Precily Assessment Report

Approach 1:

Given the Dataset, with 3 columns (UniqueID, text1, text2) and 4023 rows.

The language used for below mentioned code is R language.

Libraries used:

```
library(tidytext)
library(janeaustenr)
library(dplyr)
library(tm)
library(SnowballC)
library(caTools)
library(randomForest)
library(lsa)
library(tokenizers)
library(tidyverse)
library(SnowballC)
library(stringr)
```

Loading the dataset:

```
dataset_original <- read.csv("Text_Similarity_Dataset.csv", quote = "", stringsAsFactors = F)
```

Creating Corpus:

```
corpus_1 <- VCorpus(VectorSource(dataset_original$text1))
corpus_2 <- VCorpus(VectorSource(dataset_original$text2))
```

Cleaning the texts:

- Converting all the data to lower case

```
corpus_1 <- tm_map(corpus_1, content_transformer(tolower))
corpus_2 <- tm_map(corpus_2, content_transformer(tolower))
```

- Removing numbers

```
corpus_1 <- tm_map(corpus_1, removeNumbers)
corpus_2 <- tm_map(corpus_2, removeNumbers)
```

- Removing Punctuation

```
corpus_1 <- tm_map(corpus_1, removePunctuation)
corpus_2 <- tm_map(corpus_2, removePunctuation)
```

- Removing Stopword

```
corpus_1 <- tm_map(corpus_1, removeWords, stopwords())
corpus_2 <- tm_map(corpus_2, removeWords, stopwords())
```

- Stemming the texts

```
corpus_1 <- tm_map(corpus_1, stemDocument)
corpus_2 <- tm_map(corpus_2, stemDocument)
```

- Stripping of the white spaces

```
corpus_1 <- tm_map(corpus_1, stripWhitespace)
corpus_2 <- tm_map(corpus_2, stripWhitespace)
```

Bag of Words Matrix Creation:

Creating a matrix which contains all the word count which is stored in Document Term Matrix, each for 2 corpuses.

```
dtm_1 <- DocumentTermMatrix(corpus_1)
dtm_2 <- DocumentTermMatrix(corpus_2)
```

Forming a Data Frame:

Using the above created Document Term Matrix, 2 datasets as data frame has been created for each dtm.

```
dataset_1 <- as.data.frame(as.matrix(dtm_1))
dataset_2 <- as.data.frame(as.matrix(dtm_2))
```

Creating Functions:

Now applying the concept of TF-IDF (term frequency-inverse document frequency) for that creating 3 functions each for tf, idf, and tf-idf

```
# function for term frequency
term.freq <- function(row) {
  row / sum(row)
}
# function for inverse document frequency
inv.doc.freq <- function(col) {
  corpus.size <- length(col)
  doc.count <- length(which(col > 0))
  log10(corpus.size / doc.count)
}
# function for term frequency inverse document frequency combined
tf.idf <- function(tf, idf) {
  tf * idf
}
```

Calculating Term Frequency Matrix:

```
# calculating tf for the corpus
dataset_1.df <- apply(dataset_1, 1, term.freq)
dataset_2.df <- apply(dataset_2, 1, term.freq)
```

Calculating Inverse Document Frequency Vector:

```
dataset_1.idf <- apply(dataset_1, 2, inv.doc.freq)
dataset_2.idf <- apply(dataset_2, 2, inv.doc.freq)
```

Calculating TF-IDF Matrix:

TF-IDF is nothing but the multiplication of TF and IDF.

```
dataset_1.tfidf <- apply(dataset_1.df, 2, tf.idf, idf = dataset_1.idf)
dataset_2.tfidf <- apply(dataset_2.df, 2, tf.idf, idf = dataset_2.idf)
```

and then once created TF-IDF matrix transpose of that is done so that it again gains the form in which the words are columns and documents are rows.

```
dataset_1.tfidf <- t(dataset_1.tfidf)
dataset_2.tfidf <- t(dataset_2.tfidf)
```

Checking for empty entries:

```
incomplete.cases <- which(!complete.cases(dataset_1.tfidf))
dataset_original$text1[incomplete.cases]
incomplete.cases <- which(!complete.cases(dataset_2.tfidf))
dataset_original$text2[incomplete.cases]
```

Fixing these empty entries:

```
dataset_1.tfidf[incomplete.cases, ] <- rep(0.0, ncol(dataset_1.tfidf))
sum(which(!complete.cases(dataset_1.tfidf)))
dataset_2.tfidf[incomplete.cases, ] <- rep(0.0, ncol(dataset_2.tfidf))
sum(which(!complete.cases(dataset_2.tfidf)))
```

Making same number of columns:

The 2 data frames created may not consist the same number of words, and also the same words, so making sure that they contain the same words,

```
names1<-colnames(dataset_1)
names2<-colnames(dataset_2)
```

```
final_names<-union( names1, names2 )
names1_f<-setdiff(final_names,names1)
names2_f<-setdiff(final_names,names2)
```

Generating Similarity:

Once the final data frames are created for both text1 and text2 cosine similarity is applied for each UniqueID.

```
data_ans <- dataset_original
data_ans$text1 <- NULL
data_ans$text2 <- NULL
data_ans$similarity <- 0
for (i in 0:4022) {
  data_ans$similarity[i+1] <- cosine(x = as.numeric(as.vector(dataset_1.tfidf[i+1, ])),
                                     y = as.numeric(as.vector(dataset_2.tfidf[i+1, ])))
}
```

Summary of the Similarity:

```
summary(data_ans$similarity)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.07004 0.11932 0.14388 0.19103 0.97573
```

Excel File name: “Similarity.csv”

Similarity of first 10 IDs is shown below:

Unique_ID	similarity
0	0.269895
1	0.105942
2	0.146192
3	0.273519
4	0.332269
5	0.222687
6	0.078747
7	0.274252
8	0.134859
9	0.336608

This similarity does not take into account of semantics, another approached shown below takes semantics in account.

Code:

```
library(tidytext)
library(janeaustenr)
library(dplyr)
library(tm)
library(SnowballC)
library(caTools)
library(randomForest)
library(lsa)
library(tokenizers)
library(tidyverse)
library(SnowballC)
library(stringr)
```

```
getwd()
setwd("E:/IIT Kanpur/Summer_Internship/Precily/R_script")
dataset_original <- read.csv("Text_Similarity_Dataset.csv", quote = "", stringsAsFactors = F)

#Cleaning the texts
corpus_1 <- VCorpus(VectorSource(dataset_original$text1))
corpus_1 <- tm_map(corpus_1, content_transformer(tolower))
corpus_1 <- tm_map(corpus_1, removeNumbers)
corpus_1 <- tm_map(corpus_1, removePunctuation)
corpus_1 <- tm_map(corpus_1, removeWords, stopwords())
corpus_1 <- tm_map(corpus_1, stemDocument)
corpus_1 <- tm_map(corpus_1, stripWhitespace)

corpus_2 <- VCorpus(VectorSource(dataset_original$text2))
corpus_2 <- tm_map(corpus_2, content_transformer(tolower))
corpus_2 <- tm_map(corpus_2, removeNumbers)
corpus_2 <- tm_map(corpus_2, removePunctuation)
corpus_2 <- tm_map(corpus_2, removeWords, stopwords())
corpus_2 <- tm_map(corpus_2, stemDocument)
corpus_2 <- tm_map(corpus_2, stripWhitespace)

# Creating the Bag of Words Model
dtm_1 <- DocumentTermMatrix(corpus_1)

dtm_1 <- removeSparseTerms(dtm_1, 0.9)

dtm_2 <- DocumentTermMatrix(corpus_2)

dtm_2 <- removeSparseTerms(dtm_2, 0.9)

dataset_1 <- as.data.frame(as.matrix(dtm_1))
dataset_2 <- as.data.frame(as.matrix(dtm_2))

###### tf idf calculations

# function for term frequency
term.freq <- function(row) {
  row / sum(row)
}

# function for inverse document frequency
inv.doc.freq <- function(col) {
  corpus.size <- length(col)
  doc.count <- length(which(col > 0))
  log10(corpus.size / doc.count)
}

# function for term frequency inverse document frequency combined
tf.idf <- function(tf, idf) {
  tf * idf
}

# calculating tf for the corpus
dataset_1.df <- apply(dataset_1, 1, term.freq)
dim(dataset_1.df)
View(dataset_1.df)
```

```
# calcualtion idf for the corpus
dataset_1.idf <- apply(dataset_1, 2, inv.doc.freq)
str(dataset_1.idf)
dim(dataset_1.idf)
View(dataset_1.idf)

# calcualtion tf.idf for the corpus
dataset_1.tfidf <- apply(dataset_1.df, 2, tf.idf, idf = dataset_1.idf)
dim(dataset_1.tfidf)
View(dataset_1.tfidf)

# Transpose the matrix
dataset_1.tfidf <- t(dataset_1.tfidf)
dim(dataset_1.tfidf)
View(dataset_1.tfidf)

# checking for incomplete cases, that is after preprocessing there can a text which is empty string
incomplete.cases <- which(!complete.cases(dataset_1.tfidf))
dataset_original$text1[incomplete.cases]

# fixing incompleter cases if any present
dataset_1.tfidf[incomplete.cases, ] <- rep(0.0, ncol(dataset_1.tfidf))
dim(dataset_1.tfidf)
sum(which(!complete.cases(dataset_1.tfidf)))

# filtering tokens which are not significant on tfidf basis
dataset_1.tfidf <- round(dataset_1.tfidf, 5)

### doing for dataset_2
# calculating tf for the corpus
dataset_2.df <- apply(dataset_2, 1, term.freq)

# calcualtion idf for the corpus
dataset_2.idf <- apply(dataset_2, 2, inv.doc.freq)

# calcualtion tf.idf for the corpus
dataset_2.tfidf <- apply(dataset_2.df, 2, tf.idf, idf = dataset_2.idf)

# Transpose the matrix
dataset_2.tfidf <- t(dataset_2.tfidf)

# checking for incomplete cases, that is after preprocessing there can a text which is empty string
incomplete.cases <- which(!complete.cases(dataset_2.tfidf))
dataset_original$text2[incomplete.cases]

# fixing incompleter cases if any present
dataset_2.tfidf[incomplete.cases, ] <- rep(0.0, ncol(dataset_2.tfidf))
dim(dataset_2.tfidf)
sum(which(!complete.cases(dataset_2.tfidf)))

# rounding  tfidf
dataset_2.tfidf <- round(dataset_2.tfidf, 5)

####
dim(dataset_1.tfidf)
```

```
dim(dataset_2.tfidf)

dataset_1.tfidf <- as.data.frame(dataset_1.tfidf)
dataset_2.tfidf <- as.data.frame(dataset_2.tfidf)


names1<-colnames(dataset_1)
names2<-colnames(dataset_2)
final_names<-union( names1, names2 )

names1_f<-setdiff(final_names,names1)
names2_f<-setdiff(final_names,names2)

c <- colnames(dataset_1.tfidf) != colnames(dataset_2.tfidf)
dataset_1.tfidf$charg <- 0
dataset_1.tfidf$demand <- 0
dataset_1.tfidf$life <- 0
dataset_1.tfidf$match <- 0
dataset_1.tfidf$never <- 0
dataset_2.tfidf$fail <- 0
dataset_2.tfidf$huge <- 0
dataset_2.tfidf$pay <- 0

dataset_1.tfidf <- dataset_1.tfidf[ ,order(names(dataset_1.tfidf))]
dataset_2.tfidf <- dataset_2.tfidf[ ,order(names(dataset_2.tfidf))]
c <- colnames(dataset_1.tfidf) != colnames(dataset_2.tfidf)
c


data_ans <- dataset_original
data_ans$text1 <- NULL
data_ans$text2 <- NULL
data_ans$similarity <- 0

for (i in 0:4022) {
  data_ans$similarity[i+1] <- cosine(x = as.numeric(as.vector(dataset_1.tfidf[i+1, ])),
                                     y = as.numeric(as.vector(dataset_2.tfidf[i+1, ])))
}


max(data_ans$similarity)
min(data_ans$similarity)
summary(data_ans$similarity)

write.csv(data_ans ,file = 'Similarity.csv', row.names = F)
```

Approach 2:

Given the Dataset, with 3 columns (UniqueID, text1, text2) and 4023 rows.

The language used for below mentioned code is PYTHON.

Preprocessing:

Once the data is loaded, processing needs to be done. For this, initially phrases such are ‘re is converted to are, ‘m is made am, won’t is made will not, etc.

Further processing is done which includes removing numeric values, punctuation, whitespaces, new lines.

Also stemming of the words is done in processing the data.

Tokenizing the text is the next step.

This concludes the processing part of the data.

Model Applied:

Document to Vector(doc2vec) model which covers the semantic meaning by converting the document to a vector is used for the data here.

Once the data is processed, words which are present in our data but not in Google news vector are removed and the ones which are present their similarity is compared (refer code).

Texts with similarity 1 are the most similar and with similarity 0 are the least similar.

Doc2Vec is the model is applied for checking the similarity between 2 texts.

Result of first 10 IDs is shown below.

Unique_ID	Similarity_score
0	0.610529
1	0.707934
2	0.72711
3	0.815045
4	0.828323
5	0.750712
6	0.680868
7	0.847387
8	0.79064
9	0.901291

Excel file name: “final_score.csv”

Code:

```
import os
import numpy as np
import pandas as pd

import re
from tqdm import tqdm

import collections

from sklearn.cluster import KMeans

from nltk.stem import WordNetLemmatizer  # For Lemmetization of words
from nltk.corpus import stopwords  # Load list of stopwords
from nltk import word_tokenize # Convert paragraph in tokens

import pickle
import sys
```

```
from gensim.models import word2vec # For represent words in vectors
import gensim

# Read given data-set using pandas
os.chdir("E:\\IIT Kanpur\\Summer_Internship\\Precily\\Precily Assessment")
text_data = pd.read_csv("Text_Similarity_Dataset.csv")
print("Shape of text_data : ", text_data.shape)
text_data.head(3)

text_data.isnull().sum() # Check if text data have any null values

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


# Combining all the above stundents

preprocessed_text1 = []

# tqdm is for printing the status bar

import nltk
nltk.download('stopwords')

for sentance in tqdm(text_data['text1'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)

    sent = ' '.join(e for e in sent.split() if e not in stopwords.words('english'))
    preprocessed_text1.append(sent.lower().strip())


# Merging preprocessed_text1 in text_data

text_data['text1'] = preprocessed_text1
text_data.head(3)

# Combining all the above stundents
from tqdm import tqdm
preprocessed_text2 = []
```

```
# tqdm is for printing the status bar
for sentance in tqdm(text_data['text2'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)

    sent = ' '.join(e for e in sent.split() if e not in stopwords.words('english'))
    preprocessed_text2.append(sent.lower().strip())

# Merging preprocessed_text2 in text_data

text_data['text2'] = preprocessed_text2

text_data.head(3)

def word_tokenizer(text):
        #tokenizes and stems the text
        tokens = word_tokenize(text)
        lemmatizer = WordNetLemmatizer()
        tokens = [lemmatizer.lemmatize(t) for t in tokens]
        return tokens

# Load pre_trained Google News Vectors after download file

wordmodelfile = "GoogleNews-vectors-negative300.bin.gz"
wordmodel = gensim.models.KeyedVectors.load_word2vec_format(wordmodelfile, binary=True)

# This code check if word in text1 & text2 present in our google news vectors vocabalry.
# if not it removes that word and if present it compares similarity score between
# text1 and text2 words


similarity = [] # List for store similarity score

nltk.download('punkt')
nltk.download('wordnet')

for ind in text_data.index:

    s1 = text_data['text1'][ind]
    s2 = text_data['text2'][ind]

    if s1==s2:
        similarity.append(1.0) # 1 means highly similar

    else:

        s1words = word_tokenizer(s1)
        s2words = word_tokenizer(s2)



        vocab = wordmodel.vocab #the vocabulary considered in the word embeddings
```

```
        if len(s1words and s2words)==0:
            similarity.append(0.0)

        else:

            for word in s1words.copy(): #remove sentence words not found in the vocab
                if (word not in vocab):


                    s1words.remove(word)


            for word in s2words.copy(): #idem

                if (word not in vocab):

                    s2words.remove(word)


            similarity.append((wordmodel.n_similarity(s1words, s2words))) # as it is given 1 means highly dissimilar
& 0 means highly similar

# Get Unique_ID and similarity

final_score = pd.DataFrame({'Unique_ID':text_data.Unique_ID,
                'Similarity_score':similarity})
final_score.head(3)

# SAVE DF as CSV file

final_score.to_csv('final_score.csv',index=False)
```