



Powered by  Business IT

Nota técnica

Makers150

CONFIDENCIALIDAD

Este documento es CONFIDENCIAL. Se prohíbe su reproducción, distribución o publicación total o parcial, incluyendo su contenido: diseño gráfico, funcionalidades o recomendaciones cuya propiedad intelectual pertenece a HandytecMobi S.A. El Cliente se compromete a mantener la información confidencial en estricta reserva y no revelar ningún dato de la información a ninguna otra parte, relacionada o no, sin el consentimiento previo escrito de HandytecMobi S.A.

Documento	Modelo de Similitud – Nota Técnica
Cliente	Makers150
Autor	Isaid Valenzuela
Versión	1.0
Fecha	2025-09-30
Estado	Final

Documentos relacionados:

Documento	Formato	Ubicación
No aplica	No aplica	No aplica

Control de cambios:

Fecha	Autor	Cambios
2025-09-30	Isaid Valenzuela	Creación de versión inicial del documento



Contenido

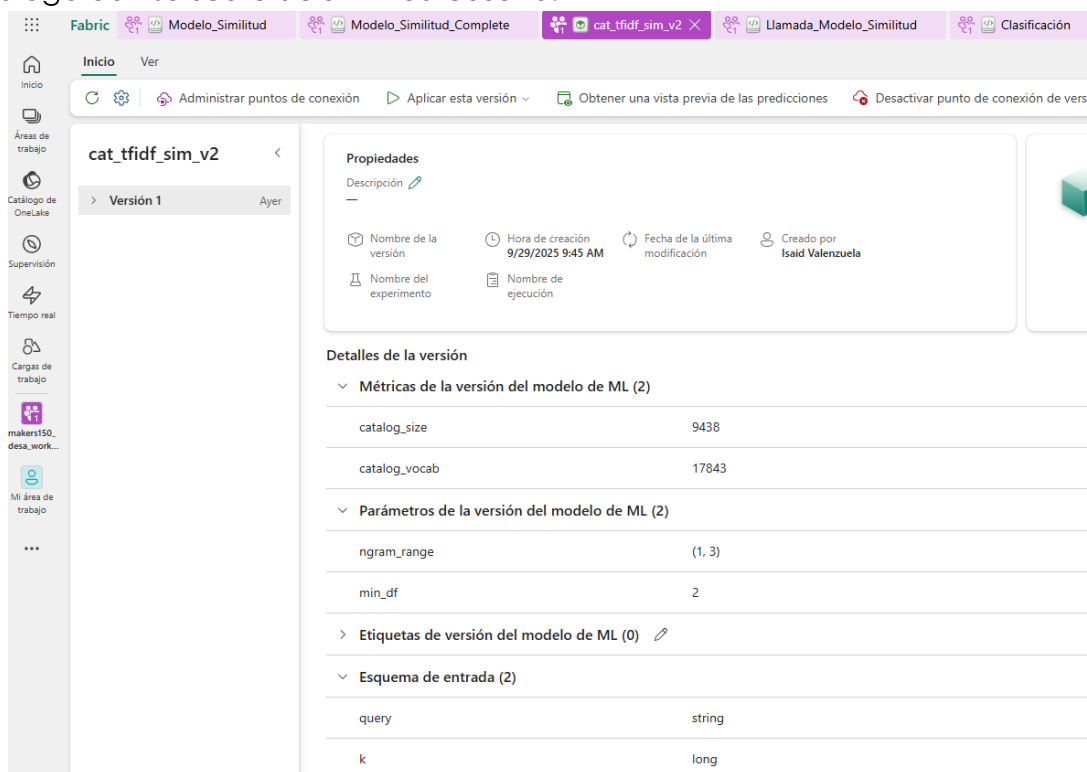
1. Generalidades	4
2. Propósito	4
3. Entradas y Salidas.....	4
3.1. Entradas	4
3.2. Salidas	5
4. Variables de Set-up y Hiperparámetros	5
5. Flujo del Código (Paso a Paso)	5
5.1. Configuración e imports.....	5
5.2. Lectura del catálogo.....	5
5.3. Entrenamiento y artefactos	6
5.4. Definición de modelo PyFunc	6
5.5. Signature e input_example	7
5.6. Log y registro en MLflow	7
6. Interfaz del Modelo (Contrato de Entrada/Salida).....	7
6.1. Entrada (pandas DataFrame)	7
6.2. Salida.....	7
7. Consumo como Endpoint (Fabric).....	7
7.1. Llamada (payload JSON)	8
7.2. Respuesta típica	8
8. Buenas Prácticas	9

1. Generalidades

En este documento se describe el modelo de búsqueda de similitud.

2. Propósito

Este documento explica el cuaderno **Modelo_Similitud_Complete.ipynb**, cuyo objetivo es **entrenar y publicar** un modelo de similitud basado en **TF-IDF** para buscar productos del **catálogo** más parecidos a consultas de texto ("queries"). El resultado es un **modelo PyFunc** registrado en **MLflow Model Registry** (servible como endpoint en Fabric) que, dado un DataFrame de consultas, devuelve los **Top-K** productos del catálogo con su **score de similitud coseno**.



The screenshot shows the MLflow Model Registry interface. The left sidebar contains navigation options like 'Inicio', 'Ver', 'Administrar puntos de conexión', 'Aplicar esta versión', 'Obtener una vista previa de las predicciones', and 'Desactivar punto de conexión de vers'. The main panel displays the details for the model version 'cat_tfidf_sim_v2'.

Propiedades

- Descripción
- Nombre de la versión: 9/29/2025 9:45 AM
- Fecha de la última modificación
- Creado por: Isaid Valenzuela
- Nombre del experimento
- Nombre de ejecución

Detalles de la versión

- Métricas de la versión del modelo de ML (2)**

catalog_size	9438
catalog_vocab	17843
- Parámetros de la versión del modelo de ML (2)**

ngram_range	(1, 3)
min_df	2
- Etiquetas de versión del modelo de ML (0)**
- Esquema de entrada (2)**

query	string
k	long

3. Entradas y Salidas

3.1. Entradas

- Tabla del catálogo (Lakehouse / Delta):**
 - Se lee con Spark y se pasa a pandas: columnas usadas: name, name_clean.

- En el ejemplo del notebook:
de_lh_200_makers150_bizner.catalog_products_vectors (se seleccionan name, name_clean).

3.2. Salidas

- **Artefactos locales (carpeta artifacts/):**
 - vectorizer.joblib → TfidfVectorizer entrenado con el catálogo.
 - X_catalog.npz → matriz TF-IDF esparsa del catálogo (scipy.sparse CSR).
 - items.csv → catálogo con índice estable idx, name, name_clean.
- **MLflow**
 - **Run** con parámetros y métricas (ngram_range, min_df, catalog_size, catalog_vocab).
 - **Modelo PyFunc** registrado (artifact model) con **signature** e **input_example** para servirlo.
 - Registro en **Model Registry** con nombre (cat_tfidf_sim_v2).

4. Variables de Set-up y Hiperparámetros

Variable	Descripción	Ejemplo/Default
V_NGRAM	Rango de n-gramas usados por TF-IDF.	(1, 3)
V_MIN_DF	Frecuencia mínima de documento para incluir términos.	2
TOP_K_DEFAULT	Número de resultados por query si no se especifica k.	10
MLFLOW_EXPERIMENT	Nombre del experimento MLflow (opcional).	"Modelo_Similitud_Complete"
REGISTERED_MODEL	Nombre para registrar el modelo en el Registry.	"cat-tfidf-sim-v2"

Notas - Elevar V_MIN_DF reduce ruido y tamaño del vocabulario; aumentar V_NGRAM (e.g. hasta 3-gramas) ayuda a captar frases compuestas. - TOP_K_DEFAULT controla el tamaño de la lista de sugerencias.

5. Flujo del Código (Paso a Paso)

5.1. Configuración e imports

- Importa numpy, pandas, scipy.sparse, utilidades de sklearn (TF-IDF, cosine), y **MLflow**.
- Fija el experimento de MLflow.

5.2. Lectura del catálogo

```
catalog_dfs = spark.read.table("<CATALOGO>") \
    .select("name", "name_clean") \
    .dropna() \
    .dropDuplicates()
```

```
catalog_df = catalog_dfs.toPandas()
```

- Se prepara un DataFrame **pandas** con name/name_clean.

5.3. Entrenamiento y artefactos

```
vec = TfidfVectorizer(ngram_range=V_NGRAM, min_df=V_MIN_DF)
```

```
X_catalog = vec.fit_transform(catalog_df["name_clean"].astype(str).values)
```

- **Fit** del vectorizador con el catálogo; se obtiene la matriz **TF-IDF** X_catalog (CSR, esparsa).
- Persistencia de artefactos:
 - `joblib.dump(vec, "artifacts/vectorizer.joblib")`
 - `sp.save_npz("artifacts/X_catalog.npz", X_catalog)`
 - `items = catalog_df[["name", "name_clean"]].reset_index().rename({'index':'idx'}, axis=1)`

5.4. Definición de modelo PyFunc

Se crea una clase CatalogTFIDFModel(mlflow.pyfunc.PythonModel) que implementa:

- **load_context(self, context)**
 - Carga vectorizer (joblib), X_catalog (npz) y items (csv) desde los **artifacts** del run.
 - Enforce de tipos:
 - `items["idx"] → int64`
 - `items["name"], items["name_clean"] → string`
- **_score_one(self, query: str, k: int) -> pd.DataFrame**
 - `q_vec = self.vectorizer.transform([str(query)]) → vector TF-IDF de la consulta.`
 - `sims = cosine_similarity(q_vec, self.X_catalog).ravel() → vector de similitudes coseno vs catálogo.`
 - Selección **Top-K** con `argpartition` + ordenamiento final.
 - Devuelve DataFrame con columnas: `idx`, `name`, `name_clean`, `score` (float64), ya ordenado.
- **predict(self, context, model_input: pd.DataFrame) -> pd.DataFrame**
 - **Entrada esperada:** DataFrame con columnas:
 - `query` (string, obligatoria)
 - `k` (int, opcional)

- Normaliza tipos: query a str, k a int (con default TOP_K_DEFAULT).
- Para cada fila, llama a `_score_one` y concatena los resultados, **prependiendo** la columna query.
- **Salida:** DataFrame con columnas ['query', 'idx', 'name', 'name_clean', 'score'] por cada query.

5.5. Signature e input_example

- Se construye un `input_example` válido y se infiere signature con `infer_signature(input_example, output_example)` para evitar errores de esquema al servir.

5.6. Log y registro en MLflow

- `mlflow.pyfunc.log_model(artifact_path="model", python_model=CatalogTFIDFModel(), artifacts={...}, signature=..., input_example=..., pip_requirements=[...])`
- Se imprime el `run_id` y el `model_uri` (`runs:/<run_id>/model`).
- `mlflow.register_model(model_uri, name="cat_tfidf_sim_v2")` para publicarlo en el **Model Registry**.

6. Interfaz del Modelo (Contrato de Entrada/Salida)

6.1. Entrada (pandas DataFrame)

Columna	Tipo	Obligatorio	Descripción
query	string	Sí	Texto de búsqueda (nombre de producto).
k	int	No	Número de resultados a devolver por consulta. Default = TOP_K_DEFAULT.

Ejemplo de input_example

```
pd.DataFrame({
    "query": pd.Series(["cable usb c 1m", "audifonos bluetooth"], dtype="string"),
    "k": pd.Series([5, 3], dtype="int64")
})
```

6.2. Salida

DataFrame concatenado por query con columnas: - query (string) - idx (int64) — índice estable del catálogo - name (string) - name_clean (string) - score (float64) — similitud coseno.

7. Consumo como Endpoint (Fabric)

7.1. Llamada (payload JSON)

Body esperado: un objeto con **inputs** que contiene **una lista** de filas (listas posicionales) en el **orden de la signature** (query, k).

```
{
  "inputs": [
    ["cable usb c 1m", 5],
    ["audifonos bluetooth", 3]
  ]
}
```

Evitar enviar objetos con claves arbitrarias (p.ej. {"dataframe_records": [...]}), ya que el endpoint validará contra la **signature** y rechazará campos extra. Código de ejemplo:

```
import requests
import json

# URL del endpoint
url = "https://api.fabric.microsoft.com/v1/workspaces/90c5a484-ec91-4525-aad7-440f3403da38/mlmodels/9fef5bb0-b9f3-4da9-8515-f904a4511d3f/endpoint/versions/1/score"

FABRIC_TOKEN = (API o FABRIC_TOKEN, se obtiene de la sesión)

headers = {"Authorization": f"Bearer {FABRIC_TOKEN}", "Content-Type": "application/json"}

payload = {
  "inputs": [
    ["cable usb c 1m", 5],
    ["cemento", 10],
    ["fierro", 3]
  ]
}

response = requests.post(url, headers=headers, data=json.dumps(payload))
```

7.2. Respuesta típica

Lista de filas con columnas query, idx, name, name_clean, score por cada consulta.

8. Buenas Prácticas

- **Normalización previa** (name_clean) coherente con el vectorizador (minúsculas, eliminar no alfanumérico, colapsar espacios).
- **Signature** e **input_example** siempre incluidos para endpoints Fabric → previene errores de esquema.
- **Versionado** en MLflow: usa alias (Staging/Production) para controlar despliegues.
- **Monitor**: registra métricas de cobertura del vocabulario (catalog_vocab) y tamaño (catalog_size).

9. Paso a producción

Los siguientes son los pasos para registrar el modelo en producción:

1. Copiar la notebook Modelo_Similitud_Complete.ipynb al workspace productivo
2. Ejecutar el notebook
3. En el experimento Modelo_Similitud_Complete registrar el modelo de ML en la opción guardar con el nombre cat_tfidf_sim_v2
4. Con el modelo guardado en el workspace, abrirlo y activar un punto de conexión del modelo.
5. Recuperar el valor https del End Point para el equipo de desarrollo.