# Capítulo 2: Análisis y diseño del sistema para las recomendaciones en las plataformas del proyecto z17.

**Introducción**

En este capítulo, a partir del estudio de los procesos del negocio, sus descripciones y su modelado, se describe el sistema a desarrollar. Se obtienen los artefactos relacionados a la ingeniería de software aplicada a la propuesta de solución tomando como punto de partida el problema de investigación. Además, se plasman los requisitos funcionales y no funcionales de la propuesta, así como los diferentes artefactos relacionados con la metodología de desarrollo

## 2.1 Descripción de la Propuesta de Solución.

A continuación se hará una detallada explicación de cómo está compuesto el sistema propuesto. Para ello el autor se basó en las etapas que lo forman y los modelos que lo integran.

### **2.1.1 Etapas.**

El sistema está compuesto por **3 etapas** fundamentales, **Recuperación, Clasificación y Re-clasificación**. Estas etapas están conectadas entre sí de manera secuencias respectivamente donde la salida de una es la entrada de la otra siendo la última etapa la encargada de devolver los datos a recomendar y la primera la encargada de obtener el corpus de elementos candidatos inicial. A continuación cada una de las etapas:

* **Recuperación**: Esta etapa es responsable de seleccionar un conjunto inicial de cientos o miles de candidatos entre todos los posibles candidatos que pueden ser millones. El objetivo principal de esta etapa es eliminar de manera eficiente a todos los candidatos que no les interesan al usuario. Dado que se puede estar tratando con millones de candidatos, tiene que ser computacionalmente eficiente. Esta etapa puede estar compuesta por uno o varios modelos generadores de candidatos donde cada uno se basa en los datos implícitos (véase epígrafe 1.1.1.2) almacenados de los usuarios.
* **Clasificación**: Toma los resultados del modelo de recuperación y los ajusta para seleccionar el mejor puñado de recomendaciones posibles basándose en los datos explícitos (véase el epígrafe 1.1.1.2) que tenga disponible. Su tarea es reducir el conjunto de elementos que pueden interesar al usuario a una lista corta de posibles candidatos que ronde entre los cientos. La idea de esta etapa es obtener candidatos más preciso que en la etapa anterior ya que al basarse en los datos explícitos tiene información de mayor valor sobre que le interesa o no al usuario.
* **Re-clasificación:** En la etapa final el sistema puede volver a clasificar para considerar criterios o limitaciones adicionales como pueden ser la actualidad, la diversidad y la equidad. Estos son algunos de los diversos factores que pueden ayudar a mejorar considerablemente las recomendaciones de un sistema.

**La actualidad** vela por que los elementos sean recientes y no generar candidatos antiguos.

**La diversidad** valida que los candidatos sean diversos según las preferencias del usuario, todas las recomendaciones no pueden ser igual a lo que el usuario ve, ya que esto elimina la visibilidad del contenido, algo fundamental en este tipo de sistemas.

**La equidad** evita sesgos en las recomendaciones como podrían ser sebos de clics o falsas tendencias y permite que todos los usuarios sean tratados de manera justa.

Cada una de estas etapas está compuesta por 1 o varios modelos de redes neuronales profundas dependiendo de qué tan complejos sean los datos a utilizar.

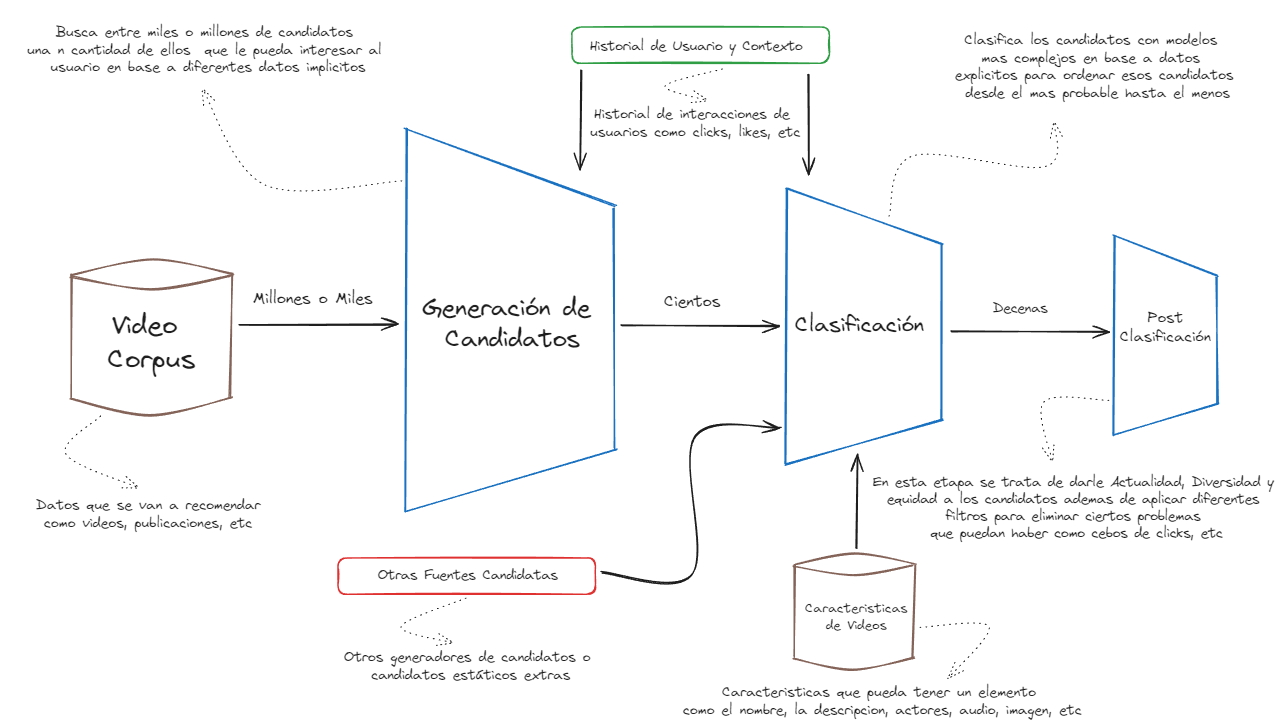


Figura 8: Arquitectura de capas del sistema.

<https://static.googleusercontent.com/media/research.google.com/es/ir/pubs/archive/45530.pdf>

<https://developers.google.com/machine-learning/recommendation?hl=es-419>

<https://www.tensorflow.org/recommenders/examples/quickstart?hl=es-419>

### **2.2.1 Modelos.**

Existen tres tipos de modelos utilizados en este sistema, Modelos de Dos Torres, Modelo de Recuperación y Modelo de Clasificación:

**Modelo de Dos Torres**: es una arquitectura de red neuronal diseñada específicamente para tareas de aprendizaje de representaciones y, en particular, para problemas de recomendación, búsqueda y clasificación. Su nombre proviene de la estructura visual que adopta: dos "torres" de redes neuronales que procesan independientemente dos tipos de datos, y luego combinan sus representaciones finales para obtener una puntuación o predicción.

En una arquitectura de dos torres, cada torre es una red neuronal que procesa características de entrada candidatas o de consulta para producir una representación integrada de esas características. Debido a que las representaciones de incrustación son simplemente vectores de la misma longitud, podemos calcular el producto escalar entre estos dos vectores para determinar qué tan cerca están. Esto significa que la orientación del espacio de incrustación está determinada por el producto escalar de cada par <query, candidato> en los ejemplos de entrenamiento.

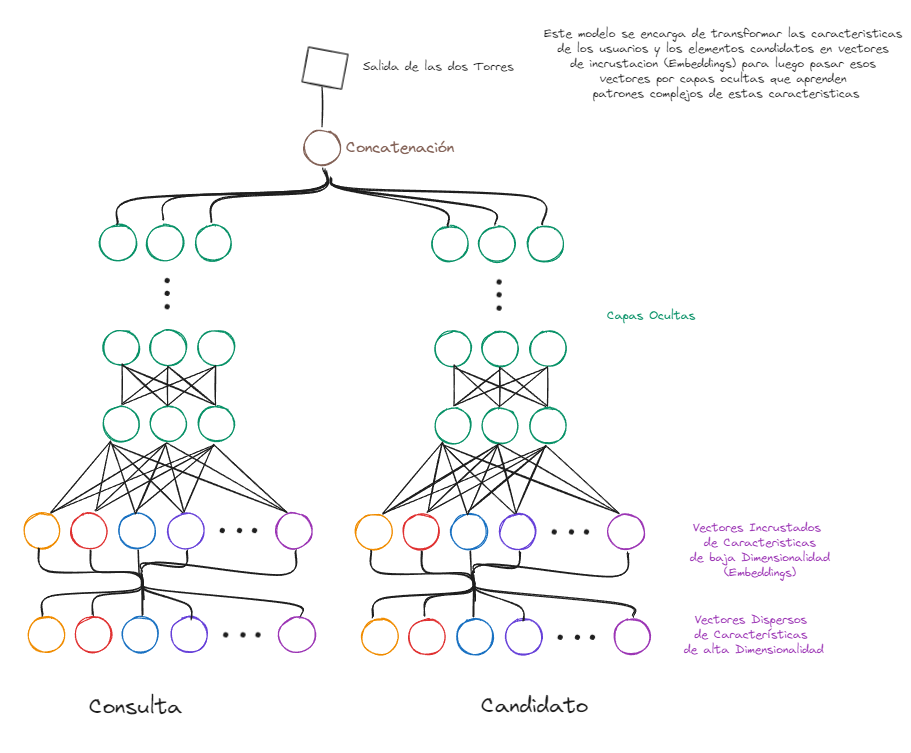


Figura 9: Arquitectura de una red neuronal de Dos Torres.

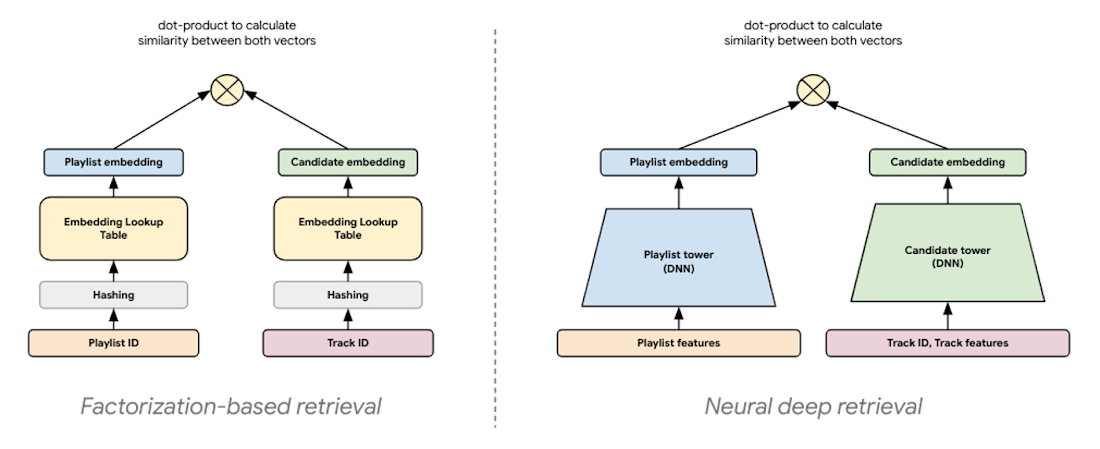


Figura 10: Las arquitecturas NDR (neural deep retrieval), como los codificadores de dos torres, son conceptualmente similares a los modelos de factorización. Ambas son técnicas de recuperación basadas en incrustaciones que calculan representaciones vectoriales de consultas y candidatos de dimensiones inferiores, donde la similitud entre estos dos vectores se determina calculando su producto escalar.

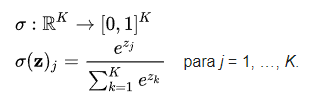
<https://cloud.google.com/blog/products/ai-machine-learning/scaling-deep-retrieval-tensorflow-two-towers-architecture>

**Modelo de Recuperación**: Es el encargado de leer el corpus de datos completo y recuperar unos miles de candidatos más probables a ser de interés para el usuario. Este tipo de modelos es entrenado con datos implícitos (véase el epígrafe 1.1.1.2) del usuario.

* **Representación**: Los modelos de recuperación a menudo se componen de dos sub modelos:
* Un modelo de consulta que calcula la representación de la consulta (normalmente un vector de incorporación de dimensionalidad fija) mediante funciones de consulta.
* Un modelo candidato que calcula la representación candidata (un vector de igual tamaño) utilizando las características candidatas.

Estos dos sub modelos son representados con el modelo de Dos Torres explicado anteriormente.

* **Función de activación de la capa de salida**: Se utiliza una capa de salida con función de activación Softmax <https://es.wikipedia.org/wiki/Funci%C3%B3n_SoftMax> para calcular la probabilidad de interés de un usuario hacia cada candidato. A continuación la función softmax.



* **Métrica y función de coste**: Para medir la precisión se utiliza una métrica llamada **FactorizedTopK** específica diseñada para evaluar el desempeño de modelos de recomendación factorizados. Esta métrica se enfoca en medir la capacidad del modelo para predecir correctamente los elementos (por ejemplo, productos, películas) más relevantes para un usuario dado en una lista ordenada de recomendaciones.

Recibe la lista de candidatos donde para cada consulta (usuario) se obtiene la lista de candidatos predichos por el modelo y se compara esta lista con la lista real de candidatos seleccionados por el usuario. Luego se calcula la precisión para diferentes valores de k (generalmente 1, 5, 10, 50, 100). La precisión se define como el porcentaje de candidatos predichos que coinciden con los candidatos reales.

* **Servicio**: Este modelo una vez entrenado es explotado para construir un servicio eficiente mediante la construcción de un índice aproximado de vecinos más cercanos (ANN) <https://es.wikipedia.org/wiki/K_vecinos_m%C3%A1s_pr%C3%B3ximos> el cual se utiliza en producción para obtener los n candidatos en la etapa de recuperación.

El sistema de recomendación puede tener varios generadores de candidatos que usen diferentes fuentes, como de la siguiente manera:

* Elementos relacionados de un modelo de factorización de matrices.
* Funciones del usuario que tienen en cuenta la personalización.
* “Local” y "distante" items; es decir, que toman información geográfica a tener en cuenta.
* Artículos populares o en tendencia.
* Un gráfico social; es decir, los elementos recomendados por amigos.

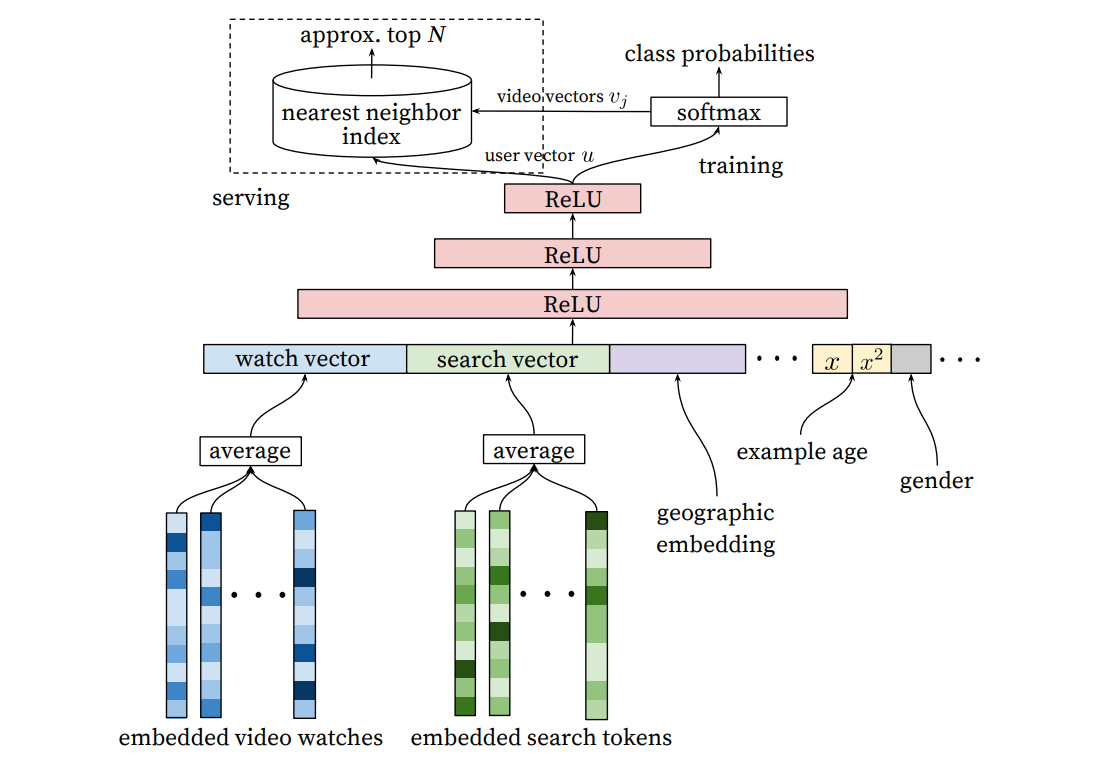


Figura 11: Arquitectura de un Modelo de Recuperación.

**Modelo de Clasificación:** Después de la generación del candidato, otro modelo califica y clasifica los candidatos para seleccionar el conjunto de elementos que se mostrarán.

El sistema combina estas diferentes fuentes en un grupo común de candidatos que se califican con un solo modelo y se clasifican según esa puntuación. Por ejemplo, el sistema puede entrenar un modelo para que prediga probabilidad de que un usuario mire un video en YouTube a partir de los datos explícitos del mismo como podrían ser los likes y teniendo en cuenta los siguientes factores:

Funciones de búsqueda (por ejemplo, historial de reproducciones del usuario, idioma, país y hora).

Elementos de video (por ejemplo, título, etiquetas, incorporación de video, duración).

* **Representación**: Este modelo utiliza la misma representación que el de Recuperación.
* **Función de activación de la capa de salida:** Este modelo puede variar en cuanto a la capa de salida ya que se podrían utilizar diferentes enfoques para clasificar a los candidatos dependiendo del set de datos que se pretenda utilizar.
* **Métrica y función de coste:** puede variar ya que se podrían utilizar diferentes enfoques para clasificar a los candidatos dependiendo del set de datos que se pretenda utilizar.
* **Servicio:** Se utiliza el propio modelo para hacer predicciones sobre los candidatos.



Figura 12: Arquitectura de un Modelo de Clasificación.

<https://www.tensorflow.org/recommenders?hl=es-419>

<https://developers.google.com/machine-learning/recommendation?hl=es-419>

<https://cloud.google.com/blog/products/ai-machine-learning/scaling-deep-retrieval-tensorflow-two-towers-architecture>

<https://www.youtube.com/playlist?list=PLQY2H8rRoyvy2MiyUBz5RWZr5MPFkV3qz>

<https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/45530.pdf>

Requisitos Funcionales

1. Entrenar el sistema
2. Actualizar el sistema
3. Generar las recomendaciones
4. Crear Configuraciones
5. Actualizar Configuraciones
6. Mostrar Configuraciones
7. Listar Configuraciones

**Patrones de Diseño:**

Los **patrones de diseño** son unas técnicas para resolver problemas comunes en el [desarrollo de *software*](https://es.wikipedia.org/wiki/Desarrollo_de_software) y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

[**https://es.wikipedia.org/wiki/Patr%C3%B3n\_de\_dise%C3%B1o**](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o)

**Patrones GRASP**

En [diseño orientado a objetos](https://es.wikipedia.org/wiki/Dise%C3%B1o_orientado_a_objetos), **GRASP** son *patrones generales de software para asignación de responsabilidades*, es el [acrónimo](https://es.wikipedia.org/wiki/Acr%C3%B3nimo) de "GRASP (object-oriented design General Responsibility Assignment Software Patterns)". Aunque se considera que más que [patrones](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o) propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

<https://es.wikipedia.org/wiki/GRASP>

**Experto**

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo.

<https://es.wikipedia.org/wiki/GRASP>

Este patrón ha sido fuertemente utilizado en el sistema propuesto ya que permite una mejor estructura y organización de la lógica en el sistema dando como resultado clases u objetos que encapsulan su propia información para llevar a cabo sus tareas haciendo que el código sea más fácil de entender y mantener. Un ejemplo donde se ha utilizado este patrón es en los modelos de aprendizaje, cada modelo se encapsulo como una clase diferente guardando cada uno sus propias tareas y acciones.



Figura #: Clase del modelo de recuperación.

**Creador**

El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o [instanciación](https://es.wikipedia.org/wiki/Instancia_(programaci%C3%B3n))) de nuevos [objetos](https://es.wikipedia.org/wiki/Objetos_(programaci%C3%B3n_orientada_a_objetos)) o [clases](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)).

La nueva instancia deberá ser creada por la clase que:

* Tiene la información necesaria para realizar la creación del objeto.
* Usa directamente las instancias creadas del objeto.
* Almacena o maneja varias instancias de la clase
* Contiene o agrega la clase.

Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización.

[**https://es.wikipedia.org/wiki/GRASP**](https://es.wikipedia.org/wiki/GRASP)

Para implementar este patrón se han utilizado “acciones”, estas “acciones” no son más que funciones que permiten realizar funcionalidades del sistema como son train (permite entrenar el sistema), fine\_tunning (permite actualizar el sistema ya entrenado con nuevos datos) y use\_engine (permite generar recomendaciones). Estas acciones son las encargadas de implementar el patrón **creador** instanciando modelos y clases para el manejo de datos.

# Alta cohesión y bajo acoplamiento

### **Alta cohesión**

Dice que la información que almacena una [clase](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)) debe ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.

**Bajo acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la [reutilización](https://es.wikipedia.org/wiki/Reutilizaci%C3%B3n_de_c%C3%B3digo), y disminuyendo la dependencia entre las clases.

<https://es.wikipedia.org/wiki/GRASP>

En el sistema propuesto el autor ha tenido muy en cuenta los siguientes puntos basándose en estos dos patrones:

1. **Cohesión Coincidente**: El módulo realiza múltiples tareas, sin ninguna relación entre ellas.
2. **Cohesión de Comunicación**: Las tareas corresponden a una secuencia de pasos propia del “producto” y todas afectan a los mismos datos.
3. **Cohesión Funcional**: Cuando el módulo ejecuta una y sólo una tarea, teniendo un único objetivo a cumplir, se dice que tiene Cohesividad Funcional.
4. **Acoplamiento de Contenido**: Cuando un módulo referencia directamente el contenido de otro módulo. (En lenguajes de alto nivel es muy raro)
5. **Acoplamiento Común**: Cuando dos módulos acceden (y afectan) a un mismo valor global.
6. **Acoplamiento de Control**: Cuando un módulo le envía a otro un elemento de control que determina la lógica de ejecución del mismo.



Figura #: La clase DataPipeline muestra una alta cohesión, ya que todas sus responsabilidades están estrechamente relacionadas con el procesamiento y manipulación de datos.



Figura #: La clase ModelConfig es utilizada para pasar configuraciones a los modelos  lo que reduce el acoplamiento entre la configuración y la implementación del modelo.

**Patrones GOF**

**Singleton:** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.



Figura #: Creación de dos instancias globales en el sistema

**Factory Method**

Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que se crea. La clase DataPipeline actúan utiliza este patron ya que muchos de sus métodos crean objetos específicos Vease la figura (clase datapipeline arriba).

**Decorator**

Añade funcionalidad a una clase dinámicamente. Este patron es utilizado en la clase FeaturesLayers donde recibe un parámetro dinámico agregando funcionalidad a la clase.



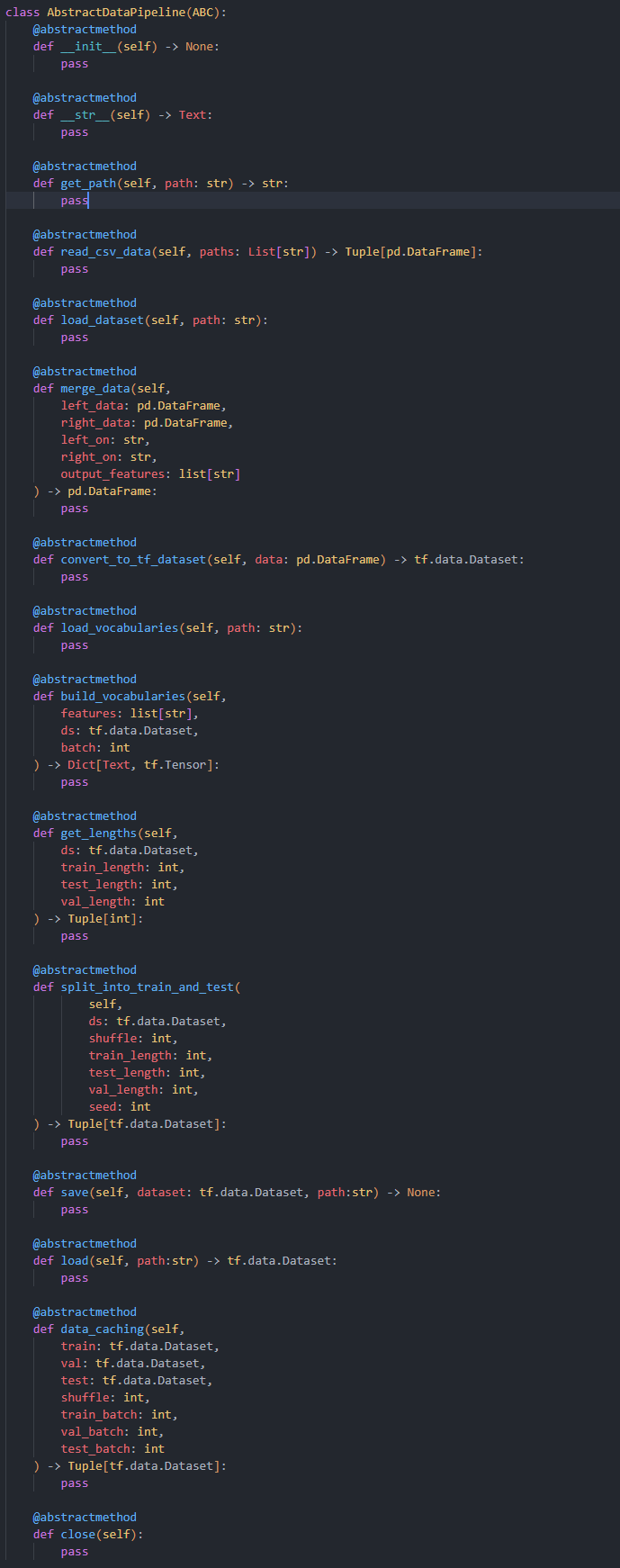
Figura #: Uso del patrón Decorator en la clase FeaturesLayers

**Facade**

Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. La clase DataPipeline utiliza este patron ya que actúa como una fachada, proporcionando una interfaz simplificada para varias operaciones complejas de procesamiento de datos. Vease la figura (clase datapipeline arriba).

**Template Method (Método plantilla)**

Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.



[**https://www.ecured.cu/Patrones\_Gof**](https://www.ecured.cu/Patrones_Gof)