# Data Representations

Author: Alikhan Semembayev

## 1. Perform necessary data preprocessing, e.g. removing punctuation and stop words, stemming, lemmatizing. You may use the outputs from previous weekly assignments. (10 points)

In [ ]:

```
from collections import defaultdict
import demoji
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from autocorrect import Speller

# Initialize tools
spell = Speller()
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def clean_text(text):
    # Replace emojis
    text = demoji.replace(text)

    # Remove smart quotes and dashes
    text = text.replace("“", "\"").replace("”","\"").replace("-", " ").replace("

    # Lowercase text
    text = text.lower()

    # Tokenize text
    words = word_tokenize(text)
    # print(words)

    # Spelling correction + replace all t with not
    words = ['not' if word == 't' else spell(word) for word in words]

    # Remove stop words and non-alphabetic tokens and punctuation
    words = [word for word in words if word.isalnum() and word not in stop_words

    # POS tagging and Lemmatization
    tagged_words = pos_tag(words)

    tag_map = defaultdict(lambda: "n")
    tag_map["N"] = "n"
    tag_map["V"] = "v"
    tag_map["J"] = "a"
    tag_map["R"] = "r"

    words = [lemmatizer.lemmatize(word, pos=tag_map[tag[0]]) for word, tag in ta
    # Return cleaned words as a single string
    return ' '.join(words)
```

In [ ]:

```
import pandas as pd

data = (pd.read_csv('../../data/text/combined_raw.csv'))
data = data.dropna(how='any')

for row in data.values:
    row[0] = clean_text(row[0])

data.to_csv('../../data/text/combined_cleaned.csv', index=False)
```

# 2. Count BoW on pre-processed data. (10 points)

In [9]:

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from collections import Counter

data = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
data = data.dropna(how='any')

def count_bow(texts):
    bow = Counter()
    for text in texts:
        words = text.split()
        bow.update(words)
    return bow

# Apply BoW counting
bow_counts = count_bow(data['text'])

# Convert the BoW count to a DataFrame
bow_df = pd.DataFrame(list(bow_counts.items()), columns=['Word', 'Count'])

# Save the result to a CSV file
bow_df.to_csv('bow_output.csv', index=False)

print(bow_df.head())
```

```
         Word  Count
0  freshwater     40
1        fish    605
2       drink    379
3       water    970
4        skin    124
```

In [1]:

```
import pandas as pd

data = (pd.read_csv('bow_output.csv'))

data.sort_values("Count", ascending=False, inplace=True)
data = data.head(10)
data.to_csv('bow_output_10.csv', index=False)
```

In [2]:

```
import nltk
import numpy as np
```

```
data = (pd.read_csv('bow_output_10.csv'))
main = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
main = main.dropna(how='any')
main = main.head(1000)

X = []
for row in main['text']:
    vector = []
    for word in data["Word"]:
        if word in nltk.word_tokenize(row):
            vector.append(1)
        else:
            vector.append(0)
    X.append(vector)
X = np.asarray(X)
```

In [3]:

```
# Convert X to a pandas DataFrame to save as CSV
X_df = pd.DataFrame(X, columns=data["Word"])

# Save the DataFrame as a CSV file
X_df.to_csv('bow_vectors.csv', index=False)

print("BoW vectors saved to 'bow_vectors.csv'.")
```

```
BoW vectors saved to 'bow_vectors.csv'.
```

# 3. Compute TF-IDF vectors on pre-processed data. (20 points)

Due to the very large number of unique words, the output of the result has been changed. The second column of the table contains a sheet with TF-IDF vectors.

In [4]:

```
import pandas as pd
import numpy as np
import math
from collections import Counter

# Term Frequency (TF)
def compute_tf(text):
    word_counts = Counter(text.split())
    total_words = len(text.split())
    tf = {word: count / total_words for word, count in word_counts.items()}
    return tf

# Inverse Document Frequency (IDF)
def compute_idf(texts):
    N = len(texts)
    idf = {}
    for text in texts:
        for word in set(text.split()):
            idf[word] = idf.get(word, 0) + 1
    for word, doc_count in idf.items():
        idf[word] = math.log(N / (doc_count + 1))  # Add 1 to avoid division by
    return idf

# Compute TF-IDF for each document
def compute_tfidf(texts):
    idf = compute_idf(texts)
    tfidf = []
    for idx, text in enumerate(texts):
        tf = compute_tf(text)
        tfidf_doc = [(word, tf[word] * idf[word]) for word in tf]  # Store word
        tfidf.append({'Document': idx + 1, 'TF-IDF': tfidf_doc})  # Use index as
    return tfidf

# Apply TF-IDF computation to the clean text
tfidf_alternative = compute_tfidf(data['text'])
```

In [14]:

```
# Convert TF-IDF results to a DataFrame for easier inspection
# Each document will be a row, and the TF-IDF scores will be a list of (word, sc
tfidf_df_alternative = pd.DataFrame(tfidf_alternative)

# Save the revised TF-IDF table to a CSV file
tfidf_df_alternative.to_csv('tfidf_output.csv', index=False)

# Print the first few rows of the TF-IDF table
print(tfidf_df_alternative.head())
```

```
   Document                                             TF-IDF
0         1  [(freshwater, 0.5148123472156599), (fish, 0.71...
1         2  [(think, 0.23834696944350164), (everyone, 0.51...
2         3  [(agree, 0.43660816089717763), (google, 1.0786...
3         4  [(thats, 0.42304351063585904), (funny, 0.36702...
4         5  [(oh, 0.2626823714954244), (yeah, 0.2042693288...
```

# 4. Perform integer encoding and one-hot encoding on one of the pre-processed data files and save the output to a txt file. (30 points)

In [6]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Initialize label encoder
label_encoder = LabelEncoder()

# Integer encoding of the emotion labels
data['emotion_encoded'] = label_encoder.fit_transform(data['emotion'])

# Initialize one-hot encoder
onehot_encoder = OneHotEncoder(sparse_output=False)

# One-hot encoding of the integer encoded labels
emotion_onehot = onehot_encoder.fit_transform(data[['emotion_encoded']])

# Save integer encoded and one-hot encoded labels to a file
encoded_df = pd.DataFrame(emotion_onehot, columns=label_encoder.classes_)
encoded_df.to_csv('emotion_onehot_encoded.txt', index=False)

# Save integer encoding as well
data[['emotion', 'emotion_encoded']].to_csv('emotion_integer_encoded.txt', index

# Display encoded data
print(data[['emotion', 'emotion_encoded']].head())
print(encoded_df.head())
```

```
     emotion  emotion_encoded
0      happy                3
1    neutral                4
2    neutral                4
3    neutral                4
4  surprised                6
   angry  disgust  fear  happy  neutral  sad  surprised
0    0.0      0.0   0.0    1.0      0.0  0.0        0.0
1    0.0      0.0   0.0    0.0      1.0  0.0        0.0
2    0.0      0.0   0.0    0.0      1.0  0.0        0.0
3    0.0      0.0   0.0    0.0      1.0  0.0        0.0
4    0.0      0.0   0.0    0.0      0.0  0.0        1.0
```

# 5. Choose an appropriate word and find the words that are the most similar to it in one of the pre-processed data files. (30 points)

In [7]:

```python
import gensim
import nltk
from gensim.models import Word2Vec

# make a list of movie review documents
# Load the preprocessed data
data = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
data = data.dropna(how='any')

chosen_word = 'happy'

documents = [nltk.word_tokenize(row) for row in data['text']]
model = Word2Vec(documents, min_count=5)
similar_words = model.wv.most_similar(positive = [chosen_word],topn = 25)

if similar_words:
    print(f"Words most similar to '{chosen_word}':")
    for word, score in similar_words:
        print(f"{word}: {score:.4f}")

    # Save the results to a text file
    with open(f'similar_words_bow_{chosen_word}.txt', 'w') as f:
        f.write(f"Words most similar to '{chosen_word}':\n")
        for word, score in similar_words:
            f.write(f"{word}: {score:.4f}\n")
```

```
Words most similar to 'happy':
thankful: 0.6261
birthday: 0.6186
holiday: 0.6133
cheer: 0.6031
thanksgiving: 0.5880
bless: 0.5746
excite: 0.5571
friends: 0.5504
busy: 0.5363
wonderful: 0.5347
invigorated: 0.5341
lovely: 0.5316
joy: 0.5316
stress: 0.5315
pleasant: 0.5294
thebodyshopuk: 0.5283
abruptly: 0.5261
sick: 0.5205
sweet: 0.5193
dinner: 0.5095
hope: 0.5093
comfortable: 0.5039
excellent: 0.4995
celebrate: 0.4974
present: 0.4961
```

In [ ]: