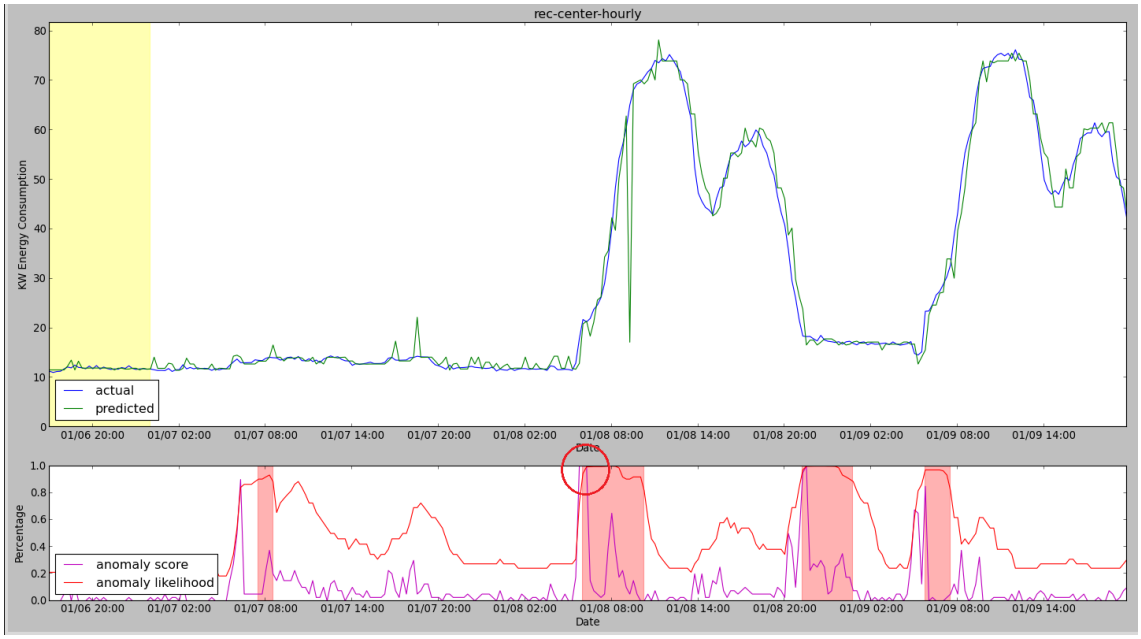

Imagen IV.21 – Caso C, 6700 iteraciones

Al finalizar las vacaciones de Navidad, con 6700 iteraciones, en mitad de la captura observamos como al retomar el consumo habitual del periodo escolar NuPIC no tiene ningún problema para ajustar la predicción. En la detección de anomalías podemos interpretar como este cambio no era algo previsto, algo lógico teniendo en cuenta que en la escuela ha pasado semana y media sin apenas actividad.

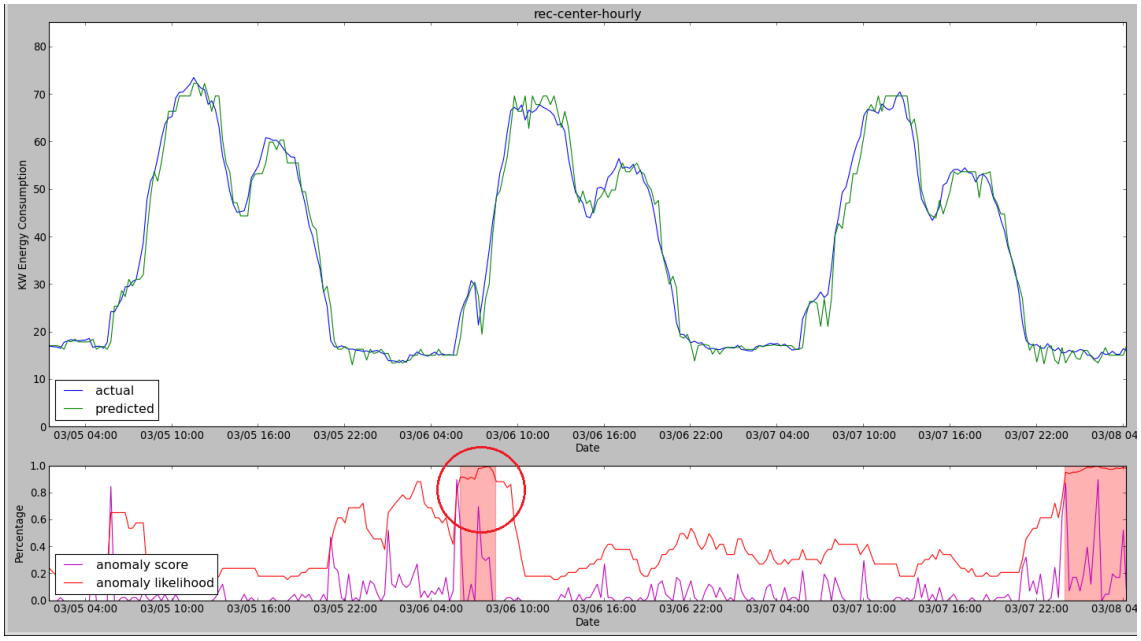

Imagen IV.22 – Caso C, 12200 iteraciones

Esta captura en concreto es interesante ya que nos sirve para comprobar la eficacia de la detección de anomalías. Ya se ha mencionado que durante la ejecución del programa aparecen signos de anomalías que difícilmente podemos interpretar ya que no se observa una relación directa sobre la predicción. En este caso contamos con un correo que en su momento mandó la escuela avisando de que con motivo de una revisión de los transformadores eléctricos de alta tensión se iba a producir un corte de corriente el día 6 de marzo entre las 7:15 y 7:30 horas.

Esta información nos proporciona una gran ventaja a la hora de comprobar la eficacia de NuPIC en la detección de anomalías ya que nos permite ir directamente a la fecha señalada y comprobar el resultado. Como se observa en la gráfica en el instante indicado aparece un anomaly score de 70% con un anomaly likelihood muy cercano al 100%. Debemos aclarar que aunque se produjo un corte del suministro eléctrico desconocemos qué duración tuvo este. Lo más seguro es que el corte durara unos pocos minutos ya que el consumo reflejado en esos 15 minutos es solo un poco más bajo que en las muestras anterior y posterior.

timestamp	kw energy	prediction	anomaly score	anomaly likelihood
2013-03-06 07:00:00	29,903	27,705	0	0.90
2013-03-06 07:15:00	21,532	19,583	0.7	0.98
2013-03-06 07:30:00	26,229	27,091	0.325	0.99

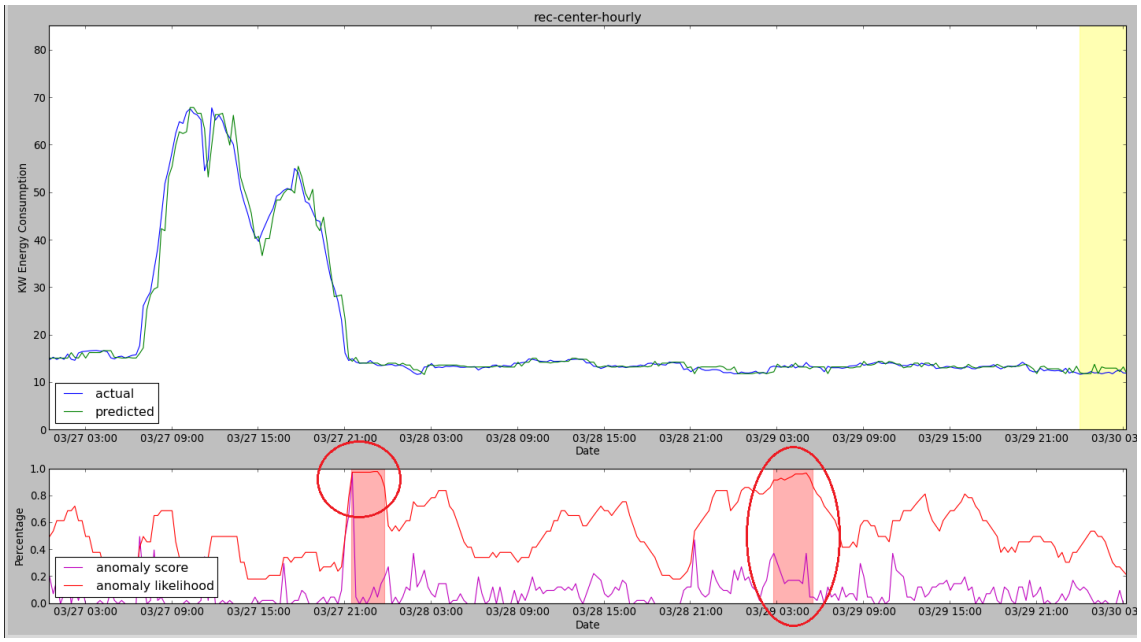

Imagen IV.23 – Caso C, 14300 iteraciones

Avanzando 2100 iteraciones en el tiempo nos situamos en otro evento inesperado. En este caso se corresponde con las vacaciones de semana Santa. Aquí ya se puede apreciar un claro ejemplo del aprendizaje que se ha llevado a cabo durante todo el proceso en que como se puede observar el sistema se ajusta perfectamente al período de bajo consumo que corresponde a las vacaciones. Sólo se detecta un pico de anomalía en el cambio de jornada. El círculo rojo de la derecha no supondría una anomalía ya que recordemos que para que tengamos en cuenta la detección de anomalía ambos valores tanto el anomaly score como el anomaly likelihood deben de estar cercanos al 1.

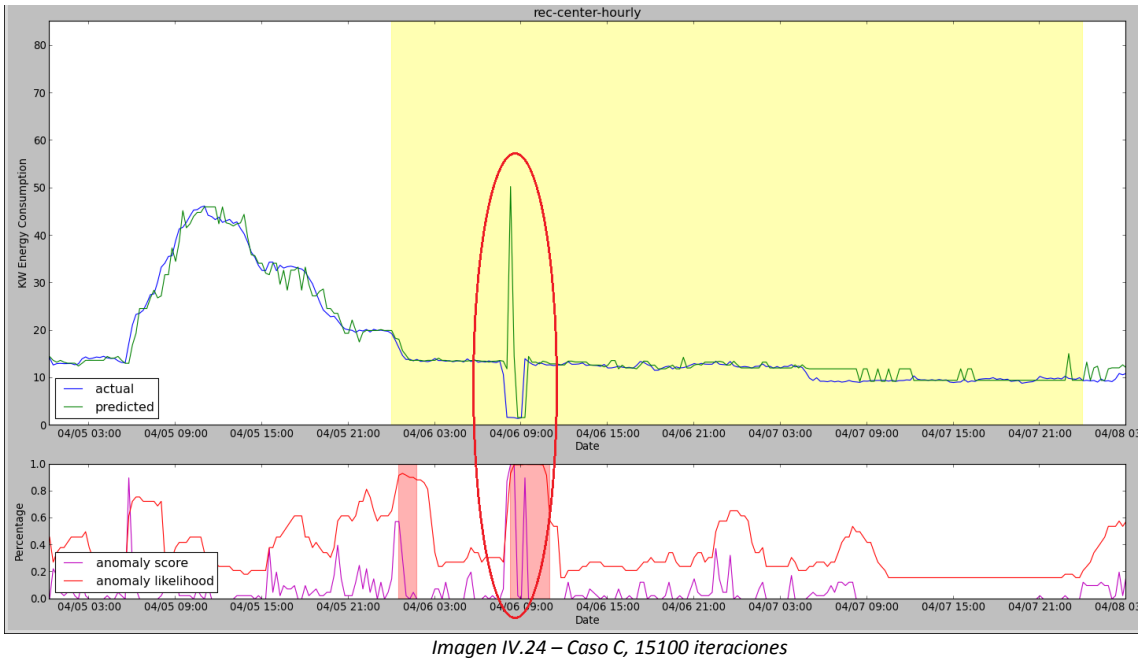

Imagen IV.24 – Caso C, 15100 iteraciones

En esta gráfica podemos observar lo que debió ser un corte del suministro eléctrico de una duración cercana a los 45 minutos. En este caso tan evidente la anomalía se dispara y el algoritmo proporciona un pico de predicción totalmente anormal debido lo más seguro a que no encontró un patrón similar en el modelo que responda a este evento.

De la misma manera que los experimentos anteriores realizamos el cálculo del error.

ECM	ECM AT	ERROR %	ERROR % AT
10.2	3.2	3.9	3.6

Observando las gráficas durante la ejecución del programa y fijándonos en el cálculo del error podemos asegurar que la predicción es bastante buena. Con este último cálculo también podemos comprobar que aunque NuPIC tiene una capacidad de aprendizaje en línea muy potente, ésta prácticamente no es necesaria si se ha creado previamente un modelo con abundante información de entrada, ya que si nos fijamos en la diferencia que hay entre el porcentaje de error antes y después del entrenamiento, ésta es bastante pequeña.

Así como los anteriores experimentos generaron una predicción con un alto porcentaje de error, algo que no es viable para la predicción del consumo eléctrico, en este experimento se pone de manifiesto el hecho de que contando con los datos necesarios se puede llegar a crear un modelo que proporcione un rendimiento más que aceptable.

V. RESUMEN, CONCLUSIONES Y LÍNEAS FUTURAS

V 1. Resumen

En el presente proyecto se ha utilizado el software NuPIC y se han evaluado sus algoritmos para la predicción y la detección de anomalías con el fin de comprobar su utilidad en el campo de la ingeniería, concretamente en la predicción del consumo eléctrico.

Para ello se ha hecho un estudio previo de un tipo de inteligencia artificial denominada memoria temporal jerárquica o HTM (del inglés *"Hierarchical Temporal Memory"*), que nace de la teoría desarrollada por Jeff Hawkins, en la que se basa el software NuPIC.

El software se ha instalado en un sistema operativo Ubuntu y se ha utilizado un pequeño programa desarrollado en lenguaje Python que genera una predicción del consumo eléctrico en función de datos de consumo reales.

Los datos utilizados corresponden a meses de datos de consumo eléctrico generado en la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación de la Universidad de Cantabria durante los años 2012/13. Estos datos abarcan las fechas del 1 de noviembre de 2012 hasta el 30 de junio de 2013 en dos formatos diferentes. Con muestras de consumo eléctrico espaciadas una hora y espaciadas 15 minutos.

Para comprobar la eficacia de este software tanto para la predicción del consumo eléctrico como la detección de anomalías, se han llevado a cabo tres experimentos utilizando los datos mencionados. Un primer experimento con los datos espaciados cada hora para realizar una predicción de una hora en el futuro del consumo eléctrico. Un segundo experimento con los datos espaciados cada hora para realizar una predicción de 10 horas. Y un último experimento con los datos espaciados cada 15 minutos para realizar una predicción de 15 minutos y evaluar la detección de anomalías.

Para realizar la predicción previamente se ha creado para cada uno de los experimentos un modelo de predicción mediante el proceso de "swarming". Además para proporcionar una medida de la eficacia del software se ha calculado el error cuadrático medio y el porcentaje medio de error que se comete en la predicción.

Con el fin de tener una estimación de la función de aprendizaje del software NuPIC el cálculo del error se ha realizado antes y después del entrenamiento. Para completar el resultado y evaluar el comportamiento del software se han analizado las gráficas que se generan durante la ejecución del programa en las que se muestran el consumo real y la predicción realizada fijándonos en los escenarios en los que suceden eventos inesperados como son días festivos o periodos de vacaciones escolares, situaciones en las que NuPIC puede dar solución a un problema que tienen los métodos estadísticos que se utilizan en la predicción del consumo eléctrico.

V 2. Conclusiones

El error que se comete en la predicción en los experimentos realizados con datos espaciados cada hora es demasiado grande, algo que lo hace inviable para un ejercicio de

predicción de consumo eléctrico. El problema reside en el modelo creado en la fase de swarming. Al modelo se le proporcionaron 5808 muestras de consumo eléctrico espaciadas una hora. Estos datos que podrían ser más que suficientes para un pequeño edificio en el que el consumo sea bastante constante, no son suficientes para unas instalaciones tan grandes como una escuela universitaria en donde se ha comprobado que el consumo es muy variable y depende de muchos factores.

Una mayor cantidad de datos o una disminución del tiempo entre muestras mejoraría el modelo creado, permitiendo al software realizar una predicción más ajustada reduciendo considerablemente el error. Aunque los resultados de estos primeros experimentos no sean satisfactorios para la predicción del consumo eléctrico nos han servido para evaluar el comportamiento del software NuPIC en el campo del aprendizaje. A lo largo de la ejecución del programa se ha podido observar cómo responde el algoritmo ante eventos inesperados como puede ser una disminución del consumo puntual, un día festivo o periodos prolongados de vacaciones.

En estos escenarios la respuesta del algoritmo ha sido la de rectificar su predicción en función de la entrada de consumo actual que está recibiendo en ese instante. Internamente en el modelo que se está creando se genera un patrón nuevo que es añadido a este y que servirá en futuros eventos inesperados. Tras uno de estos eventos el sistema se descompensa y le lleva un cierto número de interacciones ajustarse de nuevo. La capacidad de aprendizaje del algoritmo se pone de manifiesto al incluir los patrones nuevos que se generaron en los eventos inesperados ya que como se observa en la ejecución del programa cada vez que encuentra un evento nuevo NuPIC responde mejor y se recupera con mayor rapidez.

El último experimento que se realizó con datos espaciados 15 minutos mostró un error de predicción inferior al 4%. Para este experimento se creó un modelo utilizando 23220 muestras de consumo eléctrico generando un modelo muy completo durante la fase de swarming. Prueba de esto es que la diferencia entre el error calculado antes y después del entrenamiento es de un 0.3%. En este caso la respuesta ante eventos inesperados es muy eficaz y el sistema solo se descompensa en contadas ocasiones en las que se recupera prácticamente al instante.

El algoritmo de detección de anomalías es bastante eficaz. Se han hecho pruebas con datos reales y datos manipulados con el fin de verificar su funcionalidad. Cada vez que NuPIC encuentra un patrón nuevo, indica que está ocurriendo algo que no había visto antes, es decir muestra anomalía. Ha sido difícil contar con un registro de anomalías del consumo eléctrico que pudiera servirnos en este experimento ya que la mayoría de estas anomalías tienen una duración de unos pocos milisegundos algo que pasa totalmente desapercibido entre muestras de 15 minutos. No obstante pudimos contar con una fecha señalada en la que hubo un corte de suministro lo suficientemente largo como para que el programa pudiera detectarlo y aunque apenas se aprecia una diferencia del consumo eléctrico NuPIC consiguió detectar la anomalía catalogándola con un 70%.

La conclusión es que NuPIC es una herramienta de gran potencial. Las propiedades que proporcionan las HTM para la predicción, el aprendizaje y la detección de anomalías son de gran interés no sólo para el campo de la predicción del consumo eléctrico sino que puede utilizarse para cualquier propósito de la ingeniería que requiera estas propiedades.

Por contra debemos comentar que NuPIC es un software joven, tiene poco más de un año, y aunque detrás tiene una comunidad volcada que trabaja para mejorar el software aparte de los propios ingenieros de Numenta, a día de hoy la herramienta está bastante

limitada. La predicción del consumo eléctrico depende de muchos factores entre los que se pueden encontrar la temperatura y la radiación solar. Un objetivo del proyecto era realizar una predicción del consumo teniendo en cuenta estos factores pero a día de hoy el software no lo permite. Tampoco existe una documentación de las estructuras y algoritmos que contiene NuPIC algo que dificulta enormemente trabajar con esta herramienta. No obstante no cabe duda que todos estos problemas se solucionan en un futuro por lo que es interesante seguir la evolución de este software.

V 3. Líneas futuras

El presente proyecto deja abiertas varias líneas futuras de investigación:

El Software NuPIC es una herramienta potente que crece día a día con las aportaciones de los ingenieros de Numenta y de una comunidad interesada y constante de usuarios. Sin embargo a día de hoy NuPIC tiene algunas limitaciones. En el campo de la predicción, ya no sólo en el del consumo eléctrico sino en general, sería muy interesante poder trabajar con un mayor número de variables, algo que todavía no está implementado. Es decir, si por ejemplo queremos hacer la predicción del consumo eléctrico podamos añadirle datos de temperatura o de radiación ya que estos inciden de alguna manera sobre el consumo generado. Si un día laboral el sol brilla con fuerza se reducirá el consumo de luz y si un día de invierno suben las temperaturas se reducirá el consumo de calefacción.

Otra limitación surge de la cantidad de recursos que requiere el proceso de creación de modelos. En los experimentos realizados se ha comprobado que cuantos más datos se utilizan para crear el modelo mejor es la predicción. El departamento de instalaciones nos proporcionó una lista extensa de anomalías que se producían en el suministro eléctrico. El problema de estas anomalías es que tenían una media de duración de milisegundos algo que es imposible de reconocer con los modelos creados en este proyecto. Un objetivo que se podría aplicar al software es el de mejorar la eficiencia del proceso de swarming para poder crear modelos con una mayor cantidad de datos.

Otro problema que hace que sea difícil trabajar con esta herramienta es el hecho de que no existe una documentación estructurada del software en general. Existe un libro y varios documentos que explican los fundamentos teóricos de las HTM, sin embargo es escasa la información disponible sobre las funciones, algoritmos y estructuras que utiliza NuPIC. Existe una wiki en donde se recoge toda la información junto con algunos tutoriales y códigos de ejemplo en donde poco a poco se va añadiendo información.

NuPIC consigue solucionar el problema que existe en la predicción del consumo eléctrico con métodos estadísticos como la regresión lineal, modelos ARIMA o modelos de descomposición, en los escenarios en los que surgen eventos inesperados como son los días festivos. Estos métodos estadísticos incrementan su error porque predicen un consumo normal un día en el que no va a haber jornada laboral y por lo tanto el consumo es mucho menor. Como se ha demostrado en este proyecto, NuPIC reconoce los eventos inesperados y es capaz de rectificar su predicción. Esto le da al software NuPIC una gran ventaja sobre los modelos estadísticos, sin embargo para demostrar que el sistema es mejor en la predicción del consumo habría que compararlos utilizando los mismos datos con el fin de ver qué sistema da una mejor respuesta.

Uno de los objetivos que se perseguía con este proyecto era el de evaluar el potencial de esta herramienta para futuras aplicaciones. Una de estas aplicaciones en concreto se centra en

el campo de la óptica. El objetivo de esta aplicación sería el de utilizar esta herramienta para analizar la nube de puntos que genera un sistema LIDAR (del inglés "*Laser Imaging Detection and Ranging"*), instalado en el puerto y que controla el tráfico de los barcos.

Un sistema LIDAR es una tecnología que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada. Si el LIDAR realiza un barrido constante sobre la entrada del puerto en la que no hay ningún objeto devolverá una nube de puntos de posición constante. En el momento en el que un barco aparece en su línea de visión la nube de puntos cambia en las zonas donde el láser se encuentra con el barco y el sistema detectará que esos puntos están más cerca que el resto del entorno.

Éste hecho supone que en una nube de puntos constantes aparecen una serie de puntos que indican unas distancias más cortas en una zona concreta del espectro, algo que el software NuPIC detectaría como una anomalía. Al detectarlo como una anomalía significa que ha encontrado un patrón nuevo que no había visto antes y puede aprender de él. Tanto la forma del barco como su tamaño se verían reflejados en la nube de puntos con lo que barcos con formas y tamaños diferentes tendrían un patrón distinto dentro de dicha nube.

NuPIC podría identificar estos patrones, almacenarlos y aprender de ellos con el objetivo de no sólo detectar la presencia de barcos sino que podría crearse un sistema que permitiera identificar qué tipo de barco es el que está pasando por su forma, tamaño, velocidad, etc.

Bibliografía

Manuel Gómez de la Calle. *Modelo de previsión de demanda de electricidad de largo plazo*. Proyecto fin de carrera, Escuela Técnica Superior de Ingeniería (ICAI), Universidad Pontificia Comillas. Madrid, Septiembre de 2010.

Luis Hernandez, Carlos Baladrón, Javier M. Aguiar, Belén Carro, Antonio J. Sanchez-Esguevillas and Jaime Lloret. *Short-Term Load Forecasting for Microgrids Based on Artificial Neural Networks*. Published: 5 March 2013.

George Gross and Francisco D. Galiana. *Short-Term Load Forecasting*. Published: December 1987.

[1] http://es.wikipedia.org/wiki/Jeff_Hawkins

[2] http://numenta.com/

[3] http://numenta.org/nupic.html

[4] https://www.python.org/

[5] http://es.wikipedia.org/wiki/CSV

[6] http://numenta.org/cla-white-paper.html

[7] http://en.wikipedia.org/wiki/On_Intelligence

[8] https://github.com/

[9] http://www.ubuntu.com/

[10] https://github.com/numenta/nupic/wiki/Running-Swarms