Name: Onasvee Banarse

Class: BE Computer-1

Roll No: 09

Problem Statement: Perform bag-of-words approach (count occurrence, normalized count occurrence), TF-IDF on data. Create embeddings using Word2Vec. Dataset to be used: https://www.kaggle.com/datasets/CooperUnion/cardataset

In [3]:
```
import collections
import pandas as pd
import numpy as np
import warnings

warnings.filterwarnings(action = 'ignore')

import gensim
from gensim.models import Word2Vec
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from nltk.tokenize import sent_tokenize, word_tokenize
```

In [4]:
```
train_raw_df = pd.read_csv('cardataset.csv')
```

In [5]:
```
train_raw_df
```

Out[5]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Nu
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	
...	...	...	...	...	...	...	...	...	
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	
11910	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	
11911	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	
11912	Acura	ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	all wheel drive	
11913	Lincoln	Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	front wheel drive	

11914 rows × 16 columns

In [6]:

```
train_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               11914 non-null  object 
 1   Model              11914 non-null  object 
 2   Year               11914 non-null  int64  
 3   Engine Fuel Type   11911 non-null  object 
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object 
 7   Driven_Wheels      11914 non-null  object 
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object 
 10  Vehicle Size       11914 non-null  object 
 11  Vehicle Style      11914 non-null  object 
 12  highway MPG        11914 non-null  int64  
 13  city mpg           11914 non-null  int64  
 14  Popularity         11914 non-null  int64  
 15  MSRP               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

In [7]:
```
train_raw_df.dropna(inplace=True)
train_raw_df.reset_index(drop=True, inplace=True)
```

In [8]:
```
train_raw_df["train_text"] = train_raw_df[['Market Category', 'Vehicle Size',
                                            'Vehicle Style']].apply(' '.join, axis=1
```

In [9]:
```
x_train = train_raw_df["train_text"]
y_train = train_raw_df.MSRP
```

In [10]:
```
doc = " ".join(x_train)
```

In [12]:
```
count_vec = CountVectorizer()
count_occurs = count_vec.fit_transform([doc])
count_occur_df = pd.DataFrame((count, word) for word, count in zip(count_occurs.toa
count_occur_df.columns = ['Word', 'Count']
count_occur_df.sort_values('Count', ascending=False, inplace=True)
count_occur_df.head()
```

Out[12]:

	Word	Count
23	performance	3456
19	luxury	3279
20	midsize	3187
4	compact	3039
1	4dr	2771

Normalized Count Occurrence

If you think that high frequency may dominate the result and causing model bias. Normalization can be apply to pipeline easily.

In [14]:

```
norm_count_vec = TfidfVectorizer(use_idf=False, norm='l2')
norm_count_occurs = norm_count_vec.fit_transform([doc])
norm_count_occur_df = pd.DataFrame((count, word) for word, count in zip(
    norm_count_occurs.toarray().tolist()[0], norm_count_vec.get_feature_names()))
norm_count_occur_df.columns = ['Word', 'Count']
norm_count_occur_df.sort_values('Count', ascending=False, inplace=True)
norm_count_occur_df.head()
```

Out[14]:

	Word	Count
23	performance	0.386670
19	luxury	0.366867
20	midsize	0.356573
4	compact	0.340015
1	4dr	0.310030

TF-IDF

TF-IDF take another approach which is believe that high frequency may not able to provide much information gain. In another word, rare words contribute more weights to the model.

Word importance will be increased if the number of occurrence within same document (i.e. training record). On the other hand, it will be decreased if it occurs in corpus (i.e. other training records).

In [16]:

```
tfidf_vec = TfidfVectorizer()
tfidf_count_occurs = tfidf_vec.fit_transform([doc])
tfidf_count_occur_df = pd.DataFrame((count, word) for word, count in zip(
    tfidf_count_occurs.toarray().tolist()[0], tfidf_vec.get_feature_names()))
tfidf_count_occur_df.columns = ['Word', 'Count']
tfidf_count_occur_df.sort_values('Count', ascending=False, inplace=True)
tfidf_count_occur_df.head()
```

Out[16]:

	Word	Count
23	performance	0.386670
19	luxury	0.366867
20	midsize	0.356573
4	compact	0.340015
1	4dr	0.310030

Word2Vec

Word Embedding is a language modeling technique used for mapping words to vectors of real numbers. It represents words or phrases in vector space with several dimensions. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc.

Word2Vec consists of models for generating word embedding. These models are shallow two-layer neural networks having one input layer, one hidden layer, and one output layer.

Word2Vec utilizes two architectures :

1. CBOW (Continuous Bag of Words): CBOW model predicts the current word given context words within a specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the number of dimensions in which we want to represent the current word present at the output layer.

2. Skip Gram : Skip gram predicts the surrounding context words within specific window given current word. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

In [17]:

```
data = []

# iterate through each sentence in the file
for i in x_train:
    temp = []

    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())

    data.append(temp)
```

In [18]:

```
# Create CBOW model
model1 = gensim.models.Word2Vec(data, min_count = 1, vector_size = 100,
                                window = 5)

# Print results
print("Cosine similarity between 'Luxury' and 'Performance' - CBOW : ",
        model1.wv.similarity('luxury', 'performance'))

print("Cosine similarity between 'Crossover' and 'Midsize' - CBOW : ",
        model1.wv.similarity('crossover', 'midsize'))
```

```
Cosine similarity between 'Luxury' and 'Performance' - CBOW :  0.93633825
Cosine similarity between 'Crossover' and 'Midsize' - CBOW :  0.9000161
```

In [19]:

```
# Create Skip Gram model
model2 = gensim.models.Word2Vec(data, min_count = 1, vector_size = 100,
                                window = 5, sg = 1)

# Print results
print("Cosine similarity between 'Luxury' and 'Performance' - Skip Gram : ",
        model2.wv.similarity('luxury', 'performance'))

print("Cosine similarity between 'Crossover' and 'Midsize' - Skip Gram : ",
        model2.wv.similarity('crossover', 'midsize'))
```

```
Cosine similarity between 'Luxury' and 'Performance' - Skip Gram :  0.93042964
Cosine similarity between 'Crossover' and 'Midsize' - Skip Gram :  0.87030834
```