Text Mood Analysis

1) Dataset stored in 'dataset.csv' file is read. Dataset contains a list of text and its appropriate mood value. LabelEncoder is used to transform non-numerical labels(mood values) to numerical labels.

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing

dataset = pd.read_csv('dataset.csv')
X = dataset['text'].values.tolist()
encoder = preprocessing.LabelEncoder()
y = encoder.fit_transform(dataset['mood'].values.tolist())
encoding = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))
print(encoding)
```

```
{'sadness': 2, 'happiness': 1, 'anger': 0}
```

2) Dataset is splitted into training and testing sets having ratio 3:1 respectively.

In [2]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.
25, shuffle=True)
print('Training data size = %d' % len(X_train))
print('Testing data size = %d' % len(X_test))
```

```
Training data size = 6802
Testing data size = 2268
```

3) Data cleaning and pre-processing function converts text into lowercase lemmatized text containing no stop words

In [3]:

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

stop = stopwords.words('english')
lemmatizer = WordNetLemmatizer()
def pre_process(text):
    words = word_tokenize(text)
    words = [x for x in words if x.isalpha()]
    words = [x.lower() for x in words]
    words = [lemmatizer.lemmatize(x) for x in words]
    words = [x for x in words if x not in stop]
    return " ".join(words)
```

4) CountVectorizer converts a collection of text documents to a matrix of token counts. CountVectorizer extracts features from text.

In [4]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer='word', max_features=5000, preprocessor=pre_proce
ss, ngram_range=(1,3))
vectorizer.fit(X)
X_train = vectorizer.transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()
```

In [5]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
print('accuracy ', accuracy_score(y_pred, y_test))
```

```
accuracy  0.7707231040564374
```

In [6]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print('accuracy ', accuracy_score(y_pred, y_test))
```

```
accuracy  0.6441798941798942
```

In [7]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression(C=1, solver='lbfgs', multi_class='multinomial', max_iter=10
00)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('accuracy ', accuracy_score(y_pred, y_test))
```

```
accuracy  0.7839506172839507
```

In [8]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

lsvm = SGDClassifier(alpha=0.001, random_state=5, max_iter=15, tol=None)
lsvm.fit(X_train, y_train)
y_pred = lsvm.predict(X_test)
print('accuracy ', accuracy_score(y_pred, y_test))
```

```
accuracy  0.7857142857142857
```