Departamento de Informática

PROYECTO FIN DE CARRERA

MINERÍA DE DATOS EN PORTAL INMOBILIARIO

Autor: Marta Viana Colino

Tutor: Fernando Fernández Rebollo

Leganés, septiembre de 2012

Título: MINERÍA DE DATOS EN PORTAL INMOBILIARIO
Autor: Marta Viana Colino
Director: Fernando Fernández Rebollo

EL TRIBUNAL

Presidente: Daniel Borrajo

Vocal: José María Valls

Secretario: Emilio Parrado-Hernández

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 26 de Noviembre de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Me gustaría dar las gracias a todas las personas que han contribuido directa o indirectamente en este trabajo.

Principalmente, quiero agradecer a mi tutor Fernando, por la idea y por toda la ayuda brindada en estos meses.

Y agradecer a de todos mis compañeros de carrera, familiares y amigos, por sus ánimos, sin los que no hubiera llegado hasta aquí.

Resumen

A día de hoy se realizan en internet millones de búsquedas diarias en las que un usuario busca acerca de una información concreta, como pueden ser objetos en una tienda online, noticias en una web de periódicos o páginas web en un buscador. Estas búsquedas en la mayoría de los casos aplican un simple filtro sobre el total de su información, devolviendo todos los valores que cumplan dicho filtro, sin ninguna organización previa.

Con este proyecto se quiere mejorar el acceso a los resultados de una búsqueda, concretamente sobre una base de datos de viviendas en venta o alquiler sacada del portal Idealista, facilitando al usuario la selección de los objetos preferibles dentro del grupo de los resultados. Para ello, se utilizan los datos obtenidos de Idealista, para entrenar y representar un mapa auto-organizativo, una técnica de aprendizaje automático no supervisado, que permiten agrupar las viviendas dependiendo de las características que tengan las mismas. Cuánto más parecidas sean dos viviendas, más cerca se encontrarán dentro del mapa y será más fáciles de localizar las elegidas, eliminando las zonas de mapa cuyas viviendas no sean del agrado del usuario. Para calcular la similitud de viviendas, se utilizarán atributos clásicos como el precio, la localidad o en número de habitaciones, entre otras.

Se tomará una base de datos inicial que incluya todas las viviendas de Madrid extraídas mediante el API de Idealista; se les aplicará un filtro seleccionado por el usuario y se procederá a entrenar el mapa tomándolos como datos de entrenamiento. El entrenamiento consiste en hacer repetidamente el cálculo de la distancia euclídea entre las viviendas y las celdas del mapa (que estarán representadas por las mismas características que las viviendas) para encontrar la celda con menor distancia, a la que pertenecerá la vivienda.

Una vez se sepan las celdas a las que pertenece cada vivienda, se representará en un mapa bidimensional, en el que el usuario podrá navegar decidiendo qué nodos son más interesantes para él. De esta forma, el total de resultados de la búsqueda se reducirá facilitando al usuario la selección, sin tener que haber recorrido todos los resultados.

Palabras clave: búsqueda en internet, portal inmobiliario, mapas auto-organizativos, aprendizaje automático

Abstract

Nowadays, millions of searches are made over the Internet. The users are looking for specific information, like products in an online shop, news in a newspaper's web or websites in a web search engine. Most of these searches apply a simple filter into the whole information, returning all the values that fulfil that, without any previous organization.

The aim of this project is to improve the access to the results of a search, concretely over the database of housing, on rent or on sell, extracted from Idealista portal, making easier for the user to select the favourite housing over the whole result of the search. It is used the database of Idealista to train and represent a self-organizative map, a non supervised machine with learning technique, that allows to cluster the housing depending on their characteristics. As more similar two housing are, more closer they will be represented on the map, so the selected housing will be easier to found, by removing parts of the maps which housing don't agree the users likes. To calculate the similarity between housing, classic attributes will be used, like locality, number of rooms or prize and more.

It will use an initial database that includes all Madrid housing that will be extracted thanks to Idealista API; a user's filter will be applied over them, and then the map will be trained using them. The training consists on repeating the calculation of the Euclidean distance between the housing and the map nodes (that will be represented by the same characteristics as the housing) to find the node with less distance, that will be considered the winner, where the housing will belong.

Once all the housing have their belonging node, they will be represented on a bidimensional map, in which the user could navigate deciding which nodes are better for his interests. In this way, the total number of results will be decreased making the selection easier, without having to cover all the results.

Keywords: web search, estate website, self-organizing maps, automatic learning

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

En la actualidad se encuentran muchos buscadores en internet que consisten únicamente en restricciones que se aplican a una base de datos. Esto da como resultado una serie de registros que, aunque cumplen todas las cualidades solicitadas, no aportan más información.

¿Y si el resultado a la búsqueda es un largo número de registros? ¿Se necesitará recorrer todos para encontrar los que realmente se busca?

La idea de este proyecto es responder a estas preguntas utilizando la representación de esos resultados en mapas bidimensionales, de forma que se organicen en grupos, basándose en las similitudes que existan entre sus características.

Idealista es un portal Inmobiliario muy conocido en la red, que ha puesto a disposición de los desarrolladores que deseen utilizarla, un API para acceder a la información de las viviendas dadas de alta en su sistema. Un API es un conjunto de funciones y procedimientos que ofrece un componente software para comunicarse con otro; tratándose en este caso de una dirección web, modificable por ciertos parámetros, a la que el software podrá llamar para que le devuelva información de las viviendas. Esta información que se extraiga se usará como base de datos del sistema desarrollado en este proyecto.

El mismo portal dispone de dos tipos de búsqueda: uno que filtra por compra/alquiler, tipo de vivienda y provincia, y otro que busca por texto libre. Ambas búsquedas de filtrado básico que reportarán una gran cantidad de resultados.

La pantalla de búsqueda básica se muestra en la figura 1, dónde se pueden observar tres listados correspondientes al tipo de acción (alquiler/compra), el tipo de vivienda y la provincia. El usuario podrá seleccionar únicamente una opción de cada listado y al pulsar el botón "buscar" idealista devolverá un resumen de las viviendas que cumplen los criterios.

Figura 1. Búsqueda básica del portal inmobiliario Idealista

Sobre la búsqueda por texto libre (figura 2), presenta unas opciones de búsqueda más amplia; aunque como en el caso anterior tienes que estar seguro de la acción y el tipo de vivienda, ya que únicamente se puede seleccionar uno. En el campo textual se puede escribir cualquier condición que al usuario le parezca relevante: desde algún dato de localización (provincia, código postal, calle, etc.) hasta el número de habitaciones, la planta o si se trata de un inmueble luminoso.

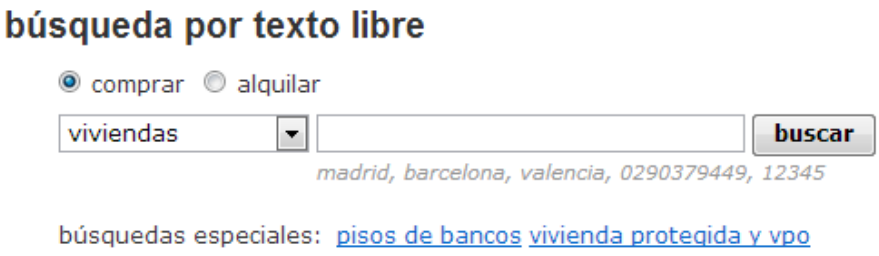


Figura 2. Búsqueda por texto libre del portal inmobiliario Idealista

Para que el usuario no tenga que lidiar con el número total de viviendas que daría como resultado una de estas búsquedas, en este proyecto se utilizarán los datos para entrenar un mapa auto-organizativo.

Los mapas auto-organizativos (también llamados mapas de Kohonen) son un tipo de red neuronal de aprendizaje no supervisado, capaz de formar mapas de características, asignando las distintas entradas a las neuronas que lo conforman, basándose en la similitud entre sus características.

Dentro del proyecto, estos mapas representan las viviendas en las rejillas de un mapa bidimensional, basándose en las similitudes que hay entre sus características, a través de la distancia euclídea entre ellas. De esta forma, las viviendas con características semejantes, se encontrarán en la misma celda del mapa, lo que ayudará al usuario a centrarse en las celdas en las que ya haya encontrado alguna vivienda que le interese, y así reducir las viviendas a estudiar.

1.2 Objetivos

El objetivo fundamental de este proyecto es el de representar viviendas de un portal inmobiliario utilizando mapas auto-organizativos. En base a ese objetivo principal, se proponen los siguientes objetivos parciales:

- Obtener los datos de las viviendas de la base de datos de Idealista haciendo uso de su API. Se almacenarán en un formato fácil de utilizar por esta herramienta.
- Tratar la información de las distintas características que representan una vivienda, de tal forma que todas sean numéricas y normalizadas para su fácil tratamiento a la hora de crear los mapas. Se hará uso del código fuente de Weka (una herramienta de análisis de datos), disponible en su web oficial, para integrarlo en nuestra aplicación y así poder utilizar varias de sus funcionalidades para este fin.
- Estudiar el funcionamiento de la herramienta SOM Tool Box, la cual se utilizará para crear y visualizar los mapas. Esta herramienta, permitirá hacer varios tipos de entrenamiento y sus distintas visualizaciones facilitarán al usuario el análisis de los datos.
- Generar los mapas auto-organizativos gracias a la herramienta SOMToolBox, preparando los ficheros necesarios con el formato correcto, tanto de datos como de parámetros.
- Crear una aplicación que integre las funcionalidades de Weka y SOMToolBox, y que sirva como prototipo de un futuro portal basado en esta tecnología
- Evaluar los resultados, para estudiar la viabilidad del uso de esta herramienta por los usuarios habituales de los portales inmobiliarios.

1.3 Fases del desarrollo

Las distintas fases del desarrollo se separan en torno a las herramientas utilizadas para cada una:

- API Idealista
    - Estudio de la forma de obtener la información sobre las viviendas, del que obtendremos las características que podemos extraer y los parámetros por los que podemos filtrar.
    - Selección de los datos a obtener, que serán utilizados para crear los mapas. De base se utilizarán las viviendas de Madrid
    - Extracción de los datos de las viviendas con llamadas al API de Idealista con los parámetros establecidos.
    - Aplicación de los filtros solicitados por el usuario. El usuario puede aplicar filtro sobre todas las características existentes en las viviendas

    - Creación de la interfaz para que el usuario pueda realizar la extracción desde Idealista
- Weka
    - Selección de los datos que se utilizarán para caracterizar una vivienda, contemplando cuáles aportan más información y pueden ser más relevantes
    - Transformación de los datos alfanuméricos a numéricos a través de una numeración secuencial de todos los valores.
    - Conexión de Weka con el programa java realizado integrándolo en nuestra aplicación
    - Normalización de las características para que existan grandes diferencias entre los valores.
- SOM Tool Box – Creación del SOM
    - Estudio del funcionamiento del algoritmo de creación del SOM. Realizando pruebas y creando diversos formatos de ficheros.
    - Modificaciones necesarias para que el entrenador utilice los datos seleccionados y les aplique los factores de importancia definidos por el usuario. Se modifica el cálculo de la distancia y los parámetros iniciales.
    - Aplicación de Growing SOM, para hacer el tamaño del mapa variable ya que el número de viviendas a representar es también variable.
- SOM Tool Box – Visualización
    - Estudio del funcionamiento del visualizador.
    - Aplicación de las modificaciones necesarias para mostrar los datos de las viviendas, útiles para que el usuario haga su selección.
    - Estudio de las visualizaciones aportadas por el programa, para facilitar el uso al usuario.
- Integración de Tecnologías
    - Diseñar interfaces que permitan al usuario realizar todas las funcionalidades necesarias para nuestro objetivo.
    - Aunar todas las tecnologías anteriores de forma que puedan interactuar con las interfaces y obtengan la funcionalidad objetivo.
- Validación y Pruebas
    - Comprobar que los resultados son visibles
    - Realizar pruebas con distintos parámetros de entrada para comprobar los distintos resultados.

1.4 Estructura de la memoria

La estructura que seguirá la memoria es la siguiente:

- **Estado actual**: En este apartado se definirá el estado actual de todas las tecnologías que se van a utilizar en el proyecto.
- **Trabajo Realizado**: Se desarrollarán las distintas fases de trabajo explicando con amplio detalle las acciones realizadas
- **Evaluación**: Se generan varios mapas de ejemplo, con distintos filtros y número de características a tener en cuenta; para probar el funcionamiento de la herramienta
- **Conclusiones**: Dar una idea general del resultado que se obtiene a través de lo desarrollado en este proyecto; así como dar una serie de mejoras futuras para el mismo.
- **Anexos**: Entre los que se encuentran la planificación, el presupuesto y el manual de usuario.

Capítulo 2

Estado Actual

En este capítulo se va a explicar la situación actual de las tecnologías y herramientas que se van a utilizar para desarrollar nuestra aplicación.

En una primera introducción se estudian los tipos de buscadores disponibles actualmente en la red, la mayoría de ellos basados en filtros básicos, haciendo especial hincapié en los sistemas de recomendación, ya utilizados en varias páginas de ventas de productos y noticias, entre otros.

A continuación se explica el aprendizaje automático, por el que un sistema es capaz de devolver la salida a un sistema a partir de experiencias previas; y los mapas auto-organizativos que son la parte más importante de este proyecto. Estos mapas auto-organizativos son un tipo de aprendizaje automático capaz de organizar los datos de entrada entre las neuronas de un mapa bidimensional basándose en las similitudes entre sus características, dando lugar a una estructura en la que los registros similares se encuentran cerca.

Para finalizar el capítulo se explicarán las dos principales herramientas que vamos a utilizar: API de Idealista y SOM Tool Box. La primera es una funcionalidad aportada por el portal inmobiliario para acceder a la información de su base de datos, mientras que la segunda es una herramienta muy amplia para la generación y visualización de mapas auto-organizativos. Se centra la explicación de ambas herramientas en su forma de uso y en los resultados obtenidos a partir de ellas.

2.1 Buscadores

La mayoría de las páginas web que se encuentran hoy en día en internet poseen un buscador para facilitar al usuario acceder a la información que desean obtener de forma más fácil y rápida que si consultaran la web de forma manual a través de los distintos links.

Estos buscadores recorren la información total existente hasta encontrar los parámetros o palabras introducidas por el usuario; utilizándolos únicamente como filtros de información; lo que es muy útil cuando el usuario tiene muy claro lo que está buscando; pero si el usuario no está seguro y aplica filtros amplios, la cantidad de resultados puede hacer que le resulte muy costoso encontrar el registro deseado.

Es importante comentar que, aunque muchos de los buscadores son básicos como hemos explicado, se han hecho útiles avances para facilitar al usuario las búsquedas:

- Uso de sinónimos de las palabras introducidas
- Permite obtener respuestas a preguntas concretas, como el buscador Wolfram Alpha.
- Búsqueda de imágenes en lugar de palabras (Google goggles)
- División de los resultados dependiendo del tipo de dato que se está buscando. Así google permite seleccionar entre Páginas Web, que sería la búsqueda normal, Imágenes, Noticias, Compras, etc.
- Búsquedas geográficas en mapas. Permitiendo al usuario seleccionar lo que está buscando a través del mapa (google maps)
- Sistemas de recomendación: En los que basamos este proyecto y que comentaremos en el siguiente punto.

2.2 Sistemas de Recomendación

En algunas de las páginas web se ha evolucionado la búsqueda utilizando sistemas de recomendación. El objetivo de estos sistemas, es mostrar al usuario una serie de productos que se parezcan, en sus características, a los que el usuario está viendo o que cubran sus gustos previamente recogidos.

Web	Uso	Ventaja
amazon	Presenta una serie de productos parecidos a los que está viendo el usuario, los que ha visto en días anteriores, o los que han visto otros usuarios que hicieron la misma búsqueda.	35% de las ventas se dan utilizándolo
google news	Muestra noticias relacionadas a la que se está viendo.	38% de los clicks se hacen usándolo.
netflix	Presenta películas con características similares a las últimas vistas.	2/3 de las películas alquiladas son recomendadas por el sistema.
last.fm	Aprovecha los hábitos de música para recomendar otras canciones.	

Tabla 1. Ejemplos de servicios que usan Sistemas de Recomendación. Fuente: [SISTEMASRECOMENDACION]

Hay dos formas de recoger la información que se va a utilizar para crear un perfil de un usuario:

- De forma implícita:
  - Solicitar al usuario que pondere en base a una escala proporcionada, el tema en particular.
  - Solicitar al usuario que pondere un conjunto de temas de una lista de temas favoritos.
  - Presentar al usuario dos temas, y solicitarle que seleccione uno e ellos.
  - Solicitar al usuario que cree una lista de temas de su preferencia
- De forma explícita:
  - Guardar un registro de los temas que el usuario ha visto en una tienda online
  - Analizar el número de visitas que recibe un artículo
  - Guardar un registro de los artículos que el usuario ha seleccionado o visto.
  - Analizar las redes sociales de las que el usuario forma parte y de esta manera conocer sus gustos y preferencias.

También se pueden crear sistemas de recomendación sin información sobre un usuario en concreto, basando la recomendación en cálculos genéricos de la página web en concreto. De esta forma, varias páginas actuales presentan selecciones de “noticias más visitadas” o “productos más vendidos”.

2.3 Aprendizaje Automático

El aprendizaje automático es una rama de la Inteligencia Artificial que pretende desarrollar técnicas que permitan a los programas aprender. Gracias a ello, se construyen sistemas capaces de adquirir el conocimiento necesario para realizar ciertas tareas, usando la experiencia acumulada.

Existen muchos sistemas conocidos basados en esta tecnología:

- Sistemas de Recomendación: El sistema aprende a través de información obtenida del usuario, para proporcionarle en el futuro ideas sobre lo que pueda estar buscando.
- Diagnóstico Médico: El sistema almacena información sobre los síntomas de las enfermedades, para posteriormente ser capaz de descubrir la enfermedad de un paciente a través de esos síntomas.
- Robótica: Utiliza el aprendizaje automático para memorizar ciertos movimientos que deberá realizar como respuesta a acciones realizadas sobre los mismos robots o en su entorno.
- Reconocimiento por voz: Tiene como objetivo permitir la comunicación hablada entre seres humanos y aparatos electrónicos. Procesa la señal de voz emitida por un ser humano y reconoce la información contenida convirtiéndola en texto o actuando como si fueran órdenes.

Dentro del aprendizaje automático se definen distintas ramas, en función de los datos de los que se aprende. Dos de estas ramas son el aprendizaje supervisado y el no supervisado, que se describen a continuación.

2.3.1 Aprendizaje supervisado

Los datos necesarios para que el sistema mejore en su comportamiento y aprenda, son suministrados por el usuario.

Estos datos se pueden representar como vectores de información, en los que uno de los valores es el resultado (o clase) y el resto de valores forman una combinación de datos de entrada con los que se generaría ese resultado

Su objetivo es el de generar sistemas que sean capaces de generalizar esta información introducida, para que se puedan dar resultados a datos no introducidos previamente. Normalmente, se centra en la aproximación de funciones, y se divide en dos grupos: clasificación y regresión.

- Clasificación: La salida de la función es un valor nominal. Se combinan las distintas posibilidades de valores de todas las características que se tienen en cuenta y se añade un valor añadido que representará el resultado obtenido en esa combinación. En la siguiente figura vemos una tabla con dicha estructura de clasificación y su representación en un árbol de decisión; una forma fácil de visualizar y obtener el resultado siguiendo sus nodos.

outlook	temperature	humidity	windy	play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rainy	71	91	TRUE	no

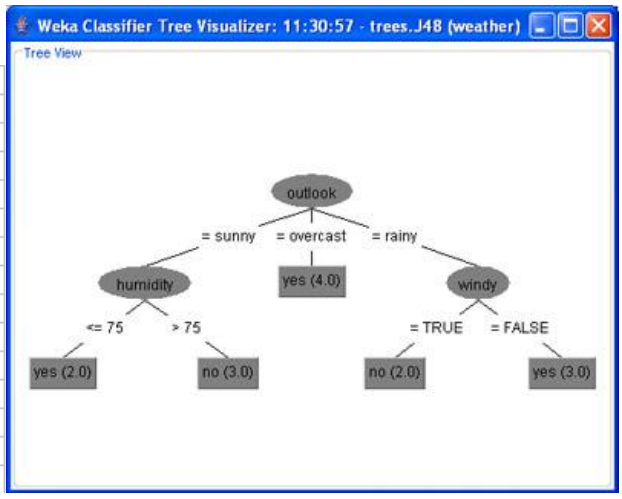


Figura 3. Ejemplo de árbol de clasificación obtenido en Weka para obtener la posibilidad de jugar al tenis dependiendo del tiempo atmosférico. Fuente: http://www.locualo.net/programacion/introduccion-mineria-datos-weka/00000018.aspx

- Regresión: La salida será una función sea capaz de calcular el resultado minimizando el error. En regresión lineal, se intenta aproximar la función original a partir de los datos con una función lineal. En la siguiente imagen se observa como la línea roja representa la función lineal que se ha aceptado como resultado para representar todos los puntos azules.

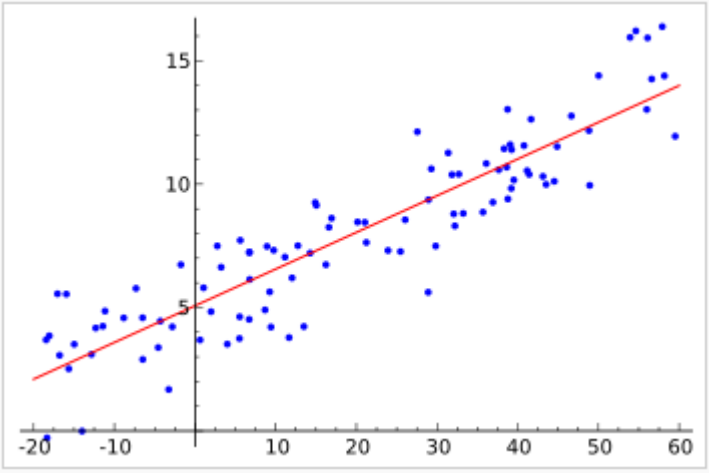


Figura 4. Ejemplo de regresión

2.3.2 Aprendizaje no supervisado

En este tipo de aprendizaje la información con la que trabaja el programa, se obtiene a partir de un conjunto de ejemplos que entran en el sistema, y sobre los que no se tiene conocimiento previo y, por tanto, el programa debe ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

Las ventajas más importantes de este tipo de aprendizaje es que no requiere de información inicial, que reajusta los parámetros automáticamente y que autoorganiza la información.

Existen distintos algoritmos que utilizan el aprendizaje no supervisado como base para organizar la información:

- *k* means: Se utiliza para encontrar los *k* puntos más densos en el conjunto de datos. Para ello:
    - Selecciona *k* centros aleatoriamente
    - Repetir hasta que los conjuntos no varíen
        - Asigna cada ejemplo al conjunto del centro más cercano
        - Calcula los puntos medios de los *k* conjuntos
- EM: Se usa para encontrar estimadores de máxima verosimilitud de parámetros en modelos que dependen de variables no observables u ocultas. El algoritmo alterna pasos donde se calcula la esperanza de verosimilitud mediante la inclusión de variables latentes como si fueran observables y pasos de maximización donde se computan estimadores de máxima verosimilitud de los parámetros mediante la maximización de la verosimilitud esperada en los pasos de cálculo de esperanza.
- COBWEB: Crea un clúster jerárquico con un árbol de clasificación mediante aprendizaje incremental. Al principio, el árbol consiste en un único nodo raíz. Las instancias se van añadiendo una a una y el árbol se va actualizando en cada paso. La actualización consiste en encontrar el mejor sitio donde incluir la nueva instancia, operación que puede necesitar la reestructuración de todo el árbol (incluyendo la generación de un nuevo nodo anfitrión para la instancia y/o la fusión/partición de nodos existentes) o simplemente la inclusión de la instancia en un nodo ya existente. La colocación de un nodo se obtiene mediante la utilidad de categoría que mide la calidad general de una partición de instancias de un segmento.
- SOM: Se describe en el siguiente punto.

2.4 Self Organizing Maps (SOM)

SOM son las siglas para representar los mapas auto-organizativos (o *self-organizing maps*). También se llaman mapas de Kohonen ya que su algoritmo fue desarrollado por Teuvo Kohonen, académico, profesor e investigador finés con varias contribuciones en el campo de las redes de neuronas artificiales. Toda la teoría sobre mapas auto-organizativos se puede encontrar en su libro *Self-Organizing Maps* [TKOHONEN].

Son un tipo de red neuronal, no supervisada, cuyo objetivo es representar los datos en una rejilla, normalmente bidimensional, basándose en las similitudes que hay entre ellos.

Además de los parecidos en cuanto a características de los registros dentro de una misma neurona, también se encuentran relaciones con las neuronas de alrededor, introduciendo el concepto de vecindario. Este vecindario puede tener formas diversas, aunque los más utilizados son el rectangular y el hexagonal.

El entrenamiento que realiza consiste en encontrar la neurona de la rejilla que más se asemeja a dicho dato, haciendo uso de la distancia Euclídea. Esta distancia es la distancia métrica más común creada como una generalización del Teorema de Pitágoras. Su fórmula es:

$$d_E(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

También se puede utilizar la distancia Euclídea Ponderada, en la que, además de la diferencia entre las características a comparar, se tiene en cuenta un peso que se le ha designado por su nivel de importancia.

Las neuronas se representan por valores inicialmente aleatorios para cada característica que se van modificando según los datos que se incluyen en ellas.

La arquitectura de los mapas de Kohonen está compuesta por dos capas de neuronas. La capa de entrada (formada por tantas neuronas como variables de entrada) se encarga de recibir y transmitir a la capa de salida la información introducida. La capa de salida es la encargada de procesar la información y formar el mapa de rasgos.

Cada neurona de entrada está conectada con una de las neuronas de salida mediante un peso. De esta forma, las neuronas de salida tienen asociado un vector de pesos llamado vector de referencia, debido a que constituye el vector prototipo de la clase representada por la neurona de salida.

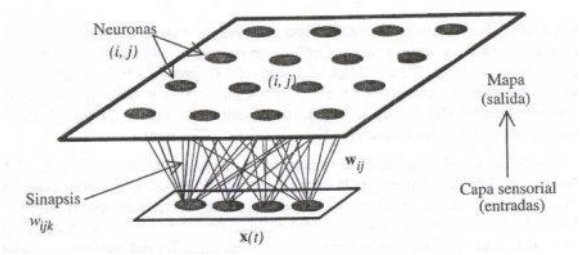


Figura 5. Arquitectura de un SOM. Fuente: http://extraccionrecuperacionnosupervisada.50webs.com/mapas-autoorganizativos.html

El algoritmo básico que siguen los mapas de Kohonen es:

1. Se inicializan los pesos de las distintas neuronas del mapa. Tendrá que tener el mismo número de características que tiene cada registro de datos, y cada uno de ellos tomará un valor aleatorio.
2. Estos pasos se repiten hasta que se haya completado el número de pasos de entrenamiento definidos a priori:
    a. Se busca la neurona del mapa más cercana al dato que estamos estudiando (normalmente a través de la distancia Euclídea)
    b. Se modifican los pesos de la neurona activa y de sus vecinas acercándolos hacia el vector de características del dato estudiado.

2.5 SOM ToolBox

SOM ToolBox es una herramienta de código libre realizada por el Instituto de tecnología Software y Sistemas Interactivos de la Universidad de Vienna, que permite el entrenamiento y posterior visualización de mapas auto-organizativos.

Al ejecutar esta herramienta presenta la siguiente interfaz:

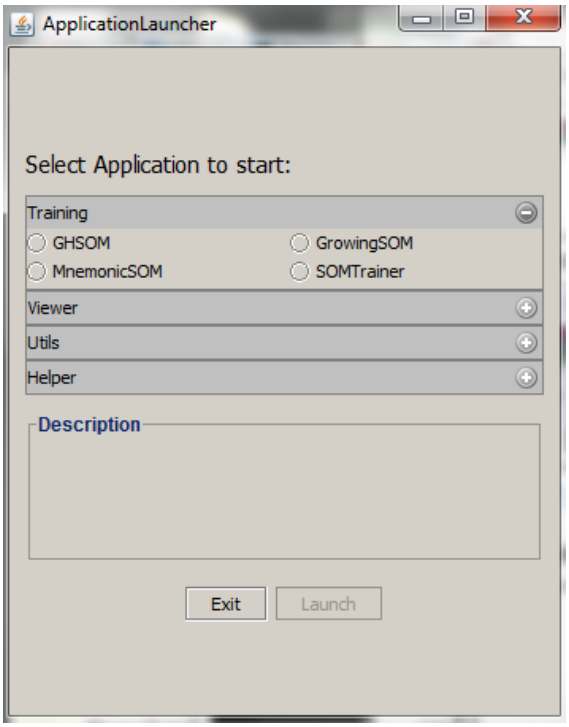


Figura 6. Pantalla de inicio de ejecución de SOM ToolBox

En ella se puede elegir el tipo de entrenamiento que se quiere aplicar para construir el mapa, o si se quiere visualizar un mapa ya entrenado.

Los cuatro algoritmos que presenta la herramienta para entrenar el mapa son:

- *GHSOM (Growing Hierarchical Self-Organizing Map)*
- *MnemonicSOM*
- *GrowingSOM*
- *SOMTrainer*. Cuyo acceso permitirá, internamente, seleccionar uno de los 3 métodos anteriores para hacer el entrenamiento.

Dependiendo del método elegido, el formato de los datos a introducir será distinto.

Para el algoritmo de entrenamiento utilizado en este proyecto (*Growing SOM*) el

formato del fichero a introducir es el de datos de entrada que sigue la siguiente estructura:

```
$TYPE <descripción de los datos del fichero>
$XDIM <Número de vectores que se encuentran en el fichero>
$YDIM <Normalmente 1. El número de vector se calcula como XDIMxYDIM,
pero este valor se deja en 1, y se pone el número completo en XDIM>
$VEC_DIM <Número de características de cada vector>
<caract1><caract2>……<caractVEC_DIM>
.
.
.
<caract1><caract2>……<caractVEC_DIM>
```

Tabla 2. Formato de fichero de introducción de datos en SOMToolBox

Dicha estructura está formada por cuatro parámetros obligatorios seguidos de los datos. Los cuatro parámetros son:

- Descripción del fichero: Nombre que se le asigna al tipo de datos
- Número de vectores o registros: Identificará la cantidad de registros que contienen el documento. Está formado por dos parámetros, X e Y, que multiplicados darán el número total. Normalmente el valor de Y se representa por 1 y el de X por el número total.
- Dimensión del vector: Número de características que tiene cada registro.

Tras estos parámetros, se listarán todos los datos, separando cada registro en una línea, y a su vez, cada característica por un espacio.

Una vez generado este fichero, dentro del programa se tienen que rellenar los siguientes campos de la interfaz, para ejecutar el entrenamiento:

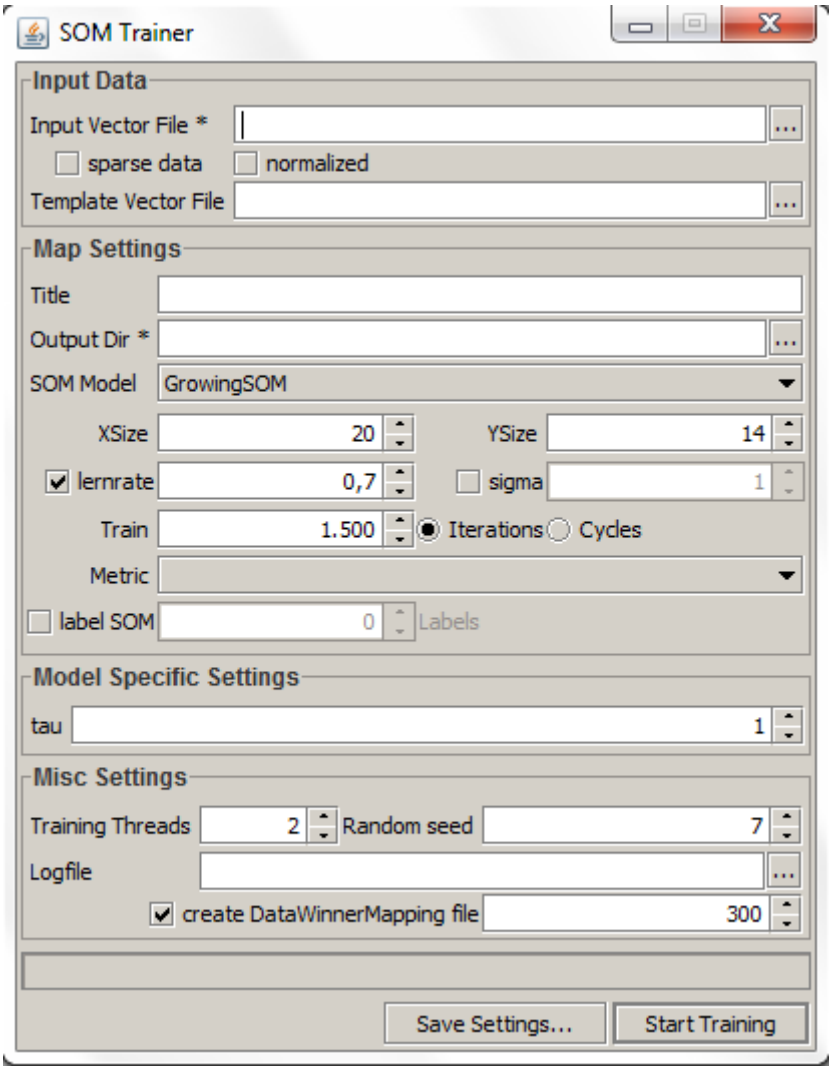


Figura 7. Pantalla de ejecución del entrenamiento

- Input Vector File: Ruta del documento que hemos generado
- Title: Título del mapa
- Output Dir: Ruta donde se van a guardar los archivos generados
- SOM Model: Modelo de entrenamiento que se va a utilizar
- XSize y YSize: dimensiones del mapa. En este caso será un valor inicial que irá aumentando según el error del entrenamiento.
- Learnrate: Tasa de aprendizaje. Es el factor que indica en qué grado deben modificarse los valores de la neurona en las actualizaciones que los acercan a los registros.
- Sigma: Indica el radio del vecindario del nodo ganador.
- Train: Número de iteraciones y ciclos que se realizan antes de finalizar el entrenamiento.
- tau: La fracción de cuantificación del error del mapa que determina la calidad del mismo. Es un argumento que se utiliza para indicar en qué medida va a crecer el mapa. Cuando su valor es 1, indica que las dimensiones serán fijas.
- Training Threads: Número de hilos que se quieren utilizar en el entrenamiento.
- Random seed: semilla que se utiliza para obtener los valores iniciales de los nodos del mapa.

El algoritmo de ejecución de dicho entrenamiento (realizado en la clase GrowingLayer.java, método trainNormal()) sigue los siguientes pasos:

1. Le da un valor para cada característica, a cada celda del mapa; de forma aleatoria.
2. Realiza el número de iteraciones elegidas en los parámetros. En cada iteración:
    a. Coge uno de los registros al azar
    b. Busca su nodo más cercano, utilizando la distancia euclídea. La fórmula que utiliza para cada nodo del mapa es la siguiente:

    $$\sqrt{\sum_{i=0}^{i=n^o\,características}(Característica_{i1} - Característica_{i2})^2}$$

    donde 1 es el nodo del mapa y 2 es el registro que estamos mirando.
    Según va recorriendo todos los nodos, va guardando el que tiene una distancia menor, al que considerará el ganador.
    c. Actualiza los valores de las características de los nodos; acercándolos al registro colocado dependiendo de los valores de *learnrate* y *sigma*.
    d. Actualiza los valores de *learnrate* y *sigma*. Los reduce para que las modificaciones en los nodos sean cada vez más pequeñas, y no provoquen alejamientos excesivos de anteriores registros estudiados.
3. Una vez hechas todas las iteraciones, comprueba si *tau* es distinto de cero. En caso contrario, que es lo que se va a utilizar en el proyecto, calcula la calidad de la distancia en ese momento (recorriendo todos los nodos, y sumando las distintas que hay respecto a los registros que en ellos residen), y si es mayor o igual a la calidad objetivo multiplicada por el valor del parámetro tau, vuelve al punto 2. Si es menor, pasa al punto 4.
4. Recalcula el nodo ganador para cada registro de datos y guarda toda la información en los ficheros necesarios para ejecutar el visualizador.

Estos ficheros generados son:

- Fichero de propiedades (*.prop) en el que se guardan las propiedades con las que se ha entrenado el mapa.
- *unitDescriptionFile* (*.unit) en el que se guarda la relación de registros que hay por cada nodo del mapa.
- *weightVectorFile* (*wgt) en el que se guardan los valores de las distintas características de los nodos del mapa.

Para visualizar el mapa, se accede de nuevo a la herramienta, pero a través de la opción *Viewer*, y seleccionando *SOMViewer*.

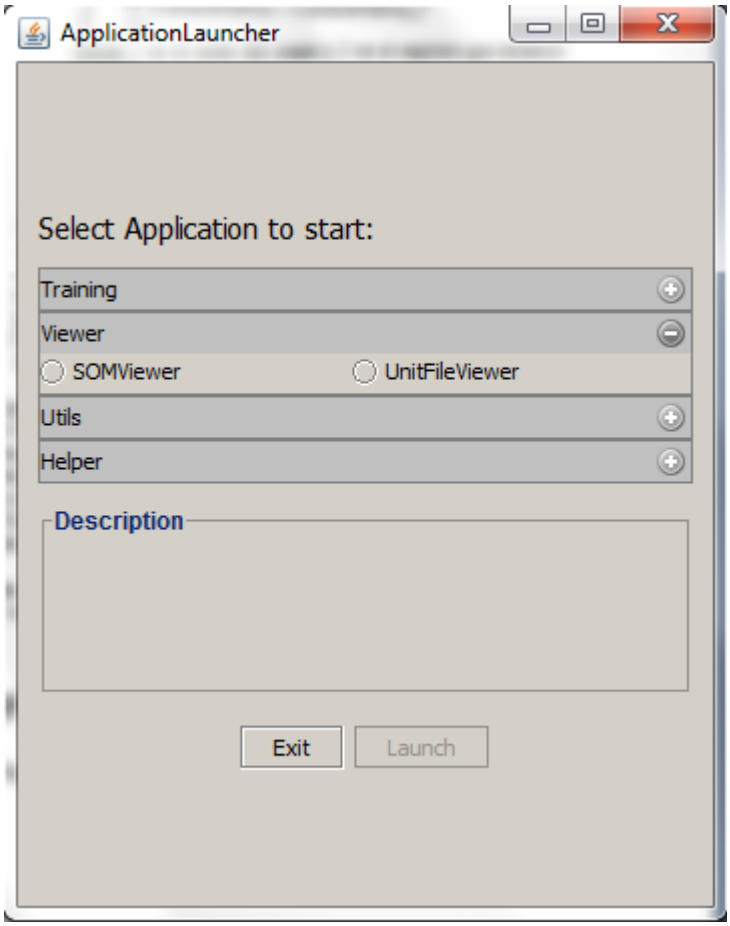


Figura 8. Pantalla de acceso al visualizador

Se abrirá la siguiente pantalla donde deberemos elegir los archivos donde tengamos nuestros datos. En este caso, se completan las rutas *unitDescriptionFile* y *weightVectorFile* con la ruta en la que se encuentren los ficheros generados por el entrenador.

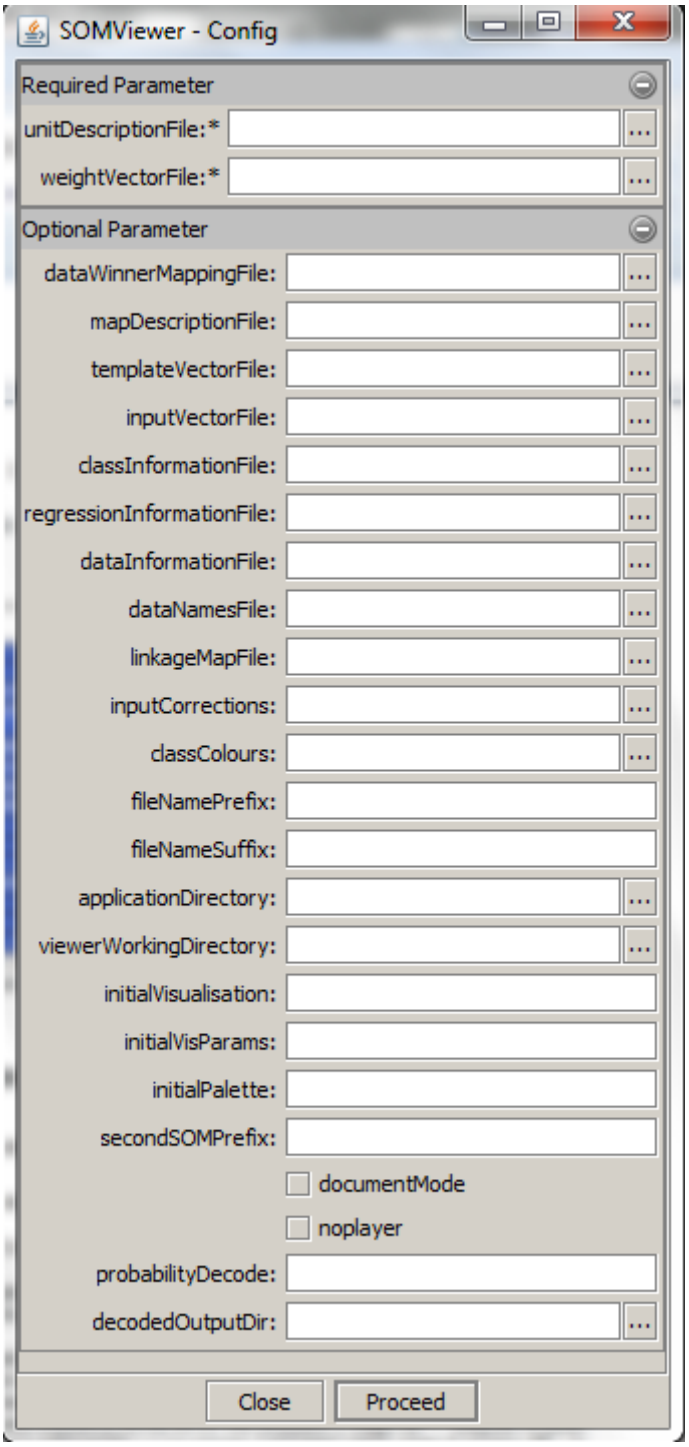


Figura 9. Pantalla de selección de datos para ejecutar el visualizador

Al ejecutar el visualizador, se verá una pantalla parecida a la siguiente:

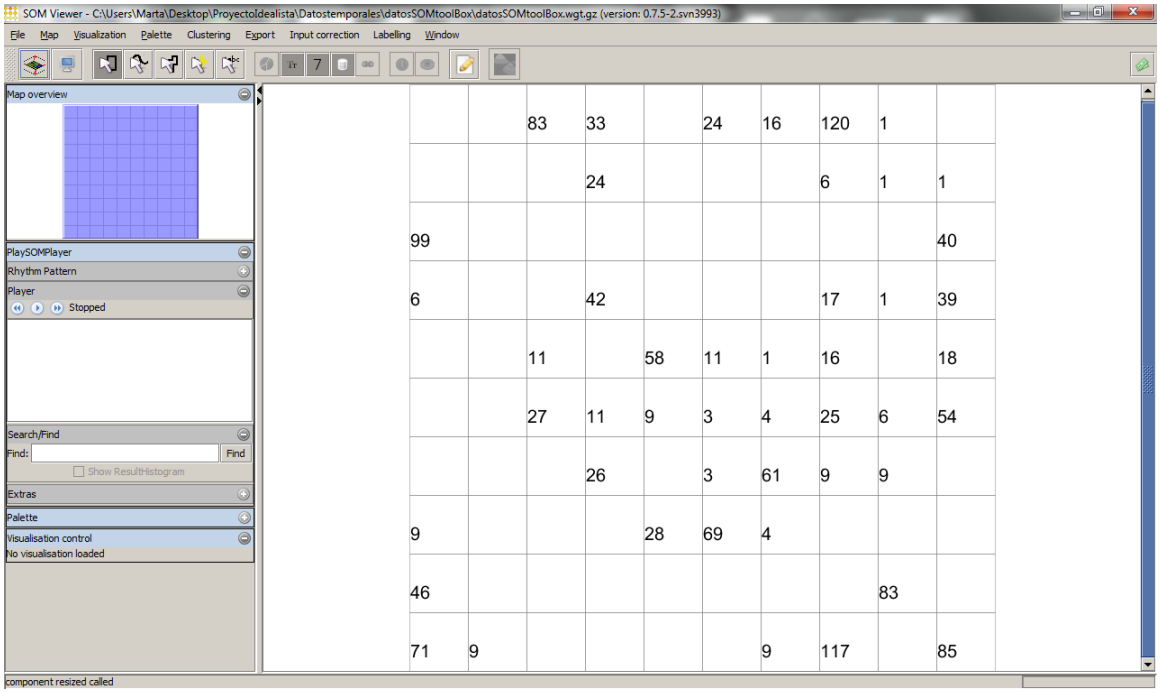


Figura 10. Pantalla del visualizador

Dentro de la pantalla anterior se pueden distinguir tres partes principales:

- Barra de Tareas Superior

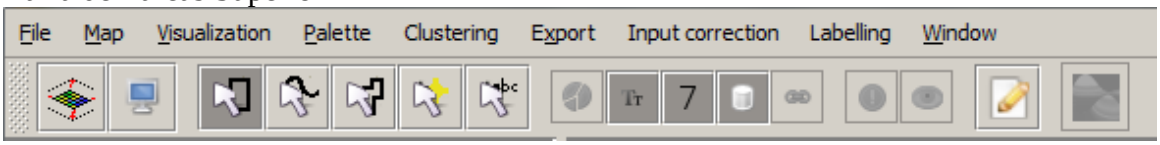


Figura 11. Barra de Tareas Superior de la Visualización

En ella cabe destacar el menú de Visualización donde se ven las distintas formas de representar los datos que nos permite el programa; el menú *Palette* donde se pueden elegir los colores de la representación; y los botones de la parte inferior que permiten moverse o realizar marcas (colores o textos) dentro del mapa, destacando el primer botón que permite devolver el mapa a su visualización general (para verlo entero) tras haber acercado la imagen hacia algún nodo en concreto.

- Barra lateral de resumen y opciones específicas
-

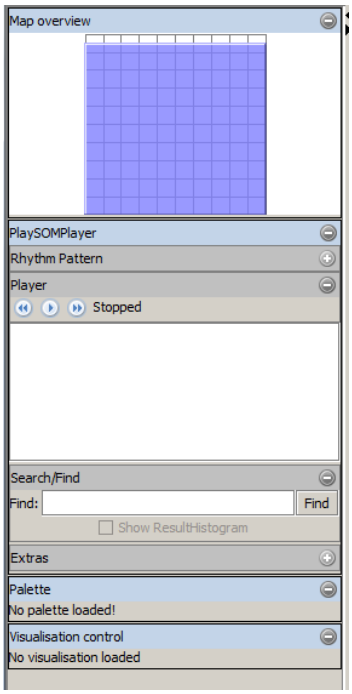


Figura 12. Barra Lateral de la Visualización

En ella se puede ver el mapa en pequeño y moverse por él, así como los datos de la Paleta y la visualización seleccionadas en el momento. Dependiendo de la visualización seleccionada, se mostrarán también (y se podrán modificar) los parámetros específicos para ese tipo de visualización.

- Visualización del mapa

		83	33		24	16	120	1	
			24				6	1	1
99									40
6			42				17	1	39
		11		58	11	1	16		18
		27	11	9	3	4	25	6	54
			26		3	61	9	9	
9				28	69	4			
46								83	
71	9					9	117		85

Figura 13. Visualización del mapa

Se muestran las distintas neuronas representadas por un número, que será el número de registros que pertenecen a esa neurona.
Cuando se acerca el mapa hacia uno de los nodos (existe la opción de hacerlo rápidamente con un doble click), en este se puede ver el nombre representativo de los registros que existen en él.

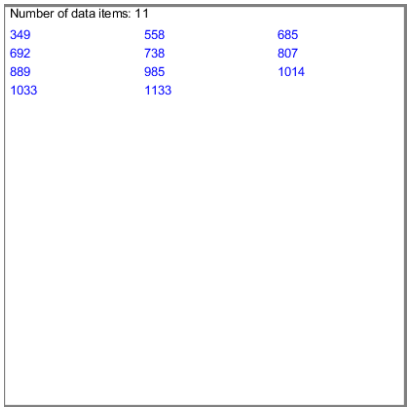


Figura 14. Visualización de un nodo

Y poniendo el ratón encima de él, el programa nos muestra más información:

- Coordenadas del nodo (En el ejemplo: 5/4 sería x=5 e y=4)
- Número de registros del nodo
- Los valores que tienen los distintos parámetros del nodo
- Nombres representativos de los registros.

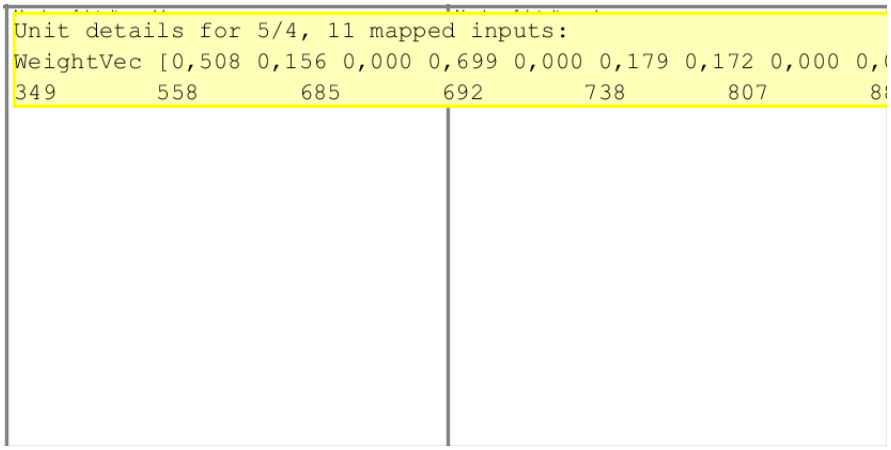


Figura 15. Datos mostrados en un nodo

Para completar este apartado es importante comentar los tipos de visualización que permite el programa. Vamos a destacar tres, que serán los que se usarán para el proyecto.

2.5.1 Visualización: Hit Histogram

Esta visualización tiene un cometido sencillo, que es el de mostrar los nodos con distintos tonos de rojo dependiendo de la cantidad de registros que tengan.

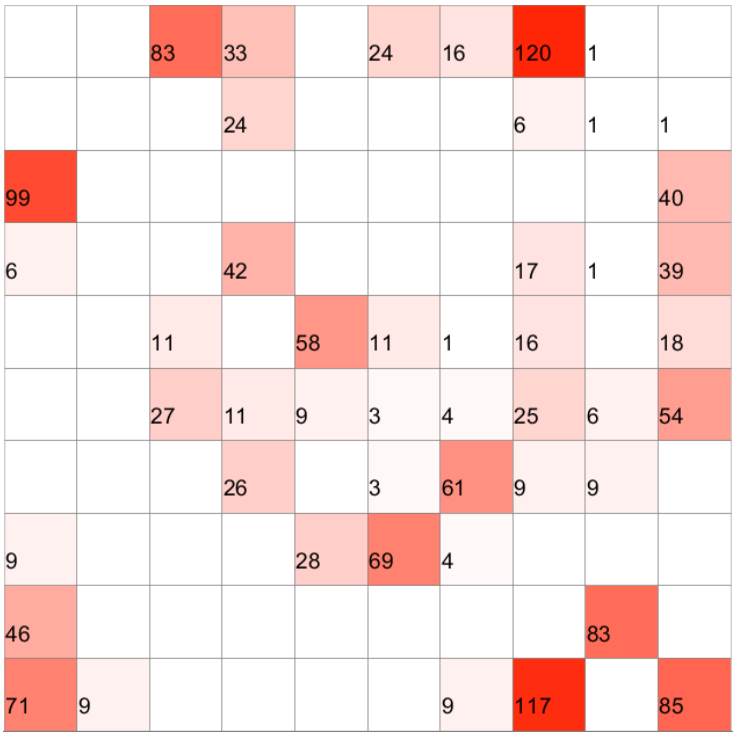


Figura 16. Visualización: Hit Histogram

En el ejemplo de la figura, podemos observar que hay 4 nodos más poblados, 3 en la parte superior y uno en la parte inferior (con tonos más oscuros). Esto nos da cierta información sobre dichos nodos, y es que las características que los identifican son bastante habituales en el grupo de las viviendas.

2.5.2 Visualización: Flow & Borderline

La unión de dos representaciones distintas, cuyo objetivo es representar el mapa mediante una estructura de clústers calculada a partir de la relación entre las características de los distintos nodos. La cantidad de nodos que tendrá en cuenta se modificará gracias al parámetro sigma (σ) que indicará el ancho de nodos en los que tiene que mirar.

Flow: Flechas que señalan el centro de clúster más apropiado para cada nodo del mapa. Se calcula tomando la distancia entre el vector que representa cada nodo y cada uno de los vectores del resto de nodos. A continuación, esas distancias se coordinarán y normalizarán para obtener unas coordinadas u y v que representarán la dirección y longitud del vector del nodo.

En la siguiente imagen podemos ver como se han calculado las distancias para uno de los nodos. Las distancias mayores están representadas en tonos más oscuros, y las claras en tonos más claros, por lo que en la figura (b) podemos ver que la flecha resultado se dirige hacia los puntos más claros del mapa.

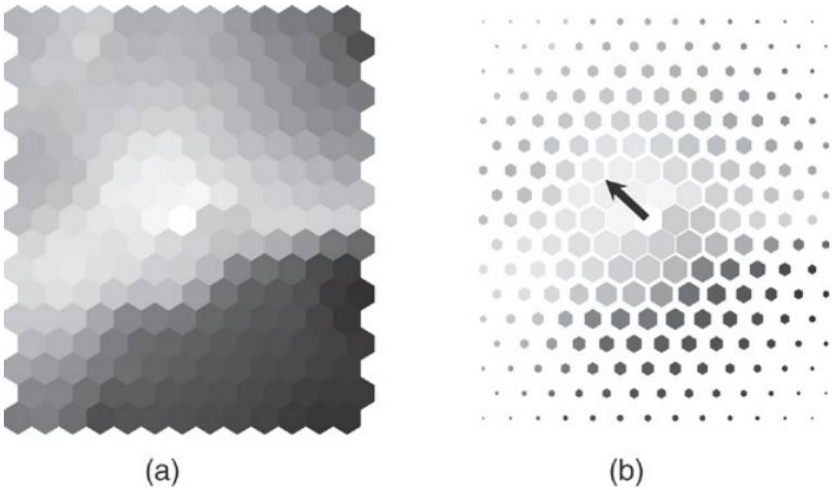


Figura 17. Distancias de un nodo

Borderline: Es una visualización derivada de la anterior que enfatiza en mostrar los límites del clúster

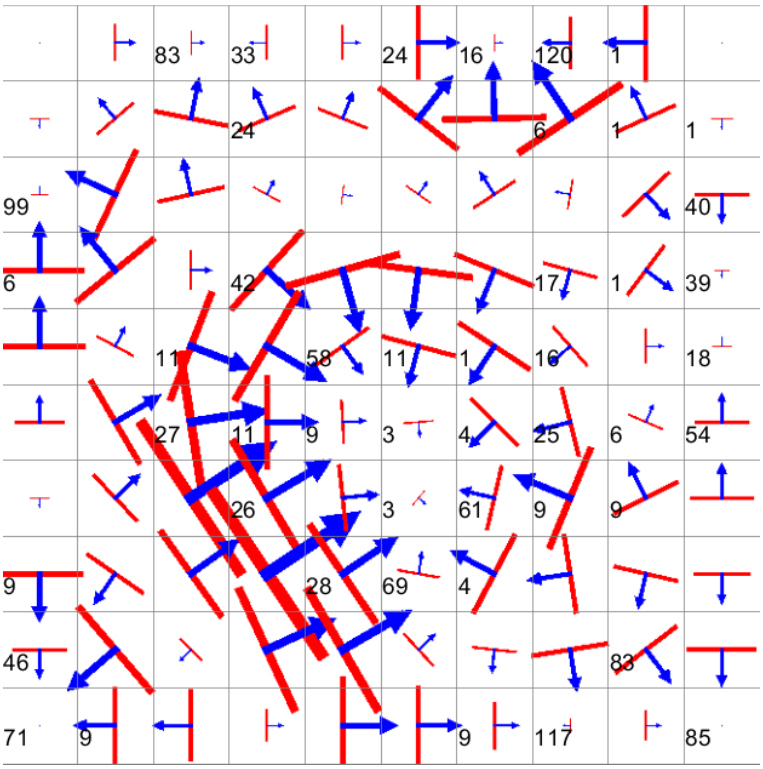


Figura 18. Visualización: Flow & Borderline

En el ejemplo, se ve como muchos de los nodos centrales se dirigen hacia sus nodos más cercanos formando un clúster. En las zonas laterales se forman más clústers pequeños, entre los que se puede destacar el de arriba a la derecha y el de la esquina

inferior izquierda. Si comprobáramos las características de los distintos nodos, se verían que las similitudes entre ellas son mayores en la dirección de las flechas.

2.5.3 Visualización: Smooth Data Histogram

Esta visualización también tiene el objetivo de representar los nodos en clústers, pero con la variante de que los clústers representarán áreas con las densidades de registros más altas.

La representación estará basada en una forma de mapa geográfico donde se ven las zonas más densas representadas como islas o montañas (blanco el punto más alto y marrones), y las zonas menos densas como océanos (azul).

Para entender mejor la combinación de colores del mapa, adjuntamos imagen de la paleta de colores que utiliza, que está ordenada de menor (azul) a mayor (blanco):

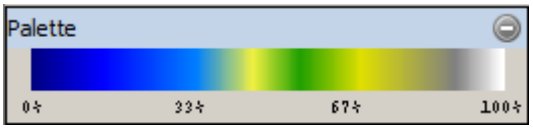


Figura 19. Visualización: Smooth Data Histogram. Paleta

Se utilizará un parámetro s (smoothing parameter) para calcular el grado de pertenencia de los nodos a cada clúster. Así, cuanto mayor sea este parámetro, menos clústers se encontrarán en el mapa. A continuación veremos dos imágenes en las que se representan los mismos datos con distinto valor de s.

En la primera imagen se ve como se han formado 8 clústers distintos que rodean las zonas más pobladas del mapa y dejan en azul las zonas menos pobladas. Destacar el clúster de abajo a la derecha que incluye dos de los nodos más poblados (con 117 y 83 registros) que se representa en blanco, color de las zonas "más altas".

En la segunda imagen, al tener un valor de s tan elevado, que implica tener en cuenta más vecinos a la hora de formar el clúster, únicamente se forma un clúster central, que englobaría la zona con más registros.

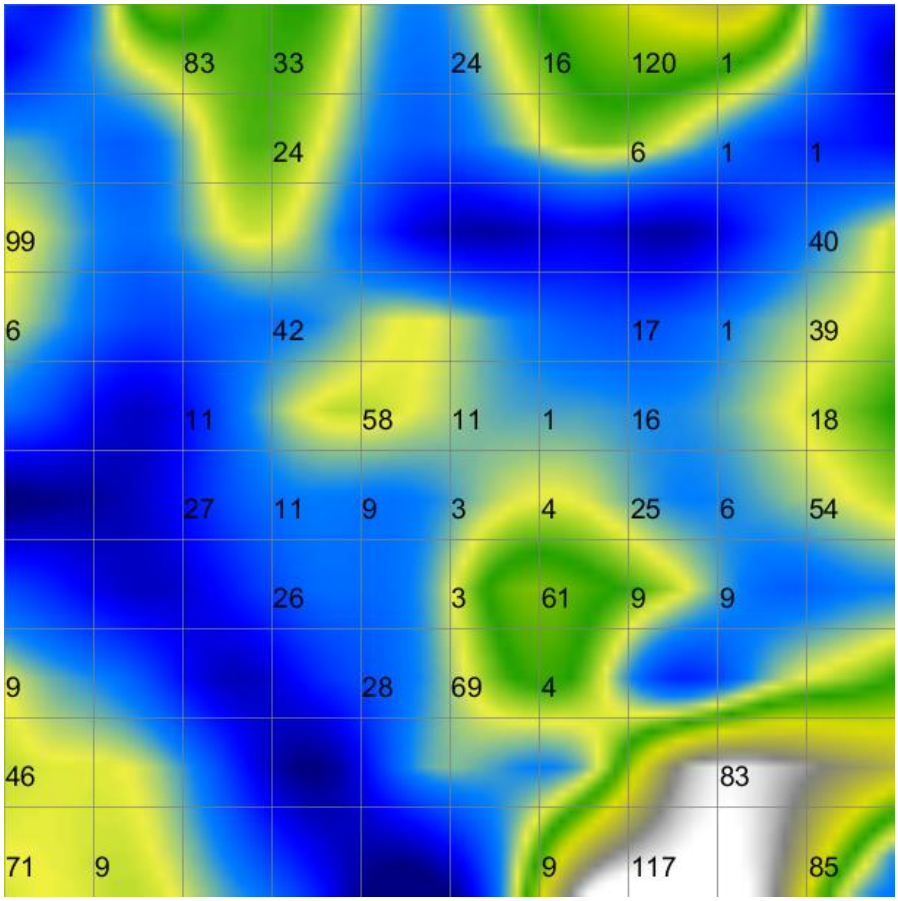


Figura 20. Visualización: Smooth Data Histogram (s=12)

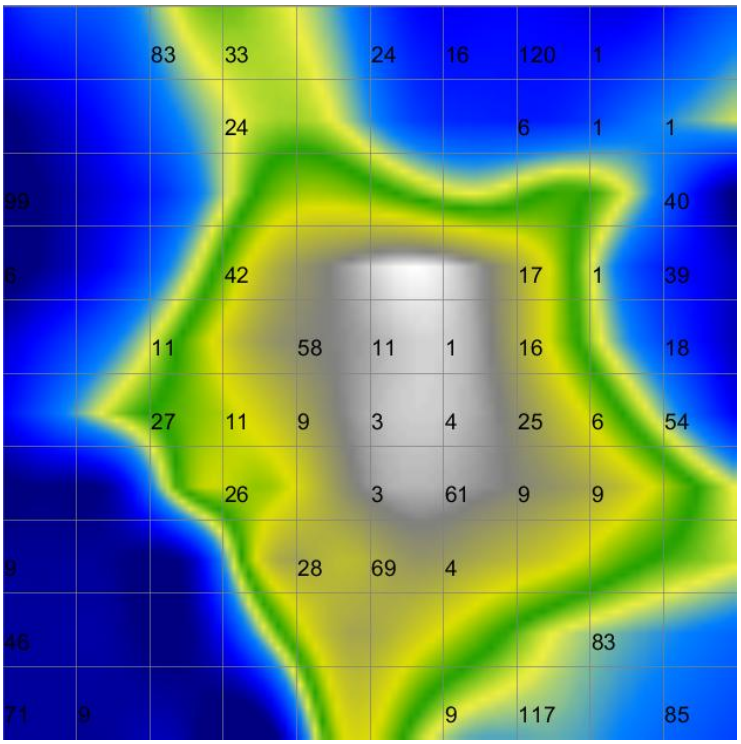


Figura 21. Visualización: Smooth Data Histogram (s=50)

2.6 Idealista API

Idealista proporciona a los desarrolladores que quieran disponer de la información sobre viviendas que poseen en su base de datos, un API con el que obtener dicha información.

A partir de una url (dirección web) creada con unas premisas definidas por Idealista, cualquier usuario puede obtener información de sus pisos en un formato parecido a csv (valores separados por comas), en el que las distintas características están separadas por comas, y que se puede tratar como un json en java. Json es un formato de intercambio de datos ligero, que ha dado lugar a una alternativa a XML más sencilla.

La url debe comenzar con el siguiente formato:

http://www.idealista.com/labs/propertyMap.htm?action=json

Añadiendo las características que se deseen para filtrar los datos a obtener, de entre las siguientes que permite utilizar Idealista:

Símbolo	Descripción
&k	Clave del Api. Es obligatoria y se obtiene de la misma web de idealista introduciendo tu dominio (o IP). Sin esta clave, y tiene que ser coincidente con la registrada por la IP que está solicitando los datos, no se permiten obtener datos. Se obtiene en la url http://www.idealista.com/labs/api.htm?action=main a través del siguiente formulario: 
&operation	El tipo de operación que se permite en la vivienda. Los posibles serán A (alquiler) y V (venta). Es obligatorio.
&center	Las medidas de latitud y longitud desde las que se empiezan a obtener los datos. Es obligatorio y se representará mediante los dos valores separados por comas.
&radio	Se puede representar como latitud y longitud, como el campo anterior, o como la distancia del radio. El área circular comprendida entre el centro y lo definido en este campo, será la zona geográfica de la que se está solicitando las viviendas. Este campo es obligatorio, en uno de sus dos formatos posibles.
&numPage	Es un valor numérico que representa la página que se quiere leer. Debido a que por cada consulta al API, sólo devuelve 20 viviendas, se necesita consultar las distintas páginas de una misma consulta. La llamada a la página 0 devuelve una serie de datos útiles para conocer el total de viviendas o de páginas, entre otras cosas. Ejemplo: *["pisos, áticos, estudios, dúplex, chalets, de todos los precios, de todos los tamaños, con dormitorios o más",{"actualPage":0 ,*

	"elementList":[], "itemsPerPage":20, "lowerRangePosition":-20, "numPaginations":0, "paginable":false, "total":621270, "totalPages":31064, "upperRangePosition":-20}, {"latitude":40.24001, "longitude":-4.23918}] Es un campo opcional, pero si no se rellena, el resultado obtenido será la página 1.
&minPrice **&maxPrice**	El mínimo y máximo precio en euros que se permite en la vivienda.
&minSize **&maxSize**	El mínimo y máximo tamaño que se permite en la vivienda.
&flat **&penthouse** **&studio** **&chalet** **&dúplex** **&garaje** **&premises** **&office** **&room**	Estas son características representadas por un valor true/false (verdadero/falso) que indica en cada caso si ese tipo de vivienda está permitido en la búsqueda. Si no se completan estos campos, se permitirá todo tipo de vivienda.
&since	Indica la fecha en la que se dio de alta la vivienda en la web. Los valores posibles son a (todas las fechas son válidas (valor por defecto)), m (si sólo se quieren obtener las viviendas del último mes) y w (si sólo se quieren obtener las viviendas de la última semana.
&pics	Si este valor es 1, únicamente se obtendrán viviendas con fotografías

Tabla 3. Campos utilizables para extraer Viviendas del API de Idealista

Una vez generada la URL con las características deseadas, el API de idealista devuelve un listado en forma de json. Esto es, un listado de las distintas características de la vivienda con el siguiente formato:

{“Caracteristica1”:”valorCaracteristica1,“Caracteristica2”:”valorCaracteristica2,…}

Característica	Descripción
address	Dirección de la vivienda.
photosURL	Dirección URL de la galería de fotos de la vivienda
distance	Distancia al centro.
agency	Campo que indica si la vivienda a sido publicada por una agencia (valor true) o no (valor false).
agentLogo	Dirección URL del logo de la agencia que ha publicado la vivienda, si procede.
bathrooms	Número de baños de que dispone la vivienda. Por un error en el API, el valor que se está devolviendo para este campo es siempre 0.
condition	Campo textual en el que el propietario puede indicar alguna condición relevante acerca de la vivienda.
country	País de la vivienda, representado por las dos primeras letras del

	mismo
district	Distrito en que se encuentra la vivienda
floor	Piso en que se encuentra la vivienda
hasVideo	Booleano que indica si el anuncio de la vivienda posee video o no
latitude	Latitud en la que se encuentra la vivienda
longitude	Longitud en la que se encuentra la vivienda
munipality	Municipio en que se encuentra la vivienda
neighborhood	Barrio en que se encuentra la vivienda
numPhotos	Número de fotos que posee el anuncio
operation	Indica si la vivienda está en venta (V) o en alquiler (A)
position	Posición de la vivienda, si es interior o exterior
price	Precio en euros de la vivienda
propertyCode	Código identificativo de la vivienda
propertyType	Tipo de vivienda
propertyTypeCode	El tipo de vivienda anterior representado por el código correspondiente
province	Provincia en que se encuentra la vivienda
region	Región en que se encuentra la vivienda
subregion	Subregión en que se encuentra la vivienda
rooms	Número de habitaciones que posee la vivienda
showAddress	Booleano que indica si se muestra la dirección rea de la vivienda o no
size	Metros cuadrados que tiene la vivienda
thumbnail	Dirección URL de la versión reducida de la fotografía de la vivienda
url	Dirección URL hacia los detalles de la vivienda
userCode	Código del propietario
videoType	Se indicará el tipo de video: p para videos desarrollados por el usuario, i para videos desarrollados por idealista y f si no hay videos

Tabla 4. Características que definen una vivienda

2.7 Weka

Weka (*Waikato Environment for Knowledge Analysis*) es una aplicación software desarrollada en Java cuya finalidad es proporcionar herramientas de aprendizaje automático y minería de datos.

Contiene una amplia colección de herramientas de visualización y algoritmos para análisis de datos a la que se puede acceder mediante interfaces de usuario. Posee 4 (*Simple CLI*, *Explorer*, *Experimenter* y *Knowledge Flow*) entre las que destaca Explorer que sería la más cómoda al usuario.

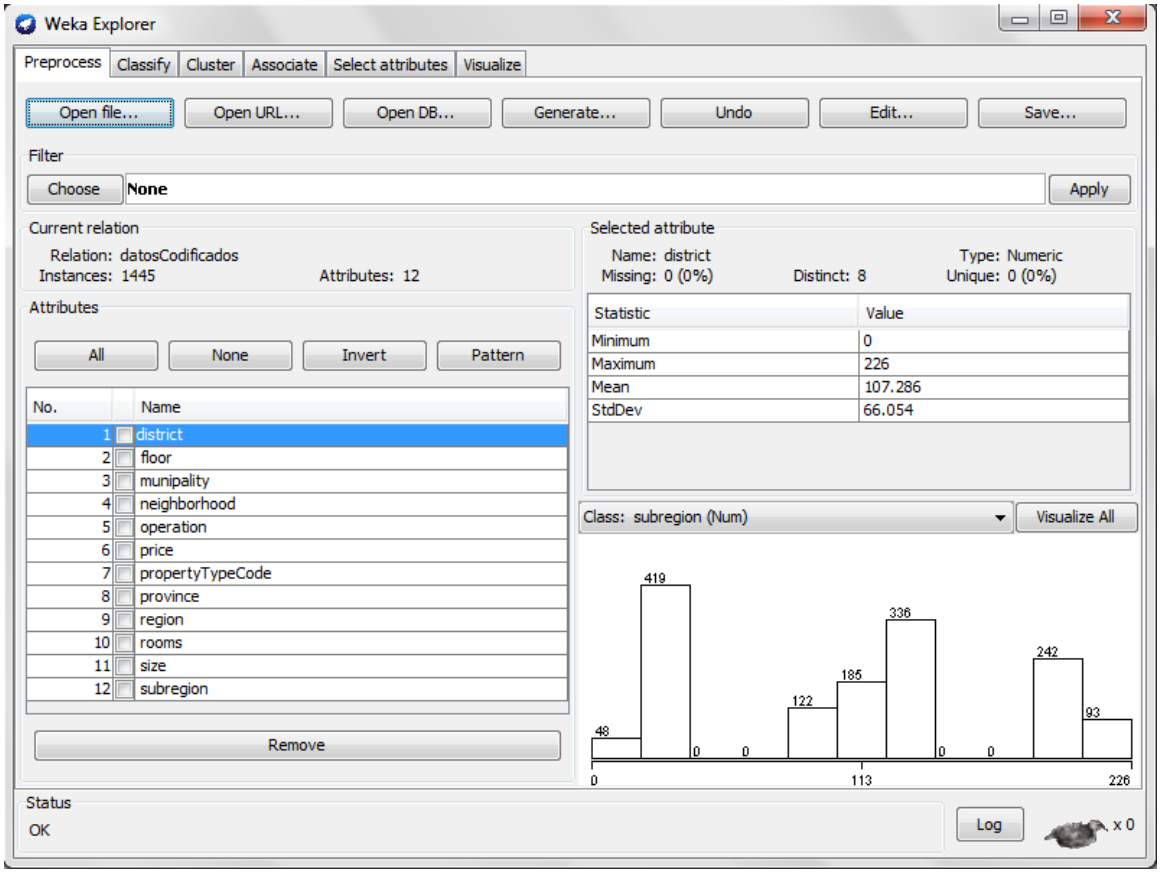


Figura 22. Interfaz Explorer de Weka

Esta interfaz permite el acceso a todas sus funcionalidades divididas en varios componentes en forma de pestañas:

- ***Preprocess***: Tratamiento inicial de los datos, incluyendo su importación y exportación y el aplicado de diversos filtros. En esta pantalla se puede conocer información básica sobre los características de los datos como su media, su desviación y sus valores máximos y mínimos.
- ***Classify***: Permite aplicar algoritmos de clasificación sobre los datos.
- ***Associate***: Proporciona acceso a las reglas de asociación de los datos, y su uso para descubrir hechos que ocurren en común dentro de ellos.

- ***Cluster***: Permite acceder a algoritmos de agrupamiento.
- ***Selected Attributes***: Proporciona algoritmos para identificar los atributos más predictivos en un conjunto de datos.
- ***Visualize***: Presenta una representación de los datos en forma de matriz en la que relaciona las características de dichos datos para realizar comparaciones.

2.8 Conclusiones

En los apartados anteriores se han estudiado distintas tecnologías que se utilizarán para realizar este proyecto con el fin de obtener un motor de búsquedas innovador y que pueda aportar algún tipo de mejora al usuario.

Por ello, se han estudiado los buscadores actuales, para evitar realizar algo ya existente.

Como datos de búsqueda se ha decidido representar viviendas, ya que el API de Idealista, como se ha visto en su apartado correspondiente, facilita tener una base de datos muy completa, lo que a su vez amplía el estudio (al tener estas una gran cantidad de características).

El resto de los apartados de este capítulo enfocan el proyecto hacia un sistema basado en la Inteligencia Artificial y, más concretamente, en el aprendizaje automático. El aprendizaje automático va a ayudar a convertir todos los datos de entrada (viviendas) en un mapa auto-organizativo estructurado según las similitudes existentes entre ellas. Se realizará entrenando el mapa hasta encontrar los valores de las características de cada neurona del mapa que hagan que todas las viviendas estén en armonía con sus vecinas (el error sea mínimo).

Para representar estos mapas, se hará uso de SOMToolBox ya que, como se ha visto, es una herramienta muy completa para hacer este tipo de representaciones. Por supuesto, se realizarán varias modificaciones para mejorarla y hacerla más agradable para el usuario y mostrarle la información que necesita, y que es capaz de procesar y entender.

Capítulo 3

Trabajo Realizado

En este capítulo se explican detalladamente las acciones realizadas para el desarrollo de la aplicación de representación de viviendas del portal inmobiliario Idealista en mapas auto-organizativos.

En el primer punto se muestra la estructura de clases utilizada, en la que se basa la herramienta.

En el segundo punto, se describen los desarrollos específicos para extraer los datos (utilizando un programa en java para hacer llamadas al API) y para representar los mapas resultantes: generación de ficheros con formatos específicos, filtrado de datos, entrenamiento del mapa y representación del mapa de viviendas en las neuronas de la rejilla mostrando sus características más relevantes.

3.1 Análisis y Arquitectura

Este proyecto está desarrollado en Java haciendo uso de la herramienta NetBeans que facilita el diseño de interfaces de usuario; que en este caso se van a utilizar para permitir al usuario seleccionar los filtros iniciales que aplica a las viviendas en las que podría estar interesado. Además se hará la integración de Weka y de SOMToolBox, dos herramientas que vamos a utilizar.

El esquema de clases del que partimos para realizar el programa se encuentra en la siguiente página

Figura 23. Diagrama de Clases

En el diagrama se pueden diferenciar las funcionalidades se la aplicación distribuidas de izquierda a derecha.

En la parte izquierda se encuentran las dos clases necesarias para hacer la extracción de datos del API de Idealista: una para la interfaz y otra para las funcionalidades en sí.

La extracción, guarda la información obtenida en formato "Vivienda" que es la clase que se encuentra a continuación y que a su vez es utilizada por la "busquedaInicial" para aplicar los filtros del usuario.

Esta clase "busquedaInicial" es una interfaz que permite al usuario seleccionar los filtros que quiere aplicar a las viviendas, mostrando los posibles valores gracias a una estructura jerárquica "Arbol". Su resultado pasará a la clase "filtrosWeka" que formateará y preparará los datos, haciendo uso de Weka, para que puedan ser utilizados por la herramienta SOMToolBox.

Esta herramienta será utilizada mediante llamadas a sus funciones necesarias que permitirán entrenar el mapa y visualizarlo en un mapa bidimensional en el que interactuará en usuario.

3.2 Diseño

El Diseño de la herramienta creada con el objetivo de ayudar a los usuarios a buscar una vivienda óptima para sus necesidades y deseos, se realiza a través de las siguientes 4 fases.

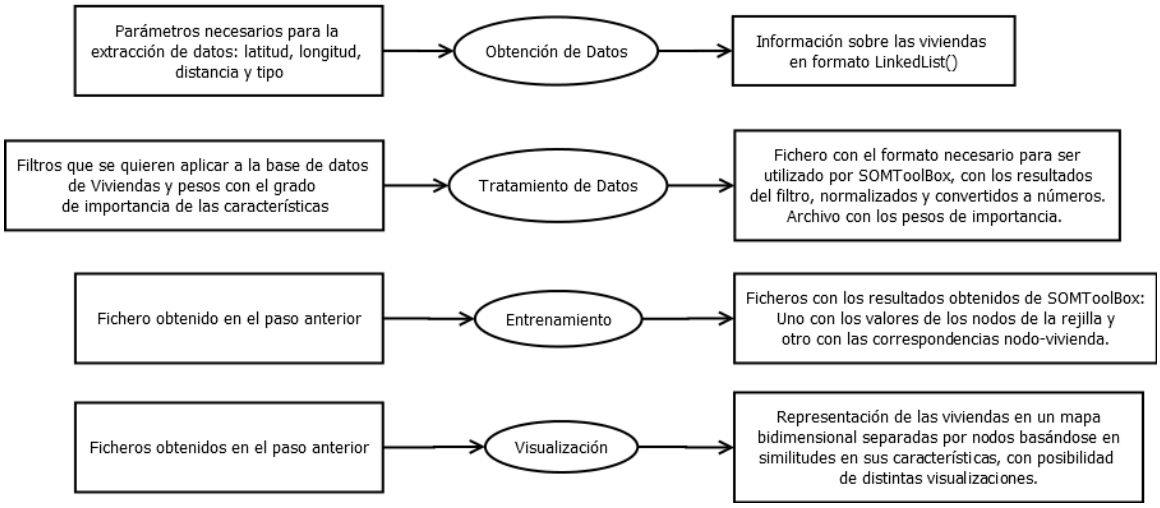


Figura 24. Fases de Diseño

3.2.1 Obtención de los datos

Los datos se obtienen con llamadas al API que proporciona Idealista. Esta API tiene un sistema de extracción de datos que no facilita demasiado una consulta concreta, debido a que entre sus parámetros de búsqueda no se encuentran básicos como la provincia (hay que hacer la búsqueda a partir de una posición (latitud y longitud) y un radio). Esto, unido al hecho de que cada llamada que se hace devuelve únicamente 20 viviendas, hace imposible realizar la obtención de los datos directamente del API cada vez que un usuario realiza una consulta.

Por ello, para este proyecto, se plantea la solución de mantener una base de datos de Idealista, actualizada cada poco tiempo; cogiendo unas dimensiones que cubran toda la zona geográfica de Madrid, y eliminando posteriormente las viviendas que pertenezcan a otra provincia.

Para cubrir toda la zona geográfica se plantea un centro peninsular, que se identifica en el cerro de los ángeles según varias páginas webs (longitud: -4,239180 y latitud: 40,240010) y una distancia máxima que será de 150 km.

Se crea la clase ExtraerDatos que realiza la siguiente funcionalidad:

- Hace llamadas al API hasta completar todas las viviendas dentro del radio seleccionado
- Elimina las viviendas cuya provincia no sea Madrid
- Guarda en un fichero un linkedlist con los datos guardados en objetos del tipo Vivienda (tipo creado en el que se representan todas las características de las

mismas). Linkedlist tipo de objeto Java con el que se pueden representar y tratar de forma sencilla listas de objetos (en este caso Viviendas).

Los datos obtenidos del API de Idealista se obtienen en un formato json, por lo que se necesita importar estas clases a nuestro proyecto, de tal forma que nos ayude a tratarlos y pasarlos al formato que deseamos.

Para facilitar la extracción de datos mediante este método, se diseña la siguiente interfaz, dentro de la clase Extracción, en la que se pueden completar todos los parámetros necesarios para realizar una búsqueda en el API, así como guardar los resultados y combinar varios ficheros de resultados.

En esta interfaz no se cubre la funcionalidad de eliminar las viviendas que no pertenezcan a Madrid, ya que esa acción de puede realizar en el siguiente paso, al filtrar las viviendas.

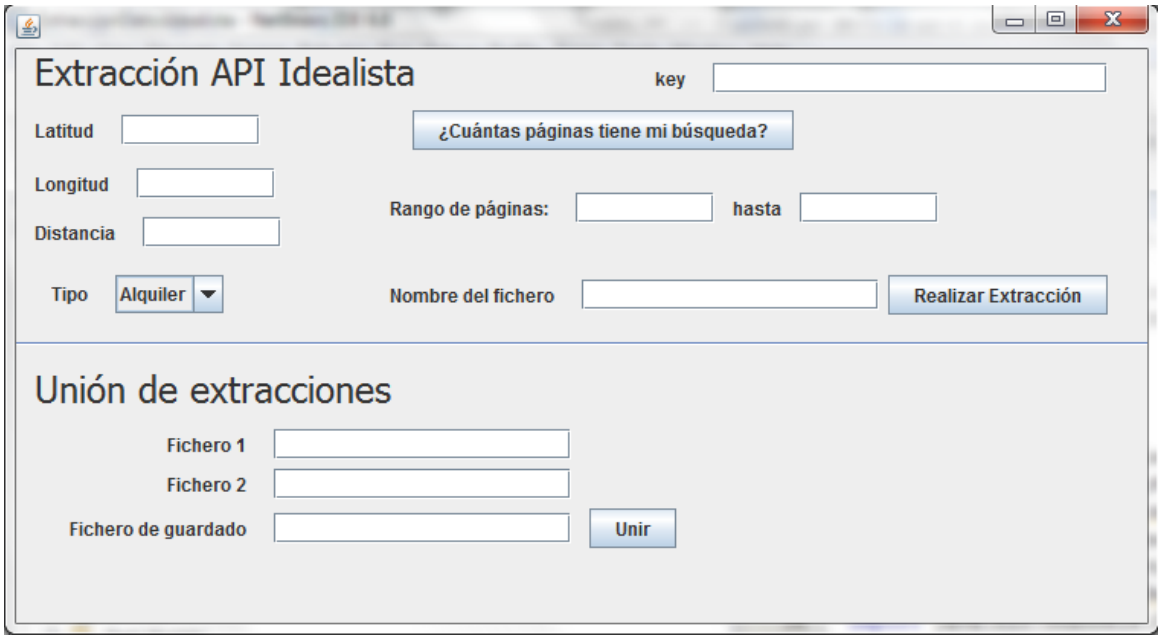


Figura 25. Interfaz para extracción de datos

3.2.2 Tratamiento de los datos

Tras extraer los datos de idealista y guardarlos en formato Vivienda en un LinkedList, el usuario seleccionará algunos filtros, a través de la siguiente interfaz, para reducir los datos a los que cubran unas necesidades mínimas del usuario.

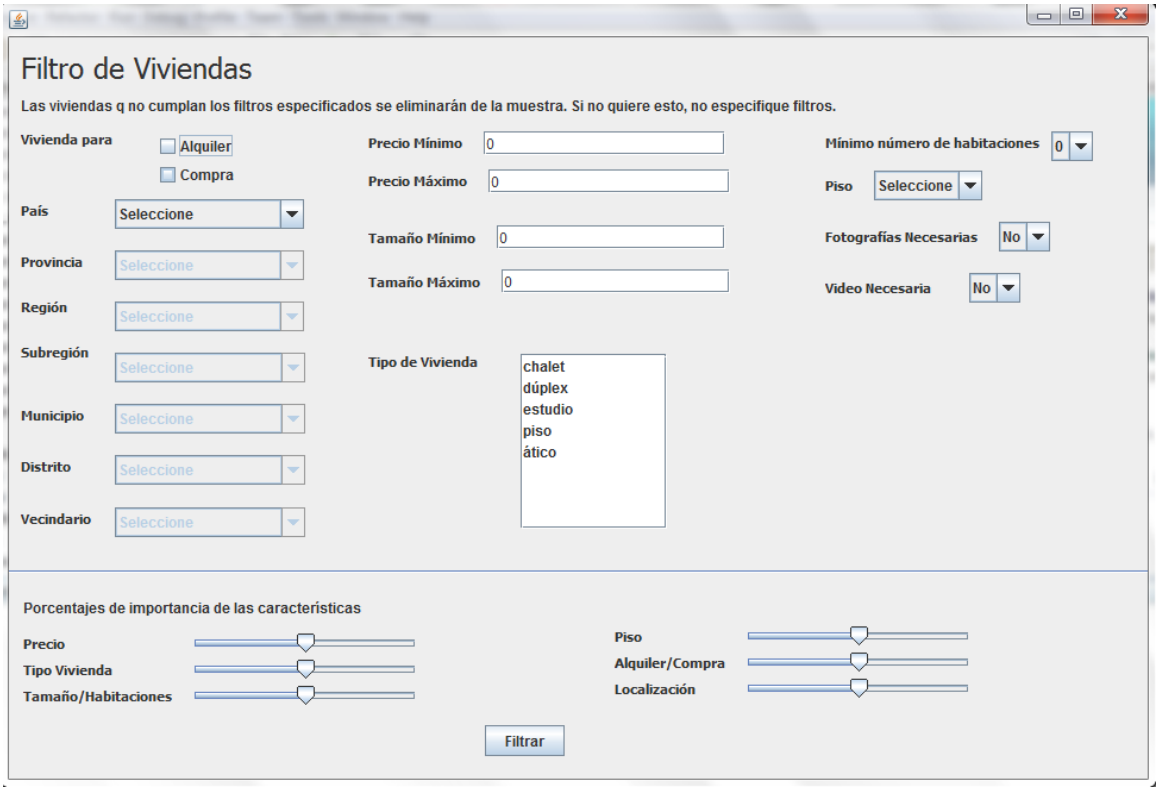


Figura 26. Pantalla de selección de filtros

Previo al acceso a esta interfaz, el sistema recorre todos los datos disponibles, para poder mostrar al usuario sólo los valores existentes en la base de datos, para las características que se tratan de enumerados; como puede ser el tipo de vivienda, o, sobre todo los datos de provincia/región/subregión/municipio/distrito/vecindario cuya aparición está relacionada con el valor elegido en el combo inmediatamente superior, por lo que se ha necesitado representar dicha jerarquía en un tipo de datos árbol creado específicamente para ello.

Esta clase Árbol, consta en un String que representará el valor de ese nodo, y un LinkedList del mismo tipo Arbol, que representará a los hijos. Dependiendo del nivel en el que se encuentre el valor, será de un tipo de dato o de otro; así el primer nivel es el país (un único nodo para España), el segundo es la provincia, y así sucesivamente.

Al acceder, el usuario verá dos zonas separadas por una línea horizontal.

La zona superior aplica filtros estrictos sobre los datos, es decir, todas las características que tengan algún valor tras ejecutar el filtro, hará que las viviendas que no cumplan con ese valor, desaparezcan de la muestra, ya que se entiende que el usuario quiere exclusivamente viviendas con los datos seleccionados. Así, por ejemplo, si el usuario selecciona que quiere una vivienda de alquiler, no se le mostrará ninguna vivienda que esté en venta.

La zona inferior permite al usuario seleccionar la importancia que le da a las características ahí representadas. El agrupamiento futuro de las viviendas, tendrá más o menos en cuenta las distintas características, dependiendo del grado de importancia que se seleccione en este apartado.

El usuario, tras hacer su selección, pulsará el botón “Filtrar”. Entonces, el programa comenzará con las tareas necesarias para depurar los datos y generar el archivo necesario para el entrenamiento.

La primera tarea es eliminar del conjunto de la muestra, las viviendas que no cumplan con los filtros especificados por el usuario. Para ello, disponemos de un método para cada uno de los campos de la interfaz que comprueban si se ha modificado el valor por defecto y, si es así, recorre todas las viviendas eliminando de la LinkedList las que no cumplan.

La segunda tarea es guardar los porcentajes de importancia en un archivo de texto para su uso posterior en la fase de entrenamiento.

La tercera tarea es generar un archivo csv (delimitado por comas) entendible por Weka, ya que esta herramienta será integrada en java para normalizar los datos. Este fichero no poseerá todos las características, debido a que algunas no son suficientemente relevantes para un usuario que quiere encontrar una casa, de tal forma que utilizaremos:

- Distrito
- Piso
- Municipio
- Barrio
- Operación
- Precio
- Código de tipo de propiedad
- Provincia
- Región
- Número de habitaciones
- Tamaño
- Subregión

De esta forma quedan fuera:

- Número de agencia
- Número de baños: ya que por un error del API, el valor es siempre 0
- País: siempre será España
- Favorito
- Tiene video
- Número de fotos
- Si muestra la dirección
- Tipo de video

A partir de las características seleccionadas, y haciendo una codificación numérica utilizando distintos números secuenciales para cada valor de los enumerados no numéricos, se genera un archivo con el siguiente formato:

```
district, floor, munipality, neighborhood, operation, price, propertyTypeCode, province, region, rooms, size, subregion
86, 1, 0, 110, 1, 110000, 6, 0, 3, 1, 5751, 0
111, 0, 0, 120, 1, 420000, 5, 0, 3, 5, 317, 0
111, 3, 0, 120, 1, 275000, 6, 0, 3, 3, 129, 0
118, 0, 0, 126, 1, 190000, 2, 0, 3, 2, 90, 0
130, 0, 0, 134, 1, 430000, 4, 0, 3, 4, 250, 0
130, 0, 0, 146, 1, 450000, 4, 0, 3, 4, 280, 0
118, 2, 0, 126, 1, 119000, 2, 0, 3, 3, 66, 0
30, 4, 0, 101, 1, 150000, 2, 0, 3, 4, 100, 0
130, 1, 0, 151, 1, 249000, 2, 0, 3, 3, 110, 0
130, 0, 0, 152, 1, 350000, 6, 0, 3, 3, 122, 0
118, 0, 0, 126, 1, 195000, 2, 0, 3, 2, 82, 0
30, 1, 0, 101, 1, 150000, 2, 0, 3, 3, 79, 0
130, 0, 0, 158, 1, 388000, 4, 0, 3, 4, 240, 0
191, 6, 0, 159, 1, 160000, 2, 0, 3, 3, 80, 0
191, 4, 0, 159, 1, 152000, 2, 0, 3, 3, 89, 0
191, 6, 0, 159, 1, 157000, 2, 0, 3, 3, 80, 0
86, 0, 0, 110, 1, 480000, 4, 0, 3, 4, 270, 0
130, 3, 0, 152, 1, 200000, 2, 0, 3, 3, 71, 0
111, 1, 0, 120, 1, 240000, 2, 0, 3, 2, 78, 0
```

Figura 27. Documento .csv para Weka

El formato tiene las características de las distintas viviendas separadas por comas, teniendo una primera fila, a modo de encabezado, con el nombre de cada característica, que Weka utilizará para referirse a ella.

Los datos de este fichero serán normalizados gracias a una llamada al filtro Normalize() de Weka, para evitar que los campos cuya diferencias son mayores, por tener un rango más amplio (como el precio) afecte en mayor medida que el resto de campos a la hora de buscar el nodo ideal dentro del mapa.

Una vez los valores estén normalizados, se genera con ellos el fichero necesario para introducir en la herramienta de entrenamiento. Este fichero tendrá el siguiente formato:

```
$TYPE Viviendas
$XDIM 1445
$YDIM 1
$VECDIM 11

0.380531 0.1 0 0.591398 0 0.084298 0.833333 0 0 0.142857 1 0
0.49115 0 0 0.645161 0 0.322843 0.666667 0 0 0.714286 0.049335 0
0.49115 0.3 0 0.645161 0 0.211265 0.833333 0 0 0.428571 0.016445 0
0.522124 0 0 0.677419 0 0.145858 0.166667 0 0 0.285714 0.009622 0
0.575221 0 0 0.72043 0 0.330537 0.5 0 0 0.571429 0.037614 0
0.575221 0 0 0.784946 0 0.345927 0.5 0 0 0.571429 0.042862 0
0.522124 0.2 0 0.677419 0 0.091224 0.166667 0 0 0.428571 0.005423 0
```

Figura 28. Documento generado para el entrenamiento

En él, se ha definido el tipo de datos como “Viviendas” y se puede ver que existen 1445 viviendas que cumplan nuestro filtro inicial (XDIM x YDIM) y que cada una de ellas tiene 11 características.

3.2.3 Entrenamiento del mapa

El entrenamiento, se hará utilizando el entrenador Growing SOM, ya que al tener un filtro inicial, el número de viviendas puede variar, por lo que no tiene sentido tener unas dimensiones de mapa fijas.

Comienza la ejecución, haciendo una llamada al entrenador SOMTrainer a partir de un método modificado a partir del original.

Las modificaciones realizadas en el método cubren la url de acceso de los datos y la generación de las propiedades seleccionadas a partir de este método. Como ya se ha explicado en el apartado del programa SOMToolBox, los parámetros que se han tenido que añadir al método para que se pueda ejecutar el entrenador son:

- Directorio de salida de datos tras el entrenamiento
- Directorio de trabajo
- Ruta del fichero de datos de entrenamiento
- Tamaño del mapa (x e y)
- Tasa de aprendizaje
- Variable tau para calcular el error del mapa, con el que se definirá si ha de ampliarse o no.
- Número de iteraciones que se van a ejecutar en el entrenamiento.

El método original extraía los valores de las distintas propiedades de la interfaz, por lo que se modificó íntegramente para añadirle los valores que se van a utilizar en este proyecto directamente.

El algoritmo, explicado ya en un apartado anterior, se modifica para que tenga en cuenta los pesos seleccionados por el usuario en la pantalla principal y así, calcule la distancia euclídea ponderada, ya explicada en el apartado *2.5 Self Organizing Maps*.

Estas modificaciones se realizan en la clase L2Metric.java, encargada de calcular la distancia euclídea entre dos vectores de características (en este caso, la vivienda, y el nodo correspondiente del mapa); Y consistirán en tener en cuenta el peso a la hora de sumar la distancia de una característica a la distancia final. Para ello, al sumar la distancia, esta se multiplicará por el peso, de tal forma que si es alto (el usuario lo ha seleccionado como muy importante) contará más a la hora de calcular la distancia; y si es un peso pequeño (o cero, porque al usuario no le interesa esa característica) prácticamente no se tendrá en cuenta en la distancia.

Como sabemos de antemano el orden de las características en el fichero de datos de las viviendas, nos resulta sencillo relacionarlas con su peso correspondiente. Esa relación se hace de la siguiente manera:

Número Peso	Peso	Característica
0	Precio	Precio
1	Tipo Vivienda	Tipo de vivienda
2	Tamaño/Habitaciones	Tamaño Habitaciones
3	Piso	Piso
4	Alquiler/Compra	Operación
5	Localización	Distrito Municipio Vecindario Provincia Region Subregion

Tabla 5. Relación entre pesos y características

3.2.4 Visualización de los mapas

La visualización se hará, como el entrenamiento, haciendo uso de las funcionalidades de la herramienta SOMToolBox. Se realizan las modificaciones explicadas a continuación en la herramienta para que sea más útil en el caso que estamos trabajando y más amigable para el usuario.

Para facilitar la visualización al usuario, lo primero que se decide es eliminar de la interfaz todos los desplegables que superan las necesidades del mismo; dejando únicamente las visualizaciones óptimas para los datos que estamos tratando.

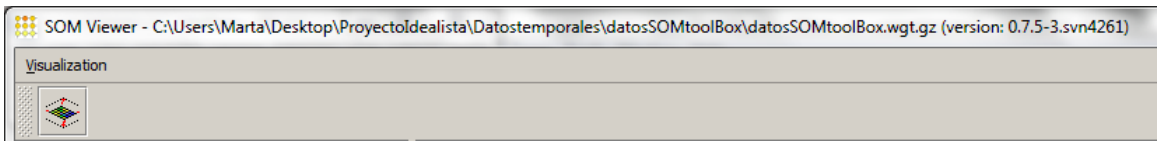


Figura 29. Barra de Tareas Superior de la Visualización tras modificaciones

Aunque se había planteado eliminar la barra de tareas lateral en un principio, finalmente se mantiene para que el usuario pueda regular los distintos parámetros de las visualizaciones que no se pueden hacer de forma automática.

De igual forma, se eliminan las visualizaciones no aptas para el usuario, por no aportar ningún tipo de información a esta representación concreta, o por necesitar archivos no generados en este proyecto.

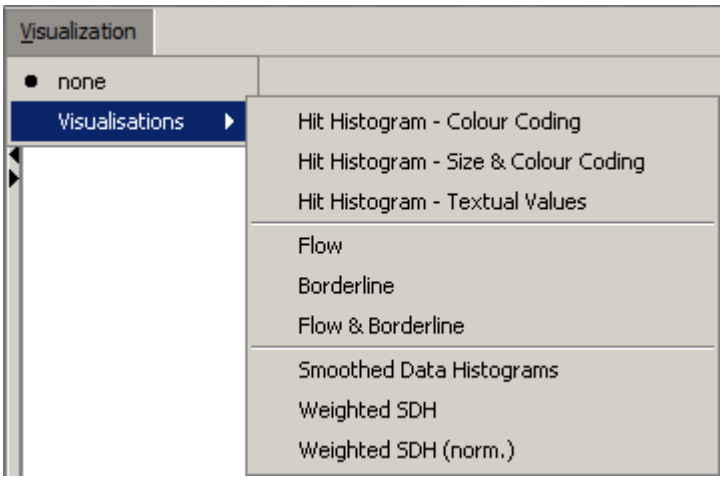


Figura 30. Visualizaciones disponibles tras las modificaciones

Otra modificación importante dentro de la visualización es la de los datos a mostrar en los nodos. Para ello, se decide mostrar como código representativo, el que define a la vivienda en Idealista, el "código de vivienda". Para ello, necesitamos una forma de saber a qué vivienda pertenece cada registro representado en los nodos, por lo que guardamos el orden secuencial que mantienen las viviendas en el fichero de datos, que será inalterable. De esta forma, si en el fichero "*unitDescriptionFile*" se guarda en cierta neurona el número 14, sabremos que será la vivienda que se encuentra en el fichero de entrada en la posición 15 (ya que se empieza a contar desde 0).

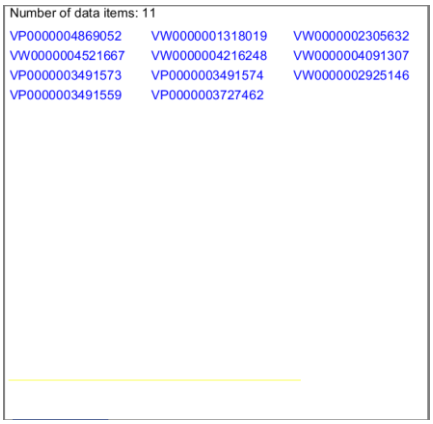


Figura 31. Visualización de un nodo tras las modificaciones.

Esto, a priori, no parece presentar ninguna información relevante al usuario, pero es una forma unívoca de presentar las viviendas en el mapa.

Para que el usuario tenga más información de cada vivienda, y esto sí es relevante, se agregarán una serie de informaciones sobre la misma al pasar el ratón encima de cada código. Estas informaciones serán las mismas que hemos utilizado para calcular las distancias euclídeas en los entrenamientos.

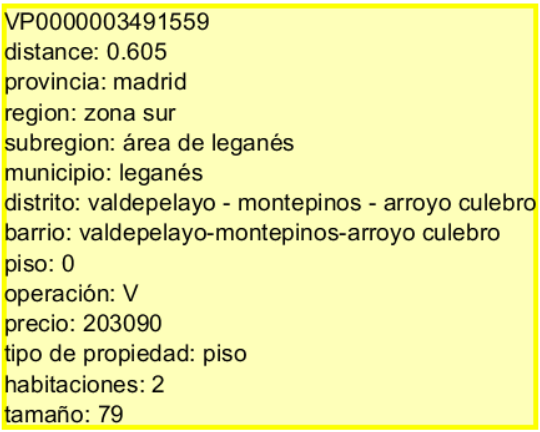


Figura 32. Visualización de las características de una vivienda a través de un nodo

Esta modificación hace que el programa, tras buscar la vivienda que corresponde con el numeral introducido en el nodo (como hemos descrito antes), añade al cuadro de texto donde únicamente se estaba mostrando la distancia, todos los datos que se requieren. En este caso serán:

- Provincia
- Región
- Subregión
- Municipio
- Distrito
- Barrio
- Piso
- Operación
- Precio
- Tipo de Propiedad
- Habitaciones
- Tamaño

Como última modificación, se agrega una funcionalidad con la que los usuarios podrán acceder a la página web de idealista, con un doble click encima del código de la vivienda que les interesa.

La visualización final quedaría de la siguiente forma:

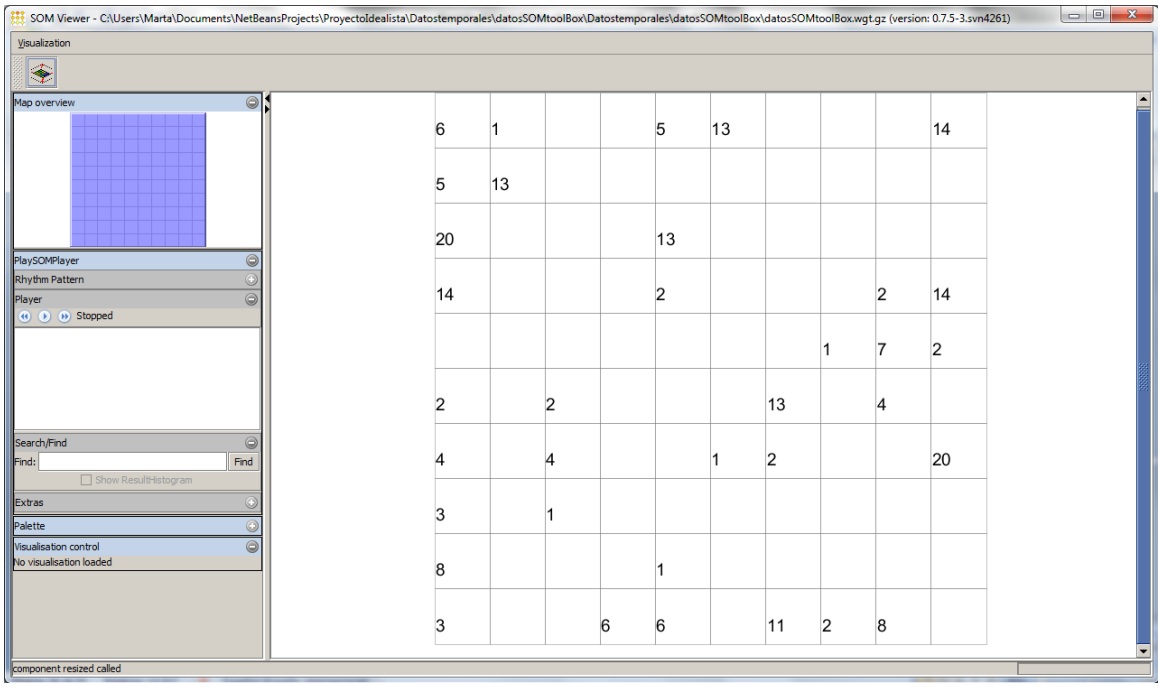


Figura 33. Resultado final de la visualización

En ella se permite en orden descendente:

- Seleccionar la visualización que se quiere aplicar al mapa.
- Centrar el mapa para que se vea completo.
- Ver las distintas propiedades aplicadas en el mapa.
  - Paleta de colores.
  - Parámetros para algunos tipos de visualizaciones.
- Visualizar el mapa.
  - Ver en cada nodo los números identificativos de las viviendas que lo componen.
  - Ver en cada vivienda los valores de sus características.
  - Acceder a la dirección web de Idealista dónde se anuncia la vivienda a través de su número.

: Trabajo Realizado

Capítulo 4

Evaluación

En este apartado se van a realizar una serie de pruebas y ejemplos sobre la herramienta para evaluar la viabilidad de usarla en un futuro como herramienta web de un usuario habitual de búsqueda de viviendas.

Para empezar, se realiza un ejemplo básico, en el que se selecciona cómo única característica importante para el usuario el precio (para el resto indicaremos un peso de 0). Se trabaja sobre la base de datos filtrada únicamente para viviendas del municipio "Leganés". El resultado obtenido es un mapa en el que se han dividido las viviendas por precio. En la siguiente figura se indica el rango de precios de cada cuadro para entender mejor la división que se ha hecho de las viviendas.

227000-229000 8	219000 1	202000-203000 6	180000 30	160000 25	145000-146000 16	125000-127500 14	103000-107000 16	94000-96000 27	69000-83000 112
234000-235000 12	224000-225000 12	206000-207000 2	185000 7		147000-149000 16	132000-135000 20	110000-112000 3	97000-102000 43	89000-93000 20
241000-242000 2	231000-233000 5	210000-214000 15	190000 16	172000 3	154000-155000 9	142000 2	124000-125000 10	224000-225000 29	107000-110000 35
247000-250000 25	238000-240000 17		200000 20	182000 5		149000-150000 14	135000-139000 10	128000-130000 20	119000-124000 46
	253000-255000 6	237000-238000 3	215000-218000 9	198000-199000 8		162000-163000 4	151000-153000 7	143000-144000 4	139000-140000 20
285000 7	270000-277000 14	255000-258000 4	241000 1	222000-223000 5	199600 3		164000-166000 19	157000-159000 16	156000-157000 6
323000-330000 11	310000 3	286000 1	265000-270000 12	243000-246000 12	220000 17	193000-195000 11	176000-178000 7	170000 13	167000-169000 13
368000-380000 17	347000-350000 10	315000 3	289000-298000 23	264000 2	236000 1	208000-209000 3	188000-189000 6		172000-176000 30
420000-435000 32	383000-402000 18	335000-345000 9	299000-308000 22	278000-282000 10	251000 4		198000 3	185000-186000 9	179000 2
>440000 69	407000-416000 6	360000-365000 14	319000-320000 7	228000 3	260000 8	230000 5	204000-205000 3	192000 4	183000 3

Figura 34. Evaluación de viviendas de Leganés por precio

Cada nodo tiene un valor de características, en este caso de precio, distinto al de los demás nodos de forma que entre todos, cubran los posibles valores del conjunto de las viviendas. Como se puede observar, no se han ordenado de forma aleatoria, si no que existe una relación entre ellos, quedando los de mayor valor en la esquina inferior izquierda, y los de menor valor en la superior derecha.

De esta forma, cuando el usuario encuentre una vivienda acorde a lo que estaba buscando, podrá acceder a más viviendas parecidas mirando los nodos más cercanos.

Este ejemplo no aporta un avance importante en cuanto a los buscadores habituales en los que se podría ordenar por precio y mirar así el rango de precios aceptables; pero como hemos comentado esto es un ejemplo básico para entender el funcionamiento y ordenación de los mapas, que será más difícil de apreciar cuando el número de características sea mayor.

Con esta ejecución también se puede obtener información sobre el precio más común entre las viviendas de Leganés. Para ello se utiliza la visualización *Smooth Data Histogram* que nos mostrará en blanco el rango de precios más común.

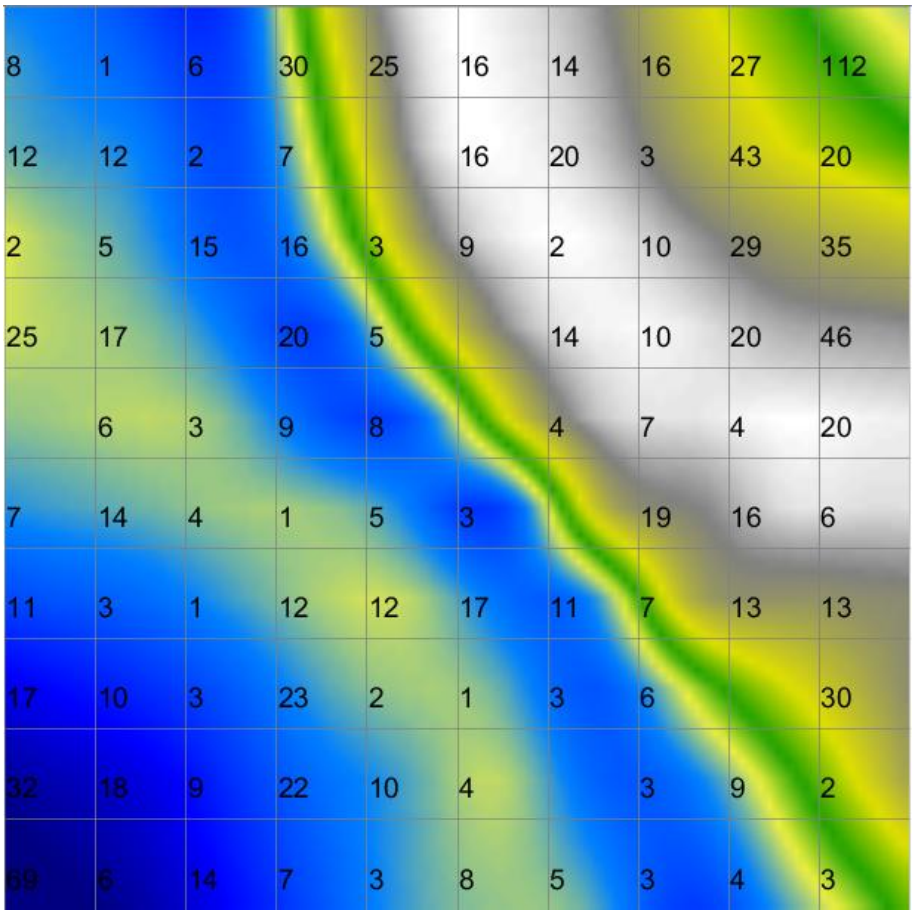


Figura 35. Evaluación de viviendas de Leganés por precio. Smooth Data Histogram

Si se compara este mapa con el anterior, se ve que el clúster más poblado, o los precios más repetidos, se encuentran entre 119 y 157 miles de euros.

En el siguiente ejemplo se utilizan dos características, precio y tipo de vivienda, para ver cómo se integran ambas en el cálculo de la distancia, y si es posible observar la clasificación en el mapa. También, se van a reducir el número de viviendas para facilitar el estudio dejando las viviendas para compra, en el municipio “Moralzarzal”.

Se muestra, como en el ejemplo anterior, el mapa obtenido con los valores de las dos características utilizadas para facilitar la observación.

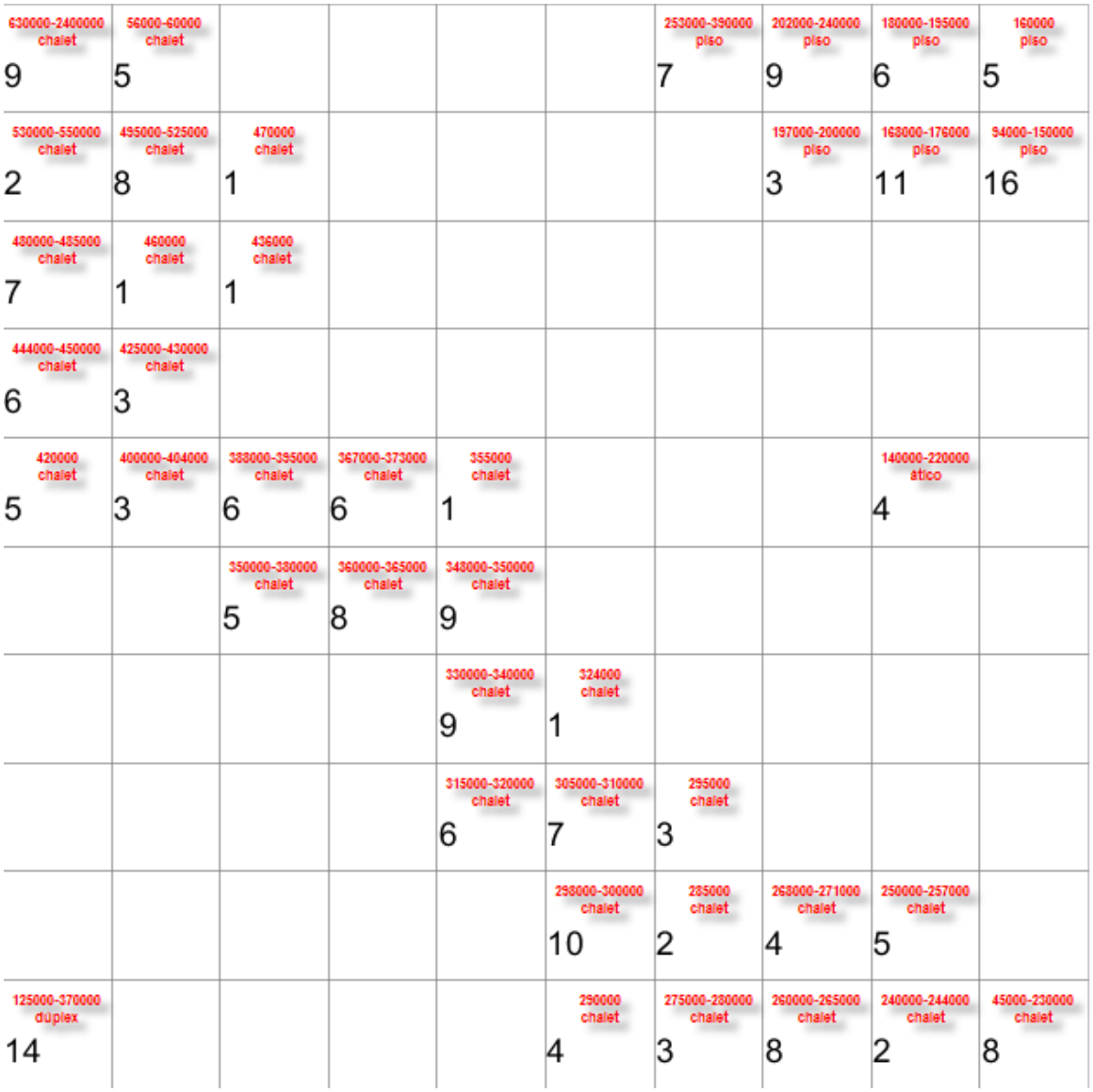


Figura 36. Evaluación de viviendas de Moralzarzal por precio y tipo

Se ve claramente que se han formado 4 grupos diferenciados, uno por cada tipo de vivienda y dentro de ellos, algunos tipos están divididos en varios nodos por precio, dependiendo del número de viviendas con ese tipo.

El usuario podrá acceder rápidamente a las viviendas del tipo que prefiera y a partir de uno de esos nodos, acceder a sus vecinos si no está conforme con el precio.

La clasificación se hace de forma escalonada, primero con el tipo y luego por el precio. Esto está provocado porque la cantidad de valores del tipo es mucho inferior que la de los precios, por lo que al normalizarlos y al calcular la diferencia, siempre contará en la distancia de forma mayor que el precio; aunque su peso asignado sea el mismo.

Para terminar, se representan todos los pisos en venta en Moralzarzal, poniendo con la misma importancia todas las características. El peso de todas será 50%. En este caso será más difícil de apreciar los datos que ha utilizado para realizar la separación en nodos,

puesto que al final la distancia con la que se hace esa separación, se calcula como una suma de las distintas distancias entre características; sin diferenciación.

Las características que se utilizan en este caso son las siguientes:

- precio
- tipo de vivienda
- número de habitaciones
- tamaño
- distrito
- barrio
- piso
- si es alquiler o compra. Debido a que se ha filtrado por compra, este filtro no aportará información.

El mapa resultado es:

6	1			5	13				14
5	13								
20				13					
14				2				2	14
							1	7	2
2		2				13		4	
4		4			1	2			20
3		1							
8				1					
3			6	6		11	2	8	

Figura 37. Evaluación de viviendas de Moralzarzal por todas las características.

Como ya se ha comentado, en este caso es más complicado analizar las características por separado, aunque en este ejemplo se pueden ver varias condiciones que estructuran el mapa:

- El piso cobra mucha fuerza debido al bajo número de valores que tiene. El piso representa la planta del edificio en que se encuentra la vivienda. Así, como en Moralzarzal no se encuentras edificios altos, al ser un pueblo de sierra, la mayoría de los edificios con bajos, lo que provoca tener valores no superiores a 3. En la zona superior izquierda se encuentran las viviendas con piso 0, apoyadas a su vez por el tipo de vivienda chalet (todos los chalets tienen piso 0 al ser edificios con vivienda única). En el borde inferior se concentran los pisos altos, mientras que en la esquina superior derecha se encuentran los pisos 1 y 0 que coinciden con los tipos de vivienda dúplex.
- No existe ninguna vivienda con barrio asignado, por lo que será una variable que no se tendrá en cuenta.
- En cuanto a los distritos, en la zona lateral izquierda se encuentran todas las viviendas de la "zona iglesia-estación", mientras que en resto de zonas se dividen más a la derecha, dejando en la zona más lateral derecha la "zona centro". Esto puede estar causado por el valor secuencial que se le asigna a cada distrito para convertir el dato en enumerado.
- Los números de habitaciones más altos se encuentran en la esquina superior izquierda y en el lateral derecho (en las coordenadas del mapa x=10 e y=3).

Capítulo 5

Conclusiones

Una vez finalizado el desarrollo y probado distintos ejemplos, se hace una reflexión sobre el resultado obtenido y las mejoras futuras que se pueden realizar. Este es el objetivo de este capítulo.

5.1 Conclusiones

Con este proyecto se ha realizado una herramienta que presenta una nueva aplicación para los mapas de Kohonen (o mapas auto organizativos) que consiste en la clasificación de una base de datos de Viviendas.

Estas viviendas se han obtenido a partir del API de Idealista. Idealista es un portal inmobiliario que ha puesto al servicio de los desarrolladores un API, con su correspondiente documentación para utilizarlo, con el que se podrá extraer todas las características que definen a las viviendas de su portal. Aunque está bien planteado, está bastante restringido a la hora de obtener información de forma masiva; cada llamada únicamente devuelve 20 registros y tarda aproximadamente 3 segundos de media, lo que da como resultado un tiempo de casi 5 horas para las 119503 viviendas que tienen de Madrid en el momento de escribir esta memoria. Esto no sería un problema si no fuera porque cada día existen modificaciones en el portal, y habría que actualizarlo diariamente, lo que tomaría 5 horas al día. Además, a la hora de decidir las viviendas que se quieren extraer del API, no da juego a hacer una selección cerrada, ya que los

parámetros que permite utilizar son únicamente de situación en el mapa (longitud y latitud, con variantes).

Como lado positivo, el formato con el que Idealista facilita los datos de sus viviendas es muy fácilmente tratable con Java.

Weka, es una herramienta de análisis de datos con una amplia lista de algoritmos con los que hacer tareas de minería de datos. Además, es código libre integrable en Java, por lo que se han utilizado algunos de sus filtros para tratar los datos de las viviendas. El problema es que no hay prácticamente ninguna documentación útil en la red que ayude a integrarla, a pesar de los múltiples manuales que existen para utilizar su versión ejecutable con interfaz de usuario.

La parte positiva del uso de Weka es la total fiabilidad que aporta, ya que es una herramienta que lleva usándose desde hace muchos años.

La herramienta que más problemas ha provocado al utilizarse es SOMToolBox, por dos razones principales

- La documentación existente es únicamente para utilizar el ejecutable; para el código se encuentra el javadoc obtenido de java con información de sus clases, pero no un documento que facilite su uso. Es una herramienta con una gran cantidad de clases y sin documentación, por lo que se ha invertido mucho tiempo estudiando su funcionamiento y buscando las partes donde se quería realizar modificaciones, sobre todo, debuggeando y siguiendo el flujo que realiza la herramienta con una ejecución normal desde el GUI.
- Es el núcleo del proyecto, por lo que la mayoría de los desarrollos se basan en ella.

Es una herramienta muy amplia, de la que sólo hemos utilizado dos partes, el entrenador *Growing SOM* y el visualizador.

Como conclusiones a los resultados obtenidos, se ha visto en el apartado de Evaluación, que se puede utilizar la herramienta para clasificar las viviendas, pero resulta más apreciable el resultado cuanto menor es el número de características que se utilizan. Esto es debido a la forma en la que se calcula la distancia (sumando la diferencia de todas las características) y a la forma de enumerar las características que son texto por defecto.

La parte del cálculo de la distancia se puede mejorar aplicando pesos; no únicamente los impuestos por el usuario, sino unos definidos por defecto, que sean menores para las características que más se tienen en cuenta.

La forma de enumerar las características también puede mejorarse buscando una relación entre los distintos valores textuales, a la hora de darles el valor numeral; por ejemplo, una relación geográfica para los distritos o una relación de tamaño para los tipos de vivienda.

A pesar de estos comentarios sobre el cálculo, sí se ven resultados completamente útiles en el caso de elegir una o dos características: las viviendas se dividen en nodos dependiendo del valor de estas características.

El problema comienza cuando se aumentan el número de características y, aunque la distancia resultado es la misma, las viviendas y sus características no tienen nada que ver. Como ejemplo:

Vivienda	Piso	TipoVivienda	Neurona (valores para sus dos características)	Distancia
1	2	1 (chalet)	(0,0)	2,24
2	1	2 (piso)	(0,0)	2,24

Tabla 6. Ejemplo de viviendas con misma distancia pero distintas características

La fórmula de la distancia es:

$$\sqrt{\sum_{i=0}^{i=n^{o}características}(Característica_{i1} - Característica_{i2})^2}$$

Aplicada a ambos ejemplos:

$$\sqrt{(0-2)^2 + (0-1)^2} = \sqrt{5} = 2{,}24$$

$$\sqrt{(0-1)^2 + (0-2)^2} = \sqrt{5} = 2{,}24$$

Comprobamos que la distancia será 2,24 aprox. para ambas, aunque sus características no sean las mismas.

Lo que plantea al final la herramienta, es una forma fácil de dividir lo obtenido en una búsqueda con muchos resultados, de forma que no haga falta recorrer todos para encontrar el deseado. El usuario, simplemente necesitará buscar desde dónde empezar, buscando alguna vivienda parecida a la objetivo, y recorrer los resultados desde ese nodo hacia sus vecinos.

La mayor desventaja que presenta esta herramienta es que la visualización no es del todo intuitiva, y una persona no acostumbrada a este tipo de representaciones podría no entender su significado. Es algo técnico que podría no ser apto para publicar en un portal que quiere tener una clientela de todas las edades y capacidades.

5.2 Ideas para el Futuro

Se plantean varias acciones a realizar a futuro para mejorar la herramienta:

- Asignar el número a los enumerados por algún tipo de orden, en lugar de ser únicamente secuencial. Por ejemplo, en los distritos, buscar los que se encuentren más cercanos y darles numerales también cercanos. Esto solucionará en la representación el hecho de que cuando se clasifican viviendas por esos campos enumerados, se muestran más cercanos los nodos con distritos con número más parecido, aunque no tengan ninguna cercanía entre ellas.
- Mejorar la velocidad de la herramienta. El entrenamiento toma mucho tiempo, y eso no se podría reducir (se podría poniendo menos iteracciones, pero no nos conviene, empeoraría la calidad de los resultados); pero también toma mucho tiempo abriendo los ficheros de datos y guardando los que va necesitando. Como solución, si la herramienta se mantuviera en ejecución, no necesitaría cargar los datos porque los tendría en memoria.
- Ampliar la base de datos. Actualmente sólo se utilizan viviendas de Madrid, pues se intentó hacer una extracción de toda España, pero, además de tomar mucho tiempo las llamadas al API de Idealista, al intentar tratarlos con la herramienta presenta faltas de memoria por parte de Java. Aunque se puede aumentar la memoria de ejecución de Java, empeoraría el punto anterior: el tiempo de ejecución y tratamiento.
- Mostrar un resumen de cada nodo del mapa, para facilitar la búsqueda al usuario. Este resumen mostraría los valores incluidos en el nodo para cada característica.
- Integrar en la interfaz de visualización un frame para mostrar imágenes e información de la vivienda en lugar de tener que acceder al enlace web en el navegador del equipo.
- Hacer un estudio de usabilidad desde el punto de vista de interfaz hombre-máquina incluyendo mejoras según el resultado de este estudio.

Glosario

API	*Application Programming Interface*
CSV	*Comma separated values*
EM	*Esperanza-maximización*
IP	*Internet protocol*
NN	*Nearest Neighbour*
PDF	*Portable document format*
SOM	*Self Organizing Maps*
URL	*Uniform resource locator*

Referencias

[APIIDEALISTA] Api de Idealista. Disponible [Internet]: <http://www.idealista.com/labs/api.htm> [21 de noviembre de 2012]

[APRENDIZAJESUPERVISADO] Constantino Malagón Luque. *Clasificacón y regresión mediante aprendizaje supervisado*. Disponible [Internet]: <http://www.slideshare.net/yerartSlide/clasificacin-y-regresin-mediante-aprendizaje-supervisado> [21 de noviembre de 2012]

[FLOWBORDERLINE] Georg Pölzbauer, Michael Dittenbach y Andreas Rauber. *Advanced Visualization of Self-Organizing Maps with vector fields*. Disponible [Internet]: <http://www.ifs.tuwien.ac.at/~andi/publications/pdf/poe_nn06.pdf> [21 de noviembre de 2012]

[GROWINGGRID] Bernd Fritzke. *Growing Grid – a self-organizing network with constant neighborhood range and adaptation strenght*. Disponible [Internet]: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=2A14403ABF6A2E2327FF19A6CD526E59?doi=10.1.1.54.8183&rep=rep1&type=pdf> [21 de noviembre de 2012]

[SISTEMASRECOMENDACION] Ismael Rafael Ponce Medellín. *Sistemas de Recomendación.* Disponible [Internet]: <http://www.slideshare.net/kamui002/sistemas-de-recomendacin > [21 de noviembre de 2012]

[SMOOTHDATAHISTOGRAMS] Elias Pampalk, Andreas Rauber y Dieter Merkl. *Using Smoothed Data Histograms for Clúster Visualization in Self-Organizing Maps.* Disponible [Internet]: <http://reference.kfupm.edu.sa/content/u/s/using_smoothed_data_histograms_for_clust_57150.pdf > [21 de noviembre de 2012]

[SOMTOOLBOX] Documentación sobre SOM Tool Box. Disponible [Internet]: <http://www.ifs.tuwien.ac.at/dm/somtoolbox/> [21 de noviembre de 2012]

[TKOHONEN] Tauvo Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York, 1995, 1997, 2001, 3rd edition.

[VECTORFIELDS] Georg Pölzbauer, Michael Dittenbach y Andreas Rauber. *A visualization technique for Self-Organizing Maps with vector fields to obtain the clúster structure at desire levels of detail*. Disponible [Internet]: <http://www.ifs.tuwien.ac.at/~poelzlbauer/publications/Poe05IJCNN.pdf> [21 de noviembre de 2012]

Anexos

Ejemplo de extracción de datos de Idealista

Con el objetivo de ver el formato con el que el API del portal inmobiliario Idealista devuelve la información sobre las viviendas, adjuntamos el ejemplo obtenido al ejecutar la siguiente URL:

http://www.idealista.com/labs/propertyMap.htm?k=072fb542a43c27aee90062a0004b9f1e&action=json&province=alicante&operation=V&distance=70000&center=40.483536,-3.664069&numPage=1

Figura 39. Extracción del API de Idealista

Planificación

A continuación se incluye las tareas que se han realizado a lo largo del tiempo que ha llevado este proyecto. Se diferencian 3 principales partes que son el análisis y estudio de las herramientas a utilizar, el desarrollo de la herramienta y la documentación

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Proyecto Idealista**	**313 días**	**jue 15/09/11**	**lun 26/11/12**	
2	Análisis del problema	10 días	jue 15/09/11	mié 28/09/11	
3	Estudio del API de Idealista	9 días	jue 29/09/11	mar 11/10/11	2
4	Estudio de la herramienta SOMToolBox	29 días	mié 12/10/11	lun 21/11/11	3
5	**Desarrollo**	**212 días**	**mar 22/11/11**	**mié 12/09/12**	
6	Extracción de los Datos	15 días	mar 22/11/11	lun 12/12/11	4
7	Creación de la Interfaz	23 días	jue 02/02/12	lun 05/03/12	6FC+37 días
8	Integración WEKA	10 días	mar 06/03/12	lun 19/03/12	7
9	Integración SOMToolBox	40 días	jue 07/06/12	mié 01/08/12	8FC+57 días
10	Visualización	30 días	jue 02/08/12	mié 12/09/12	9
11	Memoria	48 días	jue 13/09/12	lun 19/11/12	10
12	Presentación	5 días	mar 20/11/12	lun 26/11/12	

Para consultar el diagrama de Gantt, ir al final del documento.

Presupuesto

El presupuesto que se ha necesitado para realizar este proyecto se ve desglosado en las siguientes páginas.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Marta Viana Colino

2.- Departamento: Informática

3.- Descripción del Proyecto:

- Titulo: **Minería de Datos en Portal Inmobiliario**
- Duración (meses): **14**
- Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

5.996,80 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Viana Colino, Marta		Ingeniero Junior	2,4	1.791,67	4.300,00	
					0,00	
					0,00	
		Hombres mes	**2,4**	**Total**	4.300,00	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador Portatil	1.790,00	100	10	60	298,33
		100		60	0,00
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	298,33

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Conexión a Internet	Movistar	399,00
	Total	399,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	4.300
Amortización	298
Subcontratación de tareas	0
Costes de funcionamiento	399
Costes Indirectos	999
Total	5.997

ANEXOS

El presupuesto total de este proyecto asciende a la cantidad de 5.996,80 €.

Leganés a 27 de Octubre de 2012

El ingeniero proyectista

Fdo. Marta Viana Colino

Medios empleados

Para la realización de este proyecto, se ha utilizado el siguiente hardware y software:

- Hardware
  - Ordenador portátil
  - Conexión a internet
- Software
  - Navegador de internet
  - NetBeans
  - Weka
  - SOMToolBox

Manual de Usuario

Para utilizar la herramienta, descomprimir el fichero zip, en el que encontraremos el fichero ExtraccionDatosIdealista.jar, aplicación con la que obtener la información del portal inmobiliario y el fichero ProyectoIdealista.jar que es el ejecutable de la herramienta, además de varias carpetas de datos.

Para realizar la ejecución de ambas aplicaciones, acceder a la consola ms-dos, programa cmd en el menú Inicio de Windows, buscar la ruta donde se ha descomprimido y teclear "java –jar" seguido del nombre de la aplicación que se quiere ejecutar.

La ejecución de ExtraccionDatosIdealista.jar dará como resultado la siguiente interfaz:

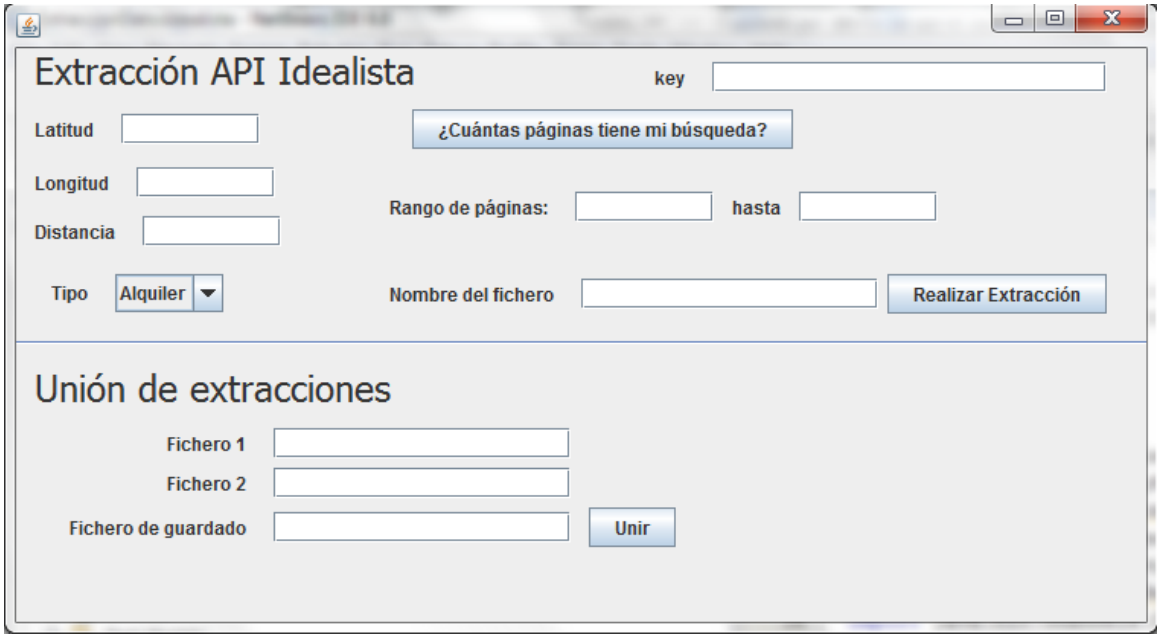


Figura 41. Manual de Usuario. Interfaz de extracción de datos

Para empezar a utilizar las funciones de esta aplicación, se necesita una key que nos permita hacer llamadas al API. Se obtiene a partir de la url http://www.idealista.com/labs/api.htm introduciendo la dirección IP desde dónde se va a ejecutar la aplicación.

En ella podemos realizar 3 acciones distintas; todas encaminadas a obtener la información de deseemos del API de Idealista. En la parte superior podemos introducir todos los datos necesarios para realizar una extracción. Estos son Latitud, Longitud, Distancia y Tipo de Acción. Una vez seleccionados estos datos, utilizaremos el botón "¿Cuántas páginas tiene mi búsqueda?" para que el sistema haga su primera llamada al API y nos devuelva el número de páginas de resultados que nos presenta esa búsqueda (cada página está formada por 20 viviendas).

Al pulsar dicho botón nos saldrá un mensaje parecido al siguiente:

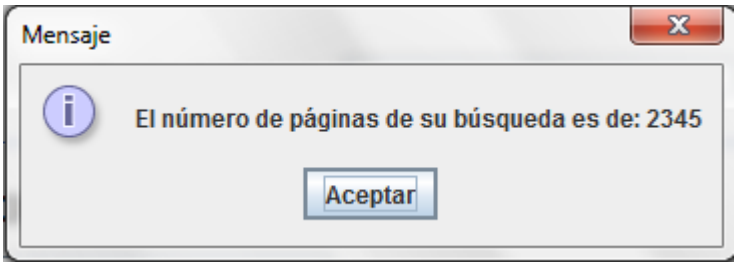


Figura 42. Manual de Usuario. Mensaje con información sobre el número de páginas

Haciendo uso de esa información (para no salirnos del rango) rellenaremos la página inicial y final que queremos extraer. Hay que tener en cuenta que una cantidad máxima de 2000 páginas puede provocar que el API deje de responder, además de que el tiempo será muy elevado: una búsqueda de 2000 páginas tarda alrededor de 2 horas.

Completamos también el nombre del fichero donde queremos guardar la extracción (sin ruta. Siempre se utilizará la carpeta "DatosExtraccion") y pulsamos el botón "Realizar Extracción". La aplicación nos mostrará dos mensajes: uno de comienzo y otro de fin:

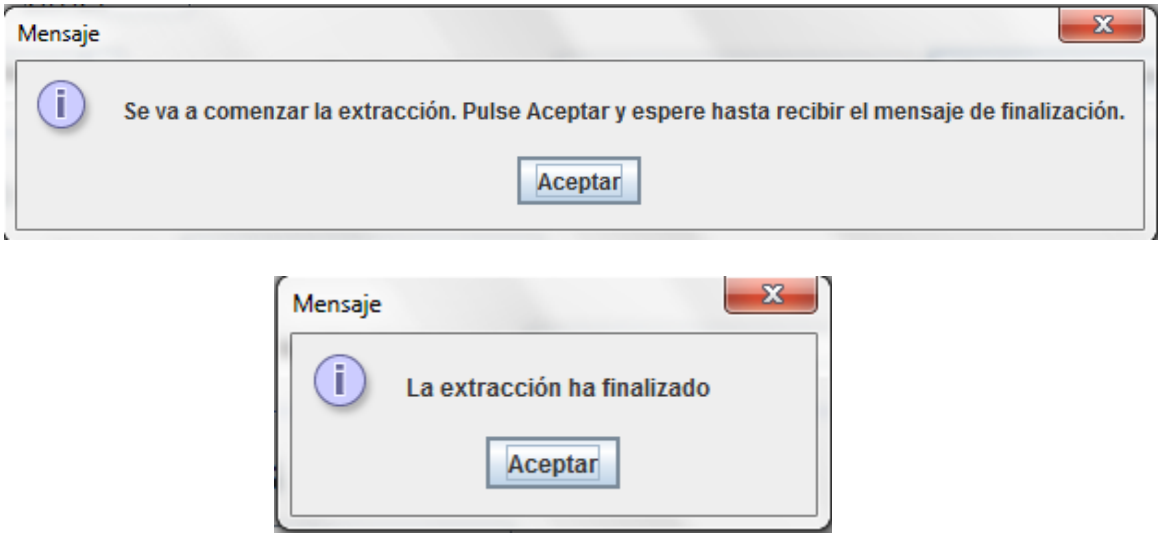


Figura 43. Manual de Usuario. Mensajes informativos sobre la extracción

El fichero se habrá creado dentro de la carpeta "DatosExtraccion" y contendrá la información sobre las viviendas que se incluían en las páginas seleccionadas.

Para completar la extracción, debido a que no se podrán en muchas ocasiones realizar las llamadas a todas las páginas de una misma vez, la aplicación provee al usuario de una funcionalidad para unir varios ficheros extraídos anteriormente. Completamos los nombres de ambos ficheros de entrada (Fichero 1 y Fichero 2) y el fichero donde se quiere guardar la unión de ambos y pulsamos el botón "Unir".

Nos avisará cuando se haya terminado la unión y podremos comprobar que el fichero de resultado se ha creado en la carpeta.

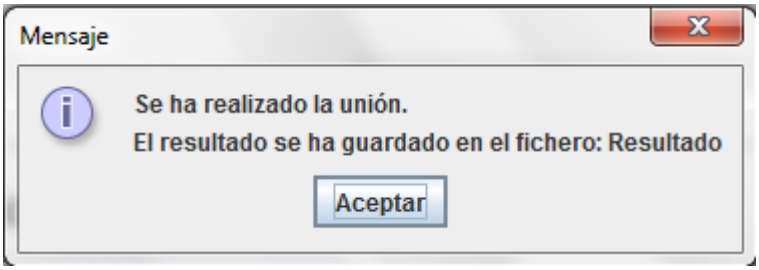


Figura 44. Manual de Usuario. Mensaje informativo sobre la finalización de la unión

Una vez que el usuario haya obtenido el fichero final que quiere utilizar como datos de entrenamiento tendrá que copiarlo en la carpeta "Datos" con el nombre "DatosIdealista.txt" y eliminar el fichero "Arbol.txt" de dicha carpeta si se ha creado con anterioridad.

Al ejecutar ProyectoIdealista.jar, tras unos minutos que requiere la herramienta para cargarse correctamente, se mostrará el siguiente formulario de selección de datos:

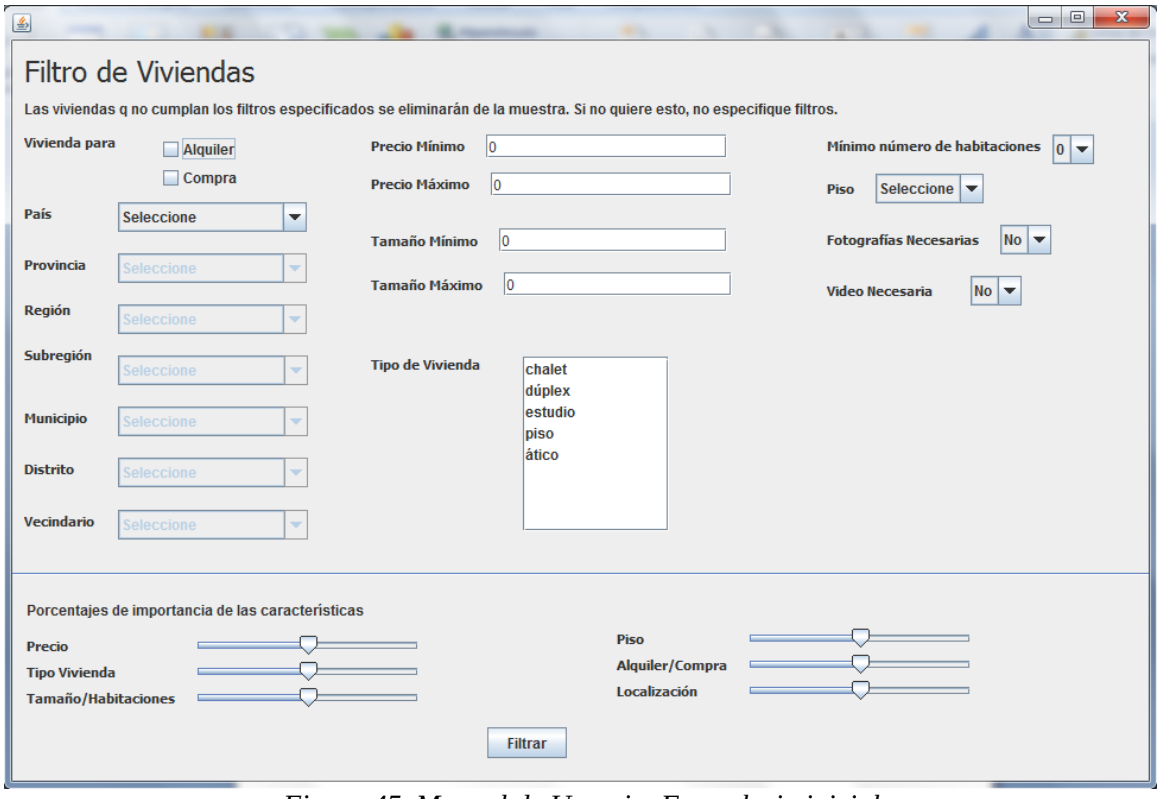


Figura 45. Manual de Usuario. Formulario inicial

En él se diferencian dos partes.

La parte superior presenta todas las características que describen a las viviendas, en la que tendremos que seleccionar valores únicamente en las características por las que se quiera filtrar. Si no se quiere aplicar ningún filtro a los datos, no se realizará ninguna modificación.

La parte inferior se utiliza para especificar la importancia que tienen esas características para el usuario; de tal forma que si el cursor se pone muy a la izquierda será poco (o nada) importante, mientras que si se pone muy a la derecha será muy importante para el usuario, por lo que se tendrá muy en cuenta a la hora de tratar los datos en el futuro.

Una vez que el usuario hace la selección deseada, pulsa el botón filtrar para que la herramienta, de forma opaca al usuario, realice las acciones necesarias para el entrenamiento del mapa y acabe mostrando al usuario la siguiente pantalla:

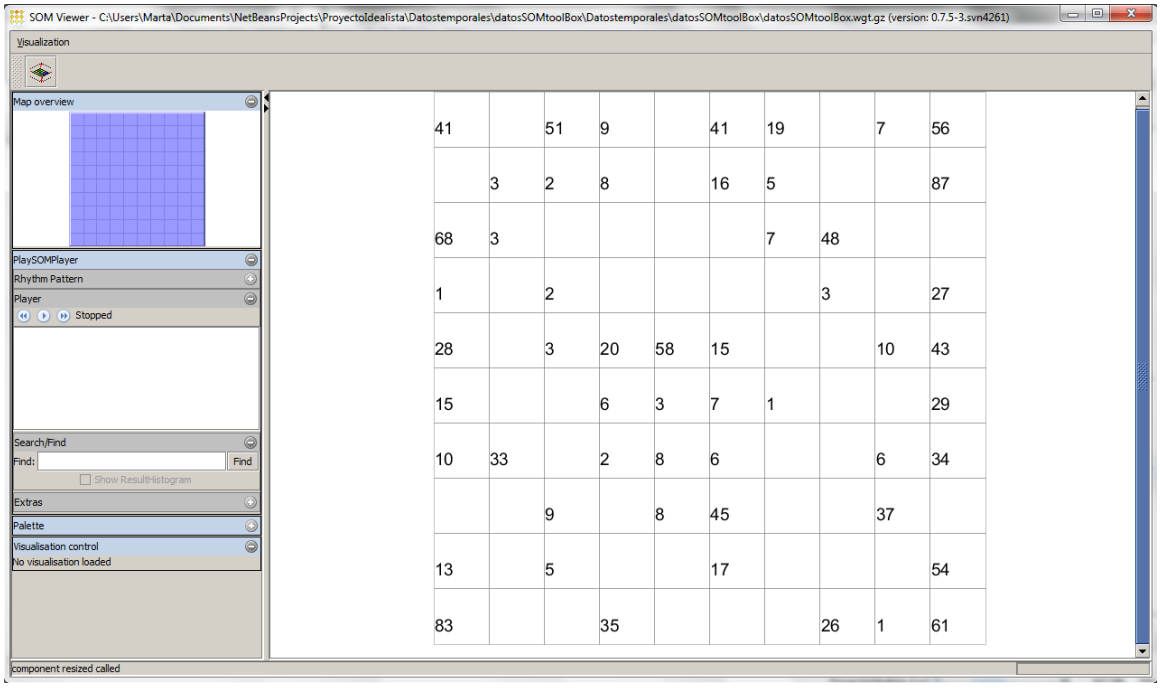


Figura 46. Manual de Usuario. Resultados

En la parte derecha se puede observar el mapa resultado, con las viviendas divididas en los distintos nodos. Para acercarse a algún nodo, se puede hacer doble click en él o utilizar la rueda del ratón.

En cada nodo, se observan las distintas viviendas que lo forman y las características de estas al pasar el ratón por encima:

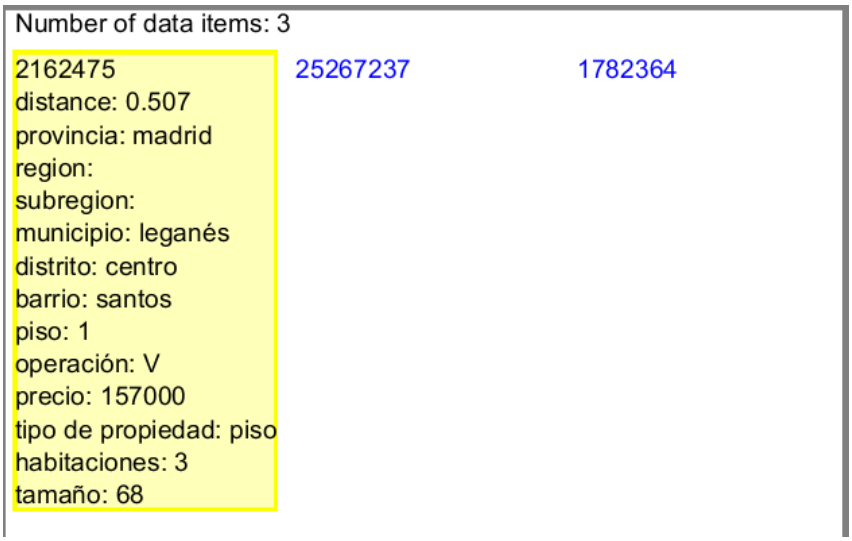


Figura 47. Manual de Usuario. Resultados en un nodo

Si el usuario ve alguna vivienda en la que esté interesado, podrá hacer doble clic sobre ella para acceder a la página web de idealista con todos los detalles, fotos, etc, que se abrirá en su navegador por defecto.

Para alejarse del nodo y volver a ver el mapa completa, el usuario puede utilizar este botón: 

En el desplegable "Visualization" el usuario puede seleccionar una visualización distinta a la básica de entre las siguientes opciones:

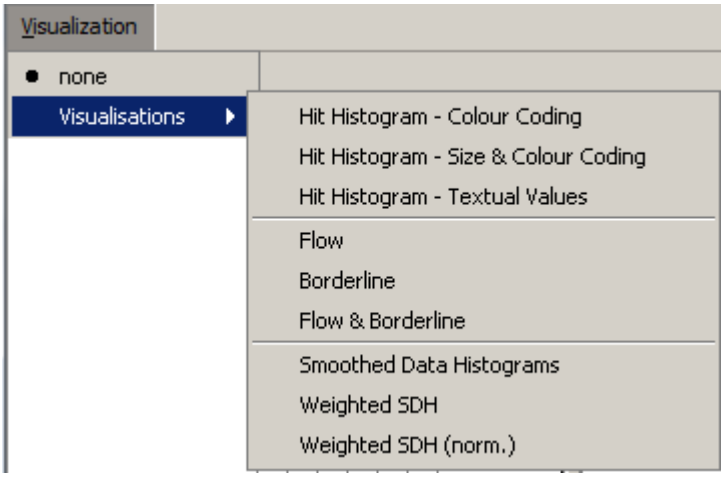


Figura 48. Manual de Usuario. Menú Visualization

Hit Histogram

Se colorean con tonos más oscuros los nodos más poblados, quedando más claros los menos poblados.

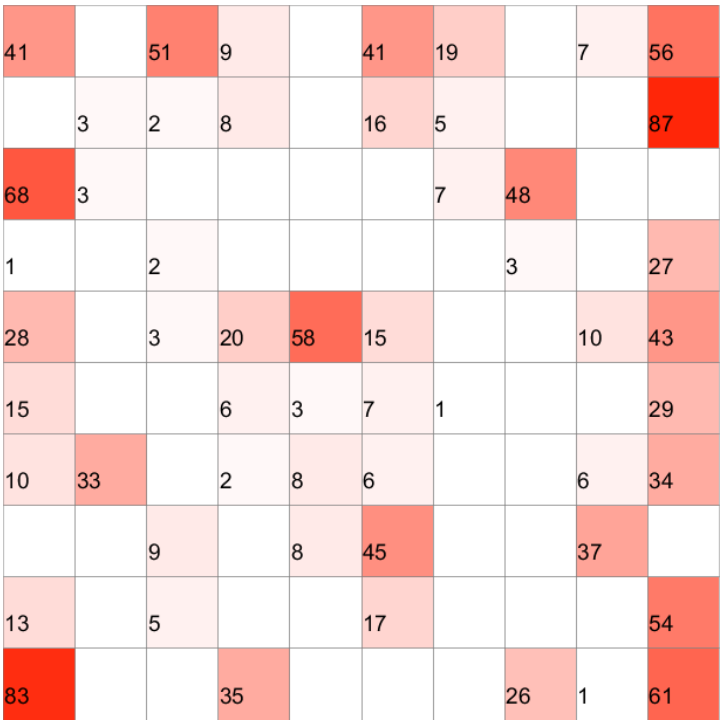


Figura 49. Manual de Usuario. Hit Histogram

Smoothed Data Histogram

Se colorean como un mapa geográfico, representando como montañas (tonos blancos y verdes) las zonas más pobladas y en azul las zonas menos pobladas.

El usuario puede modificar el parámetro *Smoothing factor* con el que se calcula el grado de pertenencia de los nodos a cada clúster, o conjunto de nodos; es decir, cuanto mayor sea este parámetro, menos clústers se formarán.

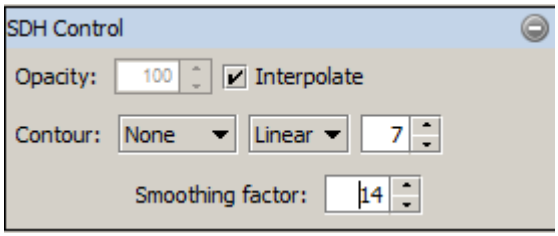


Figura 50. Manual de Usuario. Smoothed Data Histogram. Modificación de parámetros

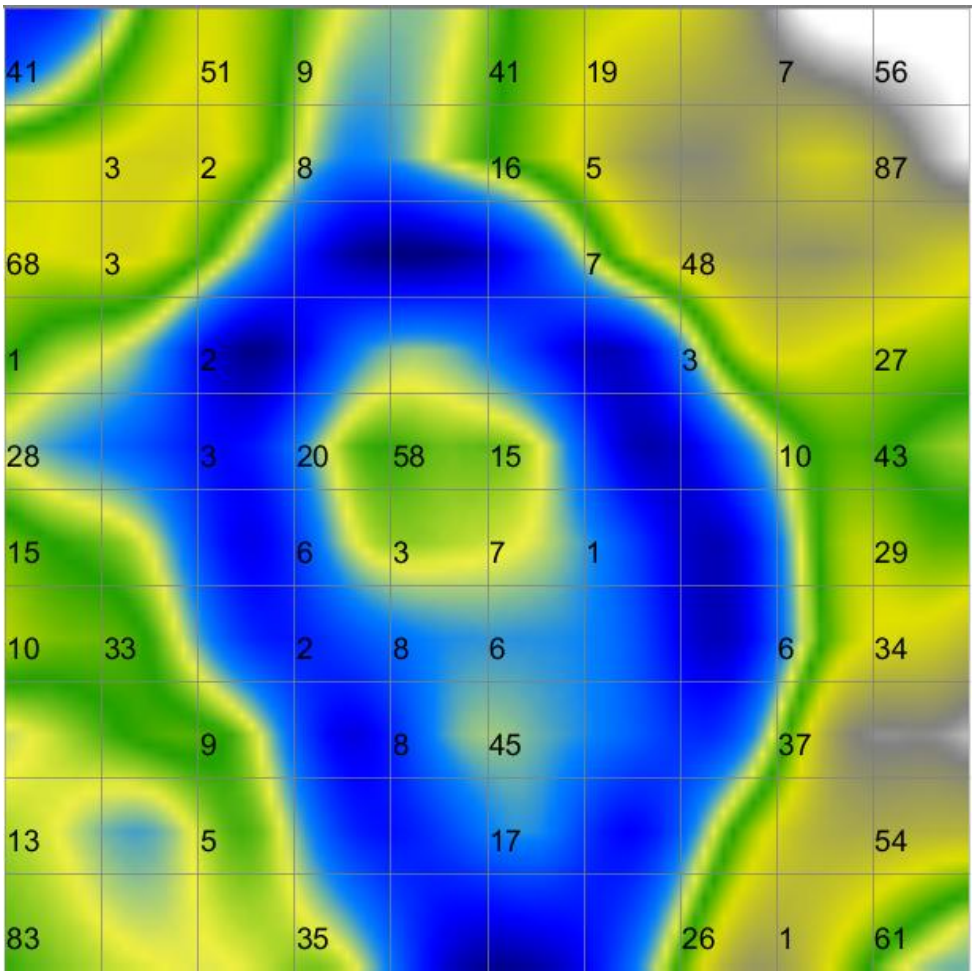


Figura 51. Manual de Usuario. Smoothed Data Histogram

Flow & Borderline

En cada nodo se muestra una flecha y una barra, ambas representan lo mismo, pero con distinta visualización. Se pueden mostrar también de forma separada.

Las flechas señalan la zona cuya distancia (en términos de diferencia entre las características del nodo se refiere) es menor, es decir, hacia sus nodos más parecidos,

mientras que la línea crea una pared de lo que acabará formando los clústers que englobarán los nodos parecidos.

El usuario puede modificar el parámetro *Sigma* que indica el número de vecinos que mirará el nodo para calcular la dirección y longitud de su flecha.

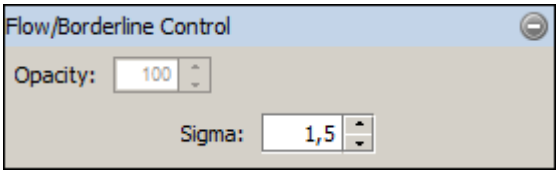


Figura 52. Manual de Usuario. Flow & Borderline. Modificación de parámetros

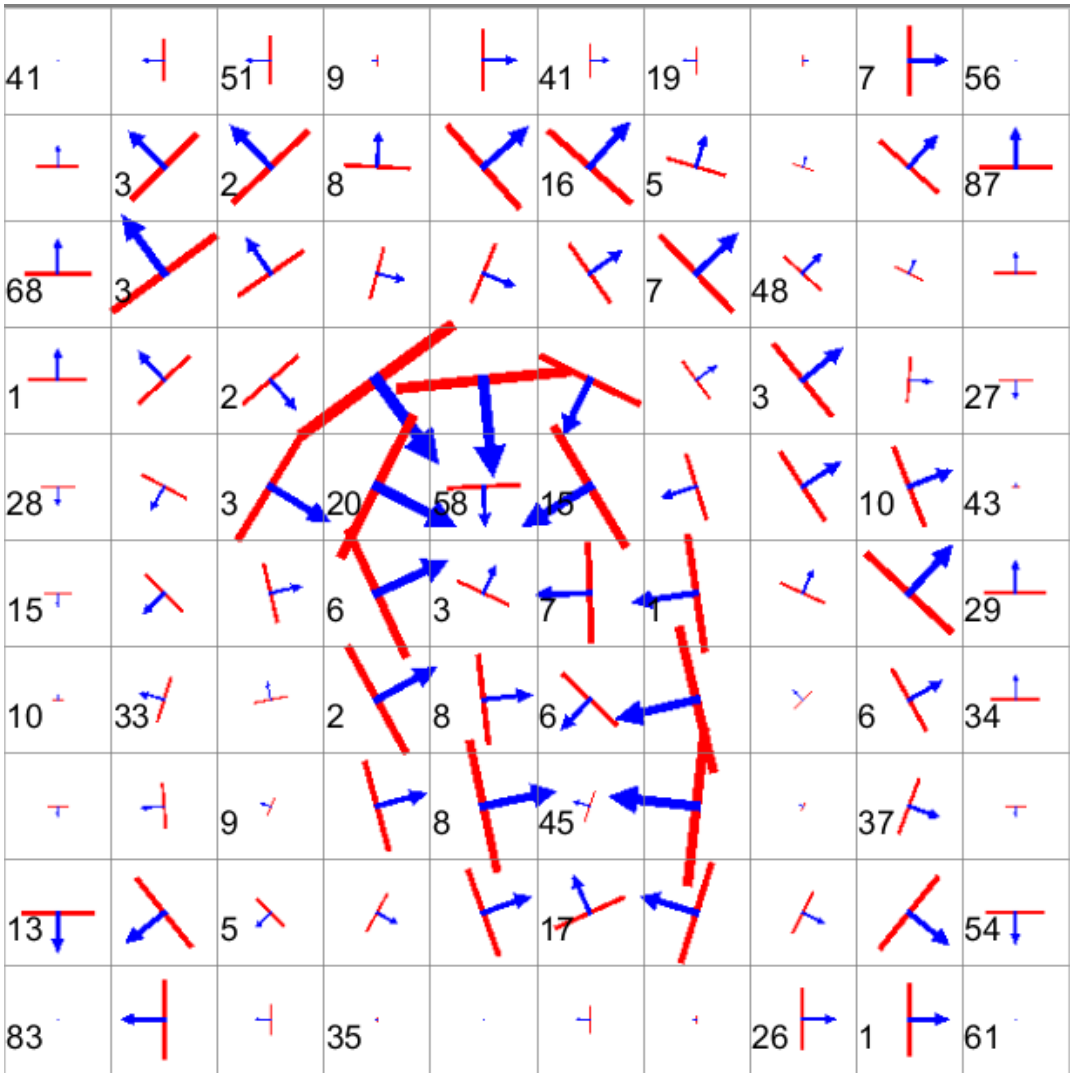
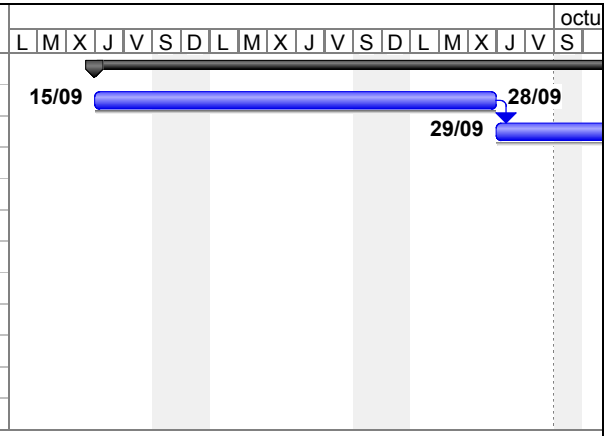


Figura 53. Manual de Usuario. Flow & Borderline

Planificación: Diagrama de Gantt

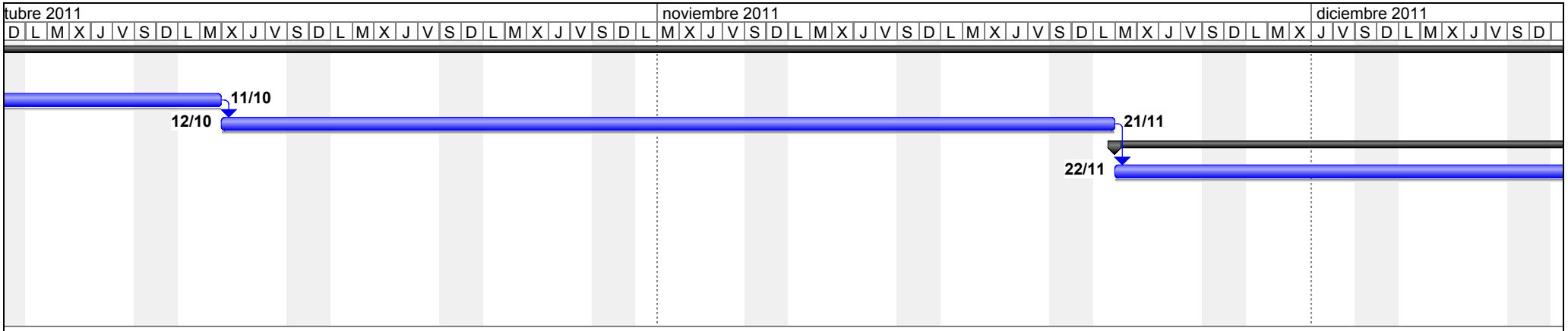
En las siguientes páginas se puede ver el diagrama de Gantt generado por la herramienta Microsoft Proyect en el que se detalla la planificación que se ha llevado a cabo a lo largo del proyecto.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Proyecto Idealista**	**313 días**	**jue 15/09/11**	**lun 26/11/12**	
2	Análisis del problema	10 días	jue 15/09/11	mié 28/09/11	
3	Estudio del API de Idealista	9 días	jue 29/09/11	mar 11/10/11	2
4	Estudio de la herramienta SOMToolBox	29 días	mié 12/10/11	lun 21/11/11	3
5	**Desarrollo**	**212 días**	**mar 22/11/11**	**mié 12/09/12**	
6	Extracción de los Datos	15 días	mar 22/11/11	lun 12/12/11	4
7	Creación de la Interfaz	23 días	jue 02/02/12	lun 05/03/12	6FC+37 días
8	Integración WEKA	10 días	mar 06/03/12	lun 19/03/12	7
9	Integración SOMToolBox	40 días	jue 07/06/12	mié 01/08/12	8FC+57 días
10	Visualización	30 días	jue 02/08/12	mié 12/09/12	9
11	Memoria	48 días	jue 13/09/12	lun 19/11/12	10
12	Presentación	5 días	mar 20/11/12	lun 26/11/12	



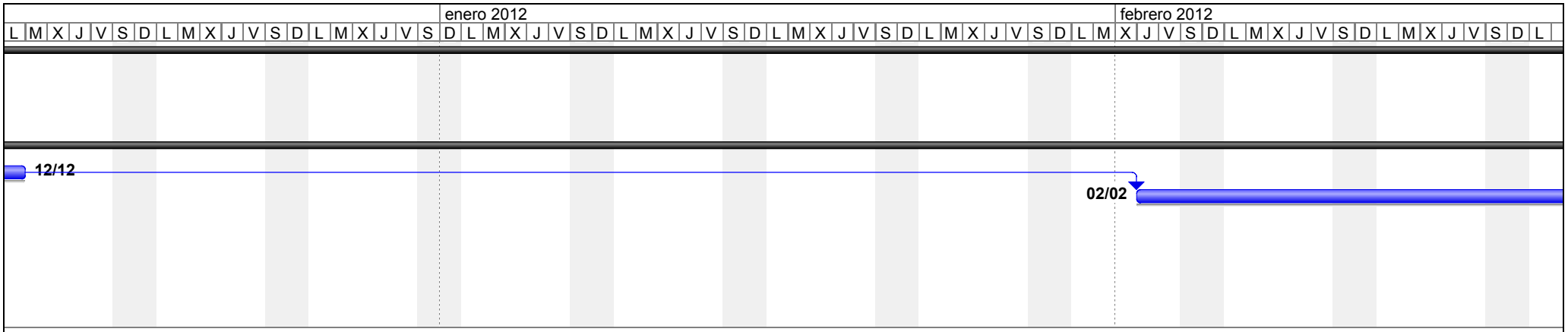
Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea	Tarea resumida	Tareas externas	
Progreso	Hito resumido	Resumen del proyecto	
Hito	Progreso resumido	Agrupar por síntesis	
Resumen	División	Fecha límite	



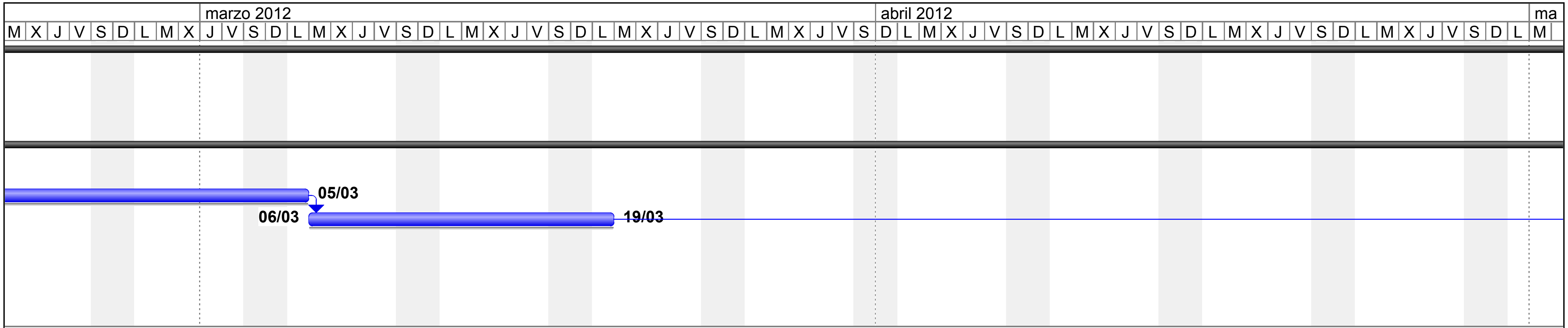
Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea		Tarea resumida		Tareas externas	
Progreso		Hito resumido	◇	Resumen del proyecto	
Hito	◆	Progreso resumido		Agrupar por síntesis	
Resumen		División		Fecha límite	⇩



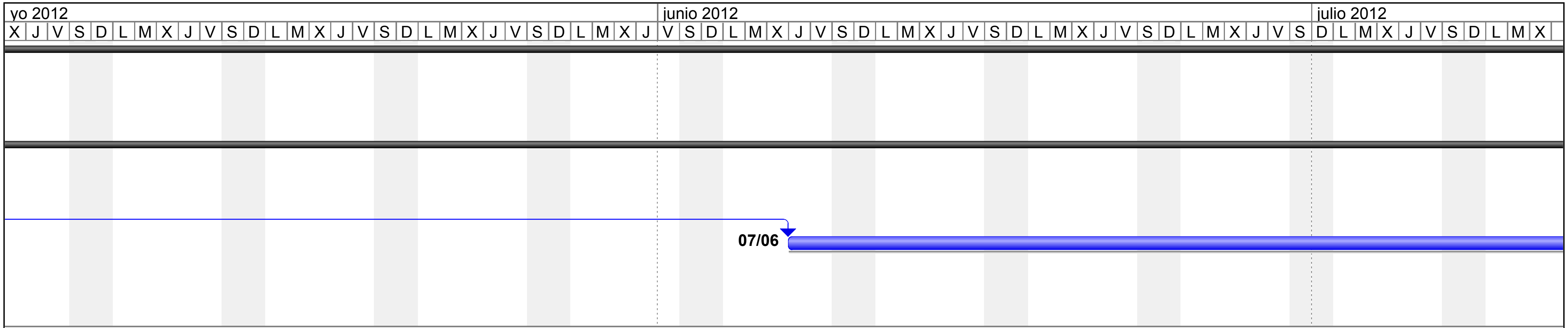
Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea		Tarea resumida		Tareas externas	
Progreso		Hito resumido	◇	Resumen del proyecto	
Hito	◆	Progreso resumido		Agrupar por síntesis	
Resumen		División		Fecha límite	⇩



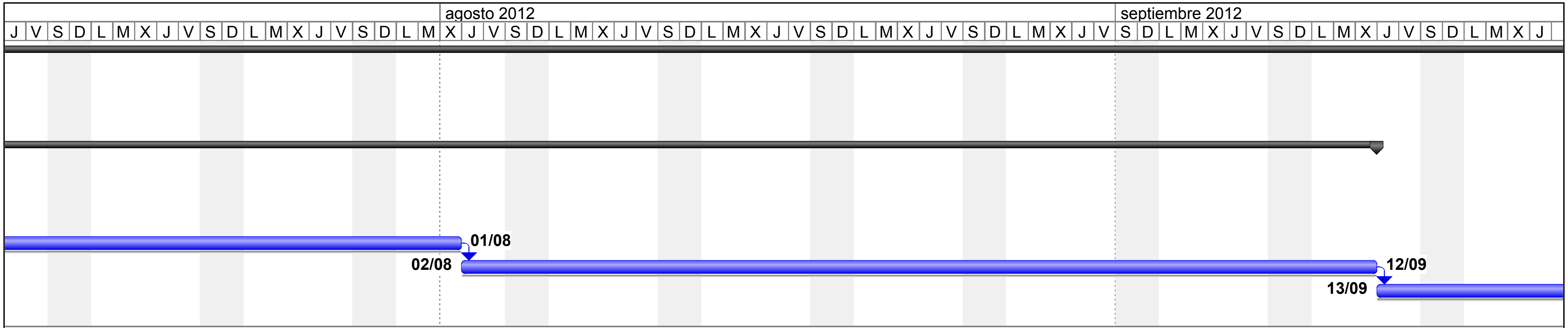
Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea		Tarea resumida		Tareas externas	
Progreso		Hito resumido		Resumen del proyecto	
Hito		Progreso resumido		Agrupar por síntesis	
Resumen		División		Fecha límite	



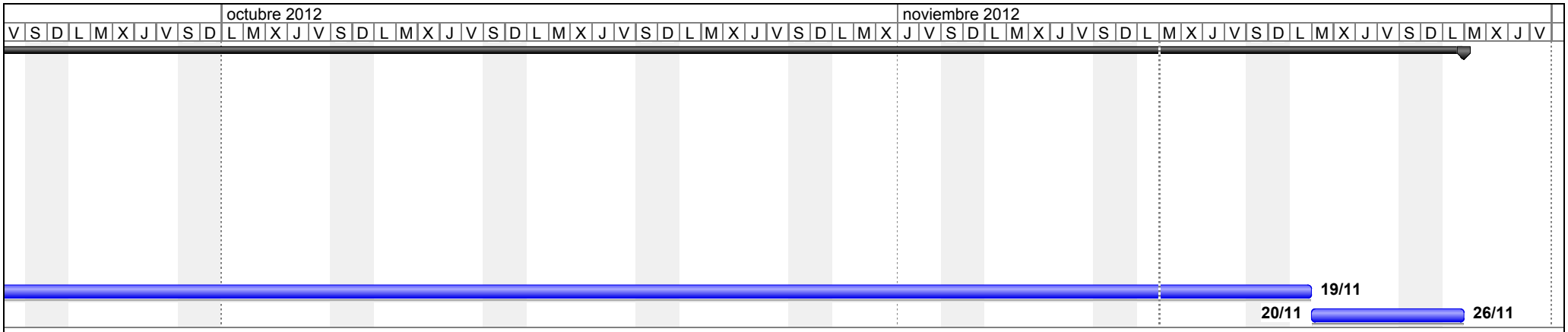
Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea	Tarea resumida	Tareas externas
Progreso	Hito resumido	Resumen del proyecto
Hito	Progreso resumido	Agrupar por síntesis
Resumen	División	Fecha límite



Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea	Tarea resumida	Tareas externas
Progreso	Hito resumido	Resumen del proyecto
Hito	Progreso resumido	Agrupar por síntesis
Resumen	División	Fecha límite



Proyecto: Planificacion
Fecha: mar 13/11/12

Tarea		Tarea resumida		Tareas externas	
Progreso		Hito resumido	◇	Resumen del proyecto	
Hito	◆	Progreso resumido		Agrupar por síntesis	
Resumen		División		Fecha límite	⇩