Gensim Encoding

Aim: To perform gensim encoding on the given dataset and train with several machine learning modules.

Description: Gensim is an open-source library for unsupervised topic modeling, document indexing, retrieval by similarity, and other natural language processing functionalities, using modern statistical machine learning. Gensim is implemented in Python and Cython for performance

```
import pandas as pd
import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df = pd.read_csv("/content/drive/MyDrive/NLP/revpre.csv")
df.head()
```

	Unnamed: 0.1	Unnamed: 0	Review	Rating	label
0	0	0	'absolutely wonderful silky sexy comfortable '	4	1
1	1	1	'love dress sooo pretty happened find store im...	5	1
2	2	2	'high hopes dress really wanted work initially...	3	0
3	3	3	'love love love jumpsuit fun flirty fabulous e...	5	1
4	4	4	'this shirt flattering due adjustable front ti...	5	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Unnamed: 0.1             23486 non-null  int64 
 1   Unnamed: 0               23486 non-null  int64 
 2   Review                   23486 non-null  object
 3   Rating                   23486 non-null  int64 
 4   label                    23486 non-null  int64 
 5   Positive Feedback Count  23486 non-null  int64 
dtypes: int64(5), object(1)
memory usage: 1.1+ MB
```

```
df['Review']= df['Review'].astype(str)
```

```
df
```

	Unnamed: 0.1	Unnamed: 0	Review	Rating
0	0	0	'absolutely wonderful silky sexy comfortable '	4
1	1	1	'love dress sooo pretty happened find store im...	5
2	2	2	'high hopes dress really wanted work initially...	3
3	3	3	'love love love jumpsuit fun flirty fabulous e...	5
4	4	4	'this shirt flattering due adjustable front ti...	5
...	...	...	...	...
23481	23481	23481	'happy snag dress great price easy slip flatte...	5
23482	23482	23482	'it reminds maternity clothes soft stretchy sh...	3
23483	23483	23483	'this fit well top see never would worked im g...	3
23484	23484	23484	'bought dress wedding summer cute unfortunatel...	3
23485	23485	23485	'this dress lovely platinum feminine fits perf...	5

23486 rows × 6 columns

```
# Train a Word2Vec model on the preprocessed text data
from gensim.models import Word2Vec
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import numpy as np


num_features=100

# Convert the cleaned & tokenized movie review text data into a list of lists o
sentences = []
for review in df.Review.values:
    sentences.append(review.split())
# Initialize & train the Word2Vec model

model=Word2Vec(sentences,  vector_size=num_features)
# For each movie review, convert the sequence of words into a fixed-length vect
#use a pre-trained model: convert words to vectors and averages all the vectors

def make_feature_vec(words, model,num_features):
    # Function to average all of the word vectors in a given paragraph
    feature_vec = np.zeros((num_features,), dtype="float32")
```

```
    nwords = 0
    for word in words:
        if word in model.wv.key_to_index:
            feature_vec = np.add(feature_vec, model.wv.get_vector(word))
            nwords += 1
    if nwords > 0:
        feature_vec = np.divide(feature_vec, nwords)
    return feature_vec

def get_avg_feature_vecs(reviews, model, num_features):
    # Function to generate vectors for all movie reviews in a dataset
    counter = 0
    review_feature_vecs = np.zeros((len(reviews), num_features), dtype="float32
    for review in reviews:
        review_feature_vecs[counter] = make_feature_vec(review, model, num_feat
        counter += 1
    return review_feature_vecs

# Convert the training and test data into fixed-length feature vectors
data_vecs = get_avg_feature_vecs(sentences, model, num_features)
```

```
data_vecs
```

```
array([[ 0.28619358,  0.57311296,  0.501779  , ...,  0.2814637 ,
         0.12866905,  0.02506588],
       [ 0.12446889,  0.02609701, -0.8904711 , ..., -0.16542947,
         0.13211656, -0.5087817 ],
       [-0.0780727 ,  0.04488135, -0.40277183, ..., -0.17080599,
         0.03903122, -0.10680101],
       ...,
       [ 0.50002384, -0.1525399 , -0.43576786, ..., -0.34090793,
         0.328015  , -0.49253386],
       [ 0.01787531,  0.10204065, -0.29116142, ..., -0.17726123,
         0.04697115, -0.17628835],
       [ 0.40875685,  0.69951516,  0.2514583 , ...,  0.216685  ,
        -0.12936245, -0.28744552]], dtype=float32)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_vecs, df.label, test_s
```

```
X_train_2d = np.stack(X_train)
X_test_2d = np.stack(X_test)

from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
scaled_train_embed = scaler.fit_transform(X_train_2d)
```

```
scaled_test_embed = scaler.transform(X_test_2d)


clf = GaussianNB()
clf.fit(scaled_train_embed, y_train)
from sklearn.metrics import classification_report

y_pred = clf.predict(scaled_test_embed)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.36      0.83      0.50       810
           1       0.95      0.69      0.80      3888

    accuracy                           0.72      4698
   macro avg       0.65      0.76      0.65      4698
weighted avg       0.85      0.72      0.75      4698
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(scaled_train_embed, y_train)
from sklearn.metrics import classification_report

y_pred = clf.predict(scaled_test_embed)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.68      0.39      0.49       810
           1       0.88      0.96      0.92      3888

    accuracy                           0.86      4698
   macro avg       0.78      0.67      0.71      4698
weighted avg       0.85      0.86      0.85      4698
```

```
from sklearn.svm import SVC
classifier = SVC(kernel='poly', random_state=0)
classifier.fit(scaled_train_embed, y_train)
y_pred = classifier.predict(scaled_test_embed)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.65      0.54      0.59       810
           1       0.91      0.94      0.92      3888

    accuracy                           0.87      4698
   macro avg       0.78      0.74      0.76      4698
```

```
weighted avg       0.86      0.87      0.87      4698
```

```
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(scaled_train_embed, y_train)
y_pred = classifier.predict(scaled_test_embed)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.44      0.54       810
           1       0.89      0.96      0.93      3888

    accuracy                           0.87      4698
   macro avg       0.80      0.70      0.74      4698
weighted avg       0.86      0.87      0.86      4698
```

```
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(scaled_train_embed, y_train)
y_pred = classifier.predict(scaled_test_embed)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.43      0.54       810
           1       0.89      0.97      0.93      3888

    accuracy                           0.87      4698
   macro avg       0.81      0.70      0.73      4698
weighted avg       0.86      0.87      0.86      4698
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(scaled_train_embed, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred = knn.predict(scaled_test_embed)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.62      0.33      0.43       810
           1       0.87      0.96      0.91      3888

    accuracy                           0.85      4698
   macro avg       0.75      0.64      0.67      4698
weighted avg       0.83      0.85      0.83      4698
```

Observations: Gensim encoding gave higher accuracy when compared to other pretrained models like spacy and fasttext. Random forest classifier gave 0.84 accuracy with preprocessed data using gensim and 0.52 with preprocessing done by nltk. Support vector classifier with linear kernel gave maximum accuracy of 0.86. Accuracy with KNN is 0.85.