Proyecto Fin de Máster

Máster en Big Data y Data Science

Universidad Complutense de Madrid

DescubreTuReceta: Una aplicación que usa Big Data e Inteligencia Artificial para ofrecer recomendaciones culinarias personalizadas

Septiembre 2022

Integrantes:

- Javier Pardo
- Daniel Píriz
- Pablo Reguilón
- Marcelo Renere
- Alejandro Vidal

1. INTRODUCCIÓN

Preparar recetas innovadoras suele presentar un desafío, cuando una persona quiere preparar una receta específica puede buscar en internet y encontrar múltiples alternativas, sin embargo, éstas no están personalizadas para cada cocinero.

Con Inteligencia Artificial y herramientas de Big Data se planteó personalizar recomendaciones culinarias, basadas en los productos que el usuario tiene disponibles. Se desarrolló un sistema que identifica los productos al tomar una foto de una factura de compra de un supermercado y/o con una imagen del producto, y luego se contrasta cada elemento con la lista de ingredientes de un dataset de recetas. El resultado se puede visualizar en una página web que permite consultar las recetas más recomendadas a seguir con base en el porcentaje de disponibilidad de los ingredientes, identificar los ingredientes que faltan para seguir una preparación y buscar ingredientes que pueden ser sustituidos o usados en conjunto con un producto dado.

El trabajo está orientado a mejorar la experiencia de todas aquellas personas que se enfrentan a la cocina, con la idea de que negocios relacionados, como pueden ser supermercados y restaurantes encuentren en DescubreTuReceta un aliado para ofrecer productos y servicios que beneficien a los usuarios de la plataforma. Es así que el volumen de datos que puede almacenar y tratar DescubreTuReceta se convierten en un activo esencial de un negocio sostenible y rentable.

Los resultados se pueden comprobar accediendo a la página web y tomando una foto de un ticket en el que se lean los productos en español.

2. OBJETIVOS

Objetivo General:

Desarrollar un proyecto real, que genere valor añadido y sirva como modelo de negocio, mediante la aplicación de técnicas de Big Data y Data Science para identificar ingredientes de cocina por medio de imágenes de productos y tickets y ofrecer recomendaciones culinarias personalizadas.

Objetivos específicos:

- Crear un programa que identifique productos comprados al tomar una foto de una factura de compra con elementos de las categorías de mercado y alimentación.

- Desarrollar un modelo de redes convolucionales que se aproxime a la identificación de ingredientes usados en la cocina.

- Desarrollar un algoritmo que recomiende recetas con base en una lista de productos disponibles y la experiencia del usuario.

- Desarrollar un algoritmo que compara los elementos disponibles con los requeridos en una receta y como resultado devuelva los faltantes.

- Aplicar un modelo de redes neuronales que determine, con base en la asociación de ingredientes usados en las recetas, los ingredientes similares o que se suelen usar juntos en recetas, de tal forma que se pueda ofrecer recomendaciones de productos para personalizar una preparación.

3. CRONOGRAMA

- La **fase 1** del proyecto corresponde a la **obtención y fase exploratoria de los datos**. Se compone de:

    - Búsqueda de datos.
    - Tipología de los datos encontrados.
    - Características y disposición de los datos, haciendo uso de estadísticas básicas y distribuciones.
    - Estandarización y normalización de los datos.

- La **fase 2** del proyecto consiste en la **preparación de la entrada** del clasificador y su **entrenamiento**.

    - Subdivisión del **dataframe** en grupos linealmente independientes.
    - Aplicación de data **augmentation** de las imágenes almacenadas.
    - Uso del modelo vgg16 y la api de Google Cloud-Vision.
    - Desarrollo de algoritmos y pipelines.

- La **fase 3** del proyecto es la **preparar la integración en la aplicación web**:

    - Elección de proveedor de cloud público.
    - Integración de modelos en el servidor web.
    - Preparar la seguridad de los accesos.
    - Optimización de rendimiento con enfoque en la velocidad.

4. HERRAMIENTAS

Principales utilidades empleadas:

- Python:
    - La utilización de este lenguaje de programación se fundamentó en su alta capacidad de integración con otras tecnologías necesarias para el proyecto, unida a su fortaleza en el campo de la analítica de datos.
- Fullstack Web:
    - HTML:
        - Lenguaje de marcado estándar para la estructuración de páginas web. Se ha empleado para generar el esqueleto básico de la página web.
    - CSS:
        - Lenguaje de marcado estándar para la descripción de estilos en páginas web. En el proyecto se ha utilizado mediante Bootstrap para integrar funcionalidades como “Responsive” de forma sencilla.
    - Javascript:
        - Lenguaje de programación que permite implementar funciones dinámicas en páginas web. Su utilización ha sido clave para poder procesar el vídeo y las imágenes de los usuarios web en tiempo real sin emplear excesivos recursos de servidor.

Principales librerías y aplicaciones empleadas:

- Tensorflow/Keras:
    - Se han utilizado en la elaboración los modelos generados para el proyecto. De este modo, ha sido el artífice del modelo de identificación de productos.
- NLTK:
    - Librería de procesamiento de lenguaje natural (NLP) para Python que, entre otras funcionalidades, cuenta con corpus de distintos temas. Además, ha permitido eliminar las stopwords de los textos.
- Googletrans:
    - Permite utilizar el traductor de Google en Python, pero presenta una limitación del número de palabras (10.000 palabras aproximadamente), que pueden ser traducidas cada 24 horas.
- OpenCV:
    - Se trata de una librería de amplio uso en el tratamiento de imágenes y, por esta razón, ha sido utilizada para la identificación de palabras en imágenes de tickets.
- Pandas:
    - La librería de Python más conocida para el manejo y visualización de dataframes, fue ampliamente utilizada en los primeros compases del proyecto.

- Numpy:
    - Librería de Python para operaciones rápidas en matrices, que incluyen manipulación matemática, lógica, de formas, clasificación, selección, transformadas, álgebra lineal, operaciones estadísticas.

- Matplotlib:
    - Librería de Python enfocada en el trazado gráfico y visualizaciones de datos. Se ha utilizado como dependencia secundaria de otras librerías y para la elaboración de gráficos suplementarios.
- Flask:
    - Framework enfocado en backend web, es el núcleo de la web del proyecto al relacionar todos los endpoints con los modelos obtenidos.
- Selenium:
    - Herramienta de webscrapping que fue utilizada para la extracción de imágenes de productos a fin de poder elaborar un dataset de imágenes.

Proveedores de web públicos:

- Google Cloud:
    - Los criterios para su elección se fundamentaron en la capacidad de acceso a servicios avanzados de forma gratuita, especialmente a la API de Cloud Vision para el reconocimiento de imágenes.

- Amazon Web Services:

    - Este proveedor fue el más utilizado en el desarrollo del proyecto debido a la experiencia previa con EC2. En este proveedor se encuentra alojada tanto la página web como la infraestructura relacionada con ésta: dominio (Route53), almacenamiento de archivos (S3) etc.

Herramientas de gestión:

- Git + Github:
    - Software de control de versiones que constituye un estándar, ha sido utilizado en el proyecto para manejar las modificaciones efectuadas durante el desarrollo del proyecto. De este modo, se instauró una rama productiva asociada a un pipeline de Github Actions que efectuaba despliegues directamente en el servidor web alojado en AWS.

- Microsoft Teams:
    - Se han aprovechado las ventajas otorgadas por la Universidad Complutense para gestionar toda la comunicación del equipo a través de videollamadas, eventos de calendario y grupos de trabajo.

- Dropbox:
    - Se ha utilizado esta solución de almacenamiento cloud en detrimento de otras opciones como S3 por su simplicidad de manejo y por su correcta integración con Python a través de su API.

5. Datos

Dataset de Recetas

Los datos han sido recopilados de **Kaggle** y se pueden consultar aquí. Para hacer este estudio, se ha basado en un dataset en tipo json que contiene alrededor de 20.000 recetas y las variables que contiene son:

- Título de la receta
- Calificación de la receta
- Cantidad de calorías
- Cantidad de grasa
- Cantidad de sodio
- Instrucciones de Preparación
- Tipo de cocina
- Nacionalidad de la receta
- El resto de las columnas son de tipo binario para indicar si una receta se enmarca dentro de las categorías previstas en cada columna

Dataset de Productos

El dataset utilizado proviene de **Kaggle** y se puede consultar aquí. Este contiene 534.019 productos de Carrefour, sin embargo, como no todos los productos son del tipo gastronómico lo primero que hay que hacer es filtrar obteniendo 2.779 productos únicos que pueden ser usados en recetas de cocina, cuenta con 8 columnas.

- Id
- Supermercado
- Categoría
- Nombre producto
- Precio
- Precio referencia
- Unidad referencia
- Fecha inserción

Dataset de imágenes

Para obtener las imágenes de los productos, se ha partido de la lista de productos únicos obtenida del dataset anterior para aplicar una técnica de webscrapping con Python y Selenium que consiste en realizar una búsqueda de cuatro imágenes en 'Google Imágenes' y almacenarlas en una carpeta para cada producto.

Se ha optado por esta clase de dataset, en vez de urls, para tener un mayor control sobre las propiedades y la disponibilidad de las imágenes obtenidas.

La ejecución del webscrapping se llevó a cabo en un contenedor de Docker hospedado en instancia de alta capacidad de procesamiento (c6a.xlarge) y tomó una duración de 14 horas aproximadamente para guardar cuatro imágenes por producto.

6. METODOLOGÍA

1.Obtención de datos

1) Dataset de recetas:
Se realizó una búsqueda exhaustiva en fuentes de datos abiertas como Kaggle y webs de chefs reconocidos y se decidió trabajar con el dataset de Epicurous por su variedad de recetas, que cuentan con variables como la calificación y por su estructura que facilita el tratamiento de los ingredientes. Además que al estar en inglés hay más herramientas para el Procesamiento de Lenguaje Natural (NLP).

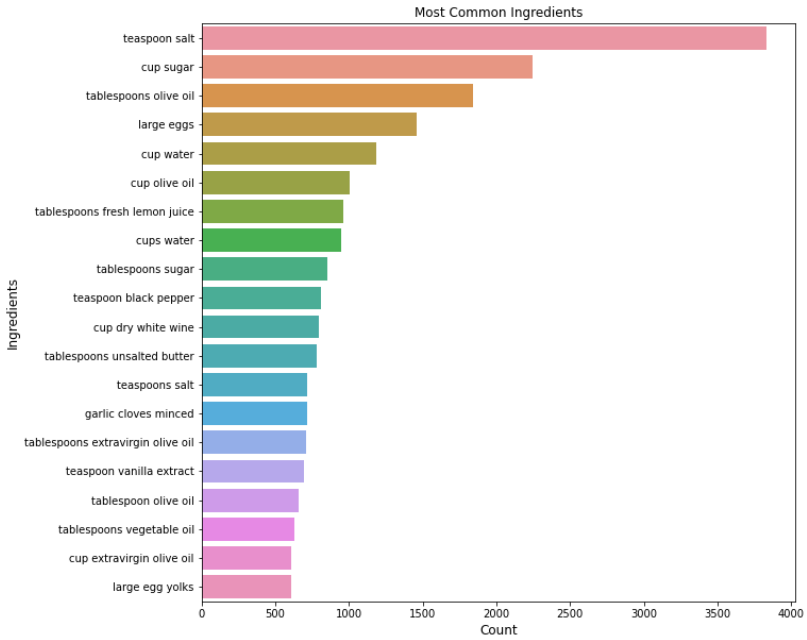
2) Dataset de productos: Después de evaluar varios datasets que incluían los productos más usados, se determinó que lo más adecuado era escoger uno con los datos de un solo supermercado y con datos de España. Además de que están relativamente actualizados (2018).

3) Dataset de imágenes:
Dada la necesidad de obtener imágenes de los productos se optó por imágenes que fueron obtenidas a través de webscrapping. La elección del webscrapping fue determinada por la compleja casuística en la que se requerían imágenes concretas de productos concretos, siendo la solución de Webscrapping la más óptima en términos de tiempo y coste de oportunidad. Durante el webscrapping se obtuvieron alrededor de 10.000 imágenes en formato png que, posteriormente, fueron limpiadas y convertidas a formato '.jpg' para reducir su tamaño sin demasiada pérdida de calidad.

2. EDA

1) Dataset de recetas:

El texto viene en inglés por lo que fue necesario traducirlo. Además, los ingredientes contienen cantidades y caracteres especiales que pueden representar un obstáculo, por lo cual fue necesario convertir a un formato estándar de UTF-8. Los ingredientes suelen estar acompañados de unidades de medida.



Es evidente que algunos ingredientes sobresalen por su alta utilización en las preparaciones. No sorprende que la sal, el azúcar, el aceite de oliva y el agua sean los más utilizados.

Hay algunos que dentro de la misma lista se repiten con algunas variaciones y que pueden ser sustitutivos de otro ingrediente, por ejemplo, aceite de oliva - aceite de oliva extra virgen (incluso podría evaluarse si agruparlos a todos junto con aceite vegetal), así mismo la sal y la sal kosher.

2)Dataset de productos:

El dataset escogido tiene 534.019 productos de todo tipo vendidos en Carrefour , como no todos los productos son del tipo gastronómico lo primero que hay que hacer es filtrar obteniendo 2.779 productos que pueden ser usados en recetas de cocina.

Son 8 variables, sin embargo, por ahora solo se usa la correspondiente al nombre de productos, en la que se encuentran caracteres especiales y cantidades, así como palabras innecesarias que se eliminan. Donde al igual que antes se normaliza para que esté en UTF-8.

3)Dataset de imágenes:

no fue necesario realizar un eda pero si un data augmentation para potenciar el modelo, estos pasos serán explicados más adelante.

3. Preparación de datos

Limpieza de datos

Dado el enfoque del proyecto, la limpieza tanto de los productos como de los ingredientes de las recetas consistió en manejo de strings, aplicando técnicas como regex para la eliminación de caracteres numéricos y especiales. En esta etapa del proceso fue de gran ayuda realizar conteo de las palabras disponibles para priorizar las que más se repetían en el dataset.

Se han aplicado procesos de eliminación de caracteres especiales, acentos, espacios sobrantes, unidades de medida, adjetivos y otras palabras no deseadas. También se han normalizado las dos listas de ingredientes y convertido a singular los ingredientes que aparecían originalmente en plural. Luego, se ha separado los ingredientes de cada lista en una lista con ingredientes de una sola palabra y otra lista con ingredientes de dos palabras. Los ingredientes de más de dos palabras no se han tenido en cuenta.

Traducción dataset

Con el fin de conseguir el match de los ingredientes reconocidos en el ticket de la compra o la foto hecha de los productos adquiridos, se ha decidido añadir al dataset una columna adicional que solamente contiene los ingredientes de cada receta, sin unidades de medida, adjetivos, etc. Para ello, se han comparado los ingredientes del dataset, tras depurarlos, con los ingredientes de un corpus de comida, obtenidos de la librería NLTK.

Adicionalmente, se han buscado y representado gráficamente los ingredientes más utilizados en las recetas del dataset. Una vez conseguidos, se han añadido a la lista de ingredientes obtenidos del dataset. A continuación, se han eliminado los ingredientes duplicados de ambas listas. Para aplicar el último filtrado, se ha comparado la lista de ingredientes extraída del dataset con la lista de ingredientes del corpus de comida de NLTK. Nuestra lista final de ingredientes solamente contiene aquellos que también aparecen en el corpus.

Finalmente, se ha procedido a la traducción de los ingredientes de inglés, en su versión original, a español, gracias a la librería Googletrans, el cual cuenta con una limitación del número de palabras que pueden ser traducidas cada 24 horas. Por este motivo, se han almacenado los ingredientes en inglés junto con sus traducciones en un dataframe que, a su vez, se han guardado en un archivo CSV. Esto permitirá realizar traducciones en el futuro sin tener que depender del traductor de Google.

Como resultado de la limpieza de los dataset se originaron diferentes archivos en formato csv con la limpieza de las variables más importantes del proyecto que son la de ingredientes en el dataset de recetas y la de nombre en el dataset de productos. Para realizar varias comprobaciones se han almacenado distintas etapas de la limpieza.

Igualmente, se ha generado un archivo csv que contiene el diccionario de las palabras más relevantes correspondientes a ingredientes usados.

Data Augmentation

Debido a la baja cantidad imágenes que se tenía para entrenar el modelo, se aplican técnicas para generar imágenes nuevas a partir de las que se disponen y de esta manera enriquecer los datos

Para la generación de nuevas imágenes se han aplicado parámetros para alterar la iluminación de las imágenes, hacer zoom sobre estas y girarlas, así como moverlas ligeramente horizontal y verticalmente. Para cubrir los espacios que resultan de estas modificaciones se ha aplicado el método “nearest”, que los rellena con base en los píxeles más cercanos, evitando perder el sentido de las imágenes.

-

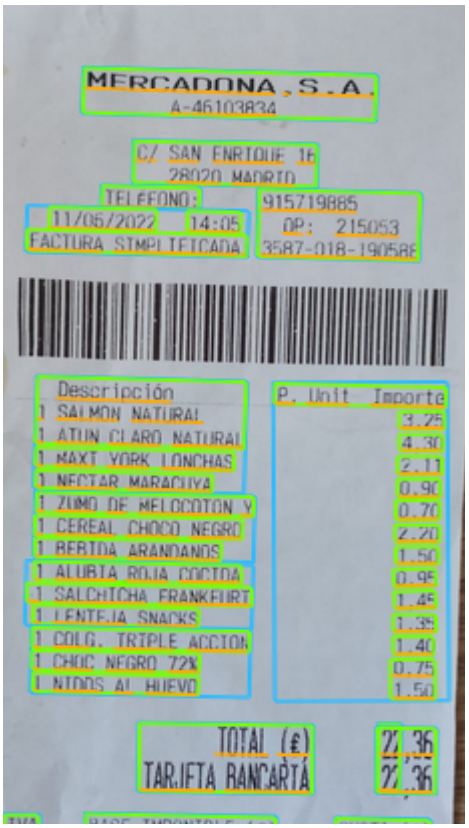
```
train_datagen = ImageDataGenerator(rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.3,
horizontal_flip=False,
fill_mode='nearest',
brightness_range=[0.4,1.5]
```

Sin embargo esta aproximación requirió un tratamiento posterior que permitiera reducir el tamaño del dataset resultante desde los 30GB finales a una cifra más razonable que permitiera entrenar el modelo en fases posteriores.

4. Reconocimiento de productos a partir de tickets de compra

Se ha decidido usar la API de Google Cloud Vision, cuyo modelo está entrenado para identificar texto en una imagen. En este punto fue necesario usar una cuenta de Google Cloud de uno de los integrantes del equipo y para vincularse con la API se obtuvieron credenciales, las cuales están integradas al código. El uso de esta API para los propósitos del proyecto tiene un uso limitado al escaneo de 1.000 imágenes de forma gratuita, lo cual es suficiente para hacer varias pruebas.

Como resultado de la lectura, se obtiene un archivo tipo json con múltiples variables. Se accede solamente a la sección donde se encuentra cada uno de los elementos de tipo texto, y se compara con la lista de elementos presentes en las recetas que corresponden a los ingredientes usados, de tal forma que todas aquellas palabras como teléfono, factura o total no sean reconocidas como producto.



5. Reconocimiento de productos a partir de imágenes de estos

Para esta funcionalidad se hace uso del modelo pre entrenado VGG16 disponible en Keras/Tensorflow. Es un modelo especializado en el reconocimiento de imágenes, a partir del cual se ha realizado un proceso de reentrenamiento mediante fine-tuning con el dataset resultante del webscrapping y el data-augmentation aplicado.

El modelo resultante es el siguiente:

```python
def set_up_training_model(self):
    """"""

    baseModel = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

    headModel = baseModel.output
    headModel = layers.Flatten(name="flatten")(headModel)
    headModel = layers.Dense(512, activation="relu")(headModel)
    headModel = layers.Dropout(0.5)(headModel)
    headModel = layers.Dense(256, activation='relu')(headModel)
    headModel = layers.Dense(self.num_classes, activation="softmax")(headModel)

    vgg_model = Model(inputs=baseModel.input, outputs=headModel)

    for layer in vgg_model.layers[:15]:
        layer.trainable = False

    learning_rate= 0.00005

    vgg_model.compile(optimizers.Adam(lr=learning_rate), loss="categorical_crossentropy", metrics=["accuracy"])

    return vgg_model
```

Sin embargo, en el proceso de entrenamiento se han encontrado diversas limitaciones técnicas, como un dataset limitado por la falta de recursos, como la capacidad computacional como para para procesar en memoria cerca de 20.000 imágenes.

De esta forma, se ha entrenado el modelo con solo una parte de las clases, con la consiguiente falta de accuracy (cercano al 50%), de forma que se pueda hacer una muestra de la funcionalidad en el la app aunque no sea representativa.

6. Creación modelo recomendador de recetas con base en los ingredientes disponibles

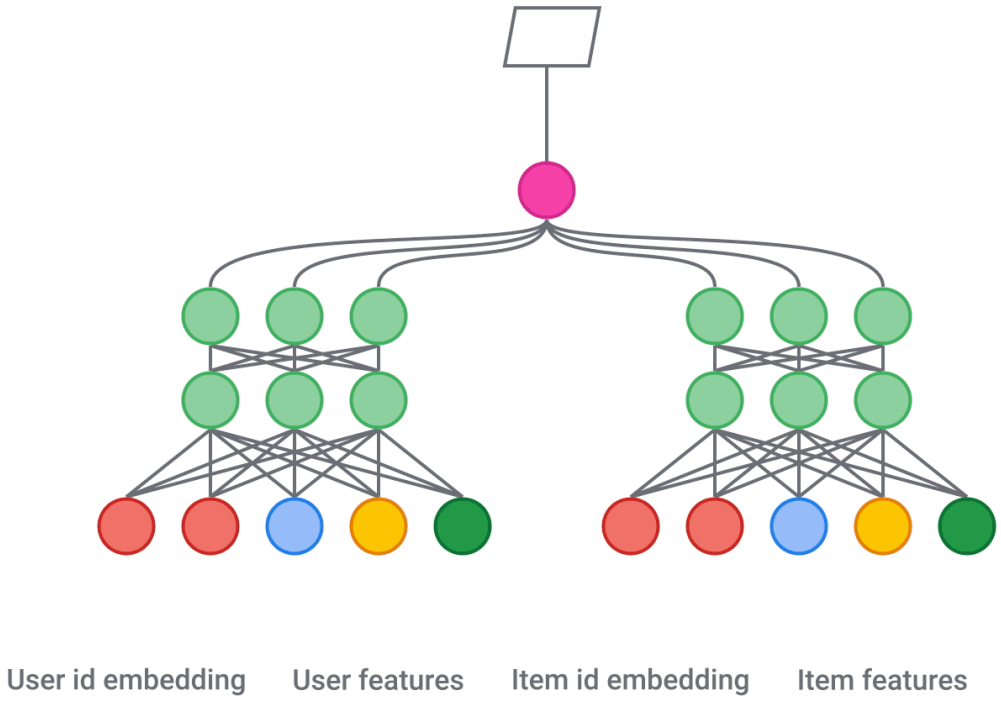
El objetivo para este punto es poder recomendar recetas a partir de los ingredientes que el usuario dispone, teniendo en cuenta su actividad en el pasado y la actividad de usuarios con perfiles similares.

Para recomendar recetas en base a los ingredientes, se ha utilizado una pipeline que parte de los productos reconocidos del ticket (y en un futuro reconocimiento de imágenes de los propios productos), que pasa por un proceso de limpieza y normalización, y finalmente se contraste cuáles de esos ingredientes son utilizados en cada una de las recetas. De esta forma se añade un score a cada receta en función de los ingredientes de los que dispone el usuario, y se seleccionan las recetas que tienen un mayor score y como segundo parámetro usado para seleccionar el top 5 de las recetas que se recomiendan, se toma en consideración las recetas con mayor puntuación (variable contenida en el dataset original).

Este proceso tiene la peculiaridad de que recibe los datos en español, que una vez normalizados son transformados al inglés para coincidir con el formato del dataframe de recetas, y que posteriormente vuelve a ser traducido, a español en este caso, una vez hecha la query del top 5 recetas recomendadas para el usuario. Los pasos de traducción se deben a las limitaciones para poder traducir un dataset grande como lo es el de recetas, mientras que haciendo traducciones a pequeños subconjuntos de datos como el

resultante de la query no supone ningún problema y sirve para realizar la demostración del funcionamiento de la app.

De cara a la mejora del modelo de recomendación, habría que introducir dos pasos extra intermedios. En primer lugar, una query que seleccione aquellas recetas que superen un ratio de ingredientes conocidos de una receta superior al 50%, y posteriormente aplicar un algoritmo de recomendación basado en el comportamiento del usuario. Para ello se ha elegido emplear el algoritmo de deep learning *TensorFlow Recommenders*, que permite incorporar multitud de variables frente al modelo más clásico de *Collaborative Filtering*.



8. Identificación de ingredientes faltantes para realizar la receta dada

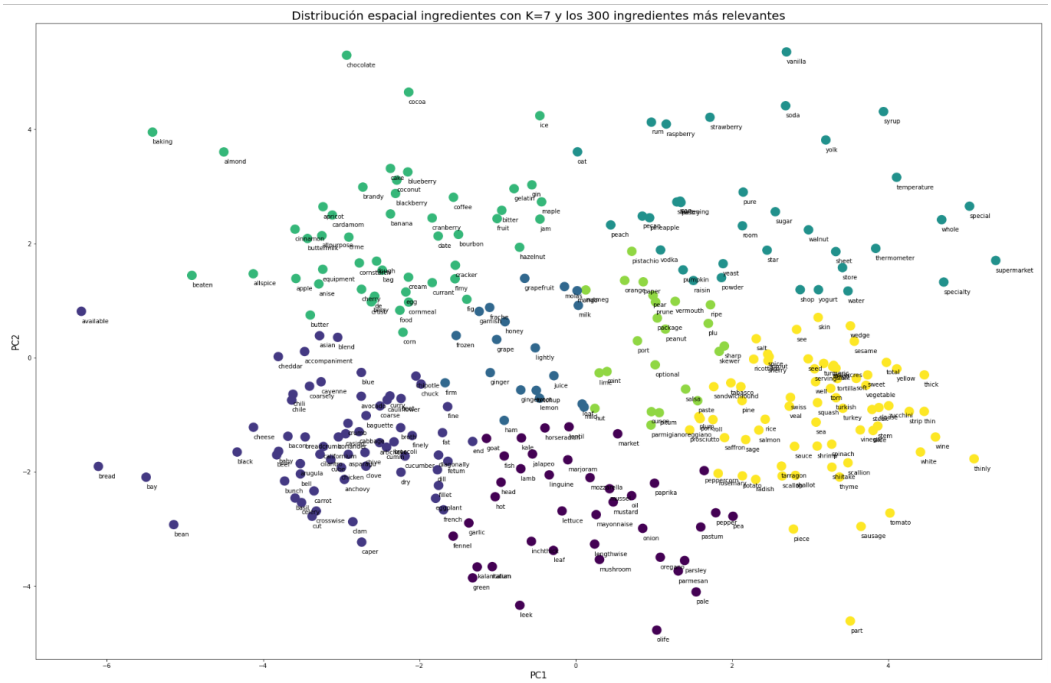
En este punto se deben tener dos listas: la lista de ingredientes identificados y la lista de ingredientes que corresponde a cada una de las recetas. Cuando se selecciona determinada receta en el dataframe, el programa accede a la columna de ingredientes (tipo lista), a la cual se le restan los ingredientes disponibles como se puede apreciar a continuación, obteniendo así una lista con los elementos que faltan para preparar la receta que se le indique.

```
missing_ingredients = list(set(ingredients_for_recipe[0]) - set(ingredientes_disponibles))
```

9. Creación de modelo que relacione productos asociados por su utilización en las preparaciones culinarias

Se aplica una técnica de procesamiento de lenguaje natural con el algoritmo word2vec que consiste en aprender asociaciones de palabras a partir de un corpus de texto, que en este caso son los ingredientes que hacen parte de una receta y se le aplica un parámetro para que solo tenga en cuenta las palabras que se repiten más de 10 ocasiones, logrando así traer las palabras que a priori son las más importantes.

El resultado se convierte en un array bidimensional, permitiendo así visualizarlo en un gráfico espacial. Se aplica un clustering de K-Means a las palabras resultantes de la aplicación del word2vec y para determinar el número de clusters se usó el método del codo y la suma de la distancia al cuadrado entre cada punto y el centroide (WCSS). A partir de lo anterior se determina que se van a agrupar las palabras en 7 clusters diferentes, así como se aprecia en el siguiente gráfico.



Se ve que algunos productos están más cercanos entre sí, un ejemplo de ello son los que se suelen usar para hacer postres, tales como el azúcar, la vainilla, el syrup), luego otros como la mayonesa y la mostaza muy cerca en el gráfico. Se marca alguna diferencia entre algunos ingredientes que por ejemplo suelen ser usados para condimentar, o para añadir toques especiales a las recetas, pero que no suelen ser los protagonistas, igualmente, se evidencian tendencias de ciertos elementos a ser usados en platos salados o dulces.

Se podría explorar más esto para crear nuevas recetas, sustitutivos de ingredientes y como herramientas de marketing en los supermercados.

10. Desarrollo página web e integración de todo el código

Para englobar todo el proyecto bajo un marco común, se decidió utilizar una página web para ubicar todos los apartados del proyecto de una forma eficaz y coherente. En otras palabras, para que el usuario pueda obtener las mejores recetas en base a fotografías de tickets de compra o de productos de cocina.

El frontend está basado en el stack estándar de tecnologías web (HTML, CSS, JavaScript) con la adición de Jinja template, el cual es un motor de plantillas web para Python usado para unir con el backend.

En el backend se usó el framework de Flask debido a su simplicidad y flexibilidad. En este backend se procesa lo que introduce el usuario para controlar lo que se muestra en la parte de Frontend. De este modo se hace amplio uso de OpenCV para el tratamiento en vivo de las imágenes y también se conecta con los modelos elaborados durante el proyecto.

La aplicación web se aloja en Amazon Web Services en una T3.large , la cual es una instancia de uso general con 4 gigabytes de ram y 2vCPUs con una AMI de AWS Linux2. También se hace uso de Route53, un servicio de AWS que actúa como servicio de DNS para enlazar la IP elástica adquirida con el dominio de ‘www.descubretureceta.com’.

Como detalle técnico adicional, se hace uso del servidor web de NGINX como reverse proxy de Flask a fin de gestionar mejor las conexiones y, sobre todo, poder hacer uso de Certbot para obtener un certificado HTTPS válido y autogestionado de “Let´s Encrypt” que permita establecer conexiones HTTPS sin avisos de seguridad en el navegador web del usuario.

7. CONCLUSIONES Y RESULTADOS

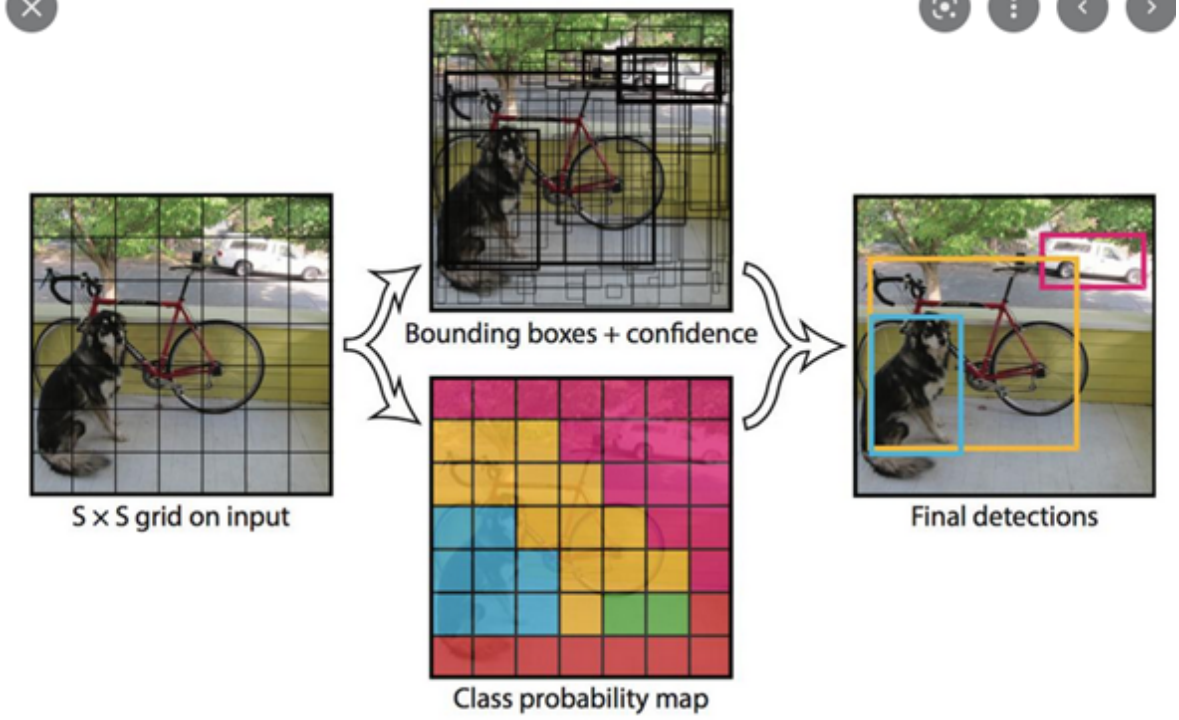
Después de un complejo proceso de tratamiento de datos es posible reconocer una lista de productos con algunas limitaciones, pues para evitar que se incluyan palabras no correspondientes a ingredientes a una receta, deben pasar ese filtro, por lo que algunos ingredientes poco usuales o escritos de forma abreviada o incorrecta no son tenidos en cuenta.

Por otro lado, el modelo de reconocimiento de productos se enfrenta a una complejidad de recursos más avanzada de lo permitido por el proyecto. De este modo, se ha intentado obtener un modelo de reconocimiento de productos en base a un dataset de imágenes aumentado en instancias cloud de hasta 64GB de memoria RAM sin éxito por falta de memoria. Este hecho se ha solventado en cierta manera con tratamientos posteriores, pero con la contrapartida de reducir enormemente su eficacia.

Dado que nuestro proyecto se dividía en dos facetas siendo una de ellas el reconocimiento de tickets, y la otra crear un programa que de reconocimiento de productos con el uso de estrategias de machine learning, se puede decir que se ha conseguido ampliamente el objetivo. Cabe mencionar que el uso de técnicas de Big Data es imprescindible para las dos facetas de nuestro proyecto. La primera por el tratamiento de datos tan masivos y la segunda por el número de operaciones que hay que hacer.

8. TRABAJO FUTURO

Una de las funcionalidades que se buscaba incluir en el proyecto pero que se tuvo que dejar a un lado por la limitación de recursos (tiempo, dinero e imágenes etiquetadas con productos) es la identificación de varios productos en una foto. Para esto, se considera que se podría aplicar un algoritmo YOLO, previamente con múltiples imágenes etiquetadas en las que aparecieran varios productos en una sola foto. La ventaja de este algoritmo sería la relativa agilidad (dada la complejidad del problema) para identificar los distintos objetos, pues el modelo pasa solamente una única vez por él.



En un futuro, se puede añadir una red neuronal y crear un generador de recetas con un modelo entrenado que pueda generar nuevas recetas que no estén incluidas en las originales. Si se entrena con otros aspectos de las recetas, incluidos los datos nutricionales, el tipo de cocina, la calificación y más. Usar un modelo LSTM o RNN.
En términos de web, se puede optar por una solución de microservicio, donde no todo está enlazado a una sola máquina, también que la información esté interconectada pero en sitios independientes.

ANEXOS

Webscrapping

```python

# Importación de librerías


from selenium import webdriver  # Webscrapping bot
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException

from selenium.webdriver.chrome.service import Service
from selenium.webdriver.firefox.service import Service

from selenium.webdriver.firefox.options import Options as FirefoxOptions
from selenium.webdriver.chrome.options import Options as ChromiumOptions

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

from webdriver_manager.chrome import ChromeDriverManager
from webdriver_manager.firefox import GeckoDriverManager
from webdriver_manager.core.utils import ChromeType

import logging  # Para generar logs
from logging.handlers import TimedRotatingFileHandler
from logging import Formatter

import datetime
import os
import time

import urllib
from shutil import make_archive


import pandas as pd  # Manejo de dataframes


# Importación de módulos


from values_scrapping_images import *


# Pequeñas funciones de apoyo


# Esta función se ha creado para mejorar comprensión de código en la configuración de logs

def UTCFormatter(logFormatter):
    '''
    Recibe un formatter de logueo
    Devuelve el horario a tiempo GMT
    '''
    logFormatter.converter = time.gmtime
    return logFormatter


def pathChecker(path):
    '''
    Recibe una ruta
    Crea la ruta en local si no es encontrada
    '''
    isExist = os.path.exists(path)
    if isExist == False:
        os.makedirs(path)
        print(path + " created")


# Comprobación de rutas

pathChecker(logPath)
pathChecker(imagesPath)

```

```
76
77  # Configuración de logs
78
79
80  # Se inicia el proceso de registro de logs a nivel de INFO.
81  logger = logging.getLogger('ScrapLog')
82  logger.setLevel(logging.INFO)
83
84  # Variables que determinan apartados posteriores
85  timestamp = datetime.datetime.utcnow().strftime('%Y%m%d_%H-%M-%S')
86  filename = f'ScrapImages{timestamp}.log'
87  formatter = logging.Formatter(
88      '[%(asctime)s] %(name)s {%(filename)s:%(lineno)d} %(levelname)s - %(message)s')
89
90
91  '''
92  Indican como se debe crear el archivo de log
93  Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
94  '''
95  '''
96  Indican como se debe crear el archivo de log
97  Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
98  '''
99  '''
100 Indican como se debe crear el archivo de log
101 Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
102 '''
103 if deleteOldLogs == True:
104     listFilesinCWD = os.listdir(logPath)
105     workPath = os.getcwd()
106     os.chdir(logPath)
107     for element in listFilesinCWD:
108         if element.endswith(".log"):
109             os.remove(os.path.join(os.getcwd(), element))
110         if element.endswith(".png"):
111             os.remove(os.path.join(os.getcwd(), element))
112     os.chdir(workPath)
113
114 fileHandler = logging.FileHandler(filename=f"{logPath}{filename}")
115 logging.Formatter.converter = time.gmtime
116
117 fileHandler.setLevel(logging.INFO)
118 fileHandler.setFormatter(UTCFormatter(formatter))
119 logger.addHandler(fileHandler)
120
121
122 # Importación de datos
123
124 # Se importa el dataset especificado en las variables generales definidas anteriormente.
125
126 if linkWebCSV:
127     df = pd.read_csv(fileToDF)
128 else:
129     df = pd.read_csv(f"{pathToDF}{fileToDF}")
130     # df.drop(columns=df.columns[0], axis=1, inplace=True)
131
132
133 # Lógica del Scrapping
134
135 def SelectImageSearched(numberImage):
136     imageSearched = f"/html/body/div[2]/c-wiz/div[3]/div[1]/div/div/div/div[1]/div[1]/span/div[1]/div[1]/div[{numberImage}]/a[1]/div[1]/img"
137     return imageSearched
138
139 # La siguiente función obtiene la url del primer resultado que aparece en google imágenes con el nombre del producto que recibe.
140
141 def ScrapFunction(position, prodToScrap, urlToScrap, driver, numberImage):
142     try:
143
144         logger.info(f"Started with: {prodToScrap} and {numberImage} iteration")
145
146         driver.get(urlToScrap)
147
148         if position == 0 and numberImage==1:
149             driver.find_element(By.XPATH, privacyButton).click()
150
151         selectImageBox=driver.find_element(By.XPATH, searchImageBar)
152         selectImageBox.send_keys(prodToScrap)
153         selectImageBox.send_keys(Keys.ENTER)
154
155         time.sleep(1)
156         driver.find_element(By.XPATH, SelectImageSearched(numberImage)).click()
157         time.sleep(1)
158
159         urlImage = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.XPATH, selectDefImage))).get_attribute("src")
160
161         logger.info(f"Scrapped: {urlImage}")
162
163         if takeScreenshot:
164             driver.save_screenshot(f"{logPath}Selenium-Progress.png")
165
166         if downloadImages:
167             pathChecker(imagesPath + prodToScrap)
168             urllib.request.urlretrieve(urlImage, f"{imagesPath}{prodToScrap}/{prodToScrap}-{numberImage}.png")
169
170         return urlImage
171
172     except:
173         if enableFullErrors:
174             logger.exception('')
175             if position == 0:
176                 driver.save_screenshot(f"{logPath}Selenium-CrashBrowser.png")
177         else:
178             logger.info(f"FUNCTIONERROR: {prodToScrap}")
179         return None
180
```

```python

# Configuración de logs


# Se inicia el proceso de registro de logs a nivel de INFO.
logger = logging.getLogger('ScrapLog')
logger.setLevel(logging.INFO)

# Variables que determinan apartados posteriores
timestamp = datetime.datetime.utcnow().strftime('%Y%m%d_%H-%M-%S')
filename = f'ScrapImages{timestamp}.log'
formatter = logging.Formatter(
    '[%(asctime)s] %(name)s {%(filename)s:%(lineno)d} %(levelname)s - %(message)s')


'''
Indican como se debe crear el archivo de log
Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
'''
'''
Indican como se debe crear el archivo de log
Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
'''
'''
Indican como se debe crear el archivo de log
Si "deleteOldLogs" es True, sólo se conservará el último archivo de log
'''
if deleteOldLogs == True:
    listFilesinCWD = os.listdir(logPath)
    workPath = os.getcwd()
    os.chdir(logPath)
    for element in listFilesinCWD:
        if element.endswith(".log"):
            os.remove(os.path.join(os.getcwd(), element))
        if element.endswith(".png"):
            os.remove(os.path.join(os.getcwd(), element))
    os.chdir(workPath)

fileHandler = logging.FileHandler(filename=f"{logPath}{filename}")
logging.Formatter.converter = time.gmtime

fileHandler.setLevel(logging.INFO)
fileHandler.setFormatter(UTCFormatter(formatter))
logger.addHandler(fileHandler)


# Importación de datos

# Se importa el dataset especificado en las variables generales definidas anteriormente.

if linkWebCSV:
    df = pd.read_csv(fileToDF)
else:
    df = pd.read_csv(f"{pathToDF}{fileToDF}")
    # df.drop(columns=df.columns[0], axis=1, inplace=True)


# Lógica del Scrapping

def SelectImageSearched(numberImage):
    imageSearched = f"/html/body/div[2]/c-wiz/div[3]/div[1]/div/div/div/div[1]/div[1]/span/div[1]/div[1]/div[{numberImage}]/a[1]/div[1]/img"
    return imageSearched

# La siguiente función obtiene la url del primer resultado que aparece en google imágenes con el nombre del producto que recibe.

def ScrapFunction(position, prodToScrap, urlToScrap, driver, numberImage):
    try:

        logger.info(f"Started with: {prodToScrap} and {numberImage} iteration")

        driver.get(urlToScrap)

        if position == 0 and numberImage==1:
            driver.find_element(By.XPATH, privacyButton).click()

        selectImageBox=driver.find_element(By.XPATH, searchImageBar)
        selectImageBox.send_keys(prodToScrap)
        selectImageBox.send_keys(Keys.ENTER)

        time.sleep(1)
        driver.find_element(By.XPATH, SelectImageSearched(numberImage)).click()
        time.sleep(1)

        urlImage = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.XPATH, selectDefImage))).get_attribute("src")

        logger.info(f"Scrapped: {urlImage}")

        if takeScreenshot:
            driver.save_screenshot(f"{logPath}Selenium-Progress.png")

        if downloadImages:
            pathChecker(imagesPath + prodToScrap)
            urllib.request.urlretrieve(urlImage, f"{imagesPath}{prodToScrap}/{prodToScrap}-{numberImage}.png")

        return urlImage

    except:
        if enableFullErrors:
            logger.exception('')
            if position == 0:
                driver.save_screenshot(f"{logPath}Selenium-CrashBrowser.png")
        else:
            logger.info(f"FUNCTIONERROR: {prodToScrap}")
        return None

```

RECONOCIMIENTO IMAGENES

```python
import matplotlib.pyplot as plt

import tensorflow as tf

from keras import optimizers
from keras.applications.imagenet_utils import preprocess_input
from keras.applications.vgg16 import VGG16
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras.models import Sequential, Model
from keras.preprocessing import image
from keras.utils import image_dataset_from_directory, load_img, img_to_array

from keras import layers

from PIL import Image

import numpy as np

# from ticket_recognition.remove_bg_and_super_resolution import delete_background

# from ticket_recognition.remove_bg_and_super_resolution

# from singleton import SingletonMeta

import os

class Recognize_Products_Images:

    def __init__(
        self,
        base_path: str,
        environment: str = "production",
    ) -> None:

        self.width_shape = 224
        self.height_shape = 224
        self.epochs = 10
        self.batch_size = 32
        self.base_path = base_path
        self.def_product_list = []
        self.image_path = ""
        self.model_weights_path = "product_recognition/model_weights.h5"

        assert environment in ["local", "production"]
        self.environment = environment

    @property
    def class_names(self):
        return os.listdir(self.base_path)

    @property
    def num_classes(self):
        return len(self.class_names)


    def set_image_path(self, image_path: str) -> None:
        """Set the image path"""
        self.image_path = image_path

    def set_model_weights_path(self, model_weights_path: str) -> None:
        """"""
        self.model_weights_path = model_weights_path

    # Predict image label
    def run(self) -> str:
        """Run method.

        Returns:
            str: list with the products as cleaned as possible.
        """

        if self.environment == "local":
            return self._local_run()

        product = self.predict_label()
        self.def_product_list.append(product)

        return self.def_product_list

    def _local_run(self) -> str:
        """Run method in local environments.
```

```
82
83          Returns:
84              str: list with the products as cleaned as possible.
85          """
86
87          checker = input("Deseas entrenar el modelo?: si o no - ")
88          while checker != "si" and checker != "no":
89              checker = input("Valor no contemplado. Por favor, introduce si desea entrenar el modelo: si o no - ")
90
91          if checker == "si":
92              if os.path.exists(self.model_weights_path) == False:
93                  self.train_model()
94
95              else:
96                  checker = input("Ya existe un modelo en la actualidad, sigue queriendo reentrenarlo?: si o no - ")
97                  while checker != "si" and checker != "no":
98                      checker = input("Valor no contemplado. Por favor, introduce si desea reentrenar el modelo: si o no - ")
99
100                 if checker == "si":
101                     self.train_model()
102
103         # Predicción de imagenes
104         print("[PREDICCIÓN DE IMÁGENES]:")
105         self.image_path = input("Introduce el enlace de la imagen del producto: ")
106         # no_bg_path = delete_background(self.image_path)
107         product = self.predict_label()
108         self.def_product_list.append(product)
109
110         checker = input("Quieres subir otro producto?: si o no - ")
111         while checker != "si" and checker != "no":
112             checker = input("Valor no contemplado. Por favor, introduce si desea o no cargar otro producto: si o no - ")
113
114         if checker == "si":
115            self._local_run()
116
117         return self.def_product_list
118
119     # Training Model
120     def train_model(self):
121         """"""
122         dataset = self.get_data_from_directory()
123         model = self.set_up_training_model()
124
125         X_train = np.concatenate([x for x, y in dataset], axis=0)
126         y_train = np.concatenate([y for x, y in dataset], axis=0)
127
128         lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=8, verbose=1, mode='max', min_lr=5e-5)
129         checkpoint = ModelCheckpoint('vgg16_finetune.h15', monitor= 'val_accuracy', mode= 'max', save_best_only = True, verbose= 1)
130
131         model.fit(
132             X_train,
133             y_train,
134             epochs=30,
135             batch_size=15,
136             validation_split=0.2,
137             callbacks=[lr_reduce,checkpoint],
138         )
139
140         model.save_weights(self.model_weights_path)
141
142     # Predict product label
143     def predict_label(self):
144         """Predict the label for a product image"""
145
146         # Set model and load weights
147         model = self.set_up_training_model()
148         model.load_weights(self.model_weights_path)
149
150         # Load and preprocess image
151         img = load_img(self.image_path, target_size=(224, 224))
152         img = img_to_array(img)
153         img = np.reshape(img, (-1, 224, 224, 3))
154         img = preprocess_input(img)
155
156         # Get the label
157         y_prob = model.predict(img)
158         label = y_prob.argmax(axis=-1)
159         lista = clase.class_names
160         lista.sort()
161         label_name = lista[label[0]]
162         print(label_name)
163         return label_name
```

```
164
165
166     def get_data_from_directory(self) -> tf.data.Dataset:
167         """Get data from a directory.
168
169         Returns:
170             tf.data.Dataset: products dataset.
171         """
172         return image_dataset_from_directory(
173             directory=self.base_path,
174             labels="inferred",
175             label_mode="categorical",
176             class_names = self.class_names,
177             image_size=(self.width_shape, self.height_shape),
178             batch_size=self.batch_size,
179         )
180
181     def set_up_training_model(self):
182         """"""
183
184         baseModel = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
185
186         headModel = baseModel.output
187         headModel = layers.Flatten(name="flatten")(headModel)
188         headModel = layers.Dense(512, activation="relu")(headModel)
189         headModel = layers.Dropout(0.5)(headModel)
190         headModel = layers.Dense(256, activation='relu')(headModel)
191         headModel = layers.Dense(self.num_classes, activation="softmax")(headModel)
192
193         vgg_model = Model(inputs=baseModel.input, outputs=headModel)
194
195         # for layer in baseModel.layers:
196         #     layer.trainable = False
197         for layer in vgg_model.layers[:15]:
198             layer.trainable = False
199
200         learning_rate= 0.00005
201
202         vgg_model.compile(optimizers.Adam(lr=learning_rate), loss="categorical_crossentropy", metrics=["accuracy"])
203
204         return vgg_model
205
206
207 ###### Set vars ######
208 images_directory = "D:\\Desktop\myProjects\exampleImages"
209
210 clase = Recognize_Products_Images(images_directory, "local")
211
212 print(clase.def_product_list)
213
```

RECONOCIMIENTO DE TICKET

```python
import io

import os

import re

from google.protobuf.json_format import MessageToDict

from google.cloud import vision

from google.cloud.vision_v1 import types

from typing import List

# from ticket_recognition.remove_bg_and_super_resolution import delete_background, super_resolution

class Recognize_Tickets_Products:

    def __init__(
        self,
        credentials_path: str,
        environment: str = "production",
    ) -> None:

        self.credentials_path = credentials_path
        self.image_path = ""
        self.super_resolution_model_path = ""
        self.def_product_list = []
        self.image_path = ""

        assert environment in ["local", "production"]
        self.environment = environment


    def run(self) -> None:
        """Run method."""

        if self.environment == "local":
            self._local_run()
            return

        product_list = self.ticket_ocr_recognition()
        self.clean_recognized_products(product_list)

        return self.def_product_list

    def _local_run(self) -> List[str]:
        """Run method in local environments.

        Returns:
            List[str]: list with the products as cleaned as possible.
        """

        self.image_path = input("Introduce el enlace del ticket: ")

        if self.super_resolution_model_path != "":
            no_bg_path = delete_background(self.image_path)
            image = super_resolution(no_bg_path, self.super_resolution_model_path)
            image.save(self.image_path)
        else:
            checker = input("¿Deseas borrar el background de la imagen y aplicar super resolución?: si o no - ")
            while checker != "si" and checker != "no":
                checker = input("Valor no contemplado. Por favor, introduce si desea o no borrar el background y aplicar super resolución: si o no - ")

            if checker == "si":
                self.super_resolution_model_path = input("Introduce al ruta del modelo: ")
                no_bg_path = delete_background(self.image_path)
                self.image_path = super_resolution(no_bg_path, self.super_resolution_model_path)

        product_list = self.ticket_ocr_recognition()
        self.clean_recognized_products(product_list)
        # product_list = product_list + ticket_ocr_recognition(credentials, image_path)

        if checker == "si":
            os.remove(no_bg_path)
            os.remove(self.image_path)

        checker = input("Quieres subir otro ticket?: si o no - ")
        while checker != "si" and checker != "no":
            checker = input("Valor no contemplado. Por favor, introduce si desea o no cargar otro ticket: si o no - ")
```

```python

        if checker == "si":
           self._local_run()

        return self.def_product_list

    def set_image_path(self, image_path: str) -> None:
        """Set the image path"""
        self.image_path = image_path


    def ticket_ocr_recognition(self):
        """Apply google api for the text ocr recognition to ticket.

        Returns:
            List[str]: list with recognized products.
        """

        # Set the credentials to the google api
        os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = self.credentials_path

        client = vision.ImageAnnotatorClient()

        if self.environment == "local":  # Get image in local environment
            with io.open(self.image_path, 'rb') as image_file:
                content = image_file.read()
            image = vision.Image(content=content)
        else:                            # Get image in production environment
            image = types.Image()
            image.source.image_uri = self.image_path

        # Apply text recognition
        response = client.text_detection(image=image)

        # Get the dictionary with products and relative positions
        texts = MessageToDict(response._pb)

        # Get the products text and split in a list.
        text = texts["textAnnotations"]
        text = text[0]
        text = text["description"]
        text = text.lower()
        text = text.split("\n")

        return text

    def clean_recognized_products(
        self,
        products_list: List[str],
    ) -> List[str]:
        """Clean the recognized products

        Args:
            products_list: list with the recognized products.
        """

        # Go through a products_list to clean each product
        for i, item in enumerate(products_list):

            # Remove numbers
            new_item = re.sub("\d+", "", item)

            # Remove punctiation marks
            new_item = re.sub("[^\w\s]",'', new_item)

            # Split each item in a list of words
            new_item = new_item.split(" ")

            # Go through each item to remove unneccesary words
            for count, word in enumerate(new_item):
                if len(word) <= 2:
                    new_item.remove(word)
                    count -= 1
                    continue


            if new_item == []:  # Check if the resultant new item is empty
                continue
            else:               # Concatenate again all the words and add to def product list
                new_item = " ".join(new_item)
                self.def_product_list.append(new_item)


clase = Recognize_Tickets_Products("My First Project-e1e084943099.json", environment= "local")

lista = clase._local_run()

print(lista)


```

RECOMENDADOR DE RECETAS

```python
import pandas as pd
import numpy as np
import re
import string
from typing import List
from googletrans import Translator


class Recommender:

    def __init__(self, df_receipes_path, df_ingredient_translation_path, products_recognized_list) -> None:
        self.df_receipes_path = df_receipes_path
        self.df_ingredient_translation_path = df_ingredient_translation_path
        self.products_recognized_list = products_recognized_list

    @property
    def df_receipes(self):
        df = pd.read_json(self.df_receipes_path)
        return df.dropna(subset=['ingredients'])

    @property
    def df_ingredients(self):
        return pd.read_csv(self.df_ingredient_translation_path, sep="|")

    @property
    def english_list(self):
        return self.df_ingredients["English"].tolist()

    @property
    def spanish_list(self):
        return self.df_ingredients["Spanish"].tolist()

    @property
    def translation_dict(self):
        return dict(zip(self.english_list, self.spanish_list))

    def run(self):
        """"""
        normalized_list = self.normalize_products()
        translated_list = self.translate_products(normalized_list)
        df_top_5 = self.get_top_5_receipes(translated_list)
        return self.translate_df(df_top_5)

    def normalize_products(self):
        """"""
        return_list = []
        for i, product in enumerate(self.products_recognized_list):
            # Casuistica de producto de una sola palabra que se encuentra exactamente igual en la lista
            if product in self.spanish_list:
                return_list.append(product)
                continue

            splitted_product = product.split(" ")

            # Casuistica producto con mas de una palabra
            if len(splitted_product) > 1:
                # Se comprueba que la primera palabra del producto exista en la lista de productos normalizados
                if splitted_product[0] in self.spanish_list:
                    return_list.append(splitted_product[0])
                # Se comprueba que si el producto igual pero escrito en otro orden y si es asi se sobreescribe
                for item in self.spanish_list:
                    counter = 0
                    splitted_item = item.split(" ")
                    for word in splitted_product:
                        if word in splitted_item:
                            counter += 1
                    if counter == len(splitted_product) and counter == len(splitted_item):
                        return_list.append(item)
                        break
                continue
        return return_list

    def translate_products(self, normalized_list):
        """"""
        return_list = []
        for key, value in self.translation_dict.items():
            for product in normalized_list:
                if product == value:
                    return_list.append(key)
        return return_list
```

```python

    def match_ingredients(self, translated_list, ingredient_list):
        """"""
        if not isinstance(ingredient_list, List):
            return 0
        elif len(ingredient_list) == 0:
            return 0
        counter = 0
        for ingredient in ingredient_list:
            splitted_ingredient = ingredient.split(" ")
            for product in translated_list:
                if product in splitted_ingredient:
                    counter += 1
                    break
                else:
                    splitted_product = product.split(" ")
                    if len(splitted_product) > 1:
                        sub_counter = 0
                        for word in splitted_product:
                            if word in splitted_ingredient:
                                sub_counter += 1
                        if sub_counter == len(splitted_product):
                            counter+=1
                            break
                        else:
                            if splitted_product[0] in splitted_ingredient:
                                counter+=1
        return int(counter)

    def get_top_5_receipes(self, product_list):
        """"""
        df = self.df_receipes
        df["score"] = self.df_receipes["ingredients"].apply(lambda x: self.match_ingredients(product_list, x))
        df.sort_values(by=["score", "rating"], ascending=False, inplace=True)
        return df.iloc[0:5]

    def translate_cell(self, cell_value, translator):
        if isinstance(cell_value, List):
            aux_list = []
            for word in cell_value:
                translated_text = translator.translate(word, src='en', dest='es')
                aux_list.append(translated_text.text)
            return aux_list
        elif isinstance(cell_value, str):
            translated_text = translator.translate(cell_value, src='en', dest='es')
            return translated_text.text
        else:
            return cell_value

    def translate_df(self, df):
        translator = Translator()
        col_list = []
        for col in df.columns:
            translated_text = translator.translate(col, src='en', dest='es')
            col_list.append(translated_text.text)
        df.columns = col_list

        for col in df.columns:
            if df[col].dtype != "object":
                continue
            df[col] = df[col].apply(lambda x: self.translate_cell(x, translator))
        return df

lista = ['CARREFOUR MARKET', 'CONDE PENALVER  MADRID  ', 'CLUB', 'VIENES', 'CARREFOURESCLUBCARREFOUR', 'SUPERMERCADOS CHAMPION', 'CIF', 'TELEFONO TIENDA ', 'ATENCION CLIENTE
lista = list(map(str.lower, lista))
clase = Recommender("recommender_system/full_format_recipes.json", "recommender_system\diccionario_ingredientes.csv", lista)

df = clase.run()

print(df.head(10))
```