

✓ Neural Network Recipe Recommendation System with Embeddings for Dietary Restriction

Project by Nicole Michaud, 02/26/2024

✓ Business Problem:

It can be hard to continuously come up with new and interesting recipes to cook, especially if you have certain dietary restrictions. Many people use websites such as food.com to find, try, and rate recipes. From user and recipe data from Food.com, can we provide users with recommendations for the next recipes that users should try, taking into account their dietary specifications?

✓ Pre-Processing and Data Exploration:

Loading the data and necessary packages:

```
1 # TF's recommender imports
2 !pip install -q tensorflow-recommenders
3 !pip install -q tensorflow_ranking
4 !pip install -q --upgrade tensorflow-datasets
5 !pip install -q scann
```

```
_____ 475.2/475.2 MB 2.9 MB/s eta 0:00:0
_____ 5.5/5.5 MB 102.7 MB/s eta 0:00:00
_____ 442.0/442.0 kB 40.9 MB/s eta 0:00:
_____ 1.7/1.7 MB 78.2 MB/s eta 0:00:00
```

```
ERROR: pip's dependency resolver does not currently take into account all the pa
scann 1.2.10 requires tensorflow~=2.13.0, but you have tensorflow 2.15.0.post1 w
ERROR: pip's dependency resolver does not currently take into account all the pa
tensorflow-serving-api 2.14.1 requires tensorflow<3,>=2.14.1, but you have tenso
```

```
1 import tensorflow as tf
2 print("TensorFlow version:", tf.__version__)
```

TensorFlow version: 2.13.1

```
1 import pandas as pd
2 import nltk
3 nltk.download('punkt')
4 from nltk.corpus import stopwords
```

```
5 import re
6 import numpy as np
7 import pickle
8 from nltk.tokenize import RegexpTokenizer, word_tokenize
9 import io
10 from collections import defaultdict
11 import os
12 import pprint
13 import tempfile
14 from typing import Dict, Text
15 import numpy as np
16 import tensorflow_datasets as tfds
17 import tensorflow_recommenders as tfre
18 import tensorflow_ranking as tfr
19 import seaborn as sns
20 import matplotlib.pyplot as plt
21 %matplotlib inline
22
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call `drive.mount('/content/drive', force_remount=True)`

Getting the data from kaggle:

```
1 #upload the kaggle.json file to load kaggle data
2 from google.colab import files
3 files.upload()
```

```
1 !mkdir -p ~/.kaggle
2 !cp kaggle.json ~/.kaggle/
```

```
1 !pip install -q kaggle
2
3 # This permissions change avoids a warning on Kaggle tool startup.
4 !chmod 600 ~/.kaggle/kaggle.json
```

```
1 #download the data from kaggle
2 !kaggle datasets download -d shuyangli94/food-com-recipes-and-user-interactions
```

```

1 #unzip data from kaggle
2 import zipfile
3
4 # Define the path to your zip file
5 file_path = '/content/drive/MyDrive/Capstone/capstone_data/food-com-recipes-and-u
6 # Unzip the file to a specific destination
7 with zipfile.ZipFile(file_path, 'r') as zip_ref:
8     zip_ref.extractall('/content/drive/MyDrive/Capstone/capstone_data')
9

```

Loading the user and item data:

```

1 #read in the specific datasets to be used:
2 user_data = pd.read_csv('/content/drive/MyDrive/Capstone/capstone_data/RAW_intera
3 recipe_data = pd.read_csv('/content/drive/MyDrive/Capstone/capstone_data/RAW_reci

```

✓ Data Exploration

Viewing the data to gain understanding of its contents:

```
1 user_data.head()
```

	user_id	recipe_id	date	rating	review
0	38094	40893	2003-02-17	4	Great with a salad. Cooked on top of stove for...
1	1293707	40893	2011-12-21	5	So simple, so delicious! Great for chilly fall...
2	8937	44394	2002-12-01	4	This worked very well and is EASY. I used not...
3	126440	85009	2010-02-27	5	I made the Mexican topping and took it to bunk...
4	57222	85009	2011-10-01	5	Made the cheddar bacon topping, adding a sprin...

```
1 user_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1132367 entries, 0 to 1132366
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   user_id     1132367 non-null    int64
1   recipe_id   1132367 non-null    int64
2   date        1132367 non-null    object
3   rating      1132367 non-null    int64
4   review      1132198 non-null    object

```

```
dtypes: int64(3), object(2)
memory usage: 43.2+ MB
```

```
1 recipe_data.head()
```

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_st
0	arriba baked winter squash mexican style	137739	55	47892	2005-09-16	['60- minutes-or- less', 'time- to-make', 'course...	[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	
1	a bit different breakfast pizza	31490	30	26278	2002-06-17	['30- minutes-or- less', 'time- to-make', 'course...	[173.4, 18.0, 0.0, 17.0, 22.0, 35.0, 1.0]	
2	all in the kitchen chili	112140	130	196586	2005-02-25	['time-to- make', 'course', 'preparation', 'mai...	[269.8, 22.0, 32.0, 48.0, 39.0, 27.0, 5.0]	
3	alouette potatoes	59389	45	68585	2003-04-14	['60- minutes-or- less', 'time- to-make', 'course...	[368.1, 17.0, 10.0, 2.0, 14.0, 8.0, 20.0]	
4	amish tomato ketchup for canning	44061	190	41706	2002-10-25	['weeknight', 'time-to- make', 'course', 'main-...	[352.9, 1.0, 337.0, 23.0, 3.0, 0.0, 28.0]	

```
1 recipe_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231637 entries, 0 to 231636
Data columns (total 12 columns):
#   Column              Non-Null Count  Dtype
---  -
0   name                 231636 non-null object
1   id                   231637 non-null int64
2   minutes              231637 non-null int64
3   contributor_id       231637 non-null int64
4   submitted            231637 non-null object
5   tags                 231637 non-null object
6   nutrition            231637 non-null object
7   n_steps              231637 non-null int64
```

```

8  steps          231637 non-null object
9  description    226658 non-null object
10 ingredients    231637 non-null object
11 n_ingredients  231637 non-null int64
dtypes: int64(5), object(7)
memory usage: 21.2+ MB

```

Renaming the 'id' column in the recipe dataframe to match the recipe_id column in the user dataframe:

```
1 recipe_data = recipe_data.rename(columns={"id": "recipe_id"})
```

Investigating the total number of unique users and recipes in the data:

```

1 print(len(user_data['user_id'].unique()))
2 print(len(recipe_data['recipe_id'].unique()))

226570
231637

```

With 226,570 users and 231,637 recipes, there are less users than there are recipes.

✓ Data Preparation

Dropping the columns I know I won't be working with:

```

1 user_recipe_ratings = user_data.drop(columns=['review', 'date'])

1 recipe_data = recipe_data.drop(columns=['contributor_id', 'submitted', 'nutrition
2

1 # Making sure text features are strings so that they can be cleaned properly
2
3 recipe_data['tags'] = recipe_data['tags'].astype(str)

```

```

1 # Creating a function to perform cleaning steps at once (Removes numbers and unne
2 nltk.download('stopwords')
3 stopwords_list = stopwords.words('english')
4
5 no_bad_chars = re.compile('[!\"#$%&()*+-./:;=>?@[\\]^_`{|}~\\n - ]')
6 no_nums = re.compile('[\\d-]')
7
8 def clean_text(text):
9     text = no_nums.sub('', text)
10    text = no_bad_chars.sub(' ', text)
11    text = text.lower()
12    text = ' '.join(word for word in text.split() if word not in stopwords_list)
13    return text

```

```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

```

```

1 #Applying text cleaning function to text columns
2 recipe_data_cleaned = recipe_data.copy()
3 recipe_data['name'] = recipe_data['name'].astype(str)
4 recipe_data_cleaned['name'] = (recipe_data['name']).apply(clean_text)
5 recipe_data['tags'] = recipe_data['tags'].astype(str)
6 recipe_data_cleaned['tags'] = (recipe_data['tags']).apply(clean_text)
7 recipe_data_cleaned.head()

```

	name	recipe_id	tags	description	ingredients
0	arriba baked winter squash mexican style	137739	'minutesorless' 'timetomake' 'course' 'maining...	autumn is my favorite time of year to cook! th...	['winter squash', 'mexican seasoning', 'mixed ...
1	a bit different breakfast pizza	31490	'minutesorless' 'timetomake' 'course' 'maining...	this recipe calls for the crust to be prebaked...	['prepared pizza crust', 'sausage patty', 'egg...
2	all in the kitchen chili	112140	'timetomake' 'course' 'preparation' 'maindish'...	this modified version of 'mom's' chili was a h...	['ground beef', 'yellow onions', 'diced tomato...
3	alouette potatoes	59389	'minutesorless' 'timetomake' 'course' 'maining...	this is a super easy, great tasting, make ahea...	['spreadable cheese with garlic and herbs', 'n...

Feature Engineering to categorize each recipe as different diet types (Vegetarian, Vegan, and/or Gluten-Free):

```

1 # creating a new column to classify recipes as gluten-free or not
2 GF = []
3 #tags column contains the most information on gluten-free recipes
4 #(see miscellaneous notebook)
5 for row in recipe_data_cleaned['tags']:
6     if "gluten-free" in row : GF.append("Gluten-Free")
7     elif "gluten free" in row : GF.append("Gluten-Free")
8     else: GF.append("None")

1 recipe_data_cleaned['GF'] = GF

1 #Ingredient lists for diet filtering:
2 vegan = ['ham', 'beef', 'meat', 'chicken', 'pork', 'bacon', 'sausage', 'lamb', 'v
3
4 vegetarian = ['ham', 'beef', 'meat', 'chicken', 'pork', 'bacon', 'sausage', 'lamb

1 # creating two new columns to classify recipes as vegetarian or not
2 # and as vegan or not:
3
4 recipe_data_cleaned['vegetarian'] = None
5 recipe_data_cleaned['vegan'] = None

1 # Filtering through the 'ingedients' column for ingredients that
2 # aren't vegetarian or vegan
3 vege_pattern = '|'.join(vegetarian)
4 vegan_pattern = '|'.join(vegan)
5
6
7 recipe_data_cleaned.vegetarian = recipe_data_cleaned.ingredients.str.contains(vege
8 recipe_data_cleaned.vegan = recipe_data_cleaned.ingredients.str.contains(vegan_pa

1 # Changing Boolean values to words to indicate the diet-type
2 recipe_data_cleaned['vegetarian'] = recipe_data_cleaned['vegetarian'].astype(str)
3 recipe_data_cleaned['vegetarian'] = recipe_data_cleaned['vegetarian'].replace({'F

1 recipe_data_cleaned['vegan'] = recipe_data_cleaned['vegan'].astype(str)
2 recipe_data_cleaned['vegan'] = recipe_data_cleaned['vegan'].replace({'False': 'Ve

1 #making one column of the diet types of each recipe combined and dropping the ind
2 recipe_data_cleaned['diets_combined'] = recipe_data_cleaned[['vegetarian', 'vegan
3 recipe_data_cleaned = recipe_data_cleaned.drop(columns=['GF', 'vegetarian', 'vega
4 recipe_data_cleaned.head()

```

	name	recipe_id	tags	description	ingredients	diets_combined
0	arriba baked winter squash mexican style	137739	'minutesorless' 'timetomake' 'course' 'maining...	autumn is my favorite time of year to cook! th...	['winter squash', 'mexican seasoning', 'mixed ...	[Vegetarian, None, None]
1	a bit different breakfast pizza	31490	'minutesorless' 'timetomake' 'course' 'maining...	this recipe calls for the crust to be prebaked...	['prepared pizza crust', 'sausage patty', 'egg...	[None, None, None]
2	all in the kitchen chili	112140	'timetomake' 'course' 'preparation'	this modified version of 'mom's' chili	['ground beef', 'yellow onions', 'diced tomato...	[None, None, None]

▼ Modeling

3 ~~recipe_data_cleaned~~ 59389 ~~recipe_data_cleaned~~ ~~recipe_data_cleaned~~ ~~recipe_data_cleaned~~ ~~recipe_data_cleaned~~

Before I can use the data with TensorFlow, I need to ensure that all features (except for rating) are strings:

```

1 recipe_data_cleaned['diets_combined'] = recipe_data_cleaned['diets_combined'].ast
2 recipe_data_cleaned['recipe_id'] = recipe_data_cleaned['recipe_id'].astype(str)
3 recipe_data_cleaned['name'] = recipe_data_cleaned['name'].astype(str)
4 user_recipe_ratings['user_id'] = user_recipe_ratings['user_id'].astype(str)
5 user_recipe_ratings['recipe_id'] = user_recipe_ratings['recipe_id'].astype(str)
6

```

```

1 #making sure this worked as intended:
2 user_recipe_ratings.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1132367 entries, 0 to 1132366
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   user_id     1132367 non-null  object
1   recipe_id   1132367 non-null  object
2   rating      1132367 non-null  int64
dtypes: int64(1), object(2)
memory usage: 25.9+ MB

```

Next, I create one dataframe with all the necessary features to make things simpler:

```

1 merged_df = user_recipe_ratings.merge(recipe_data_cleaned, on="recipe_id", how="l
2 merged_df.info()

```



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1132367 entries, 0 to 1132366
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   user_id                1132367 non-null  object
1   recipe_id              1132367 non-null  object
2   rating                 1132367 non-null  int64
3   name                   1132367 non-null  object
4   tags                   1132367 non-null  object
5   description             1108857 non-null  object
6   ingredients             1132367 non-null  object
7   diets_combined         1132367 non-null  object
dtypes: int64(1), object(7)
memory usage: 77.8+ MB
```

```
1 merged_df = merged_df.drop(columns=['tags', 'description', 'ingredients'])
2 merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1132367 entries, 0 to 1132366
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  -
0   user_id                1132367 non-null  object
1   recipe_id              1132367 non-null  object
2   rating                 1132367 non-null  int64
3   name                   1132367 non-null  object
4   diets_combined         1132367 non-null  object
dtypes: int64(1), object(4)
memory usage: 51.8+ MB
```

This merged dataframe needs to be turned into a TensorFlow dataset:

```
1 merged_ds = tf.data.Dataset.from_tensor_slices(dict(merged_df))
2
3 recipes_ds = tf.data.Dataset.prefetch(merged_ds, buffer_size=tf.data.AUTOTUNE)

1 print(recipes_ds)

<_PrefetchDataset element_spec={'user_id': TensorSpec(shape=(), dtype=tf.string,
```

Preparing the data for modeling:

```
1 #Selecting the necessary features from the dataset:
2 ratings = (recipes_ds.map(lambda x: {
3     "user_id": x["user_id"],
4     "rating": x["rating"],
5     "name": x["name"],
6     "diets_combined": x["diets_combined"],
7
8     })))
9 recipes = (recipes_ds.map(lambda x:x["name"]))

1 #mapping all values in each column
2 #recipe_ids = ratings.map(lambda x: x["recipe_id"])
3 user_ids = ratings.map(lambda x: x["user_id"])
4 names = ratings.map(lambda x: x["name"])
5 diets = ratings.map(lambda x: x["diets_combined"])
6
7

1 unique_user_ids = merged_df["user_id"].unique().astype(str)
2 unique_names = merged_df["name"].unique().astype(str)
3 unique_diets = merged_df["diets_combined"].unique().astype(str)
4
5

1 #Then shuffle, batch, and cache the training and evaluation data:
2 tf.random.set_seed(42)
3 shuffled = recipes_ds.shuffle(100_000, seed=42, reshuffle_each_iteration=False)
4
5 train = shuffled.take(100_000)
6 test = shuffled.skip(100_000).take(30_000)
7
8 cached_train = train.shuffle(100_000).batch(8192)
9 cached_test = test.batch(4096).cache()
```

✓ Multi-task model:

```

1 # This is multitask recommender model adapted from TensorFlow's website (https://
2 # It conducts both two-tower retrieval and ranking tasks depending on which weig
3 # This model contains no extra feature embeddings (only looks at user_id, recipe
4
5 class UserRecipesModel(tfrs.models.Model):
6
7     def __init__(self, rating_weight: float, retrieval_weight: float) -> None:
8         # We take the loss weights in the constructor: this allows us to instantiate
9         # several model objects with different loss weights.
10
11         super().__init__()
12
13         embedding_dimension = 32
14
15         # User and recipe models.
16         self.recipe_model: tf.keras.layers.Layer = tf.keras.Sequential([
17             tf.keras.layers.StringLookup(
18                 vocabulary=unique_names, mask_token=None),
19             tf.keras.layers.Embedding(len(unique_names) + 1, embedding_dimension)
20         ])
21         self.user_model: tf.keras.layers.Layer = tf.keras.Sequential([
22             tf.keras.layers.StringLookup(
23                 vocabulary=unique_user_ids, mask_token=None),
24             tf.keras.layers.Embedding(len(unique_user_ids) + 1, embedding_dimension)
25         ])
26
27         # A small model to take in user and recipe embeddings and predict ratings.
28         # We can make this as complicated as we want as long as we output a scalar
29         # as our prediction.
30         self.rating_model = tf.keras.Sequential([
31             tf.keras.layers.Dense(256, activation="relu"),
32             tf.keras.layers.Dense(128, activation="relu"),
33             tf.keras.layers.Dense(1),
34         ])
35
36         # The tasks
37         self.rating_task: tf.keras.layers.Layer = tfrs.tasks.Ranking(
38             loss=tf.keras.losses.MeanSquaredError(),
39             metrics=[tf.keras.metrics.RootMeanSquaredError()],
40         )
41         self.retrieval_task: tf.keras.layers.Layer = tfrs.tasks.Retrieval(
42             metrics=tfrs.metrics.FactorizedTopK(
43                 candidates=recipes.batch(128).map(self.recipe_model)
44             )
45         )
46
47
48         # "since we have two tasks and two losses - we need to decide on how importa
49         # We can do this by giving each of the losses a weight, and treating these v
50
51         # The loss weights.

```

```

52     self.rating_weight = rating_weight
53     self.retrieval_weight = retrieval_weight
54
55     def call(self, features: Dict[Text, tf.Tensor]) -> tf.Tensor:
56         # We pick out the user features and pass them into the user model.
57         user_embeddings = self.user_model(features["user_id"])
58         # And pick out the recipe features and pass them into the recipe model.
59         recipe_embeddings = self.recipe_model(features["name"])
60
61         return (
62             user_embeddings,
63             recipe_embeddings,
64             # We apply the multi-layered rating model to a concatenation of
65             # user and recipe embeddings.
66             self.rating_model(
67                 tf.concat([user_embeddings, recipe_embeddings], axis=1)
68             ),
69         )
70
71     def compute_loss(self, features: Dict[Text, tf.Tensor], training=False) -> tf.
72
73         ratings = features.pop("rating")
74
75         user_embeddings, recipe_embeddings, rating_predictions = self(features)
76
77         # We compute the loss for each task.
78         rating_loss = self.rating_task(
79             labels=ratings,
80             predictions=rating_predictions,
81         )
82         retrieval_loss = self.retrieval_task(user_embeddings, recipe_embeddings)
83
84         # And combine them using the loss weights.
85         return (self.rating_weight * rating_loss

```

```

1 #Ranking specialized model (only the ranking task has weight)
2 #Adam optimizer
3 model_1a = UserRecipesModel(rating_weight=1.0, retrieval_weight=0.0)
4 model_1a.compile(optimizer=tf.keras.optimizers.Adam(0.05))
5

```

```

1 #For these models I only have them set to fit for 1 epoch currently, due to compu
2 model_1a.fit(cached_train, epochs=1)
3

```

```

13/13 [=====] - 3350s 256s/step - root_mean_squared_err
<keras.src.callbacks.History at 0x7df566d2e2f0>

```

```

1 metrics = model_1a.evaluate(cached_test, return_dict=True)
2
3

```

```

8/8 [=====] - 1022s 126s/step - root_mean_squared_error
Ranking RMSE: 2.248.

```

```

1 print(f"Retrieval top-100 accuracy: {metrics['factorized_top_k/top_100_categorical']:.3f}")
2 print(f"Ranking RMSE: {metrics['root_mean_squared_error']:.3f}.")

```

- Top-100 accuracy: 3.0e-4
- RMSE: 2.248

```

1 #Retrieval specialized model (only the retrieval task has weight)
2
3 model_1b = UserRecipesModel(rating_weight=0.0, retrieval_weight=1.0)
4 model_1b.compile(optimizer=tf.keras.optimizers.Adam(0.05))

```

```

1 model_1b.fit(cached_train, epochs=1)
2 metrics = model_1b.evaluate(cached_test, return_dict=True)
3
4

```

```

13/13 [=====] - 3250s 248s/step - root_mean_squared_err
8/8 [=====] - 1007s 124s/step - root_mean_squared_error
Ranking RMSE: 4.684.

```

```

1 print(f"Retrieval top-100 accuracy: {metrics['factorized_top_k/top_100_categorical']:.3f}")
2 print(f"Ranking RMSE: {metrics['root_mean_squared_error']:.3f}.")

```

```

Retrieval top-100 accuracy: 0.000.
Ranking RMSE: 4.684.

```

- Top-100 accuracy: 0.000
- RMSE: 4.684

```

1 #joint model (both tasks have weight)
2
3 model_1 = UserRecipesModel(rating_weight=1.0, retrieval_weight=1.0)
4 model_1.compile(optimizer=tf.keras.optimizers.Adam(0.05))

```

```

1 model_1.fit(cached_train, epochs=1)
2 metrics = model_1.evaluate(cached_test, return_dict=True)
3 print(metrics)
4 print(f"Retrieval top-100 accuracy: {metrics['factorized_top_k/top_100_categorical_accuracy']:.3f}.")
5 print(f"Ranking RMSE: {metrics['root_mean_squared_error']:.3f}.")

13/13 [=====] - 3210s 245s/step - root_mean_squared_error: 2.196
8/8 [=====] - 988s 122s/step - root_mean_squared_error: 2.196
{'root_mean_squared_error': 2.195901870727539, 'factorized_top_k/top_1_categorical_accuracy': 0.001}
Retrieval top-100 accuracy: 0.001.
Ranking RMSE: 2.196.

```

- Top-100 accuracy: 0.001
- RMSE: 2.196

✓ Model with extra embeddings:

```

1 #the user model with no additional embeddings:
2 class UserModel2(tf.keras.Model):
3     def __init__(self):
4         super().__init__()
5         self.user_embeddings = tf.keras.Sequential(
6             [tf.keras.layers.StringLookup(vocabulary=unique_user_ids, mask_token=None)], tf.
7         )
8     def call(self, inputs):
9         return self.user_embeddings(inputs["user_id"])

```

```
1 # recipe model with only the diet embedding added
2 class RecipeModel2(tf.keras.Model):
3
4     def __init__(self):
5         super().__init__()
6
7         max_tokens = 10_000
8
9         # self.recipe_embedding = tf.keras.Sequential([
10         #     tf.keras.Sequential(
11         #         [tf.keras.layers.StringLookup(vocabulary=unique_recipe_names, mask_to
12         #         tf.keras.layers.Embedding(len(unique_recipe_ids)+1, 32)
13         #         ])
14         #     ])
15         self.recipe_embedding = tf.keras.Sequential([
16             tf.keras.layers.StringLookup(vocabulary=unique_names, mask_token=None),
17             tf.keras.layers.Embedding(len(unique_names)+1, 32)])
18
19         self.diet_embedding = tf.keras.Sequential([
20             tf.keras.layers.StringLookup(vocabulary=unique_diets, mask_token=None),
21             tf.keras.layers.Embedding(len(unique_diets)+1, 32)])
22
23         self.text_vectorizer = tf.keras.layers.TextVectorization(max_tokens=max_toke
24
25         self.text_vectorizer.adapt(diets)
26         # self.text_vectorizer.adapt(names)
27
28     def call(self, inputs):
29         return tf.concat( [self.recipe_embedding(inputs), self.diet_embedding(inputs)
30
```

```

1 #combined model
2
3
4 #https://blog.searce.com/recommendation-systems-using-tensorflow-recommenders-d7d
5 class RecipeRecommendModel2(tfrs.models.Model):
6
7     def __init__(self, rating_weight, retrieval_weight):
8         super().__init__()
9         embedding_dimension = 32
10        self.query_model = tf.keras.Sequential([UserModel2(), tf.keras.layers.Dense(
11        self.candidate_model = tf.keras.Sequential([RecipeModel2(), tf.keras.layers.Dense(
12        self.rating_model = tf.keras.Sequential(
13            [tf.keras.layers.Dense(256, activation="relu"),
14             tf.keras.layers.Dense(128, activation="relu"),
15             tf.keras.layers.Dense(1)]
16        )
17        self.retrieval_task = tfrs.tasks.Retrieval(
18            metrics=tfrs.metrics.FactorizedTopK(candidates=recipes.batch(128).map
19        )
20        self.rating_task = tfrs.tasks.Ranking(
21            loss=tf.keras.losses.MeanSquaredError(), metrics=[tf.keras.metrics.Ro
22        # The loss weights.
23        self.rating_weight = rating_weight
24        self.retrieval_weight = retrieval_weight
25
26    def call(self, features: Dict[Text, tf.Tensor]) -> tf.Tensor:
27        user_embeddings = self.query_model({"user_id": features["user_id"]})
28        recipe_embeddings = self.candidate_model({"name": features["name"]})
29        return (user_embeddings, recipe_embeddings, self.rating_model(tf.concat([
30
31    def compute_loss(self, features: Dict[Text, tf.Tensor], training=False) -> tf
32
33        ratings = features.pop("rating")
34        user_embeddings, recipe_embeddings, rating_predictions = self(features)
35        # We compute the loss for each task.
36        rating_loss = self.rating_task(labels=ratings, predictions=rating_predict
37        retrieval_loss = self.retrieval_task(user_embeddings, recipe_embeddings)
38        # And combine them using the loss weights.
39        return (self.rating_weight * rating_loss + self.retrieval_weight * retrie
40

```



```

1 model_2 = RecipeRecommendModel2(1, 1)
2 model_2.compile(optimizer=tf.keras.optimizers.Adam(0.05))

1 from keras.callbacks import EarlyStopping
2 es = EarlyStopping(monitor='root_mean_squared_error', patience =2)

1 history = model_2.fit(cached_train, epochs=3, callbacks=es )

```



```

Epoch 1/3
13/13 [=====] - 3101s 236s/step - factorized_top_k/top_
Epoch 2/3
13/13 [=====] - 3145s 241s/step - factorized_top_k/top_
Epoch 3/3
13/13 [=====] - 3126s 239s/step - factorized_top_k/top_

```

```
1 history.history
```

```

{'factorized_top_k/top_1_categorical_accuracy': [0.00023999999393709004,
0.0020099999383091927,
0.019139999523758888],
'factorized_top_k/top_5_categorical_accuracy': [0.00033000000985339284,
0.0024300001095980406,
0.029020000249147415],
'factorized_top_k/top_10_categorical_accuracy': [0.00044999999227002263,
0.003120000008493662,
0.039570000022649765],
'factorized_top_k/top_50_categorical_accuracy': [0.001339999958872795,
0.007689999882131815,
0.08913999795913696],
'factorized_top_k/top_100_categorical_accuracy': [0.0022700000554323196,
0.011950000189244747,
0.12381000071763992],
'root_mean_squared_error': [15.090824127197266,
1.998063564300537,
1.5534210205078125],
'loss': [12518.3125, 11603.45703125, 9435.779296875],
'regularization_loss': [0, 0, 0],
'total_loss': [12518.3125, 11603.45703125, 9435.779296875]}

```

```
1 model_2.evaluate(cached_test)
```

```

8/8 [=====] - 981s 121s/step - factorized_top_k/top_1_c
[3.333333370392211e-05,
6.666666740784422e-05,
6.666666740784422e-05,
0.00019999999494757503,
0.00060000000284984708,
1.3955219984054565,
10845.0078125,
0,
10845.0078125]

```

- RMSE: 1.3955
- Top-100 accuracy: 6.0000e-04 (0.0006)

This was the best performing model in terms of the lowest RMSE value. The top-100 accuracy value is lower than the previous joint model, but considering the sparsity of the data, this is understandable. Perhaps this value could be improved further with other features or tuning.

✓ Efficient Serving (ScaNN)

First, creating a Brute Force retrieval method to compare the retrieval efficiency

```
1 #trying out baseline method for serving:
2
3 # Override the existing streaming candidate source.
4 brute_force = tfers.layers.factorized_top_k.BruteForce(model_2.query_model)
5
6 brute_force.index_from_dataset(
7     recipes.batch(128).map(lambda name: (name, model_2.candidate_model(name)))
8 )
9
```

<tensorflow_recommenders.layers.factorized_top_k.BruteForce at 0x7e7ef53112d0>

```
1 #Looking for user_id to use for example recommendations:
2 user_data.head()
```

	user_id	recipe_id	date	rating	review
0	38094	40893	2003-02-17	4	Great with a salad. Cooked on top of stove for...
1	1293707	40893	2011-12-21	5	So simple, so delicious! Great for chilly fall...
2	8937	44394	2002-12-01	4	This worked very well and is EASY. I used not...
3	126440	85009	2010-02-27	5	I made the Mexican topping and took it to bunk...
4	57222	85009	2011-10-01	5	Made the cheddar bacon topping, adding a sprin...

```
1 %timeit _, names = brute_force({"user_id":tf.constant(["38094"])}), k=3)
```

24.6 ms ± 165 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

Using ScaNN (scalable nearest neighbors) to improve retrieval efficiency:

```
1
2 scann = tfers.layers.factorized_top_k.ScaNN(
3     model_2.query_model,
4     num_leaves=100,
5     num_leaves_to_search=400,
6
7 )
8
1 %timeit _, names = scann({"user_id":tf.constant(["38094"])}), k=3)

3.96 ms ± 27.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Evaluating the two serving methods:

```
1 # Override the existing streaming candidate source.
2 model_2.retrieval_task.factorized_metrics = tfers.metrics.FactorizedTopK(
3     candidates=brute_force
4 )
5 # Need to recompile the model for the changes to take effect.
6 model_2.compile()
7
8
9 %time bf result = model_2.evaluate(cached test, return dict=True, verbose=False)
```