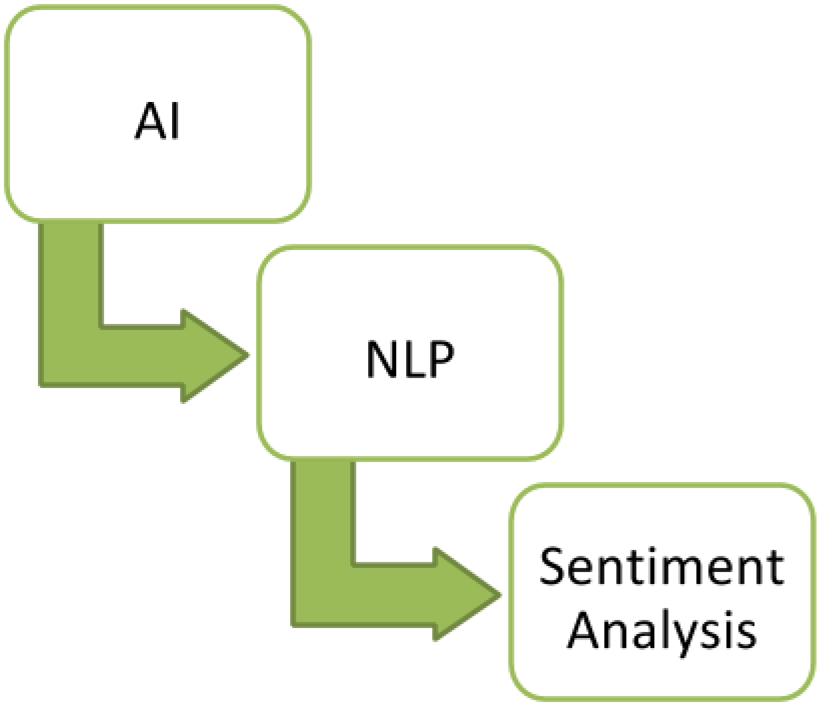
Bigdata Diploma Final Project

Detecting Emotions in Arabic Tweets

Mina Adel Sonbol
Hesham Shabana
Sherif Kamal Farahat

Introduction



The rise of social networking including microblogging and microposts in websites like Twitter, LinkedIn and Facebook, has led to the rise of sentiment analysis, which in turn aim to determine the attitude of the writer. Sentiment Analysis is not just about detecting the polarity of some given text (positive, negative or neutral), but also about understanding the emotion being conveyed by the text (sadness, anger, joy, disgust, etc.).

Sentiment analysis has been used in various applications, such as recommender systems (Amazon, Netflix), search engines (Google, Bing), conversational commerce and chatbots (Apple's Siri, Google Now, Amazon Echo). The success experienced by many of these applications has led to increased funding (229% increase in investments into chatbots between 2015 & 2016).

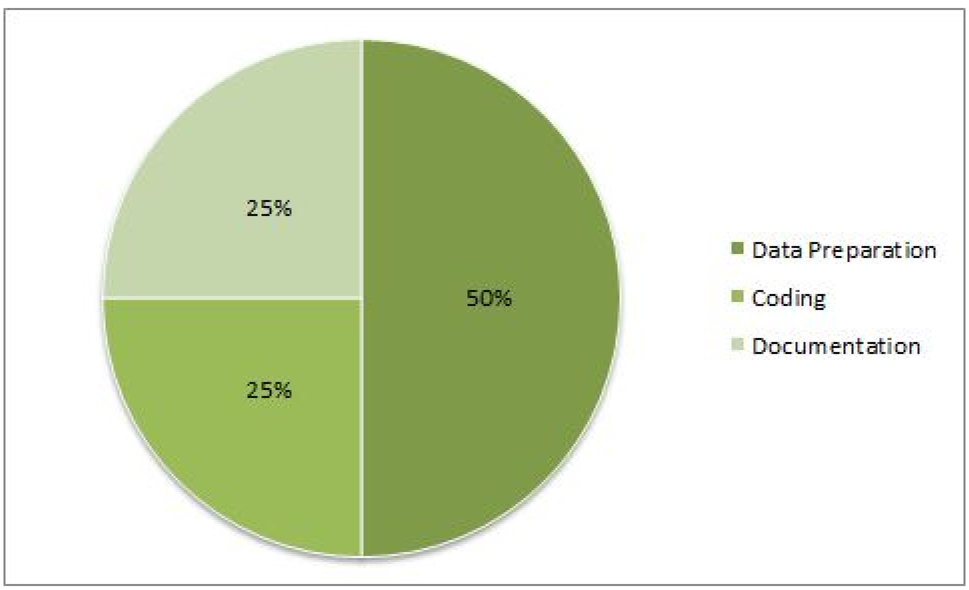
In this project, we manually annotated an Arabic corpus collected from Twitter, assigning each tweet with one of the following seven emotions: Anger, Happiness, Sadness, Fear, Surprise, Disgust, and Relaxed. We then used the annotated corpus to train a classifier that can automatically discover the emotions in a given tweet. Finally, we calculate the efficiency of the classifier using Accuracy and F-Score.

Dataset

The dataset consists of random Arabic tweets, used in the mini project of CIT 653. We annotated 1000 tweets, applying a rigorous 2-round annotation process. In round 1, two persons assigned one of the seven emotions to each tweet. Round 2 saw a third person adding a vote to either choice in tweets that had been assigned different emotions in the first round.

id	tweet	class	Mina		Sherif		Hisham
271	فينو الاهبل ابن الاهبل '	neg	anger		anger	1	anger
131	على المصرييييين وجمالهم ربنا يحميهم #MinaAtta http://t.co/NkOvSx6mgD '	pos	happiness		happiness	1	happiness
118	@Kholoudkewan دول كتير اوى ودمهم خفيف العمارة اللي انا فيها كلها سوريين والأطفال عسل '	pos	happiness		happiness	1	happiness
6	انا بعد كده خلى اللى يوعنى بحاجه همضى على وصل امانه علشان اضمن انو مش يخون '	neg	anger		surprise	FALSE	surprise
3430	انا هنتحر '	neg	sadness		sadness	1	sadness

Workflow



Within the project's lifecycle, 50% of the time was consumed in data preparation and cleaning, while 25% of the time was consumed for both coding and documentation.

1. Data Exploration: Visulizing most common words, number of instances in each emotion class, to get a sense of the dataset.

2. Data Preprocessing: Cleaning the data by applying the following: Removing stopwords, stemming, removing URLs, removing mentions/hashtags and removing punctuation marks.

3. Model selections: We used 3 models, bag of words, N-Grams and TF-IDF, with the bag of words acting as our base value.

4. Training: Train our model using the aforementions models.

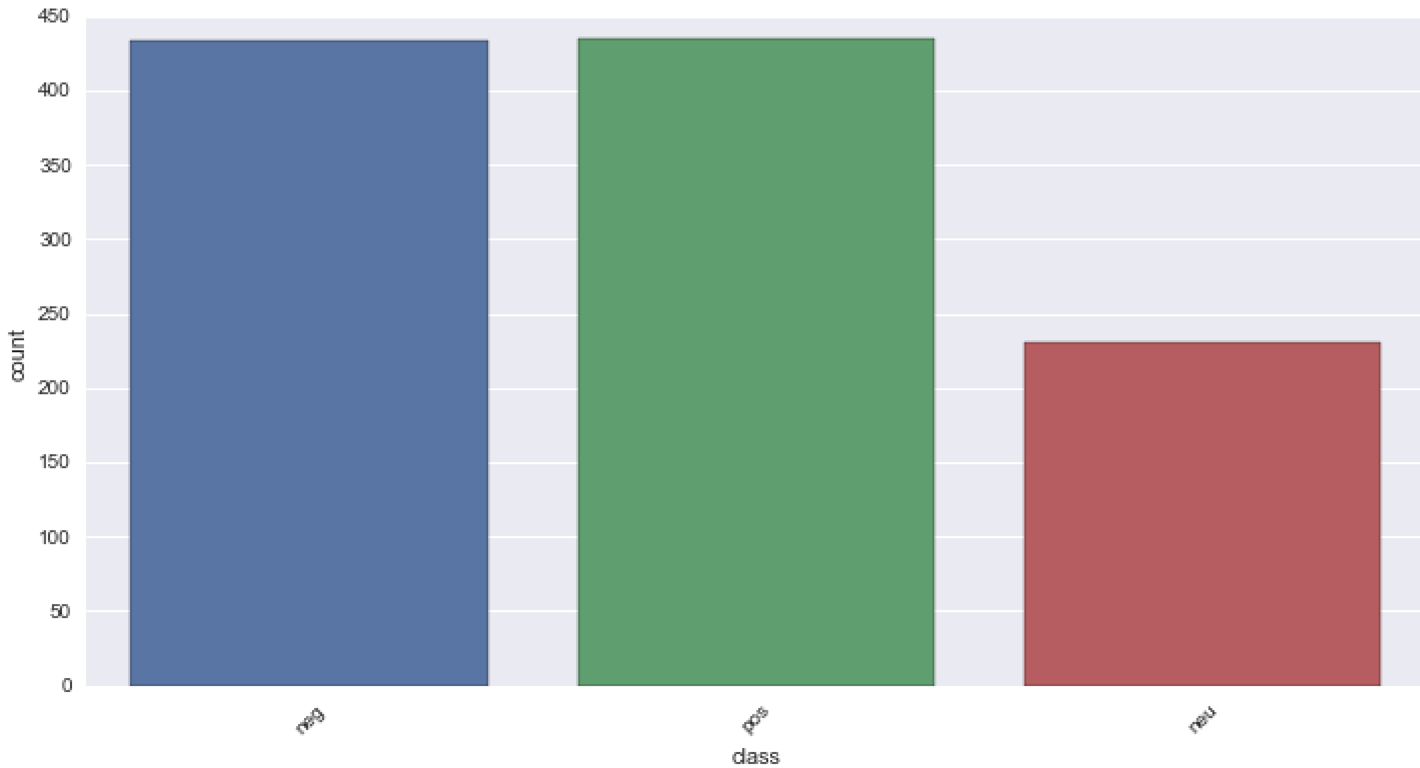
5. Evaluation: Evaluate the results of each model, based on its F-Score and Accuracy.

6. Conclusion: Comparing the results and deciding which model had the best scores.

Data Visualization

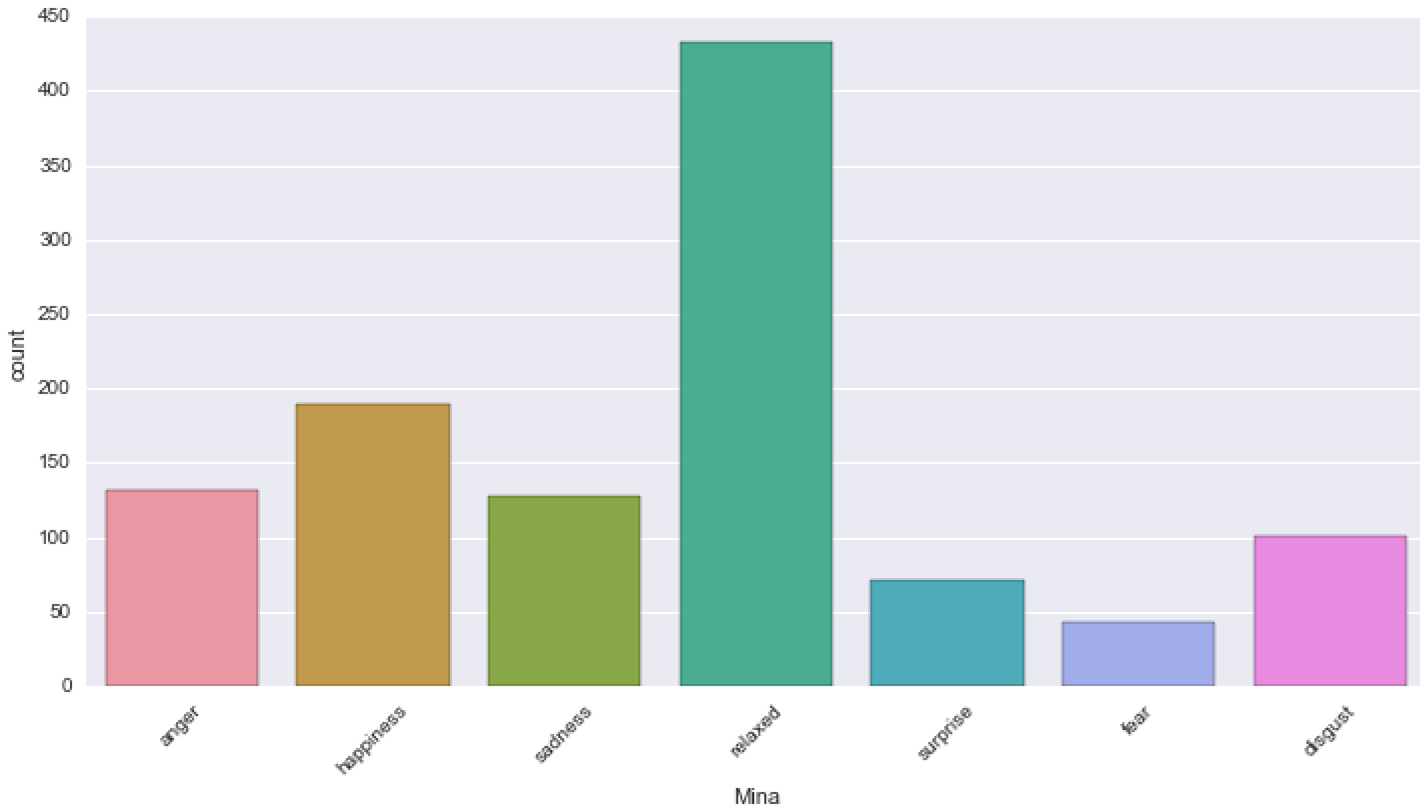
- Original Data

The original dataset classified tweets into three classes positive, negative and neutral. As shown in the figure below.



- Annotated Data

After annotating the dataset into more classes happiness, relaxed, surprise, fear, anger, disgust, sadness.

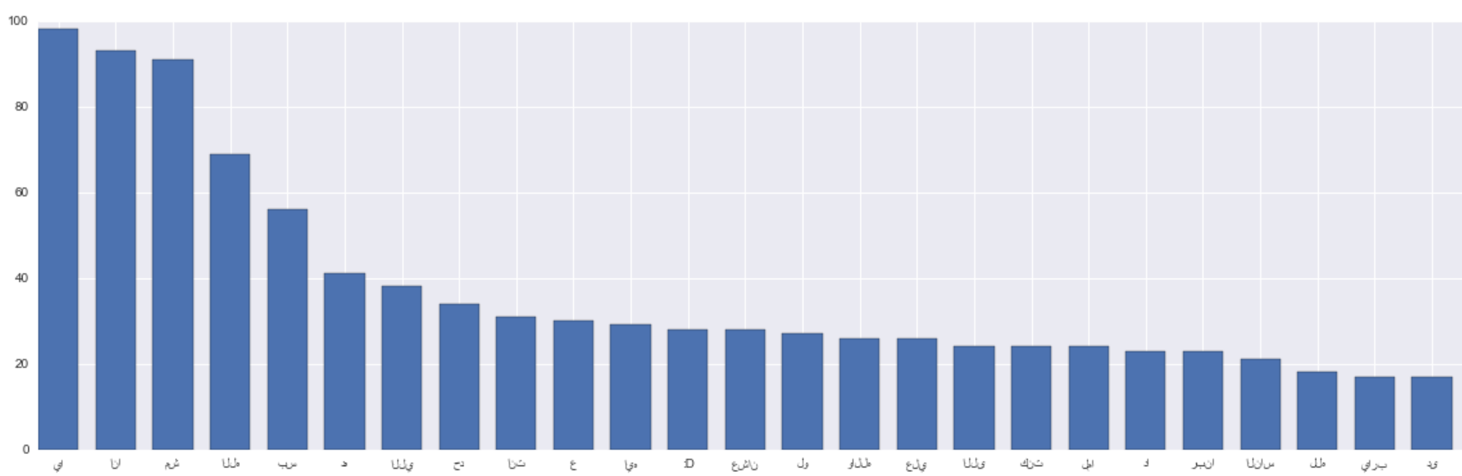


We can see that most tweets falls under the relaxed category.

- Top terms frequency

After examining the dataset the below are the most frequent words are:

عشان لو والله علي اللى كنت لما دا ربنا الناس :D يا انا مش الله بس ده اللي حد انت ايه
لله يارب دى



Data Preprocessing

To be able to train the classifiers, we had to clean the data. To do so, we used a pandas dataframe (to accommodate the original csv file), and we performed the following actions:

1- Remove Stopwords: As illustrated in the previous segment, many of the most common words are stopwords. These words lack significance, so we decided to remove them. Ex.:

حالة من الاكتئاب و البؤس .. '	حالة الاكتئاب البؤس .. '

2- Stemming: All words are returned to their base, to be able to get the true sentiment of a given word regardless of use or tense. Ex.:

حالة من الاكتئاب و البؤس .. '	حلة كئب بؤس .. '

3- Remove URLs: URLs give no added meaning to the tweet, and therefore do not contribute to the tweet's sentiment. Ex.:

في حفظ الله يا ريس http://t.co/RGKtP9QHZB '	حفظ الل يا ريس '

4- Remove Mentions/Hashtags: Mentions and hashtags can be used to give context to a tweet. However, we prefered to remove them from our consideration. Ex.:

@Kholoudkewan دول كتير اوى ودمهم خفيف العمار...	دول كتر اوى ودم خفف عمر فيه كله سور طفل عسل '

5- Remove Punctuation: For simplicity, we also decided to not deal with punctuations here. Ex.:

' http://t.co/RGKtP9QHZB في حفظ الله يا ريس	حفظ الل يا ريس

Error Metric

1. Accuracy: The percentage of correct predictions.

2. F-Score: Is a measure that combines precision and recall. Precision is: the number of true positives divided by all positive results. Recall is: The number of true positives divided by the total real positives. The F-Score equation is:

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

Classifiers

1. RandomForest: fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting
2. Gaussian Naive Bayes: implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian
3. Decision Tree: is a non-parametric supervised learning method used for classification. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

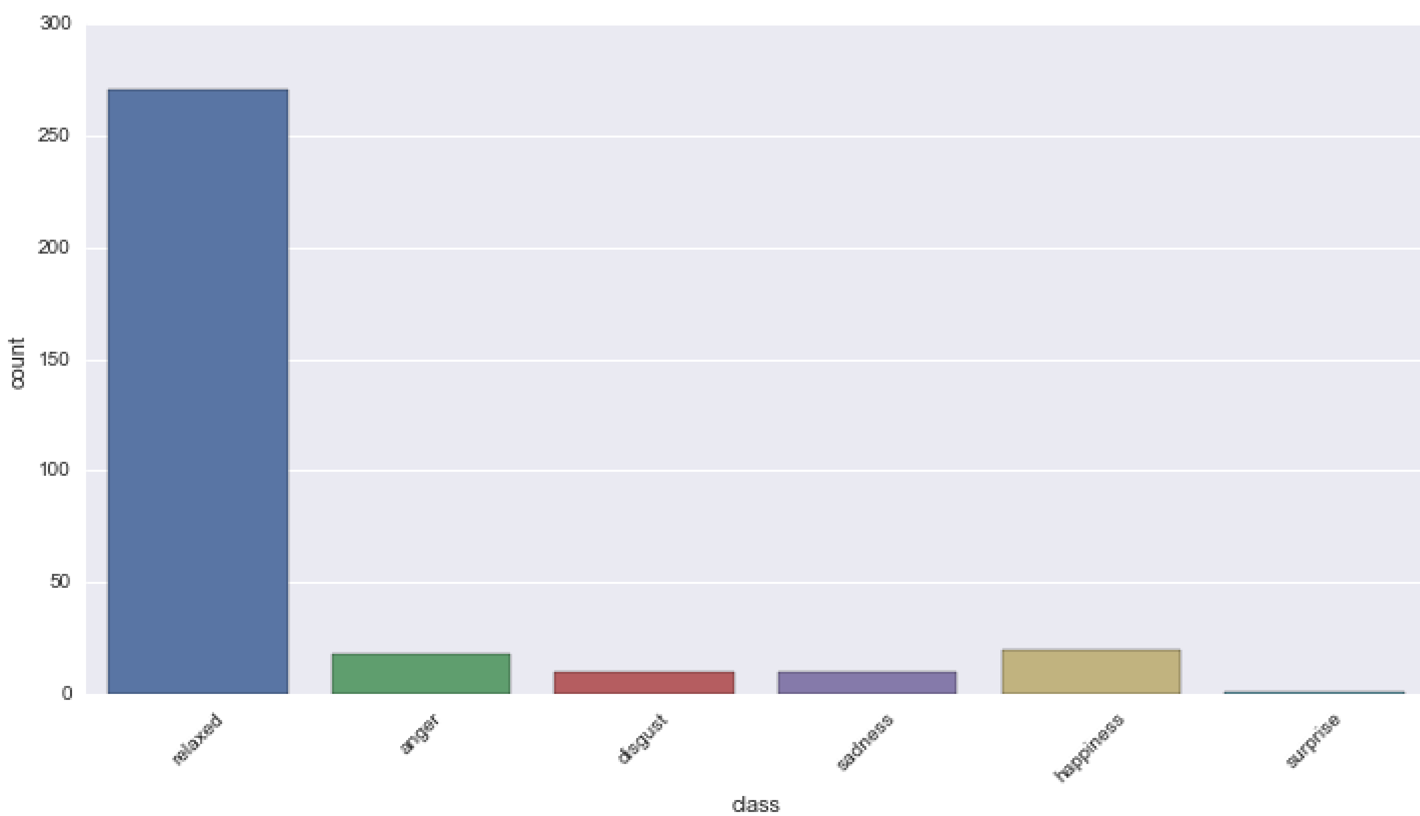
Reference: http://scikit-learn.org/

Models

1. **Bag of words**

Bag-Of-Words is used to simplify the representation for information retrieval, natural language processing also it's commonly used in document classification where the frequency of each word is used as a feature for training a classifier.

After applying Bag_Of_Words to transform the tweets the new data set has 4212 parameter with only 1099 observation, which made our model very likely to overfit, therefore, we used the most important 1000 parameter to train our RandomForest classifier the below graph show the predicted class distribution.

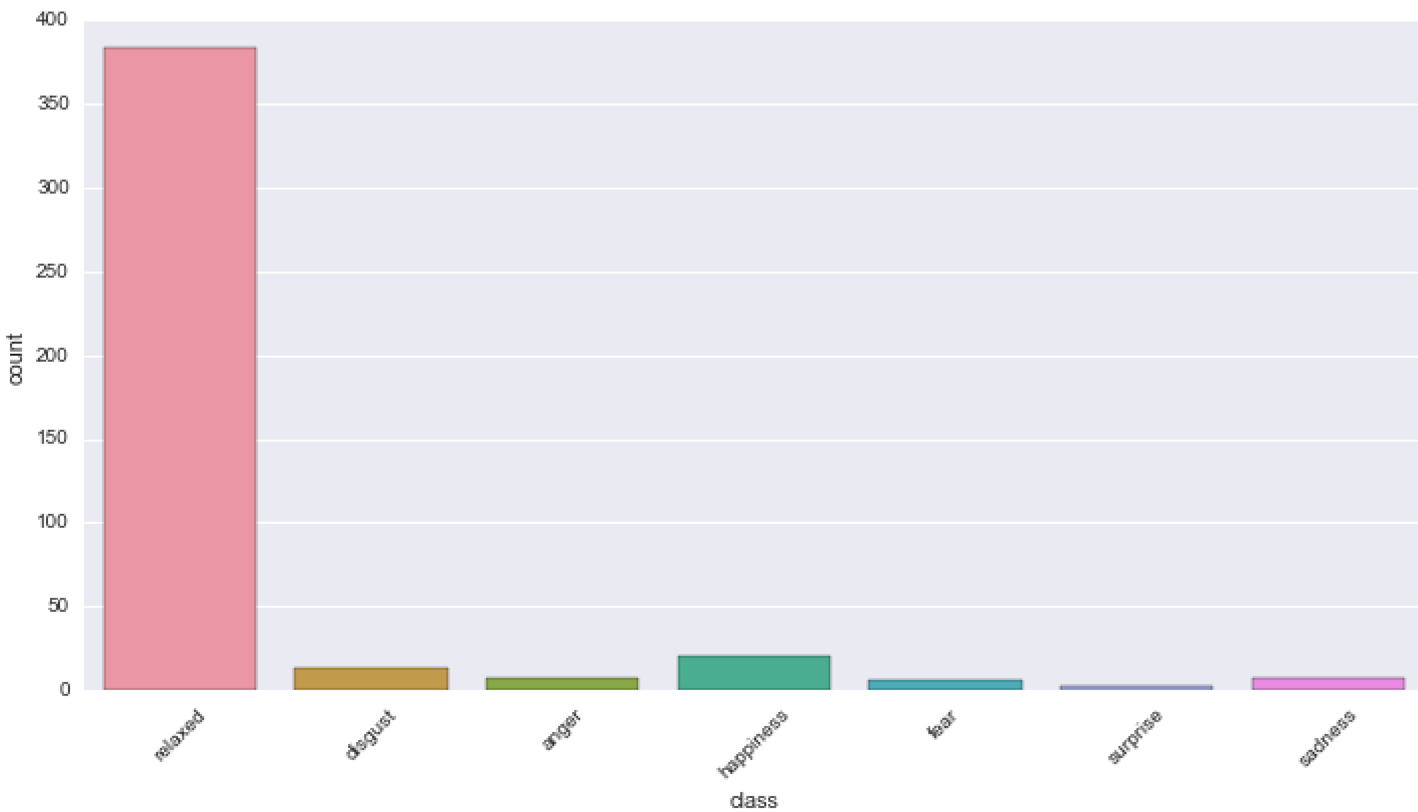


Classifier comparison:

Classifier	Accuracy Train Score	Accuracy Test Score	Number of mislabeled points	F-Score
Random Forests	0.95	0.34	32	0.36
Gaussian Naive Bayes	0.66	0.30	261	0.35
Decision Tree	0.94	0.31	32	0.31

2. N-Gram

The Bag_Of_Words model is considered as unigram where only one term frequency is considered which could ignore some important information for example a word negation could be considered as positive while it should be considered as negative. Therefore we used the N-Gram model with a size of n “digram”, below are the result after train our classifier.

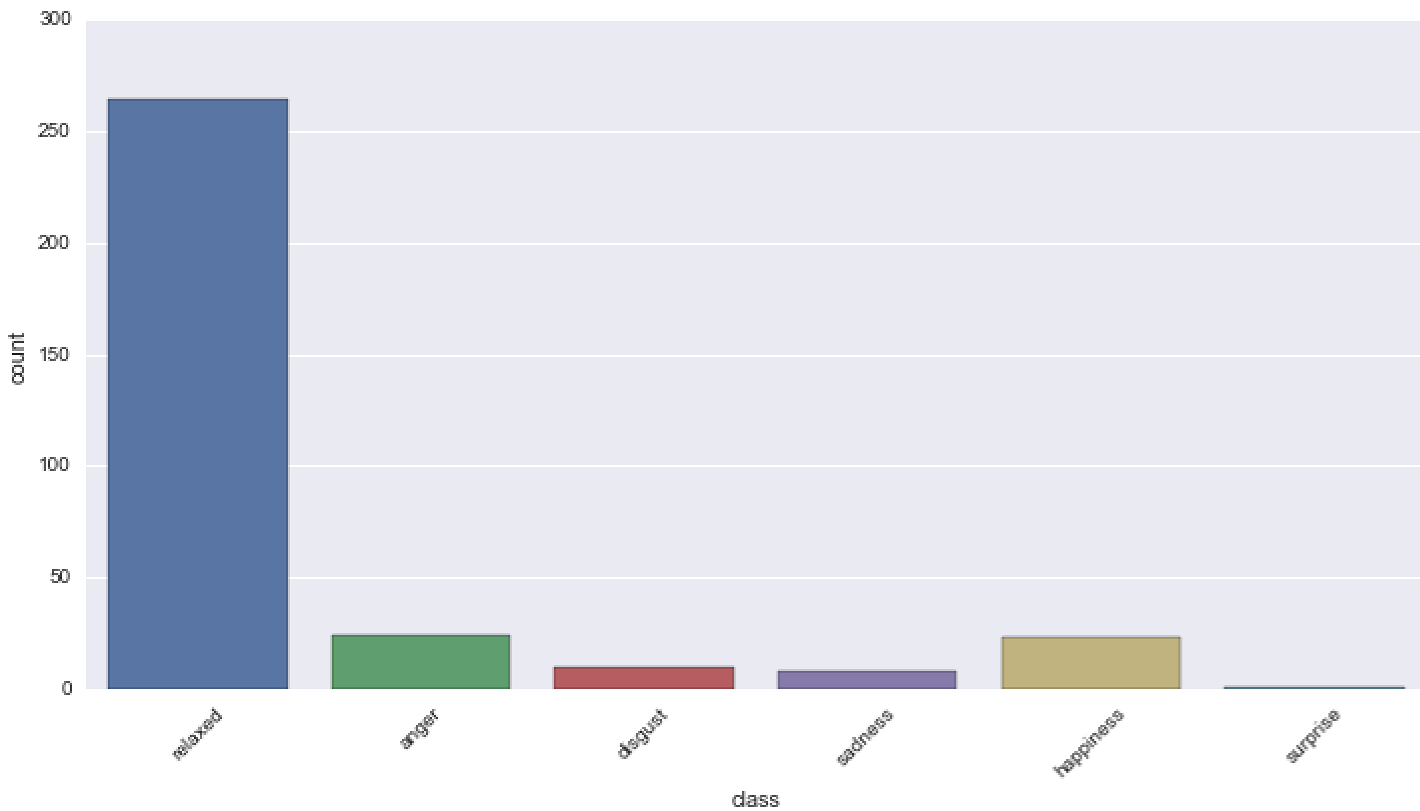


Classifier comparison:

Classifier	Accuracy Train Score	Accuracy Test Score	Number of mislabeled points	F-Score
Random Forests	0.76	0.36	155	0.26
Gaussian Naive Bayes	0.59	0.1	268	0.11
Decision Tree	0.76	0.35	155	0.25

3. TF-IDF

TF–IDF is a short for term frequency inverse document frequency and it is intended to reflect how important a word is in a specific tweet compared with all set of words/corps. The tf-idf value increases proportionally to the number of times a word appears in the document, and it's offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general like stopwords.



Classifier comparison:

Classifier	**Accuracy Train Score**	**Accuracy Test Score**	**Number of mislabeled points**	**F-Score**
Random Forests	0.98	0.41	8	0.32
Gaussian Naive Bayes	NA	NA	NA	NA
Decision Tree	0.98	0.39	8	0.32

Conclusion

The low number of observation and the high number of parameters introduced a very high train score which could be a sign for overfitting, which is caused by using data transformation like bag of words.

Applying sentiment analysis on Arabic text is challenging and it's not supported by default in most libraries, in addition that most of the tweets language is considered as a slang language and in many cases the classifier misinterpret the overall meaning of the sentence.

Preparing, annotating and cleaning the data took a lot of time, however, it is a very important step to arrange the data in a way that it easy for the classifier to work with.

Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import re
import string
import codecs
from nltk.tokenize import word_tokenize, wordpunct_tokenize, sent_tokenize
import csv
import nltk
from nltk import ISRIStemmer
from nltk.corpus import stopwords

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
/Users/gr8h/anaconda/envs/nup/lib/python2.7/site-packages/IPython/html.py:14: ShimWarning: The
`IPython.html` package has been deprecated since IPython 4.0. You should import from
`notebook` instead. `IPython.html.widgets` has moved to `ipywidgets`.
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

In [526]:

```
def unicode_csv_reader(utf8_data, dialect=csv.excel):
    #csv_reader = csv.reader(utf8_data, dialect=dialect)
    for row in utf8_data:
        row = [unicode(cell, 'utf-8') for cell in row]
    return utf8_data
```

In [527]:

```
def label_transform(label):
    if label == 'happiness':
        return 7
    elif label == 'relaxed':
        return 6
    elif label == 'surprise':
        return 5
    elif label == 'fear':
        return 4
    elif label == 'anger':
        return 3
    elif label == 'disgust':
        return 2
    elif label == 'sadness':
        return 1
```

In [540]:

```
pattern = re.compile('[\u0627-\u064a]')
re_arabic_p = re.compile(pattern)
def remove_stopwords(sentence):
    return [word for word in sentence.split() if word not in set(stopwords.words('arabic'))]
```

In [552]:

```
def stem_text(sentence):
    stemmer = ISRIStemmer()
    return [stemmer.stem(word) for word in sentence]
```

In [542]:

```
def clean_sentence(sentence):
    sentence = sentence.translate(None, string.punctuation)
    letters_only = re.sub(re_arabic_p, "", sentence)
    return letters_only
```

In [543]:

```
def printArabic(stringToPrint):
    print unicode(stringToPrint, 'utf-8')
```

In [544]:

```
dataset = pd.read_excel('dataset.xlsx', 'NU_EG_Twitter_corpus_train.csv')
del dataset['Unnamed: 4']
del dataset['Unnamed: 6']
del dataset['Mostafa']
del dataset['Sherif']
```

In [545]:

```
negators = pd.read_csv('negators.txt', header=None)
```

In [546]:

```
dataset = dataset[pd.notnull(dataset['Mina'])]
```

In [547]:

```
dataset['sentiment'] = dataset['Mina'].apply(lambda x: label_transform(x))
```

In [548]:

```
dataset.head()
```

Out[548]:

In [484]:

```
#dataset['tweet_clean'] = dataset['tweet'].apply(lambda x:
clean_sentence(x.encode('utf-8').strip()))
#dataset.head()
```

In [549]:

```
dataset['tweet_st'] = dataset['tweet'].apply(lambda x: remove_stopwords(x))
dataset.head()
```

Out[549]:

In [553]:

```
dataset['tweet_sm'] = dataset['tweet_st'].apply(lambda x: stem_text(x))
dataset.head()
```

Out[553]:

In [554]:

```
from nltk.corpus import brown
from nltk.metrics import edit_distance
```

In [555]:

```
def build_lexicon():
    reader = pd.read_excel('NileULex.xlsx', 'Sheet1')

    weightedLexicon = []

    for row in reader:
        word = row[0]

        if row[1] == "neg":
            score = -1
        elif row[1] == "compound_neg":
            score = -2
        elif row[1] == "pos":
            score = 1
        elif row[1] == "compound_pos":
            score = 2
        else:
            score = 0
        weightedLexicon.append((word, score))

    return weightedLexicon
```

In [556]:

```
def lookUpWordScore(word, lexicon):
    stemmer = ISRIStemmer()

    for key, score in lexicon:
        if key == word:
            return score

    for key, score in lexicon:
        if stemmer.stem(key) == stemmer.stem(word):
            print word
            print key
            return score*0.25

    for key, score in lexicon:
        med = edit_distance(word, key)
        match = 1 - (float(med)/len(word))
        if match > 0.7:
            return score*0.25
    return 0
```

In [557]:

```
def tweetScore(sentence, lexicon):

    tweetScore = 0
    words = wordpunct_tokenize(sentence)

    for index, word in enumerate(words):
        term = word
        if len(words[index:]) < 6:
            maxim = index + len(words[index:])-1
        else:
            maxim = index + 5
```

```
        for i in range(index+1, maxim + 1):
            term = term + " " + words[i]
            tweetScore = tweetScore + lookUpWordScore(term, lexicon)

    return tweetScore
```

In [558]:

```
lexicon = build_lexicon()
```

In [559]:

```
dataset['tweet_clean'] = dataset['tweet_sm'].apply(lambda x: " ".join(x))
dataset.head()
```

Out[559]:

In [561]:

```
edit_distance('هل', 'هبل')
```

Out[561]:

```
2
```

In [562]:

```
dataset.tail()
```

Out[562]:

In [563]:

```
dataset = dataset.dropna()
```

In [564]:

```
nltk_tokenizer = nltk.tokenize.TreebankWordTokenizer()
```

Bag of words

In [565]:

```
from sklearn import preprocessing
from sklearn.cross_validation import train_test_split
```

In [566]:

```
from sklearn.feature_extraction.text import CountVectorizer

arabic_sw = stopwords.words("arabic")
# Initialize the "CountVectorizer" object, which is scikit-learn's
# bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = arabic_sw,   \
                             max_features = 5000)

# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of
# strings.
data_features = vectorizer.fit_transform(dataset['tweet_clean'])

# Numpy arrays are easy to work with, so convert the result to an
# array
data_features = data_features.toarray()
```

In [568]:

```
y = dataset['sentiment']
```

In [569]:

```
X_train, X_test, y_train, y_test = train_test_split(data_features, y, test_size=0.3,
random_state=0)
```

In [570]:

```
vocab = vectorizer.get_feature_names()
```

Error metric

In [571]:

```
#https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Train

In [572]:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators = 500)

# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
#
# This may take a few minutes to run
forest = forest.fit( X_train, y_train )
```

In [573]:

```
y_pred = forest.predict(X_train)
print 'Train accuracy_score:', accuracy_score(y_train, y_pred)
matrix = confusion_matrix(y_train, y_pred)
print 'Train confusion_matrix:', matrix
report = classification_report(y_train, y_pred)
print report

y_pred = forest.predict(X_test)
print 'Test accuracy_score:', accuracy_score(y_test, y_pred)
matrix = confusion_matrix(y_test, y_pred)
print 'Test confusion_matrix:', matrix
report = classification_report(y_test, y_pred)
print report
```

```
Train accuracy_score: 1.0
Train confusion_matrix: [[ 92   0   0   0   0   0   0]
 [  0  72   0   0   0   0   0]
 [  0   0  94   0   0   0   0]
 [  0   0   0  34   0   0   0]
 [  0   0   0   0  48   0   0]
 [  0   0   0   0   0 305   0]
 [  0   0   0   0   0   0 124]]
             precision    recall  f1-score   support

          1       1.00      1.00      1.00        92
          2       1.00      1.00      1.00        72
          3       1.00      1.00      1.00        94
```

```
          4       1.00      1.00      1.00        34
          5       1.00      1.00      1.00        48
          6       1.00      1.00      1.00       305
          7       1.00      1.00      1.00       124

avg / total       1.00      1.00      1.00       769

Test accuracy_score: 0.409090909091
Test confusion_matrix: [[  1   0   0   1   0  32   2]
 [  0   2   1   2   0  22   2]
 [  0   5   4   0   0  27   2]
 [  0   0   0   0   0   9   0]
 [  1   0   3   1   0  17   2]
 [  1   1   0   0   0 120   6]
 [  0   0   1   0   0  57   8]]
             precision    recall  f1-score   support

          1       0.33      0.03      0.05        36
          2       0.25      0.07      0.11        29
          3       0.44      0.11      0.17        38
          4       0.00      0.00      0.00         9
          5       0.00      0.00      0.00        24
          6       0.42      0.94      0.58       128
          7       0.36      0.12      0.18        66

avg / total       0.35      0.41      0.30       330
```

TF-IDF

In [574]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=5,
                             max_df = 0.8,
                             sublinear_tf=True,
                             use_idf=True)
tfidf_data_features = vectorizer.fit_transform(dataset['tweet_clean'])
```

In [575]:

```
X_train, X_test, y_train, y_test = train_test_split(tfidf_data_features, y, test_size=0.3,
random_state=0)
```

In [576]:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators = 500)

# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
#
# This may take a few minutes to run
forest = forest.fit( X_train, y_train )
```

In [577]:

```
y_pred = forest.predict(X_train)
```

```
print 'Train accuracy_score:', accuracy_score(y_train, y_pred)
matrix = confusion_matrix(y_train, y_pred)
print 'Train confusion_matrix:', matrix
report = classification_report(y_train, y_pred)
print report

y_pred = forest.predict(X_test)
print 'Test accuracy_score:', accuracy_score(y_test, y_pred)
matrix = confusion_matrix(y_test, y_pred)
print 'Test confusion_matrix:', matrix
report = classification_report(y_test, y_pred)
print report
```

```
Train accuracy_score: 0.955786736021
Train confusion_matrix: [[ 87   0   0   0   0   5   0]
 [  0  69   0   0   0   3   0]
 [  0   0  90   0   1   3   0]
 [  1   0   0  31   0   2   0]
 [  1   1   0   0  43   3   0]
 [  0   0   0   0   0 303   2]
 [  0   0   0   0   0  12 112]]
             precision    recall  f1-score   support

          1       0.98      0.95      0.96        92
          2       0.99      0.96      0.97        72
          3       1.00      0.96      0.98        94
          4       1.00      0.91      0.95        34
          5       0.98      0.90      0.93        48
          6       0.92      0.99      0.95       305
          7       0.98      0.90      0.94       124

avg / total       0.96      0.96      0.96       769

Test accuracy_score: 0.366666666667
Test confusion_matrix: [[ 3  1  3  0  2 20  7]
 [ 2  3  5  1  1 13  4]
 [ 4  6  8  0  0 16  4]
 [ 0  0  1  0  0  8  0]
 [ 1  1  5  1  0 14  2]
 [ 9  6  4  3  7 90  9]
 [ 3  0  6  1  1 38 17]]
             precision    recall  f1-score   support

          1       0.14      0.08      0.10        36
          2       0.18      0.10      0.13        29
          3       0.25      0.21      0.23        38
          4       0.00      0.00      0.00         9
          5       0.00      0.00      0.00        24
          6       0.45      0.70      0.55       128
          7       0.40      0.26      0.31        66

avg / total       0.31      0.37      0.32       330
```

In [ ]: