Configuration Manual

MSc Internship
Cybersecurity

Manoj Kumar Murugesan
Student ID: x18129668

School of Computing
National College of Ireland

Supervisor: Christos Grecos

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Manoj Kumar Murugesan
Student ID:	x18129668
Programme:	MSc. Cybersecurity **Year:** 2019
Module:	Internship
Supervisor:	Prof. Christos Grecos
Submission-Due Date:	12/12/2019
Project Title:	Comparative Analysis of Machine learning Algorithms using NLP Techniques in Automatic Detection Fake News on Social Media Platforms
Word Count:	1892 **Page Count** 22

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.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: ……………………………………………………………………………………………………

Date: 12.12.2019

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

Comparative Analysis of Machine learning Algorithms using NLP Techniques in Automatic Fake News Detection on Social Media Platforms.

Manoj Kumar Murugesan
X18129668

1 Introduction

Configuration manual demonstrates step by step instruction to implement the research topic **"Comparative Analysis of Machine learning Algorithms using NLP Techniques in Automatic Fake News Detection on Social Media Platforms."** The software and hardware requirement for the implementation are specified in the following sections. Required programming code, corresponding aim, and output results are displayed in sequence. The primary objective of the research is to design an API model that accurately classifies fake news with a low latency rate and evaluates fake rates out of the text. Machine learning algorithms such as LightGBM, XGBoost, AdaBoost, random-forest, and decision-tree were used along with NLP techniques. The following are the description of various technologies our achieve our results.

2 System Requirements

This section describes the system requirements to implement the project without hassles, and always the knowledge on the system specification is an advantage before computing experiments.

2.1 Hardware Requirements

The research was conducted both on the local system and on google cloud platform called **"Colab."** We used a local server to host API. Therefore, a part of the project was implemented using the local system.

2.1.1 Following are the specification used on the local system

1. Hard-Disk Memory – 1TB(HDD)

2.Processor – Intel® Core™ i5-6200U CPU @ 2.30GHz 2.40 GHz
3. RAM – 8GB
4.System OS – 64-bit Windows 10 OS.

2.1.2 Following are the specification cloud platform google 'Colab'

1. Memory Space – 358 BG
2. RAM – 25GB
3. Runtime Type – Python 3
4. Hardware Accelerator - GPU

2.2 Software Requirements

1. Python 3 - Python was used throughout the implementation process from cleaning the dataset till deploying the final API.

2. MicrosoftOffice365 Excel – was used to import and export datasets. A few cleaning processes were performed with coding through developer option in excel sheet. Datasets were used .CSV file format.

3. Jupyter Notebook – [2] Python code was programmed and executed in the Jupyter Notebook IDE platform. It is an open-source web application that allows users to code, execute, visualize, and share documents. Jupyter Notebook version 6.0.1 was used to code with Python 3.7.2

4. Google Colab – [1] The major part of the project, which is the evaluation of the classification model, was carried out in the google cloud platform called colab. It is a collaboration of jupyter environment on the google cloud environment. The platform is exclusively designed for data researchers to code, analyze data, visualize, and evaluate machine learning models. Google account is enough to get a session allocated with colab.

3 Data Pre-Processing and Evaluation

3.1 Step-by-Step Instruction – Google Colab

1. Sign-in to the google account
2. Open https://colab.research.google.com/notebooks/welcome.ipynb.
3. Choose File -> Python 3->Connect notebook for working environment.
4. Choose runtime->Change runtime type->choose GPU for fast execution.
5. Rename the default file name and saved it in google drive.

3.2 Installing Packages and Importing raw Data

Base Dataset was acquired from https://www.kaggle.com/c/fake-news/data. Datasets for upsampling 1s (fake news) was obtained from https://www.kaggle.com/jruvika/fake-news-detection and 0s (real news) from https://www.kaggle.com/snapcrack/all-the-news#articles1.csv. All three datasets are downloaded from the respective sources and saved in local drive in .csv format. Packages should be installed before importing and pre-processing of data.

```
import os
import re
import sys
import numpy as np
from scipy.sparse import hstack
import time
import pandas as pd
from nltk.stem.snowball import SnowballStemmer
from sklearn.preprocessing import LabelEncoder
import wordbatch
from wordbatch.extractors import WordBag, WordHash
from nltk.corpus import stopwords
import pickle as pkl
import gzip
from wordcloud import WordCloud, STOPWORDS
```

3.3 Importing Datasets

Required datasets are uploaded in google drive and imported from there .Panda library imported as pd is used to import data.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire

    Enter your authorization code:
    ··········
    Mounted at /content/drive
```

```
train_dataset = pd.read_csv('G:/spam-thesis/Kaggle-tweetsets/fakenewstraindata1.csv',encoding='ISO-8859-1')
train_fake_dataset = pd.read_csv('G:/spam-thesis/Kaggle-tweetsets/fakedataAdd1.csv')
train_real_dataset = pd.read_csv('Documents/realnewsadd.csv')
test_dataset = pd.read_csv('G:/spam-thesis/Kaggle-tweetsets/fakenewstestdata1.csv')
```

Variable Description:

Train_dataset and test_dataset are the base dataset, train_fake_dataset is the 1s fake news dataset, and train_real_dataset is 0s real news dataset.

3.4 Data Merging and cleaning

The base dataset and other two datasets for upsampling are imported and merged as follows.

```python
if __name__ == "__main__":
    print("Loading Datasets")
    train_dataset = pa.read_csv('G:/spam-thesis/Kaggle-tweetsets/fakenewstraindata1.csv',encoding='ISO-8859-1')
    train_fake_dataset = pa.read_csv('G:/spam-thesis/Kaggle-tweetsets/fakedataAdd1.csv')
    train_real_dataset = pa.read_csv('Documents/realnewsadd.csv')


    train_fake_dataset = train_fake_dataset[train_fake_dataset['language'] == 'english']

    print("Fill Missing Data")
    train_fake_dataset['author'].fillna('No auhtor', inplace=True)
    train_fake_dataset['title'].fillna('No title', inplace=True)
    train_fake_dataset['text'].fillna('No tweets', inplace=True)

    train_fake_dataset.drop(['thread_title', 'main_img_url', 'domain_rank'], axis=1, inplace=True)

    train_fake_dataset['label'] = 1

    train_dataset_final = train_fake_dataset[['title', 'author', 'text', 'label']]

    train_dataset.drop('id', axis=1, inplace=True)

    print("Merge both datasets")

    dataset_df = train_dataset.append(train_dataset_final)
    print("merged")

    print("Saving final Dataset")

    dataset_df.to_csv('G:/spam-thesis/Kaggle-tweetsets/final_Datasetfake3.csv', index=False)

    print("Done")

Loading Datasets
Fill Missing Data
Merge both datasets
merged
Saving final Dataset
Done
```

Description: First Base dataset and 1s fake dataset is filled to remove empty records, drop uncommon columns other than 'author', 'text', 'title', 'label'. Fake news dataset is added with a new column called label and filled in with 1s. Finally merged dataset is exported.

```python
    train_real_dataset = pa.read_csv('Documents/realnewsadd.csv')
    dataset_fake = pa.read_csv('G:/spam-thesis/Kaggle-tweetsets/final_Datasetfake3.csv',encoding='ISO-8859-1')

    train_real_dataset['label'] = 0

    train_real_dataset.drop(['id', 'publication', 'date', 'year', 'month'], axis=1, inplace=True)

    train_real_dataset_final = train_real_dataset[['title', 'text', 'author', 'label']]

    print("Filling the spaces")
    # fill nans
    train_real_dataset_final['author'].fillna('No author', inplace=True)
    train_real_dataset_final['title'].fillna('No title', inplace=True)
    train_real_dataset_final['text'].fillna('No news', inplace=True)

    print("Merging with Fake and train dataset")
    dataset_final = dataset_fake.append(train_real_dataset_final)
    dataset_final.dropna(inplace=True)

    print("Saving final Dataset")

    dataset_final.to_csv('G:/spam-thesis/Kaggle-tweetsets/final_Datasetv3.csv', index=False)

    print("Done")
Filling the spaces
Merging with Fake and train dataset
```

Description: In the second step, the previously exported dataset is merged with 0s real news. The real news dataset is added with label column 0s, and the final merged dataset is exported.

3.5 Removing Noises

Unwanted symbols and characters are removed to reduce space complexity and the efficiency of the algorithm. This is done using developer option in excel and following code.

```
Function Remove(Str As String) As String
    Dim xChars As String
    Dim I As Long
    xChars = "/."",0123456789_#$%@!()^*®Âš¯Ž&Ãƒ™€¢Â£Ð½Ð½Ñ‹Ðµœâ€©"
    For I = 1 To Len(xChars)
        Str = Replace$(Str, Mid$(xChars, I, 1), "")
    Next
    Remove = Str
End Function
```

Description: In the above code, special character which are to be removed are blacklisted and removed.

3.6 Data Pre-processing

3.6.1 Variable Encoding

The following code encoded the author column.

```
[ ] print("encoded")
    train['author'].fillna('no auth', inplace=True)
    lab = LabelEncoder()
    train['author_category'] = lab.fit_transform(train['author'])
    train.head()
```

encoded

	author	label	text	title	author_category
0	Daniel J. Flynn	0	Ever get the feeling your life circles the rou...	FLYNN: Hillary Clinton Big Woman on Campus - B...	862
1	Daniel Nussbaum	0	In these trying times Jackie Mason is the Voic...	Jackie Mason: Hollywood Would Love Trump if He...	869
2	Alissa J. Rubin	0	PARIS □ France chose an idealistic traditio...	Beno®t Hamon Wins French Socialist Partys Pres...	182
3	Megan Twohey and Scott Shane	0	A week before Michael T Flynn resigned as nati...	A Back-Channel Plan for Ukraine and Russia Cou...	2363
4	Aaron Klein	0	Organizing for Action the activist group that ...	Obamas Organizing for Action Partners with Sor...	65

3.6.2 Stemming, Normalization, Removal of empty cells and stopwords

Empty cells are filled in with the code below.

```
[ ] print("Fill up")
    data_append['author'].fillna('no auth', inplace=True)
    data_append['title'].fillna('no tit', inplace=True)
    data_append['text'].fillna('no txt', inplace=True)

Fill up
```

Texts were stemmed to extract root words from its branches using the following code.

```
[ ] print("stemming starts")
    data_append['title_stemmed'] = data_append['title'].map(lambda x: ' '.join([stemmer.stem(y) for y in x.split(' ')]))

    print("stemming_text")
    data_append['text_stemmed'] = data_append['text'].map(lambda x: ' '.join([stemmer.stem(y) for y in x.split(' ')]))

    stemming starts
    stemming_text
```

Texts were normalized, and stopwords were removed with the following code.

```
[ ] stemmer = SnowballStemmer("english")


    # Define helpers for text normalization
    stopwords = {x: 1 for x in stopwords.words('english')}
    non_alphanums = re.compile(u'[^A-Z a-z 0-9]+')

    def normalize_text(text):
        return u" ".join(
            [x for x in [y for y in non_alphanums.sub(' ', text).lower().strip().split(" ")] \
             if len(x) > 1 and x not in stopwords])
```

3.7 Evaluation of LightGBM and XGBoost Classification Models.

3.7.1 Bag of Words – Document-Term Matrix

Train Dataset size is calculated and stored in train_size variable and label column that has 0s and 1s are stored in variable 'y' as shown below.

```
[ ] train_size = train.shape[0]
    y = train['label']
    test_ids = test['id']
    test_size = test.shape[0]
    print(train.shape)

    (35952, 4)
```

Description: Once labels are stored in variable 'y', column label is dropped to reduce space complexity.

```
[ ] train.drop(['label'], axis=1, inplace=True)
    test.drop(['id'], axis=1, inplace=True)
    test.shape
    train.shape

    (35952, 4)
```

Description: Data is transformed using WordBatch library and from which WordBag function is imported to assign a weight to each word and generate features. The dataset is then normalized using Normalize_text function, which calls above mentioned function and does the pre-processing process. Finally, the transformed dataset is sparsed horizontally using hstack.

```
[ ] wb = wordbatch.WordBatch(normalize_text, extractor=(WordBag, {"hash_ngrams": 2, "hash_ngrams_weights": [0.5, -1.0], "hash_size": 2 ** 23, "norm": 'l2', "tf": '

    wb.dictionary_freeze = True

[ ] X_title = wb.transform(data_append['title_stemmed'])
    X_text = wb.transform(data_append['text_stemmed'])
    X_author = data_append['author_category'].values
    X_author = X_author.reshape(-1, 1)
    sparse_data = hstack((X_title, X_text, X_author)).tocsr()

    Normalize text
    Extract wordbags
    Normalize text
    Extract wordbags
```

3.7.2 LightGBM

Dataset split is done to train and validated with the remaining portion of the dataset, as shown below. Earl stopping round is given to stop when validation results go too weak.

```
[ ] X = sparse_data[:train_size]
    X_test = sparse_data[train_size:]

    train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.05, random_state=100)



    d_train = lgb.Dataset(train_X, label=train_y)
    d_valid = lgb.Dataset(valid_X, label=valid_y)
    watchlist = [d_train, d_valid]

[ ]  params = {

            'metric': 'binary_logloss',

        }

    model = lgb.train(params, train_set=d_train,  valid_sets=watchlist,
                      early_stopping_rounds=200, verbose_eval=1)
```

The Classification model is evaluated with metric imported from sklearn, as shown below.

```
[ ] lgbpreds = model.predict(valid_X)
    lgb_accuracy_before_tuning = accuracy_score(valid_y, np.round(lgbpreds))
    lgb_f1_before_tuning = f1_score(valid_y, np.round(lgbpreds))
    lgb_recall_before_tuning= recall_score(valid_y, np.round(lgbpreds))
    lgb_precision_before_tuning= precision_score(valid_y, np.round(lgbpreds))
    lgb_auc_before_tuning = metrics.roc_auc_score(valid_y, lgbpreds)
    print("LGB dev f1_score:", f1_score(valid_y, np.round(lgbpreds)))
    print("LGB dev accuracy_score:", accuracy_score(valid_y, np.round(lgbpreds)))
    print("LGB dev recall_score:", recall_score(valid_y, np.round(lgbpreds)))
    print("LGB dev precision_score:", precision_score(valid_y, np.round(lgbpreds)))
    print('Area under the curve : %f' % (metrics.roc_auc_score(valid_y, lgbpreds)))

    LGB dev f1_score: 0.9267480577136517
    LGB dev accuracy_score: 0.9265850945494994
    LGB dev recall_score: 0.9007551240560949
    LGB dev precision_score: 0.9542857142857143
    Area under the curve : 0.962087
```

The randomized search function is used to find the best combination of parameters for tuning, as shown below.

```
[ ] # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

    # Maximum number of levels in tree
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    num_leaves = [int(x) for x in np.linspace(10, 110, num = 11)]
    # Minimum number of samples required to split a node
    min_samples_split = [2, 5, 10]
    # Minimum number of samples required at each leaf node
    min_samples_leaf = [1, 2, 4]
    bagging_fraction = [i/10.0 for i in range(5,8)]
    bagging_frequency= range(2, 9, 1)
    subsample= [i/100.0 for i in range(70,90,5)]
    colsample_bytree = [i/100.0 for i in range(30,90,5)]
    min_split_gain= [i/10.0 for i in range(0,5)]
    learning_rate= [0.01, 0.03, 0.1, 0.3, 1]
    reg_alpha = [1e-5, 1e-2, 0.1, 1, 100]

    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'max_depth': max_depth,
                   'num_leaves': num_leaves,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'bagging_fraction': bagging_fraction,
                   'bagging_frequency': bagging_frequency,
                   'subsample': subsample,
                   'colsample_bytree': colsample_bytree,
                   'min_split_gain': min_split_gain,
                   'learning_rate': learning_rate,
                   'reg_alpha': reg_alpha,
                   }

    print(random_grid)

    lg = LGBMClassifier(n_jobs=-1)

    lg_random = RandomizedSearchCV(estimator = lg, param_distributions = random_grid, n_iter = 10, cv = 2, verbose=2, random_state=42)
    # Fit the random search model
    lg_random.fit(train_X, train_y)

    print (lg_random.best_params_)
```

The best parameters, as chosen by a randomized search, is displayed below.

```
Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed: 40.1min finished
'subsample': 0.85, 'reg_alpha': 0.01, 'num_leaves': 60, 'n_estimators': 2000, 'min_split_gain': 0.3, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_depth': 20, 'learning_rate': 0.03, 'colsample_byt
```

The model was executed with given best parameters from the hyper-tuning technique.

```
[ ]  params = {
            'subsample': 0.85, 'metric': 'binary_logloss', 'reg_alpha': 0.01, 'num_leaves': 60, 'n_estimators': 2000, 'min_split_gain': 0.3, 'min_s
        }

    modelt = lgb.train(params, train_set=d_train, valid_sets=watchlist, early_stopping_rounds=200,
                        verbose_eval=1)
```

3.7.3 XGBoost

XGBoost model was built with bag of words data transformation technique, and data splits were did as shown below.

```
X = sparse_data[:train_size]
X_test = sparse_data[train_size:]

train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_state=5)

print('train_X')
print(train_X.shape)
print(X_test)
print(valid_X.shape)
```

XGBoost built with default parameters.

```
[ ] from xgboost.sklearn import XGBClassifier
    xgbmodelt= XGBClassifier()
    print(xgbmodelt.get_xgb_params())

{'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsample_bytree': 1, 'gamma': 0, 'learning_rate': 0.1,
```

XGBoost algorithm was tuned using a randomized search technique, and the best combination can be found with code given below.

```
[ ]  from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
     import xgboost


     classifier=xgboost.XGBClassifier()
     params={
      "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
      "max_depth"        : [ 3, 4, 5, 6],
      "min_child_weight" : [ 1, 3, 5, 7 ],
      "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
      "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

     }

     random_search=RandomizedSearchCV(classifier,param_distributions=params,n_iter=3,scoring='roc_auc',n_jobs=-1,cv=2,verbose=3)

     from datetime import datetime
     # Here we go

     random_search.fit(train_X,train_y)
```

Hyper-parameter tuning gave the best parameters, as shown below.

```
[ ]  random_search.best_params_

{'colsample_bytree': 0.7,
 'gamma': 0.2,
 'learning_rate': 0.1,
 'max_depth': 5,
 'min_child_weight': 5}
```

XGBoost model with tuning parameters.

```
from xgboost.sklearn import XGBClassifier
xgbmodel= XGBClassifier(colsample_bytree= 0.7,
                        gamma=0.2,
                        learning_rate= 0.1,
                        max_depth= 5,
                        min_child_weight= 5)
print(xgbmodel.get_xgb_params())

{'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsample_bytree': 0.7, 'gamma': 0.2, 'learning_rate':
```

3.8 Evaluation Decision tree, Random-forest and AdaBoost Algorithms

3.8.1 TF-IDF – Document-Term Matrix

Ensemble-based machine learning algorithms are efficient with the TF-IDF transformation technique than Bag of words. Texts are vectorized and given n_gram of range from 1 to 3. Algorithms choose the best feature for the best accuracy rate.

```
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.feature_extraction.text import TfidfTransformer

test=test_en.fillna(' ')
train=train_en.fillna(' ')
test_en['total']=test_en['title_stemmed']+' '+test_en['author']+test_en['text_stemmed']
train_en['total']=train_en['title_stemmed']+' '+train_en['author']+train_en['text_stemmed']

#tfidf
transformer = TfidfTransformer(smooth_idf=False)
count_vectorizer = CountVectorizer(ngram_range=(1, 3))
counts = count_vectorizer.fit_transform(train_en['total'].values)
tfidf = transformer.fit_transform(counts)
```

Test and Train data is split into 30% and 70%. Label values are stored in the target variable for validation.

```
[ ] targets =train_en['label'].values
    test_counts = count_vectorizer.transform(test_en['total'].values)
    test_tfidf = transformer.fit_transform(test_counts)

    #split in samples
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(tfidf, targets,test_size=0.3, random_state=4)
```

3.8.2 AdaBoost

The AdaBoost algorithm was fed with the dataset and ran with default parameters. The code is given below.

```
[ ] from sklearn.ensemble import AdaBoostClassifier
    from sklearn.tree import DecisionTreeClassifier

    Adab= AdaBoostClassifier(DecisionTreeClassifier())
    Adab.fit(X_train, y_train)

    #
    adpred = Adab.predict(X_test)
    print('confusion matrix')
    print(metrics.confusion_matrix(y_test, adpred))
    print('classification report')
    print(metrics.classification_report(y_test, adpred))
    ad_accuracy_before_tuning = (metrics.accuracy_score(y_test, adpred))
    ad_f1score_before_tuning = metrics.f1_score(y_test, adpred)
    ad_recall_before_tuning = metrics.recall_score(y_test, adpred)
    ad_roc_before_tuning = metrics.roc_auc_score(y_test, adpred)
    print('Accuracy : %f' % (metrics.accuracy_score(y_test, adpred)))
    print('Area under the curve : %f' % (metrics.roc_auc_score(y_test, adpred)))
```

AdaBoost was fine-tuned with the hyper-parameter tuning technique. Randomized search technique was used to find the best combination of parameters, and code is as given below.

```
from sklearn.ensemble import AdaBoostClassifier
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint



base_est = [DecisionTreeClassifier(max_depth = i) for i in [10,13,16,19]]
params = {'base_estimator': base_est
          , 'n_estimators': np.logspace(2,3.2, 5).astype(int) #array([ 100,  199,  398,  794, 1584])
          , 'learning_rate': [0.01, 0.03, 0.1, 0.3, 1]
         }

adac = AdaBoostClassifier()
grid = RandomizedSearchCV(estimator = adac, param_distributions = params, n_iter =2, n_jobs = -1, cv = 2, scoring = 'accuracy', verbose = 2)

grid.fit(X_train, y_train)
```

The best parameters chosen for better results.

```
[ ]  grid.best_params_

{'base_estimator': DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=16,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best'),
 'learning_rate': 0.03,
 'n_estimators': 100}
```

The AdaBoost algorithm was optimized with the best parameters given by the hyper-parameter tuning technique, as shown below.

```
[ ]  from sklearn.tree import DecisionTreeClassifier

     Adabt= AdaBoostClassifier(DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=16,
                                                max_features=None, max_leaf_nodes=None,
                                                min_impurity_decrease=0.0, min_impurity_split=None,
                                                min_samples_leaf=1, min_samples_split=2,
                                                min_weight_fraction_leaf=0.0, presort=False,
                                                random_state=None, splitter='best'),
                                                learning_rate = 0.03,
                                                n_estimators = 100)
     Adabt.fit(X_train, y_train)

     #


     adtpred = Adabt.predict(X_test)
     ad_accuracy_after_tuning = (metrics.accuracy_score(y_test, adtpred))
     ad_f1score_after_tuning = metrics.f1_score(y_test, adtpred)
     ad_recall_after_tuning = metrics.recall_score(y_test, adtpred)
     ad_roc_after_tuning = metrics.roc_auc_score(y_test, adtpred)
     print('confusion matrix')
     print(metrics.confusion_matrix(y_test, adtpred))
     print('classification report')
     print(metrics.classification_report(y_test, adtpred))
```

3.8.3 Random-Forest Algorithm

The random-forest algorithm with default parameters is as given below. The performance was not good without tuning and hence tuned later.

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     from sklearn import metrics
     from sklearn.metrics import f1_score, accuracy_score , recall_score , precision_score
     Rand= RandomForestClassifier()

     Rand.fit(X_train, y_train)

     #

     rfpred = Rand.predict(X_test)
     print('confusion matrix without tuning')
     print(metrics.confusion_matrix(y_test, rfpred))
     print('classification report')
     rf_accuracy_before_tuning = metrics.accuracy_score(y_test, rfpred)
     rf_f1score_before_tuning = metrics.f1_score(y_test, rfpred)
     rf_precison_before_tuning = metrics.precision_score(y_test, rfpred)
     rf_recall_before_tuning = metrics.recall_score(y_test, rfpred)
     rf_auc_before_tuning = metrics.roc_auc_score(y_test, rfpred)
     print(metrics.classification_report(y_test, rfpred))
     print('Accuracy : %f' % (metrics.accuracy_score(y_test, rfpred)))
     print('Area under the curve : %f' % (metrics.roc_auc_score(y_test, rfpred)))
```

Hyper-parameter tuning with a randomized search technique is used to find the best combination of parameters to boost up a random forest algorithm from the impoverished state. The code is as given below

```
[ ]  from sklearn.model_selection import RandomizedSearchCV
     from sklearn.ensemble import RandomForestClassifier

      # Number of trees in random forest
     n_estimators = [10, 20, 100, 200, 400, 600]
     # Number of features to consider at every split
     max_features = ['auto', 'sqrt']
     # Maximum number of levels in tree
     max_depth = [10, 110]
     max_depth.append(None)
     # Minimum number of samples required to split a node
     min_samples_split = [2, 5, 10]
     # Minimum number of samples required at each leaf node
     min_samples_leaf = [1, 2, 4]
     # Method of selecting samples for training each tree
     bootstrap = [True, False]
     # Create the random grid
     random_grid = {'n_estimators': n_estimators,
                    'max_features': max_features,
                    'max_depth': max_depth,
                    'min_samples_split': min_samples_split,
                    'min_samples_leaf': min_samples_leaf,
                    'bootstrap': bootstrap}

     rf = RandomForestClassifier()
     # Random search of parameters, using 3 fold cross validation,
     # search across 100 different combinations, and use all available cores
     rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 10, cv = 2, verbose=2, random_state=42, n_jobs = -1)
     # Fit the random search model
     rf_random.fit(X_train, y_train)
```

The best combination of parameters given by the hyper-parameters technique is as given below.

```
[ ] rf_random.best_params_

{'bootstrap': False,
 'max_depth': 110,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 10,
 'n_estimators': 400}
```

The random-forest algorithm reacted very well with tuning parameters and got boosted up to 10%. The code is as follows.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import f1_score, accuracy_score , recall_score , precision_sc
Rand= RandomForestClassifier(bootstrap = False,
 max_depth= 110,
 max_features = 'auto',
 min_samples_leaf = 1,
 min_samples_split = 10,
 n_estimators = 400)

Rand.fit(X_train, y_train)

#

predrf = Rand.predict(X_test)
rf_accuracy_after_tuning = metrics.accuracy_score(y_test, predrf)
rf_f1score_after_tuning = metrics.f1_score(y_test, predrf)
rf_precison_after_tuning = metrics.precision_score(y_test, predrf)
rf_recall_after_tuning = metrics.recall_score(y_test, predrf)
rf_auc_after_tuning = metrics.roc_auc_score(y_test, predrf)
print('confusion matrix with tuning')
print(metrics.confusion_matrix(y_test, predrf))
print('classification report')
print(metrics.classification_report(y_test, predrf))
print('Accuracy : %f' % (metrics.accuracy_score(y_test, predrf)))
print('Area under the curve : %f' % (metrics.roc_auc_score(y_test, predrf)))
```

3.8.4 Decision Tree Algorithm

The decision tree algorithm with default parameters has given excellent performance, and code is as given below.

```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn import metrics
    from sklearn.metrics import f1_score, accuracy_score , recall_score , precision_score

    clf = DecisionTreeClassifier()

    # Train Decision Tree Classifer
    clf = clf.fit(X_train,y_train)

    #Predict the response for test dataset

    #
    dpred = clf.predict(X_test)
    print('confusion matrix')
    print(metrics.confusion_matrix(y_test, dtpred))
    print('classification report')
    print(metrics.classification_report(y_test, dpred))
    dcc_accuracy_before_tuning = metrics.accuracy_score(y_test, dpred)
    dcc_f1score_before_tuning = metrics.f1_score(y_test, dpred)
    dcc_recall_before_tuning = metrics.recall_score(y_test, dpred)
    dcc_roc_before_tuning = metrics.roc_auc_score(y_test, dpred)
    print('Accuracy : %f' % (metrics.accuracy_score(y_test, dpred)))
    print('Area under the curve : %f' % (metrics.roc_auc_score(y_test, dpred)))
```

Hyper-Parameter tuning was used to find the best combination of parameters. The result and the code is as shown below.

```
[ ] from sklearn.model_selection import RandomizedSearchCV
    from sklearn.tree import DecisionTreeClassifier
    from scipy.stats import randint

    param_dist = {"max_depth" : [3, None],
                  "min_samples_leaf": randint(1,9),
                  "criterion": ["gini","entropy"]}

    tree = DecisionTreeClassifier()

    tree_cv = RandomizedSearchCV(tree, param_dist, cv=3)

    tree_cv.fit(X_train, y_train)

    print("Tuned Decision Tree parameters: {}".format(tree_cv.best_params_))
    print("Best score is {}".format(tree_cv.best_score_))

    Tuned Decision Tree parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 8}
    Best score is 0.8781338080403501
```

Decision Tree with tuning parameters has shown a small improvement, and the code is as given below.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import f1_score, accuracy_score , recall_score , precision_score

clf = DecisionTreeClassifier(criterion= 'gini', max_depth=None, min_samples_leaf=8)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

#
dtpred = clf.predict(X_test)
dcc_accuracy_after_tuning = metrics.accuracy_score(y_test, dtpred)
dcc_f1score_after_tuning = metrics.f1_score(y_test, dtpred)
dcc_precison_after_tuning = metrics.precision_score(y_test, dtpred)
dcc_recall_after_tuning = metrics.recall_score(y_test, dtpred)
dcc_auc_after_tuning = metrics.roc_auc_score(y_test, dtpred)
print('confusion matrix')
print(metrics.confusion_matrix(y_test, dtpred))
print('classification report')
print(metrics.classification_report(y_test, dtpred))
print('Accuracy : %f' % (metrics.accuracy_score(y_test, dtpred)))
print('Area under the curve : %f' % (metrics.roc_auc_score(y_test, dtpred)))
```

4 Evaluation metrics

Finally, all the classification model is evaluated based on a few metrics imported from sklearn library like accuracy, precision, f1-score, recall rate, and AUC. The code for the evaluation and results of all the models together tabulated as follows.

```python
from tabulate import tabulate
table = [["Decision Tree without Hyper-parameter Tuning",dcc_accuracy_before_tuning, dcc_f1score_before_tuning, dcc_recall_before_tuning, dcc_precision_before_
         ["Decision Tree with Hyper-parameter Tuning",dcc_accuracy_after_tuning, dcc_f1score_after_tuning, dcc_recall_after_tuning, dcc_precison_after_tuning,

       ["Random forest without Hyper-parameter Tuning",rf_accuracy_before_tuning,
          rf_f1score_before_tuning,
          rf_recall_before_tuning,
          rf_precison_before_tuning,
          rf_auc_before_tuning],
         ["Random forest with Hyper-parameter Tuning",rf_accuracy_after_tuning,
          rf_f1score_after_tuning,
          rf_recall_after_tuning,
          rf_precison_after_tuning,

          rf_auc_after_tuning],

        ["AdaBoost without Hyper-parameter Tuning",ad_accuracy_before_tuning , ad_f1score_before_tuning, ad_recall_before_tuning, ad_precision_before_tuning ,
         ["AdaBoosst Tree with Hyper-parameter Tuning",ad_accuracy_after_tuning, ad_f1score_after_tuning,ad_recall_after_tuning,ad_precision_after_tuning,ad_rc

        ["XGBoost without Hyper-parameter Tuning",xg_accuracy_before_tuning,  xg_f1score_before_tuning, xg_recall_before_tuning, xg_precison_before_tuning,xg_a
         ["XGBoost with Hyper-parameter Tuning",xg_accuracy_after_tuning,xg_f1score_after_tuning ,xg_recall_after_tuning,xg_precison_after_tuning,xg_auc_after_

        ["LightGBM without Hyper-parameter Tuning",lgb_accuracy_before_tuning, lgb_f1_before_tuning, lgb_recall_before_tuning, lgb_precision_before_tuning,lgb_
         ["LightGBM with Hyper-parameter Tuning",lgb_accuracy_after_tuning,lgb_f1_after_tuning ,lgb_recall_after_tuning,lgb_precision_after_tuning,lgb_auc_afte

print (tabulate(table, headers=["Algorithm", "Accuracy", "f1-score", "Recall", "Precision","AUC"]))
```

Algorithm	Accuracy	f1-score	Recall	Precision	AUC
Decision Tree without Hyper-parameter Tuning	0.886241	0.892113	0.894394	0.889844	0.885797
Decision Tree with Hyper-parameter Tuning	0.891619	0.896098	0.888752	0.903567	0.891775
Random forest without Hyper-parameter Tuning	0.797144	0.793857	0.742772	0.852489	0.800111
Random forest with Hyper-parameter Tuning	0.908214	0.90944	0.87641	0.945057	0.909949
AdaBoost without Hyper-parameter Tuning	0.877897	0.88424	0.886812	0.881683	0.877411
AdaBoosst Tree with Hyper-parameter Tuning	0.903486	0.906612	0.890867	0.922922	0.904174
XGBoost without Hyper-parameter Tuning	0.905989	0.910138	0.914189	0.906123	0.905634
XGBoost with Hyper-parameter Tuning	0.917486	0.919978	0.910806	0.929337	0.917775
LightGBM without Hyper-parameter Tuning	0.926585	0.926748	0.900755	0.954286	0.962087
LightGBM with Hyper-parameter Tuning	0.931591	0.931931	0.908306	0.956818	0.963168

Interpretation: LightGBM has given outstanding performance with the highest AUC and Accuracy rates. Hence, an API model is designed with the LightGBM mechanism of classification and predicted fake news with high accuracy and efficiency.

4 API Model

4.1 Saving Models

LightGBM classification model with the best iteration is saved and exported as a text file that is to be used for the classification model. The code is as given below.

```
[ ] model.save_model('/content/drive/My Drive/model_copy.txt', num_iteration=model.best_itertaion)
    with open('/content/drive/My Drive/wb_transform.pkl', 'wb') as model file:
      pkl.dump(wb, model_file, protocol=2)
```

4.2 API Interface Design

The API is developed with the swagger tool, and data-preprocessing such as normalization, word bags are carried out before the classification model makes the prediction.

```
In [ ]: import re
        from flask import Flask
        from flask_restful import reqparse, abort, Api, Resource
        from flask_cors import CORS
        from gevent.pywsgi import WSGIServer
        from scipy.sparse import hstack
        from flasgger import Swagger
        import pickle as pkl
        import numpy as np
        import lightgbm as lgb
        from nltk.stem.snowball import SnowballStemmer
        from nltk.corpus import stopwords
        import pandas as pd



        app = Flask(__name__)
        #api = Api(app)
        cors = CORS(app, resources={r"/*": {"origins": "*"}})
        api = Api(app)
        swagger = Swagger(app)

        stemmer = SnowballStemmer("english")

        # Define helpers for text normalization
        stopwords = {x: 1 for x in stopwords.words('english')}
        non_alphanums = re.compile(u'[^A-Za-z0-9]+')

        #app = Flask(__name__)
        #api = Api(app)
        def normalize_text(text):
            return u" ".join(
```

```
        This examples uses FlaskRESTful Resource
        ---
        tags:
          - "is fake news?"
        summary: "Check if news is fake"
        description: ""
        operationId: "isfakenews"
        consumes:
          - "application/json"
        produces:
          - "application/json"
        parameters:
          - in: "body"
            name: "body"
            description: "Check if News is fake"
            required: true
            schema:
              $ref: "#/definitions/News"
        responses:
          '200':
            description: 'news checked'
          '400':
            description: 'invalid input, object invalid'
        definitions:
          News:
            type: object
            required:
              - "title"
              - "author"
              - "text"
            properties:
              title:
                type: "string"
                description: "Title of the news"
              author:
                type: "string"
                description: "Author of the news"
              text:
                type: "string"
                description: "Body of the news"
        """
```

```
            text:
              type: "string"
              description: "Body of the news"
        """
        # use parser and find the user's query
        args = parser.parse_args()
        title = args['title']
        author = model.encode_author(args['author'])
        text = args['text']

        X = model.vector_and_stack(title=title, text=text, author=author)

        prediction = model.predict(X)

        # Output either 'Negative' or 'Positive' along with the score
        if round(prediction[0]) == 0:
            pred_text = 'Reliable News'
        else:
            pred_text = 'Unreliable News'

        # round the predict proba value and set to new variable
        confidence = round(prediction[0], 3)

        # create JSON object
        output = {'prediction': pred_text, 'fake_rate': confidence}

        return output, 200

# Setup the Api resource routing here
# Route the URL to the resource
api.add_resource(PredictFakeNews, '/junknewsdetector')

if __name__ == '__main__':
    # Debug/Development
    # app.run(debug=True, host="0.0.0.0", port="5000")
    # Production
    http_server = WSGIServer(('', 5000), app)
    http_server.serve_forever()
```

4.3 Installing Packages

Before running the previous code, application file path must be created with package file ‘npm’. Therefore ‘npm’ is installed using ‘pip install ‘npm’ and use code ‘npm i -g junknewsdetector’ to create directory where all the necessary software packages to host swaggerAPI is imported.


```
{
  "author": "Megan Twohey and Scott Shane",
  "text": "A week before Michael T Flynn resigned as national security adviser a sealed proposal was   to his office outlining a way for President Trump to lift sanctions against Russia Mr Flynn is gone having been caught lying about his own discussion of sanctions with the Russian ambassador But the proposal a peace plan for Ukraine and Russia remains along with those pushing it: Michael D Cohen the presidents personal lawyer who delivered the document Felix H Sater a business associate who helped Mr Trump scout deals in Russia and a Ukrainian lawmaker trying to rise in a political opposition movement shaped in part by Mr Trumps former campaign manager Paul Manafort At a time when Mr Trumps ties to Russia and the people connected to him are under heightened scrutiny ”  with investigations by American intelligence agencies the F B I and Congress ”  some of his associates remain willing and eager to wade into  efforts behind the scenes Mr Trump has confounded Democrats and Republicans alike with his repeated praise for the Russian president Vladimir V Putin and his desire to forge an  alliance While there is nothing illegal about such unofficial efforts a proposal that seems to tip toward Russian interests may set off alarms The amateur diplomats say their goal is simply to help settle a grueling  conflict that has cost  lives Who doesnt want to help bring about peace?□ Mr Cohen asked But the proposal contains more than just a peace plan Andrii V Artemenko the Ukrainian lawmaker who sees himself as a  leader of a future Ukraine claims to have evidence ”  names of companies wire transfers□ ”  showing corruption by the Ukrainian president Petro O Poroshenko that could help oust him And Mr Artemenko said he had received encouragement for his plans from top aides to Mr Putin A lot of people will call me a Russian agent a U S agent a C I A agent□ Mr Artemenko said But how can you find a good solution between our countries if we do not talk?□ Mr Cohen and Mr Sater said they had not spoken to Mr Trump about the proposal and have no experience in foreign policy Mr Cohen is one of several Trump associates under scrutiny in an F B I counterintelligence examination of links with Russia according to law enforcement officials he has denied any illicit connections The two
```


Response body


```
{
  "prediction": "Unreliable News",
  "fake_rate": 0.541
}
```


Download

Response headers

```
access-control-allow-origin: http://localhost:5000
content-length: 54
content-type: application/json
date: Thu, 12 Dec 2019 11:26:30 GMT
vary: Origin
```

Description: Above screenshots depicts correct prediction of fake news text which is an actual fake news data.

5 Conclusion

Hence, step by step implementation, which is given in this report, works 100%, if any interested third-party people repeat it. Therefore, the research is successful that attained the objectives framed.

References

[1] Medium. (2019). *Google Colab Free GPU Tutorial*. [online] Available at: https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d [Accessed 12 Dec. 2019].

[2] Jupyter.org. (2019). *Project Jupyter*. [online] Available at: https://jupyter.org/ [Accessed 12 Dec. 2019].