In [ ]:
```
#!/usr/bin/env python3
import fasttext
import math
import random
import numpy as np
import pandas as pd
import json
import csv
import re
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn.svm import SVC
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import GaussianNB
import torch
from torch.utils.data import DataLoader, Dataset, TensorDataset, random_split, IterableDataset
from torch.utils.data.sampler import SequentialSampler
from transformers import BertTokenizer
import transformers as ppb
import logging
import gensim
from gensim.models import Word2Vec
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from gensim import corpora
from gensim.models import TfidfModel
from gensim.parsing.preprocessing import remove_stopwords
from gensim.utils import simple_preprocess
import transformers as ppb

nltk.download('words')
nltk.download('punkt')
stopwords.words('english')

import logging
logging.basicConfig(level=logging.ERROR)
import warnings
warnings.filterwarnings('ignore')

#needs to be GLOBAL
words = set(nltk.corpus.words.words())
stop_words = set(stopwords.words('english'))

def remove_digit(text):
    return re.sub(r'\d+', '', text)

def remove_non_english(text):
    text = [w for w in nltk.wordpunct_tokenize(text) if w in text or not w.isalpha()]
    return ' '.join(text)

def remove_special_chars(text):
    return re.sub("(\\d|\\W)+"," ",text)

def remove_shortwords(text):
    tokens = word_tokenize(text)
    text = [i for i in tokens if len(i) > 2]
    return ' '.join(text)

def remove_nonUTF8(data):
    return bytes(data, 'utf-8').decode('utf-8', 'ignore')

def preprocess(df):
    df['sentence'] = df['sentence'].str.replace(r'http(\S)+', r'')
    df['sentence'] = df['sentence'].str.replace(r'http(\S)+', r'')
    df['sentence'] = df['sentence'].str.replace(r'http ...', r'')
    df['sentence'] = df['sentence'].str.replace(r'(RT|rt)[ ]*@[ ]*[\S]+',r'')
    df['sentence'] = df['sentence'].str.replace(r'@[\S]+',r'')
    df['sentence'] = df['sentence'].str.replace(r'_[\S]?',r'')
    df['sentence'] = df['sentence'].str.replace(r'[ ]{2, }',r' ')
    df['sentence'] = df['sentence'].str.replace(r'&amp;?',r'and')
    df['sentence'] = df['sentence'].str.replace(r'&lt;',r'<')
    df['sentence'] = df['sentence'].str.replace(r'&gt;',r'>')
    df['sentence'] = df['sentence'].str.replace(r'([\w\d]+)([^\w\d ]+)', r'\1 \2')
    df['sentence'] = df['sentence'].str.replace(r'([^\w\d ]+)([\w\d]+)', r'\1 \2')
    df['sentence'] = df['sentence'].str.lower()
    df['sentence'] = df['sentence'].str.strip()
    df['sentence'] = df['sentence'].apply(remove_stopwords)
    df['sentence'] = df['sentence'].apply(remove_digit)
    df['sentence'] = df['sentence'].apply(remove_non_english)
    df['sentence'] = df['sentence'].apply(remove_special_chars)
    df['sentence'] = df['sentence'].apply(remove_nonUTF8)
    df['sentence'] = df['sentence'].str.replace("\'", "")
    df['sentence'] = df['sentence'].str.replace("\"", "")
    df['sentence'] = df['sentence'].apply(remove_shortwords)
    return df

#Bag of Words model ONLY
def bow_evaluate(fullsetdf,subsetdf):
    df = preprocess(fullsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    vectorizer = TfidfVectorizer(analyzer = 'word', strip_accents= 'ascii',smooth_idf = True, use_idf=True,max_df = 10000, min_df = 5,  stop_words = 'english')
    X = vectorizer.fit_transform(df['sentence'])
    tfidf_df = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
    y_train = pd.DataFrame(fullsetdf['label'])
    #dp = pd.concat([dp,y_label],axis=1)
    X_train = pd.DataFrame(tfidf_df)
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset TFxIDF features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
```

```
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    df = preprocess(subsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    vectorizer = TfidfVectorizer(analyzer = 'word', strip_accents= 'ascii',smooth_idf = True, use_idf=True,max_df = 10000, min_df = 5,  stop_words = 'english')
    X = vectorizer.fit_transform(df['sentence'])
    tfidf_df = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
    #print(dp.head)
    y_train = pd.DataFrame(subsetdf['label'])
    #dp = pd.concat([dp,y_label],axis=1)
    X_train = pd.DataFrame(tfidf_df)
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset TFxIDF features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()


def bert_based_features_evaluate(data):
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    torch.cuda.set_device(3)
    batch_size = 16
    if data == 'covid':
        dataset = FullDataset('/home/joao/covid.ORG.tsv','covid')
    if data ==  'crisislext6':
        dataset = FullDataset('/home/joao/crisislext6.ORG.tsv','crisislext6')
    if data == 'crisislext26':
        dataset = FullDataset('/home/joao/crisislext26.ORG.tsv','crisislext26')
    if data == 'crisismmd':
        dataset = FullDataset('/home/joao/crisismmd.ORG.tsv','crisismmd')
    dataloader = DataLoader(dataset,sampler = SequentialSampler(dataset), batch_size = batch_size )
    # Model 1.
    # Load pretrained model/tokenizer
    model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-base-uncased')
    model = model_class.from_pretrained(pretrained_weights)
    model.to(device)
     # For each batch of training data...
    _bertLabels = pd.DataFrame()    # dataframe with the Labels Features only
    _bertFeatures = pd.DataFrame()  # dataframe with the Bert features only
    for batch in dataloader:
        with torch.no_grad():
            b_input_ids, b_input_mask, hand_features, b_labels = tuple(t.to(device) for t in batch)
            last_hidden_states = model(b_input_ids,attention_mask = b_input_mask)
            bertfeatures = last_hidden_states[0][:,0,:]#Let's slice only the part of the output that we need. That is the output corresponding the first token of each sentence.  The way B
            bertfeatures = bertfeatures.cpu().detach().numpy()
            labels = b_labels.cpu().detach().numpy()
            _bertLabels = _bertLabels.append(pd.DataFrame(labels),ignore_index = True)
            _bertFeatures = _bertFeatures.append(pd.DataFrame(bertfeatures),ignore_index = True)
    # Model 2.
    # The output from BERT is going to be input to SKLEARN models
    X_train = _bertFeatures
    y_train = _bertLabels
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset BERT encoded features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    if data == 'covid':
        dataset = FullDataset('/home/joao/covid.subset.tsv','covid')
    if data ==  'crisislext6':
        dataset = FullDataset('/home/joao/crisislext6.subset.tsv','crisislext6')
    if data == 'crisislext26':
        dataset = FullDataset('/home/joao/crisislext26.subset.tsv','crisislext26')
```

```python
    if data == 'crisismmd':
        dataset = FullDataset('/home/joao/crisismmd.subset.tsv','crisismmd')
    dataloader = DataLoader(dataset,sampler = SequentialSampler(dataset), batch_size = batch_size )
    # Model 1.
    # Load pretrained model/tokenizer
    model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-base-uncased')
    model = model_class.from_pretrained(pretrained_weights)
    model.to(device)
     # For each batch of training data...
    _bertLabels = pd.DataFrame()    # dataframe with the Labels Features only
    _bertFeatures = pd.DataFrame()  # dataframe with the Bert features only
    for batch in dataloader:
        with torch.no_grad():
            b_input_ids, b_input_mask, hand_features, b_labels = tuple(t.to(device) for t in batch)
            last_hidden_states = model(b_input_ids,attention_mask = b_input_mask)
            bertfeatures = last_hidden_states[0][:,0,:]#Let's slice only the part of the output that we need. That is the output corresponding the first token of each sentence.  The way B
            bertfeatures = bertfeatures.cpu().detach().numpy()
            labels = b_labels.cpu().detach().numpy()
            _bertLabels = _bertLabels.append(pd.DataFrame(labels),ignore_index = True)
            _bertFeatures = _bertFeatures.append(pd.DataFrame(bertfeatures),ignore_index = True)
    # Model 2.
    # The output from BERT is going to be input to SKLEARN models
    X_train = _bertFeatures
    y_train = _bertLabels
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset dataset BERT encoded features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()

#Doc2Vec model
def doc2vec_evaluate(fullsetdf,subsetdf):
    size = 300
    window = 5
    min_count = 1
    workers = 4
    sg = 1
    df = preprocess(fullsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(df['tokenized_text'])]
    #Initialize the model
    doc2vec_model = Doc2Vec(documents, vector_size=size, window=window, min_count=min_count, workers=workers)
    for index, row in df.iterrows():
        model_vector = doc2vec_model.infer_vector(row['tokenized_text'])
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header = header.split(',')
            doc2vec_df = pd.DataFrame([], columns = header)
        #if type(model_vector) is list:
        line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
        line1 = line1.split(',')
        #else:
        #    line1 = ",".join([str(0) for i in range(size)])
        a_series = pd.Series(line1, index = doc2vec_df.columns)
        doc2vec_df =  doc2vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(fullsetdf['label'])
    X_train = doc2vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    #ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset doc2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    #scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    #print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*10
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    df = preprocess(subsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(df['tokenized_text'])]
    #Initialize the model
    doc2vec_model = Doc2Vec(documents, vector_size=size, window=window, min_count=min_count, workers=workers)
    for index, row in df.iterrows():
        model_vector = doc2vec_model.infer_vector(row['tokenized_text'])
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header = header.split(',')
            doc2vec_df = pd.DataFrame([], columns = header)
            # Check if the line exists else it is vector of zeros
        #if type(model_vector) is list:
```

```
        line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
        #else:
        #    line1 = ",".join([str(0) for i in range(size)])
        line1 = line1.split(',')
        a_series = pd.Series(line1, index = doc2vec_df.columns)
        doc2vec_df =  doc2vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(subsetdf['label'])
    X_train = doc2vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    #ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset doc2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    #scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    #print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*10
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()

def word2vec_evaluate(fullsetdf,subsetdf):
    size = 300
    window = 5
    min_count = 1
    workers = 4
    sg = 1
    df = preprocess(fullsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    w2v_model = Word2Vec(df['tokenized_text'].values, min_count = min_count, size = size, workers = workers, window = window, sg = sg)
    for index, row in df.iterrows():
        model_vector = (np.mean([w2v_model[token] for token in row['tokenized_text']], axis=0)).tolist()
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header = header.split(",")
            word2Vec_df = pd.DataFrame([], columns = header)
          # Check if the line exists else it is vector of zeros
        if type(model_vector) is list:
            line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
    #else:
       # line1 = ",".join([str(0) for i in range(size)])
#line1 = line1.split(',')
    a_series = pd.Series(line1, index = word2Vec_df.columns)
    word2Vec_df =  word2Vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(fullsetdf['label'])
    X_train = word2Vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset word2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    df = preprocess(subsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    w2v_model = Word2Vec(df['tokenized_text'].values, min_count = min_count, size = size, workers = workers, window = window, sg = sg)
    for index, row in df.iterrows():
        model_vector = (np.mean([w2v_model[token] for token in row['tokenized_text']], axis=0)).tolist()
        if index == 0:
            header = ",".join(str(ele) for ele in range(1000))
            header = header.split(",")
            word2Vec_df = pd.DataFrame([], columns = header)
            # Check if the line exists else it is vector of zeros
        if type(model_vector) is list:
            line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
        #else:
        #    line1 = ",".join([str(0) for i in range(1000)])
        #line1 = line1.split(',')
        a_series = pd.Series(line1, index = word2Vec_df.columns)
        word2Vec_df =  word2Vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(subsetdf['label'])
    X_pretrain = word2Vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset word2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
```

```python
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()



def handcrafted_features_evaluate(fullsetdf,subsetdf):
    X_train = fullsetdf[['nchars','nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex']]
    y_train = fullsetdf['label']
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset handcrafted features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    X_train = subsetdf[['nchars','nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex','usr_vrf', 'num_followers', 'num_friends', 'num_tweets']]
    y_train = subsetdf['label']
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset handcrafted features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()


def bow_plus_handcrafted_features_evaluate(fullsetdf,subsetdf):
    boW_features = bow_features(fullsetdf)
    H_train = fullsetdf[['nchars','nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex']]
    X_train = pd.concat([H_train,boW_features],axis=1)
    y_train = fullsetdf['label']
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset Bag-Of-Words  + handcrafted features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
    boW_features = bow_features(subsetdf)
    H_train = subsetdf[['nchars','nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex','usr_vrf', 'num_followers', 'num_friends', 'num_tweets']]
    X_train = pd.concat([H_train,boW_features],axis=1)
    y_train = subsetdf['label']
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Subset Bag-Of-Words + handcrafted features *** ")
```

```
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS             ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
    print()

def bow_features(fullsetdf):
    df = preprocess(fullsetdf)
    #df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    vectorizer = TfidfVectorizer(analyzer = 'word', strip_accents= 'ascii',smooth_idf = True, use_idf=True,max_df = 10000, min_df = 5,  stop_words = 'english')
    X = vectorizer.fit_transform(df['sentence'])
    bow_features = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
    return bow_features

class FullDataset():
    def __init__(self,filename,name):
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
        if name == 'crisismmd':
            self.df = pd.read_csv(filename,delimiter='\t',quoting=csv.QUOTE_NONE,error_bad_lines=False,encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisismmd.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisismmd.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if name == 'covid':
            self.df = pd.read_csv(filename,delimiter='\t',quoting=csv.QUOTE_NONE,error_bad_lines=False,encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/covid.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/covid.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if name ==  'crisislext6':
            self.df = pd.read_csv(filename,delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisislext6.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisislext6.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if name == 'crisislext26':
            self.df = pd.read_csv(filename,delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisislext26.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            #self.df = pd.read_csv('/home/joao/crisislext26.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        #self.df = self.df[['tweet_id','sentence','label']]
        self.sentences = self.df['sentence']
        #'Unnamed: 0'
        self.df.drop(self.df.columns[[0]], axis=1, inplace=True)
        self.labels = self.df['label'].values
        self.df['sentence'] = self.df['sentence'].str.replace(r'http(\S)+', r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'http(\S)+', r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'http ...', r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'(RT|rt)[ ]*@[ ]*[\S]+',r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'@[\S]+',r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'_[\S]?',r'')
        self.df['sentence'] = self.df['sentence'].str.replace(r'[ ]{2, }',r' ')
        self.df['sentence'] = self.df['sentence'].str.replace(r'&amp;?',r'and')
        self.df['sentence'] = self.df['sentence'].str.replace(r'&lt;',r'<')
        self.df['sentence'] = self.df['sentence'].str.replace(r'&gt;',r'>')
        self.df['sentence'] = self.df['sentence'].str.replace(r'([\w\d]+)([^\w\d ]+)', r'\1 \2')
        self.df['sentence'] = self.df['sentence'].str.replace(r'([^\w\d ]+)([\w\d]+)', r'\1 \2')
        self.df['sentence'] = self.df['sentence'].str.lower()
        self.df['sentence'] = self.df['sentence'].str.strip()
        self.df['sentence'] = self.df['sentence'].apply(remove_special_chars)
        self.df['sentence'] = self.df['sentence'].apply(remove_digit)
        self.df['sentence'] = self.df['sentence'].apply(remove_stopwords)
        self.df['sentence'] = self.df['sentence'].str.replace("\'", "")
        self.df['sentence'] = self.df['sentence'].str.replace("\"", "")
        self.df['sentence'] = self.df['sentence'].apply(remove_non_english)
        self.df['sentence'] = self.df['sentence'].apply(remove_nonUTF8)
        self.df['sentence'] = self.df['sentence'].apply(remove_shortwords)
        self.hand_crafted_features = self.df[['nchars', 'nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex']]
        self.hand_crafted_features_DF = pd.DataFrame(self.hand_crafted_features, columns = ['nchars', 'nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','bintj','tlex']).
        self.maxlen = 0
        #if name == 'covid':
        self.maxlen = 80
        #else:
        #    for sent in self.sentences:
        #        input_ids = self.tokenizer.encode(sent, add_special_tokens=True)
        #        self.maxlen = max(self.maxlen, len(input_ids))
    def __len__(self):
        return len(self.df)
    def __getitem__(self, idx):
        sentence = self.df.loc[idx, 'sentence']
        label = self.df.loc[idx, 'label']
        h_features = self.hand_crafted_features_DF.loc[idx,:]
        h_tensor = torch.tensor(h_features).to(device)
        tokens = self.tokenizer.tokenize(sentence)
        if len(tokens) == 0:
            tokens = ['']
        encoded_dict = self.tokenizer.encode_plus(tokens, add_special_tokens = True, max_length = self.maxlen, pad_to_max_length = True,return_attention_mask = True)
        tokens_ids = encoded_dict['input_ids']
        tokens_ids_tensor = torch.tensor(tokens_ids).to(device) #Converting the list to a pytorch tensor
        attn_mask = encoded_dict['attention_mask']
        attn_mask_tensor = torch.tensor(attn_mask).to(device)
        label_tensor = torch.tensor(label).to(device)
        return tokens_ids_tensor,attn_mask_tensor,h_tensor,label_tensor

def main():
    datasets = ['covid', 'crisislext6', 'crisislext26', 'crisismmd']
    #datasets = ['covid']
    for data in datasets :
        print("=== {} ===".format(data))
        #bert_based_features_evaluate(data)
        if data == 'covid':
            fullsetdf = pd.read_csv('/home/joao/covid.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            subsetdf = pd.read_csv('/home/joao/covid.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if data ==  'crisislext6':
            fullsetdf = pd.read_csv('/home/joao/crisislext6.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            subsetdf = pd.read_csv('/home/joao/crisislext6.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if data == 'crisislext26':
            fullsetdf = pd.read_csv('/home/joao/crisislext26.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
            subsetdf = pd.read_csv('/home/joao/crisislext26.subset.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
        if data == 'crisismmd':
            fullsetdf = pd.read_csv('/home/joao/crisismmd.ORG.tsv',delimiter='\t',quoting=csv.QUOTE_NONE,error_bad_lines=False, encoding='utf-8',lineterminator="\n")
            subsetdf = pd.read_csv('/home/joao/crisismmd.subset.tsv',delimiter='\t',quoting=csv.QUOTE_NONE,error_bad_lines=False,encoding='utf-8',lineterminator="\n")
        #bert_based_features_evaluate(data)
        #fast_text_evaluate(fullsetdf,subsetdf)
        ### Bag of Words Features
        #fullset_boW_df = boW_features(fullsetdf)
```

```
        #subsetdf_boW_df = boW_features(subsetdf)
        ### Word2Vec features
        #fullset_Word2Vec_df = Word2Vec_features(fullsetdf)
        #subsetdf_Word2Vec_df = Word2Vec_features(subsetdf)
        ### Doc2Vec features
        #fullset_Doc2Vec_df = Doc2Vec_features(fullsetdf)
        #subsetdf_Doc2Vec_df = Doc2Vec_features(subsetdf)
        #handcrafted_features_evaluate(fullsetdf,subsetdf)
        #bow_evaluate(fullsetdf,subsetdf)
        #bow_plus_handcrafted_features_evaluate(fullsetdf,subsetdf)
        #word2vec_evaluate(fullsetdf,subsetdf)
        #doc2vec_evaluate(fullsetdf,subsetdf)
        glove_evaluate(fullsetdf,subsetdf)


main()


###def error_analysis(fullsetdf,subsetdf):
fullsetdf = pd.read_csv('/home/joao/crisismmd.ORG.tsv',delimiter='\t',encoding='utf-8',lineterminator="\n")
X_train = fullsetdf[['nchars','nwords','bhash','nhash','blink','nlink','bat','nat','brt','bslang','tlex']]
y_train = fullsetdf['label']
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train,y_train)
y_pred = cross_val_predict(rf_clf, X_train, y_train, cv=10)
conf_mat = confusion_matrix(y_train, y_pred)
fullsetdf['ypred'] = y_pred
fullsetdf[['sentence','label','ypred']].to_csv('/home/joao/crisismmd.err.csv',sep='\t', encoding='utf-8')
```

```
In [ ]: ### method to calculate prediction with FastText
        def fast_text_evaluate(fullsetdf,subsetdf):
            ft = fasttext.load_model('cc.en.300.bin')
            df = preprocess(fullsetdf)
            df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
            words = []
            idx = 0
            word2idx = {}
            emb_dim = 300
            vectors = []
            fasttextDF = pd.DataFrame([], columns = list(range(emb_dim)))
            for i, tokens  in enumerate(df['tokenized_text']):
                target_vocab = tokens
                matrix_len = len(target_vocab)
                weights_matrix = np.zeros((matrix_len, emb_dim))
                words_found = 0
                for i, word in enumerate(target_vocab):
                    try:
                        weights_matrix[i] = ft.get_word_vector(word)
                        words_found += 1
                    except KeyError:
                        weights_matrix[i] = np.random.normal(scale=0.6, size=(emb_dim, ))
                weights_matrix = np.mean(weights_matrix,axis=0)
                fasttextDF = fasttextDF.append([weights_matrix])
            fasttextDF = fasttextDF.reset_index(drop=True)
            fasttextDF = fasttextDF.replace(np.nan,0)
            y_train = pd.DataFrame(fullsetdf['label'])
            X_train = fasttextDF
            lr_clf = LogisticRegression()
            dt_clf = DecisionTreeClassifier()
            rf_clf = RandomForestClassifier()
            ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
            nb_clf = GaussianNB()
            nn_clf = MLPClassifier(random_state=1, max_iter=300)
            svm_clf = svm.SVC(gamma=0.001, C=100.)
            scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
            print(" *** Full dataset fasttext features *** ")
            scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
            print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
            print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
            print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
            print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
            print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
            print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
            print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
            print()
            #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
            ### SUBSET ###
            #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
            df = preprocess(subsetdf)
            df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
            words = []
            idx = 0
            word2idx = {}
            emb_dim = 300
            vectors = []
            fasttextDF = pd.DataFrame([], columns = list(range(emb_dim)))
            for i, tokens  in enumerate(df['tokenized_text']):
                target_vocab = tokens
                matrix_len = len(target_vocab)
                weights_matrix = np.zeros((matrix_len, emb_dim))
                words_found = 0
                for i, word in enumerate(target_vocab):
                    try:
                        weights_matrix[i] = ft.get_word_vector(word)
                        words_found += 1
                    except KeyError:
                        weights_matrix[i] = np.random.normal(scale=0.6, size=(emb_dim, ))
                weights_matrix = np.mean(weights_matrix,axis=0)
                fasttextDF = fasttextDF.append([weights_matrix])
            fasttextDF = fasttextDF.reset_index(drop=True)
            fasttextDF = fasttextDF.replace(np.nan,0)
            y_train = pd.DataFrame(subsetdf['label'])
            X_train = fasttextDF
            lr_clf = LogisticRegression()
            dt_clf = DecisionTreeClassifier()
            rf_clf = RandomForestClassifier()
            ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
            nb_clf = GaussianNB()
            nn_clf = MLPClassifier(random_state=1, max_iter=300)
            svm_clf = svm.SVC(gamma=0.001, C=100.)
            scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
            print(" *** Subset Fasttext features *** ")
            scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
            print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
            print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
            print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
            print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
            #print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*10
            print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
            print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
            print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
            print()

        ### method to calculate prediction with Glove embeddings
        #http://nlp.stanford.edu/data/wordvecs/glove.6B.zip
        def glove_evaluate(fullsetdf,subsetdf):
            df = preprocess(fullsetdf)
            df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
            glove_path = "/home/joao/"
            words = []
            idx = 0
            word2idx = {}
            emb_dim = 300
            vectors = []
            with open(f'{glove_path}/glove.6B.300d.txt', 'rb') as f:
                for l in f:
                    line = l.decode().split()
                    word = line[0]
```

```
        words.append(word)
        word2idx[word] = idx
        idx += 1
        vect = np.array(line[1:]).astype(np.float)
        vectors.append(vect)
glove = {w: vectors[word2idx[w]] for w in words}
gloveDF = pd.DataFrame([], columns = list(range(emb_dim)))
for i, tokens  in enumerate(df['tokenized_text']):
    target_vocab = tokens
    matrix_len = len(target_vocab)
    weights_matrix = np.zeros((matrix_len, emb_dim))
    words_found = 0
    for i, word in enumerate(target_vocab):
        try:
            weights_matrix[i] = glove[word]
            words_found += 1
        except KeyError:
            weights_matrix[i] = np.random.normal(scale=0.6, size=(emb_dim, ))
    weights_matrix = np.mean(weights_matrix,axis=0)
    gloveDF = gloveDF.append([weights_matrix])
gloveDF = gloveDF.reset_index(drop=True)
gloveDF = gloveDF.replace(np.nan,0)
y_train = pd.DataFrame(fullsetdf['label'])
X_train = gloveDF
lr_clf = LogisticRegression()
dt_clf = DecisionTreeClassifier()
rf_clf = RandomForestClassifier()
ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
nb_clf = GaussianNB()
nn_clf = MLPClassifier(random_state=1, max_iter=300)
svm_clf = svm.SVC(gamma=0.001, C=100.)
scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
print(" *** Full dataset glove features *** ")
scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
print("Logistic Regression ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
print("Decision Tree       ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
print("Random Forest       ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
print("Adaboost            ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
print("NaiveBayes          ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
print("MLP                 ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
print("SVM                 ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
print()
#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
### SUBSET ###
#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@#
df = preprocess(subsetdf)
df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
gloveDF = pd.DataFrame([], columns = list(range(emb_dim)))
for i, tokens  in enumerate(df['tokenized_text']):
    target_vocab = tokens
    matrix_len = len(target_vocab)
    weights_matrix = np.zeros((matrix_len, emb_dim))
    words_found = 0
    for i, word in enumerate(target_vocab):
        try:
            weights_matrix[i] = glove[word]
            words_found += 1
        except KeyError:
            weights_matrix[i] = np.random.normal(scale=0.6, size=(emb_dim, ))
    weights_matrix = np.mean(weights_matrix,axis=0)
    gloveDF = gloveDF.append([weights_matrix])
gloveDF = gloveDF.reset_index(drop=True)
gloveDF = gloveDF.replace(np.nan,0)
y_train = pd.DataFrame(subsetdf['label'])
X_train = gloveDF
lr_clf = LogisticRegression()
dt_clf = DecisionTreeClassifier()
rf_clf = RandomForestClassifier()
ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
nb_clf = GaussianNB()
nn_clf = MLPClassifier(random_state=1, max_iter=300)
svm_clf = svm.SVC(gamma=0.001, C=100.)
scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
print(" *** Subset glove features *** ")
scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
print("MODELS              ,Accuracy , Precision  ,  Recall   ,  F-score          ", flush=True)
print("Logistic Regression ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mean()*100.0,scores_lr_clf['test_precision_macro'].mean()*100
print("Decision Tree       ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mean()*100.0,scores_dt_clf['test_precision_macro'].mean()*100
print("Random Forest       ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mean()*100.0,scores_rf_clf['test_precision_macro'].mean()*100
print("Adaboost            ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mean()*100.0,scores_ab_clf['test_precision_macro'].mean()*100
print("NaiveBayes          ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mean()*100.0,scores_nb_clf['test_precision_macro'].mean()*100
print("MLP                 ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mean()*100.0,scores_nn_clf['test_precision_macro'].mean()*100
print("SVM                 ,{:.2f}   ,    {:.2f}   ,  {:.2f}   , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].mean()*100.0,scores_svm_clf['test_precision_macro'].mean()*1
print()
```

In [ ]:
```
### Gensim method for calculation TF idf
def bag_of_Words_evaluate(fullsetdf,subsetdf):
    df = preprocess(fullsetdf)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    mydict = corpora.Dictionary(df['tokenized_text'])
    corpus = [mydict.doc2bow(line) for line in df['tokenized_text']]
    # TF-IDF Model
    tfidf_model = TfidfModel(corpus)
    tfidf_filename = '/home/joao/tfidf.csv'
    vocab_len = len(mydict.token2id)
    with open(tfidf_filename, 'w') as tfidf_file:
        for index, row in df.iterrows():
            doc = mydict.doc2bow(row['tokenized_text'])
            label = row['label']
            features = gensim.matutils.corpus2csc([tfidf_model[doc]], num_terms=vocab_len).toarray()[:,0]
            if index == 0:
                tfidf_file.write(header+str(",label"))
                tfidf_file.write("\n")
            line1 = ",".join( [str(vector_element) for vector_element in features] )
            tfidf_file.write(line1+str(",")+str(label))
            tfidf_file.write('\n')

    # Read the TFIDF vectors
    tfidf_df = pd.read_csv('/home/joao/tfidf.csv',low_memory=False)
    X_train = tfidf_df.iloc[:,:-1]
    y_train = tfidf_df['label']
```

In [ ]:

In [ ]:

In [ ]: