Assignment 3 – Kaggle home depot challenge

Preface

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.

In this competition, 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 (rated by values between 1 to 3) 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.

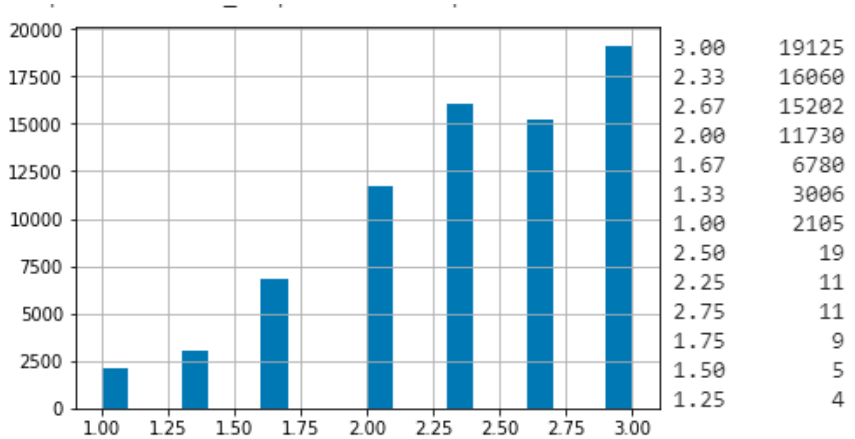
Our challenge is to predict a relevance score for the provided combinations of search terms and products.

In this report we will describe our solution to this task by using mainly Siamese NN (using Mahathir distance) and using the internal layers for feature extracting with classic model.

The main conclusions we have got from this task is the importance of understanding the data in order to choose the right validation strategy, how preprocess can improve results of NLP tasks, the limitation of embedding texts, and we were also exposed to feature engineering we can do on texts such as statistics of word appearance in sentence, length, and some more.

first step- explore the data

- Relevance distribution



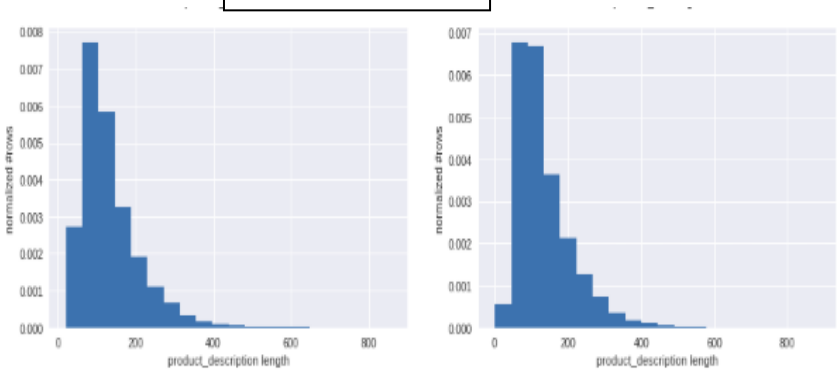
3.00	19125
2.33	16060
2.67	15202
2.00	11730
1.67	6780
1.33	3006
1.00	2105
2.50	19
2.25	11
2.75	11
1.75	9
1.50	5
1.25	4

The train relevance rate as we can notice there were significant number of rows that voted with rate above 2 than below.

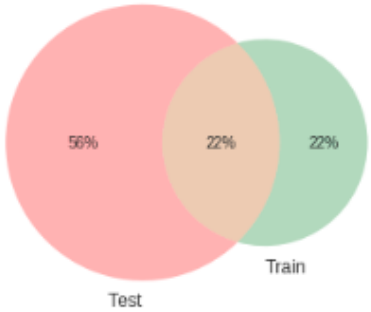
- We found that search term consists many spelling and typing mistakes.

- The data almost doesn't contain null values. (only the features data set)
- We found for search term and product description the following statistics: maximining of length, frequency of presence of digits and punctuations how many time each of them appear and examples for each punctuation. During the assignment we found this step as very useful to decide how to clean the text in the preprocess step.
- Some additional findings: uppercase & lowercase, description consist html tags, sometimes there are 2 or more words without space (but it is not often).
- Explore the similarity between the train and test sets – this step we could do because this is Kaggle competition and we know precisely how the test set look like. We can create a data set like this one by dividing our 'train set' to test set in addition to validation we use to evaluate our models.

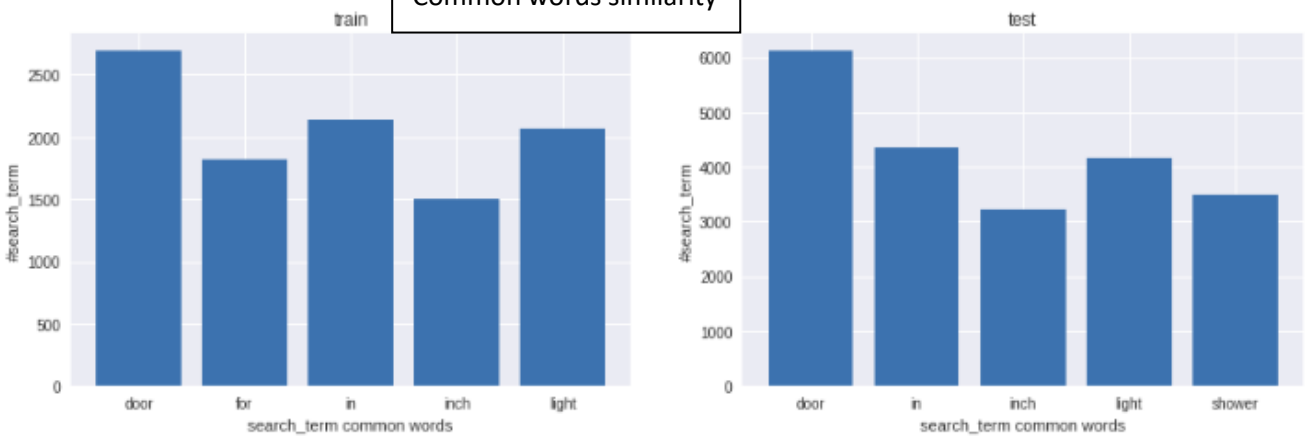
Length similarity.



22% of product uids are common to both of the data sets



Common words similarity



Q1

a.We did the same preprocessing that we did for the word level lstm.

We ended up with 68 distinct characters (without preprocessing there are 94 distinct characters)

For the length of the teams we decided to take the mean of each category (search term and product description) we think that this is a good balance between too sparse and loosing to much from the terms.

Our validation strategy is the same as Q2 .

Architecture :

```
model = siamese()
model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
term_input (InputLayer)      (None, 14, 1)             0
_________________________________________________________________________________________________
product_input (InputLayer)   (None, 586, 1)            0
_________________________________________________________________________________________________
shared_model (Sequential)    (None, 128)               28628       term_input[0][0]
                                                                   product_input[0][0]
_________________________________________________________________________________________________
prediction (Lambda)          (None, 1)                 0           shared_model[1][0]
                                                                   shared_model[2][0]
=================================================================================================
Total params: 28,628
Trainable params: 28,628
Non-trainable params: 0
_________________________________________________________________________________________________
```

We used sheared weights because it gave better results for the shared weights and hence, we decided to use this strategy. The model build from two dense layer followed by one layer of LSTM. We chose Manhattan distance to measure the gap between the search term and the description.

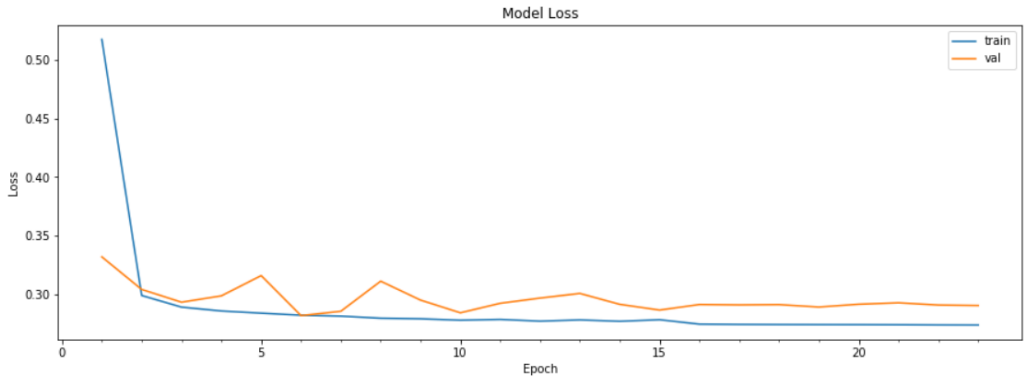
Results:

Model type	runtime	Train RMSE	Val-RMSE	Test-RMSE	Train MAE	Val-MAE	Test-MAE
_final_char_leve_emmbedings	6.540339068571726	0.523769	0.539189	0.534762	0.432266	0.432609	0.436270
6.540339068571726	_final_char_leve_emmbedings	0.523769	0.539189	0.534762	0.432266	0.432609	0.436270
mean	0	0.531591	0.544169	0.537423	0.442231	0.439502	0.441831
median	0	0.583851	0.647295	0.608532	0.439055	0.495352	0.457978
XGBoost - feature extracted	60	0.505233	0.533758	0.534254	0.420006	0.428764	0.437846
GradientBoostingRegressor - feature extracted	2	0.527753	0.542829	0.536493	0.438080	0.437583	0.440069
RandomForest - feature extracted	5	0.280962	0.544745	0.556197	0.211891	0.436775	0.451630
_char_level_depper	4.407435746987661	0.523109	0.538836	0.534139	0.431642	0.432261	0.435514

The first marked model was with 50 hidden lstm layers and only one dense

The second one used 80 lstm hidden layers and two dense

There is a small improvement so we think that adding more layers will improve the score much more.



The histogram show good converge and relative small diff between the train and the validation .

C – naïve Benchmark

We tried to use the test median and the mean .

	mean	0	0.531591	0.544169	0.537423	0.442231	0.439502	0.441831
	median	0	0.583851	0.647295	0.608532	0.439055	0.495352	0.457978

The mean yield good validation mae and rmse

D – transfer learning

We used xgboost, gradient boosting and random forest

the xgboost regressor managed to improve our score !

Model type	runtime	Train RMSE	Val-RMSE	Test-RMSE	Train MAE	Val-MAE	Test-MAE
_final_char_leve_emmbedings	6.540339068571726	0.523769	0.539189	0.534762	0.432266	0.432609	0.436270
6.540339068571726	_final_char_leve_emmbedings	0.523769	0.539189	0.534762	0.432266	0.432609	0.436270
mean	0	0.531591	0.544169	0.537423	0.442231	0.439502	0.441831
median	0	0.583851	0.647295	0.608532	0.439055	0.495352	0.457978
XGBoost - feature extracted	60	0.505233	0.533758	0.534254	0.420006	0.428764	0.437846
GradientBoostingRegressor - feature extracted	2	0.527753	0.542829	0.536493	0.438080	0.437583	0.440069
RandomForest - feature extracted	5	0.280962	0.544745	0.556197	0.211891	0.436775	0.451630
_char_level_depper	4.407435746987661	0.523109	0.538836	0.534139	0.431642	0.432261	0.435514

Q2 – word level lstm

We chose to use word level processing to try to predict search relevance.

Preprocess we did in the data:

- Uniform all the measurements words. (inches, inch, in -> in)
- Remove stopwords.
- Stemming the text.
- Feature engineering - we had the following statistics:
  1) Add brand name feature from features set.
  2) How often the search term words exist in product title, description and in the brand name.
  3) Lengths of product title, description and brand name features.
- Remove html tags from description
- Decided to remove some punctuations, we removed different punctuations for each of the features. (for example, '*' and '.' we didn't remove because there is meaning of measurement and end of sentence respectively. (*more details about our decision you can see in the exploration.pynb and in preprocess segment in the second notebook*)
- Convert all the text to lowercase.
- Fix spelling and typing mistakes in search term.
- Concatenate between product title and description.

Create dictionary:

We created dictionary by using tokenizer. Each text feature was converted to index vector. The length of index was chosen to be around the median length for each feature.

Validation strategy:

we tried many different validation sets to create the validation set in such a way that describe the best the test set.

- as we noticed at the *exploration* part by the Venn diagram, there is 22% of the products that exist in both the train and test set, and the others exist only in one of the sets. we wanted to retain this percentage between the train and validation sets as well.
- we also notice that the relevance in the train set is not balanced, there are many rows that voted with ~2.5 and few the voted ~1.5 so we decided that it will be clever to split the train and validation and save this proportion of votes. (like stratified fold do)

Architecture:

As we requested we form an Siamese model, we tried two different types of model, one with separate weights for the search term and description and the other with shared weights, we have got better results for the shared weights and hence we decided to use this model. The model build from two dense layer followed by one layer of LSTM. We chose Manhattan distance to measure the gap between the search term and the description.

We tried to train the model with and without embeddings, we have got the following results for the Siamese models we trained:

	Model type	runtime	Train RMSE	Val-RMSE	Test-RMSE	Train MAE	Val-MAE	Test-MAE
0	siamese using Google embeddings	21	0.531161	0.536808	0.53657	0.437928	0.432741	0.43925
1	siamese using Glove embeddings	21	0.532119	0.539829	0.537811	0.438348	0.434985	0.439731
2	siamese using not pretrained embeddings	12	0.532115	0.535606	0.534717	0.438177	0.430251	0.437226
3	siamese model without emb	20	0.530884	0.537128	0.534985	0.436286	0.431503	0.436756
4	siamese model without emb - add brand feature	19	0.525778	0.53476	0.53044	0.428695	0.428168	0.429952
5	siamese model without emb - add brand and statistics features	33	0.481516	0.492714	0.490126	0.390827	0.399148	0.396747

We first try to find the best representation for the text features, so we built the first 4 models, our goal was to find the best model to continue with. To choose this one we examined 2 characteristics mainly: the validation loss (rmse and mae) and the distribution of the relevance in the prediction for the validation set, we wanted to take the model with the lowest loss and the most similar distribution to the real relevance values for validation set.

We found that the best model to use is the one without embeddings. (this is probably because many words were not existing in the corpus of google and glove). After we found this model we gradually add more features to find if it is improving the model and we found that it is.

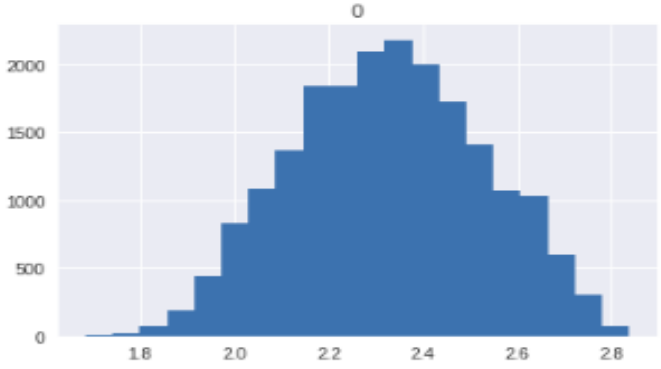
Some notes:

- for some of the models we used sgd optimizer and for others adam. In order to find the best fit for each one.
- The embedding layer we made included words that was not in the google/glove corpus, and initialized these vectors with random values in order it will converge to a significant value while training.

Analyze our best Siamese model (indexed 5 in the chart above):

As we can see there is a small overfitting, from epoch 15 the train set continues to converge slowly while the validation set doesn't improve. Hence, we used earlystooping callback, so we took the weight just before epoch 15.

```
min rate prediction: 1.6833518743515015
 max rate prediction: 2.8378372192382812
```



We can notice the the distribution of the relevance is not include all the relevance values. But on the other hand only few samples were with relevance < ~1.6 . and because we will use this model as feature extraction we decided that this is good enough for now.

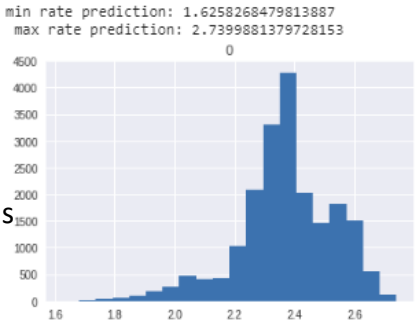
Feature extraction

- We used the last dense layer as feature extraction.
- We tried three different models
  - XGBoost
  - GradientBoostingRegressor
  - RandomForest
- We have got the following results:

	Model type	runtime	Train RMSE	Val-RMSE	Test-RMSE	Train MAE	Val-MAE	Test-MAE
0	siamese using Google embeddings	21	0.531161	0.536808	0.536570	0.437928	0.432741	0.439500
1	siamese using Glove embeddings	21	0.532119	0.539829	0.537811	0.438348	0.434985	0.439731
2	siamese using not pretrained embeddings	12	0.532115	0.535606	0.534717	0.438177	0.430251	0.437226
3	siamese model without emb	20	0.530884	0.537128	0.534985	0.436286	0.431503	0.436756
4	siamese model without emb - add brand feature	19	0.525778	0.534760	0.530440	0.428695	0.428168	0.429952
5	siamese model without emb - add brand and stat	33	0.481516	0.492714	0.490126	0.390827	0.399148	0.386747
6	XGBoost - feature extracted	37	0.490643	0.510963	0.508493	0.406271	0.413724	0.417046
7	GradientBoostingRegressor - feature extracted	4	0.500899	0.519252	0.510057	0.411102	0.416861	0.414542
8	RandomForest - feature extracted	11	0.211239	0.509951	0.514343	0.166157	0.415349	0.422507

For the last two models we have got almost the same results, unfortunately we didn't success to improve our best benchmark.

- All the three models predict with almost
  The same distribution, we can notice that
  The gap of values is almost the same as before,
  But the distribution here is skew left. Which means more rows predicted with higher rate.



Our best result was: 0.49 RMSE on test set. 😊

What else can we do ?

- Extract more relevant attributes from the features set that describe the product, such as measurement (length x width).
- Try to change the length of the vector that describe the product and the search term, maybe it's too noisy. (too much padding or on the other hand describe to text very poorly because pruning)
- Hyper parameter.
- Try to extract data from prior layer of the nn. (we took from the last one)
- *Extract more statistics feature* – we saw it gave us great improvement (from 0.53 -> 0.49)
- Try more classic model and use stacking to create one stronger ensemble model.

What we learnt from the process?

- Importance of exploring data to create good validation set.
- Using pretrained embedding is limited because texts are very diverse and don’t always contain words from the embedding corpus.
- Using another optimizer than Adam. (in some cases, we have got better results with SGD)
- We were first exposed to Siamese architecture and its uses