Identify and Classify Emotions in Text

Introduction

Nowadays the emotional aspects of Natural Language Processing(NLP) or Text Mining attract the attention of many research areas, not only in computer science, but also in psychology, healthcare, communication etc.

Generally, two approaches (knowledge-based approaches and machine learning approaches) were adopted for automatic analysis of emotions in text, aiming to detect the writer's emotional state. The first approach consists of using linguistic models or prior knowledge to classify emotional text. The second one uses supervised learning algorithms to build models from annotated corpora. For sentiment analysis, machine learning techniques tend to obtain better results than lexical-based techniques, because they can adapt well to different domains. [1]

Background

In a variation on the popular task of sentiment analysis, this dataset contains labels for the emotional content (such as happiness, sadness, and anger) of texts. Hundreds to thousands of examples across 13 labels. A subset of this data is used in an experiment uploaded to Microsoft's Cortana Intelligence Gallery.

Added on Data.world : July 15, 2016 by CrowdFlower

Data points/ Rows: 40000

Label	No of Items	Label	No of Items
Anger	110	**Neutral**	8638
Boredom	63	**Relief**	1526
Enthusiasm	759	**Sadness**	5165
Fun	1776	**Surprise**	2187
Happiness	5209	**Worry**	8459
Hate	1323	**Empty**	827
Love	3842		

Table 1: Classes and their distribution of the dataset

Methodology

1. Cleaning the dataset

1.1. Dropping some less important Features.

Tweet-id and author of the tweets has no relation with that help to decide the emotional content of the tweet. So I decided to drop those features.

```
# Read in the data
path="C:/Users/dawit/Google Drive/Data Mining/project"
filename_read = os.path.join(path,"text_emotion.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
df
```

	tweet_id	sentiment	author	content
0	1956967341	empty	xoshayzers	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	wannamama	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	coolfunky	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	czareaquino	wants to hang out with friends SOON!
4	1956968416	neutral	xkilljoyx	@dannycastillo We want to trade with someone w...
5	1956968477	worry	xxxPEACHESxxx	Re-pinging @ghostridah14: why didn't you go to...
6	1956968487	sadness	ShansBee	I should be sleep, but im not! thinking about ...
7	1956968636	worry	mcsleazy	Hmmm. http://www.djhero.com/ is down

Figure 1:the head the dataset

```
df=df.drop('tweet_id', axis= 1)
df=df.drop('author', axis= 1)
df.head()
```

Figure 2:feature dropout implementation.

1.2.NLTK stop words: Text may contain stop words like ‘the’, ‘is’, ‘are’. Stop words can be filtered from the text to be processed. There is no universal list of stop words in nlp research, however the nltk module contains a list of stop words. [2]

```
all_words=[]
filter_words = []
stop_words = set(stopwords.words('english'))
for index, row in df.iterrows():
    word_tokens = word_tokenize(row['content'])
    filtered_sentence = [w.lower() for w in word_tokens if not w in stop_words]
    all_words.extend(filtered_sentence)
all_words = nltk.FreqDist(all_words)

word_features = list(all_words.keys())[:1000]
```

Figure 3: nltk stop words implementation

2. **Scikit-Learn :** Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. [3]

2.1. Model selection

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. Note that the word "experiment" is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally.

train_test split

```
X_train, X_test, y_train, y_test = train_test_split(df['content'],
                                                    df['sentiment'],test_size = 0.33,
                                                    random_state=0)
```

Figure 4: Sci-kit learn train_test split

2.2. Encoding the text to use in a machine learning models

2.2.1. CountVectorizer (Bag of Words approach)

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

I use it as follows:

- Create an instance of the CountVectorizer class.
- Call the fit() function in order to learn a vocabulary from one or more documents.
- Call the transform() function on one or more documents as needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the SciPy. Sparse package.

The vectors returned from a call to transform () will be sparse vectors, and you can transform them back to numpy arrays to look and better understand what is going on by calling the toarray() function. [4]

from sklearn.feature_extraction.text import CountVectorizer

```
# Fit the CountVectorizer to the training data
vect = CountVectorizer().fit(X_train)
vect.get_feature_names()[::2000]
len(vect.get_feature_names())
```

```
36449
```

Figure 5: Sci-kit learn CountVectorizer implementation

2.2.2. TFIDF Vectorizer

An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. This is an acronym than stands for "Term Frequency – Inverse Document" Frequency which are the components of the resulting scores assigned to each word.

- Term Frequency: This summarizes how often a given word appears within a document.
- Inverse Document Frequency: This downscales words that appear a lot across documents.

Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternately, if you already have a learned CountVectorizer, you can use it with a Tfidf Transformer to just calculate the inverse document frequencies and start encoding documents. [5]

```
# Fit the TfidfVectorizer to the training data specifiying a minimu
vect = TfidfVectorizer(min_df=5).fit(X_train)
len(vect.get_feature_names())
X_train_vectorized = vect.transform(X_train)
```

Figure 6:Sci=kit learn TFiDF implementation

2.2.3. ngramVectorizer (sequence of words features)

I also use the modified version of CountVectorizer. Which count not only a single work but a group of them. This helps to get some contextual understanding to.

I test number of values for the following parameters

- **Min_df =5** minimum support of the frequence a word in a document/class to be counted.

- **Ngram_range =(1,2)** how many word group range(1,2) means 2 group of words or bigram.

```
vect = CountVectorizer(min_df=5, ngram_range=(1,2)).fit(X_train)

X_train_vectorized = vect.transform(X_train)

len(vect.get_feature_names())
```

Figure 7: Sci-kit Learn N-Gram implementation

2.3. Classification Algorithms

2.3.1. Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable.

from sklearn.linear_model import LogisticRegression

```
# Train the model
model = LogisticRegression()
model.fit(X_train_vectorized, y_train)
# Predict the transformed test documents
predictions = model.predict(vect.transform(X_test))
print('Accurarcy: ',accuracy_score(y_test, predictions))
```

```
Accurarcy:  0.342424242424
```

Figure 8: Sci-kit learn Logistic regression implementation

```
Confusion matrix
[[   0    0    1    0    0    7    1    1   10    1    3    1   12]
 [   0    0    0    0    2    3    3    1   17    0   11    0   23]
 [   0    1    1    1    1   14    2    4  145    2   18    3   64]
 [   0    1    1    1    5   45    3   13   93    1   17    5   54]
 [   0    0    0    4   29  128    4   36  202    7   41   13  103]
 [   0    0    2    3   43  586    3  168  520   26   59   44  262]
 [   0    2    0    0    5   18   66    4   97    0   80    8  125]
 [   0    0    0    0   17  259    8  497  269   17   63   16  153]
 [   1    1    6    5   32  227   22  105 1630   20  198   42  582]
 [   0    0    1    0   12   85    2   29  177   24   27    7  121]
 [   0    0    4    1   11   79   48   52  433    9  457   15  632]
 [   0    0    0    1    8  101    9   41  270    6   54   32  198]
 [   0    0    3    1   19  153   51   71  828   24  413   44 1197]]
```

Figure 9: confusion matrix for the logistic regression classifier

2.3.2. NaiveBayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.

from sklearn import naive_bayes

Using CountVector

```
clfrNB = naive_bayes.MultinomialNB()
clfrNB.fit(X_train_vectorized,y_train)
X_test_vectorized = vect.transform(X_test)
predicted_labels = clfrNB.predict(X_test_vectorized)
from sklearn import metrics
metrics.f1_score(y_test,predicted_labels,average='micro')
```

```
0.30363636363636365
```

Figure 10: Sci-kit Naive Bayes using Countervectorizer encoding

Using TFIDF

```
clfrNB = naive_bayes.MultinomialNB()
clfrNB.fit(X_train_vectorized,y_train)
X_test_vectorized = vect.transform(X_test)
predicted_labels = clfrSVM.predict(X_test_vectorized)
metrics.f1_score(y_test,predicted_labels,average='micro')
```

```
0.3175
```

Figure 11: Sci-kit learn Naive Bayes using TFIDF encoding

2.3.3. Support Vector Machine

Support Vector Machines belong to the discriminant model family: they try to find a combination of samples to build a plane maximizing the margin between the two classes. The C parameter sets regularization: a small value for C means the margin is calculated using many or all of the observations around the separating line (more regularization); a large value for C means the margin is calculated on observations close to the separating line (less regularization).

from sklearn import svm

Using CountVectorizer

```
from sklearn import svm
clfrSVM = svm.SVC(kernel='linear', C=0.1)
clfrSVM.fit(X_train_vectorized, y_train)
predicted_labels = clfrSVM.predict(X_test_vectorized)
metrics.f1_score(y_test,predicted_labels,average='micro')
```

```
0.35325757575757577
```

Figure 12: Sci-kit learn SVM using CountVectorizer encoding

```
clfrSVM = svm.SVC(kernel='linear', C=0.1)
clfrSVM.fit(X_train_vectorized, y_train)
predicted_labels = clfrSVM.predict(X_test_vectorized)
metrics.f1_score(y_test,predicted_labels,average='micro')
```

```
0.3175
```

Figure 13: Sci-kit learn SVM using TFIDF encoding

2.4.NLTK's NaiveBayesClassifer

I also try to implement NLTK's Naïve Bayes classifier to see if the perform better but as I trying to prepare the dataset to pass to the classifier , I keep on running 'memory error '.

```
from nltk.classify import NaiveBayesClassifier

Classifier = NaiveBayesClassifier.train(training_set)

classifier.classify_many(X_test)
nltk.classify.util.accuracy(classifier,testing_set)
```

Figure 14:NLTK's Naive Bayes Classifiers

Conclusion

As the content of the dataset was tweets of people, it was so disorganized and messy with a lot of abbreviate and slang words. The content to a class proportion was small. Because this and working on unorganized text data is hard the accuracy I got from the classifiers was bad.

To get a better classification on this or any unorganized text data, its better to use multi-layer neural net with GPU processor.

References

[1] "Using a Heterogeneous Dataset for Emotion Analysis in Texts".

[2] "https://pythonspot.com/en/nltk-stop-words/".

[3] "SCIKIT-LEARN: MACHINE LEARNING IN PYTHON".

[4] "https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/".

[5] "https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/".