National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Siddhant Bakshi
Student ID: X21143846

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Siddhant Bakshi……

Student ID: X21143846……..……

Programme: Data Analytics…… **Year:** …2022……..

Module: ……MSc Academic Internship……..……

Supervisor: Vladimir Milosavljevic……..……

Submission Due Date: …15/12/2022……..……

Project Title: Study of Topic Modelling and Sentiment Analysis with Word Vectorization for a Hotel Review data …….

Word Count: ……530…… **Page Count**……12……..……..

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: ……Siddhant Bakshi……

Date: ……15/12/2022……

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

Siddhant Bakshi

X21143846

1 Introduction

The below steps show the specifications, tools and steps that are needed to configure the code. Sentiment analysis and topic modelling has been performed using machine learning and deep learning also word vectorization is done.

2 System Specification

Following are the system configuration:

- Operating System: Windows 11
- Processor: Intel Core i5 8th Gen
- Hard Drive: 500SSD
- RAM: 8GB

3 Software Tools

Some of the software tools used to implement this project are:

- Python
- Jupyter Notebook

3.1 Software Installation

This presents the processes taken in installing the tools used.

- Download and Installation of Python 3.9.6. The download link is https://www.python.org/downloads

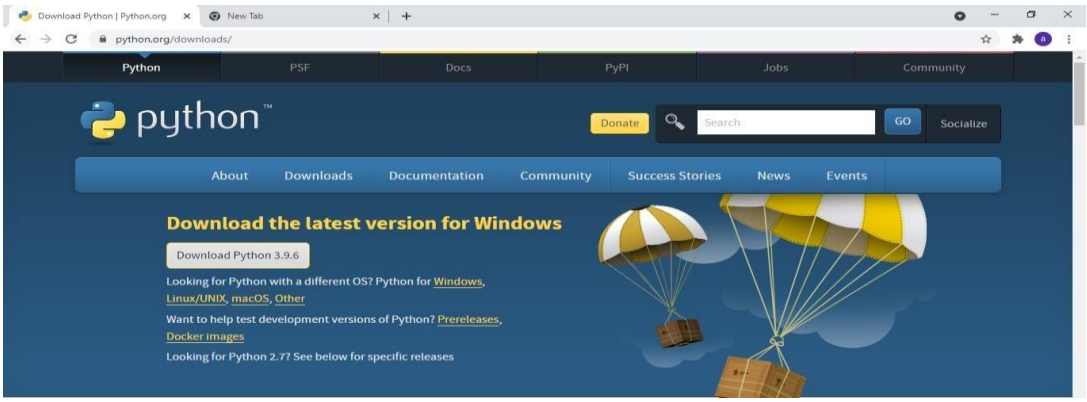


Fig 1: Python Download

Fig 2: Python Installation

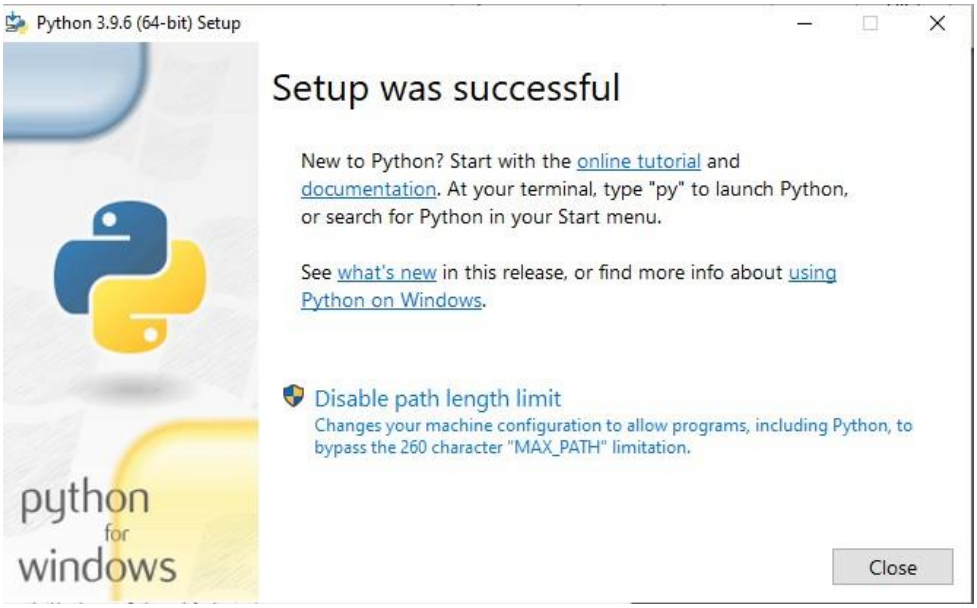


Fig 3: Completion of Installation

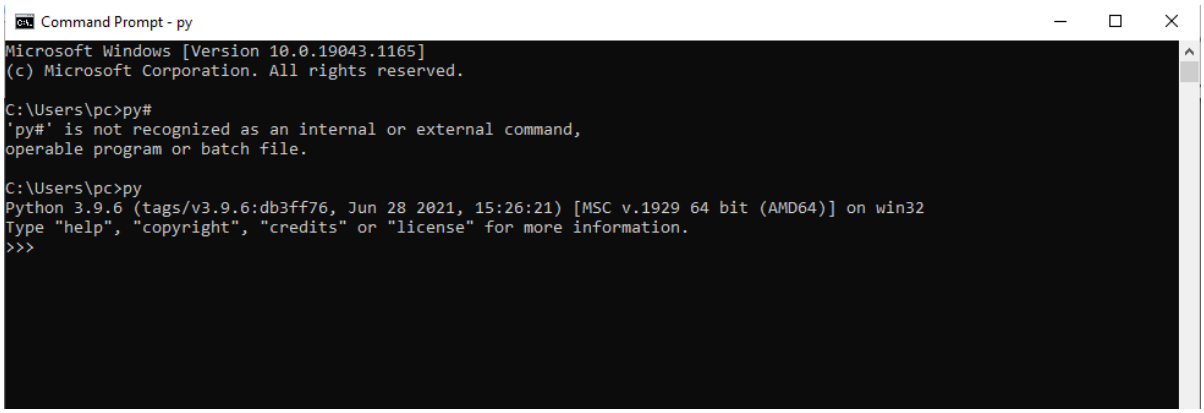


Fig 4: Confirmation of Python Installation

4 Implementation

The libraries from python used in implementing this project:

- Scikit-Learn
- Keras
- Pandas
- Pickle
- Numpy
- Genism
- Nltk
- Enchant
- Scacy
- Matplotlib
- Seaborn

```
reviews_data1.head()
```

	reviewer_display_name	reviewer_url	rating	review_date	review_title	review_text	date_of_stay	trip_type	additional_reviews	helpful	reviewer_us
0	OceanloverBC	/Profile/OceanloverBC	5	2022-06-09	Excellent Way to Start Trip	Best decision ever to start our organized tour...	2022-06-30	NaN	[]	0	Ocear
1	J2CAS	/Profile/J2CAS	5	2022-06-08	Great location and hotel and even better staff.	This hotel is attached right to the airport so...	2022-06-30	NaN	[]	0	
2	Paul Wylie	/Profile/paulwylie6F6	4	2022-06-08	Regular customer; we love our stays	Consistently great experience, and very conven...	2022-05-31	NaN	[{'rating': 4, 'ratingLabel': 'Value'}, {'rati...	0	paul
3	Tony and Jenny R	/Profile/F2972FNtonyr	5	2022-06-08	Great layover stay	Had to spend multiple hours in airport due to ...	2022-06-30	NaN	[]	0	F297

Fig 5: Checking the data

```
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2083 entries, 0 to 2082
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   reviewer_display_name  2083 non-null   object
 1   reviewer_url           2083 non-null   object
 2   rating                 2083 non-null   int64
 3   review_date            2083 non-null   object
 4   review_title           2083 non-null   object
 5   review_text            2083 non-null   object
 6   date_of_stay           2080 non-null   object
 7   trip_type              0 non-null      float64
 8   additional_reviews     2083 non-null   object
 9   helpful                2083 non-null   int64
 10  reviewer_username      2081 non-null   object
 11  reviewer_contributions 2083 non-null   int64
 12  reviewer_helpful       2083 non-null   int64
 13  photos                 2083 non-null   int64
 14  reviewer_following     2083 non-null   int64
 15  reviewer_location      1490 non-null   object
dtypes: float64(1), int64(6), object(9)
memory usage: 260.5+ KB
```

Fig 6: Data info

```
In [18]: df_reviews.isnull().sum()
```

```
Out[18]: reviewer_display_name        0
         reviewer_url                 0
         rating                       0
         review_date                  0
         review_title                 0
         review_text                  0
         date_of_stay                 3
         trip_type                 2083
         additional_reviews           0
         helpful                      0
         reviewer_username            2
         reviewer_contributions       0
         reviewer_helpful             0
         photos                       0
         reviewer_following           0
         reviewer_location          593
         dtype: int64
```

```
In [19]: df_reviews.drop(['trip_type'],axis=1)
         df_reviews.drop(['reviewer_location'],axis=1)
```

Fig 7: Treating null values

```
def remove_stop_words(text):
    """This funtion removes stop words such as 'he','i','it','and' 'if'"""
    stop_words = set(stopwords.words('english'))
    text = [word.lower() for word in text]
    return([token for token in text if token not in stop_words])

def remove_punct(text):
    """This removes punctuations from the given input text which remove_PropNouns fails to remove, hence ca
    puncts = "~`!@#€$%^&*()_-+={[}]|\:;'<,>.?/“”‘’"
    for indx in range(len(text)-1):
        tmp_val=text[indx]
        for punct in puncts:
            tmp_val = tmp_val.replace(punct,"")
        text[indx]=tmp_val
    text = list(filter(None, text))
    return text
#    return ([char for char in text if char not in char])

# Lemmatiation
def lemmatize_words(text):
    """This convertes a word to its root word. eg: Running will become Run"""
    stemmer = WordNetLemmatizer()
    return ([stemmer.lemmatize(token) for token in text])

# Stemming
def stem_words(text):
```

Fig 8: Removing punctuations, special characters

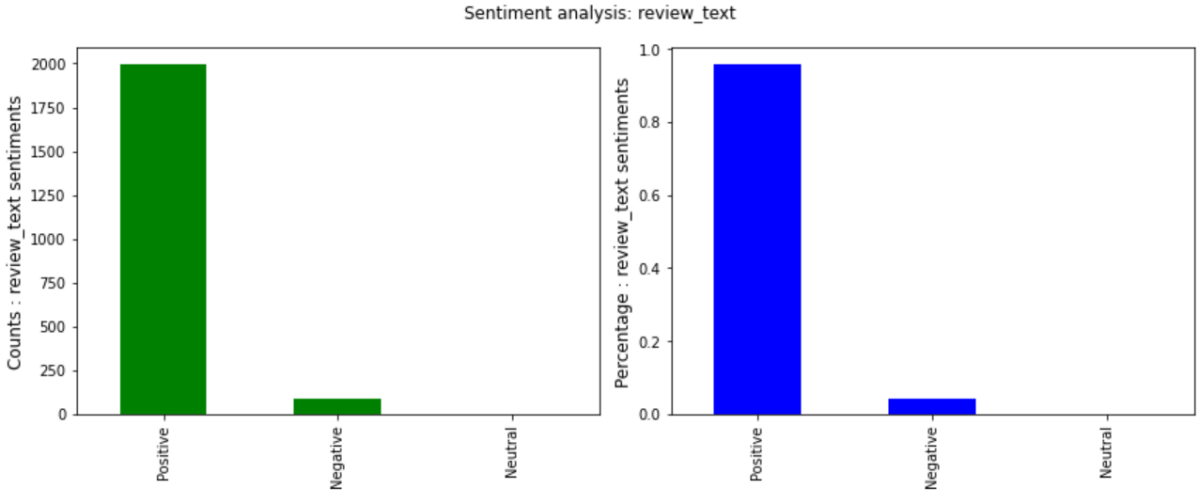


Fig 9: Count and percentage of positive and negatives in data

```
#Word2Vec Vectorization
X_train_Word2Vec = pd.DataFrame()
X_test_Word2Vec = pd.DataFrame()
X_train_Word2Vec['tokenized'] =  X_sentiment_train['review_cleansed'].apply(lambda x: tokenize(x))
X_test_Word2Vec['tokenized'] =  X_sentiment_test['review_cleansed'].apply(lambda x: tokenize(x))
X_train= pd.Series(X_train_Word2Vec['tokenized']).values
X_test= pd.Series(X_test_Word2Vec['tokenized']).values
w2v = gensim.models.Word2Vec(X_train,
                             vector_size=100,
                             window=5,
                             min_count=2)
```

Fig 10: Word2vec Vectorization

```
#TF-IDF Vectorization
#Spitting part of dataset for hyper parameter tuning
sentiment_paramter_tuning =  df_reviews[["review_cleansed","rating"]].iloc[:1000,:]
new_decision_reviews = df_reviews.iloc[1000:,:]
X_sent_para_tune = sentiment_paramter_tuning[["review_cleansed"]]
y_sent_para_tune = sentiment_paramter_tuning[["rating"]]
X_sent_para_tune["review_cleansed"].fillna(" ", inplace=True)
tfidf_tune_vectorizer = TfidfVectorizer(max_features = 2500)
X_sent_para_tune_vec_fit = tfidf_tune_vectorizer.fit(X_sent_para_tune["review_cleansed"])
X_sent_para_tune_vec = vectorize(X_sent_para_tune_vec_fit, X_sent_para_tune["review_cleansed"])

#Sentiment analysis Input and Response data
X_sentiment = new_decision_reviews[["review_cleansed"]]
y_sentiment = new_decision_reviews[["rating"]]
X_sentiment["review_cleansed"].fillna(" ", inplace=True)
X_sentiment_train, X_sentiment_test, y_sentiment_train, y_sentiment_test = train_test_split(X_sentiment, y_sentiment, tes
tfidf_vectorizer = TfidfVectorizer(max_features = 2500)
X_sentiment_train_fit = tfidf_vectorizer.fit(X_sentiment_train["review_cleansed"])
X_sentiment_train_vec = vectorize(X_sentiment_train_fit, X_sentiment_train["review_cleansed"])
X_sentiment_test_vec = vectorize(X_sentiment_train_fit, X_sentiment_test["review_cleansed"])
```

Fig 11: TFIDF Vectorization

```
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))
    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

```
print(cv.best_estimator_)
```

```
RandomForestClassifier(criterion='log_loss', max_depth=10, n_estimators=5)
```

```
scores = cross_val_score(randomForest_model, X_sentiment_train_vec, y_sentiment_train.values.ravel(), cv = 5)
print(scores)
scores.mean()
```

```
[0.59195402 0.58959538 0.58959538 0.58959538 0.58959538]

0.5900671051757358
```

```
df_reviews['sent']=df_reviews['rating'].apply(lambda x: 1 if int(x)>2.5 else 0)
```

```
#Topic Modelling Input and Response data
X_review = df_reviews["review_cleansed"]
X_review.dropna(inplace=True)
```

Fig 12: Hyper-parameterized tuning for Random Forest

```
#Hyperparameter tuning of SVM using GridSearchCV
svc = SVC()
parameters = {
    'kernel': ["linear", "poly", "rbf", "sigmoid"],
    'gamma': ["scale", "auto"],
    'degree' : [1, 2, 3]
}
# cv = GridSearchCV(rf, parameters, scoring = ["accuracy","f1", "roc_auc"], refit = "accuracy")
svm_cv = GridSearchCV(svc, parameters, scoring = "accuracy")
```

```
svm_cv.fit(X_sent_para_tune_vec, y_sent_para_tune["rating"].values.ravel())
```

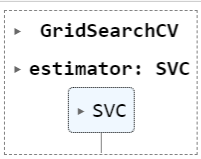


Fig 13: Hyper-parameterized tuning for SVM

```
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=max_length))
# LSTM
model.add(LSTM(50, dropout=0.2, recurrent_dropout=0.2))

#model.add(Dense(10, activation='relu'))
model.add(Dense(5, activation='softmax'))
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 693, 100)          571200

 lstm (LSTM)                 (None, 50)                30200

 dense (Dense)               (None, 5)                 255

=================================================================
Total params: 601,655
Trainable params: 601,655
Non-trainable params: 0
_________________________________________________________________
None
```

Fig 13: Building LSTM

```
# Define helper functions
def get_top_n_words(n, keys, document_term_matrix, count_vectorizer):
    '''
    returns a list of n_topic strings, where each string contains the n most common
    words in a predicted category, in order
    '''
    top_word_indices = []
    for topic in range(n_topics):
        temp_vector_sum = 0
        for i in range(len(keys)):
            if keys[i] == topic:
                temp_vector_sum += document_term_matrix[i]
        temp_vector_sum = temp_vector_sum.toarray()
        top_n_word_indices = np.flip(np.argsort(temp_vector_sum)[0][-n:],0)
        top_word_indices.append(top_n_word_indices)
    top_words = []
    for topic in top_word_indices:
        topic_words = []
        for index in topic:
            temp_word_vector = np.zeros((1,document_term_matrix.shape[1]))
            temp_word_vector[:,index] = 1
            the_word = count_vectorizer.inverse_transform(temp_word_vector)[0][0]
            topic_words.append(the_word.encode('ascii', 'ignore').decode('utf-8'))
        top_words.append(" ".join(topic_words))
    return top_words
```

Fig 14: LSA

```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence).encode('utf-8'), deacc=True))  # deacc=True removes punctuations
```

```
data_words = list(sent_to_words(X_positives["review_cleansed"]))
print(data_words[:1])
```

```
[['everything', 'marvelous', 'hotel', 'room', 'beautiful', 'garden', 'swimming', 'pool', 'beach', 'animation', 'team', 'great',
'hotel', 'cleanliness', 'everywhere', 'room', 'spacious', 'tidy', 'time', 'staff', 'friendly', 'enjoyed']]
```

```
# Create Dictionary
id2word = corpora.Dictionary(data_words)

# Create Corpus
texts = data_words

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 2), (11, 1), (12, 1), (13, 2), (14, 1),
(15, 1), (16, 1), (17, 1), (18, 1), (19, 1)]]
```

Fig 15:LDA built

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=4,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

```
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.015*"get" + 0.013*"bar" + 0.012*"restaurant" + 0.010*"one" + 0.009*"day" '
  '+ 0.008*"pool" + 0.007*"main" + 0.006*"room" + 0.006*"area" + '
  '0.006*"buffet"'),
 (1,
  '0.032*"hotel" + 0.016*"room" + 0.014*"food" + 0.013*"good" + 0.012*"pool" + '
  '0.011*"staff" + 0.011*"would" + 0.011*"time" + 0.010*"great" + 0.009*"day"'),
 (2,
  '0.012*"said" + 0.011*"star" + 0.009*"rude" + 0.006*"told" + 0.006*"asked" + '
  '0.005*"system" + 0.005*"broken" + 0.005*"sleep" + 0.005*"lesson" + '
  '0.005*"terrible"'),
 (3,
```

Fig 16: LDA

5 EVALUATION:

```
w2v.wv.index_to_key[:10]
```

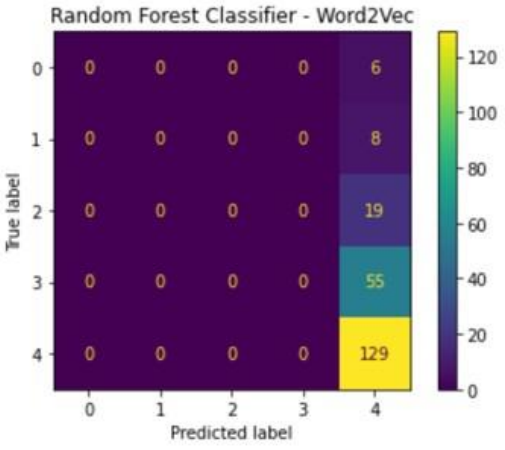
```
['hotel',
 'room',
 'food',
 'pool',
 'good',
 'staff',
 'would',
 'time',
 'great',
 'day']
```

```
# Find the most similar words
w2v.wv.most_similar('amazing')
```

```
[('fantastic', 0.9992648363113403),
 ('brilliant', 0.9990436434745789),
 ('enjoyed', 0.9989259839057922),
 ('fabulous', 0.9988408088684082),
 ('everything', 0.9988296627998352),
 ('perfect', 0.998756468296051),
 ('especially', 0.9987457394599915),
 ('fault', 0.9986956715583801),
 ('best', 0.9986544251441956),
 ('job', 0.9986509680747986)]
```

Fig 17: Word2vec output

```
Random Forest Classifier - Word2Vec:
precision : 0.59
recall : 0.59
accuracy : 0.59
```




```
Random Forest Classifier - TF-IDF:
precision : 0.59
recall : 0.59
accuracy : 0.59
```

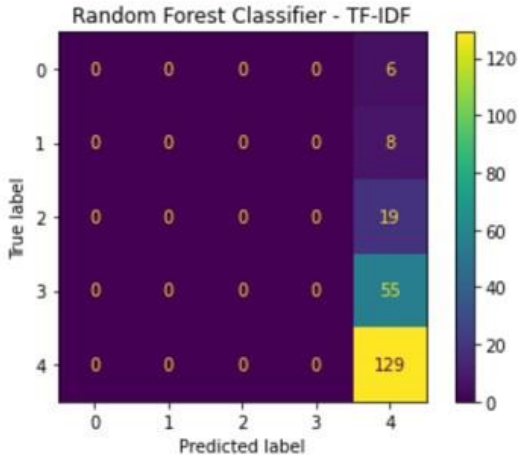


Fig 18: Random Forest output

```
# Instantiate and fit a basic Random Forest model on top of the vectors
rf = RandomForestClassifier(n_estimators = 5, criterion="log_loss", max_depth = 2,random_state = 32)
rf_model = rf.fit(X_train_vect_avg, y_sentiment_train.values.ravel())
rf_model.score(X_test_vect_avg, y_sentiment_test)
```

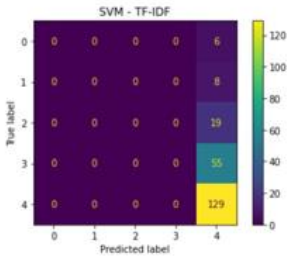
```
0.5944700460829493
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	6
2	0.00	0.00	0.00	8
3	0.00	0.00	0.00	19
4	0.00	0.00	0.00	55
5	0.59	1.00	0.75	129
accuracy			0.59	217
macro avg	0.12	0.20	0.15	217
weighted avg	0.35	0.59	0.44	217

```
[[  0   0   0   0   6]
 [  0   0   0   0   8]
 [  0   0   0   0  19]
 [  0   0   0   0  55]
 [  0   0   0   0 129]]
```

Fig 19: Accuracy and precision of Random Forest

```
SVM - TF-IDF:
precision : 0.59
recall : 0.59
accuracy : 0.59
```



The model score is higher in the case of SVM, it gives a score of 0.63

```
score = model.score(X_sentiment_test_vec, y_sentiment_test)
score
```

```
0.631336405529954
```

Fig 20: SVM output

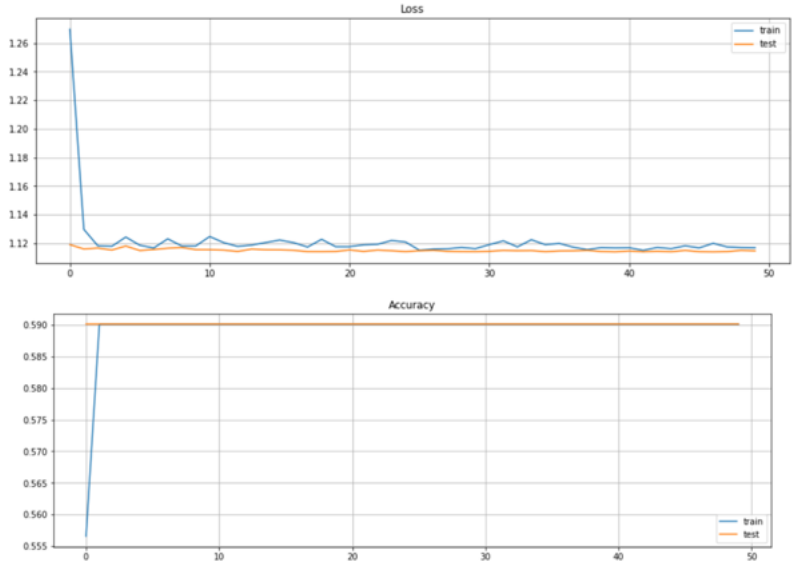


Fig 21: LSTM Loss and Accuracy

```
[(0,
  '0.015*"get" + 0.013*"bar" + 0.012*"restaurant" + 0.010*"one" + 0.009*"day" '
  '+ 0.008*"pool" + 0.007*"main" + 0.006*"room" + 0.006*"area" + '
  '0.006*"buffet"'),
 (1,
  '0.032*"hotel" + 0.016*"room" + 0.014*"food" + 0.013*"good" + 0.012*"pool" + '
  '0.011*"staff" + 0.011*"would" + 0.011*"time" + 0.010*"great" + 0.009*"day"'),
 (2,
  '0.012*"said" + 0.011*"star" + 0.009*"rude" + 0.006*"told" + 0.006*"asked" + '
  '0.005*"system" + 0.005*"broken" + 0.005*"sleep" + 0.005*"lesson" + '
  '0.005*"terrible"'),
 (3,
  '0.011*"boat" + 0.006*"waterfall" + 0.006*"coast" + 0.006*"diver" + '
  '0.005*"sail" + 0.005*"dive" + 0.005*"ala" + 0.004*"regarding" + '
  '0.004*"titanic" + 0.004*"co"')]


Topic 0:  hotel room good great pool food staff would time restaurant
Topic 1:  hotel friendly back amazing everybody came staff holiday perfect friend
Topic 2:  pool unfortunately swimming enjoy teenage plenty also inclusive clean good
Topic 3:  hotel stayed star internet nice struggled located excellent carte fact
```

```
[['everything', 'marvelous', 'hotel', 'room', 'beautiful', 'garden', 'swimming', 'pool', 'beach', 'animation', 'team', 'great',
'hotel', 'cleanliness', 'everywhere', 'room', 'spacious', 'tidy', 'time', 'staff', 'friendly', 'enjoyed']]
```

Fig 22: Topic Modelling output