Configuration Manual of Research Project: Topic Modelling of Online Reviews for Airports In Europe

MSc Research Project
Data Analytics

Dona Elizabeth John
Student ID: x21228531

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dona Elizabeth John
Student ID:	x21228531
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Catherine Mulwa
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual of Research Project: Topic Modelling of Online Reviews for Airports In Europe
Word Count:	629
Page Count:	10

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:	DONA ELIZABETH JOHN
Date:	18th September 2023

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 of Research Project: Topic Modelling of Online Reviews for Airports In Europe

Dona Elizabeth John
x21228531

1 Introduction

This configuration manual is a comprehensive documentation of the various configurations and settings that influence the results of the study "Topic Modelling of Online Reviews for Airports in Europe." The configuration techniques, software requirements and an overview of the code artifacts used to achieve the goals of the research project are described in great detail within the pages of this paper.

2 System specifications

The study was carried out using Google Colab, a cloud computing platform well-known for its ability to code and run deep learning and machine learning models. TensorFlow and Keras are combined in Google Colab, enabling faster execution rates than with a CPU alone. If necessary, Google Colab can speed up execution by using a GPU or TPU. The detailed specification is displayed in Figure 1

Platform	Google colab
GPU	12GB VRAM
CPU	13GB RAM
Storage	78GB
Driver	NVIDIA Tesla K80

Figure 1: Platform specification

3 Libraries

Gensim package was installed using pip command. The libraries were imported to colab session and is displayed in Figure 2

```
[1] pip install scikit-learn gensim
```

```
[23]
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import gensim
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
import numpy as np
import gensim.corpora as corpora
```

Figure 2: Library import

4 Data import

Data was imported using pandas package as shown in Figure 3

```
df = pd.read_csv("airport_reviews_dataset.csv")
```

```
df.columns
```

Figure 3: Data Import

5 Data filtering

All irrelevant data is excluded from building topic model. This numeric tokens which will impact in building topics. The exclusion of the data was displayed in Figure 5

```
df1=df[~(df['content'].isna())].copy()
num_val=df1[df1.content.str.isnumeric()].copy()
df2=df1[~(df1['content'].isin(num_val.content))].copy()
df2['dt']=df2['date'].astype('str')
```

```
df2.dtypes

df2['yr']=df2['dt'].apply(lambda x:  x.split('/')[2] if len(x.split('/'))>1 else '0')
```

```
df2.shape
```

Figure 4: Filtering

6 Preliminary Data Analysis

This section includes exploratory data analysis on the review columns. The distribution of topics across years was displayed in Figure 5

```
df2['title'].value_counts()
```

```
df2['yr'].value_counts().plot.bar()
plt.xlabel("Year")
plt.ylabel("Number of Reviews")
plt.title("Number of Reviews Posted in Different Years")
```

Figure 5: EDA

The plot is shown in Figure 6

```
plt.xlabel("Year")
plt.ylabel("Number of Reviews")
plt.title("Number of Reviews Posted in Different Years")
```

```
Text(0.5, 1.0, 'Number of Reviews Posted in Different Years')
```

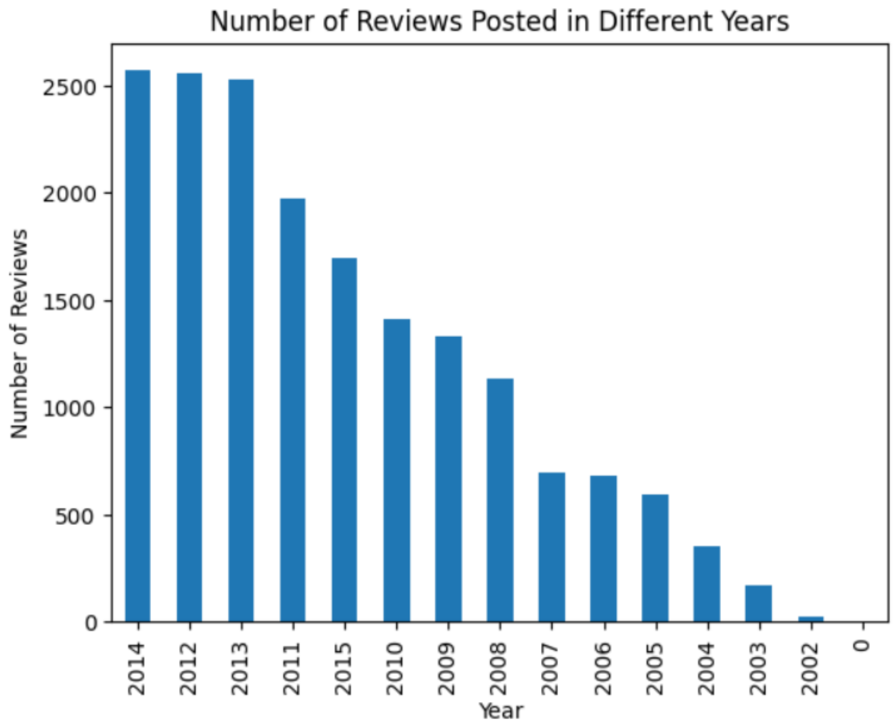

Figure 6: EDA Plot

7 Data transformation

Each review was converted as an item in a list. This is displayed in Figure 7

```
val_list=df2['content'].tolist()
```

Figure 7: Reviews in list

8 TF-IDF Implementation

The first phase of study includes the implementation of tf-idf LDA model. Each review was transformed into tf-idf vectors. This is shown in Figure 8

```
[12] # Initializing regex tokenizer
     tokenizer = RegexpTokenizer(r'\w+')

     # Vectorizing the data using TF-IDF
     tfidfVect = TfidfVectorizer(lowercase=True,
                                 stop_words='english',
                                 ngram_range = (1,1),
                                 tokenizer = tokenizer.tokenize)

     # Fit and Transform the documents
     tData = tfidfVect.fit_transform(val_list)
```

Figure 8: TF-IDF

9 TF-IDF Model

Model was built using TF-IDF vectors and is dsplayed in Figure 9

```
[13] # Defining the number of topics
     n=15

     # Creating LDA object
     ldaModel=LatentDirichletAllocation(n_components=n)

     # Fitting and transforming model on data
     ldaMatrix1 = ldaModel.fit_transform(tData)

     # Getting components
     ldaComponents1=ldaModel.components_
```

Figure 9: TF-IDF Model

10 TF-IDF Evaluation

TF-IDF model was evaluated in this section., The topics were distributed and analyzed as shown in Figure 10

```
[14] # Printing the topics with their corresponding words
     words = tfidfVect.get_feature_names_out()

     for index, component in enumerate(ldaComponents1):
         zipped = zip(words, component)
         top_words_key=sorted(zipped, key = lambda t: t[1], reverse=True)[:7]
         top_words_list=list(dict(top_words_key).keys())
         print("Topic "+str(index)+": ",top_words_list)
```

Figure 10: TF-IDF Evaluation

11 TF-IDF Stability test

The stability test was performed on TF-IDF model and the result were not convincing. The test performed is shown in Figure 11

12 Count Vectorization

The second phase of implementation was Count Vectorization. The model was built using this technique. The evaluation of the topics were performed after seggregating LDA components and is displayed in Figure 12

13 Coherence Score

The count vectorization model performed better than TF-IDF model and the coherence score for count vectorization model was calculated. It is shown in Figure 13

14 Topic distribution

The topic distribution from the count vectorization model was ploted as it is shown in Figure 14

15 Count vectorization stability test

The stability test was performed for count vectorization model and is displayed in Figure 15

```
[33] # Stability test for TF-IDF Model
     # Number of subsets and stability threshold
     subNum = 5

     # Performing model stability test
     stabScores = []
     num=15
     for i in range(subNum):
         # Generating a random subset of data
         subIndices = np.random.choice(tData.shape[0], size=int(tData.shape[0] * 0.8), replace=False)
         subset = tData[subIndices]

         # Fitting LDA model
         lda_model1 = LatentDirichletAllocation(n_components=num, random_state=42)
         lda_model1.fit(subset)

         # Calculating jaccard similarity between topics of different subsets
         jSimilarity1 = []
         for otherSubIndices in range(subNum):
             if otherSubIndices == i:
                 continue
             otherSubset = tData[otherSubIndices]
             otherTopics = ldaModel.transform(otherSubset)
             currentTopics = ldaModel.transform(subset)
             jSim = np.min(np.minimum(otherTopics, currentTopics).sum(axis=1))
             jSimilarity1.append(jSim)

         stabilityScore = np.mean(jSimilarity1)
         stabScores.append(stabilityScore)

     # Calculating stability score
     meanStab = np.mean(stabScores)
     print("Mean Stability Score:", meanStab)
```

Figure 11: TF-IDF Stabilitry test

```
[18] # Vectorizing the data using CountVectorization
     cVect = CountVectorizer(stop_words='english', max_df=.1, max_features=5000)
     X = cVect.fit_transform(val_list)
```

```
[19] # Defining the number of topics
     n=15
     # Creating LDA object
     ldaModel2=LatentDirichletAllocation(n_components=n)
     # Fitting and transforming model on data
     ldaMatrix2 = ldaModel2.fit_transform(X)
     # Getting components
     ldaComponents2=ldaModel2.components_
```

```
[20] # Printing the topics with their corresponding words
     words2 = cVect.get_feature_names_out()
     fList=[]
     for index, component in enumerate(ldaComponents2):
         zipped = zip(words2, component)
         top_words_key2=sorted(zipped, key = lambda t: t[1], reverse=True)[:15]
         top_words_list2=list(dict(top_words_key2).keys())
         print("Topic "+str(index)+": ",top_words_list2)
         fList=fList+[top_words_list2]
```

Figure 12: Count Vectorization

```
[27] # Coherence Score Test

     def get_Cv(ldaModel2, dfColumn):
       topics = ldaModel2.components_

       ntopWords = 15
       cWords = [[word for word in doc.split()] for doc in dfColumn]

       # creating the dictionary
       cDictionary = corpora.Dictionary(cWords)
       # Creating a gensim dictionary from the word count matrix

       featureNames = [cDictionary[i] for i in range(len(cDictionary))]

       # Get the top words for each topic from the components_ attribute
       topWords = []
       for topic in topics:
           topWords.append([featureNames[i] for i in topic.argsort()[:-ntopWords - 1:-1]])

       cohModel = CoherenceModel(topics=topWords, texts=cWords, dictionary=cDictionary, coherence='c_v')
       coh = cohModel.get_coherence()
       return coh

     cohScore=get_Cv(ldaModel2,val_list)

[25] cohScore
```

Figure 13: Coherence Score

16 Topic seggregation

The topics received from count vectorization model is analyzed and the certain topics which have similar context which were captured in different topics were merged. The travel desk topics were merged using Topic 6, Topic 7 and Topic 0 as shown in Figure 16

Similarly topics for parking desk and outlet desk were created by merging corresponding similar topics and unwanted tokens were also removed. This is displayed in Figure 17

Category column was created to segregate the reviews and were send to the corresponding airport departments. and Figure 18

```
# Aggregating the topic proportions across all data
agTopic = np.sum(tAssign, axis=0)

# Normalizing proportions
totData = len(tAssign)
normTopic = agTopic / totData

# Visualizing the normalized topic proportions
topicNum = len(normTopic)
topics = [f"Topic {i}" for i in range(topicNum)]

plt.bar(topics, normTopic)
plt.xticks(rotation='vertical')
plt.xlabel("Topics")
plt.ylabel("Proportion")
plt.title("Topic Distribution Proportions")
plt.show()
```

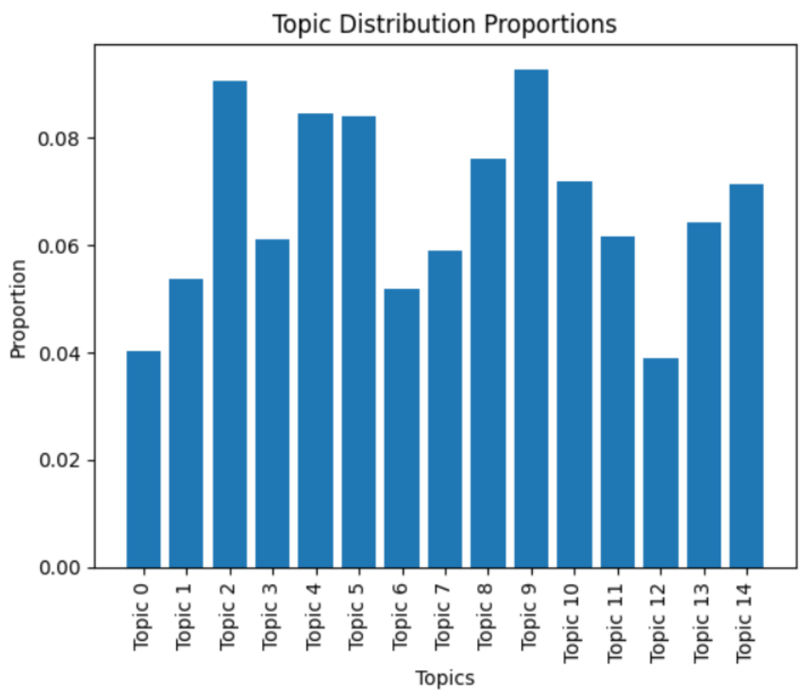

Figure 14: Coherence Score

```python
# Number of subsets and stability threshold
subNum2 = 5
#number of topics
num=15
# Model stability test
stabilScores = []

for j in range(subNum2):
    # Generate a random subset of data
    subIndices2 = np.random.choice(X.shape[0], size=int(X.shape[0] * 0.8), replace=False)
    subset2 = X[subIndices2]

    # Fit LatentDirichletAllocation model
    lda_model2 = LatentDirichletAllocation(n_components=num, random_state=42)
    lda_model2.fit(subset2)

    # Calculate Jaccard similarity between topics of different subsets
    jSimilarity2 = []
    for othersubIndices2 in range(subNum2):
        if othersubIndices2 == j:
            continue
        otherSubset2 = X[othersubIndices2]
        otherTopics2 = ldaModel2.transform(otherSubset2)
        currentTopics2 = ldaModel2.transform(subset2)
        jSim = np.min(np.minimum(otherTopics2, currentTopics2).sum(axis=1))
        jSimilarity2.append(jSim)

    stabScore = np.mean(jSimilarity2)
    stabilScores.append(stabScore)
```

```python
# Calculating stability score
meanStab = np.mean(stabScores)
print("Mean Stability Score:", meanStab)
```

Figure 15: Stability test for Count vectorization model

```python
topic_travel_desk=final_list[6]+final_list[7]+final_list[0]
remove_desk=['did','didn','english',
 'went',
 'need','process',
 'did','going',
 'finally',
 'canada',
 'got',
 'tsa',
 'told',
 'agent','10','30','20','morning', 'quite']
topic_travel_desk_1 = [ele for ele in topic_travel_desk if ele not in remove_desk]
```

Figure 16: Topics for travel desk

```python
topic_parking_desk=final_list[11]+final_list[12]+final_list[13]
remove_park=['told', 'got','having', '10', 'charge','luton', 'did', 'went','class', 'efficient', 'quite', 'city', 'kong', 'hong', 'ticket', 'counters', 'station', 'hall', 'star',
            'signage', 'lines', 'confusing', 'signs', 've', 'claim','tsa', 'lax', 'easy', 'lot']

topic_park_desk_1 = [ele for ele in topic_parking_desk if ele not in remove_park]
```

```python
topic_restaurant_desk=final_list[9]
remove_rest=['don','world', 'building', 'toilets', 'worst','place', 'country', 'really','departures', 'need']

topic_rest_desk_1 = [ele for ele in topic_restaurant_desk if ele not in remove_rest]
```

Figure 17: Topics for parking desk

```
df2['category']='Generic'
df2['category'] = np.where(df2['content'].str.split().map(set).apply(lambda x: any(x.intersection(topic_travel_desk_1))*1)==1,'Travel desk',df2['category'])
df2['category'] = np.where(df2['content'].str.split().map(set).apply(lambda x: any(x.intersection(topic_park_desk_1))*1)==1,'Parking desk',df2['category'])
df2['category'] = np.where(df2['content'].str.split().map(set).apply(lambda x: any(x.intersection(topic_rest_desk_1))*1)==1,'Outlet desk',df2['category'])
```

```
df2['category'].value_counts()
```

Figure 18: Topics for outlet desk