In [2]:
```
import gensim
import logging
import os
from pprint import pprint

from collections import namedtuple
import psycopg2
import hashlib
from string import digits
from psycopg2 import extras
import numpy as np
import pandas as pd
import datetime


from tqdm import tqdm
tqdm.pandas()
```

```
/home/ubuntu/anaconda3/lib/python3.7/site-packages/psycopg2/__init__.p
y:144: UserWarning: The psycopg2 wheel package will be renamed from re
lease 2.8; in order to keep installing from binary please use "pip ins
tall psycopg2-binary" instead. For details see: <http://initd.org/psyc
opg/docs/install.html#binary-install-from-pypi>.
  """)
```

In [3]:
```
from multiprocessing import Queue, Manager, Process, cpu_count
import queue # To catch queue.Empty exception

import re
from itertools import permutations
```

In [4]:
```
from pandarallel import pandarallel

pandarallel.initialize(progress_bar=True)
```

```
INFO: Pandarallel will run on 32 workers.
INFO: Pandarallel will use Memory file system to transfer data between
the main process and workers.
```

In [5]:
```
def log(func):
    def wrapper(*args, **kwargs):
        print(f"[{datetime.datetime.now()}] Calling {func.__name__}")
        original_result = func(*args, **kwargs)
        return original_result
    return wrapper
```

In [12]:

```
class ProductData:
    """
    Fetches the product data from the database. This product data is us
    The process is:
        1. For the training of the model get from classified product da
            - prod_name
            - [classification_col_name] i.e. 'prod_cat_1'
           by excluding NULL values, and self._MISSING_DATA_NAMES.
           Save it in self.raw_prod_data

        2. For the future prediction get from unclassified product data
           Save it in self.missing_raw_prod_data

        3. Get a list of the classification columns (Distinct names in
           Save it in self.classification_col_list

        4. Clean text with by:
            1. Extract the words from the prod_name with a regex
            2. Iterate over those words and extract the word stems if t
            3. Return the new prod_name as a string separated by whites
    """

#     SPACY_MODEL_NAME = 'de_core_news_sm'
#     POS_TO_EXCLUDE = []# 'PUNCT', 'SPACE', 'ADP', 'NUM']

    _MIN_WORD_LENGTH_TO_SPLIT = 8
    _MISSING_DATA_NAMES = ['not given', 'nicht zugeordnet'] # case inse
    _CLASSIFICATION_CASES = 30 # How many cases should a category have

    classification_col_list = None

    raw_prod_data = None
    missing_raw_prod_data = None
    clean_prod_data = None
    clean_missing_prod_data = None

    context = None
    clean_context = None

    accuracy = None

    def __init__(self, test=False):
        """
        :param classification_col_name: Which is the second column that
                                        The classification will be done
        :param test: [True/False]. If true - gets a sample of 300 rows
                     multiprocessing for the data cleaning, because of
        """
        self.test = test
        if test:
            self.LIMIT = 300
        else:
            self.LIMIT = 100000000

        # SQL Queries
        self.raw_prod_data = self._get_raw_prod_data()
```

```
        self._set_clean_prod_data()

    def _get_classification_col_list(self):
        """Get a list of the distinct columns to be classified"""
        walmart_amazon_path = (
            r'/home/ubuntu/jupyter/ServerX/1_Standard Data Integration/S
            r'/Unprocessed Data/product_samples/walmart_amazon'
        )

        walmart_data = pd.read_csv(
            os.path.join(walmart_amazon_path, 'walmart.csv'),
            sep = ',',
            usecols = ['custom_id', 'title', 'brand','longdescr', 'price
        )
        cat_counts = walmart_data['groupname'].value_counts()
        classification_col_list = cat_counts[cat_counts > self._CLASSIFI
        self.classification_col_list = classification_col_list
        return classification_col_list

    @log
    def _get_raw_prod_data(self):
        walmart_amazon_path = (
            r'/home/ubuntu/jupyter/Hadoco/1_Standard Data Integration/Sa
            r'/Unprocessed Data/product_samples/walmart_amazon'
        )
        raw_prod_data = pd.read_csv(
            os.path.join(walmart_amazon_path, 'walmart.csv'),
            sep = ',',
            usecols = ['custom_id', 'title', 'brand','longdescr', 'price
        )
        raw_prod_data = raw_prod_data[raw_prod_data['groupname'].isin(se

        raw_prod_data['prod_name'] = raw_prod_data['title'] + ' ' + raw_
        raw_prod_data = raw_prod_data[['prod_name', 'groupname']]

        return raw_prod_data

    def word_extractor(self, text):
        try:
            words = [word for word in re.findall(r"[\w']+", text)]
        except:
            words=[]
        return words

    def word_splitter(self, word):
        """
        If the length of the word is above self._MIN_WORD_LENGTH_TO_SPLI

        Process:
            char_split.split_compound returns an array with [prob, word1
            If the first split succeeds (prob > 0)
                stop
            Else
                if first word can be split (prob > 0)
                    split it
                else
```

```
                    append the word to parts

                if second word can be split (prob > 0)
                    split it
                else
                    append the word to parts

        !!! NB:
        This works only for 3 Stems max and is not perfect

        """
        if len(word) > self._MIN_WORD_LENGTH_TO_SPLIT:

            parts = []

            res = np.array(char_split.split_compound(word))
            if float(res[0][0]) > 0:
                parts = list(res[0][1:])
            else:
                if char_split.split_compound(res[0][1])[0][0] > 0:
                    parts += char_split.split_compound(res[0][1])[0][1:
                else:
                    parts += [res[0][1]]

                if char_split.split_compound(res[0][2])[0][0] > 0:
                    parts += char_split.split_compound(res[0][2])[0][1:
                else:
                    parts += [res[0][2]]
        else:
            parts = [word]

        return parts

    def _clean_text(self, text):
        words = self.word_extractor(text)
#         tokens = [word_splitter(word) for word in words]
#         tokens = [item for sublist in tokens for item in sublist]
        cleaned_text = ' '.join(words)

        return cleaned_text


    @log
    def _set_clean_prod_data(self):

        self.clean_prod_data = self.raw_prod_data.copy()

        if self.test:
            self.clean_prod_data.prod_name = self.clean_prod_data.prod_
                lambda x: self._clean_text(x))
        else:
            self.clean_prod_data.prod_name = self.clean_prod_data.prod_
                lambda x: self._clean_text(x))
```

In [17]:

```
from sklearn.model_selection import train_test_split
from multiprocessing import cpu_count

class Model:

    # Model
    model = None
    X_train, X_test, y_test = None, None, None

    # Data settings
    _TEST_SIZE = 0.2

    # Results
    accuracy, accuracy_table = None, pd.DataFrame()

    def __init__(self,
                 data_type='clean',
                 test=False):
        """
        :param data_type: Should the model be trained on the ['clean',
        """
        self.data_type = data_type
        self.product_data = ProductData(test=test)


    def _build_context(self, prod_data_array):
        context = [
            [classific_col] + prod_name.split(' ')
            for prod_name, classific_col in prod_data_array
        ]
        return context

    @log
    def _get_train_test_data(self):

        # Split data
        prod_data = getattr(self.product_data, f'{self.data_type}_prod_
        X_train, X_test, y_train, y_test = train_test_split(prod_data['
                                                            prod_data['
                                                            test_size=s
                                                            random_stat
                                                            stratify=pr


        # Build context
        ## Train data
        X_train = [
            prod_name.split(' ') + [classific_col]
            for prod_name, classific_col in zip(X_train, y_train)
        ]

        return X_train, X_test, y_test

    @log
    def _calculate_accuracy(self, X_test, y_test, predict_top_n, thresh
        """
        1. Iterate through zip(X_test, y_test)
```

```
        2. predict the class
        3. If match - true + 1 else false + 1
        4. Calculate the matches per class and put them in accuracy_tab
        """
        sep = '||-||'
        n_true, n_false = 0, 0

        fp_combos = list(map(list, permutations(self.product_data.class
        tp_combos = [[col, col] for col in self.product_data.classifica

        accuracy_dict = {
            f'{y}{sep}{y_hat}': 0
            for y, y_hat in tp_combos + fp_combos
        }

        accuracy_table = pd.DataFrame(index=self.product_data.classific
                                      columns=self.product_data.classif

        q = Queue()
        manager = Manager()
        [q.put(ix) for ix in range(len(X_test))]
        y_hat_dict = manager.dict()

        consumer_list = []
        for i in range(cpu_count()):
            consumer = Process(target=self._predict_multiprocess, args=


            consumer.start()
            consumer_list.append(consumer)

        [consumer.join() for consumer in consumer_list]
        y_hat_dict = dict(y_hat_dict)

        for ix in range(len(X_test)):
            y_hat = y_hat_dict[ix]
            y = y_test[ix]
            if y_hat == y:
                n_true += 1
            else:
                n_false += 1

            accuracy_dict[f'{y}{sep}{y_hat}'] += 1

        accuracy = {
            'n_true': n_true,
            'n_false': n_false
        }


        for k in accuracy_dict:
            k_true, k_pred = k.split('||-||')
            accuracy_table.loc[k_true, k_pred] = accuracy_dict[k]

        return accuracy, accuracy_table

    def _predict_multiprocess(self, X_test, y_test,
```

```
                            q, y_hat_dict,
                            predict_top_n, threshold):

        while not q.empty():
            try:
                ix = q.get(True, 0.05)
                case = X_test[ix]
                y_hat, _  = self.predict(case, predict_top_n, threshold

                y_hat_dict[ix] = y_hat

            except queue.Empty:
                pass


    @log
    def fit(self,
            size=100,
            alpha=0.025,
            window=5,
            min_count=20,
            max_vocab_size=None,
            sample=0.001,
            workers=cpu_count(),
            min_alpha=0.0001,
            sg=0,
            hs=0,
            negative=5,
            ns_exponent=0.75,
            cbow_mean=1,
            iter=100,
            null_word=0,
            trim_rule=None,
            sorted_vocab=1,
            batch_words=10000,
            compute_loss=False,
            max_final_vocab=None,

            predict_top_n=150,
            threshold=0.1,
           ):
        """
        1. Get the train and test data
        2. Build the model
        3. Calculate the accuracy

        """

        # 1
        X_train, X_test, y_test = self._get_train_test_data()

        # 2
        self.model = gensim.models.Word2Vec(X_train,

                                            size=size,
                                            alpha=alpha,
                                            window=window,
```

```
                                                min_count=min_count,
                                                max_vocab_size=max_vocab_si
                                                sample=sample,
                                                workers=workers,
                                                min_alpha=min_alpha,
                                                sg=sg,
                                                hs=hs,
                                                negative=negative,
                                                ns_exponent=ns_exponent,
                                                cbow_mean=cbow_mean,
                                                iter=iter,
                                                null_word=null_word,
                                                trim_rule=trim_rule,
                                                sorted_vocab=sorted_vocab,
                                                batch_words=batch_words,
                                                compute_loss=compute_loss,
                                                max_final_vo

        # 3
        self.accuracy, self.accuracy_table = self._calculate_accuracy(X_
                                                                       y_
                                                                       p
                                                                       t


        # 4 save
        self.X_train, self.X_test, self.y_test = X_train, X_test, y_tes

    def predict(self, text, predict_top_n, threshold):
        classification_score = {classification_col: 0 for classificatio
        for token in text.split(' '):
            try:
                scores = np.array(self.model.wv.most_similar(token, topn

                for classification_col in self.product_data.classificat
                    if classification_col in scores[:, 0] and float(sco
                        classification_score[classification_col] += floa
            except KeyError:  # not in vocab
                pass

        best_match = max(classification_score, key=classification_score
        best_score = classification_score[best_match]

        return best_match, best_score

    @log
    def predict_all(self):
        """
        Using the model generated predict every class of the missing da
        Return a pandas data frame with
            prod_name, predicted_class
        """
        raise NotImplementedError("TODO")
        return
```

In [13]:
```
data = ProductData(True)
```

```
[2020-02-03 14:07:53.734797] Calling _get_raw_prod_data
[2020-02-03 14:07:53.860823] Calling _set_clean_prod_data
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
```

In [18]:
```
m = Model(test=False)
```

100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67
100.00%	67 / 67

In [19]:
```
m.fit(workers=1, min_count=8, threshold=0.1, window=5, predict_top_n=20(
```

```
[2020-02-03 14:08:56.267867] Calling fit
[2020-02-03 14:08:56.268001] Calling _get_train_test_data
[2020-02-03 14:09:11.981107] Calling _calculate_accuracy
```

In [20]:
```
print(m.accuracy)
display(m.accuracy_table)
```

```
{'n_true': 121, 'n_false': 309}
```

	Electronics - General	Stationery & Office Machinery	MP3 Accessories	Printers	USB Drives	Networking	Computers	H Dri
Electronics - General	15	4	11	5	7	33	12	
Stationery & Office Machinery	1	22	2	11	1	0	1	
MP3 Accessories	2	0	10	3	2	1	2	
Printers	0	1	0	16	0	0	0	
USB Drives	0	0	0	0	11	0	0	
Networking	0	0	0	0	1	8	1	
Computers	1	0	0	0	2	1	0	
Hard Drives	1	0	0	2	2	1	0	
TV Accessories	1	0	0	0	0	2	0	
Car Stereos	0	0	0	0	0	4	0	
Mice	0	0	0	0	0	0	1	
Photography - General	1	0	0	1	2	0	0	
Memory	0	0	0	0	0	0	0	
Software	1	0	0	0	1	0	0	
Digital Cameras	0	0	0	0	0	0	0	
Furniture	0	0	0	1	0	0	1	