Text Emotion Detection

MINI PROJECT REPORT

Submitted by

ARVIND BALAAJEE J B (201804011)
JERRIN FREDRICK P (201804036)

MADHESH RAJ S(201804050)

in

MACHINE LEARNING USING TENSORFLOW

DEPARTMENT OF COMPUTER SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE
SIVAKASI

DECEMBER 2021

Text Emotion Detection

Introduction:-

Emotion Detection and Recognition from text is a recent field of research that is closely related to Sentiment Analysis and it aims to detect and recognize types of feelings through the expression of texts, such as anger, disgust, fear, happiness, sadness, and surprise. Emotion detection may have useful applications such as “Gauging how happy our citizens are”.

A human can express his emotions in any form, such as face, gestures, speech and text. The detection of text emotions is a content-based classification problem.

Novelty:-

Based on the given set of text data relevant emoji’s were mapped.

Detailed Explanation:-

From the given data set there are more than 7000 instances each instance has a vector representation with a sentence which signifies the emotion , where the values are embedded using one-hot encoding here 7 emoji expressions such as [angry,shame,joy,disgusting,fear,sad,guilt] for each expression the active state is represented as 1 in its corresponding bit position.

Usually, emotions are expressed as joy, sadness, anger, surprise, hate, fear, etc. Recognizing this type of emotion from a text written by a person plays an important role in applications such as chatbots, customer support forum, customer reviews etc. In the section below, through a machine learning project on Text Emotions Detection using Python where we will build a machine learning model to classify the emotions of a text.

In machine learning, the detection of textual emotions is the problem of content-based classification, which is the task of natural language processing.

Detecting a person's emotions is a difficult task, but detecting the emotions using text written by a person is even more difficult as a human can express his emotions in any form.

For detecting emotions from the text, few steps will be performed that will start with preparing the data. Then the next step will be tokenization where the textual data will be converted into tokens and from these tokens, we have to identify the emotional words.

These emotional words will be the keyword to classify the emotions of a text. Next, we'll frame this task in such a way that a text will be taken as an input and the emoji that represents the emotions in that text is generated as the output.

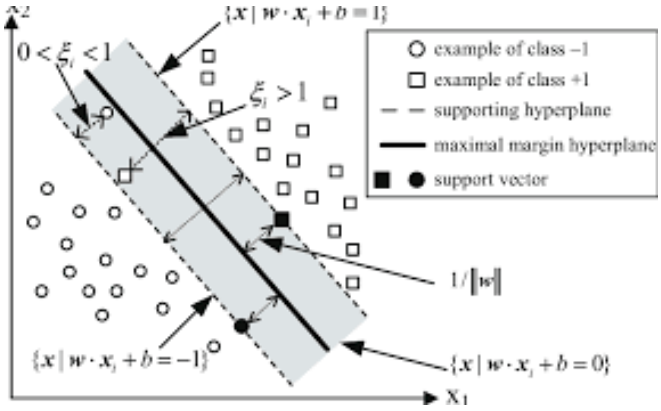
Steps involved:-

- Initiate this task by importing the necessary Python libraries and the dataset
- Create two Python functions for tokenization and generating the features of an input sentence such as N-gram is a function to tokenize the text input and features gets extracted over it.
- Create a Python function to store the labels, our labels will be based on emotions such as Joy, Fear, Anger, and so on.
- Split the data into training and test sets.
- Train four machine learning models such as Random forest , support vector Machine(SVM) ,Decision tree and then choose the model that works best on the training and testing sets.
- Random forest is an ensembling bagging technique, which creates decision trees on data samples and then gets the

prediction from each of them and finally selects the best solution by means of voting.

- The goal of the decision tree is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. Here samples are represented as sentence format.
- In Support Vector Machine(SVM), to create a linear SVM model in scikit-learn, there are two functions from the same module svm : SVC and LinearSVC . Since we want to create an SVM model with a linear kernel.
- The objective of a Linear SVC is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.
- From the given sentence features are extracted as keywords and assigns 1(active) to its corresponding expression index in vector and remaining entries are encoded as 0(inactive). Vectorizer function used for transforming into vector format.
- In this model, linear SVC provides **best accuracy** when compared to other classifiers .
- Assign an emoji to each label that is emotions in this problem, then write some input sentences for testing the model , then use the trained model to take a look at the emotions of corresponding input sentences.

Linear SVM diagram:-



Code:-

Importing Libraries

```
import re
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction import DictVectorizer
```

Load the dataset

```
def read_data(file):
    data = []
    with open(file, 'r')as f:
        for line in f:
            line = line.strip()
            label = ' '.join(line[1:line.find("]")].strip().split())
            text = line[line.find("]")+1:].strip()
            data.append([label, text])
    return data
file = '/content/drive/MyDrive/text.txt'
data = read_data(file)
print("Number of instances: {}".format(len(data)))
```

Tokenize the words and create features

```
def ngram(token, n):
    output = []
    for i in range(n-1, len(token)):
        ngram = ' '.join(token[i-n+1:i+1])
        output.append(ngram)
    return output
def create_feature(text, nrange=(1, 1)):
    text_features = []
    text = text.lower()
    text_alphanum = re.sub('[^a-z0-9#]', ' ', text)
    for n in range(nrange[0], nrange[1]+1):
        text_features += ngram(text_alphanum.split(), n)
    text_punc = re.sub('[a-z0-9]', ' ', text)
    text_features += ngram(text_punc.split(), 1)
    return Counter(text_features)
```

Label conversion

```
def convert_label(item, name):
    items = list(map(float, item.split()))
    label = ""
    for idx in range(len(items)):
        if items[idx] == 1:
            label += name[idx] + " "

    return label.strip()
emotions = ["joy", 'fear', "anger", "sadness", "disgust", "shame", "gui
lt"]
X_all = []
y_all = []
for label, text in data:
    y_all.append(convert_label(label, emotions))
    X_all.append(create_feature(text, nrange=(1, 4)))
```

Splitting the data set into train data and test data

```
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_
size = 0.2, random_state = 123)
def train_test(clf, X_train, X_test, y_train, y_test):
    clf.fit(X_train, y_train)
    train_acc = accuracy_score(y_train, clf.predict(X_train))
    test_acc = accuracy_score(y_test, clf.predict(X_test))
    return train_acc, test_acc
```

Vectorize the words

```
vectorizer = DictVectorizer(sparse = True)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

Compile the model

```
svc = SVC()
lsvc = LinearSVC(random_state=123)
rforest = RandomForestClassifier(random_state=123)
dtree = DecisionTreeClassifier()
clifs = [svc, lsvc, rforest, dtree]
print("| {:25} | {} | {} |".format("Classifier", "Training Accuracy", "
Test Accuracy"))
print("| {} | {} | {} |".format("-"*25, "-"*17, "-"*13))
for clf in clifs:
    clf_name = clf.__class__.__name__
    train_acc, test_acc = train_test(clf, X_train, X_test, y_train, y_t
est)
    print("| {:25} | {:17.7f} | {:13.7f} |".format(clf_name, train_acc,
 test_acc))
```

Accuracy for the classifiers

Classifier	Training Accuracy	Test Accuracy
SVC	0.9067513	0.4512032
LinearSVC	0.9988302	0.5768717
RandomForestClassifier	0.9988302	0.5541444
DecisionTreeClassifier	0.9988302	0.4659091

Assign the labels

```
l = ["joy", 'fear', "anger", "sadness", "disgust", "shame", "guilt"]
l.sort()
label_freq = {}
for label, _ in data:
    label_freq[label] = label_freq.get(label, 0) + 1
for l in sorted(label_freq, key=label_freq.get, reverse=True):
    print("{:10}({})  {}".format(convert_label(l, emotions), l, label_f
req[l]))
```

Test the Model

```
emoji_dict = {"joy":"□", "fear":"😱", "anger":"😠", "sadness":"☹", "dis
gust":"😒", "shame":"😳", "guilt":"😔"}
t1 = "My dog died yesterday"
t2 = "I feel shame on his behavior"
t3="I have a fear on him"
t4="Someone attacked my wife with a knife."
t5="My friend had cheated me"
texts = [t1, t2, t3, t4,t5]
for text in texts:
    features = create_feature(text, nrange=(1, 6))
    features = vectorizer.transform(features)
    prediction = clf.predict(features)[0]
    print( text,emoji_dict[prediction])
```

Visualization

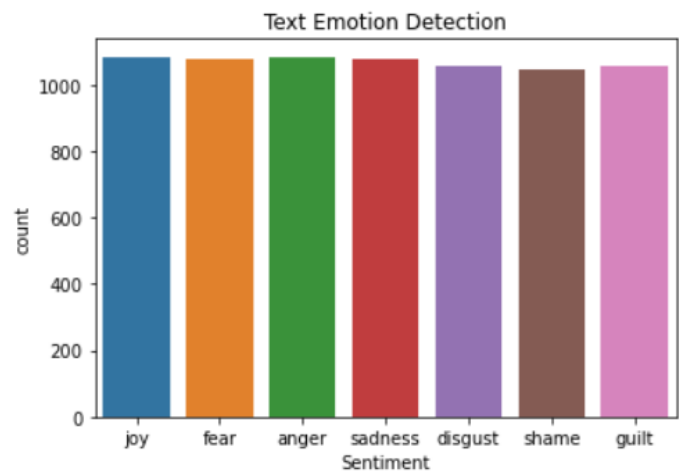
```
sns.countplot(df_train.Sentiment)

plt.title("Text Emotion Detection")

plt.show()
```

Output

```
joy        (1. 0. 0. 0. 0. 0. 0.)  1084
anger      (0. 0. 1. 0. 0. 0. 0.)  1080
sadness    (0. 0. 0. 1. 0. 0. 0.)  1079
fear       (0. 1. 0. 0. 0. 0. 0.)  1078
disgust    (0. 0. 0. 0. 1. 0. 0.)  1057
guilt      (0. 0. 0. 0. 0. 0. 1.)  1057
shame      (0. 0. 0. 0. 0. 1. 0.)  1045
```



```
My dog died yesterday 😢
I feel shame on his behavior 😳
I have a fear on dogs 😱
Someone attacked my brother with a knife. 😠
My friend had cheated me 😒
```

Conclusion

Thus the model gets trained and implemented for finding the emotion from the given text using machine learning algorithms