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", "aaaaaaaaaaaaaaaaaaah"])


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