National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Akshaansh Gautam
Student ID: x20151438

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akshaansh Gautam
Student ID:	x20151438
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	15/08/2018
Project Title:	Configuration Manual
Word Count:	XXX
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:	
Date:	14th August 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

Akshaansh Gautam
x20151438

1 Introduction

The purpose of the configuration manual is to explain how to implement the scripts for the research topic. The configuration manual also contains the hardware configuration of the machine on which the scripts are run. By following the procedure of the configuration manual will help to run the scripts for the project's outcome.

2 System Specification

- Device Name: LAPTOP-795QIV6D
- Processor: AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
- Installed ram: 8.00 GB (7.35 GB usable)
- System Type: 64-bit operating system, x64-based processor

Figure 1: Device Specification

3 Software Requirement

3.1 Google Colab

The scripts for this research project are written using python on google colab. So the initial step in order to execute the scripts of this research project is to sign up for a Gmail account as no execution can be performed without sign-in in Gmail. Colab is a free web IDE from Google Research that is widely used for machine learning and deep learning projects.

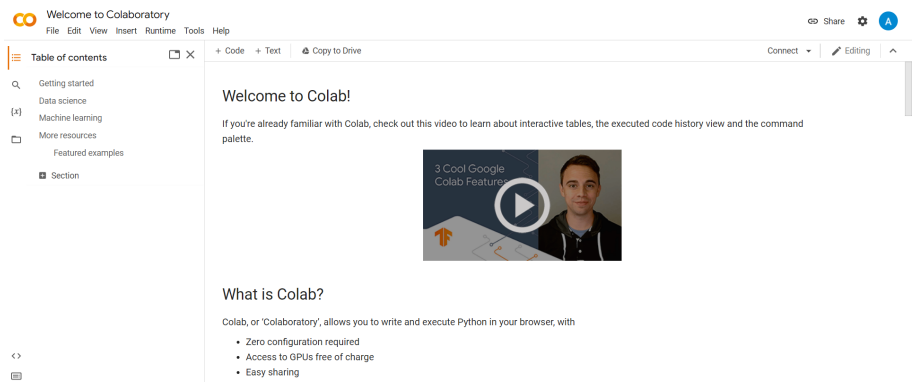

Figure 2: Google Colab

3.2 Data Source

This research project uses two datasets Amazon and Hotel booking which are available online. The available Amazon data is a subset of a larger data which was divided on the basis of categories of products available on amazon. The hotel dataset was developed by researchers for research purposes and provided proper citations to use their data. (Ott et al.; 2011) and (Ott et al.; 2013)

4 Project development

The script for this project can be executed from the google drive and most of the libraries used for this project are already installed on google colab. If in case the library is missing it can be install using the pip command. The way the command is used in google colab is as follows:
Command for colab: !pip install transformers

4.1 Import Libraries

The libraries that are used in this project for the implementation of the scripts are shown below. Import the required libraries to successfully execute the program. There are various libraries used in this project for data handling, visualization, machine learning algorithms, deep learning algorithms etc. Some of the libraries in the project are:

- Matplotlib
- Sklearn
- Numpy
- Pandas

```
#%% Importing the libraries

import pandas as pd
from textblob import TextBlob
import re
import matplotlib.pyplot as plt
import seaborn
import itertools
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud
from wordcloud import  STOPWORDS
from sklearn import ensemble
from sklearn import tree
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.preprocessing import Binarizer, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
```

Figure 3: Libraries Used

The drive should be mounted in order to access the data for the implementation of the project. The script to mount the drive is shown below.

```
from google.colab import drive
drive.mount('/gdrive/')
%cd /gdrive
```

Figure 4: Mount the drive

After the drive is mounted on the google colab, next step is to load the data. The script to load the data is shown below,

```
#loading the data
df1 = pd.read_csv("My Drive/Deceptive_Data_Training.csv")
df2 = pd.read_csv("My Drive/Amazon_Dataset.csv")
```

Figure 5: Loading the data

After the data is loaded data is then cleaned and preprocessed. The data is cleaned by removing hyperlinks, punctuations, whitespace, hashtags, tokenization, removing stopwords, converting to lower case.

```
def clean_text(text):
        '''
        Utility function to clean text text by removing links, special characters
        using simple regex statements.
        '''
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", str(text)).split())
```

Figure 6: Data Cleaning process

```
    def regularExpression(textToFilter):
        filteredtext = []
        retextPattern = 'RT|@RT'
        urlPattern = 'https://[a-zA-Z0-9+&@#/%?=~_|!:,.;]*'

        for textLine in textToFilter:
            text = re.sub(retextPattern,'',textLine)
            text = re.sub(urlPattern,'',text)
            filteredtext.append(text)
        return filteredtext

    def nltkTokenizer(textToTokenize):
        filteredSentence = []
        usersPattern = re.compile('@[a-zA-Z0-9]*',re.UNICODE)
        hashtagPattern = re.compile('#[a-zA-Z0-9]*',re.UNICODE)
        stop_words = stopwords.words('english')

        for textLine in textToTokenize:
            words = re.sub(usersPattern,'',textLine)
            words = re.sub(hashtagPattern,'',words)
            words = word_tokenize(words)
            for w in words:
                if w not in stop_words and w not in '@' and w not in '#':
                    filteredSentence.append(w)
        return filteredSentence

    def text_to_words(raw_text):
        text = ''.join(c for c in raw_text if c not in string.punctuation)
        text = re.sub('((www\S+)|(http\S+))', 'urlsite', text)
        text = re.sub(r'\d+', 'contnum', text)
```

Figure 7: Data Cleaning Process

```
def split_into_tokens(Text):
    return TextBlob(Text).words

def split_into_lemmas(Text):
    Text = Text.lower()
    words = TextBlob(Text).words
    # for each word, take its "base form" = lemma
    return [word.lemma for word in words]
```

Figure 8: Data Cleaning Process

After executing the scripts for data cleaning and preprocessing the output shows the transformation.

	Text	Polarity	clean_tweet
0	when i first checked the hotel's website and r...	deceptive	first checked hotels website reviews completel...
1	I had really high hopes for this hotel. The lo...	deceptive	really high hopes hotel location seemed great ...
2	Hotel Monaco is simply amazing. I travel quite...	deceptive	hotel monaco simply amazing travel quite bit u...
3	My experiences at the Fairmont Chicago were le...	truthful	experiences fairmont chicago less desirable tr...
4	I have traveled to Chicago many times for busi...	truthful	traveled chicago many times business expectati...

Figure 9: Cleaning Output

The polarity count of hotel dataset shows 800 truthful and 800 deceptive reviews.

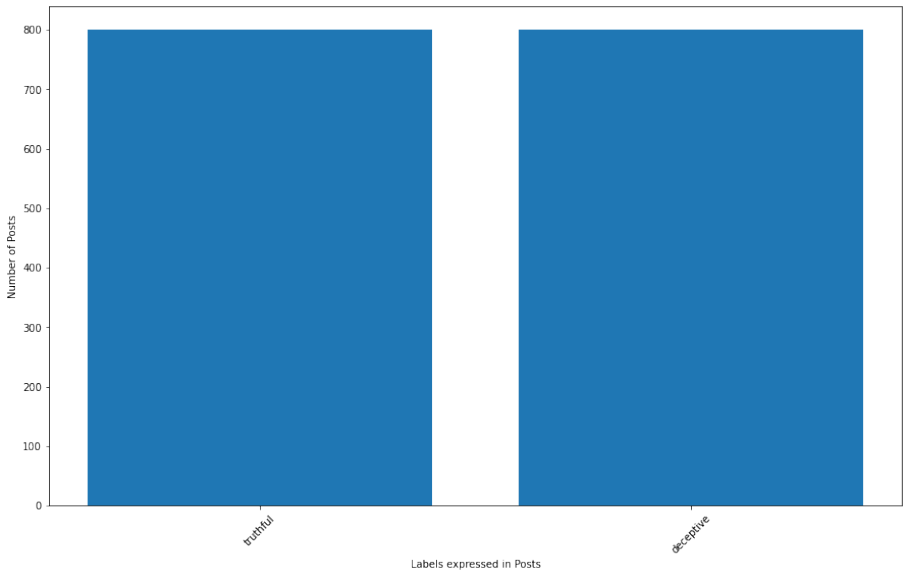

Figure 10: Polarity

After the data cleaning process the data was split into test and train data set after importing test_train_split in the ration of 70:30.

```
from sklearn.model_selection import train_test_split
X= tweet
Y=Polarity
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.30, random_state = 42)
```

Figure 11: Test and Train Split

4.2 Model Building

After the data is cleaned the implementation of machine learning algorithms is carried by using Countvectorizor and TF-IDF to convert the text into numerical data. The code for implementation is given below.

```
v = CountVectorizer(analyzer = "word")
train_features= v.fit_transform(train_clean_tweet)
test_features=v.transform(test_clean_tweet)


Classifiers = [
    LogisticRegression(C=0.001,multi_class='multinomial',max_iter=10,solver='sag', tol=1e-1),

    RandomForestClassifier(n_estimators=200, bootstrap=True, class_weight=None, criterion='gini',
            max_depth=50, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_jobs= -1,
            oob_score=False, random_state=10),

    AdaBoostClassifier(n_estimators=100, random_state=10),

    BernoulliNB(),

    MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True),

    KNeighborsClassifier(algorithm='auto', metric='minkowski',
           metric_params=None, n_neighbors=2, p=2,
           weights='uniform'),

    tree.DecisionTreeClassifier(),

    ensemble.ExtraTreesClassifier(n_estimators=100,
                                  max_features= 50,
                                  criterion= 'entropy'),
```

Figure 12: Machine Learning Algorithms(CountVectorizor)

```
    ensemble.GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.001,n_estimators=50, random_state=None, verbose = 1)]

dense_features=train_features.toarray()
dense_test= test_features.toarray()
Accuracy=[]
Model=[]
print("Entering into the Classifiers: ==========================================")
for classifier in Classifiers:
    try:
        fit = classifier.fit(train_features,train['Polarity'])
        pred = fit.predict(test_features)
    except Exception:
        fit = classifier.fit(dense_features,train['Polarity'])
        pred = fit.predict(dense_test)
    accuracy = accuracy_score(pred,test['Polarity'])
    print("======================================================================")
    print("**********************************************************************")
    print('Accuracy of '+classifier.__class__.__name__+'is '+str(accuracy))
    print("**********************************************************************")
    print("======================================================================")
    Accuracy.append(accuracy)
    print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(test['Polarity'], pred)))
    cm = metrics.confusion_matrix(test['Polarity'], pred)
    print("----------------------------------------------------------------------")
    print("Confusion matrix:\n%s" % cm)
    print("----------------------------------------------------------------------")
    Model.append(classifier.__class__.__name__)
    print("**********************************************************************")
```

Figure 13: Machine Learning Algorithms(CountVectorizor)

```
print("Using the TF-IDFVECTORIZER: ============================================")
v = TfidfVectorizer(min_df=5, max_df = 0.8, sublinear_tf=True, use_idf=True,stop_words='english')
train_features= v.fit_transform(train_clean_tweet)
test_features=v.transform(test_clean_tweet)


Classifiers = [
    LogisticRegression(C=0.001,multi_class='multinomial',max_iter=10,solver='sag', tol=1e-1),

    RandomForestClassifier(n_estimators=200, bootstrap=True, class_weight=None, criterion='gini',
            max_depth=50, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_jobs= -1,
            oob_score=False, random_state=10),

    AdaBoostClassifier(n_estimators=100, random_state=10),

    BernoulliNB(),

    MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True),

    KNeighborsClassifier(algorithm='auto', metric='minkowski',
           metric_params=None, n_neighbors=2, p=2,
           weights='uniform'),

    tree.DecisionTreeClassifier(),

    ensemble.ExtraTreesClassifier(n_estimators=100,
                                  max_features= 50,
                                  criterion= 'entropy'),
```

Figure 14: Machine Learning Algorithms(TF-IDF)

The following code is used to plot the machine learning algorithms in the form of bar chart for comparison.

```
print("Ploting the Model Performances: ========================================")
Index = [1,2,3,4,5,6,7,8,9]
plt.figure(1,figsize=(20, 10))
font = {'weight' : 'bold',
        'size'   : 25}

plt.rc('font', **font)

plt.bar(Index,Accuracy)
plt.xticks(Index, Model,rotation=45)
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.title('Accuracies of Models')
```

Figure 15: Machine Learning Algorithms(TF-IDF)

After implementing the machine learning algorithm, lstm, gru were implemented. import the important libraries that will be required to implement the code. The libraries that are used are:

```
import keras
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.preprocessing import sequence
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from keras.layers import Embedding
from keras.callbacks import ModelCheckpoint, EarlyStopping

import random
random.seed(45)
```

```
from sklearn.preprocessing import OneHotEncoder
```

Figure 16: Libraries Required

After importing the libraries, the algorithm is implemented, the code for implementation are given below,

The code for implementing BiLSTM is given below,

```
#word embedding dim:32
# bi-directional LSTM 2 layer: [64,32] Node
#then dense layer: 64 Node
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2)
])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
# tf.keras.metrics.AUC(),tf.keras.metrics.Precision()
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_data=(X_test, Y_test),callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)]
```

Figure 17: BiLSTM Implementation

The code for the implementation of LSTM is shown below,

```
epochs = 50
#word embedding dim:32
# LSTM 1 layer: 50 Node
model = Sequential([
Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]),
tf.keras.layers.LSTM(50, dropout=0.2, recurrent_dropout=0.2),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(2,activation="sigmoid", name="predictions")
])

# model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy',tf.keras.metrics.AUC(),tf.keras.metrics.Precision()])

history1 = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_data=(X_test, Y_test),callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)
```

Figure 18: LSTM Implementation

The code for the implementation of BiLSTM with 2 layer is shown below,

```
#word embedding dim:32
# bi-directional LSTM 2 layer: [64,32] Node
#then dense layer: 64 Node
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2,activation="sigmoid", name="predictions")
])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy',tf.keras.metrics.AUC(),tf.keras.metrics.Precision()])

# tf.keras.metrics.AUC(),tf.keras.metrics.Precision()
history2 = model.fit(X train, Y train, epochs=epochs, batch size=batch size,validation data=(X test, Y test),callbacks=[EarlyStopping(monitor='val loss', patience=3, min delta=0.0001)
```

Figure 19: BiLSTM Implementation with 2 layer

The code implementing LSTM with attention layer is shown below,

```
inputs=Input((MAX_SEQUENCE_LENGTH,))
x=Embedding(input_dim=MAX_NB_WORDS+1,output_dim=32,input_length=MAX_SEQUENCE_LENGTH,\
            embeddings_regularizer=keras.regularizers.l2(.001))(inputs)
att_in=LSTM(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(x)
att_out=attention()(att_in)
outputs=Dense(2,activation='sigmoid',trainable=True)(att_out)
model=Model(inputs,outputs)
model.summary()
```

```
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 250)]             0

 embedding_3 (Embedding)     (None, 250, 32)           1600032

 lstm_5 (LSTM)               (None, 250, 50)           16600

 attention (attention)       (None, 50)                300

 dense_4 (Dense)             (None, 2)                 102

=================================================================
Total params: 1,617,034
Trainable params: 1,617,034
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: LSTM with attention layer

After this GRU has been implemented. The difference between GRU and LSTM is that GRU has only two gates that are reset and update while the LSTM has three gates input, output and forget which makes it more complex than GRU. Because if this GRU uses less parameters and executes faster than LSTM.The code for implementation is given below,

The code for bi-directional GRU with 2 layer is shown below,

```
#word embedding dim:32
# bi-directional GRU 2 layer: [64,32] Node
#then dense layer: 64 Node
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2)
])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
# tf.keras.metrics.AUC(),tf.keras.metrics.Precision()
history4 = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_data=(X_test, Y_test),callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)
```

Figure 21: Bi-directional GRU 2 layer

The code for GRU single layer is shown below,

```
epochs = 50
#word embedding dim:32
# GRU 1 layer: 50 Node
model = Sequential([
Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]),
tf.keras.layers.GRU(50, dropout=0.2, recurrent_dropout=0.2),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(2,activation="sigmoid", name="predictions")
])

# model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy',tf.keras.metrics.AUC(),tf.keras.metrics.Precision()])

history5 = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_data=(X_test, Y_test),callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)
```

Figure 22: GRU 1 layer

After the complete execution of the code the output results shows that the accuracy increased on the merged dataset in comparison to the single dataset. SVM with TFIDF performed better with 95.39% accuracy.

```
**********************************************************************
Running Support Vector Machines on TF-IDF
**********************************************************************
predicted: ['truthful']
expected: truthful
['truthful' 'truthful' 'truthful' ... 'truthful' 'truthful' 'truthful']
accuracy 0.9539530842745438
confusion matrix
 [[141  35]
 [ 18 957]]
(row=expected, col=predicted)
```

Figure 23: SVM Results with TF-IDF

5 Other Software Used

Other than the software discussed above, the tool Overleaf was used to prepare the configuration manual.

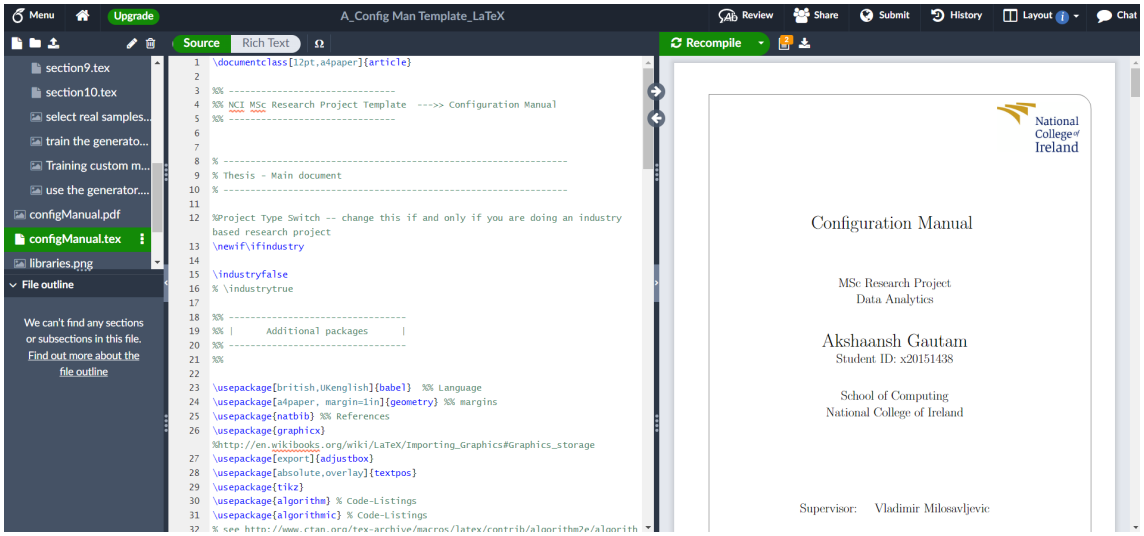

Figure 24: Overleaf

References

Ott, M., Cardie, C. and Hancock, J. T. (2013). Negative deceptive opinion spam, *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 497–501.

Ott, M., Choi, Y., Cardie, C. and Hancock, J. T. (2011). Finding deceptive opinion spam by any stretch of the imagination, *arXiv preprint arXiv:1107.4557* .