**PROJECT REPORT**

ON

MODEL GENERATION

BY

UMAR AYOUB HAJAM

(PROJECT LEAD)

And

Sayali Ghadage

(Model Generation Team)

Executive Summery

This is the second phase of the project twitter sentiment analysis on climate change. The Data used is the pre-processed file Frist\_Processed.csv. We have used the semi-supervised support vector machine to train the model. We downloaded the unlabelled data from Kaggle “climate-change.csv” and tuned the hyper parameters in the support vector machine to find the best results.

Machine and ide details

PyCharm 2021.1 (Community Edition)

Build #PC-211.6693.115, built on April 6, 2021

Runtime version: 11.0.10+9-b1341.35 amd64

VM: Dynamic Code Evolution 64-Bit Server VM by JetBrains s.r.o.

Windows 10 10.0

GC: ParNew, ConcurrentMarkSweep

Memory: 6933M

Cores: 8

Modules/Libraries

The modules/libraries used in model generation are

1. Sklearn
2. Pandas
3. Matplotlib
4. Numpy

Importing the required libraries

import pandas as pd  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split,\  
 RandomizedSearchCV, GridSearchCV, cross\_val\_score  
from sklearn import metrics  
from sklearn.metrics import classification\_report, confusion\_matrix,\  
 f1\_score, precision\_score, recall\_score, accuracy\_score  
from sklearn.svm import SVC  
from sklearn import preprocessing  
from sklearn.multiclass import OneVsRestClassifier

Loading the Dataset

Data = pd.read\_csv("C:/Users/Um Ar/PycharmProjects/Internship-2/First\_processed.csv")

X = Data["message"]  
Y = Data["sentiment"]

Splitting the dataset into training, validation and testing

# Splitting the data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.20, random\_state=1103)  
  
# Splitting the data into validation  
X\_test, x\_val, y\_test, y\_val = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=1103)

Using tfidf vectorizer with

Uni-grams, bi-grams and tri-grams

Maximum number of features = 20000

tfidf = TfidfVectorizer(ngram\_range=(1, 3), max\_features=20000, use\_idf=True)  
tfidf.fit\_transform(X\_train)  
tfidf.fit\_transform(x\_val)  
  
X\_train = tfidf.transform(X\_train)  
x\_val = tfidf.transform(x\_val)

Using Normalizer

MinMaxScaler = preprocessing.Normalizer()  
X\_train = MinMaxScaler.fit\_transform(X\_train)  
x\_val = MinMaxScaler.fit\_transform(x\_val)

Defining the Hyper-Parameters

param\_grid = {'C': [10, 15, 0.1, 1],  
 'gamma': [1.5, 2, 0.0001, 0.001, 0.1, 1],  
 'kernel': ['rbf', 'poly', ‘linear’]}

Defining the grid search and classifier

Here we have used 5 fold cross-validation

svm = SVC(verbose=True)  
grid = GridSearchCV(svm, param\_grid=param\_grid, cv=5, refit=True, n\_jobs=7, verbose=True)

Fitting the model to the data

grid.fit(X\_train, y\_train)

# Evaluating the model

predictions = svm.predict(x\_val)  
print("ACCURACY SCORE:", metrics.accuracy\_score(y\_val, predictions))  
print("::::Confusion Matrix::::")  
print(confusion\_matrix(y\_val, predictions))  
print("\n")  
  
print(":::Classification Report:::")  
print(classification\_report(y\_val, predictions, target\_names=['Class 1', 'Class 2', 'Class 3', 'Class 4']))  
print("\n")  
  
print(pd.crosstab(y\_val, predictions, rownames=["Orgnl"], colnames=['Predicted']))

Plotting the confusion matrix

class\_names = ["-1", "0", "1", "2"]  
disp = metrics.plot\_confusion\_matrix(rfc, tfidf.transform(X\_test), y\_test,  
 display\_labels=class\_names,  
 cmap=plt.cm.Blues)  
plt.show()

Results

The best hyper-parameters for svm are

C = 10.

Gamma = 1.

Kernel = rbf.

After the results we trained and tested the model even further.

PHASE 1 COMPLETE

PHASE 2

We have trained and tested the model, and download the unlabelled data from the Kaggle website. The dataset downloaded contained 400 samples.

The First step was to pre-process the unlabelled data, we used the same techniques.

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from nltk.stem import PorterStemmer  
import regex as re  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
  
  
Data = pd.read\_csv("Semi-Supervised\_SVM/Climate\_twitter.csv")  
# Data['word\_counts'] = Data['message'].str.split().str.len()  
# Data["Text Length"] = Data["message"].str.len()  
# Data.groupby('sentiment')['word\_counts'].mean()  
  
# Exploratory analysis  
Data.describe()  
print(Data.columns)  
Data.head()  
  
# print(Data["message"])  
# Data Visualization  
# sns.histplot(data=Data, x="sentiment", binwidth=0.4, color='lime')  
# sns.histplot(x=Data["sentiment"], y=Data["Text Length"], color='blue', binwidth=0.4)  
  
# Checking for missing values  
Data.isna().sum()  
  
  
# Cleaning the data  
def msg\_cleaning(msg):  
 # Removing @abc12  
 msg = re.sub(r'@[A-Za-z0-9]+', '', msg)  
 # Removing Hashtags  
 msg = re.sub(r'#', '', msg)  
 # Removing Chines  
 msg = re.sub(r'[^\x00-\x7F]+', '', msg)  
 # Removing Retweets  
 msg = re.sub(r'RT[\s]+', '', msg)  
 msg = re.sub(r'rt[\s]+', '', msg)  
 # Removing HyperLinks  
 msg = re.sub(r'https?:\/\/\s+', '', msg)  
 # Removing numeric values  
 msg = re.sub(r'\d+', '', msg)  
 msg = re.sub(r'aa[A-Za-z0-9]+', '', msg)  
 msg = re.sub(r'zz[A-Za-z0-9]+', '', msg)  
 return msg  
  
  
Data['text'] = Data['text'].apply(msg\_cleaning)  
Data["text"] = Data["text"].str.lower()  
# print(Data["message"])  
  
  
def identify\_tokens(row):  
 ide\_words = row["text"]  
 tokens = word\_tokenize(ide\_words)  
  
 token\_words = [w for w in tokens if w.isalpha()]  
 return token\_words  
  
  
Data["text"] = Data.apply(identify\_tokens, axis=1)  
print(Data['text'])  
  
  
stemming = PorterStemmer()  
  
  
def stem\_list(row):  
 my\_list = row["text"]  
 stemmed\_list = [stemming.stem(word) for word in my\_list]  
 return (stemmed\_list)  
  
  
Data["text"] = Data.apply(stem\_list, axis=1)  
print(Data["text"])  
  
stops = set(stopwords.words("english"))  
stops.update(["aa", "aaa", "aaaa", "aaaaa", "aaaaaa", "aaaaaaa", "aaaaaaaa", "aaaaaaaaa", "aaaaaaaaaaaaaaaaaaaah"])  
  
  
def remove\_stops(row):  
 my\_list = row["text"]  
 meningful\_words = [w for w in my\_list if not w in stops]  
 return(meningful\_words)  
  
  
Data["text"] = Data.apply(remove\_stops, axis=1)  
print(Data["text"])  
  
Data.to\_csv("SEMI.csv")

The pre-processed unlabelled dataset was saved as a csv file

PHASE 3

Labelling the Unlabelled data with our support vector machine model.

Reading the datasets

import pandas as pd  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.svm import SVC  
from sklearn import preprocessing  
  
Data = pd.read\_csv("C:/Users/Um Ar/PycharmProjects/Internship-2/First\_processed.csv")  
X = Data["message"]  
Y = Data["sentiment"]  
  
# Splitting the data  
val = pd.read\_csv("C:/Users/Um Ar/PycharmProjects/Internship-2/SEMI.csv")  
x\_val = val["text"]

Using tfidfvectorizer

tfidf = TfidfVectorizer(ngram\_range=(1, 3), max\_features=20000, use\_idf=True)  
tfidf.fit\_transform(X)  
tfidf.fit\_transform(x\_val)  
  
X = tfidf.transform(X)  
MinMaxScaler = preprocessing.Normalizer()  
  
X = MinMaxScaler.fit\_transform(X)  
x\_val = tfidf.transform(x\_val)  
x\_val = MinMaxScaler.fit\_transform(x\_val)

Fitting the model to the data

svm = SVC(C=10, gamma=1, kernel='rbf', verbose=True)  
svm.fit(X, Y)

Saving the predictions/labels for our unlabelled data

predictions = svm.predict(x\_val)  
val["sentiment"] = predictions  
val.to\_csv("SEMI\_PREDICTED.csv")

PHASE 4

Importing the modules and reading the datasets

import pandas as pd  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split,\  
 RandomizedSearchCV, GridSearchCV, cross\_val\_score  
from sklearn import metrics  
from sklearn.metrics import classification\_report, confusion\_matrix,\  
 f1\_score, precision\_score, recall\_score, accuracy\_score  
from sklearn.svm import SVC  
from sklearn import preprocessing  
from sklearn.multiclass import OneVsRestClassifier  
  
Data = pd.read\_csv("C:/Users/Um Ar/PycharmProjects/Internship-2/First\_processed.csv")  
val = pd.read\_csv("C:/Users/Um Ar/PycharmProjects/Internship-2/SEMI\_PREDICTED.csv")  
  
Data\_Set = pd.concat([Data, val])  
X = Data\_Set["message"]  
Y = Data\_Set["sentiment"]

Splitting the Dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.20, shuffle=True, random\_state=1103)

Defining and fitting the classifier to the data

svm = SVC(C=10, gamma=1, kernel='rbf', decision\_function\_shape='ovr', verbose=True)  
svm.fit(X\_train, y\_train)

Evaluating the model

predictions = svm.predict(X\_test)  
print("ACCURACY SCORE:", metrics.accuracy\_score(y\_test, predictions))  
print("::::Confusion Matrix::::")  
print(confusion\_matrix(y\_test, predictions))  
print("\n")  
  
print(":::Classification Report:::")  
print(classification\_report(y\_test, predictions, target\_names=['Class 1', 'Class 2', 'Class 3', 'Class 4']))  
print("\n")  
  
print(pd.crosstab(y\_test, predictions, rownames=["Orgnl"], colnames=['Predicted']))

Result/Finding

The result show that using semi-supervised support vector machine can improve the accuracy. In this test we only used 400 new samples/unlabelled data, using more sample may significantly increase the accuracy