```
Classifying Reddit comments that are about physics, chemistry and biology.
- First, we preprocess the corpus using regex and nltk's tokenization and lemmatization
- Then we use word2vec to create a representation for each word
- after that, we cluster the representations in order to find the words that are close together in terms of their meaning
- We then replace all the worlds of each cluster with the representative word of that cluster in the corpus. (this effectively makes sure that all the words in a cluster take the same spot in the bag of words vector.
- Finally, we create a representation for each comment using bag of words and TF-IDF and try to classify the test data set.
@arashsm79
```

Contents:

Pre-processing

In [2]:
```python
import pandas as pd
import numpy as np
import nltk
import string
import re

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer

from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```python
# nltk.download('stopwords')
# nltk.download('punkt')
# nltk.download('wordnet')
# nltk.download('omw-1.4')
```

In [2]:
```python
raw_train = pd.read_csv("train.csv")
raw_train.head()
```

Out[2]:

	Id	Comment	Topic
0	0x840	A few things. You might have negative- frequen...	Biology
1	0xbf0	Is it so hard to believe that there exist part...	Physics
2	0x1dfc	There are bees	Biology
3	0xc7e	I'm a medication technician. And that's alot o...	Biology
4	0xbba	Cesium is such a pretty metal.	Chemistry

Dataframe Cleanup

In [3]:
```python
# Convert categorical data to numerical
raw_train['Topic'] = LabelEncoder().fit_transform(raw_train['Topic'])

# Drop the Id column
raw_train = raw_train.drop(columns=["Id"])

raw_train.head()
```

Out[3]:

	Comment	Topic
0	A few things. You might have negative- frequen...	0
1	Is it so hard to believe that there exist part...	2
2	There are bees	0
3	I'm a medication technician. And that's alot o...	0
4	Cesium is such a pretty metal.	1

Text Cleanup

In [4]:
```
def text_cleanup_regex(txt: str) -> str:
    # Convert to lower case and remove bounding space
    txt = txt.lower().strip()

    # Remove reddit specific texts
    txt = re.sub("\[removed\]", " ", txt)
    txt = re.sub("\[deleted\]", " ", txt)

    # Remove URLs
    txt = re.sub("http\S+", " ", txt)
    txt = re.sub("www\.\S+", " ", txt)

    # Remove everything that is not a letter
    txt = re.sub("[^a-zA-Z]", " ", txt)
    # txt = re.sub('[%s]' % re.escape(string.punctuation), ' ', txt)

    # Remove single and double letter words
    txt = re.sub(r'\b\w{1,2}\b', ' ', txt)

    return txt
```

Cleanup Pipeline

In [5]:
```
english_stopwords = stopwords.words("english")
lemmatizer = WordNetLemmatizer()

def text_cleanup_pipeline(txt: str) -> [str]:
    # basic clean up using regex
    txt: str   = text_cleanup_regex(txt)

    # tokenize (convert string of words to array of words)
    txt: [str] = word_tokenize(txt)

    # make sure the word is not a stopword
    txt: [str] = [w for w in txt if w not in english_stopwords]

    # take  out the root of the word
    txt: [str] = [lemmatizer.lemmatize(w) for w in txt]

    return txt
```

In [6]:
```
# Clean up comments using text_cleanup_pipeline
raw_train['Comment'] = raw_train['Comment'].apply(lambda x : text_cleanup_pipeline(x))
raw_train.head()
```

Out[6]:

	Comment	Topic
0	[thing, might, negative, frequency, dependent,...	0
1	[hard, believe, exist, particular, detect, any...	2
2	[bee]	0
3	[medication, technician, alot, drug, liver, pr...	0
4	[cesium, pretty, metal]	1

In [7]:
```
# Drop rows with empty comment
size_before_empty_drop = raw_train.shape[0]
raw_train = raw_train[raw_train['Comment'].apply(lambda x: len(x) > 0)]
print("Dropped: {}".format(size_before_empty_drop - raw_train.shape[0]))
raw_train.head()
```

```
Dropped: 286
```

Out[7]:

	Comment	Topic
0	[thing, might, negative, frequency, dependent,...	0
1	[hard, believe, exist, particular, detect, any...	2
2	[bee]	0
3	[medication, technician, alot, drug, liver, pr...	0
4	[cesium, pretty, metal]	1

In [8]:
```
# Convert list of words to string
raw_train['Comment'] = raw_train['Comment'].apply(lambda x : " ".join(x))
raw_train.head()
```

Out[8]:

	Comment	Topic
0	thing might negative frequency dependent selec...	0
1	hard believe exist particular detect anything ...	2
2	bee	0
3	medication technician alot drug liver probably...	0
4	cesium pretty metal	1

In [11]:
```
# Do the same preprocessing for train data
raw_test = pd.read_csv("test.csv")
raw_test['Topic'] = LabelEncoder().fit_transform(raw_test['Topic'])
raw_test = raw_test.drop(columns=["Id"])
raw_test['Comment'] = raw_test['Comment'].apply(lambda x : text_cleanup_pipeline(x))
raw_test = raw_test[raw_test['Comment'].apply(lambda x: len(x) > 0)]
raw_test['Comment'] = raw_test['Comment'].apply(lambda x : " ".join(x))
```

To avoid having to preprocess the data again in case something goes wrong, copy the preprocessed data to a new variable.

In [12]:
```
train_data = raw_train.copy()
train_data.shape
```

Out[12]:
```
(8409, 2)
```

In [13]:
```
test_data = raw_test.copy()
test_data.shape
```

Out[13]:
```
(1586, 2)
```

Word2Vec

Take out the unique words.

In [158…
```
unique_words: [str] = list(set(" ".join(train_data['Comment'].str.lower().values.tolist()).split(" ")))
unique_words.sort()
print("Size: {}".format(len(unique_words)))
unique_words: np.ndarray = np.array(unique_words)
unique_words[10:20]
```

```
Size: 14574
```

Out[158]:
```
array(['abbreviation', 'abd', 'abdomen', 'abdominal', 'abduction',
       'abetted', 'abhor', 'abi', 'ability', 'abiotic'], dtype='<U46')
```

- Get a list o unique words
- Use googles trained W2V to get a vector for each word
- Create a dataframe and combine the words with their vectors
- Save te data frame to avoid loading the 1.5 Gb model everytime

In [38]:
```
from gensim.models import KeyedVectors
from collections import Counter
```

In [39]:
```
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

In [66]:
```
# Select words that have a vector
unique_words = np.fromiter((x for x in unique_words if model.has_index_for(x)), dtype=unique_words.dtype)
print("Size: {}".format(len(unique_words)))
unique_words[10:20]
```

```
Size: 12164
```

Out[66]:
```
array(['abhor', 'abi', 'ability', 'abiotic', 'abject', 'able', 'abnormal',
       'abolish', 'abomination', 'abort'], dtype='<U46')
```

In [67]:
```
# Get the corresponding vector for each word
unique_words_vectors = np.array(list(map(lambda x: model[x], unique_words)))
unique_words_vectors.shape
```

Out[67]:
```
(12164, 300)
```

In [70]:
```
# Zip words and vectors together
word_vecs = pd.DataFrame(unique_words_vectors)
word_vecs.insert(loc=0, column='word', value=unique_words)
word_vecs.head()
```

Out[70]:

	word	0	1	2	3	4	5	6	7	8	...	290	291	292	
0	aaa	0.031738	0.021973	0.041016	0.328125	-0.113281	-0.337891	0.082520	-0.208984	0.410156	...	0.044922	0.449219	-0.138672	-0.2
1	aah	0.163086	-0.010132	0.105957	0.229492	-0.070801	-0.458984	0.206055	-0.213867	0.150391	...	-0.002899	0.392578	-0.214844	-0.0
2	ab	0.064941	0.241211	0.054443	0.191406	-0.075684	0.199219	-0.051514	-0.163086	-0.243164	...	0.068359	0.042969	-0.139648	-0.0
3	abandon	0.007660	0.078613	0.109375	0.339844	-0.208984	0.044678	-0.036621	-0.041992	0.192383	...	-0.033203	0.355469	-0.205078	0.0
4	abbreviation	0.218750	-0.209961	0.138672	0.386719	-0.277344	-0.208008	-0.382812	-0.376953	0.161133	...	-0.020020	-0.084473	-0.120117	-0.1

5 rows × 301 columns

In [71]:
```
# Save the dataframe to a file
word_vecs.to_pickle("reddit_w2v.pkl")
```

Load saved vectors

In [3]:
```
# Load the dataframe from a file
output = pd.read_pickle("reddit_w2v.pkl")
output.head()
```

Out[3]:

	word	0	1	2	3	4	5	6	7	8	...	290	291	292	
0	aaa	0.031738	0.021973	0.041016	0.328125	-0.113281	-0.337891	0.082520	-0.208984	0.410156	...	0.044922	0.449219	-0.138672	-0.2
1	aah	0.163086	-0.010132	0.105957	0.229492	-0.070801	-0.458984	0.206055	-0.213867	0.150391	...	-0.002899	0.392578	-0.214844	-0.0
2	ab	0.064941	0.241211	0.054443	0.191406	-0.075684	0.199219	-0.051514	-0.163086	-0.243164	...	0.068359	0.042969	-0.139648	-0.0
3	abandon	0.007660	0.078613	0.109375	0.339844	-0.208984	0.044678	-0.036621	-0.041992	0.192383	...	-0.033203	0.355469	-0.205078	0.0
4	abbreviation	0.218750	-0.209961	0.138672	0.386719	-0.277344	-0.208008	-0.382812	-0.376953	0.161133	...	-0.020020	-0.084473	-0.120117	-0.1

5 rows × 301 columns

In [7]:
```
compression_opts = dict(method='zip',
                        archive_name='out.csv')
output.to_csv("reddit_w2v.zip", index=False, compression=compression_opts)
```

In [26]:
```
words = output['word']
output: pd.DataFrame = output.iloc[:, 1:301]
vectors = output.to_numpy()
vectors.shape
```

Out[26]:
```
array([[ 0.03173828,  0.02197266,  0.04101562, ...,  0.01818848,
        -0.37695312,  0.16894531],
       [ 0.16308594, -0.01013184,  0.10595703, ..., -0.296875  ,
        -0.1171875 ,  0.25      ],
       [ 0.06494141,  0.24121094,  0.05444336, ...,  0.33789062,
        -0.05102539,  0.04760742],
       ...,
       [-0.07421875, -0.10205078,  0.20117188, ..., -0.25390625,
         0.06054688, -0.21289062],
       [ 0.18554688,  0.16894531,  0.00267029, ..., -0.33007812,
         0.05004883, -0.21191406],
       [-0.08935547,  0.10351562,  0.06787109, ...,  0.0378418 ,
         0.05053711,  0.01940918]], dtype=float32)
```

Clustering-Word2Vec

In [135…
```
from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from IPython.display import clear_output
import time
```

In [ ]:
```
pca_vectors = PCA(n_components=2).fit_transform(vectors)
pca_vectors.shape
```

In [48]:
```
std_vectors = StandardScaler().fit_transform(pca_vectors)
```

In [145…
```
key_iden = KMeans(n_clusters=10000, random_state=0).fit(pca_vectors)
```


In [ ]:
```
# iterate through the clusters and print some of the clustered words
```

```
for i in range(key_iden.n_clusters):
    n = words[key_iden.labels_ == i].shape[0]
    if n > 1 and n < 10:
        print(words[key_iden.labels_ == i].shape)
        print(words[key_iden.labels_ == i])
        time.sleep(1)
        clear_output(wait=True)
```

In [184…
```
key_iden.labels_
```

Out[184]:
```
array([4269, 6165,  546, ..., 3362, 6058, 5095], dtype=int32)
```

In [151…
```
N_ELEMENTS_UPPER_THRESHOLD = 10
N_ELEMENTS_LOWER_THRESHOLD = 1

# Find cluster numbers with proper number of elements
good_cluster_numbers = []
for i in range(key_iden.n_clusters):
    n = words[key_iden.labels_ == i].shape[0]
    if n > N_ELEMENTS_LOWER_THRESHOLD and n < N_ELEMENTS_UPPER_THRESHOLD:
        good_cluster_numbers.append(i)

# Find a representation for each cluster (here we take the first word)
good_cluster_words = []
for i in good_cluster_numbers:
    good_cluster_words.append(words[key_iden.labels_ == i].iloc[0])
```

In [172…
```
topic_modeled_train_data = train_data.copy()
for i, cluster_num in enumerate(good_cluster_numbers):
    for word in words[key_iden.labels_ == cluster_num]:
        topic_modeled_train_data['Comment'].replace(word, good_cluster_words[i], inplace=True, regex=True)
```

In [173…
```
topic_modeled_unique_words: [str] = list(set(" ".join(topic_modeled_train_data['Comment'].str.lower().values.tolist()).split(" ")))
print("Unique words: {}".format(len(unique_words)))
print("Topic modeled unique words: {}".format(len(topic_modeled_unique_words)))
print("Diff: {}".format(len(unique_words) - len(topic_modeled_unique_words)))
```

```
Unique words: 14574
Topic modeled unique words: 12961
Diff: 1613
```

In [200…
```
topic_modeled_test_data = test_data.copy()
for i, cluster_num in enumerate(good_cluster_numbers):
    for word in words[key_iden.labels_ == cluster_num]:
        topic_modeled_test_data['Comment'].replace(word, good_cluster_words[i], inplace=True, regex=True)
```

Bag-of-words and TF-IDF

In [176…
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
```

In [14]:
```
# Seperate labels
x_train = train_data["Comment"]
y_train = train_data["Topic"]

x_test = test_data["Comment"]
y_test = test_data["Topic"]
```

Bag of Words

In [104…
```
# BoW
bow_vectorizer = CountVectorizer()
bow_train = bow_vectorizer.fit_transform(x_train)
bow_test = bow_vectorizer.transform(x_test)
```

In [108…
```
# Classification
clf = MultinomialNB(alpha=0.1)
clf.fit(bow_train, y_train)
y_pred = clf.predict(bow_test)

# Classification score
print(classification_report(y_test, y_pred))
print("Train Score: {}".format(clf.score(bow_train, y_train)))
print("Test Score: {}".format(clf.score(bow_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.90      0.82      0.86       614
           1       0.79      0.85      0.82       506
           2       0.84      0.86      0.85       466

    accuracy                           0.84      1586
   macro avg       0.84      0.85      0.84      1586
weighted avg       0.85      0.84      0.84      1586

Train Score: 0.8800095136163634
Test Score: 0.8442622950819673
```

TF-IDF

In [109…
```
# TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_train = tfidf_vectorizer.fit_transform(x_train)
tfidf_test = tfidf_vectorizer.transform(x_test)
```

In [111…
```
# Classification
clf = MultinomialNB(alpha=0.1)
clf.fit(tfidf_train, y_train)
y_pred = clf.predict(tfidf_test)

# Classification score
print(classification_report(y_test, y_pred))
print("Train Score: {}".format(clf.score(tfidf_train, y_train)))
print("Test Score: {}".format(clf.score(tfidf_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.84      0.86       614
           1       0.79      0.85      0.82       506
           2       0.85      0.84      0.85       466

    accuracy                           0.84      1586
   macro avg       0.84      0.84      0.84      1586
weighted avg       0.85      0.84      0.84      1586

Train Score: 0.9086692829111666
Test Score: 0.8430012610340479
```

Bag-of-words and TF-IDF with Word2Vec Topic Modeling

In [201…
```
# Seperate labels
x_train = topic_modeled_train_data["Comment"]
y_train = topic_modeled_train_data["Topic"]

x_test = topic_modeled_test_data["Comment"]
y_test = topic_modeled_test_data["Topic"]
```

Bag of Words

In [202…
```
# BoW
bow_vectorizer = CountVectorizer()
bow_train = bow_vectorizer.fit_transform(x_train)
bow_test = bow_vectorizer.transform(x_test)
```

In [203…
```
# Classification
clf = MultinomialNB(alpha=0.1)
clf.fit(bow_train, y_train)
y_pred = clf.predict(bow_test)

# Classification score
print(classification_report(y_test, y_pred))
print("Train Score: {}".format(clf.score(bow_train, y_train)))
print("Test Score: {}".format(clf.score(bow_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       614
           1       0.79      0.84      0.81       506
           2       0.84      0.87      0.86       466

    accuracy                           0.84      1586
   macro avg       0.84      0.84      0.84      1586
weighted avg       0.84      0.84      0.84      1586

Train Score: 0.8741824235937686
```

TF-IDF

In [204…

```
# TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_train = tfidf_vectorizer.fit_transform(x_train)
tfidf_test = tfidf_vectorizer.transform(x_test)
```

In [205…

```
# Classification
clf = MultinomialNB(alpha=0.1)
clf.fit(tfidf_train, y_train)
y_pred = clf.predict(tfidf_test)

# Classification score
print(classification_report(y_test, y_pred))
print("Train Score: {}".format(clf.score(tfidf_train, y_train)))
print("Test Score: {}".format(clf.score(tfidf_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.88      0.82      0.85       614
           1       0.79      0.85      0.82       506
           2       0.85      0.86      0.86       466

    accuracy                           0.84      1586
   macro avg       0.84      0.84      0.84      1586
weighted avg       0.84      0.84      0.84      1586

Train Score: 0.904150315138542
Test Score: 0.8398486759142497
```

In [ ]: