- Natural Language Processing
- Text Analysis
- Customer feedback analysis from review analysis

Kim, Da Ye 8971-8937

Text Pattern Analysis

Text analysis and Text analytics

- Detecting Patterns & Trends
- Ambiguity of human language
- Sophisticated classification
- Quantitively analyze

- Word Frequency
- Positive or negative
- Emotions
- Sentiment analysis

How to analyze Text

Tokenization

Splitting text into identifiable data

Stemming

Normalizing words into its base form or root form ex) having -have

Lemmatization

grouping together the different inflected forms of a word

Syntax tree

Classifying words in sentences in grammatical elements.

Word frequency

JAVA : ECLIPSE

Fibonacci Heap
Hash Table

Detecting trends
Quantitative analysis
SNS / Review

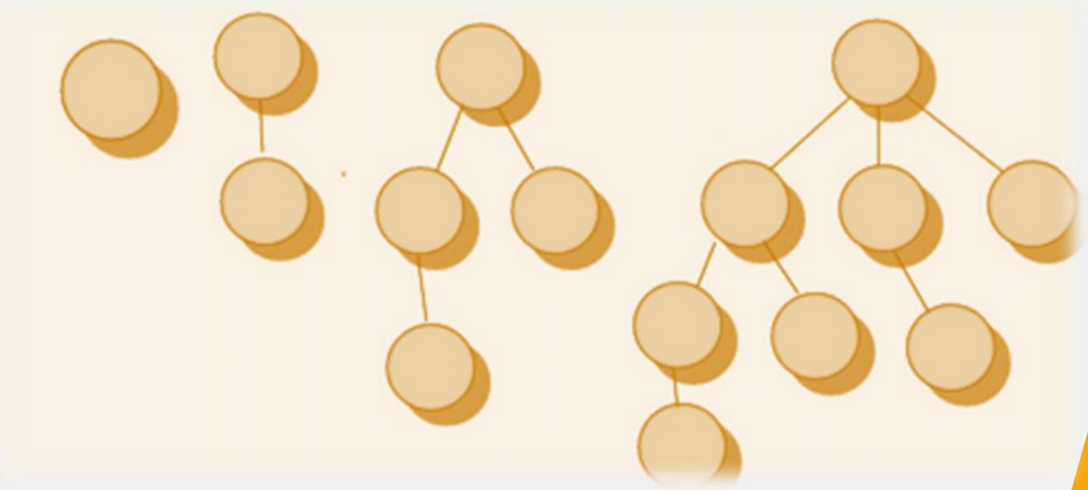
Detect Emotion

Python : Google Colab

The Natural Language ToolKit
Naïve Bayes classifier

Sentiment Analysis
Sophisticated classification
Review

Fibonacci HEAP



HASH TABLE

Word Frequency

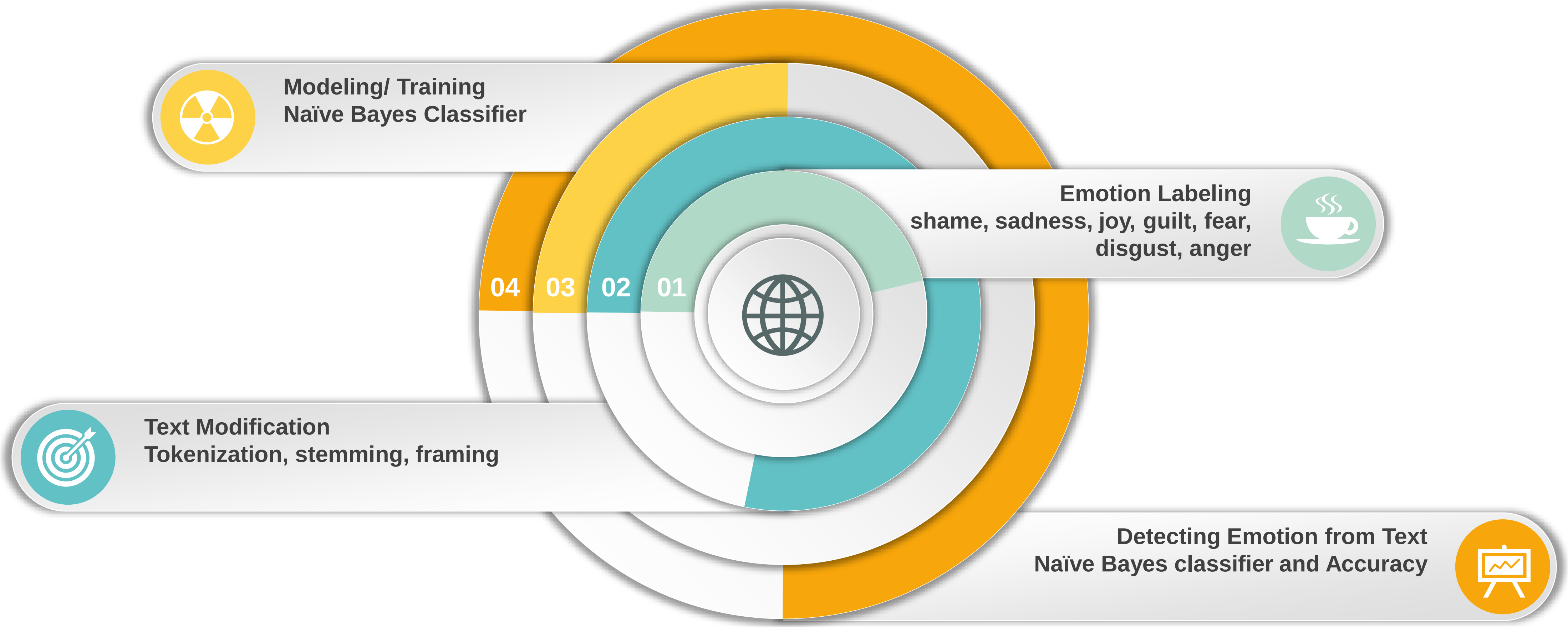
ESSENTIAL FLOW OF MAX FIBONACCI HEAP

1. Detect '#' and read the string[name] and the int[key] from the input-file

2. Create new node and store this in hashmap with the name
,if there is not the same node with the name.
Or, Add up the key from input and the key of the node whose name is the same with the name of input string[name]

3. Performs increaseKey operations in Fibonacci Heap

Detecting Emotion Process



Naive Bayes

Multinomial Naive Bayes

This algorithm is for multinomially distributed data and is used in text classification.

Bernoulli Naive Bayes

This algorithm implements the naïve Bayes training and classification algorithms for data which is distributed according to multivariate Bernoulli distribution.

Categorical Naïve Bayes

this is used for categorically distributed data.

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- *P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

Implementation

Emotion Labels

```
emotion_labels = ['joy', 'fear', 'anger', 'sadness', 'disgust', 'shame',
 'guilt']
# emotion_labels = ['joy', 'fear', 'anger', 'sadness', 'disgust']
```

Negation Words

```
negation_words = ['not', 'neither', 'nor', 'but', 'however', 'although',
 'nonetheless', 'despite', 'except', 'even though', 'yet']
```

Implementation

Removal

```
def removal(sentences):
    sentence_list = []
    count = 0
    s = nltk.word_tokenize(sentenc
es)
    characters = ["á", "\xc3", "\x
a1", "\n", ",", ".", "[", "]", ""]
    l = []
    for t in s:
        if t not in characters:
            l.append(t)
    return l
```

Stemming

```
def stemming(sentences):
    sentence_list = []
    sen_string = []
    sen_token = []
    stemmer = PorterStemmer()
    i = 0
    i += 1
    st = ""
    for word in sentences:
        word_l = word.lower()
        if len(word_l) >= 3:
            st += stemmer.stem(word_l) + " "
    sen_string.append(st)
    w_set = nltk.word_tokenize(st)
    sen_token.append(w_set)
    w_text = nltk.Text(w_set)
    sentence_list.append(w_text)
    return w_text, st, w_set
```

Implementation

Create Frame

```
def create_frame(Data):
    labels = []
    sen = []
    sen_s = []
    sen_t = []
    labelset = []
    for i in range(len(Data)):
        if i >= 0:
#               print i,
            emotion = Data[0][i]
            sit = Data[1][i]
#   if emotion not in ['shame', 'guilt']:
            labels.append(emotion)
            labelset.append([emotion])
            sent = removal(sit)
```

```
nava, sent_pt = pos_tag(sent)
            sentences, sen_string, sen_token
 = stemming(sent_pt)
            sen.append(sentences)
            sen_s.append(sen_string)
            sen_t.append(sen_token)

    frame = pd.DataFrame({0 : labels,
                          1 : sen,
                          2 : sen_s,
                          3 : sen_t,
                          4 : labelset})
    return frame, sen_t, labels, sen_s

-------------------------------------------
c, st, labels, senten = create_frame(Data)
```

Implementation

Input

```
Write to file
'''
def write_to_file(filename, text):
    o = open(filename,'w')
    o.write(str(text))
    o.close()
```

```
import os
def readfile(filename):
    path=os.path.join('/content/Emotion-
Detector/',filename)
    f = open(path,'r')
    representative_words = []
    for line in f.readlines():
        characters = ["\n", " ", "\r", "\t"]
        new = ''.join([i for i in line if no
t [e for e in characters if e in i]])
        representative_words.append(new)
    return representative_words
```

Process

```
'''
Makes a list of all words semantically rel
ated to an emotion and Stemming
'''
def affect_wordlist(words):

Creating an emotion wordnet
'''
def emotion_word_set(emotions):

Getting synonyms from wordnet
'''
def get_synonyms():

Testing for Naive Bayes Classifier
'''
def testing(cl, test):
```

Implementation

Extract

```
def extract_features(document)
:

document_words = set(document)
features = {}
for word in word_features:
        features['contains(%s)
' % word] = (word in document_
words)
    return features
```

Create test data

```
def create_test(sentence, em
otion):
    data = []
    sen = []
    emo = []
    for s in sentence:
        sen.append(str(s))
    for e in emotion:
        emo.append(e)
    for i in range(len(sen))
:
        temp = []
        temp.append(sen[i])
        temp.append(emo[i])
        data.append(temp)
    return data
```

Classifier

```
def classify_dataset(data):
    return \
        classifier.classify(
extract_features(nltk.word_t
okenize(data)))
```

Implementation

Get Accuracy

```
Get accuracy
'''
def get_accuracy(test_data, classifier):
    total = accuracy = float(len(test_data))
    for data in test_data:
        if classify_dataset(data[0]) != data[1]:
            accuracy -= 1
            print "----------Wrong prediction---------
", data, classify_dataset(data[0]), data[1]
        else:
            print data, classify_dataset(data[0]), data[1]
    print('Total accuracy: %f%% (%d/20).' % (accuracy / total * 100, acc
uracy))
    final = accuracy / total * 100
    return final
```

Evaluation

Word Frequency

Detecting emotion from text data

Detecting Patterns & Ambiguity of human language

- Naive Bayes theorem needs well distributed input dataset. After performing this implementation, it shows low accuracy which has a little above 60 percent. This low accuracy value occurs because it has low value of alpha for Laplace smoothing there are words which counts 0. In order to perform high accuracy, we need enormous size of text data which has the huge amount of emotion representing words. However, with this gigantic size of dataset, the implementation runs in inefficient time to get informative value.

Quantitative analysis

- Using the efficient data structure and mathematical concepts, it performs timely to show informative data from analyzing. This quantitative analysis takes usually polynomial time and especially, max Fibonacci heap takes O(1) in amortized time complexity.

Conclusion

Sophisticated classification

-The most important criteria to have accuracy of interpretation human language is to construct highly detailed classified dataset. The more densely the dataset is collected, the more accurate output can come out. Naïve Bayes theorem implements classifications with well distributed data. However, the accuracy depends on how appropriate Naïve Bayes model is used in each different conditions such as the size of data, the number of detectable words, and the distribution of data.

Quantitative analysis

- This performance hardly requires the enormous size of dataset. However, it efficiently performs and provide high accuracy since it is based on fundamental arithmetic operations instead of high abstract mathematical theory.