Deep Learning Course - Assignment 3 Report

Home Depot Product Search Relevance

Submitted by: Itay Bouganim and Ido Rom

Problem Statement

Shoppers rely on Home Depot's product authority to find and buy the latest products and to get timely solutions to their home improvement needs. From installing a new ceiling fan to remodeling an entire kitchen, with the click of a mouse or tap of the screen, customers expect the correct results to their queries – quickly. Speed, accuracy and delivering a frictionless customer experience are essential.

Task Description

Home Depot is asking to help them improve their customers' shopping experience by developing a model that can accurately predict the relevance of search results.

Search relevancy is an implicit measure Home Depot uses to gauge how quickly they can get customers to the right products. Currently, human raters evaluate the impact of potential changes to their search algorithms, which is a slow and subjective process. By removing or minimizing human input in search relevance evaluation, Home Depot hopes to increase the number of iterations their team can perform on the current search algorithms.

Home Depot Product Search Relevance dataset link

Preface

Task goal: Finding the relevance value of the connection between a search phrase and a product description in a hardware store website.

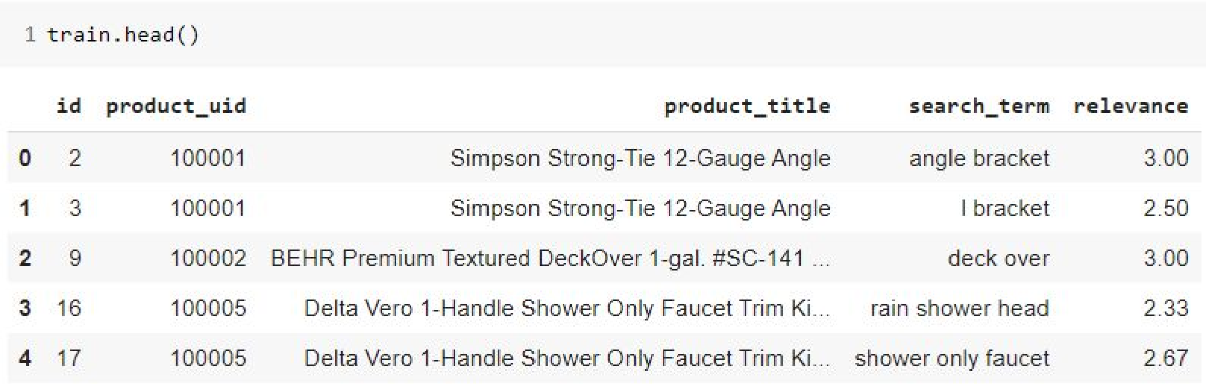
In the following sections we are going to train multiple Siamese LSTM based networks with the goal of determining the relevance value for each product according to the search term and product description.

At first we are going to perform our task at a character based level, meaning that every search term/product description will be tokenized and represented as a sequence of numeric character representations.

In the second stage we will produce a network on a word level, each search term/product description will be represented as a tokenized sequence of preprocessed numeric words representation.

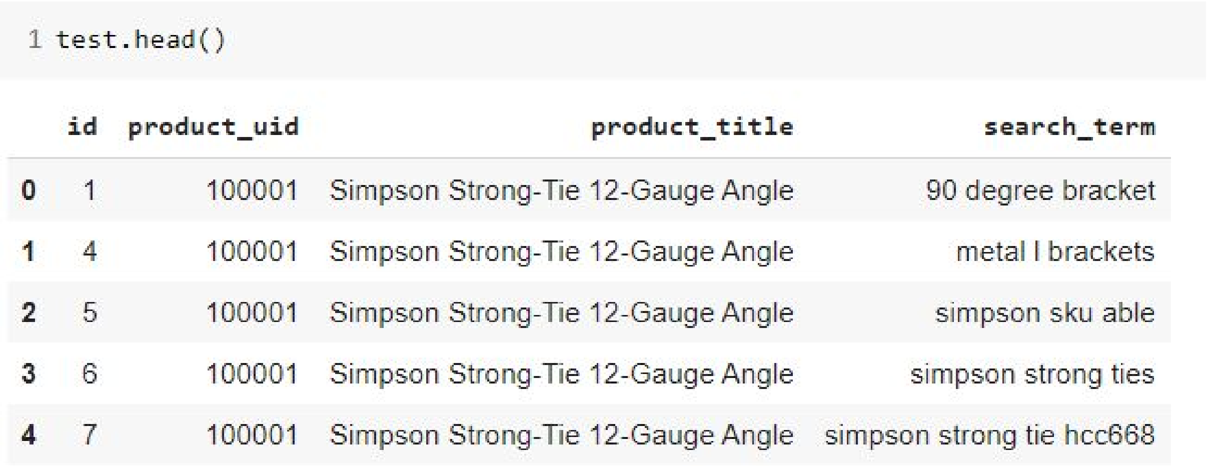
We will also use embedding in the second stage in order to produce a vectorized representation for every tokenized word in our corpus.

Data Analysis -



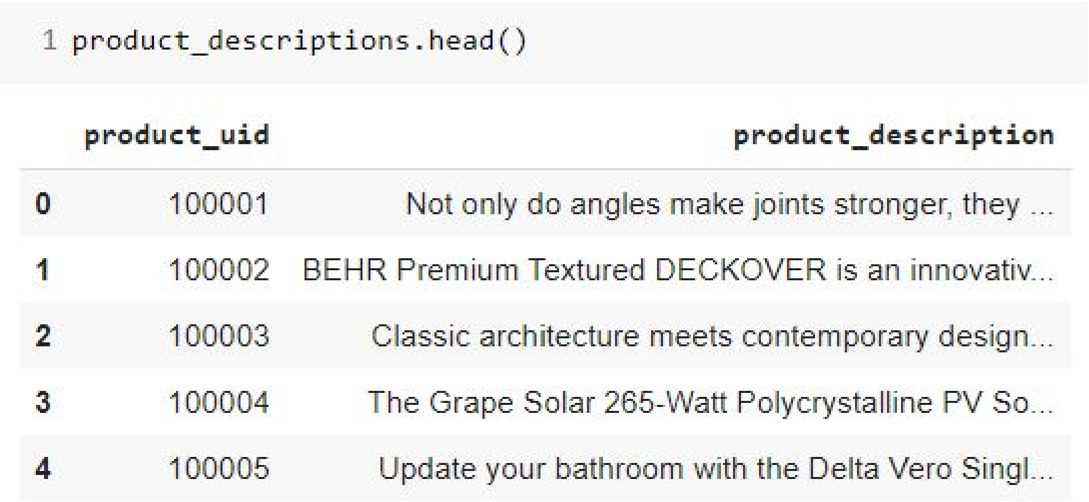
```
train.head()
```

	id	product_uid	product_title	search_term	relevance
0	2	100001	Simpson Strong-Tie 12-Gauge Angle	angle bracket	3.00
1	3	100001	Simpson Strong-Tie 12-Gauge Angle	l bracket	2.50
2	9	100002	BEHR Premium Textured DeckOver 1-gal. #SC-141 ...	deck over	3.00
3	16	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	rain shower head	2.33
4	17	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	shower only faucet	2.67



```
test.head()
```

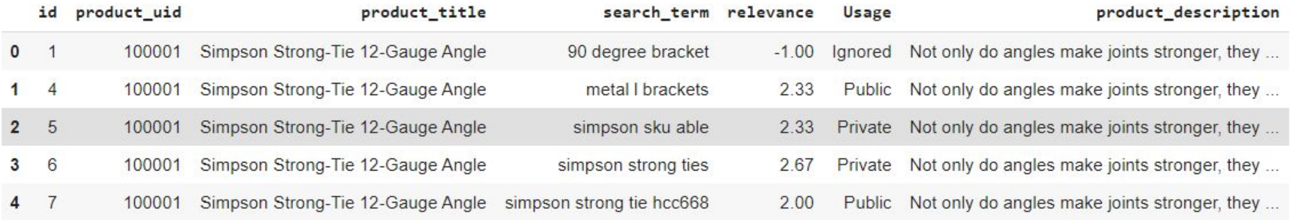
	id	product_uid	product_title	search_term
0	1	100001	Simpson Strong-Tie 12-Gauge Angle	90 degree bracket
1	4	100001	Simpson Strong-Tie 12-Gauge Angle	metal l brackets
2	5	100001	Simpson Strong-Tie 12-Gauge Angle	simpson sku able
3	6	100001	Simpson Strong-Tie 12-Gauge Angle	simpson strong ties
4	7	100001	Simpson Strong-Tie 12-Gauge Angle	simpson strong tie hcc668



```
product_descriptions.head()
```

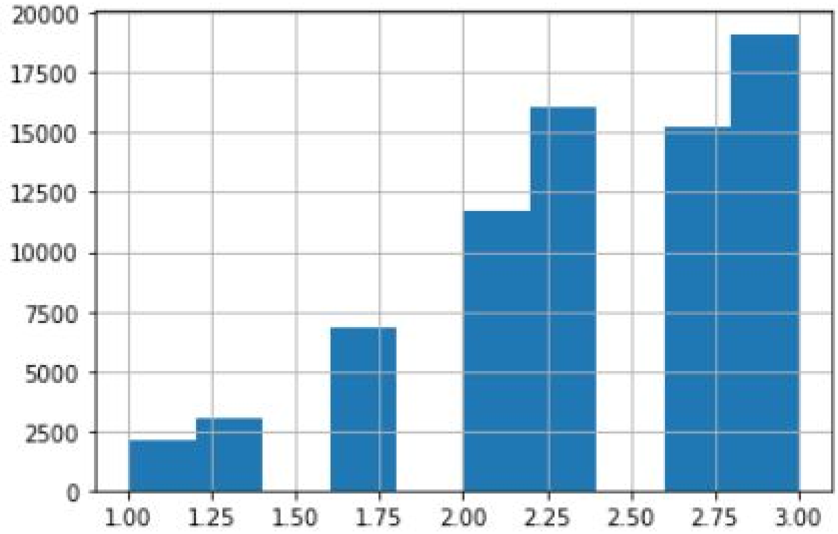
	product_uid	product_description
0	100001	Not only do angles make joints stronger, they ...
1	100002	BEHR Premium Textured DECKOVER is an innovativ...
2	100003	Classic architecture meets contemporary design...
3	100004	The Grape Solar 265-Watt Polycrystalline PV So...
4	100005	Update your bathroom with the Delta Vero Singl...

We can see that the product descriptions, attributes and titles relevance data are seperate. Therefore we joined the data together.



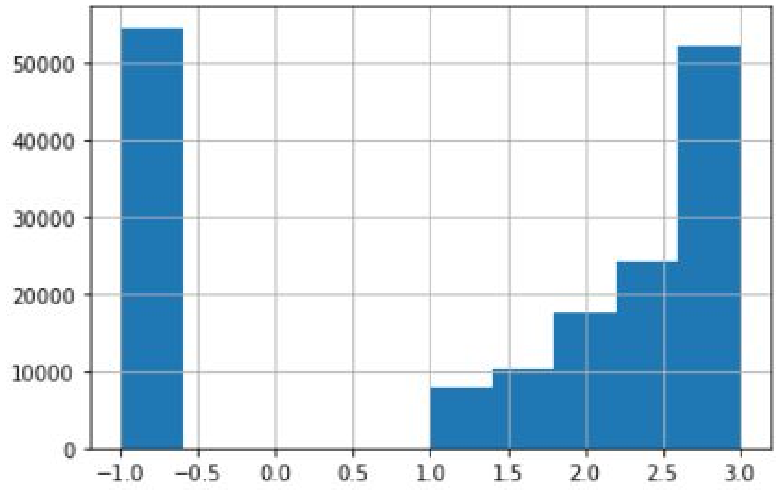
	id	product_uid	product_title	search_term	relevance	Usage	product_description
0	1	100001	Simpson Strong-Tie 12-Gauge Angle	90 degree bracket	-1.00	Ignored	Not only do angles make joints stronger, they ...
1	4	100001	Simpson Strong-Tie 12-Gauge Angle	metal l brackets	2.33	Public	Not only do angles make joints stronger, they ...
2	5	100001	Simpson Strong-Tie 12-Gauge Angle	simpson sku able	2.33	Private	Not only do angles make joints stronger, they ...
3	6	100001	Simpson Strong-Tie 12-Gauge Angle	simpson strong ties	2.67	Private	Not only do angles make joints stronger, they ...
4	7	100001	Simpson Strong-Tie 12-Gauge Angle	simpson strong tie hcc668	2.00	Public	Not only do angles make joints stronger, they ...

Now, we wanted to check the histogram of relevance column in order to realize the distribution of the data:

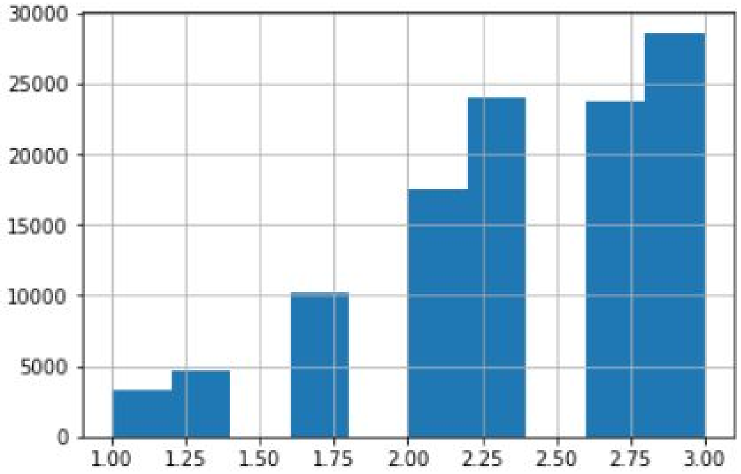


Train mean relevance value: 2.381633791027016

We saw that the test data contains many -1 values. These values we wanted to ignore.
Test relevance histogram -



Test relevance histogram after ignoring -1 values -



Test mean relevance value: 2.3805248645900994
Train data size: 74067
Test data size: 112067

1.a. Character Level LSTM Model Preprocess

Character Tokenization

Character Tokenization splits a piece of text into a set of characters.

Character Tokenizers handle OOV words coherently by preserving the information of the word.

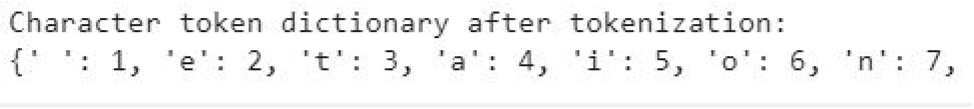
It breaks down the OOV word into characters and represents the word in terms of these characters.

It also limits the size of the vocabulary.

Drawbacks of Character Tokenization

Character tokens solve the OOV problem but the length of the input and output sentences increases rapidly as we are representing a sentence as a sequence of characters.
As a result, it becomes challenging to learn the relationship between the characters to form meaningful words.

Some insights and values from the preprocess as can be seen in the notebook -



```
Character token dictionary after tokenization:
{' ': 1, 'e': 2, 't': 3, 'a': 4, 'i': 5, 'o': 6, 'n': 7,
```

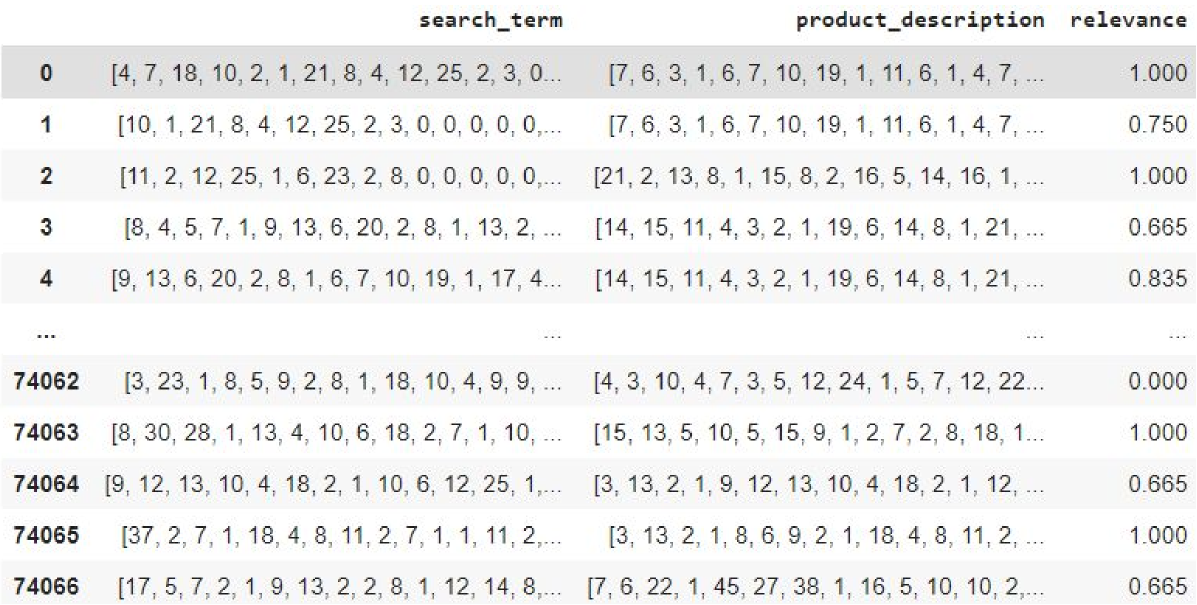
Max train data search term length: 60
Min train data search term length: 2
Mean train data search term length: 19.008816341960657

Max train data product description length: 5516
Min train data product description length: 153
Mean train data product description length: 885.663750388162

Max test data search term length: 60
Min test data search term length: 1
Mean test data search term length: 18.929319068057502

Max test data product description length: 5479
Min test data product description length: 8
Mean test data product description length: 891.1579858477518

We normalized the relevance values to range between [0, 1] instead of [1, 3]

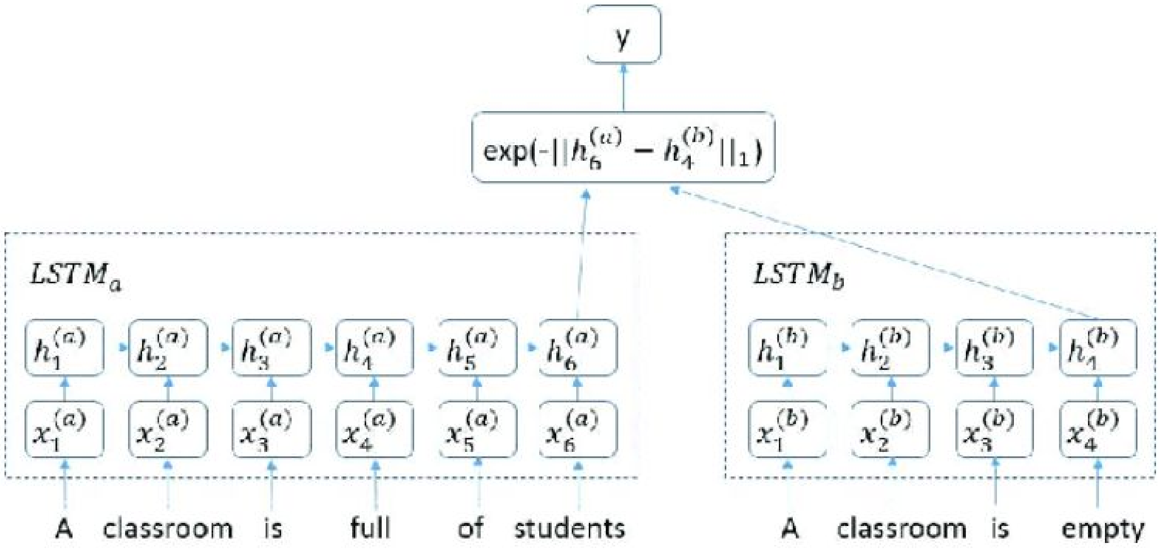


	search_term	product_description	relevance
0	[4, 7, 18, 10, 2, 1, 21, 8, 4, 12, 25, 2, 3, 0...	[7, 6, 3, 1, 6, 7, 10, 19, 1, 11, 6, 1, 4, 7, ...	1.000
1	[10, 1, 21, 8, 4, 12, 25, 2, 3, 0, 0, 0, 0, 0,...	[7, 6, 3, 1, 6, 7, 10, 19, 1, 11, 6, 1, 4, 7, ...	0.750
2	[11, 2, 12, 25, 1, 6, 23, 2, 8, 0, 0, 0, 0, 0,...	[21, 2, 13, 8, 1, 15, 8, 2, 16, 5, 14, 16, 1, ...	1.000
3	[8, 4, 5, 7, 1, 9, 13, 6, 20, 2, 8, 1, 13, 2, ...	[14, 15, 11, 4, 3, 2, 1, 19, 6, 14, 8, 1, 21, ...	0.665
4	[9, 13, 6, 20, 2, 8, 1, 6, 7, 10, 19, 1, 17, 4...	[14, 15, 11, 4, 3, 2, 1, 19, 6, 14, 8, 1, 21, ...	0.835
...	...	...	...
74062	[3, 23, 1, 8, 5, 9, 2, 8, 1, 18, 10, 4, 9, 9, ...	[4, 3, 10, 4, 7, 3, 5, 12, 24, 1, 5, 7, 12, 22...	0.000
74063	[8, 30, 28, 1, 13, 4, 10, 6, 18, 2, 7, 1, 10, ...	[15, 13, 5, 10, 5, 15, 9, 1, 2, 7, 2, 8, 18, 1...	1.000
74064	[9, 12, 13, 10, 4, 18, 2, 1, 10, 6, 12, 25, 1,...	[3, 13, 2, 1, 9, 12, 13, 10, 4, 18, 2, 1, 12, ...	0.665
74065	[37, 2, 7, 1, 18, 4, 8, 11, 2, 7, 1, 1, 11, 2,...	[3, 13, 2, 1, 8, 6, 9, 2, 1, 18, 4, 8, 11, 2, ...	1.000
74066	[17, 5, 7, 2, 1, 9, 13, 2, 2, 8, 1, 12, 14, 8,...	[7, 6, 22, 1, 45, 27, 38, 1, 16, 5, 10, 10, 2,...	0.665

1.b. Character Level LSTM Model

We used RMSE loss function. To determine each model output vector semantic meaning similarity we put them through the defined similarity function -

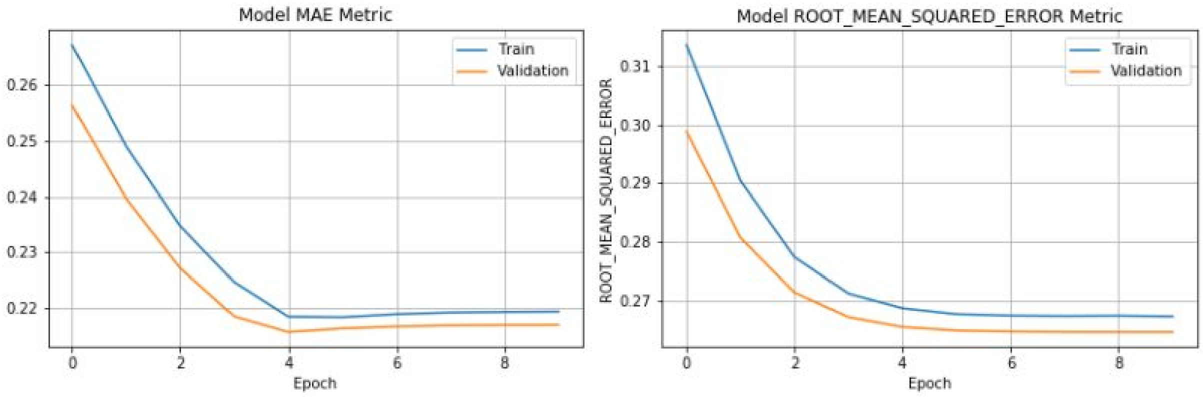
$$exp(-\|h^{(left)} - h^{(right)}\|_1)$$


y
$\exp(-||h_6^{(a)} - h_4^{(b)}||_1)$
$LSTM_a$
$LSTM_b$
$h_1^{(a)}$ $h_2^{(a)}$ $h_3^{(a)}$ $h_4^{(a)}$ $h_5^{(a)}$ $h_6^{(a)}$
$x_1^{(a)}$ $x_2^{(a)}$ $x_3^{(a)}$ $x_4^{(a)}$ $x_5^{(a)}$ $x_6^{(a)}$
$h_1^{(b)}$ $h_2^{(b)}$ $h_3^{(b)}$ $h_4^{(b)}$
$x_1^{(b)}$ $x_2^{(b)}$ $x_3^{(b)}$ $x_4^{(b)}$
A classroom is full of students
A classroom is empty

First we tried a network without joined weights.
We trained a network with the same models for search phrase input and product description input (different models) and combined them by using the exponent manhattan distance function to determine similarities.

```
Model: "Example Siamese Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 60, 1)]           0

_________________________________________________________________
lstm (LSTM)                  (None, 60, 128)           66560

_________________________________________________________________
dropout (Dropout)            (None, 60, 128)           0

_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               131584
=================================================================
Total params: 198,144
Trainable params: 198,144
Non-trainable params: 0

_________________________________________________________________
Model: "functional_1"
________________________________________________________________________
___________________________
Layer (type)                    Output Shape         Param #
Connected to
========================================================================
===========================
input_2 (InputLayer)            [(None, 60, 1)]      0

________________________________________________________________________
___________________________
input_3 (InputLayer)            [(None, 885, 1)]     0
```

```
________________________________________________________________________
___________________________
search_siamese (Functional)     (None, 128)          198144
input_2[0][0]

________________________________________________________________________
___________________________
description_siamese (Functional (None, 128)          198144
input_3[0][0]

________________________________________________________________________
___________________________
lambda (Lambda)                 (None, 1)            0
search_siamese[0][0]

description_siamese[0][0]

________________________________________________________________________
___________________________
dense (Dense)                   (None, 1)            2
lambda[0][0]
========================================================================
===========================
Total params: 396,290
Trainable params: 396,290
Non-trainable params: 0

________________________________________________________________________
___________________________
```



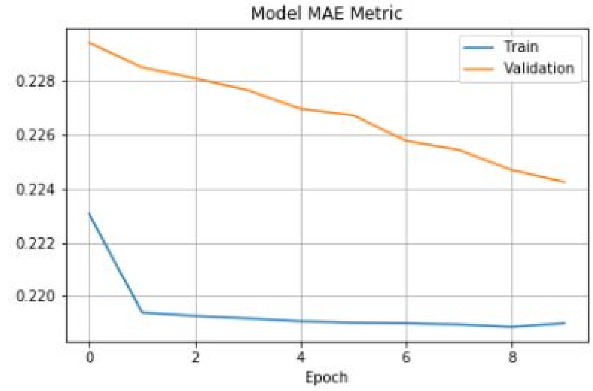
Test data results:
Char Seq Siamese RMSE: 0.26763587052709054
Char Seq Siamese MSE: 0.07162895919279358
Char Seq Siamese MAE: 0.21933228496245008

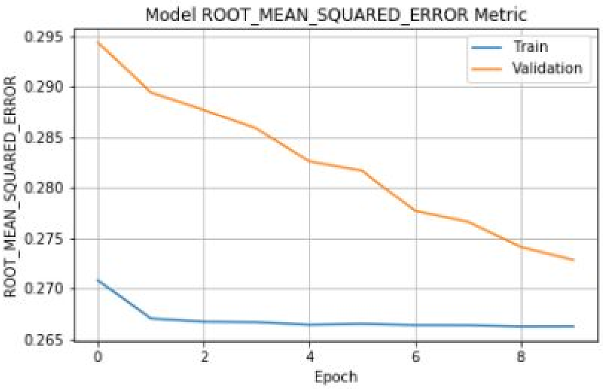
Validation data results:
Char Seq Siamese RMSE: 0.26489372462495714
Char Seq Siamese MSE: 0.07016868534568263
Char Seq Siamese MAE: 0.21699708677905766

Train data results:
Char Seq Siamese RMSE: 0.26751203080482217
Char Seq Siamese MSE: 0.07156268662532013
Char Seq Siamese MAE: 0.21939520269499052

We now used a siamese model with common weights (train a single layer for both inputs to train on). we trained a true siamese network (train on the same model with same weights for both input vectors). We used the mean description length as the char sequence size.

```
Model: "functional_5"
________________________________________________________________________
___________________________
Layer (type)                     Output Shape          Param #
Connected to
========================================================================
===========================
input_8 (InputLayer)             [(None, 885, 1)]      0
________________________________________________________________________
___________________________
input_9 (InputLayer)             [(None, 885, 1)]      0
________________________________________________________________________
___________________________
lstm_8 (LSTM)                    (None, 885, 128)      66560
input_8[0][0]

input_9[0][0]
________________________________________________________________________
___________________________
lstm_9 (LSTM)                    (None, 128)           131584
lstm_8[0][0]

lstm_8[1][0]
________________________________________________________________________
___________________________
lambda_2 (Lambda)                (None, 1)             0
lstm_9[0][0]

lstm_9[1][0]
________________________________________________________________________
___________________________
dense_2 (Dense)                  (None, 1)             2
lambda_2[0][0]
========================================================================
===========================
Total params: 198,146
Trainable params: 198,146
Non-trainable params: 0
________________________________________________________________________
___________________________
```





Test data results:
Char Seq Joined Siamese RMSE: 0.27407735574900677
Char Seq Joined Siamese MSE: 0.07511839693436762
Char Seq Joined Siamese MAE: 0.22332423770733104

Validation data results:
Char Seq Joined Siamese RMSE: 0.27410554637302154
Char Seq Joined Siamese MSE: 0.07513385055245266
Char Seq Joined Siamese MAE: 0.22425943334543286

Train data results:
Char Seq Joined Siamese RMSE: 0.27318294490269157
Char Seq Joined Siamese MSE: 0.07462892138570701
Char Seq Joined Siamese MAE: 0.2230227561248734

1.c. Baseline

Naive mean calculation baseline.
We started with most Naive baseline by calculating the mean relevance for the training and testing data and calculated the MSE, MAE and RMSE metrics for it.
Train mean relevance: 2.381633791027016
Test mean relevance: 2.3805248645900994

results -
Test data results:
naive_baseline RMSE: 0.5352715864190928
naive_baseline MSE: 0.2865156712276123
naive_baseline MAE: 0.4386375183244027

Train data results:
naive_baseline RMSE: 0.5339803436691322
naive_baseline MSE: 0.2851350074250045
naive_baseline MAE: 0.43788185944303953

Baseline Count Vectorizer model

As we have seen, In order to use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called tokenization.

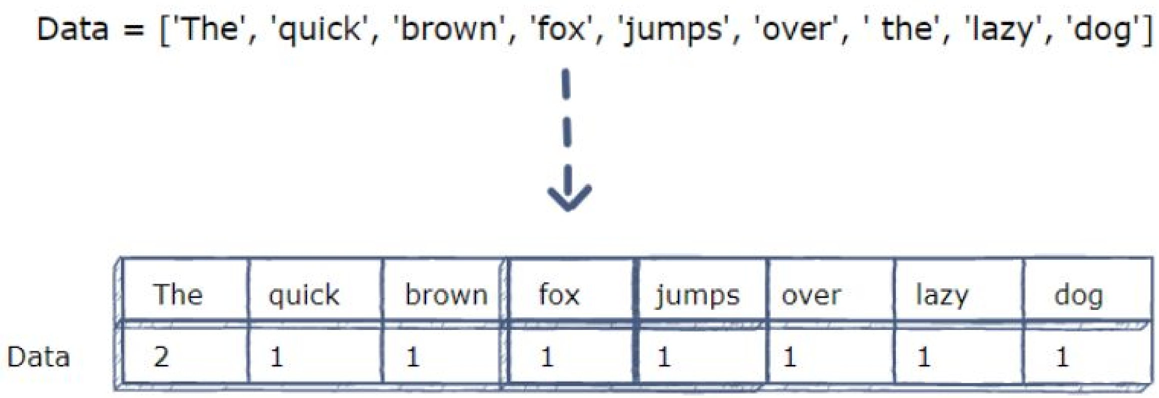
These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms.

This process is called feature extraction (or vectorization).

Scikit-learn's **CountVectorizer** is used to convert a collection of text documents to a vector of term/token counts.

It also enables the pre-processing of text data prior to generating the vector representation.

This functionality makes it a highly flexible feature representation module for text.

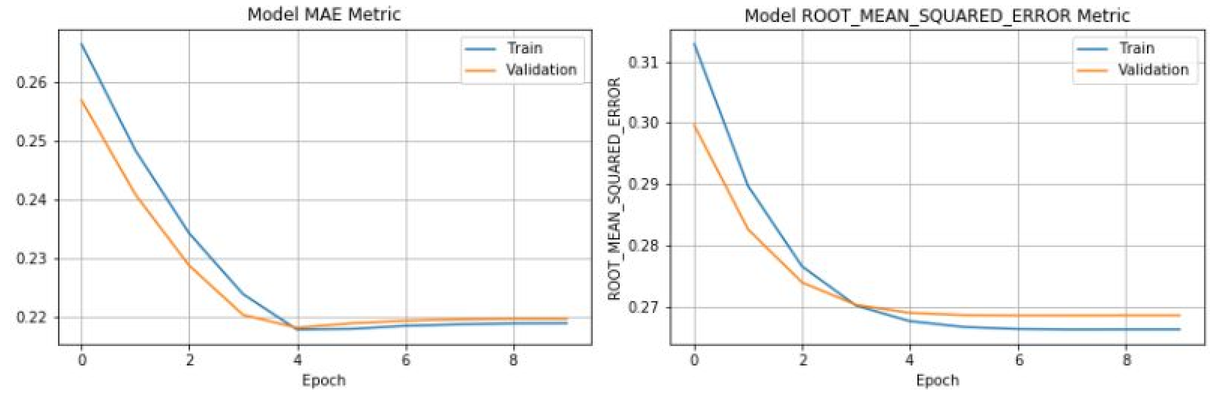


Data = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', ' the', 'lazy', 'dog']

	The	quick	brown	fox	jumps	over	lazy	dog
Data	2	1	1	1	1	1	1	1

```
Model: "Example Benchamrk Siamese Model"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

input_10 (InputLayer)        [(None, 68)]              0

_________________________________________________________________

dense_2 (Dense)              (None, 512)               35328

_________________________________________________________________

dense_3 (Dense)              (None, 256)               131328

=================================================================

Total params: 166,656

Trainable params: 166,656

Non-trainable params: 0

_________________________________________________________________

Model: "functional_5"
```

```
_______________________________________________________________________
___________________________
Layer (type)                    Output Shape         Param #
Connected to
=======================================================================
===========================
input_11 (InputLayer)           [(None, 68)]         0

_______________________________________________________________________
___________________________
input_12 (InputLayer)           [(None, 68)]         0

_______________________________________________________________________
___________________________
search_siamese (Functional)     (None, 256)          166656
input_11[0][0]

_______________________________________________________________________
___________________________
description_siamese (Functional (None, 256)          166656
input_12[0][0]

_______________________________________________________________________
___________________________
lambda_2 (Lambda)               (None, 1)            0
search_siamese[0][0]

description_siamese[0][0]

_______________________________________________________________________
___________________________
dense_8 (Dense)                 (None, 1)            2
lambda_2[0][0]
=======================================================================
===========================
Total params: 333,314
Trainable params: 333,314
Non-trainable params: 0
_______________________________________________________________________
___________________________
```



Test data results:

Benchmark Char Seq Siamese RMSE: 0.26765307157572427

Benchmark Char Seq Siamese MSE: 0.07163816672391977

Benchmark Char Seq Siamese MAE: 0.2191165586465967

Validation data results:

Benchmark Char Seq Siamese RMSE: 0.268867743504799

Benchmark Char Seq Siamese MSE: 0.07228986349736238

Benchmark Char Seq Siamese MAE: 0.21925096545606326

Train data results:

Benchmark Char Seq Siamese RMSE: 0.2665490056182621

Benchmark Char Seq Siamese MSE: 0.07104837239608433

Benchmark Char Seq Siamese MAE: 0.2185385792614681

We can see that out baseline benchmark model produced similar results to the Character Level LSTM model.

1.d. Use LSTM model as Feature Extractor for classical ML algorithms

We tried to use the LSTM model as a feature extractor for xgboost, catboost and random forest regressor models.

results -

xgboost

Test data results:

xgboost model RMSE: 0.26791087236420574

xgboost model MSE: 0.07177623553094974

xgboost model MAE: 0.21998285287213729

Validation data results:

xgboost model RMSE: 0.2632566637928397

xgboost model MSE: 0.06930407103133623

xgboost model MAE: 0.21603435530270804

Train data results:

xgboost model RMSE: 0.2566659991637794

xgboost model MSE: 0.06587743512674121

xgboost model MAE: 0.21066640730590463

catboost

Test data results:

catboost model RMSE: 0.26677886795277717

catboost model MSE: 0.07117096438616531

catboost model MAE: 0.2189584896691271

Validation data results:

catboost model RMSE: 0.26350179605869045

catboost model MSE: 0.06943319652615569

catboost model MAE: 0.2161510798585918

Train data results:

catboost model RMSE: 0.26541509622279963

catboost model MSE: 0.07044517330295798

catboost model MAE: 0.21800092602024698

random forest

Test data results:

random forest model RMSE: 0.2676357934535374

random forest model MSE: 0.07162891793750452

random forest model MAE: 0.21931951898243815

Validation data results:

random forest model RMSE: 0.26489555634610457

random forest model MSE: 0.07016965577191227

random forest model MAE: 0.21698373674949994

Train data results:

random forest model RMSE: 0.2675119784079979

random forest model MSE: 0.07156265859176113

random forest model MAE: 0.21938119417562996

2.a. Word Level LSTM Model Preprocess

We saw that the most common words in our corpus are english stop words.

Therefore we performed some preprocessing steps on our data in order to reduce its size and to gain a more informative words list for every sentence in the corpus

Word preprocessing

Tokenization

We will use punctuation tokenization in order to remove any punctuation symbols (e.g. . , ! ( )) From the individual words.

This will help us reduce the word count in our corpus and will also cause simillar words have the same tokenized value (e.g. doors, doors. and doors will all be tokenized as doors).

Lowercasing:

This is the simplest technique of text preprocessing which consists of lowercasing every single token of the input text.

It helps in dealing with sparsity issues in the dataset.

For example, a text is having mixed-case occurrences of the token ‘Canada’, i.e., at some places token ‘canada’ is and in other ‘Canada’ is used.

Stop-word removal:

Stop-words are commonly used words in a language. Examples are ‘a’, ’an’, ’the’, ’is’, ’what’ etc.

Stop-words are removed from the text so that we can concentrate on more important words and prevent stop-words from being analyzed.

If we search ‘what is text preprocessing’, we want to focus more on ‘text preprocessing’ rather than ‘what is’.

Stemming:

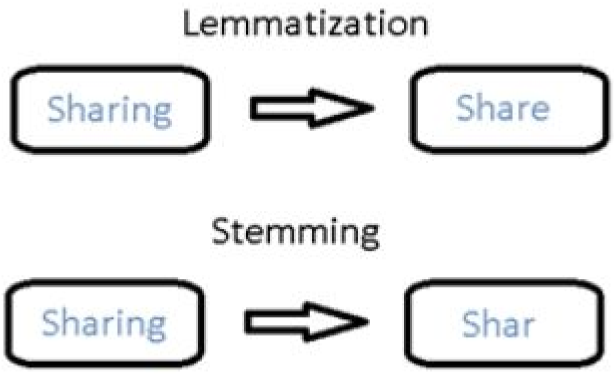
Stemming is the elementary rule-based process of removal of inflectional forms from a token.

The token is converted into its root form. For example, the word ‘troubled’ is converted into ‘trouble’ after performing stemming.

Lemmatization: (will not be used here)

Lemmatization is similar to stemming, the difference being that lemmatization refers to doing things properly with the use of vocabulary and morphological analysis of words, aiming to remove inflections from the word and to return the base or dictionary form of that word, also known as the lemma.

It does a full morphological analysis of the word to accurately identify the lemma for each word.



2.b. Create word embedding vectors

We used the word2vec model in order to produce vectorized representation for every word in our preprocesses corpus.

Word2Vec

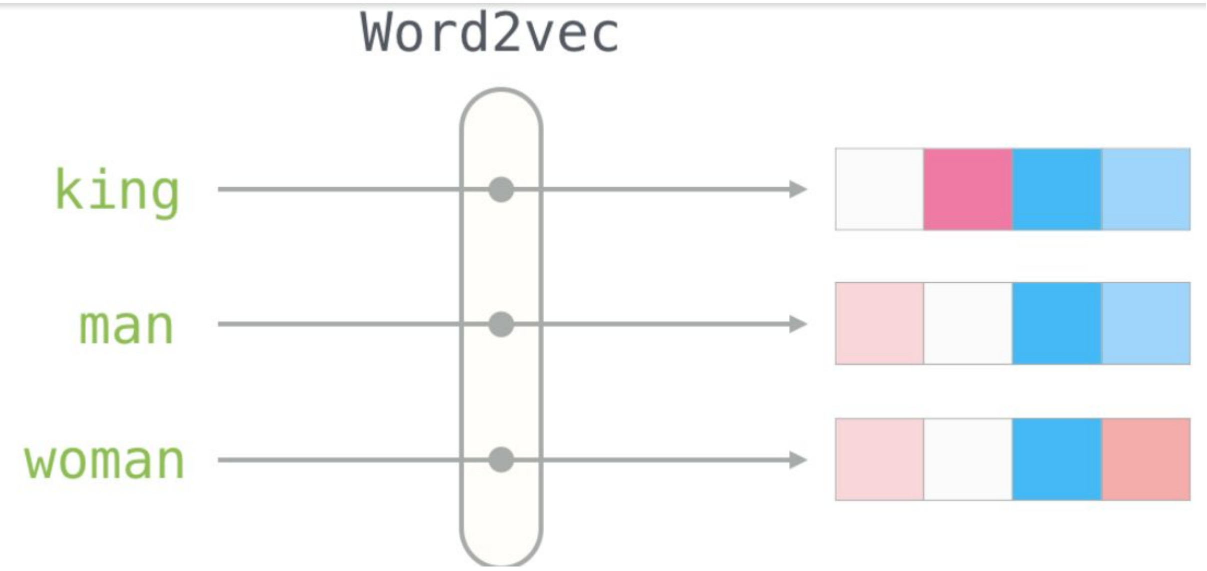
Word2vec is a technique for natural language processing.

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text.

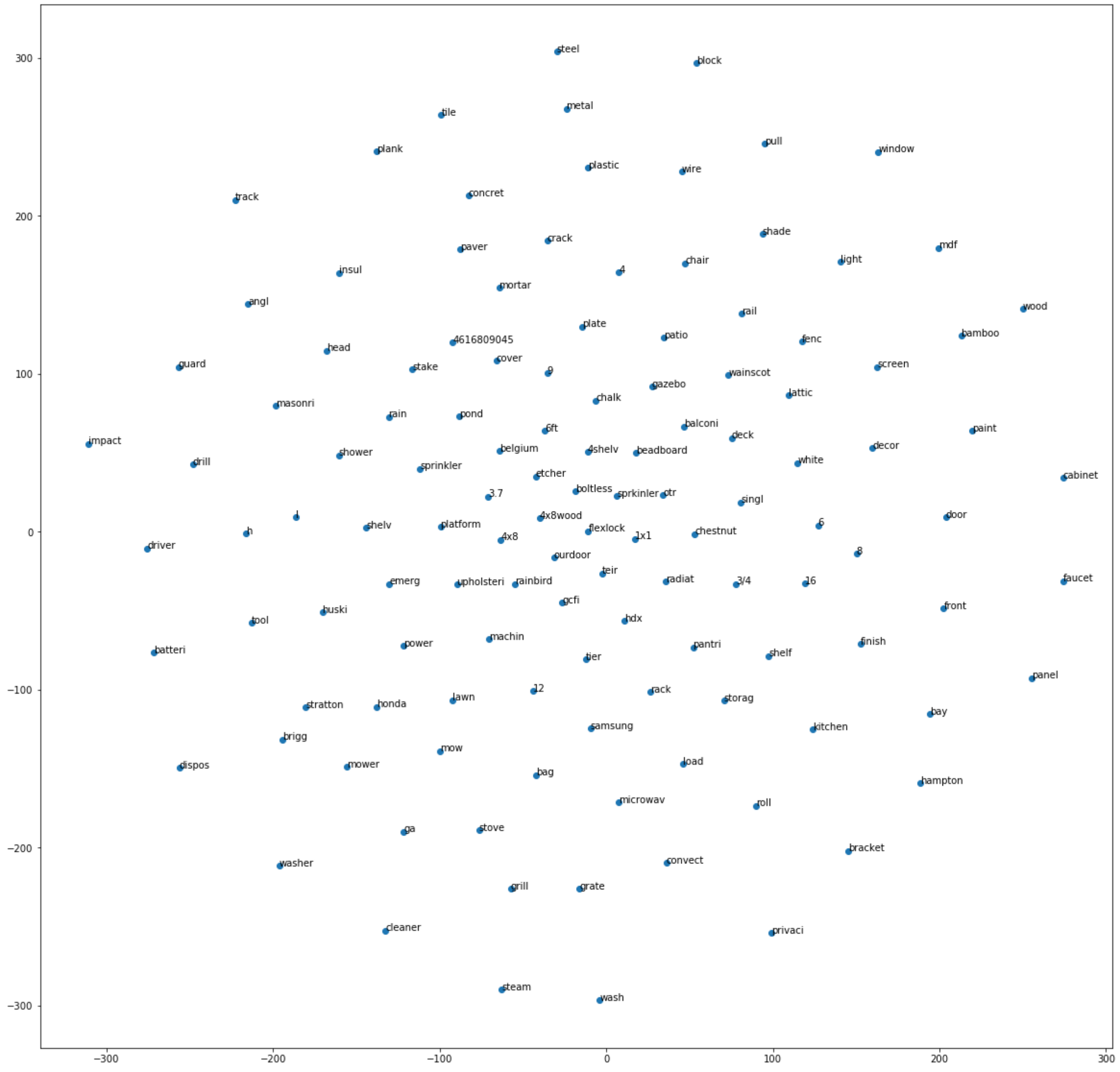
Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.

As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector.

The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.



Simulate the word2vec word distance (or similarity) on a 2D plane -



Max train data search term length: 11

Min train data search term length: 0

Mean train data search term length: 3.0278126561086585

Max train data product description length: 685

Min train data product description length: 16

Mean train data product description length: 92.61781900171466

Max test data search term length: 11

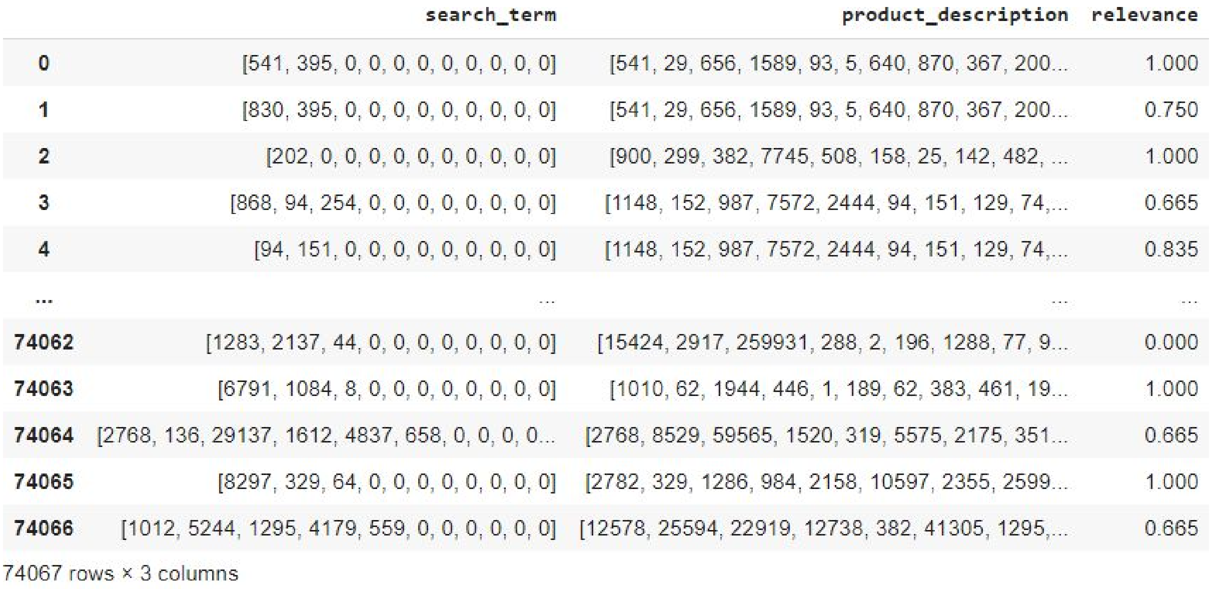
Min test data search term length: 0

Mean test data search term length: 3.021326527880643

Max test data product description length: 685

Min test data product description length: 1

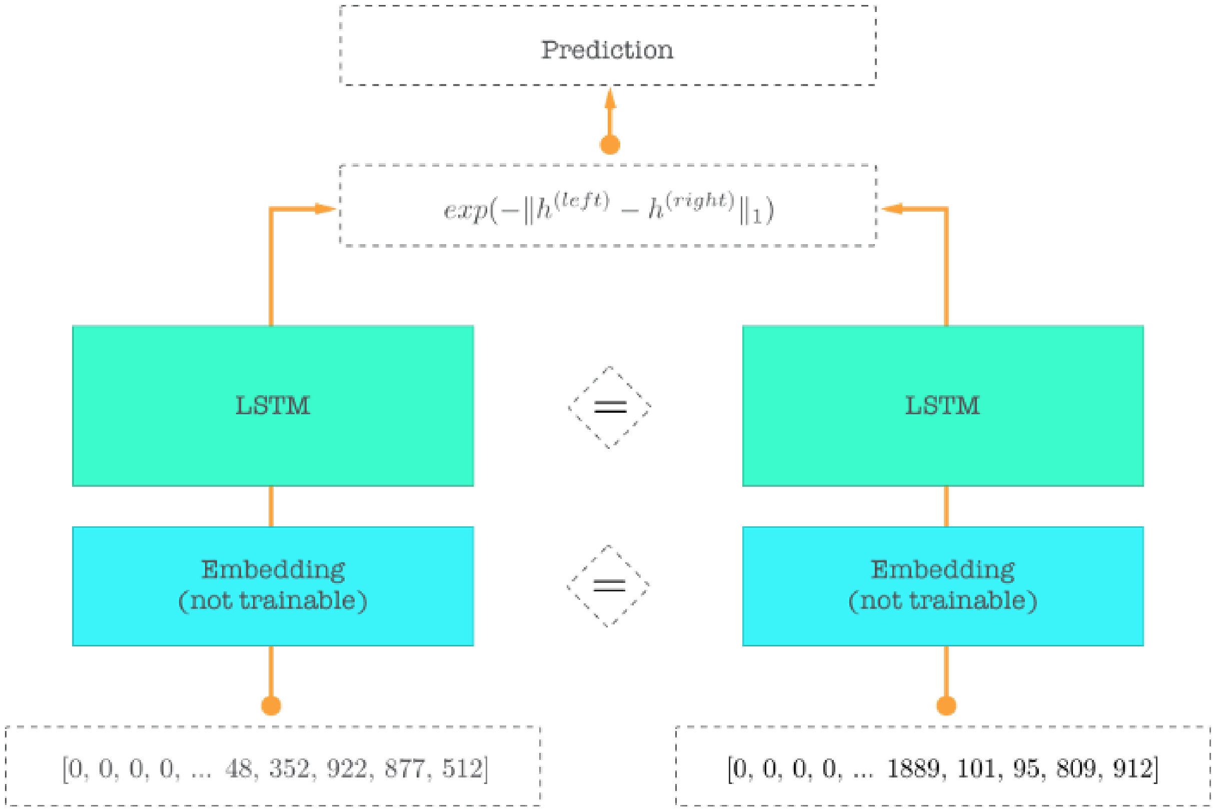
Mean test data product description length: 93.15989541970428



	search_term	product_description	relevance
0	[541, 395, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[541, 29, 656, 1589, 93, 5, 640, 870, 367, 200...	1.000
1	[830, 395, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[541, 29, 656, 1589, 93, 5, 640, 870, 367, 200...	0.750
2	[202, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[900, 299, 382, 7745, 508, 158, 25, 142, 482, ...	1.000
3	[868, 94, 254, 0, 0, 0, 0, 0, 0, 0, 0]	[1148, 152, 987, 7572, 2444, 94, 151, 129, 74,...	0.665
4	[94, 151, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1148, 152, 987, 7572, 2444, 94, 151, 129, 74,...	0.835
...	...	...	...
74062	[1283, 2137, 44, 0, 0, 0, 0, 0, 0, 0, 0]	[15424, 2917, 259931, 288, 2, 196, 1288, 77, 9...	0.000
74063	[6791, 1084, 8, 0, 0, 0, 0, 0, 0, 0, 0]	[1010, 62, 1944, 446, 1, 189, 62, 383, 461, 19...	1.000
74064	[2768, 136, 29137, 1612, 4837, 658, 0, 0, 0, 0...	[2768, 8529, 59565, 1520, 319, 5575, 2175, 351...	0.665
74065	[8297, 329, 64, 0, 0, 0, 0, 0, 0, 0, 0]	[2782, 329, 1286, 984, 2158, 10597, 2355, 2599...	1.000
74066	[1012, 5244, 1295, 4179, 559, 0, 0, 0, 0, 0, 0]	[12578, 25594, 22919, 12738, 382, 41305, 1295,...	0.665

74067 rows × 3 columns

2.c. Training Word Level LSTM model



We used cosine similarity this time to measure similarity:

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

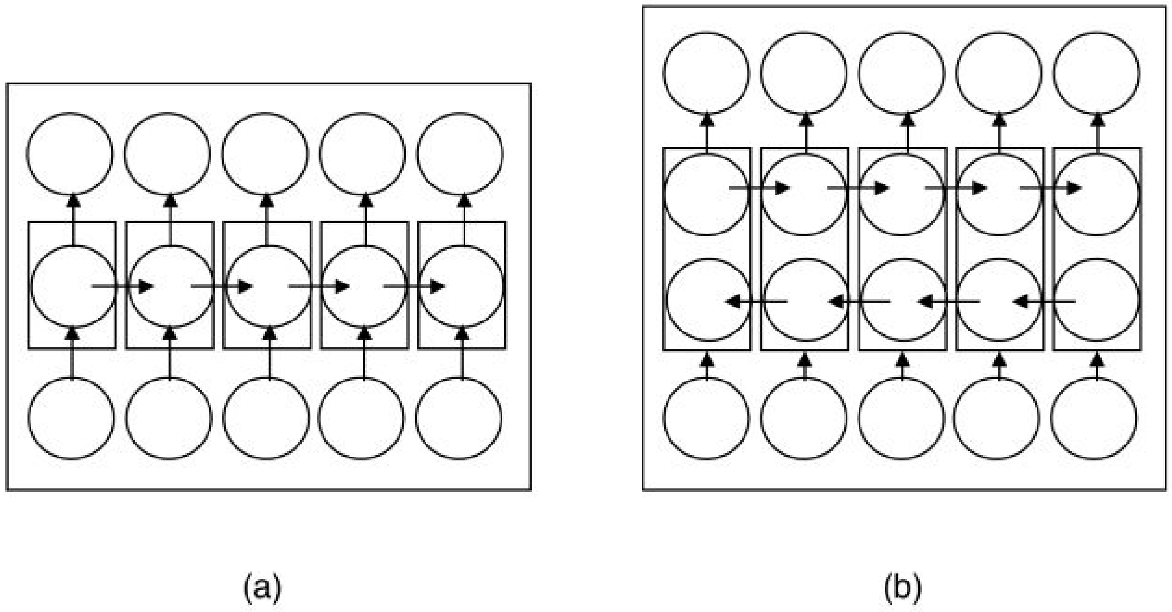
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

For the word level model we will use Bidirectional LSTM with feed-forward to convolutional layer that will then produce a representation vector of size 128.

Bidirectional Recurrent Neural Networks

Bidirectional Recurrent Neural Networks (BRNN) connect two hidden layers of opposite directions to the same output.

With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously.



Structure overview
(a) unidirectional RNN
(b) bidirectional RNN

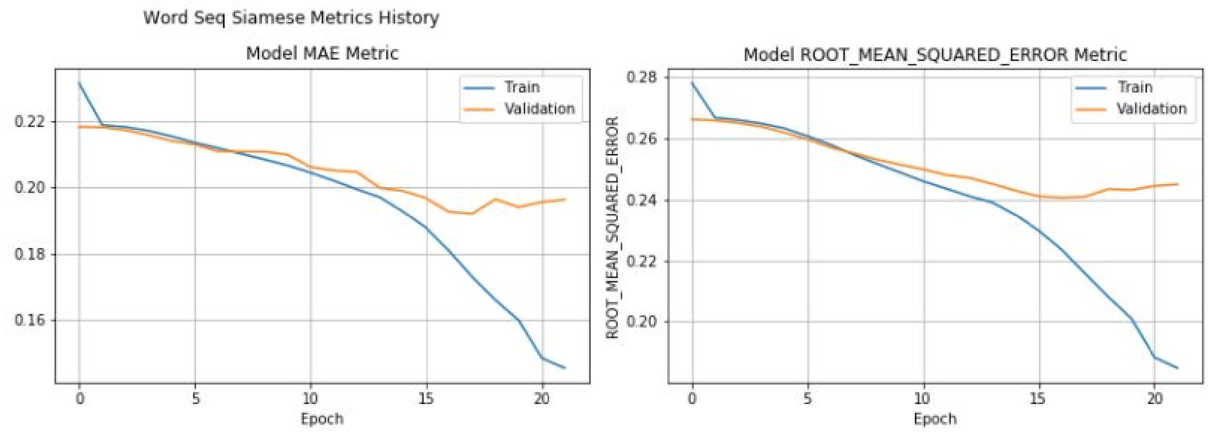
Using Individual models for inputs

Again, we first tried with individual models (one for each input) and combined them with both cosine similarity and exponent manhattan distance.

```
Model: "Example Siamese Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 11)]              0
_________________________________________________________________
embedding (Embedding)        (None, 11, 128)           42372480
_________________________________________________________________
bidirectional (Bidirectional (None, 11, 256)           263168
_________________________________________________________________
conv1d (Conv1D)              (None, 11, 32)            65568
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 3, 32)             0
_________________________________________________________________
```

```
activation (Activation)      (None, 3, 32)              0

__________________________________________________________________

flatten (Flatten)            (None, 96)                 0

__________________________________________________________________

dense (Dense)                (None, 128)                12416
==================================================================

Total params: 42,713,632

Trainable params: 341,152

Non-trainable params: 42,372,480

__________________________________________________________________

Model: "functional_1"

________________________________________________________________________
___________________________

Layer (type)                    Output Shape         Param #
Connected to

========================================================================
===========================

input_2 (InputLayer)            [(None, 11)]         0

________________________________________________________________________
___________________________

input_3 (InputLayer)            [(None, 92)]         0

________________________________________________________________________
___________________________

search_siamese (Functional)     (None, 128)          42713632
input_2[0][0]

________________________________________________________________________
___________________________

description_siamese (Functional (None, 128)          42824224
input_3[0][0]

________________________________________________________________________
___________________________

lambda (Lambda)                 (None, 1)            0
search_siamese[0][0]

description_siamese[0][0]

________________________________________________________________________
___________________________

lambda_1 (Lambda)               (None, 1)            0
search_siamese[0][0]
```

```
description_siamese[0][0]

________________________________________________________________________
___________________________
concatenate (Concatenate)        (None, 2)            0
lambda[0][0]


lambda_1[0][0]

________________________________________________________________________
___________________________
conc_layer (Dense)               (None, 100)          300
concatenate[0][0]

________________________________________________________________________
___________________________
dropout (Dropout)                (None, 100)          0
conc_layer[0][0]

________________________________________________________________________
___________________________
dense_3 (Dense)                  (None, 1)            101
dropout[0][0]
========================================================================
===========================
Total params: 85,538,257

Trainable params: 793,297

Non-trainable params: 84,744,960

________________________________________________________________________
___________________________
```



Test data results:

Word Seq Siamese RMSE: 0.2635100872257146

Word Seq Siamese MSE: 0.06943756606970372

Word Seq Siamese MAE: 0.21107161651310402

Validation data results:

Word Seq Siamese RMSE: 0.2407452175820478

Word Seq Siamese MSE: 0.057958259788627536

Word Seq Siamese MAE: 0.19267459909668394

Train data results:

Word Seq Siamese RMSE: 0.21502652804767838

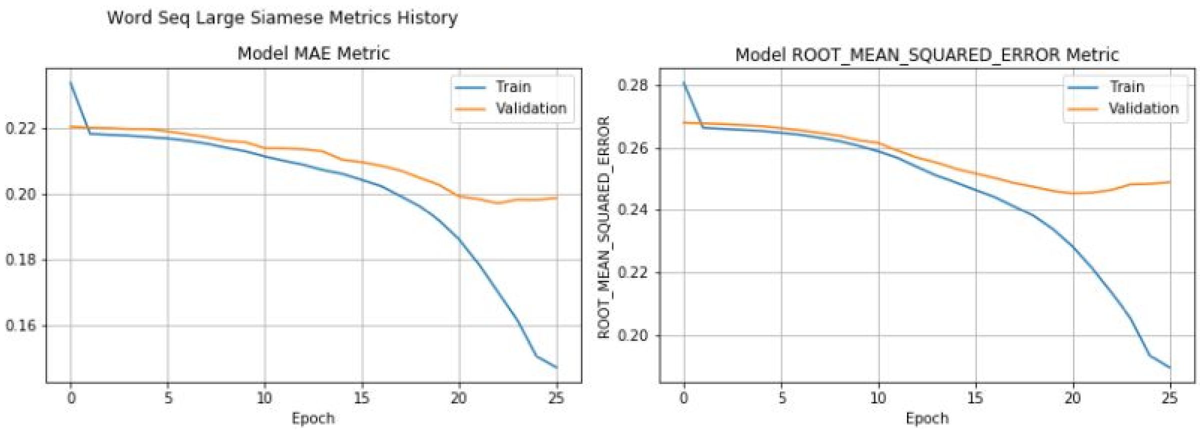
Word Seq Siamese MSE: 0.04623640776423902

Word Seq Siamese MAE: 0.17089774222985007

Now We tried to experiment with larger embedding vector dimensions for each word (128 -> 512) to see if we can get better results.

```
Model: "functional_3"

______________________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #
Connected to
==============================================================================
==========================
input_6 (InputLayer)            [(None, 11)]         0

______________________________________________________________________________
____________________________
input_7 (InputLayer)            [(None, 92)]         0

______________________________________________________________________________
____________________________
search_siamese (Functional)     (None, 128)          170224288
input_6[0][0]

______________________________________________________________________________
____________________________
description_siamese (Functional (None, 128)          170334880
input_7[0][0]
```

```
________________________________________________________________________
__________________________
lambda_2 (Lambda)               (None, 1)            0
search_siamese[0][0]

description_siamese[0][0]
________________________________________________________________________
__________________________
lambda_3 (Lambda)               (None, 1)            0
search_siamese[0][0]

description_siamese[0][0]
________________________________________________________________________
__________________________
concatenate_1 (Concatenate)     (None, 2)            0
lambda_2[0][0]

lambda_3[0][0]
________________________________________________________________________
__________________________
conc_layer (Dense)              (None, 100)          300
concatenate_1[0][0]
________________________________________________________________________
__________________________
dropout_1 (Dropout)             (None, 100)          0
conc_layer[0][0]
________________________________________________________________________
__________________________
dense_6 (Dense)                 (None, 1)            101
dropout_1[0][0]
========================================================================
==========================
Total params: 340,559,569
Trainable params: 1,579,729
Non-trainable params: 338,979,840
________________________________________________________________________
__________________________
```



Test data results:

Word Seq Large Siamese RMSE: 0.2618027554262483

Word Seq Large Siamese MSE: 0.06854068274877598

Word Seq Large Siamese MAE: 0.21221592030787353

Validation data results:

Word Seq Large Siamese RMSE: 0.24560258921785827

Word Seq Large Siamese MSE: 0.06032063183051603

Word Seq Large Siamese MAE: 0.1993638936268168

Train data results:

Word Seq Large Siamese RMSE: 0.22067101074443726

Word Seq Large Siamese MSE: 0.04869569498297154

Word Seq Large Siamese MAE: 0.17764586276827632

Using a true siamese network

Next we tried to train the same model for the two inputs so that the model shared the trained weights for both inputs.

Working with same size words input vector

We tried making the two words input vectors (search terms and descriptions) the same length.

We took the mean description word count as both vector length.

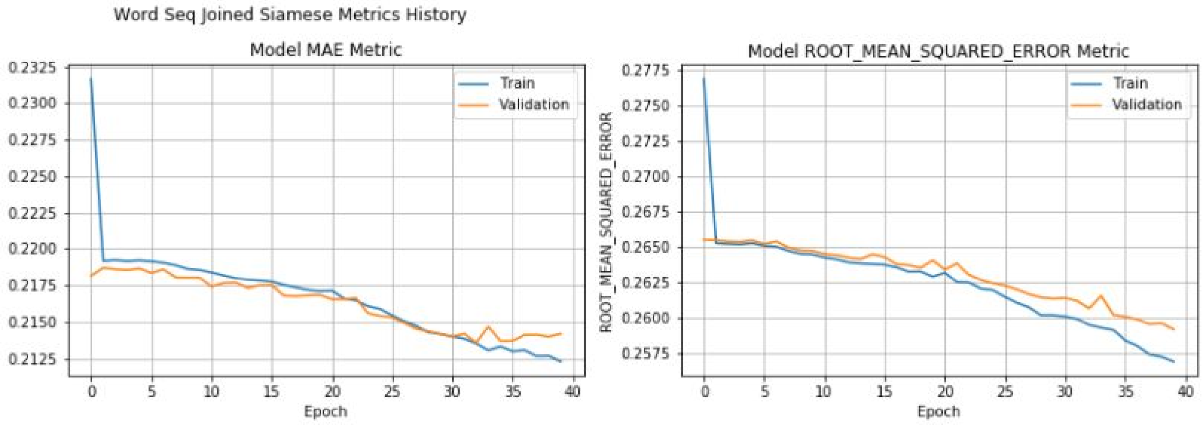
Increasing the Convolutional layers depth and working with common siamese layers

We used common siamese models by using the same layers in order to share the layers weights. Also, we increased the convolutional layers depth and filters (the output from the bi derectional LSTM will be the input for the convolution).

```
Model: "functional_9"

________________________________________________________________________

__________________________

Layer (type)                    Output Shape         Param #
Connected to

========================================================================
==========================

input_11 (InputLayer)           [(None, 92)]         0

________________________________________________________________________

__________________________

input_12 (InputLayer)           [(None, 92)]         0

________________________________________________________________________

__________________________

embedding_5 (Embedding)         (None, 92, 512)      169489920
input_11[0][0]


input_12[0][0]

________________________________________________________________________

__________________________

bidirectional_5 (Bidirectional) (None, 92, 256)      656384
embedding_5[0][0]


embedding_5[1][0]

________________________________________________________________________

__________________________
```

```
bidirectional_6 (Bidirectional) (None, 92, 256)      394240
bidirectional_5[0][0]

bidirectional_5[1][0]
_______________________________________________________________________
___________________________
conv1d_8 (Conv1D)               (None, 92, 128)      98432
bidirectional_6[0][0]

bidirectional_6[1][0]
_______________________________________________________________________
___________________________
conv1d_9 (Conv1D)               (None, 92, 128)      49280
conv1d_8[0][0]

conv1d_8[1][0]
_______________________________________________________________________
___________________________
max_pooling1d_2 (MaxPooling1D)  (None, 30, 128)      0
conv1d_9[0][0]

conv1d_9[1][0]
_______________________________________________________________________
___________________________
activation_2 (Activation)       (None, 30, 128)      0
max_pooling1d_2[0][0]

max_pooling1d_2[1][0]
_______________________________________________________________________
___________________________
flatten_3 (Flatten)             (None, 3840)         0
activation_2[0][0]
```

```
activation_2[1][0]
________________________________________________________________________
__________________________
dense_7 (Dense)                 (None, 128)          491648
flatten_3[0][0]


flatten_3[1][0]
________________________________________________________________________
__________________________
lambda_4 (Lambda)               (None, 1)            0
dense_7[0][0]


dense_7[1][0]
________________________________________________________________________
__________________________
dense_8 (Dense)                 (None, 1)            2
lambda_4[0][0]
========================================================================
==========================
Total params: 171,179,906

Trainable params: 1,689,986

Non-trainable params: 169,489,920
________________________________________________________________________
__________________________
```



Test data results:

Word Seq Joined Siamese RMSE: 0.26387441828132885

Word Seq Joined Siamese MSE: 0.0696297086233097

Word Seq Joined Siamese MAE: 0.21735656760481345

Validation data results:

Word Seq Joined Siamese RMSE: 0.26080213837719046

Word Seq Joined Siamese MSE: 0.0680177553821152

Word Seq Joined Siamese MAE: 0.21418499618998035

Train data results:

Word Seq Joined Siamese RMSE: 0.25811213134864436

Word Seq Joined Siamese MSE: 0.06662187234933985

Word Seq Joined Siamese MAE: 0.21271295076768296

2.d. Feature extraction from the Word Level LSTM model to classical ML algorithms

Again, we tried to use the word LSTM model as a feature extractor for xgboost, catboost and random forest regressor models.

results -

xgboost

Test data results:

xgboost word model RMSE: 0.2687339463187322

xgboost word model MSE: 0.07221793390403923

xgboost word model MAE: 0.2148875198089219

Validation data results:

xgboost word model RMSE: 0.2497396072957575

xgboost word model MSE: 0.06236987145223917

xgboost word model MAE: 0.19896083725477273

Train data results:

xgboost word model RMSE: 0.21074713926888536

xgboost word model MSE: 0.044414356710018955

xgboost word model MAE: 0.16525269131751963

catboost

Test data results:

catboost word model RMSE: 0.2679880387444478

catboost word model MSE: 0.07181758891009567

catboost word model MAE: 0.21428073589958177

Validation data results:

catboost word model RMSE: 0.24914501528708874

catboost word model MSE: 0.06207323864240368

catboost word model MAE: 0.19852621510022464

Train data results:

catboost word model RMSE: 0.2162364992776332

catboost word model MSE: 0.046758223619845864 catboost word model MAE: 0.1701239980507089

random forest

Test data results:

random forest word model RMSE: 0.2946582614149908

random forest word model MSE: 0.08682349102010505

random forest word model MAE: 0.23364442804276161

Validation data results:

random forest word model RMSE: 0.2727456961819896

random forest word model MSE: 0.0743902147857982

random forest word model MAE: 0.21463268438176952

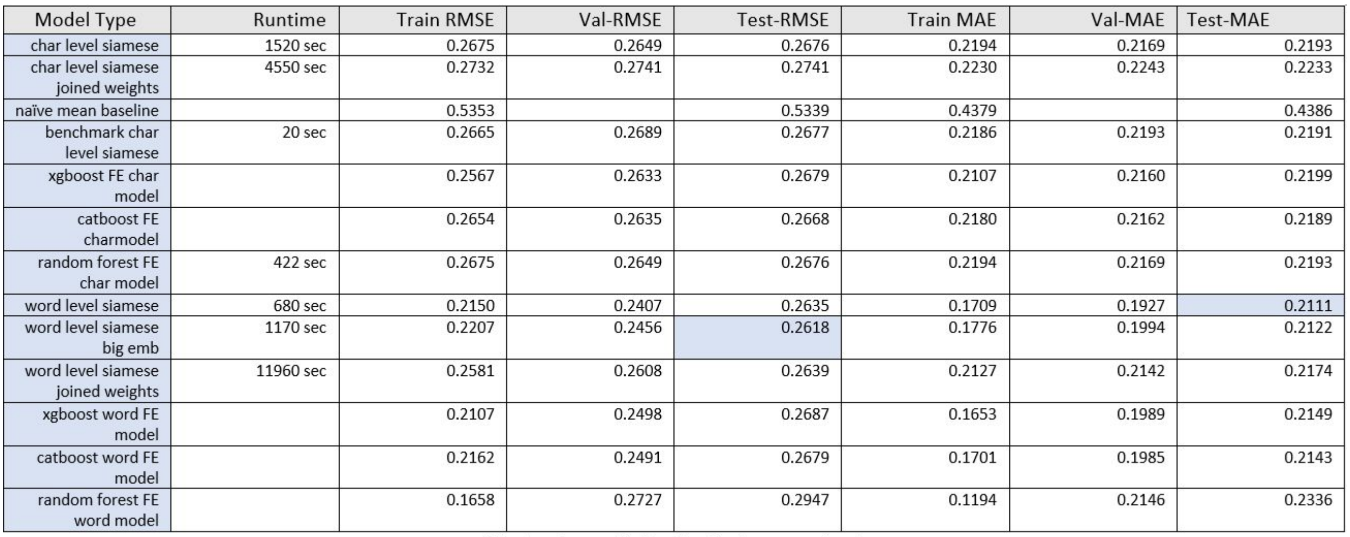
Train data results:

random forest word model RMSE: 0.16579184205926029

random forest word model MSE: 0.027486934893402704

random forest word model MAE: 0.11935689066952868

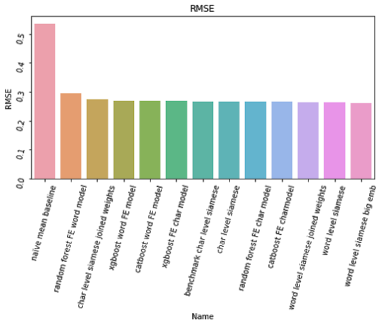
3.

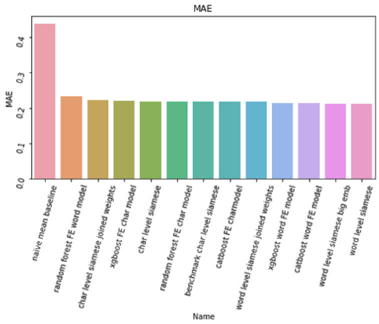


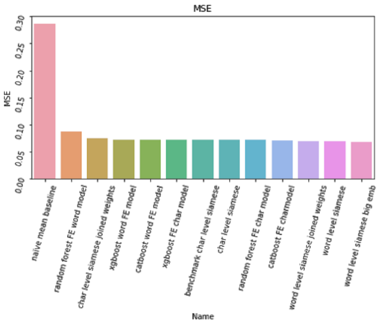
Model Type	Runtime	Train RMSE	Val-RMSE	Test-RMSE	Train MAE	Val-MAE	Test-MAE
char level siamese	1520 sec	0.2675	0.2649	0.2676	0.2194	0.2169	0.2193
char level siamese joined weights	4550 sec	0.2732	0.2741	0.2741	0.2230	0.2243	0.2233
naïve mean baseline		0.5353		0.5339	0.4379		0.4386
benchmark char level siamese	20 sec	0.2665	0.2689	0.2677	0.2186	0.2193	0.2191
xgboost FE char model		0.2567	0.2633	0.2679	0.2107	0.2160	0.2199
catboost FE charmodel		0.2654	0.2635	0.2668	0.2180	0.2162	0.2189
random forest FE char model	422 sec	0.2675	0.2649	0.2676	0.2194	0.2169	0.2193
word level siamese	680 sec	0.2150	0.2407	0.2635	0.1709	0.1927	0.2111
word level siamese big emb	1170 sec	0.2207	0.2456	0.2618	0.1776	0.1994	0.2122
word level siamese joined weights	11960 sec	0.2581	0.2608	0.2639	0.2127	0.2142	0.2174
xgboost word FE model		0.2107	0.2498	0.2687	0.1653	0.1989	0.2149
catboost word FE model		0.2162	0.2491	0.2679	0.1701	0.1985	0.2143
random forest FE word model		0.1658	0.2727	0.2947	0.1194	0.2146	0.2336

The best results for the test are marked

Statistics Summary







Conclusion And Future Work

Our Learning Process

In this assignment we learned how to use siamese networks in order to determine similarity between word/character sequences. We learned how to perform two forms of word/character tokenization (by using the keras Tokenizer class and the CountVectorizer class). We also learned how to perform text analysis, preprocessing(splitting, stemming and stop words removal) and using the word2vec models in order to create predefined embedding layers with vectorized representations for words in corpus. We used feature extraction on pretrained models and combined them with a few ML algorithms such as XGBoost, CatBoost and Random Forests.

Suggestions for further improvement

We can suggest a few ways in order to improve our model and overcome the overfitting that occurs while training:

- **Classification instead of regression -** since the relevance values (Y) are discrete in the range [1, 3] we could view the problem as a classification problem.
  Instead of trying to use regression to produce the relevance value associated with the search term and the description,
  we could try to classify the similarity between them to one of the unique relevance values in our dataset.
- **Increase the representation of word/character vectors -** we used the mean value of the word/character count as the size of the sentence representation vector in order to avoid exploding runtime.
  By doing that we may have lost information regarding the rest of the sentence for sentences that are longer than the mean value.
  We could represent all of the sentences as the maximum value or increase it to some value instead of the mean to avoid losing data.(down side is it will cause also for some sentences to have more zero padding at the end of each sentence, can cause the model to converge slower).

- **Use better/more suited tokenization and word processing methods -** we used a pretty simplistic sentence/word tokenization techniques (stemming, lowercasing etc.) Going forward we could try and use tokenization techniques that take the context of the sentence into account in order to better determine similarity.
  We could also try and explore tokenization without using lowercasing or with using lemmenization instead of stemming. Another option is to keep the stop words in our corpus to better preserve context. Maybe we should keep some of the punctuation in the sentence to preserve some of the original meaning of tokenize by spaces and not by punctuation altogether. It is also a possibility that the preprocessing of the sentence was too aggressive due to the stemming and stop words removal process and this will result in a lower quality embedding since we lose some connections - we could try different types of sentence preprocessing.
- **Increase Embedding dimensions -** We could try using an even larger word vector representation to preserve more sentence data con context in the embedding vectorized word representation.
- **Exploring better similarity functions -** We tried using two different similarity functions (Exponent manhattan distance and Cosine similarity).
  We could try and explore more similarity metrics that may be better suited to our problem.
- **Networks architecture improvements suggestions -**
  - Add LSTM hidden nodes to better preserve history.
  - Add Dense/Conv layers.
  - Add more dropout to avoid overfitting to training data.
  - Try different activations/loss functions (if converted to classification tasks we could use categorical cross entropy instead of RMSE and softmax activation on the last layer instead of sigmoid).