In [1]:
```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline
```

In [2]:
```
corpus = pd.read_csv(r"./Sample Data 1.csv")
```

In [3]:
```
# Remove blank rows
corpus['Description'].dropna(inplace=True)
# Change all text to lower case
corpus['Description'] = [entry.lower() for entry in corpus['Description']]
# Tokenization
corpus['Description']= [word_tokenize(entry) for entry in corpus['Description']]
corpus.head()
```

Out[3]:

	Genre	Description
0	Places & Travel	[the, budget-minded, traveler, podcast, is, yo...
1	Food	[the, official, podcast, of, bourbon, ., featu...
2	Social Sciences	[..., our, most, human, experiences, ...]
3	News & Politics	[a, show, about, politics, with, no, agenda, ,...
4	Careers	[made, for, profit, is, a, podcast, where, we,...

In [4]:
```
# Remove stop words, non-numeric and perfom Word Stemming/Lemmenting.
tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
for index,entry in enumerate(corpus['Description']):
    Final_words = []
    word_Lemmatized = WordNetLemmatizer()
    for word, tag in pos_tag(entry):
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
    corpus.loc[index,'description_final'] = str(Final_words)
```

In [5]:
```
corpus.head()
```

Out[5]:

	Genre	Description	description_final
0	Places & Travel	[the, budget-minded, traveler, podcast, is, yo...	['traveler', 'podcast', 'source', 'everyday', ...
1	Food	[the, official, podcast, of, bourbon, ., featu...	['official', 'podcast', 'bourbon', 'feature', ...
2	Social Sciences	[..., our, most, human, experiences, ...]	['human', 'experience']
3	News & Politics	[a, show, about, politics, with, no, agenda, ,...	['show', 'politics', 'agenda', 'adam', 'curry'...
4	Careers	[made, for, profit, is, a, podcast, where, we,...	['make', 'profit', 'podcast', 'talk', 'busines...

In [6]:
```
Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(corpus['description_final'],corpus['Genre'],test_size=0.3)
```

In [7]:
```
#OLD
#Encoder = LabelEncoder()
#Train_Y = Encoder.fit_transform(Train_Y)
#Test_Y = Encoder.fit_transform(Test_Y)
```

In [8]:
```
#OLD
#Tfidf_vect = TfidfVectorizer(max_features=5000)
#Tfidf_vect.fit(corpus['description_final'])
#Train_X_Tfidf = Tfidf_vect.transform(Train_X)
#Test_X_Tfidf = Tfidf_vect.transform(Test_X)
```

In [9]:
```
#OLD
# fit the training dataset on the NB classifier
#Naive = naive_bayes.MultinomialNB()
#Naive.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
#predictions_NB = Naive.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
#print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB, Test_Y)*100)
```

In [21]:
```
# Build the model
NBmodel = make_pipeline(TfidfVectorizer(), naive_bayes.MultinomialNB())
SVMmodel = make_pipeline(TfidfVectorizer(), svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto'))
# Train the model using the training data
NBmodel.fit(Train_X, Train_Y)
SVMmodel.fit(Train_X, Train_Y)
# Predict the categories of the test data
NB_predicted_categories = NBmodel.predict(Test_X)
SVM_predicted_categories = SVMmodel.predict(Test_X)
```

In [22]:
```
print("Naive Bayes Accuracy Score -> ",accuracy_score(NB_predicted_categories, Test_Y)*100)
print("SVM Accuracy Score -> ",accuracy_score(SVM_predicted_categories, Test_Y)*100)
```

```
Naive Bayes Accuracy Score ->  45.42566709021601
SVM Accuracy Score ->  51.207115628970776
```

In [12]:
```
corpus2 = pd.read_csv(r"./Sample Data 2.csv")
```

In [14]:
```
# Remove blank rows
corpus2['Description'].dropna(inplace=True)
# Change all text to lower case
corpus2['Description'] = [entry.lower() for entry in corpus2['Description']]
# Tokenization
corpus2['Description']= [word_tokenize(entry) for entry in corpus2['Description']]
corpus2.head()
```

Out[14]:

	Genre	Description
0	Arts	[bestselling, author, elizabeth, gilbert, retu...
1	Video Games	[the, leaders, in, gaming, news, hand-pick, th...
2	Business News	[the, tech, m, &, a, podcast, pulls, from, the...
3	Management & Marketing	[a, podcast, about, entrepreneurs, who, quit, ...
4	Medicine	[a, podcast, about, how, doctors, think, ., pr...

In [15]:
```
# Remove stop words, non-numeric and perfom Word Stemming/Lemmenting.
tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
for index,entry in enumerate(corpus2['Description']):
    Final_words = []
    word_Lemmatized = WordNetLemmatizer()
    for word, tag in pos_tag(entry):
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
    corpus2.loc[index,'description_final'] = str(Final_words)
```

In [18]:
```
predicted_categories = model.predict(corpus2['description_final'])

print(predicted_categories)
print("Naive Bayes Accuracy Score -> ",accuracy_score(predicted_categories, corpus2['Genre'])*100)
```

```
['Careers' 'Video Games' 'Gadgets' ... 'Careers' 'Video Games' 'Sexuality']
Naive Bayes Accuracy Score ->  44.7361840460115
```

In [ ]:
```

```