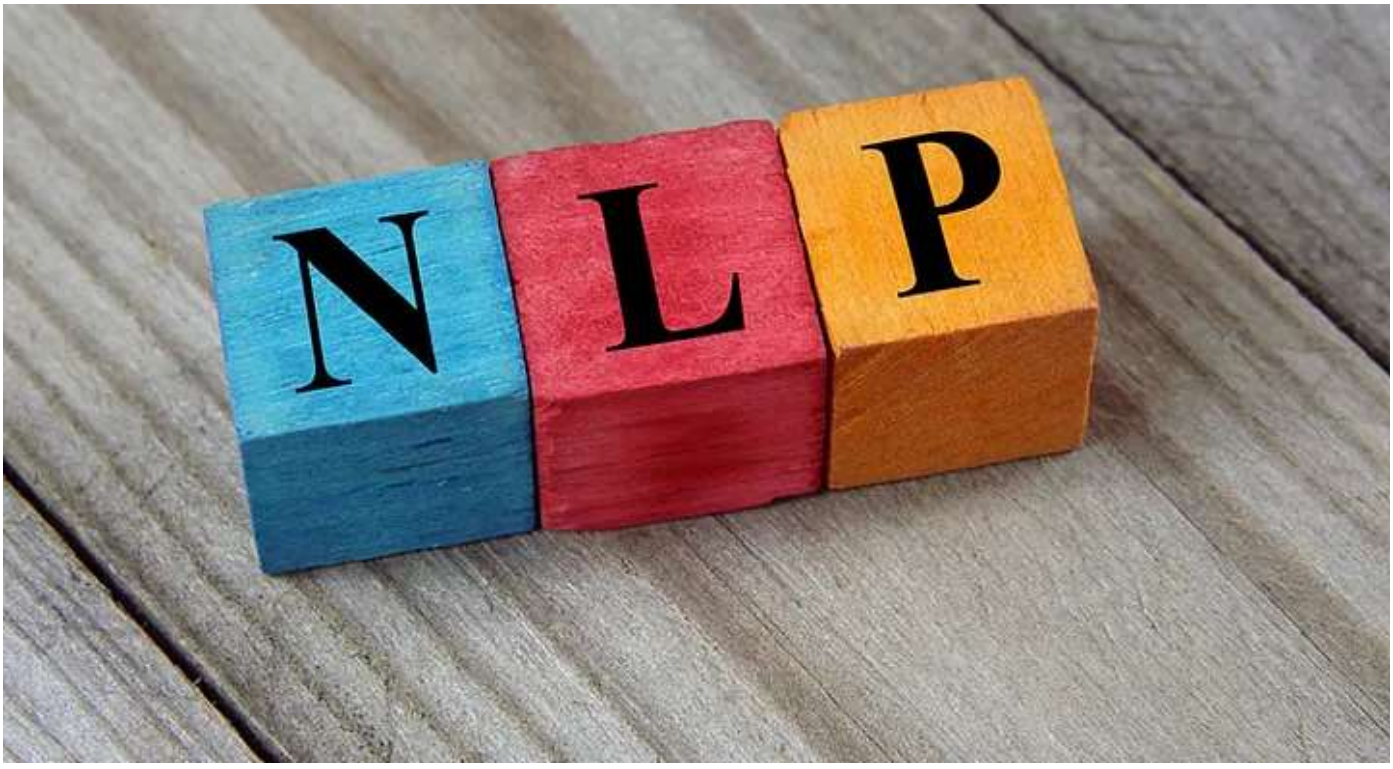
Zhi Li Follow

May 30, 2019 · 8 min read · ✨ · Listen

Save

A Beginner's Guide to Word Embedding with Gensim Word2Vec Model



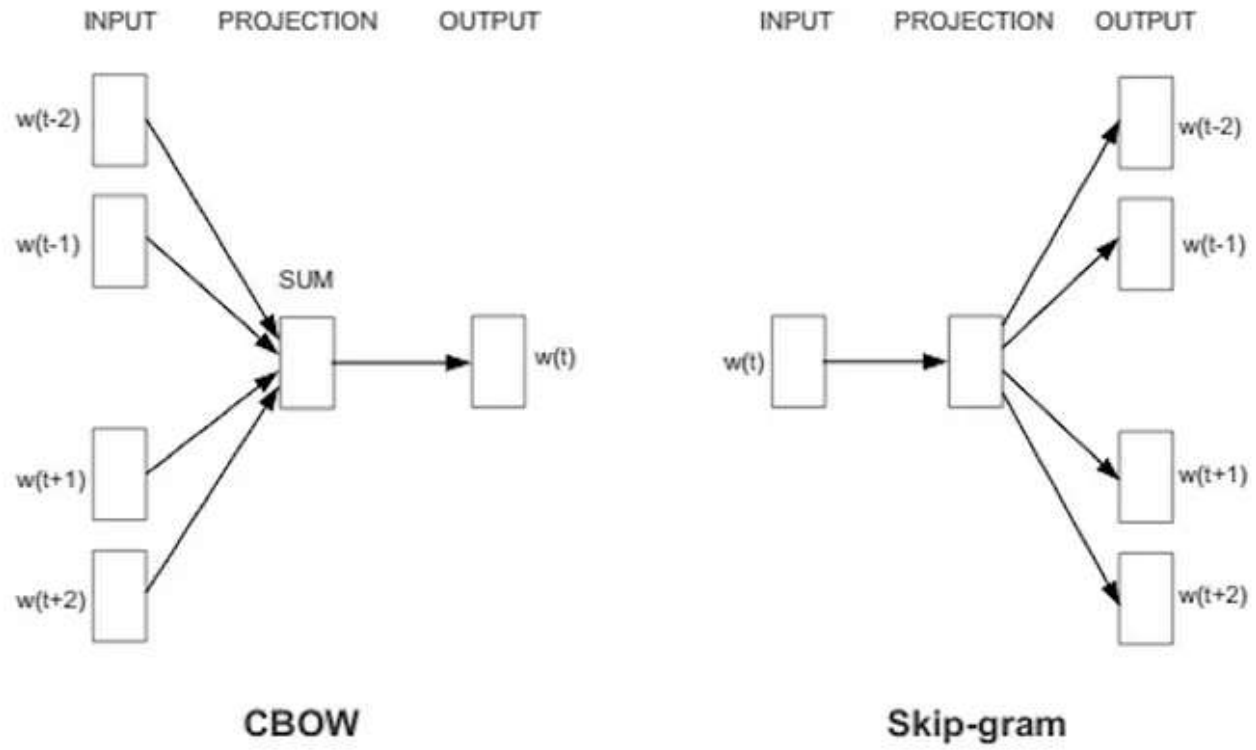
> *Word embedding is one of the most important techniques in natural language processing(NLP), where words are mapped to vectors of real numbers. Word embedding is*

> *capable of capturing the meaning of a word in a document, semantic and syntactic similarity, relation with other words. It also has been widely used for recommender systems and text classification. This tutorial will show a brief introduction of genism word2vec model with an example of generating word embedding for the vehicle make model.*

Table of Contents

1. Introduction of Word2vec

Word2vec is one of the most popular technique to learn word embeddings using a two-layer neural network. Its input is a text corpus and its output is a set of vectors. Word embedding via word2vec can make natural language computer-readable, then further implementation of mathematical operations on words can be used to detect their similarities. A well-trained set of word vectors will place similar words close to each other in that space. For instance, the words women, men, and human might cluster in one corner, while yellow, red and blue cluster together in another.



There are two main training algorithms for word2vec, one is the continuous bag of words(CBOW), another is called skip-gram. The major difference between these two methods is that CBOW is using context to predict a target word while skip-gram is using a word to predict a target context. Generally, the skip-gram method can have a better performance compared with CBOW method, for it can capture two semantics for a single word. For instance, it will have two vector representations for Apple, one for the company and another for the fruit. For more details about the word2vec algorithm, please check here.

2. Gensim Python Library Introduction

Gensim is an open source python library for natural language processing and it was developed and is maintained by the Czech natural language processing researcher Radim Řehůřek. Gensim library will enable us to develop word embeddings by training our own word2vec models on a custom corpus either with CBOW of skip-grams algorithms.

At first, we need to install the genism package. Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 2.7+ and NumPy. Gensim depends on the following software:

- **Python** >= 2.7 (tested with versions 2.7, 3.5 and 3.6)
- **NumPy** >= 1.11.3
- **SciPy** >= 0.18.1
- **Six** >= 1.5.0
- **smart_open** >= 1.2.1

There are two ways for installation. We could run the following code in our terminal to install genism package.

```
pip install --upgrade gensim
```

Or, alternatively for *Conda* environments:

```
conda install -c conda-forge gensim
```

3. Implementation of word Embedding with Gensim Word2Vec Model

This vehicle dataset includes features such as make, model, year, engine, and other properties of the car. We will use these features to generate the word embeddings for each make model and then compare the similarities between different make model. The full python tutorial can be found here.

```
>>> df = pd.read_csv('data.csv')
>>> df.head()
```


	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transm'			Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity
and output; double click to hide output															
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916

3.1 Data Preprocessing:

Since the purpose of this tutorial is to learn how to generate word embeddings using genism library, we will not do the EDA and feature selection for the word2vec model for the sake of simplicity.

Genism word2vec requires that a format of 'list of lists' for training where every document is contained in a list and every list contains lists of tokens of that document. At first, we need to generate a format of 'list of lists' for training the make model word embedding. To be more specific, each make model is contained in a list and every list contains lists of features of that make model.

To achieve this, we need to do the following things :

a. Create a new column for Make Model

```
>>> df['Maker_Model']= df['Make']+ " " + df['Model']
```

b. Generate a format of ' list of lists' for each Make Model with the following features: Engine Fuel Type, Transmission Type, Driven_Wheels, Market Category, Vehicle Size, Vehicle Style.

```
# Select features from original dataset to form a new dataframe
>>> df1 = df[['Engine Fuel Type','Transmission
Type','Driven_Wheels','Market Category','Vehicle Size', 'Vehicle
Style', 'Maker_Model']]

# For each row, combine all the columns into one column
>>> df2 = df1.apply(lambda x: ','.join(x.astype(str)), axis=1)

# Store them in a pandas dataframe
>>> df_clean = pd.DataFrame({'clean': df2})

# Create the list of list format of the custom corpus for gensim
modeling
>>> sent = [row.split(',') for row in df_clean['clean']]

# show the example of list of list format of the custom corpus for
gensim modeling
>>> sent[:2]
[['premium unleaded (required)',
  'MANUAL',
  'rear wheel drive',
  'Factory Tuner',
  'Luxury',
  'High-Performance',
  'Compact',
  'Coupe',
  'BMW 1 Series M'],
```

```
 ['premium unleaded (required)',
  'MANUAL',
  'rear wheel drive',
  'Luxury',
  'Performance',
  'Compact',
  'Convertible',
  'BMW 1 Series']]
```

3.2. Genism word2vec Model Training

We can train the genism word2vec model with our own custom corpus as following:

```
>>> model = Word2Vec(sent, min_count=1,size= 50,workers=3, window =3,
sg = 1)
```

Let's try to understand the hyperparameters of this model.

size: The number of dimensions of the embeddings and the default is 100.

window: The maximum distance between a target word and words around the target word. The default window is 5.

min_count: The minimum count of words to consider when training the model; words with occurrence less than this count will be ignored. The default for min_count is 5.

workers: The number of partitions during training and the default workers is 3.

sg: The training algorithm, either CBOW(0) or skip gram(1). The default training algorithm is CBOW.

After training the word2vec model, we can obtain the word embedding directly from the training model as following.

```
>>> model['Toyota Camry']

array([-0.11884457,  0.03035539, -0.0248678 , -0.06297892,
-0.01703234,
       -0.03832747, -0.0825972 , -0.00268112, -0.09192555,
-0.08458661,
```

```
       -0.07199778,  0.05235871,  0.21303181,  0.15767808, -0.1883737
,
        0.01938575, -0.24431638,  0.04261152,  0.11865819,
0.09881561,
       -0.04580643, -0.08342388, -0.01355413, -0.07892415,
-0.08467747,
       -0.0040625 ,  0.16796461,  0.14578669,  0.04187112,
-0.01436194,
       -0.25554284,  0.25494182,  0.05522631,  0.19295982,
0.14461821,
        0.14022525, -0.2065216 , -0.05020927, -0.08133671,
0.18031682,
        0.35042757,  0.0245426 ,  0.15938364, -0.05617865,
0.00297452,
        0.15442047, -0.01286271,  0.13923576,  0.085941  ,
0.18811756],
      dtype=float32)
```

4. Compute Similarities

Now we could even use Word2vec to compute the similarity between two Make Models in the vocabulary by invoking the model.similarity( ) and passing in the relevant words. For instance, model.similarity('Porsche 718 Cayman', 'Nissan Van') This will give us the Euclidian similarity between Porsche 718 Cayman and Nissan Van.

```
>>> model.similarity('Porsche 718 Cayman', 'Nissan Van')
0.822824584626184

>>> model.similarity('Porsche 718 Cayman', 'Mercedes-Benz SLK-Class')
0.961089779453727
```

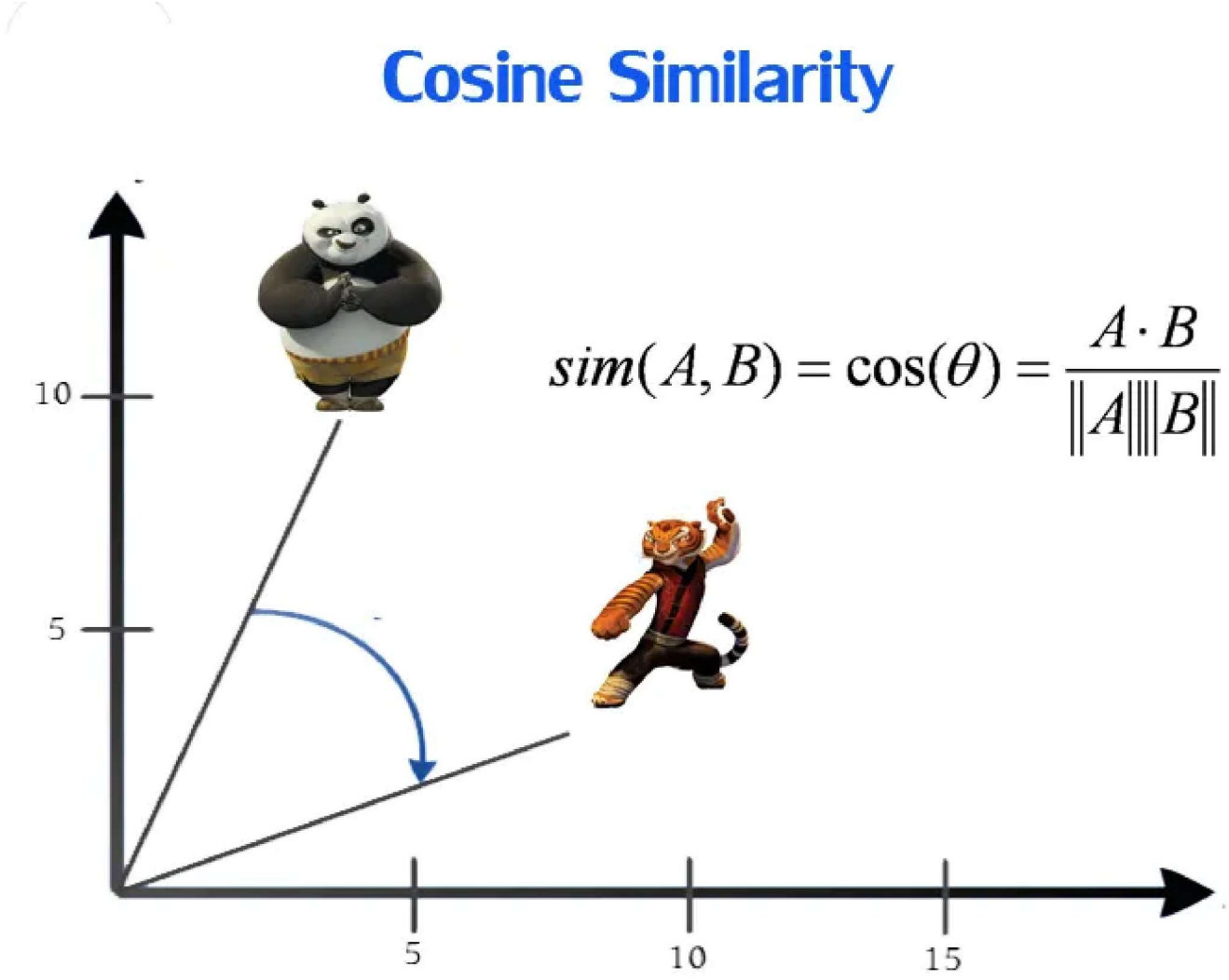
From the above examples, we can tell that Porsche 718 Cayman is more similar to Mercedes-Benz SLK-Class than Nissan Van. We also can use the built-in function model.most_similar() to get a set of the most similar make models for a given make model based on the Euclidean distance.

```
>>> model1.most_similar('Mercedes-Benz SLK-Class')[:5]

[('BMW M4', 0.9959905743598938),
 ('Maserati Coupe', 0.9949707984924316),
 ('Porsche Cayman', 0.9945154190063477),
```

```
('Mercedes-Benz SLS AMG GT', 0.9944609999656677),
('Maserati Spyder', 0.9942780137062073)]
```

However, Euclidian similarity cannot work well for the high-dimensional word vectors. This is because Euclidian similarity will increase as the number of dimensions increases, even if the word embedding stands for different meanings. Alternatively, we can use cosine similarity to measure the similarity between two vectors. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity captures the angle of the word vectors and not the magnitude. Under cosine similarity, no similarity is expressed as a 90-degree angle while the total similarity of 1 is at a 0-degree angle.



The following function shows how can we generate the most similar make model based on cosine similarity.

```
def cosine_distance (model, word,target_list , num) :
    cosine_dict ={}
    word_list = []
    a = model[word]
    for item in target_list :
        if item != word :
            b = model [item]
            cos_sim = dot(a, b)/(norm(a)*norm(b))
            cosine_dict[item] = cos_sim
    dist_sort=sorted(cosine_dict.items(), key=lambda dist:
dist[1],reverse = True) ## in Descedning order
    for item in dist_sort:
        word_list.append((item[0], item[1]))
    return word_list[0:num]

# only get the unique Maker_Model
>>> Maker_Model = list(df.Maker_Model.unique())

# Show the most similar Mercedes-Benz SLK-Class by cosine distance
>>> cosine_distance (model,'Mercedes-Benz SLK-Class',Maker_Model,5)

[('Mercedes-Benz CLK-Class', 0.99737006),
 ('Aston Martin DB9', 0.99593246),
 ('Maserati Spyder', 0.99571854),
 ('Ferrari 458 Italia', 0.9952333),
 ('Maserati GranTurismo Convertible', 0.994994)]
```

5. T-SNE Visualizations

It's hard to visualize the word embedding directly, for they usually have more than 3 dimensions. T-SNE is a useful tool to visualize high-dimensional data by dimension reduction while keeping relative pairwise distance between points. It can be said that T-SNE looking for a new data representation where the neighborhood relations are preserved. The following code shows how to plot the word embedding with T-SNE plot.

```
def display_closestwords_tsnescatterplot(model, word, size):

    arr = np.empty((0,size), dtype='f')
    word_labels = [word]

close_words = model.similar_by_word(word)

arr = np.append(arr, np.array([model[word]]), axis=0)
    for wrd_score in close_words:
        wrd_vector = model[wrd_score[0]]
        word_labels.append(wrd_score[0])
```

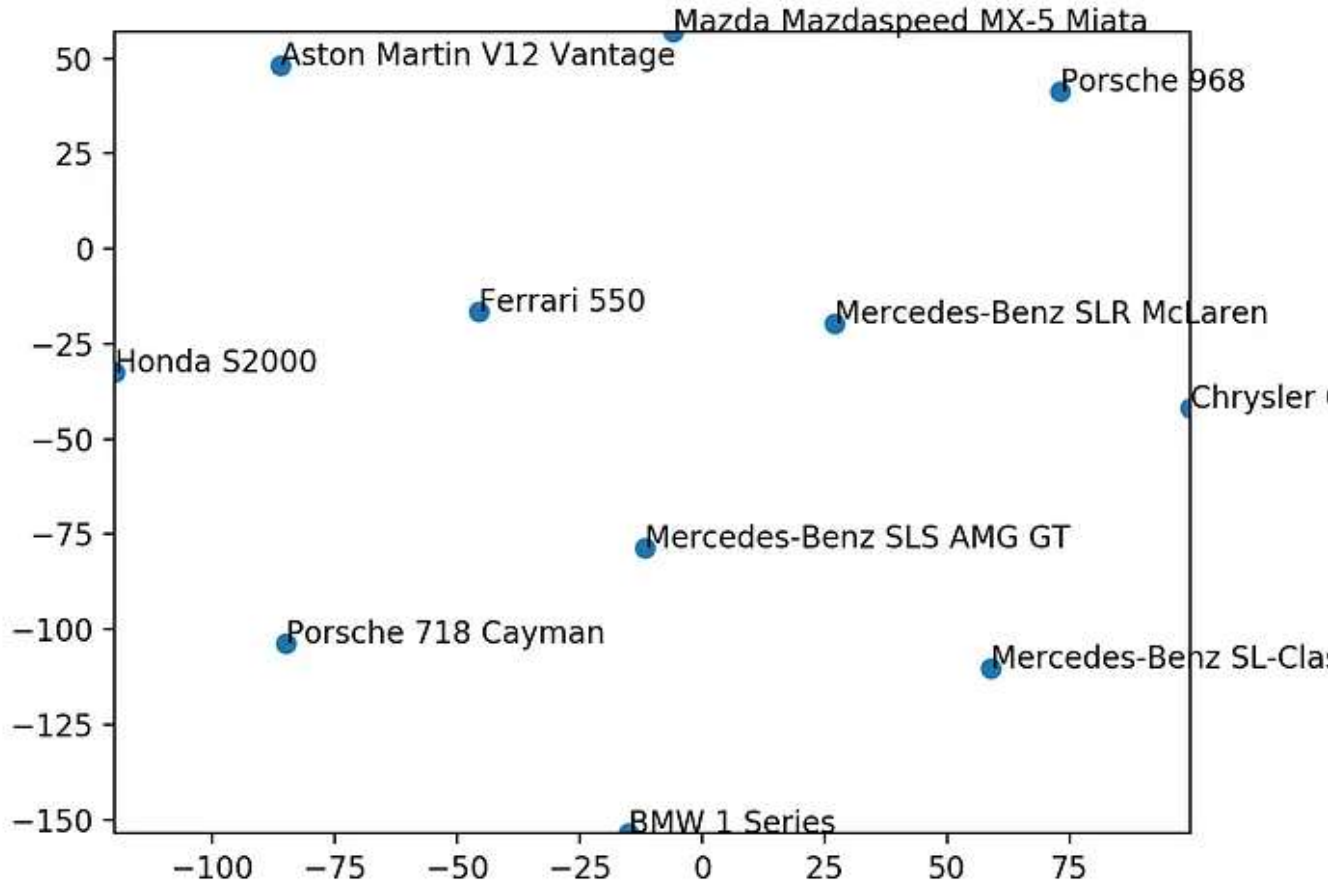
```
        arr = np.append(arr, np.array([wrd_vector]), axis=0)

    tsne = TSNE(n_components=2, random_state=0)
    np.set_printoptions(suppress=True)
    Y = tsne.fit_transform(arr)

x_coords = Y[:, 0]
    y_coords = Y[:, 1]
    plt.scatter(x_coords, y_coords)

for label, x, y in zip(word_labels, x_coords, y_coords):
        plt.annotate(label, xy=(x, y), xytext=(0, 0),
textcoords='offset points')
    plt.xlim(x_coords.min()+0.00005, x_coords.max()+0.00005)
    plt.ylim(y_coords.min()+0.00005, y_coords.max()+0.00005)
    plt.show()

>>> display_closestwords_tsnescatterplot(model, 'Porsche 718 Cayman',
50)
```



This T-SNE plot shows the top 10 similar vehicles to the Porsche 718 Cayman in two-dimensional space.

About Me

I am a master student in Data Science at the University of San Francisco. I am passionate about using Machine Learning to solve business challenges. You can also find me through Linkedin.

Machine Learning NLP Word Embeddings Gensim Word2vec