Configuration Manual

MSc Research Project
Programme Name

Adelola Adebo
Student ID: X19134711

School of Computing
National College of Ireland

Supervisor: Dr Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Adelola Adebo

Student ID: X19134711

Programme: Data Analytics **Year:** 2020

Module: Msc Research Project

Lecturer: Dr Vladimir Milosavljevic

Submission Due Date: 17th August 2020.

Project Title: A Natural Language Processing Approach to a Skincare Recommendation Engine

Word Count: ..... **Page Count:** 17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: .....

Date: .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Adelola Adebo
Student ID: X19134711

1 Introduction

This configuration manual provides detailed documentation on the I.T solution deployed as part of the research thesis, A Natural Language Processing Approach to a Skincare Recommendation. The document covers all steps taken to implement the solution.

2 Hardware Setup

The deep learning models used in this research was implemented using Google Collaboratory cloud machine due to the large dataset that we used. The specifications for the host machine are:

- Operating system: MacOS 10.15.5
- Processor: 2.5 GHz Dual-Core Intel Core i5
- RAM: 8 GB
- Hard drive: 500 GB

3 Software environment and library dependencies

Python 3.7 is used for scripting. Jupyter notebook which is part of the Anaconda Python package is used for the overall implementation of this project. Other library dependencies used are:

- Numpy
- Pandas
- NLTK
- Genism
- Matplotlib
- Scikit-learn
- Tensorflow
- Keras

4 Datasets

A total of three datasets was used for this research. The first dataset used is skincare comments from Reddit which was sourced from Google BigQuery and downloaded as csv files. The second dataset was skincare products sourced from here and the third is a list of skincare ingredients sourced from here. All three datasets were ethically sourced.

A screenshot of downloading the Reddit comments from Google BigQuery.

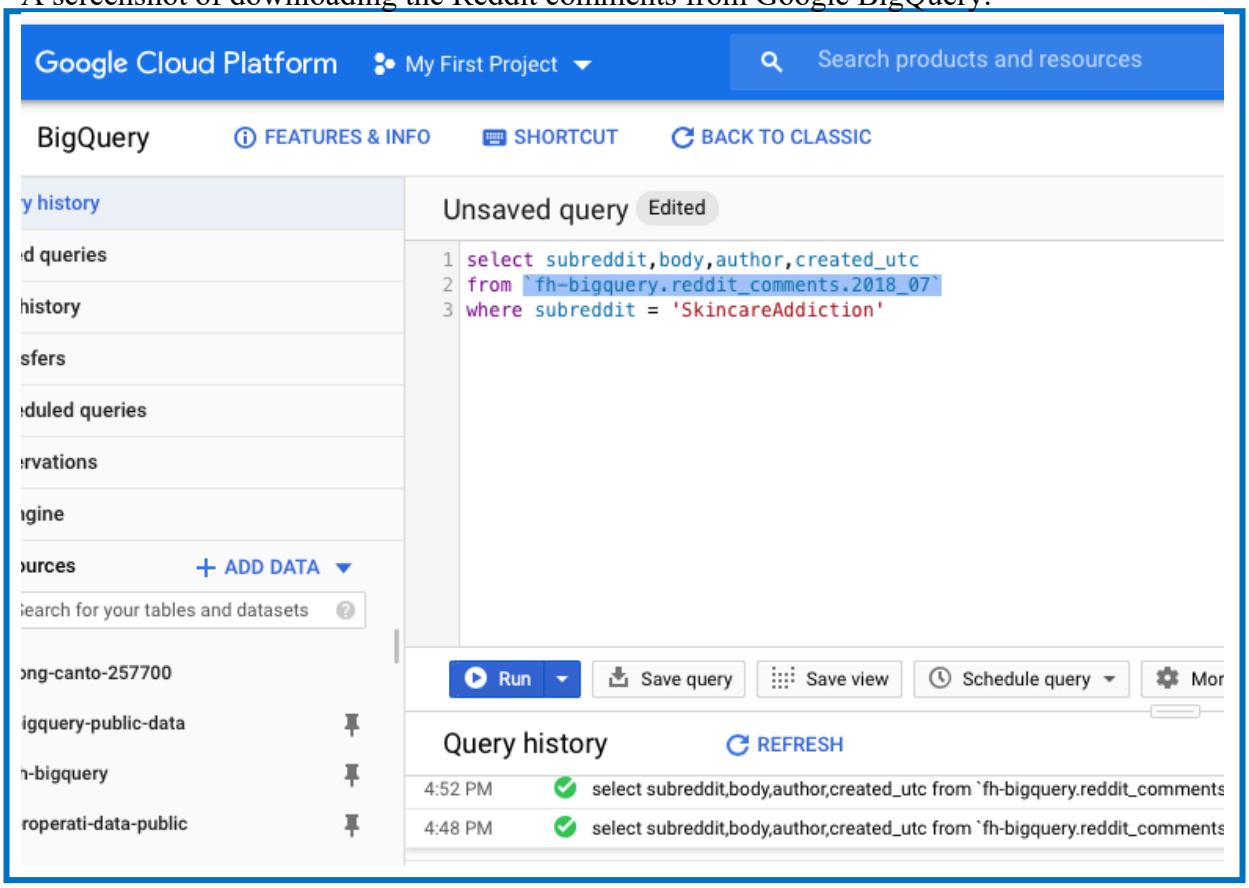


Figure 1: Extracting the user review comments from Google BigQuery

The following sections show the project implementation.

5 Data cleaning and pre-processing

Loading and Cleaning the data: Removing punctuations, stopwords, HTML tags, special characters, and dropping irrelevant comments, etc was performed before modelling.

Importing libraries

In [1]:

```
import pandas as pd
import pickle
import spacy
import re
import string
import nltk

from glob import glob

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem import WordNetLemmatizer

from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer

```


Load the data

In [ ]:

```
csv_list = glob('reddit*.csv')
```

In [ ]:

```
csv_list
```

In [ ]:

```
def clean_reviews(text):

    text = remove_stopwords(text)
    text = re.sub('http://\S+|https://\S+', '', text)
    text = re.sub('\n\n', '', text)
    text = re.sub('\n', '', text)
    text = re.sub('😍', '', text)
    text = re.sub('❤️', '', text)
    text = re.sub('😉', '', text)
    text = re.sub('☺️', '', text)
    text = re.sub('👌', '', text)
    text = re.sub('💥', '', text)
    text = re.sub('‿ ↼ ‿ ↽','', text)
    text = re.sub('❤️❤️', '', text)
    text = re.sub('❤️', '', text)
    text = re.sub('🏻', '', text)
    text = re.sub('❤', '', text)
    text = re.sub('✨', '', text)
    text = re.sub('\n\n\n\n', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.translate(str.maketrans('','','’”•–'))
    text = remove_stopwords(text)
    text = wordnet_lemmatizer.lemmatize(text)


    return text



```

In [ ]:

```
def automate(csv_list,):
    all_reviews = []
    for file in csv_list:
        df = pd.read_csv(file)
        df = df.drop(columns=['subreddit','created_utc'])



```

In [ ]:

```
def automate(csv_list,):
    all_reviews = []
    for file in csv_list:
        df = pd.read_csv(file)
        df = df.drop(columns=['subreddit','created_utc'])
        |       df = df.drop(df[df['author'] == 'AutoModerator'].index)
        df = df.drop(df[df['author'] == 'sunscreenpuppy'].index)
        df = df.drop(df[(df['author'] == 'removed') & (df['author'] == 'deleted')].index)
        df = df.drop(df[df['body'] == 'removed'].index)
        df = df.drop(df[df['body'] == 'deleted'].index)


        review_data = df.drop(columns=['author'])

        review_data['clean_review'] = review_data['body'].apply(lambda x: clean_reviews(x))

        cleaned_reviews = list(review_data['clean_review'])

        all_reviews.append(cleaned_reviews)

    return all_reviews
```

In [ ]:

```
all_reviews = automate(csv_list)
```

In [ ]:

```
all_reviews[3][58260]
```

In [ ]:

```
pd.to_pickle(all_reviews, 'all_reviews.pkl')
```

Figure 2: Steps to clean the data.

6 Feature Engineering

After cleaning the datasets, we used custom functions to match reviews with keywords from the dictionary created.

Matching

In [ ]:
```
skincare_ingredients = pd.read_pickle('skincare_ingrednts.pkl')
product_names = pd.read_pickle('product_names.pkl')
```

In [ ]:
```
skincare_ingredients
```

In [ ]:
```
product_names
```

In [ ]:
```
skin_type = ['oily', 'dry', 'combination', 'normal', 'sensitive', 'acne',
             'hyperpigmentation', 'anti-aging', 'discolouration', 'breakout']
```

In [ ]:
```
extend_ingredients = ['ha', 'aha', 'bha','micellar water' 'vit c', 'pie', 'sunscreen', 'spf', 'snail cream',
           'chemical exfoliation', 'irritation', 'double cleansing', 'cleanser oil', 'hydrating', 'retinoid',
          'exfoliate', 'exfoliating','retin-a', 'aloe', 'accutane', 'tret']
extend_products = ['to', 'cetaphil', 'cerave cream', 'cerave pm',
              'cetaphil non-soap bar','The ordinary's 10 niacinamide solution',
              'Paula's Choice skin perfecting 2 BHA', 'Cetaphil moisturizing lotion',
              'Cetaphil oil control 30 spf moisturizing sunscreen PM', 'cerave foaming facial cleanser',
              'Cetaphil moisturizing cream',
              'the ordinary cold pressed rosehip oil',
              'the ordinary 5 lactic acid solution',
              'inkey list under eye caffeine treatment',
              'Hada Labo premium lotion',
              'TO Niacinamide',
              'Hada Labo sunscreen',
              'cosrx snail mucin',
              'Cerave healing ointment',
              'cosrx snail essence',
              'Cerave Resurfacing Retinol',
              'Paula Choice 2 BHA',
              'Thayer Witch Hazel Toner',
              'LaNeige Water Bank Essence',
              'Drunk Elephant C-Firma Day Serum',
              'Boscia Makeup Breakup Cleansing Oil',
              'Dr Jart Ceramidin Lotion',
                 'ole henriksen',
                 'shiseido sunscreen',
                 'Ole Henriksen Truth Serum',
                 'Ole Henriksen Banana Bright',
                 'Olay Regenerist Retinol24',
                 'aztec secret clay mask',
                 'la rocheposay anthelios antishine spf50'

]
```

In [ ]:
```
#extend lists
for item in extend_ingredients:
    skincare_ingredients.append(item)

for item in extend_products:
    product_names.append(item)
```

In [ ]:
```
def skin_match(x, match):
    splitreview = x.split(" ")
    setreview = set(splitreview)
    setmatch = set(match)

    matches = setreview.intersection(setmatch)
    matches = list(matches)

    if len(matches) >= 1:
        match_string = ','.join(matches)
        return match_string
    else:
        return ''
```

In [ ]:
```
def pi_match(x, match):
    matches = []
    for product in match:
        splitproduct = product.split(" ")
        if len(splitproduct) > 1:
            if product.lower() in x.lower():
                matches.append(product)
        else:
            splitreview = set(x.split(" "))
            splitproduct = set(splitproduct)

            m = splitproduct.intersection(splitreview)
            m = list(m)
            if m == []:
                continue
            else:
                matches.append(str(m[0]))


    if len(matches) >= 1:
        match_string = ','.join(matches)
        return match_string
    else:
        return ''
```

In [ ]:
```
final_df = pd.DataFrame(columns=['review','skin_type','products','ingredients'])
final_df
```

In [ ]:
```
for review in all_reviews[18]:
    review_items = []
    review_items.append(review)
    review_items.append(skin_match(review, skin_type))
    review_items.append(pi_match(review, product_names))
    review_items.append(pi_match(review, skincare_ingredients))
```

In [ ]:
```
for review in all_reviews[18]:
    review_items = []
    review_items.append(review)
    review_items.append(skin_match(review, skin_type))
    review_items.append(pi_match(review, product_names))
    review_items.append(pi_match(review, skincare_ingredients))

    final_df.loc[len(final_df)] = review_items

```

In [ ]:
```
len(final_df)
```

In [ ]:
```
#to check how many items were matched. Just change the value in the dataframe

s = list(final_df['ingredients'])
len([i for i in s if i != ''])
```

check matches across skin, product, and ingredients. change variable name from t

In [ ]:
```
t = final_df[final_df['ingredients'] != '']
```

In [ ]:
```
t = t[final_df['skin_type'] != '']
```

In [ ]:
```
t = t[final_df['products'] != '']
```

In [ ]:
```
t = t.reset_index()
t = t.drop(columns=['index'])
t
```

In [ ]:
```
t.iloc[4]['review']
```

Figure 3: Matching reviews with keywords from the skincare dictionary created.

All reclassified comments are joined into one final dataframe. The final output of keyword matching.

Joining all the reclassified comments into one dataframe

In [200]:
```
all_matches = glob('newmatching*.csv')

list_ = []
for file in all_matches:
    df = pd.read_csv(file, lineterminator='\n')
    list_.append(df)
final_match = pd.concat(list_, ignore_index=True)
```

In [201]:
```
final_match
```

Out[201]:

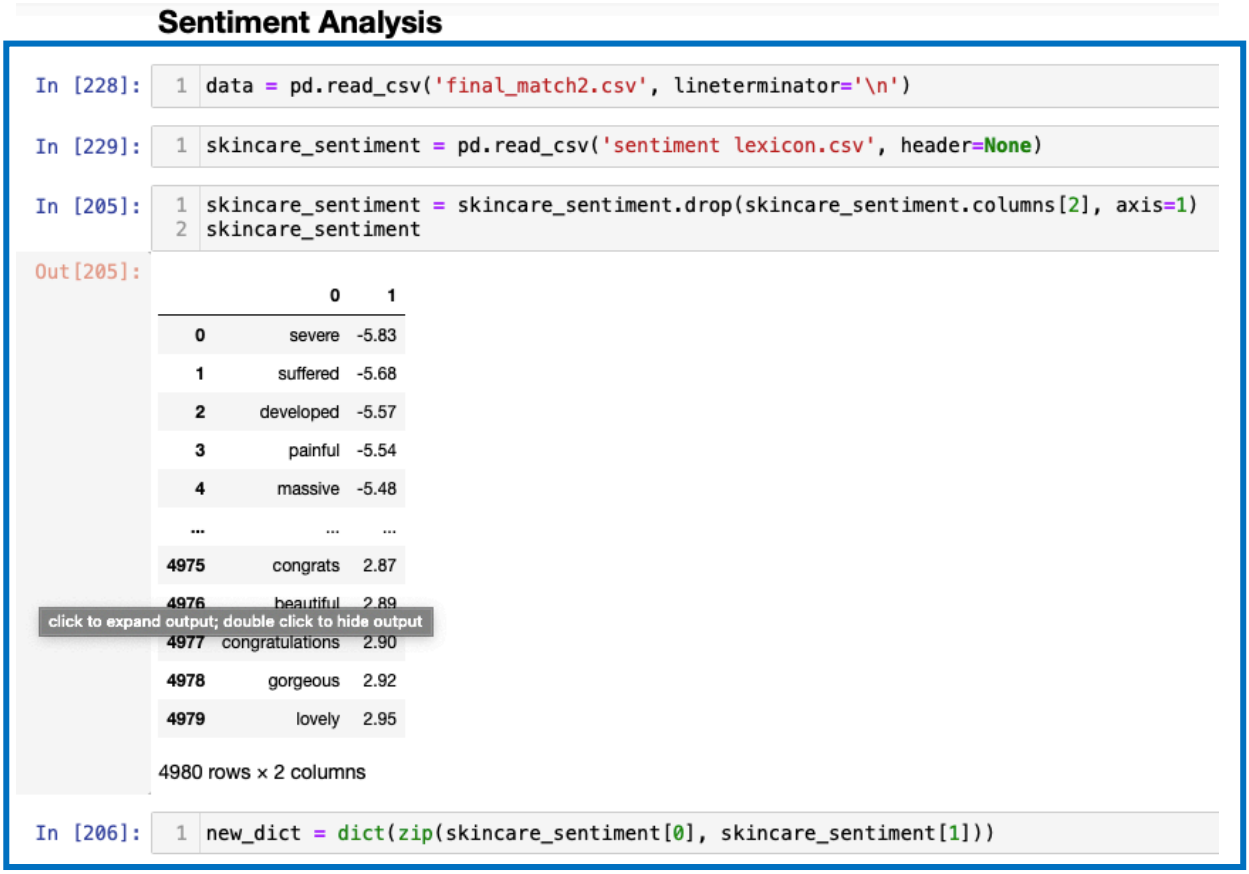
	Unnamed: 0	review	skin_type	products	ingredients
0	0	moisturize every day things aveeno drugstorebu...	acne	Vitamin C Serum,The Ordinary	niacinamide,salicylic acid,vitamin C
1	1	similar routines skin typeconcerns going 28 ye...	dry,acne	Salicylic Acid Cleanser	benzoyl peroxide,retinol,salicylic acid,vitami...
2	2	say anything else primary care appointment ja...	dry,sensitive	cetaphil	hyaluronic acid,soap
3	3	drugstore cleansers general tend harsh irritat...	acne,oily	cerave	fragrance,menthol,salicylic acid
4	4	acne pretty bad growing face back back somewha...	acne,sensitive	Cerave Foaming Cleanser,cerave	fragrance
...	...	...	...	...	...
23118	1151	dry skin burn lotion applied even without alco...	dry	cerave,Cerave healing ointment	alcohol,coconut,coconut oil
23119	1152	thayers hydrating toner nothing acne cerave cl...	acne	cerave	salicylic acid
23120	1153	sunscreenfor much products approach multiple h...	dry	cerave	amino acids,squalane,urea
23121	1154	ive used glow tonic impressed formulated prope...	acne	Glow Tonic	ascorbic acid,fragrance,glycolic acid,salicyli...
23122	1155	cerave cleanser moisturizer treat acne acne sp...	acne	cerave	salicylic acid,retinoid

23123 rows × 5 columns

Figure 4: Final output of reclassified comments.

Sentiment Analysis

We perform sentiment analysis to infer ratings. The sentiment scores were derived using the VADER library in NLTK. The VADER sentiment lexicon was first updated with word sentiment scores from the SocialSent sentiment lexicon.



Sentiment Analysis

```
In [228]:
data = pd.read_csv('final_match2.csv', lineterminator='\n')

In [229]:
skincare_sentiment = pd.read_csv('sentiment lexicon.csv', header=None)

In [205]:
skincare_sentiment = skincare_sentiment.drop(skincare_sentiment.columns[2], axis=1)
skincare_sentiment
```

Out[205]:

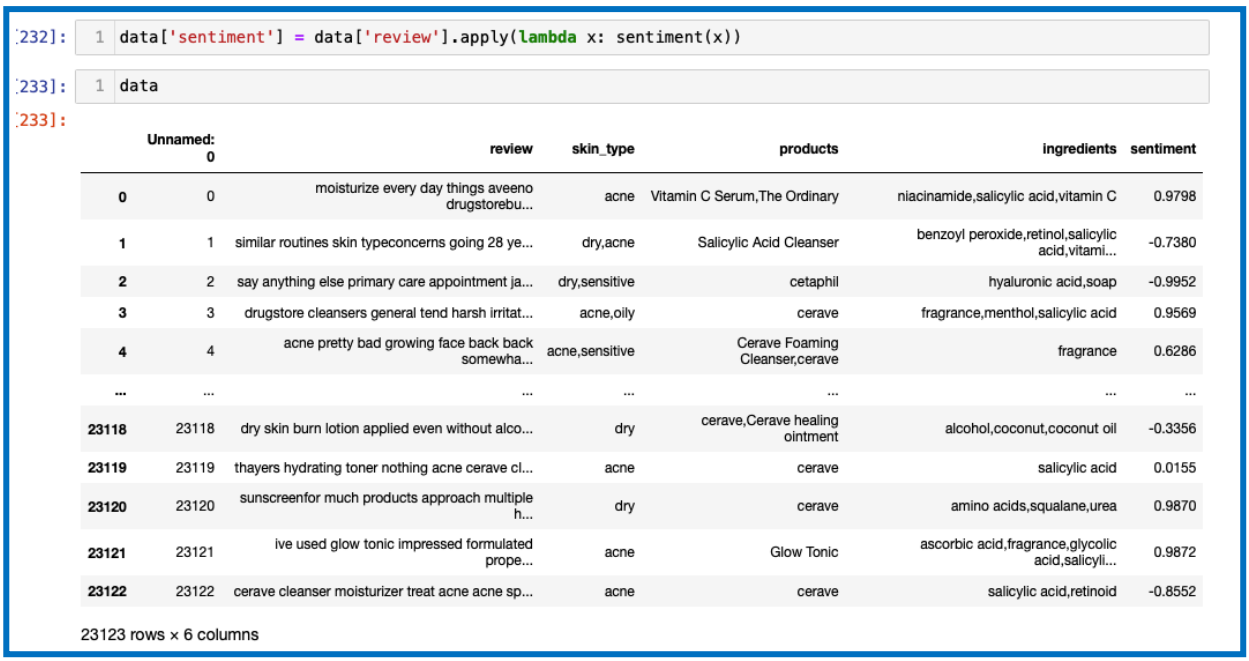
	0	1
0	severe	-5.83
1	suffered	-5.68
2	developed	-5.57
3	painful	-5.54
4	massive	-5.48
...	...	...
4975	congrats	2.87
4976	beautiful	2.89
4977	congratulations	2.90
4978	gorgeous	2.92
4979	lovely	2.95

click to expand output; double click to hide output

4980 rows × 2 columns

```
In [206]:
new_dict = dict(zip(skincare_sentiment[0], skincare_sentiment[1]))
```

Figure 5: Updating VADER sentiment lexicon with SocialSent sentiment lexicon



```
232]:
data['sentiment'] = data['review'].apply(lambda x: sentiment(x))

233]:
data
```

233]:

	Unnamed: 0	review	skin_type	products	ingredients	sentiment
0	0	moisturize every day things aveeno drugstorebu...	acne	Vitamin C Serum,The Ordinary	niacinamide,salicylic acid,vitamin C	0.9798
1	1	similar routines skin typeconcerns going 28 ye...	dry,acne	Salicylic Acid Cleanser	benzoyl peroxide,retinol,salicylic acid,vitami...	-0.7380
2	2	say anything else primary care appointment ja...	dry,sensitive	cetaphil	hyaluronic acid,soap	-0.9952
3	3	drugstore cleansers general tend harsh irritat...	acne,oily	cerave	fragrance,menthol,salicylic acid	0.9569
4	4	acne pretty bad growing face back back somewha...	acne,sensitive	Cerave Foaming Cleanser,cerave	fragrance	0.6286
...	...	...	...	...	...	...
23118	23118	dry skin burn lotion applied even without alco...	dry	cerave,Cerave healing ointment	alcohol,coconut,coconut oil	-0.3356
23119	23119	thayers hydrating toner nothing acne cerave cl...	acne	cerave	salicylic acid	0.0155
23120	23120	sunscreenfor much products approach multiple h...	dry	cerave	amino acids,squalane,urea	0.9870
23121	23121	ive used glow tonic impressed formulated prope...	acne	Glow Tonic	ascorbic acid,fragrance,glycolic acid,salicyli...	0.9872
23122	23122	cerave cleanser moisturizer treat acne acne sp...	acne	cerave	salicylic acid,retinoid	-0.8552

23123 rows × 6 columns

Figure 6: Performing sentiment analysis using the VADER library in NLTK.

After deriving sentiment scores, we created a user-item matrix. The code below shows creating the final structured dataset used for modelling.

Creating user-item matrix

In [2]:

```
user_id = []
product_id = []
sentiment = []
product_name = []
skins = []
reviews = []

for user in range(len(data['review'])):
    products = data.iloc[user,2]
    skin = data.iloc[user,1]
    sent = data.iloc[user,4]
    review = data.iloc[user,0]
    ingredients = data.iloc[user,3]
    split_ing = ingredients.split(',')
    split_products = products.split(',')
    for item in split_products:
        item = item.strip()
        user_id.append(user+1)
        reviews.append(review)
        skins.append(skin)
        sentiment.append(sent)
        if item in product_name:
            item_index = product_name.index(item)
            p_id = product_id[item_index]
            product_name.append(item)
            product_id.append(p_id)
        else:
            product_name.append(item)
            product_id.append(len(product_id)+1)
    for ing in split_ing:
        ing=ing.strip()
        user_id.append(user+1)
        reviews.append(review)
```

```
            item_index = product_name.index(ing)
            p_id = product_id[item_index]
            product_name.append(ing)
            product_id.append(p_id)
        else:
            product_name.append(ing)
            product_id.append(len(product_id)+1)
```

In [236]:
```
d = pd.DataFrame({'user_id':user_id, 'product_id':product_id,'product_name':product_name,'skin_type':sk
```

In [237]:
```
d
```

Out[237]:

	user_id	product_id	product_name	skin_type	rating	review
0	1	1	Vitamin C Serum	acne	0.9798	moisturize every day things aveeno drugstorebu...
1	1	2	The Ordinary	acne	0.9798	moisturize every day things aveeno drugstorebu...
2	1	3	niacinamide	acne	0.9798	moisturize every day things aveeno drugstorebu...
3	1	4	salicylic acid	acne	0.9798	moisturize every day things aveeno drugstorebu...
4	1	5	vitamin C	acne	0.9798	moisturize every day things aveeno drugstorebu...
...	...	...	...	...	...	...
115243	23122	4	salicylic acid	acne	0.9872	ive used glow tonic impressed formulated prope...
115244	23122	5	vitamin C	acne	0.9872	ive used glow tonic impressed formulated prope...
115245	23123	15	cerave	acne	-0.8552	cerave cleanser moisturizer treat acne acne sp...
115246	23123	4	salicylic acid	acne	-0.8552	cerave cleanser moisturizer treat acne acne sp...
115247	23123	11	retinoid	acne	-0.8552	cerave cleanser moisturizer treat acne acne sp...

115248 rows × 6 columns

Figure 7: Steps to create the final user-item matrix used for modeling

7 Modelling

The four machine learning models implemented and evaluated are MF, HFT, NCF and NCF + reviews.

7.1 Modelling Matrix Factorisation

Matrix factorisation

n [6]:
```
from __future__ import print_function, division
from builtins import range, input
```

n [7]:
```
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from datetime import datetime
```

n [8]:
```
# load in the data
import os
if not os.path.exists('user2product.json') or \
   not os.path.exists('product2user.json') or \
   not os.path.exists('userproduct2rating.json') or \
   not os.path.exists('userproduct2rating_test.json'):
   import preprocess2dict


with open('user2product.json', 'rb') as f:
    user2product = pickle.load(f)

with open('product2user.json', 'rb') as f:
    product2user = pickle.load(f)

with open('userproduct2rating.json', 'rb') as f:
    userproduct2rating = pickle.load(f)

with open('userproduct2rating_test.json', 'rb') as f:userproduct2rating_test = pickle.load(f)
```

```
N = np.max(list(user2product.keys())) + 1
# the test set may contain productss the train set doesn't have data on
m1 = np.max(list(product2user.keys()))
m2 = np.max([m for (u, m), r in userproduct2rating_test.items()])
M = max(m1, m2) + 1
print("N:", N, "M:", M)


# initialize variables
K = 10 # latent dimensionality
W = np.random.randn(N, K)
b = np.zeros(N)
U = np.random.randn(M, K)
c = np.zeros(M)
mu = np.mean(list(userproduct2rating.values()))

# prediction[i,j] = W[i].dot(U[j]) + b[i] + c.T[j] + mu

def get_loss(d):
    N = float(len(d))
    sse = 0
    for k, r in d.items():
        i, j = k
        p = W[i].dot(U[j]) + b[i] + c[j] + mu
        sse += (p - r)*(p - r)
    return sse / N


# train the parameters
epochs = 10
reg = 0.0001 # regularization penalty
train_losses = []
test_losses = []
for epoch in range(epochs):
    print("epoch:", epoch)
    epoch_start = datetime.now()
```

```
        matrix = np.eye(K) * reg
        vector = np.zeros(K)

        # for c
        cj = 0
        try:
            for i in product2user[j]:
                r = userproduct2rating[(i,j)]
                matrix += np.outer(W[i], W[i])
                vector += (r - b[i] - c[j] - mu)*W[i]
                cj += (r - W[i].dot(U[j]) - b[i] - mu)
            # set the updates
            U[j] = np.linalg.solve(matrix, vector)
            c[j] = cj / (len(product2user[j]) + reg)

            if j % (M//10) == 0:
                print("j:", j, "M:", M)
        except KeyError:
            # possible not to have any ratings for a product
            pass
    print("updated U and c:", datetime.now() - t0)
    print("epoch duration:", datetime.now() - epoch_start)


    # store train loss
    t0 = datetime.now()
    train_losses.append(get_loss(userproduct2rating))

    # store test loss
    test_losses.append(get_loss(userproduct2rating_test))
    print("calculate cost:", datetime.now() - t0)
    print("train loss:", train_losses[-1])
    print("test loss:", test_losses[-1])


print("train losses:", train_losses)
```

Figure 8: Implementation of matrix factorization model.

MSE: 0.39

7.2 Modelling Hidden Factors as Topics (HFT)

```
# load in the data
with open('edited5.pkl', 'rb') as f:
  dataset = pickle.load(f)
dataset

with open('topic_vector1.pkl', 'rb') as f:
  topics = pickle.load(f)
topics
```

```
df = pd.DataFrame(dataset)
```

```
tp = []
for i in topics:
  tp.append(np.array(i))
```

```
tp = np.array(tp)
```

```
N = df.userId.max() + 1 # number of users
M = df.product_idx.max() + 1 # number of prouct
```

```
# split into train and test
from sklearn.model_selection import train_test_split
df_train, df_test, tp_train, tp_test = train_test_split(df, tp, test_size=0.2, random_state = 42)
```

```
# initialize variables
K = 10 # latent dimensionality
mu = df_train.rating.mean()
epochs = 10
```

```
# keras model
u = Input(shape=(1,))
m = Input(shape=(1,))
u_embedding = Embedding(N, K)(u) # (N, 1, K)
m_embedding = Embedding(M, K)(m) # (N, 1, K)
u_embedding = Flatten()(u_embedding) # (N, K)
m_embedding = Flatten()(m_embedding) # (N, K)
x = Concatenate()([u_embedding, m_embedding]) # (N, 2K)

topic_input = Input(shape=(5, ))
```

```
r = model.fit(
  x=[df_train.userId.values, df_train.product_idx.values, tp_train],
  y=df_train.rating.values - mu,
  epochs=epochs,
  batch_size=128,
  validation_data=(
    [df_test.userId.values, df_test.product_idx.values, tp_test],
    df_test.rating.values - mu
  )
)


# plot losses
plt.plot(r.history['loss'], label="train loss")
plt.plot(r.history['val_loss'], label="test loss")
plt.legend()
plt.show()

# plot mse
plt.plot(r.history['mean_squared_error'], label="train mse")
plt.plot(r.history['val_mean_squared_error'], label="test mse")
plt.legend()
plt.show()
```

```
Epoch 1/10
721/721 [==============================] - 4s 5ms/step - loss: 0.5616 - mean_squared_error: 0.5616 - val
Epoch 2/10
721/721 [==============================] - 3s 5ms/step - loss: 0.4797 - mean_squared_error: 0.4797 - val
Epoch 3/10
721/721 [==============================] - 3s 5ms/step - loss: 0.4809 - mean_squared_error: 0.4809 - val
Epoch 4/10
721/721 [==============================] - 3s 5ms/step - loss: 0.4800 - mean_squared_error: 0.4800 - val
Epoch 5/10
```

Figure 9: Implementation of Hidden Factors as Topics model.

MSE: 0.66

7.3 Modelling Neural Collaborative Filtering (NCF)

```
# keras model
u = Input(shape=(1,), name='Users')
m = Input(shape=(1,), name='Items')
u_embedding = Embedding(N, K)(u) # (N, 1, K)
m_embedding = Embedding(M, K)(m) # (N, 1, K)
u_embedding = Flatten()(u_embedding) # (N, K)
m_embedding = Flatten()(m_embedding) # (N, K)
x = Concatenate()([u_embedding, m_embedding]) # (N, 2K)

# the neural network
x = Dense(400)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(100)(x)
# x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dense(1)(x)

model = Model(inputs=[u, m], outputs=x)
model.compile(
  loss='mse',
  # optimizer='adam',
  # optimizer=Adam(lr=0.01),
  optimizer=SGD(lr=0.08, momentum=0.9),
  metrics=['mean_squared_error'],
)

r = model.fit(
  x=[df_train.userId.values, df_train.product_idx.values],
  y=df_train.rating.values - mu,
  epochs=epochs,
```

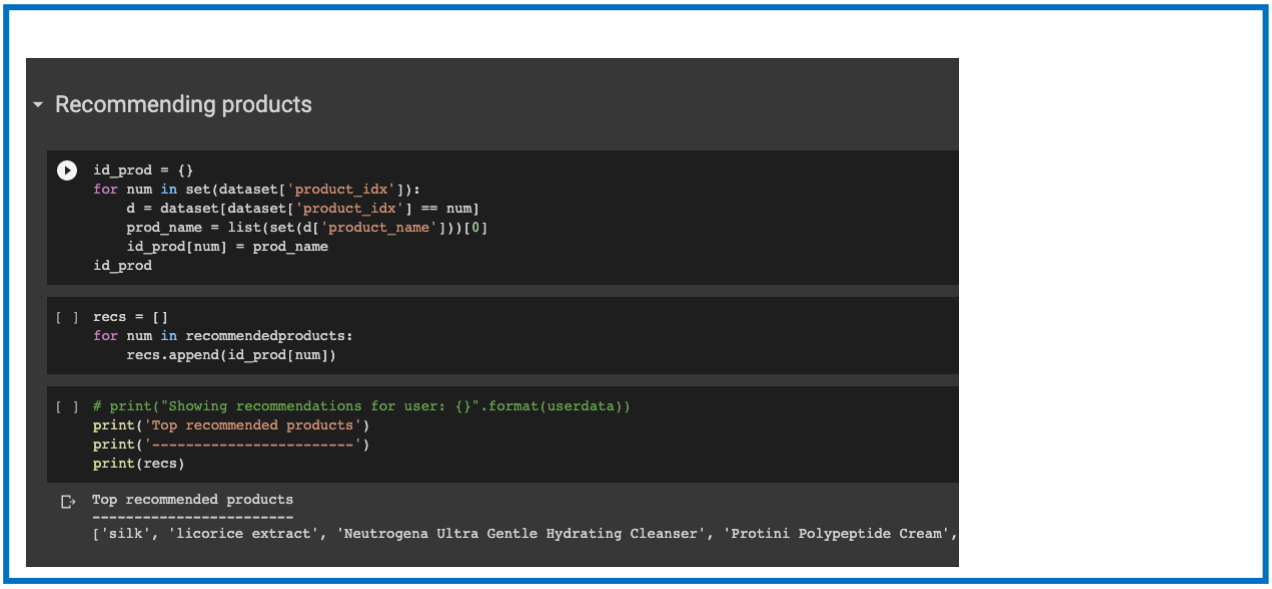


Figure 10: Implementation of Neural Collaborative Filtering model.

MSE: 0.32

7.4 Modelling Proposed Methodology, Neural Collaborative Filtering + reviews.

```
N = df.userId.max() + 1 # number of users
M = df.product_idx.max() + 1 # number of product

# split into train and test
df = shuffle(df)
cutoff = int(0.8*len(df))
df_train = df.iloc[:cutoff]
df_test = df.iloc[cutoff:]

# tokenizing word and converting into sequences.
vocab_size = 1000
embedding_dim = 128
max_length = 400
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"


from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(df_train['review'])
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(df_train['review'])
review_padded = pad_sequences(sequences,maxlen=max_length, padding=padding_type,
                              truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(df_test['review'])
review_testing_padded = pad_sequences(testing_sequences,maxlen=max_length,
```

```
# keras model
u = Input(shape=(1,))
m = Input(shape=(1,))
u_embedding = Embedding(N, K)(u) # (N, 1, K)
m_embedding = Embedding(M, K)(m) # (N, 1, K)
u_embedding = Flatten()(u_embedding) # (N, K)
m_embedding = Flatten()(m_embedding) # (N, K)
x = Concatenate()([u_embedding, m_embedding]) # (N, 2K)

review_input = Input(shape=(max_length, ))
review_embedding = Embedding(vocab_size, embedding_dim, input_length=max_length)(review_input)
review_embedding = Bidirectional(LSTM(100, dropout = 0.2))(review_embedding)
review_final = Dense(100, activation='tanh', name ="doc_latent_vector")(review_embedding)
merged = Concatenate()([x, review_final])

# the neural network
x = Dense(400)(merged)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(100)(x)
# x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dense(1)(x)

model = Model(inputs=[u, m, review_input], outputs=x)
model.compile(
  loss='mse',
  # optimizer='adam',
  # optimizer=Adam(lr=0.01),
  optimizer=SGD(lr=0.08, momentum=0.9),
  metrics=['mean_squared_error'],)
model.summary()
```

```
r = model.fit(
  x=[df_train.userId.values, df_train.product_idx.values, review_padded],
  y=df_train.rating.values - mu,
  epochs=epochs,
  batch_size=128,
  validation_data=(
    [df_test.userId.values, df_test.product_idx.values, review_testing_padded],
    df_test.rating.values - mu
  )
)


# plot losses
plt.plot(r.history['loss'], label="train loss")
plt.plot(r.history['val_loss'], label="test loss")
plt.legend()
plt.show()

# plot mse
plt.plot(r.history['mean_squared_error'], label="train mse")
plt.plot(r.history['val_mean_squared_error'], label="test mse")
plt.legend()
plt.show()
```

```
Epoch 1/10
721/721 [==============================] - 1658s 2s/step - loss: 0.4060 - mean_squ
Epoch 2/10
721/721 [==============================] - 1640s 2s/step - loss: 0.1734 - mean_squ
Epoch 3/10
721/721 [==============================] - 1639s 2s/step - loss: 0.0572 - mean_squ
Epoch 4/10
```

Figure 11: Implementation of Neural Collaborative Filtering + reviews model.

MSE: 0.30

References

Hamilton, W.L., Clark, K., Leskovec, J. and Jurafsky, D. (2016) 'Inducing domain-specific sentiment lexicons from unlabeled corpora', In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, 2016, p. 595, ArXiv preprint arxiv:1606.02820.