Matching

In [ ]:
```
# Data
import re
import numpy as np
import pandas as pd
import json
from dotmap import DotMap
```

In [ ]:
```
# Models
from sentence_transformers import SentenceTransformer, util
import torch
import faiss
```

In [ ]:
```
# Lexical Models
from fuzzywuzzy import fuzz, process
```

In [ ]:
```
# Translation
from googletrans import Translator
from deep_translator import GoogleTranslator
```

In [ ]:
```
# Parallel procesaing
from joblib import Parallel, delayed
```

In [ ]:
```
# Measures of code speed
import time
```

In [ ]:
```
# helpers
from tqdm import tqdm
from tqdm_joblib import tqdm_joblib
import gc
```

In [ ]:
```
tqdm.pandas()
```

In [ ]:
```
faiss.omp_set_num_threads(1)
```

In [ ]:
```
class NlpPilepine:

    def __init__(self, embeddings_path, data_path, grain_emb_path ,
veggies_emb_path,
                 dairy_emb_path , petrol_emb_path , data_source =
None,matching_data_params_path = None, filtering_atg_codes_path = None):

        self.perform_isolated_matching = False if data_source == 'Inv'
else True
        self.matching_data_params_path = matching_data_params_path
        self.filtering_atg_codes_path = filtering_atg_codes_path


        self.grain_emb = pd.read_csv(grain_emb_path)
        self.veggies_emb = pd.read_csv(veggies_emb_path)
```

```
self.diary_emb=pd.read_csv(dairy_emb_path)self.petrol_emb=pd.read_csv(petrol_emb_p
ath)self.embeddings_data=pd.read_csv(embeddings_path)self.master_data=pd.concat([s
elf.embeddings_data,self.grain_emb,self.veggies_emb,self.diary_emb,self.petrol_emb
])ifself.perform_isolated_matching:passelse:self.embeddings_data=self.master_datas
elf.data=pd.read_csv(data_path,converters=
{'ADG_CODE':self.converter_func})self.device=self.find_device()self.model=self.get
_model()# prepare reference data for isolated
matchingdefprepare_isolated_matching_data(self):withopen(self.matching_data_params
_path,'r')asjson_file:isolated_data_params=json.load(json_file)isolated_data_param
s=DotMap(isolated_data_params)#prepare matching datacolumns=
["cleaned_good_name2","sub_category",'emb','final_cat_emb']self.target_veggies=pd.
concat([self.embeddings_data[(self.embeddings_data['category'].isin(isolated_data_
params.veggies.reference_data.from_category))&
(~self.embeddings_data['sub_category'].isin(isolated_data_params.veggies.reference
_data.not_from_subcategory))&~
((self.embeddings_data['product_name'].str.contains(isolated_data_params.veggies.r
eference_data.product_name_contains,case=False))&
(self.embeddings_data['sub_category']!=isolated_data_params.veggies.reference_data
.product_name_contains))]
[columns],self.veggies_emb[columns],]).reset_index(drop=True)self.target_veggies.l
oc[self.target_veggies.cleaned_good_name2.str.contains('asparagus',case=False),'su
b_category']='Asparagus'self.target_grain=pd.concat([self.embeddings_data[(self.em
beddings_data['category'].isin(isolated_data_params.grain.reference_data.from_cate
gory))&
(~self.embeddings_data['sub_category'].isin(isolated_data_params.grain.reference_d
ata.not_from_sub_category))]
[columns],self.embeddings_data[(self.embeddings_data['sub_category'].isin(isolated
_data_params.grain.reference_data.from_subcategory))]
[columns],self.grain_emb[columns],]).reset_index(drop=True)self.target_diary=pd.co
ncat([self.embeddings_data[(self.embeddings_data['category'].isin(isolated_data_pa
rams.dairy.reference_data.from_category))&
(~self.embeddings_data['sub_category'].isin(isolated_data_params.dairy.reference_d
ata.not_from_subcategory))]
[columns],self.diary_emb[columns]]).reset_index(drop=True)self.target_petrol=pd.co
ncat([self.embeddings_data[self.embeddings_data['category'].isin(isolated_data_par
ams.petrol.reference_data.from_category)]
[columns],self.petrol_emb[columns],]).reset_index(drop=True)self.target_canned_foo
d=self.embeddings_data[self.embeddings_data['category'].isin(isolated_data_params.
canned_food.reference_data.from_category)&
(~self.embeddings_data['sub_category'].isin(isolated_data_params.canned_food.refer
ence_data.not_from_subcategory))]
[columns]self.target_meat_products=self.embeddings_data[self.embeddings_data['cate
gory'].isin(isolated_data_params.meat_products.reference_data.from_category)&
(~self.embeddings_data['sub_category'].isin(isolated_data_params.meat_products.ref
erence_data.not_from_subcategory))][columns]#prepare checking
dataisolated_data_params.veggies.checking_list_veggies.original_list.extend(self.e
mbeddings_data[(self.embeddings_data['category'].isin(isolated_data_params.veggies
.checking_list_veggies.extension_category))|
(self.embeddings_data['sub_category'].isin(isolated_data_params.veggies.checking_l
ist_veggies.extension_subcategory))]["brand"].str.replace('_','
').dropna().unique().tolist())isolated_data_params.grain.checking_list_groats.orig
inal_list.extend(self.embeddings_data[self.embeddings_data['category'].isin(isolat
ed_data_params.grain.checking_list_groats.extension_category)]
["brand"].str.replace('_','
').dropna().unique().tolist())self.checking_list_veggies=isolated_data_params.vegg
ies.checking_list_veggies.original_listself.checking_list_groats=isolated_data_par
ams.grain.checking_list_groats.original_listself.checking_list_diary=isolated_data
_
```

```
_params.dairy.checking_list_diary.original_listself.checking_list_petrol=isolated_
data_params.petrol.checking_list_petrol.original_listself.checking_list_canned_foo
d=isolated_data_params.canned_food.checking_list_canned_food.original_listself.che
cking_list_meat_products=isolated_data_params.meat_products.checking_list_meat_pro
ducts.original_list# isolate data for isolated
matchingdefisolated_data(self,SheetName,HdmData,CheckingList):# loading atg codes
datafiltering_atg_codes_df=pd.read_excel(self.filtering_atg_codes_path,sheet_name=
SheetName,converters={'ATG
CODES':self.converter_func})filtering_atg_codes=filtering_atg_codes_df['ATG
CODES'].to_list()# getting working
dataworking_df=HdmData[HdmData['ADG_CODE'].isin(filtering_atg_codes)]checking_list
=
[x.lower()forxinCheckingList]checking_pattern=r'\b(?:'+'|'.join(checking_list)+r')
\b'working_df=working_df[working_df['GOOD_NAME_CL_TR2'].notna()]working_df=working
_df[~working_df['GOOD_NAME_CL_TR2'].str.lower().str.contains(checking_pattern,rege
x=True)]returnworking_df# ================================= Lexical Matching codes
==================================@staticmethoddefadjusted_token_set_ratio(query,
choice):# Compute the token set ratio
scorescore=fuzz.token_set_ratio(query,choice)# Calculate word
setsquery_words=set(query.lower().split())choice_words=set(choice.lower().split())
# Calculate the proportion of matched
wordsmatched_words=query_words.intersection(choice_words)total_words=query_words.u
nion(choice_words)#word_match_proportion = len(matched_words) / len(total_words) if
total_words else 0unmatched_proportion=(len(total_words)-
len(matched_words))/len(total_words)iftotal_wordselse0# Adjust the score by
multiplying with the word match proportionadjusted_score=score-
unmatched_proportion*20returnadjusted_score# lexical matching
deffind_best_match_token_set_ratio(self,good_name,reference_df):choices=reference_
df['cleaned_good_name2'].tolist()# Proceed with matchingifgood_name.strip():# Use
the standard
scorerbest_match=process.extractOne(good_name,choices,scorer=self.adjusted_token_s
et_ratio)returnbest_matchelse:returnNone# helper function for lexical matching,
process one
rowdefprocess_row_for_lexical_match(self,row,reference_df):good_name=row['GOOD_NAM
E_CL_TR2']# Skip if good_name is just punctuation or
whitespaceifnotgood_nameorgood_name.isspace():returnNonebest_match=self.find_best_
match_token_set_ratio(good_name,reference_df)ifbest_match:best_value,best_score=be
st_match[0],best_match[1]match=reference_df.loc[reference_df['cleaned_good_name2']
==best_value,'sub_category']ifnotmatch.empty:best_category=match.values[0]returngo
od_name,best_value,best_score,best_categoryreturnNone# parallelize lexical
matching
codesdeflexical_matching(self,working_df,reference_df,threshold=70,category_name=N
one):withtqdm_joblib(desc=f"Lexical Matching:
{category_name}",total=len(working_df))asprogress_bar:results=Parallel(n_jobs=-1)
(delayed(self.process_row_for_lexical_match)
(row,reference_df)foridx,rowinworking_df.iterrows())# Filter out None
resultsresults=[resforresinresultsifresisnotNone]# Create a DataFrame from the
resultsdf_results_token_set_ratio_combined=pd.DataFrame(results,columns=
['GOOD_NAME_CL_TR2','sim_cand','max_value','sim_cand_category'])results_df=df_resu
lts_token_set_ratio_combined[df_results_token_set_ratio_combined['max_value']>thre
shold].reset_index(drop=True)remaining_df=df_results_token_set_ratio_combined[df_r
esults_token_set_ratio_combined['max_value']
<=threshold].reset_index(drop=True)returnresults_df,remaining_dfdefencode_data(sel
f,column_list):        """        Encodes a list of text data into embeddings
using a specified model.        Args:            column_list (list): A list of
text items to encode.            model (Model): The model used for encoding the
text.        Returns:            Tensor: The embeddings tensor generated from the
```

```
input text.

"""pool=self.model.start_multi_process_pool([self.device,self.device,self.device,self.device])embeddings=self.model.encode_multi_process(column_list,pool,show_progress_bar=True)# Encoding the text data into embeddingsself.model.stop_multi_process_pool(pool)returnembeddingsdefconvert_to_list(self,embedding_str):        """        Converts a string or numpy array representation of embeddings into a list of floats.        Args:        embedding (str or np.ndarray): The embedding, either as a string representation or a numpy array.        Returns:        list: A list of floats extracted from the input.        """ifisinstance(embedding_str,str):# Use regex to find all numbers in the string (handles scientific notation as well)numbers=re.findall(r"[-+]?\d*\.\d+e[-+]?\d+|\d*\.\d+|\d+",embedding_str)# Convert the extracted strings to floatsreturn[float(num)fornuminnumbers]elifisinstance(embedding_str,np.ndarray):returnembedding_str.tolist()else:raiseTypeError(f"Unsupported type for embedding: {type(embedding_str)}, {(embedding_str)}")defget_embeddings(self,data,column_name,model,ready_embeddings=False,emb_col="emb"):        """        Retrieves and processes embeddings for various categories and names from provided data.        Args:        HDM_data (DataFrame): DataFrame containing GOOD_NAME_CL_TR2 data.        reference_data (DataFrame): DataFrame with sub_category and cleaned_good_name2 data.        model (Model): The model used for encoding the data.        Returns:        tuple: A tuple containing processed query, embeddings, and document data.        """# Ensure the column has no NaN values and is of string typedata[column_name]=data[column_name].fillna('').astype(str)ifready_embeddings:# If embeddings are already calculated and are in the data# Process datadata[f'{emb_col}_open']=data[emb_col].apply(self.convert_to_list)if"sub_category"indata.columns:data['sub_category']=data['sub_category'].apply(lambdax:str(x).lower().strip())else:pass# Embeddings from dataoriginal_data_list=data[column_name].tolist()original_data_emb=torch.tensor(data[f'{emb_col}_open'].tolist()).to(self.device)# send preobtained embeddings to deviceelse:original_data_list=data[column_name].tolist()original_data_emb=self.encode_data(column_list=original_data_list)returnoriginal_data_list,original_data_embdefcalculate_similarity(self,docs,doc_embeddings,query,query_emb):        """        Calculates the cosine similarity between query embeddings and document embeddings using Faiss.        Args:        docs (list): List of documents.        doc_embeddings (numpy.ndarray): Embeddings of the documents, expected numpy array.        query (str): Query identifier.        query_emb (numpy.ndarray): Embeddings of the query, expected numpy array.        Returns:        DataFrame: A DataFrame containing similarity scores and maximum candidate details.        """start=time.time()# Ensure the embeddings are in numpy array formatifnotisinstance(doc_embeddings,np.ndarray):doc_embeddings=doc_embeddings.cpu().numpy()ifnotisinstance(query_emb,np.ndarray):query_emb=query_emb.cpu().numpy()# Normalize the embeddings to use cosine similarityfaiss.normalize_L2(doc_embeddings)faiss.normalize_L2(query_emb)# Create a Faiss index for inner product (cosine similarity)d=doc_embeddings.shape[1]index=faiss.IndexFlatIP(d)index.add(doc_embeddings)# Perform the searchD,I=index.search(query_emb,k=1)# Find the most similar document# Extract the top scores and indicesmax_scores=D.flatten()max_indices=I.flatten()# Create a DataFrame to store resultssimilarity_df=pd.DataFrame(columns=['max_cand','max_score'],index=[query])similarity_df['max_cand']=[docs[idx]foridxinmax_indices]similarity_df['max_score']=max_scoresprint("Similarity Calculation Completed in --- %s seconds ---"%(time.time()-start))returnsimilarity_dfdefisolated_matching(self,ShetName,HdmData,CheckingList,reference_df,model=None,lexical_treshold=70,semantic_treshold=0.5,lexical_match=Tr-
```

```
ue,semantic_matching=True,special_words=None,category_name=None,force_semantic_wor
ds=None):working_df=self.isolated_data(ShetName,HdmData,CheckingList)# Check if
working_df is emptyifworking_df.empty:print("No data to process after initial
isolation.")returnNone# Terminate the function if no data to process#
========================================================================#
Prepare Force Semantic Data#
========================================================================ifforce_sema
ntic_words:# Create a regex pattern for
force_semantic_wordsforce_semantic_pattern='|'.join(force_semantic_words)#
Identify entries that contain
force_semantic_wordsforce_semantic_mask=working_df['GOOD_NAME_CL_TR2'].str.contain
s(force_semantic_pattern,case=False,na=False,regex=True)force_semantic_df=working_
df[force_semantic_mask]# Remove these entries from working_df to exclude from
lexical
matchingworking_df=working_df[~force_semantic_mask]else:force_semantic_df=pd.DataF
rame()# ========================================================================#
Lexical Matching#
========================================================================iflexical_ma
tch:lexical_result_df,remaining_df=self.lexical_matching(working_df,reference_df,t
hreshold=lexical_treshold,category_name=category_name)# Scale the matching score
of the lexical
methodlexical_result_df['max_value']=lexical_result_df['max_value']/100# Prepare
remaining_df for semantic
matchingifremaining_df.emptyandsemantic_matching:remaining_df=force_semantic_df.co
py()else:# Add force_semantic_df to
remaining_dfremaining_df=pd.concat([remaining_df,force_semantic_df],ignore_index=T
rue).drop_duplicates()else:print(f"Lexical matching not required for
{category_name}
category")remaining_df=pd.concat([working_df,force_semantic_df],ignore_index=True)
.drop_duplicates()#
========================================================================#
Semantic Matching#
========================================================================ifsemantic_m
atchingandnotremaining_df.empty:semantic_results=self.semantic_matching(remaining_
df,reference_df,category_matching=False)else:semantic_results=pd.DataFrame(columns
=['GOOD_NAME_CL_TR2','sim_cand','sim_cand_category','max_value'])# Combine
resultsiflexical_matchandsemantic_matching:final_results=pd.concat([lexical_result
_df,semantic_results[['GOOD_NAME_CL_TR2','sim_cand','sim_cand_category','max_value
']]],ignore_index=True).reset_index(drop=True)elifnotsemantic_matchingandlexical_m
atch:final_results=lexical_result_dfelifnotlexical_matchandsemantic_matching:final
_results=semantic_results[['GOOD_NAME_CL_TR2','sim_cand','sim_cand_category','max_
value']]else:final_results=pd.DataFrame()# Merge with original
datacombined_working_df=pd.concat([working_df,force_semantic_df],ignore_index=True
)final_result_merged=final_results.drop_duplicates('GOOD_NAME_CL_TR2').merge(combi
ned_working_df[['GOOD_NAME','GOOD_NAME_CL_TR2']],on='GOOD_NAME_CL_TR2',how='left')
# Check special
wordsifspecial_words:forwordinspecial_words:condition=final_result_merged['GOOD_NA
ME'].str.contains(word,case=False,na=False,regex=True)final_result_merged.loc[cond
ition,'sim_cand_category']=special_words[word]final_result_merged.loc[condition,'s
im_cand']=wordfinal_result_merged.loc[condition,'max_value']=float(0.9)final_resul
t_merged_final=final_result_merged[final_result_merged['max_value']>semantic_tresh
old]returnfinal_result_merged_finaldefsemantic_matching(self,working_data,embeddin
gs_data,category_matching=True):# ================================================
Get Embeddings
========================================================================query,query_emb=
self.get_embeddings(data=working_data,column_name='GOOD_NAME_CL_TR2',model=self.mo
del)# Docs (Good Name on Good
```

```
Name)

docs_GN,doc_GN_emb=self.get_embeddings(data=embeddings_data,column_name='cleaned_good_name2',emb_col='emb',model=self.model,ready_embeddings=True)# Docs (Good Name on Final
Category)docs_cat,doc_cat_emb=self.get_embeddings(data=embeddings_data,column_name='sub_category',emb_col='final_cat_emb',model=self.model,ready_embeddings=True)#
================================================ Matching Good Name on Good Name
================================================print("Good on good
matching")good_on_good_df=self.calculate_similarity(docs_GN,doc_GN_emb,query,query_emb)deldocs_GN,doc_GN_emb,# Select the relevant columns and reset the
indexgood_on_good_df=good_on_good_df[['max_cand','max_score']].reset_index()#
Merge with city scrape
DataFramemerged_results=good_on_good_df.merge(embeddings_data[['sub_category','cleaned_good_name2']].drop_duplicates(subset='cleaned_good_name2'),how='left',right_on='cleaned_good_name2',left_on='max_cand').iloc[:,:-1]# Rename columns for
clarity merged_results=merged_results.rename(columns=
{'level_0':'GOOD_NAME_CL_TR2','max_cand':'sim_cand','max_score':'max_value','sub_category':'sim_cand_category',})delgood_on_good_dfself.empty_device_cache()ifnotcategory_matching:returnmerged_results#
================================================ Matching Good Name on Final
Category ================================================print("Good on category
matching")good_on_cat_df=self.calculate_similarity(docs_cat,doc_cat_emb,query,query_emb)deldocs_cat,doc_cat_emb,query,query_emb,#print("Good Name on Final Category
Completed in \n--- %s seconds ---" % (time.time() - start_time))# Merge and refine
final
resultsres_df=merged_results.merge(good_on_cat_df[['max_cand','max_score']].reset_index().drop_duplicates(subset='level_0'),how='left',right_on='level_0',left_on='GOOD_NAME_CL_TR2').drop(['level_0'],axis=1).drop_duplicates(subset='GOOD_NAME_CL_TR2')res_df=res_df.merge(working_data[['GOOD_NAME','GOOD_NAME_CL_TR2']],on='GOOD_NAME_CL_TR2',how='left')returnres_dfdefconditional_decision_logic(self,all_matching_results):# Apply conditional decision logic for final
resultsall_matching_results['sub_category']=np.where(all_matching_results['max_cand']=="books,
magazines",all_matching_results['max_cand'],np.where(all_matching_results['max_score']>=all_matching_results['max_value'],all_matching_results['max_cand'],all_matching_results['sim_cand_category']))all_matching_results['category_score']=np.where(all_matching_results['max_cand']=="books,
magazines",all_matching_results['max_score'],# Assuming you want to keep the
original max_score for "books,
magazines"np.where(all_matching_results['max_score']>=all_matching_results['max_value'],all_matching_results['max_score'],all_matching_results['max_value']))all_matching_results=all_matching_results[['GOOD_NAME','sub_category','category_score','sim_cand','max_value','GOOD_NAME_CL_TR2']]all_matching_results.sub_category=all_matching_results.sub_category.str.lower()self.master_data.sub_category=self.master_data.sub_category.str.lower()all_matching_results=all_matching_results.merge(self.master_data[['sub_category','category','high_category']].drop_duplicates(subset='sub_category'),on='sub_category',how='left')all_matching_results.sub_category=all_matching_results.sub_category.str.capitalize()all_matching_results=all_matching_results.merge(self.data[['GOOD_NAME','GOOD_NAME_CL_TR','ADG_CODE']],how='left',on='GOOD_NAME')returnall_matching_resultsdefregular_matching_categorization(self):res_df=self.semantic_matching(self.data,self.embeddings_data)res_df=self.conditional_decision_logic(res_df)returnres_dfdefisolated_matching_categorization(self):        """
Processes the matching of GOOD_NAME data with reference data using specified
embeddings.        Args:            None        Returns:            DataFrame: The
final DataFrame after processing matches and merging data.        """#
================================================================================
```

```
================================================
        #                                                    Isolated Matching: Fruits
and Veggies Matching #
================================================================================
================================================fruits_results=self.isolated_match
ing(ShetName='Fruits_veggies',HdmData=self.data,CheckingList=self.checking_list_ve
ggies,reference_df=self.target_veggies,semantic_treshold=0.5,model=self.model,lexi
cal_treshold=61,category_name="Fruits and Veggies",special_words={"չիր|չոր|
չամիչ":"Dried_fruits_and_vegetables","սառ":"Frozen_fruits_vegetables_and_berries"
,"դդմիկ":"Zucchini",'կանաչի':'Greens'},force_semantic_words=
['cherry'])iffruits_resultsisnotNone:# Only execute this code block if
fruits_results is a
DataFrameWorking_data=self.data[~self.data['GOOD_NAME'].isin(fruits_results['GOOD_
NAME'])]else:# Handle the case where fruits_results is Noneprint("No fruits results
to exclude from HDM data.")Working_data=self.data# Empty device cahce beofr
proceeding to whole data matchingself.empty_device_cache()#
================================================================================
================================================#
Isolated Matching: Grain#
================================================================================
================================================grain_results=self.isolated_matchi
ng(ShetName='Grain',HdmData=Working_data,CheckingList=self.checking_list_groats,re
ference_df=self.target_grain,lexical_treshold=79,semantic_matching=True,model=self
.model,semantic_treshold=0.7,special_words=
{'բլղուր':'Bulgur','հնդկաձավար':'Buckwheat','\bձավար\b':'Wheat_groat','սպիտակա
ձավար|
մանի':'Semolina','սիսեռ':'Chickpeas','գարեձավար':'Pearl_barley','հաճար':'Emmer'
,'փոխինձ':'Other_grain','եգիպտացորեն':'Dried_corn','բրինձ':'Rice','գարոխ|
ոլոռ':'Peas','ոսպ':'Lentil','լոբի':'Bean','կորեկաձավար|
կորեկ":"Millet_bran",'սագախոտ|կինուա|կինոա|քինոա|
քինուա':'Quinoa','փաթիլներ':'Flakes','վարսակ':'Oat',"ալյուր":"Flour",},force_sema
ntic_words=['flake','barley'],category_name="Grain")ifgrain_resultsisnotNone:#
Only execute this code block if fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(grain_results['
GOOD_NAME'])]else:# Handle the case where fruits_results is Noneprint("No grain
results to exclude from HDM data.")Working_data=Working_data# Empty device cahce
beofr proceeding to whole data matchingself.empty_device_cache()#
================================================================================
================================================#
Isolated Matching: Mix of grain and fresh veggies#
================================================================================
================================================mixed_results=self.isolated_matchi
ng(ShetName='Mixed_category',HdmData=Working_data,CheckingList=
['marinated'],reference_df=pd.concat([self.target_veggies,self.target_grain]),sema
ntic_treshold=0.6,model=self.model,lexical_treshold=60,category_name="Mixed
Category",force_semantic_words=['cherry'])ifmixed_resultsisnotNone:# Only execute
this code block if fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(mixed_results['
GOOD_NAME'])]else:# Handle the case where fruits_results is Noneprint("No mixed
category results to exclude from HDM data.")Working_data=Working_data# Empty
device cahce beofr proceeding to whole data matchingself.empty_device_cache()#
================================================================================
================================================#
Isolated Matching: Dairy#
================================================================================
================================================dairy_results=self.isolated_matchi
ng(ShetName='Dairy',model=self.model,semantic_matching=True,HdmData=Working_data,C
-
```

```
heckingList
=self.checking_list_diary,reference_df=self.target_diary,lexical_treshold=75,seman
tic_treshold=0.7,special_words=
{"թթվասեր":"Sour_cream","կաթնաշոռ":"Cottage_cheese","պանիր":"Cheese",'սփրեդ':'
Spread','կարագ':'Butter',"մածնաբրդոշ":"other_dairy","\bսերուցք\b":"Cream",},cate
gory_name="Dairy")ifdairy_resultsisnotNone:# Only execute this code block if
fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(dairy_results['
GOOD_NAME'])]else:# Handle the case where fruits_results is Noneprint("No dairy
results to exclude from HDM data.")Working_data=Working_data# Empty device cahce
beofr proceeding to whole data matchingself.empty_device_cache()#
==========================================================================
==============================================#
Isolated Matching: Petrol#
==========================================================================
==============================================
petrol_results=self.isolated_matching(ShetName='Petrol',lexical_treshold=0,HdmData
=Working_data,CheckingList=self.checking_list_petrol,reference_df=self.target_petr
ol,semantic_matching=False,category_name="Petrol")ifpetrol_resultsisnotNone:# Only
execute this code block if fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(petrol_results[
'GOOD_NAME'])]else:# Handle the case where fruits_results is Noneprint("No petrol
results to exclude from HDM data.")Working_data=Working_data# Empty device cahce
beofr proceeding to whole data matchingself.empty_device_cache()#
==========================================================================
==============================================#
Isolated Matching: Canned Food#
==========================================================================
==============================================canned_food_results=self.isolated_
matching(ShetName='Canned_food',model=self.model,semantic_treshold=0.8,semantic_ma
tching=True,HdmData=Working_data,CheckingList=self.checking_list_canned_food,refer
ence_df=self.target_canned_food,lexical_treshold=80,category_name="Canned
Food")ifcanned_food_resultsisnotNone:# Only execute this code block if
fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(canned_food_res
ults['GOOD_NAME'])]else:# Handle the case where canned_food_results is
Noneprint("No canned food results to exclude from HDM
data.")Working_data=Working_data# Empty device cahce beofr proceeding to whole data
matchingself.empty_device_cache()#
==========================================================================
==============================================#
Isolated Matching: Meat #
==========================================================================
==============================================meat_products_results=self.isolate
d_matching(ShetName='meat_products',model=self.model,HdmData=Working_data,Checking
List=self.checking_list_meat_products,reference_df=self.target_meat_products,lexic
al_match=False,semantic_matching=True,semantic_treshold=0.75,special_words=
{"երշիկ":"Boiled_sausage","նրբերշիկ":"Sausages","բաստուրմա|
սուջուխ":"Basturma_sujuk"},category_name="Meat and Meat
Products")ifmeat_products_resultsisnotNone:# Only execute this code block if
fruits_results is a
DataFrameWorking_data=Working_data[~Working_data['GOOD_NAME'].isin(meat_products_r
esults['GOOD_NAME'])]else:# Handle the case where meat_products_results is
Noneprint("No meat products results to exclude from HDM
data.")Working_data=Working_data# Empty device cahce beofr proceeding to whole data
matchingself.empty_device_cache()# ==============================================
Prepare matching data
```

```
============================================================

        # Debug prints to understand the filtering processprint("Initial
reference_data
length:",len(self.embeddings_data))reference_data=self.embeddings_data[~self.embed
dings_data['cleaned_good_name2'].str.lower().isin(self.target_veggies['cleaned_goo
d_name2'].str.lower())]reference_data=reference_data[~reference_data['cleaned_good
_name2'].str.lower().isin(self.target_grain['cleaned_good_name2'].str.lower())]ref
erence_data=reference_data[~reference_data['cleaned_good_name2'].str.lower().isin(
self.target_diary['cleaned_good_name2'].str.lower())]reference_data=reference_data
[~reference_data['cleaned_good_name2'].str.lower().isin(self.target_petrol['cleane
d_good_name2'].str.lower())]reference_data=reference_data.drop_duplicates(subset='
cleaned_good_name2')print("Final reference_data length after dropping
duplicates:",len(reference_data))# ========================================Perform
Regular Matching on the rest of the
data==========================================res_df=self.semantic_matching(Workin
g_data,reference_data)all_matches=pd.concat([fruits_results,grain_results,mixed_re
sults,dairy_results,petrol_results,canned_food_results,meat_products_results,res_d
f],ignore_index=True)#.drop_duplicates(subset='GOOD_NAME_CL_TR2')all_matches=self.c
onditional_decision_logic(all_matches)returnall_matchesdefmatch_category(self):ifs
elf.perform_isolated_matching:self.prepare_isolated_matching_data()res_df=self.iso
lated_matching_categorization()else:res_df=self.regular_matching_categorization()s
elf.data=res_dfreturnself.datadefget_model(self):returnSentenceTransformer("all-
mpnet-base-v2").to(self.device)@staticmethoddeffind_device():        """
Find and return the best available device for computation.        """# Check if
MPS is availableiftorch.backends.mps.is_available():print("Using MPS
device")return"mps"# Check if CUDA is
availableeliftorch.cuda.is_available():print("Using CUDA device")return"cuda"#
Default to CPU if neither MPS nor CUDA is availableelse:print("Using CPU
device")return"cpu"defempty_device_cache(self):        """        Empties the
cache of the current device to free up memory.                This function clears
the cache for either MPS on macOS devices, CUDA on GPU-enabled devices, or
collects garbage if the device is set to CPU.
"""ifself.device=="mps":torch.mps.empty_cache()elifself.device=='cuda':torch.cuda.
empty_cache()else:gc.collect()defconverter_func(self,x):        """
Converts a given value to a zero-padded 4-digit string if the value is numeric.
This function checks if the input `x` is a numeric value and attempts to convert
it to a 4-digit string, padded with leading zeros if necessary.
Args:            x (any): The value to be converted. Can be a number, a string, or
`NaN`.        Returns:            str or any: A zero-padded 4-digit string if `x`
is a numeric value, or the original value if it cannot be converted.
"""ifpd.isna(x):returnxelse:try:ifisinstance(x,float)andx.is_integer():int_x=int(x
)returnf"{int_x:04d}"else:int_x=int(float(x))returnf"
{int_x:04d}"exceptValueError:returnx
```