In [1]:
```
!python -m pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\mryua\anaconda3\lib\site-p
ackages (1.6.1)
Requirement already satisfied: scipy in c:\users\mryua\anaconda3\lib\site-pac
kages (from xgboost) (1.5.0)
Requirement already satisfied: numpy in c:\users\mryua\anaconda3\lib\site-pac
kages (from xgboost) (1.18.5)
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import f1_score,accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
import altair as alt
import seaborn as sns
import re
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
```

In [3]:
```
#Make sure numpy version is < 1.20
np.version.version
```

Out[3]:
```
'1.18.5'
```

In [4]:
```
#Install known version of numpy that works
!python -m pip install numpy==1.18.5
```

```
Requirement already satisfied: numpy==1.18.5 in c:\users\mryua\anaconda3\lib
\site-packages (1.18.5)
```

In [5]:
```
#Install gensim
!python -m pip install gensim
```

```
Requirement already satisfied: gensim in c:\users\mryua\anaconda3\lib\site-pa
ckages (4.1.2)
Requirement already satisfied: numpy>=1.17.0 in c:\users\mryua\anaconda3\lib
\site-packages (from gensim) (1.18.5)
Requirement already satisfied: Cython==0.29.23 in c:\users\mryua\anaconda3\li
b\site-packages (from gensim) (0.29.23)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\mryua\anaconda3
\lib\site-packages (from gensim) (6.0.0)
Requirement already satisfied: scipy>=0.18.1 in c:\users\mryua\anaconda3\lib
\site-packages (from gensim) (1.5.0)
```

In [6]:
```
import gensim
from gensim.models.word2vec import Word2Vec
from tqdm.notebook import tqdm
from nltk.tokenize import sent_tokenize, word_tokenize
from collections import Counter
from nltk.corpus import stopwords
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mryua\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mryua\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mryua\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[6]: True

In [7]:
```
RANDOM_SEED=694
```

1.Loading data

Training Dataset

In [8]:
```
train_path = 'Data/WikiLarge_Train.csv'
df = pd.read_csv(train_path, skiprows=0, skipfooter=0, engine='python')
df.head()
```

Out[8]:

	original_text	label
0	There is manuscript evidence that Austen conti...	1
1	In a remarkable comparative analysis , Mandaea...	1
2	Before Persephone was released to Hermes , who...	1
3	Cogeneration plants are commonly found in dist...	1
4	Geneva -LRB- , ; , ; , ; ; -RRB- is the second...	1

In [9]:
```
len(df[df['label']==1])/len(df) # the dataset label is well balanced
```

Out[9]: 0.5

In [10]:
```
#Adding a column of text length for exploration purpose only
df['text_length'] = df['original_text'].apply(len)
```

In [11]:
```
#Inspecting data with different label and text length combinations
df[(df['label']==0) & (df['text_length']==5)].head()
```

Out[11]:

	original_text	label	text_length
208709	Pages	0	5
208988	Plain	0	5
209004	Drama	0	5
209374	Child	0	5
209606	equal	0	5

In [12]:
```
df.head()
```

Out[12]:

	original_text	label	text_length
0	There is manuscript evidence that Austen conti...	1	216
1	In a remarkable comparative analysis , Mandaea...	1	156
2	Before Persephone was released to Hermes , who...	1	248
3	Cogeneration plants are commonly found in dist...	1	246
4	Geneva -LRB- , ; , ; , ; ; -RRB- is the second...	1	202

In [13]:
```
df['original_text'].apply(lambda x: len(x)).mean()
# This means all texts are considered short text, which allows us to use dense
representations,
# as dense representations work well with short text.
# Gensim.KeyedVectors.load('assets/wikipedia.100.word-vecs.kv')??? How to gene
rate and use this???
# Maybe we should train word2vec model on the entire corpus. Just training dat
a? TOP 100 word-vectors(features)
# Alternatively we could use bag-of-words model, which is term-document matrix
representation, having much more features
```

Out[13]: 117.921906192414

In [14]:
```
X = df['original_text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=42)
```

Testing Dataset

In [15]:
```
test_path = 'Data/WikiLarge_Test.csv'
test_df = pd.read_csv(test_path, skiprows=0, skipfooter=0, engine='python')
test_df.head()
```

Out[15]:

	id	original_text	label
0	0	-2011	NaN
1	1	-2011	NaN
2	2	-2000	NaN
3	3	-1997	NaN
4	4	1.636	NaN

Sample Submission

In [16]:
```
samplesubmission_path = 'Data/sampleSubmission.csv'
samplesubmission_df = pd.read_csv(samplesubmission_path, skiprows=0, skipfoote
r=0, engine='python')
samplesubmission_df.head()
```

Out[16]:

	id	label
0	0	0
1	1	0
2	2	1
3	3	1
4	4	0

To conclude, the dataframes we are working with are:

train_df, test_df, samplesubmission_df

2. Data Preprocessing

In [17]:
```
# Using tf-idf model, to generate a vectorized representation of the document
s.
vectorizer = TfidfVectorizer(min_df=10,stop_words='english',ngram_range=(1,2))
X_train_transform = vectorizer.fit_transform(X_train)
X_test_transform  = vectorizer.transform(X_test)
```

In [18]:
```
X_train_transform
```

Out[18]:
```
<333414x57773 sparse matrix of type '<class 'numpy.float64'>'
	with 4071111 stored elements in Compressed Sparse Row format>
```

In [19]:
```
len(set(stopwords.words('english')))
```
Out[19]:
```
179
```

== dale_chall.txt ==

This is the Dale Chall 3000 Word List, which is one definition of words that are considered "basic" English.

A summary is at https://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php (https://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php)

In [20]:
```
#Basic english words
dalechall_path = 'Data/dale_chall.txt'
dale_chall = pd.read_csv(dalechall_path,delimiter='\t',header=None,names=['word'])
dale = set(dale_chall['word'].values)
```

In [21]:
```
len(dale)
```
Out[21]:
```
2946
```

The 2946 words in dale can be combined with the nltk stopwords, as they are considered easy words.

We will use a geo dataset to add city and country names to the stopwords library

In [22]:
```
!python -m pip install datapackage
```

```
Requirement already satisfied: datapackage in c:\users\mryua\anaconda3\lib\si
te-packages (1.15.2)
Requirement already satisfied: chardet>=3.0 in c:\users\mryua\anaconda3\lib\s
ite-packages (from datapackage) (3.0.4)
Requirement already satisfied: requests>=2.8 in c:\users\mryua\anaconda3\lib
\site-packages (from datapackage) (2.24.0)
Requirement already satisfied: jsonschema>=2.5 in c:\users\mryua\anaconda3\li
b\site-packages (from datapackage) (3.2.0)
Requirement already satisfied: tableschema>=1.12.1 in c:\users\mryua\anaconda
3\lib\site-packages (from datapackage) (1.20.2)
Requirement already satisfied: tabulator>=1.29 in c:\users\mryua\anaconda3\li
b\site-packages (from datapackage) (1.53.5)
Requirement already satisfied: click>=6.7 in c:\users\mryua\anaconda3\lib\sit
e-packages (from datapackage) (7.1.2)
Requirement already satisfied: jsonpointer>=1.10 in c:\users\mryua\anaconda3
\lib\site-packages (from datapackage) (2.3)
Requirement already satisfied: unicodecsv>=0.14 in c:\users\mryua\anaconda3\l
ib\site-packages (from datapackage) (0.14.1)
Requirement already satisfied: six>=1.10 in c:\users\mryua\anaconda3\lib\site
-packages (from datapackage) (1.15.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
c:\users\mryua\anaconda3\lib\site-packages (from requests>=2.8->datapackage)
(1.25.9)
Requirement already satisfied: idna<3,>=2.5 in c:\users\mryua\anaconda3\lib\s
ite-packages (from requests>=2.8->datapackage) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\mryua\anaconda3
\lib\site-packages (from requests>=2.8->datapackage) (2021.5.30)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\mryua\anaconda3
\lib\site-packages (from jsonschema>=2.5->datapackage) (0.16.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\mryua\anaconda3\lib
\site-packages (from jsonschema>=2.5->datapackage) (19.3.0)
Requirement already satisfied: setuptools in c:\users\mryua\anaconda3\lib\sit
e-packages (from jsonschema>=2.5->datapackage) (49.2.0.post20200714)
Requirement already satisfied: cached-property>=1.5 in c:\users\mryua\anacond
a3\lib\site-packages (from tableschema>=1.12.1->datapackage) (1.5.2)
Requirement already satisfied: python-dateutil>=2.4 in c:\users\mryua\anacond
a3\lib\site-packages (from tableschema>=1.12.1->datapackage) (2.8.1)
Requirement already satisfied: rfc3986>=1.1.0 in c:\users\mryua\anaconda3\lib
\site-packages (from tableschema>=1.12.1->datapackage) (2.0.0)
Requirement already satisfied: isodate>=0.5.4 in c:\users\mryua\anaconda3\lib
\site-packages (from tableschema>=1.12.1->datapackage) (0.6.1)
Requirement already satisfied: ijson>=3.0.3 in c:\users\mryua\anaconda3\lib\s
ite-packages (from tabulator>=1.29->datapackage) (3.1.4)
Requirement already satisfied: jsonlines>=1.1 in c:\users\mryua\anaconda3\lib
\site-packages (from tabulator>=1.29->datapackage) (3.0.0)
Requirement already satisfied: sqlalchemy>=0.9.6 in c:\users\mryua\anaconda3
\lib\site-packages (from tabulator>=1.29->datapackage) (1.3.18)
Requirement already satisfied: openpyxl>=2.6 in c:\users\mryua\anaconda3\lib
\site-packages (from tabulator>=1.29->datapackage) (3.0.4)
Requirement already satisfied: xlrd>=1.0 in c:\users\mryua\anaconda3\lib\site
-packages (from tabulator>=1.29->datapackage) (1.2.0)
Requirement already satisfied: linear-tsv>=1.0 in c:\users\mryua\anaconda3\li
b\site-packages (from tabulator>=1.29->datapackage) (1.1.0)
Requirement already satisfied: boto3>=1.9 in c:\users\mryua\anaconda3\lib\sit
e-packages (from tabulator>=1.29->datapackage) (1.23.0)
Requirement already satisfied: jdcal in c:\users\mryua\anaconda3\lib\site-pac
kages (from openpyxl>=2.6->tabulator>=1.29->datapackage) (1.4.1)
```

```
Requirement already satisfied: et-xmlfile in c:\users\mryua\anaconda3\lib\sit
e-packages (from openpyxl>=2.6->tabulator>=1.29->datapackage) (1.0.1)
Requirement already satisfied: s3transfer<0.6.0,>=0.5.0 in c:\users\mryua\ana
conda3\lib\site-packages (from boto3>=1.9->tabulator>=1.29->datapackage) (0.
5.2)
Requirement already satisfied: botocore<1.27.0,>=1.26.0 in c:\users\mryua\ana
conda3\lib\site-packages (from boto3>=1.9->tabulator>=1.29->datapackage) (1.2
6.0)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in c:\users\mryua\anaco
nda3\lib\site-packages (from boto3>=1.9->tabulator>=1.29->datapackage) (0.10.
0)
```

In [23]:
```
# Run a second time if this cell fails.
from datapackage import Package
package = Package('https://datahub.io/core/world-cities/datapackage.json')
# print list of all resources:
print(package.resource_names)
```

```
['validation_report', 'world-cities_csv', 'world-cities_json', 'world-cities_
zip', 'world-cities_csv_preview', 'world-cities']
```

In [25]:
```
# Run a second time if this cell fails.
world_cities = []
for resource in package.resources:
    if resource.descriptor['datahub']['type'] == 'derived/csv':
        world_cities = resource.read()
```

In [27]:
```
world_cities_df = pd.DataFrame(world_cities, columns=['name', 'country', 'subc
ountry', 'geonameid'])
```

In [28]:
```
world_cities_df.head()
```

Out[28]:

	name	country	subcountry	geonameid
0	les Escaldes	Andorra	Escaldes-Engordany	3040051
1	Andorra la Vella	Andorra	Andorra la Vella	3041563
2	Umm al Qaywayn	United Arab Emirates	Umm al Qaywayn	290594
3	Ras al-Khaimah	United Arab Emirates	Ra’s al Khaymah	291074
4	Khawr Fakkān	United Arab Emirates	Ash Shāriqah	291696

In [29]:
```
world_cities_df = world_cities_df.applymap(lambda s:s.lower() if type(s) == st
r else s)
```

In [30]:
```
world_cities_df[world_cities_df['country']=='france'].head()
```

Out[30]:

	name	country	subcountry	geonameid
6633	yerres	france	île-de-france	2967245
6634	wittenheim	france	alsace-champagne-ardenne-lorraine	2967318
6635	wattrelos	france	nord-pas-de-calais-picardie	2967421
6636	wasquehal	france	nord-pas-de-calais-picardie	2967438
6637	voiron	france	auvergne-rhône-alpes	2967758

In [31]:
```
cities = set(world_cities_df['name'].unique())
countries = set(world_cities_df['country'].unique())
subcountries = set(world_cities_df['subcountry'].unique())
```

In [32]:
```
#We will add this to stopwords
geo_data = cities | countries | subcountries
```

In [33]:
```
len(geo_data)
```

Out[33]: 23803

We will use a language dataset to add language names to the stopwords library

In [34]:
```
language_package = Package('https://datahub.io/core/language-codes/datapackag
e.json')

# print list of all resources:
print(language_package.resource_names)
```

```
['validation_report', 'language-codes_csv', 'language-codes-3b2_csv', 'langua
ge-codes-full_csv', 'ietf-language-tags_csv', 'language-codes_json', 'languag
e-codes-3b2_json', 'language-codes-full_json', 'ietf-language-tags_json', 'la
nguage-codes_zip', 'language-codes', 'language-codes-3b2', 'language-codes-fu
ll', 'ietf-language-tags']
```

In [35]:
```
languages_data = language_package.resources[1].read()
```

In [36]:
```
languages_df = pd.DataFrame(languages_data, columns=['alpha2', 'english'])
languages_df = languages_df.applymap(lambda s:s.lower() if type(s) == str else s)
languages_df.head()
```

Out[36]:

	alpha2	english
0	aa	afar
1	ab	abkhazian
2	ae	avestan
3	af	afrikaans
4	ak	akan

In [37]:
```
languages = set(languages_df['english'].unique())
```

In [38]:
```
len(languages)
```

Out[38]: 184

We will use a nationality dataset to add nationality names to the stopwords library

In [39]:
```
nationality_path = 'Data/CH_Nationality_List_20171130_v1.csv'
nationality_df = pd.read_csv(nationality_path, skiprows=0, skipfooter=0, engine='python')
nationality_df = nationality_df.applymap(lambda s:s.lower() if type(s) == str else s)
nationality_df.head()
```

Out[39]:

	Nationality
0	afghan
1	albanian
2	algerian
3	american
4	andorran

In [40]:
```
nationalities = set(nationality_df['Nationality'].unique())
len(nationalities)
```

Out[40]: 225

We will use a state name dataset to add state names to the stopwords library

In [41]:
```
states_path = 'Data/states.csv'
states_df = pd.read_csv(states_path, skiprows=0, skipfooter=0, engine='python'
)
states_df = states_df.applymap(lambda s:s.lower() if type(s) == str else s)
states_df.head()
```

Out[41]:

	id	name	country_id	country_code	country_name	state_code	type	latitude	lon
0	3901	badakhshan	1	af	afghanistan	bds	NaN	36.734772	70.8
1	3871	badghis	1	af	afghanistan	bdg	NaN	35.167134	63.7
2	3875	baghlan	1	af	afghanistan	bgl	NaN	36.178903	68.7
3	3884	balkh	1	af	afghanistan	bal	NaN	36.755060	66.8
4	3872	bamyan	1	af	afghanistan	bam	NaN	34.810007	67.8

In [42]:
```
states = set(states_df['name'].unique())
len(states)
```

Out[42]: 4896

We will use a continent name dataset to add continent names to the stopwords library

In [43]:
```
continents_path = 'Data/continents2.csv'
continents_df = pd.read_csv(continents_path, skiprows=0, skipfooter=0, engine=
'python')
continents_df = continents_df.applymap(lambda s:s.lower() if type(s) == str el
se s)
continents_df.head()
```

Out[43]:

	ï»¿name	alpha-2	alpha-3	country-code	iso_3166-2	region	sub-region	intermediate-region	region-code	s regi co
0	afghanistan	af	afg	4	iso 3166-2:af	asia	southern asia	NaN	142.0	3
1	ã…land islands	ax	ala	248	iso 3166-2:ax	europe	northern europe	NaN	150.0	15
2	albania	al	alb	8	iso 3166-2:al	europe	southern europe	NaN	150.0	3
3	algeria	dz	dza	12	iso 3166-2:dz	africa	northern africa	NaN	2.0	1
4	american samoa	as	asm	16	iso 3166-2:as	oceania	polynesia	NaN	9.0	6

In [44]:
```
continents = set(continents_df['region'].unique())
len(continents)
```

Out[44]: 6

We will use a firstname dataset to add first names to the stopwords library

In [45]:
```
firstname_path = 'Data/new-top-firstNames.csv'
firstname_df = pd.read_csv(firstname_path, skiprows=0, skipfooter=0, engine='p
ython')
firstname_df = firstname_df.applymap(lambda s:s.lower() if type(s) == str else
s)
firstname_df.head()
```

Out[45]:

	Unnamed: 0	name	newPerct2013
0	1	michael	0.011577
1	2	james	0.010218
2	3	john	0.009675
3	4	robert	0.009493
4	5	david	0.008943

In [46]:
```
firstnames = set(firstname_df['name'].unique())
len(firstnames)
```

Out[46]: 100

In [47]:
```
firstname_path2 = 'Data/babynames-clean.csv'
firstname_df2 = pd.read_csv(firstname_path2, header= None, skiprows=0, skipfoo
ter=0, engine='python')
firstname_df2 = firstname_df2.applymap(lambda s:s.lower() if type(s) == str el
se s)
firstname_df2.head()
```

Out[47]:

	0	1
0	john	boy
1	william	boy
2	james	boy
3	charles	boy
4	george	boy

In [48]:
```
firstnames2 = set(firstname_df2[0].unique())
len(firstnames2)
```

Out[48]: 6782

In [49]:
```
firstnames = firstnames | firstnames2
len(firstnames)
```

Out[49]: 6782

We will use a surname dataset to add surnames to the stopwords library

In [50]:
```
surname_path = 'Data/new-top-surnames.csv'
surname_df = pd.read_csv(surname_path, skiprows=0, skipfooter=0, engine='pytho
n')
surname_df = surname_df.applymap(lambda s:s.lower() if type(s) == str else s)
surname_df.head()
```

Out[50]:

	Unnamed: 0	name	perct2013
0	1	smith	0.007999
1	2	johnson	0.006346
2	3	williams	0.005330
3	4	brown	0.004724
4	5	jones	0.004676

In [51]:
```
surnames = set(surname_df['name'].unique())
len(surnames)
```

Out[51]: 100

We will add calendar words to the stopwords library

In [52]:
```
days=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
months=['January','February','March', 'April','May','June','July','August','Se
ptember','October','November','December']
calendar = days.copy()
calendar.extend(months)
calendar = set([w.lower() for w in calendar])
```

Pre-process data for supervised learning

In [53]:
```
X_train
```

Out[53]:
```
304501    1979-80 Buffalo Sabres NHL 32 1880 74 1 4 2.36...
162313    Diseases Lentils in culture Lentils are mentio...
336845    Railroads , like the Lehigh Valley Railroad , ...
150625    An example of this would be an individual anim...
40240     Both the Matanuska and Susitna Rivers have maj...
                                ...
259178    After the Germans invaded Norway in April 1940...
365838    July 28 - Henry Bennet , 1st Earl of Arlington...
131932    Pancake restaurants are popular family restaur...
146867                                 A cycling domestique
121958    David Boreanaz 's first paid acting appearance...
Name: original_text, Length: 333414, dtype: object
```

In [54]:
```
tokenized_text_train=[]
tokenized_text_test=[]
stopWords = set(stopwords.words('english')) | dale | geo_data | languages | na
tionalities | states | continents | firstnames | surnames | calendar
# This cell will run 4 minutes
import gensim
from nltk.stem.porter import *
def lemmatize_stemming(text):
    stemmer = PorterStemmer()
    #Un-hash next line to use stemming
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))
    #Un-hash next line to NOT use stemming
    #return WordNetLemmatizer().lemmatize(text, pos='v')

# Tokenize and Lemmatize
def preprocess(text):
    result=[]
    for token in gensim.utils.simple_preprocess(text) :
        if token not in stopWords and len(token) > 3:
            #Un-hash next line to use lemmatization/stemming
            result.append(lemmatize_stemming(token))
            #Un-hash next line to NOT use lemmatization/stemming
            #result.append(token)

    return result

tokenized_text_train = [preprocess(text) for text in X_train]
tokenized_text_test=[preprocess(text) for text in X_test]

#for text in tqdm(X_train):
#    tokens_in_text = word_tokenize(text)
#    tokens_in_text = [word for word in tokens_in_text if word.lower() not in
 stopWords]
#    tokenized_text_train.append(tokens_in_text)

#for text in tqdm(X_test):
#    tokens_in_text = word_tokenize(text)
#    tokens_in_text = [word for word in tokens_in_text if word.lower() not in
 stopWords]
#    tokenized_text_test.append(tokens_in_text)
```

In [55]:
```
len(stopWords)
```
Out[55]:
```
36872
```

In [56]:
```
model = Word2Vec(vector_size=100,window=2,min_count=100,seed= RANDOM_SEED,work
ers=4)
model.build_vocab(tokenized_text_train)
model.train(tokenized_text_train,total_examples=model.corpus_count,epochs=mode
l.epochs)
```
Out[56]:
```
(6205482, 9373610)
```

In [57]:
```
word_vectors = model.wv
```

In [58]:
```
#word_vectors.vocab
```

In [59]:
```
word_dict = word_vectors.key_to_index
```

In [60]:
```
words_in_vector = word_vectors.index_to_key
len(words_in_vector)
```
Out[60]:
```
2718
```

Adding word's difficulty to the vector

== Concreteness_ratings_Brysbaert_et_al_BRM.txt ==

This file contains concreteness ratings for 40 thousand English lemma words gathered via Amazon Mechanical Turk. The ratings come from a larger list of 63 thousand words and represent all English words known to 85% of the raters.

The file contains eight columns:

1. The word
2. Whether it is a single word or a two-word expression
3. The mean concreteness rating
4. The standard deviation of the concreteness ratings
5. The number of persons indicating they did not know the word
6. The total number of persons who rated the word
7. Percentage participants who knew the word
8. The SUBTLEX-US frequency count (on a total of 51 million; Brysbaert & New, 2009)
9. The dominant part-of-speech usage

Original source: http://crr.ugent.be/archives/1330 (http://crr.ugent.be/archives/1330)

Brysbaert, M., Warriner, A.B., & Kuperman, V. (2014). Concreteness ratings for 40 thousand generally known English word lemmas. Behavior Research Methods, 46, 904-911. http://crr.ugent.be/papers/Brysbaert_Warriner_Kuperman_BRM_Concreteness_ratings.pdf (http://crr.ugent.be/papers/Brysbaert_Warriner_Kuperman_BRM_Concreteness_ratings.pdf)

In [61]:
```
#Concreteness rating - the higher Conc.M, the easier the word is.
concreteness_path = 'Data/Concreteness_ratings_Brysbaert_et_al_BRM.txt'
concrete_df = pd.read_csv(concreteness_path,delimiter='\t', keep_default_na=Fa
lse)
concreteset=(concrete_df['Word'].values)
```

In [62]:
```
concrete_df.head()
```

Out[62]:

	Word	Bigram	Conc.M	Conc.SD	Unknown	Total	Percent_known	SUBTLEX	Dom_F
0	roadsweeper	0	4.85	0.37	1	27	0.96	0	
1	traindriver	0	4.54	0.71	3	29	0.90	0	
2	tush	0	4.45	1.01	3	25	0.88	66	
3	hairdress	0	3.93	1.28	0	29	1.00	1	
4	pharmaceutics	0	3.77	1.41	4	26	0.85	0	

In [63]:
```
# Stem words in concrete_df to match stemmed words in the vector
concrete_df['stem'] = concrete_df['Word'].apply(lemmatize_stemming)
```

In [64]:
```
concrete_df.head()
```

Out[64]:

	Word	Bigram	Conc.M	Conc.SD	Unknown	Total	Percent_known	SUBTLEX	Dom_F
0	roadsweeper	0	4.85	0.37	1	27	0.96	0	
1	traindriver	0	4.54	0.71	3	29	0.90	0	
2	tush	0	4.45	1.01	3	25	0.88	66	
3	hairdress	0	3.93	1.28	0	29	1.00	1	
4	pharmaceutics	0	3.77	1.41	4	26	0.85	0	

In [65]:
```
np.min(concrete_df['Conc.M'])
```

Out[65]:
```
1.04
```

In [66]:
```
np.max(concrete_df['Conc.M'])
```

Out[66]:
```
5.0
```

Concreteness values range from 1 - 5, we could possible use the inverse value of concreteness to scale it to a 0-1 range and give easier words less weight.

In [67]:
```
#concrete_words = list(concrete_df['Word'].values)
concrete_words = list(concrete_df['stem'].values)
```

In [68]:
```
len(concrete_words)
```

Out[68]:
```
39954
```

In [69]:
```
# How many words are not covered by the concreteness dataset?
concrete_complement = [word for word in words_in_vector if word not in concret
e_words]
len(concrete_complement)
```

Out[69]:
```
255
```

In [70]:
```
# What are these words?
concrete_complement[:10]
```

Out[70]:
```
['largest',
 'ndash',
 'picardi',
 'european',
 'aquitain',
 'disney',
 'britain',
 'alp',
 'oldest',
 'larger']
```

In [71]:
```
# How many words are covered by the concreteness dataset?
concrete_intersect = [word for word in words_in_vector if word in concrete_wor
ds]
len(concrete_intersect)
```

Out[71]:
```
2463
```

In [72]:
```
concrete_intersect[:10]
```

Out[72]:
```
['unit',
 'commun',
 'depart',
 'region',
 'state',
 'includ',
 'call',
 'nation',
 'play',
 'area']
```

In [73]:
```
# Multiply the inverse of the mean concreteness value to the vector
for word in concrete_intersect:
    word_vectors[word] = word_vectors[word] * 1/concrete_df[concrete_df['stem'
]==word]['Conc.M'].values.mean()
```

== AoA_51715_words.csv ==

This file contains "Age of Acquisition" (AoA) estimates for about 51k English words, which refers to the approximate age (in years) when a word was learned. Early words, being more basic, have lower average AoA.

The main columns you will be interested in are "Word" and "AoA_Kup_lem". But the others may be useful too.

The file contains these columns:

Word :: The word in question Alternative.spelling :: if the Word may be spelled frequently in another form Freq_pm :: Freq of the Word in general English (larger -> more common) Dom_PoS_SUBTLEX :: Dominant part of speech in general usage Nletters :: number of letters Nphon :: number of phonemes Nsyll :: number of syllables Lemma_highest_PoS :: the "lemmatized" or "root" form of the word (in the dominant part of speech. e.g. The root form of the verb "abates" is "abate". AoA_Kup :: The AoA from a previous study by Kuperman et al. Perc_known :: Percent of people who knew the word in the Kuperman et al. study AoA_Kup_lem :: Estimated AoA based on Kuperman et al. study lemmatized words. THIS IS THE MAIN COLUMN OF INTEREST. Perc_known_lem :: Estimated percentage of people who would know this form of the word in the Kuperman study. AoA_Bird_lem :: AoA reported in previous study by Bird (2001) AoA_Bristol_lem :: AoA reported in previous study from Bristol Univ. (2006) AoA_Cort_lem :: AoA reported in previous study by Cortese & Khanna (2008) AoA_Schock :: AoA reported in previous study by Schock (2012)

Original source : [http://crr.ugent.be/archives/806 (http://crr.ugent.be/archives/806)](http://crr.ugent.be/archives/806)

In [74]:
```
aoawords_path = 'Data/AoA_51715_words.csv'
AoA = pd.read_csv(aoawords_path,encoding = 'unicode_escape')
AoA = AoA[AoA['Word'].notna()]
AoA_set = set(AoA['Word'].values)
AoA.head(5)
```

Out[74]:

	Word	Alternative.spelling	Freq_pm	Dom_PoS_SUBTLEX	Nletters	Nphon	Nsyll	Lemma_
0	a	a	20415.27	Article	1	1	1	
1	aardvark	aardvark	0.41	Noun	8	7	2	
2	abacus	abacus	0.24	Noun	6	6	3	
3	abacuses	abacuses	0.02	Noun	8	9	4	
4	abalone	abalone	0.51	Verb	7	7	4	

In [75]:
```
# Stem words in AoA to match stemmed words in the vector
AoA['stem'] = AoA['Word'].apply(lemmatize_stemming)
```

In [76]:
```
# We are going to impute all Nan values in AoA_Kup_lem as the max AoA value 2
5, as they appear to be hard words.
AoA['AoA_Kup_lem'].fillna(value=AoA['AoA_Kup_lem'].max(), inplace=True)
```

In [77]:
```
AoA.AoA_Kup_lem.min()
```

Out[77]: 1.58

In [78]:
```
AoA.AoA_Kup_lem.max()
```

Out[78]: 25.0

AoA values range from 0 - 25, which means the smaller the AoA value, the easier the word is. We could possibly use the AoA value to give easier words less weight.

In [79]:
```
aoa_words = list(AoA['stem'].values)
```

In [80]:
```
len(aoa_words)
```

Out[80]: 51714

In [81]:
```
aoa_complement = [word for word in words_in_vector if word not in aoa_words]
aoa_intersect = [word for word in words_in_vector if word in aoa_words]
```

In [82]:
```
len(aoa_complement)
```

Out[82]: 264

In [83]:
```
aoa_complement[:10]
```

Out[83]:
```
['ndash',
 'picardi',
 'european',
 'commonli',
 'aquitain',
 'atlant',
 'lower',
 'disney',
 'throughout',
 'britain']
```

In [84]:
```
len(aoa_intersect)
```

Out[84]:
```
2454
```

In [85]:
```
aoa_intersect[:10]
```

Out[85]:
```
['unit',
 'commun',
 'depart',
 'region',
 'state',
 'includ',
 'call',
 'nation',
 'play',
 'area']
```

In [86]:
```
# How many words are covered in both AoA and concreteness dataset?
len([word for word in aoa_intersect if word in concrete_intersect])
```

Out[86]:
```
2404
```

In [87]:
```
# Multiply the scaled-down mean AoA value to the vector
for word in aoa_intersect:
    word_vectors[word] = word_vectors[word] * AoA[AoA['stem']==word]['AoA_Kup_
lem'].values.mean()/25
```

Generate 100 dense features to reduce dimentionality

In [88]:
```
def generate_dense_features(tokenized_text,word_vectors):
    dense_list=[]
    words=[]
    for _ in tokenized_text:
        words =[word for word in _ if word in word_vectors.key_to_index]

        if len(words) >0:
            dense_list.append(np.mean(word_vectors[words],axis=0))

        else:
            dense_list.append(np.zeros(word_vectors.vector_size))

    return np.array(dense_list)
```

In [89]:
```
X_train_wv = generate_dense_features(tokenized_text_train,word_vectors)
X_test_wv = generate_dense_features(tokenized_text_test,word_vectors)
```

In [90]:
```
X_train_wv.shape
```

Out[90]:
```
(333414, 100)
```

Bag of Words Model

In [91]:
```
# A dummy classifier to compare
def dummy_fun(doc):
    return doc
vectorizer = TfidfVectorizer(analyzer='word',tokenizer=dummy_fun, preprocessor
=dummy_fun, token_pattern=r'(?u)\b\w\w+__\([\w\s]*\)')
X_train_transform = vectorizer.fit_transform(tokenized_text_train)
X_test_transform  = vectorizer.transform(tokenized_text_test)
```

In [92]:
```
model_word = set(word_vectors.index_to_key) #around 6k words in the Word2Vec m
odel
```

In [93]:
```
len(model_word)
```

Out[93]:
```
2718
```

In [94]:
```
len(model_word.intersection(concreteset))
```

Out[94]:
```
1495
```

In [95]:
```
lemmatizer = WordNetLemmatizer()
word_list = []
for word in model_word:
    word_list.append((word,lemmatizer.lemmatize(word.lower())))
df = pd.DataFrame(word_list,columns=['Original','word'])
df = df.merge(AoA,left_on='word',right_on='Word',how='left')
df = df[['Original','word','Perc_known','AoA_Kup_lem']]
word_not_matched = set(df[df['Perc_known'].isnull()].word.values)

for i in range(len(df)):
    if df['word'][i][0] in set(('0','1','2','3','4','5','6','7','8','9')) or l
en(df['word'][i])==1:
        df['AoA_Kup_lem'][i] = 3
mean_value = df['AoA_Kup_lem'].mean()
df['AoA_Kup_lem'].fillna(value=mean_value,inplace=True)
```

In [96]:
```
#df.loc[df['Original']==['troops','weapons']]
df[df['Original'].isin(['troops','weapon'])]
```

Out[96]:

	Original	word	Perc_known	AoA_Kup_lem
165	weapon	weapon	1.0	6.95

In [97]:
```
def generate_perc_known(tokenized_text,df):
    avg_perc_know=None
    perc_know_list=[]
    for _ in tokenized_text:
        words =[word for word in _ if word in word_vectors.key_to_index]

        if len(words) >0:
            avg_perc_know = np.mean(df[df['Original'].isin(words)]['AoA_Kup_le
m'])
            perc_know_list.append(avg_perc_know)
        else:

            perc_know_list.append(0)

    return perc_know_list
```

In [98]:
```
df_train = pd.DataFrame(X_train_wv)
#df_train['year'] = generate_perc_known(tokenized_text_train,df)
```

In [99]:
```
df_test = pd.DataFrame(X_test_wv)
#df_test['year'] = generate_perc_known(tokenized_text_test,df)
```

In [100]:
```
df_test.head()
```

Out[100]:

	0	1	2	3	4	5	6	7	
0	0.100077	-0.035655	-0.001215	-0.018198	-0.099120	-0.056466	0.015681	-0.066352	0.02258
1	0.033296	0.012803	-0.011375	-0.042152	0.057174	-0.062355	-0.130445	0.033912	-0.06848
2	0.007512	-0.013255	0.002345	-0.036628	-0.004910	-0.014912	-0.015589	-0.011666	-0.01488
3	0.011922	-0.037546	-0.014055	0.005939	0.013799	-0.076801	-0.040416	0.020176	0.03308
4	0.038907	0.072647	0.093327	-0.059442	-0.031119	-0.067880	-0.053051	0.039328	-0.02491

5 rows × 100 columns

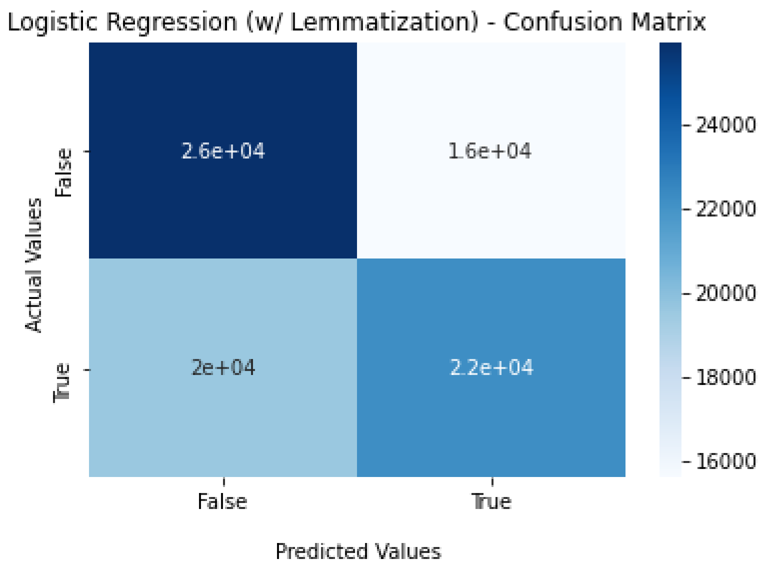
In [101]:
```
lr_wv = LogisticRegression(random_state=RANDOM_SEED,max_iter=1000).fit(df_trai
n,y_train)
```

In [102]:
```
accuracy_score(y_test,lr_wv.predict(df_test))
```

Out[102]: 0.5781966072414041

In [103]:
```
cmatrix_lr_wv=metrics.confusion_matrix(y_test, lr_wv.predict(X_test_wv))

ax = sns.heatmap(cmatrix_lr_wv, annot=True, cmap='Blues')

ax.set_title('Logistic Regression (w/ Lemmatization) - Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
```

Out[103]:
```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



2. Supervised Learning

Random Classifier

In [104]:
```
dummy_bow = DummyClassifier(strategy='uniform',random_state=RANDOM_SEED).fit(X
_train_transform,y_train)
```

In [105]:
```
accuracy_score(y_test, dummy_bow.predict(X_test_transform))
```

Out[105]:
```
0.5011277203253593
```

In [106]:
```
dummy_wv = DummyClassifier(strategy='uniform',random_state=RANDOM_SEED).fit(X_
train_wv,y_train)
```

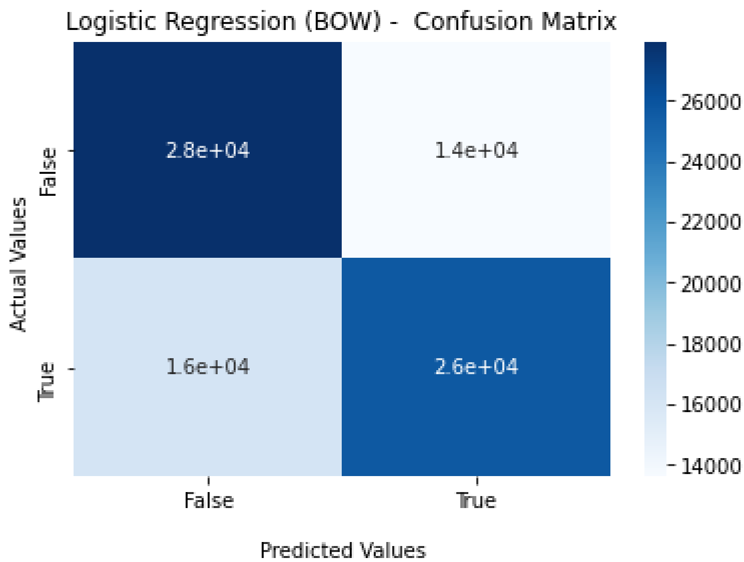
In [107]:
```
accuracy_score(y_test,dummy_wv.predict(X_test_wv))
```
Out[107]: 0.5011277203253593

Logistic Regression Classifier

In [108]:
```
lr_bow = LogisticRegression(random_state=RANDOM_SEED,max_iter=1000).fit(X_trai
n_transform,y_train)
```

In [109]:
```
accuracy_score(y_test,lr_bow.predict(X_test_transform))
```
Out[109]: 0.6433644456174868

In [110]:
```
cmatrix_lr_bow=metrics.confusion_matrix(y_test, lr_bow.predict(X_test_transfor
m))

ax = sns.heatmap(cmatrix_lr_bow, annot=True, cmap='Blues')

ax.set_title('Logistic Regression (BOW) -  Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```
Out[110]: [Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]

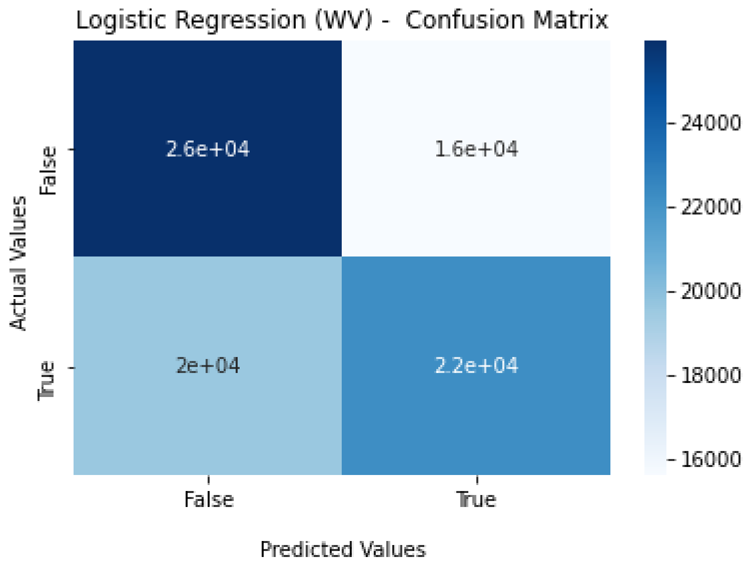


In [111]:
```
lr_wv = LogisticRegression(random_state=RANDOM_SEED,max_iter=1000).fit(X_train
_wv,y_train)
```

In [112]:
```
accuracy_score(y_test,lr_wv.predict(X_test_wv))
```
Out[112]: 0.5781966072414041

In [113]:
```
cmatrix_lr_wv=metrics.confusion_matrix(y_test, lr_wv.predict(X_test_wv))

ax = sns.heatmap(cmatrix_lr_wv, annot=True, cmap='Blues')

ax.set_title('Logistic Regression (WV) -  Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

Out[113]: [Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]



Random Forest Classifier

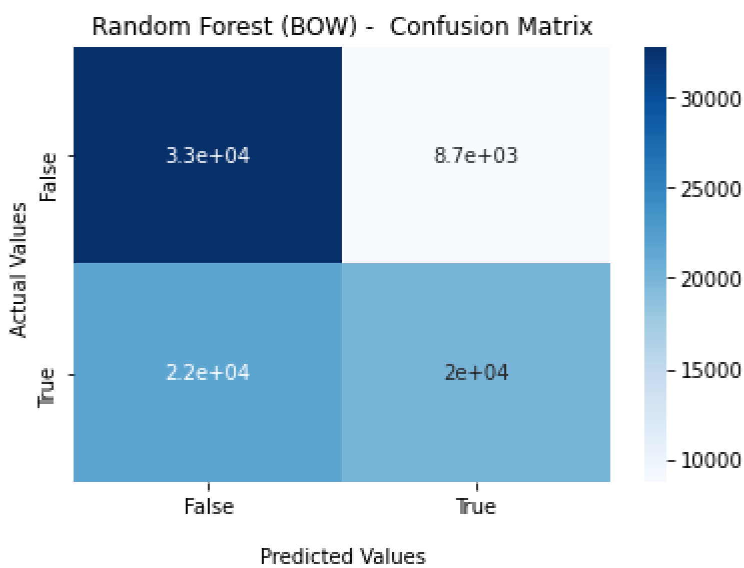
In [114]:
```
rf_bow = RandomForestClassifier(n_estimators=500,max_depth=5,random_state=RAND
OM_SEED).fit(X_train_transform,y_train)
```

In [115]:
```
accuracy_score(y_test,rf_bow.predict(X_test_transform))
```

Out[115]: 0.6336948436787677

In [116]:
```
cmatrix_rf_bow=metrics.confusion_matrix(y_test, rf_bow.predict(X_test_transform))

ax = sns.heatmap(cmatrix_rf_bow, annot=True, cmap='Blues')

ax.set_title('Random Forest (BOW) -  Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

Out[116]:
```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



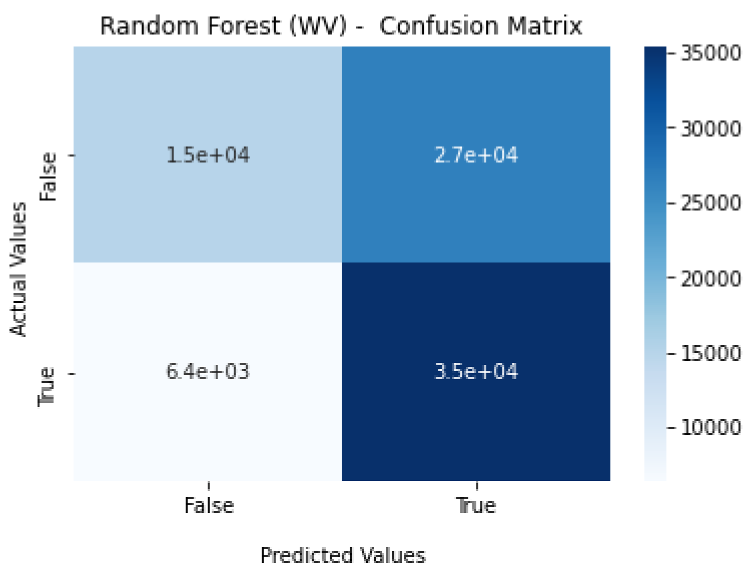
In [117]:
```
rf_wv = RandomForestClassifier(n_estimators=100,max_depth=5,random_state=RANDOM_SEED).fit(X_train_wv,y_train)
```

In [118]:
```
accuracy_score(y_test,rf_wv.predict(X_test_wv))
```

Out[118]:
```
0.605405859346882
```

In [119]:
```
cmatrix_rf_wv=metrics.confusion_matrix(y_test, rf_wv.predict(X_test_wv))

ax = sns.heatmap(cmatrix_rf_wv, annot=True, cmap='Blues')

ax.set_title('Random Forest (WV) -  Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

Out[119]: [Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]



XGBoost Classifier

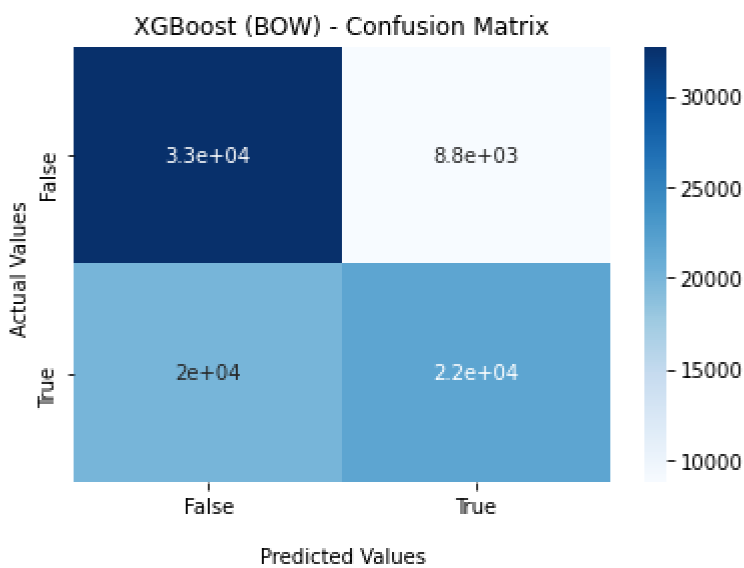
In [120]:
```
xgb_bow = XGBClassifier(random_state=RANDOM_SEED).fit(X_train_transform,y_trai
n)
```

In [121]:
```
accuracy_score(y_test,xgb_bow.predict(X_test_transform))
```

Out[121]: 0.6543297262278955

In [122]:
```
# plot
cmatrix_xgb_bow=metrics.confusion_matrix(y_test, xgb_bow.predict(X_test_transf
orm))

ax = sns.heatmap(cmatrix_xgb_bow, annot=True, cmap='Blues')

ax.set_title('XGBoost (BOW) - Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

Out[122]:
```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



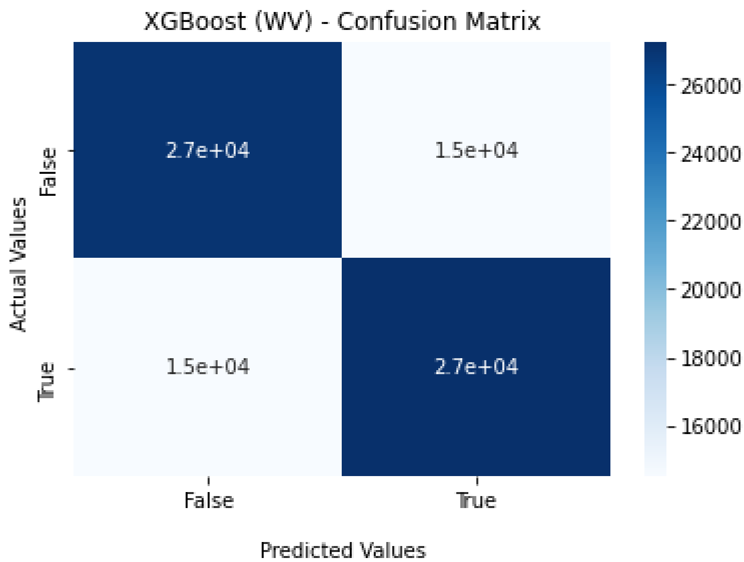
In [123]:
```
xgb_wv = RandomForestClassifier(random_state=RANDOM_SEED).fit(X_train_wv,y_tra
in)
```

In [124]:
```
accuracy_score(y_test,xgb_wv.predict(X_test_wv))
```

Out[124]:
```
0.6513184730186914
```

In [125]:
```
# plot
cmatrix_xgb_wv=metrics.confusion_matrix(y_test, xgb_wv.predict(X_test_wv))

ax = sns.heatmap(cmatrix_xgb_wv, annot=True, cmap='Blues')

ax.set_title('XGBoost (WV) - Confusion Matrix');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
```

Out[125]:
```
[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]
```



3. Unsupervised Learning

K-MEANS

In [126]:
```
kmeans = KMeans(n_clusters=2,random_state=RANDOM_SEED).fit(X_train_transform)
```

In [127]:
```
cluster_df = pd.DataFrame({'cluster':kmeans.labels_,'y_label':y_train,'text':X
_train})
```

In [128]:
```
cluster_df[cluster_df['cluster']==0].head()
```

Out[128]:

	cluster	y_label	text
360394	0	0	It is found in the region Picardie in the Aisn...
239614	0	0	It is found in the region Picardie in the Aisn...
177781	0	1	Vimy -LRB- ; Ã‹ viÃ‹ mi -RRB- is a commune in ...
180579	0	1	Nielles-l Ãƒ s-Ardres is a commune in the Pas-...
100538	0	1	Muret-et-Croutes is a commune in the Aisne de...

In [129]:
```
cluster_df[cluster_df['cluster']==1].head()
```

Out[129]:

	cluster	y_label	text
304501	1	0	1979-80 Buffalo Sabres NHL 32 1880 74 1 4 2.36...
162313	1	1	Diseases Lentils in culture Lentils are mentio...
336845	1	0	Railroads , like the Lehigh Valley Railroad , ...
150625	1	1	An example of this would be an individual anim...
40240	1	1	Both the Matanuska and Susitna Rivers have maj...

NMF

In [130]:
```
from sklearn.decomposition import NMF
```

In [131]:
```
nmf = NMF(n_components=5,random_state=RANDOM_SEED)
W = nmf.fit_transform(X_train_transform)
H = nmf.components_
```

In [132]:
```
W_test = nmf.transform(X_test_transform)
```

In [133]:
```
words = np.array(vectorizer.get_feature_names())
for i, topic in enumerate(H):
     print("Topic {}: {}".format(i + 1, ",".join([str(x) for x in words[topic.
argsort()[-5:]]])))
```

```
Topic 1: bass,picardi,commun,region,depart
Topic 2: largest,releas,presid,state,unit
Topic 3: call,origin,usual,commonli,refer
Topic 4: current,hockey,leagu,play,nation
Topic 5: largest,locat,censu,area,popul
```

In [134]:
```
lr_tm = LogisticRegression(random_state=RANDOM_SEED,max_iter=1000).fit(W,y_tra
in)
```

In [135]:
```
accuracy_score(y_test,lr_tm.predict(W_test))
```
Out[135]: 0.53163615423375

T-SNE

In [136]:
```
from sklearn.manifold import TSNE
```
In [137]:
```
tsne = TSNE(n_components = 2, init = 'random', random_state = RANDOM_SEED, per
plexity = 50)
```
In [138]:
```
X_train_wv_embedded= tsne.fit_transform(X_train_wv[164707:168707])
```
In [139]:
```
t_df = pd.DataFrame(X_train_wv_embedded,columns=['dimension0','dimension1'])
t_df['label'] = y_train.values[164707:168707]
t_df1=t_df[t_df['label']==1]
t_df0=t_df[t_df['label']==0]
```
In [140]:
```
import matplotlib.pyplot as plt
plt.scatter(t_df1['dimension0'],t_df1['dimension1'],color='purple')
plt.scatter(t_df0['dimension0'],t_df0['dimension1'],color='orange');
```