SURVEY OF MACHINE LEARNING

Spring 2020

Predicting Question Similarity and Topics of Quora Questions using Deep Learning and Latent Dirichlet Allocation

Apoorva Sonavani
Harsh Singal
Krithika Jayaraman
Sagar Sahoo

MS Business Analytics

Abstract

In the current world, the rate at which questions are being asked in online platforms is exceeding the capacity of qualified people to answer them, and many of the questions are repetitive. Identifying similar questions could enable businesses answer customer queries much more efficiently. Quora is a platform where in users ask questions to people with quality insights and answers. Quora gives importance to similar questions problem because it wants to provide a unique experience for both the user asking the question and the user answering the question. Through this work we focused on two things. First, checking the similarity of questions using deep learning. The main idea was to take up the questions, pre-process, tokenize and then vectorize the questions. We then used pre-trained GLOVE word embeddings trained on 6 Billion corpus and represented in 300 dimensions to represent the vectors. Then we fed the vectors to LSTM and Dense layers and predicted the similarity of question pair. The second part of the project comprises an application of Latent Dirichlet Allocation to categorize the Quora Question corpus into topic clusters.

Through our experiment we achieved a training accuracy of 89% for the question similarity and classified the question corpus into 10 topic clusters.

Keywords: *NLP, LDA, Question Similarity, Deep Learning, Word Embeddings*

Approach

The problem this work addresses is: Given a pair of Quora questions, identify whether they have the same intent and in order to tackle this problem we applied Natural Language Processing techniques to classify a given question pair as similar or not. And then identify the possible topics that can be assigned to the questions.

About the Data

The data consists of about 404351 training examples having the following format:

- id - the id of the question pair
- qid1, qid2 - unique ids of each question
- question1, question2 - the full text of each question
- is_duplicate - the target variable, set to 1 if question1 and question2 have essentially the same meaning, and 0 otherwise.

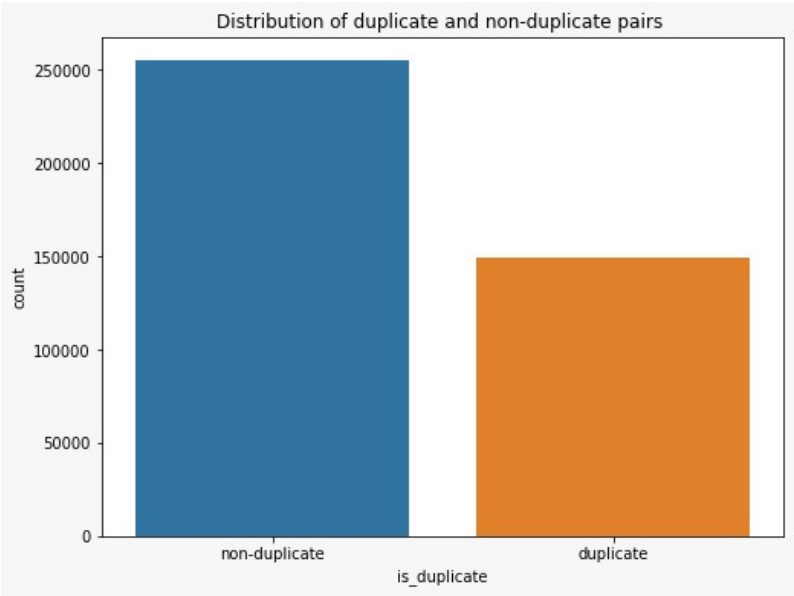

Figure: Class Distribution

Part I - Question Similarity

Traditional Machine Learning algorithms cannot work with text directly. Instead they should be converted into vector of numbers. One of the traditional popular feature extraction methods used is the Bag of Words approach.

The BOW approach involves collecting the data, designing the vocabulary, creating the document vectors (using occurrences or popular TF-IDF score). Here, documents are described by word occurrences while completely ignoring the relative position information of the words in the document. TF-IDF is a weighting technique which allows us to normalize the count/frequency of each word which we obtained from the bag-of-words representation.

The details of TF-IDF vectorization is explained below:

- Term Frequency (TF) is computed by dividing the number of times a token occurs in a document (in our case it is a Quora question) by the total number of tokens in the document. This division by the document length prevents a bias towards longer document by normalizing the raw frequency of the term into a comparable scale.
- Inverse Document Frequency (IDF) is computed by taking the logarithmic of the total number of documents in the corpus divided by the number of documents where the term occurred. This normalization is to up-weigh the rare terms in the corpus.

The overall mathematical interpretation of TF-IDF vectorization is

$$tfidf_{t,d} = f_{t,d} * inv f_{t,d}$$

$$= f_{t,d} * \left(\log \frac{M}{df_t}\right)$$

where

- the term frequency $f_{t,d}$ counts the number of occurrences of term *'t'* in document '*d*'.
- the document frequency $df_{t,d}$ counts the number of documents '*d'* that contain the term '*t'*.
- M is the total number of documents in the corpus. Our corpus here contains the questions from 'Quora'.

The TF-IDF value grows proportionally to the occurrences of the word in the TF, but the effect is balanced by the occurrences of the word in every other document (IDF). Therefore, here by applying a TF-IDF Vectorizer, we are creating a matrix that ranks the importance of each word in the document weighed by their frequency.

The next step is the creation of word embeddings i.e. a representation of text where similar meaning words have a similar representation. Each word is mapped to one vector and the values of the vector are learned using a model. The key objective is to use dense distribution instead of sparse data.

Now, learning the word embeddings requires a lot of text i.e. millions and billions of words. It is common to use pre-trained word embeddings available for free for further re-use.

Pretrained word embeddings capture the semantic and syntactic meaning of a word as they are trained on large datasets. They are capable of boosting the performance of an NLP model. Learning word embeddings from scratch is a challenging task because of sparsity of training data and large number of parameters to train.

There are 2 popular pre-trained word embeddings being used – Google's **Word2Vec** and Stanford's **GloVe.** In our work, we have Stanford's GloVe embeddings which was trained on 6 Billion examples and each word is represented in 300 dimensions.

As per Stanford sources, “GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.”

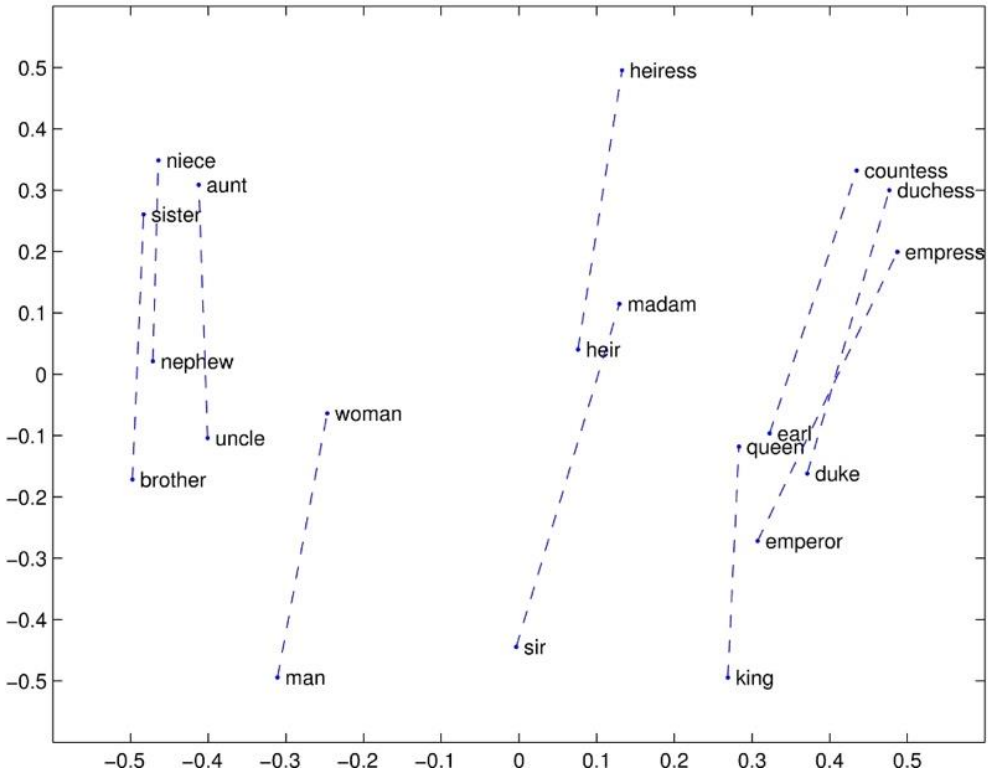

Figure: Sample GloVe Representation of “man-woman”

Now the embedding matrix was set for both the questions of the training dataset. Each question is separately fed into LSTM Layers. LSTMs are specific type of Recurrent Neural Networks which include a ‘memory cell’ that can maintain information in memory. Gates are used to control when information enters the memory, outputs it and its forgotten. This architecture lets them learn longer-term dependencies.

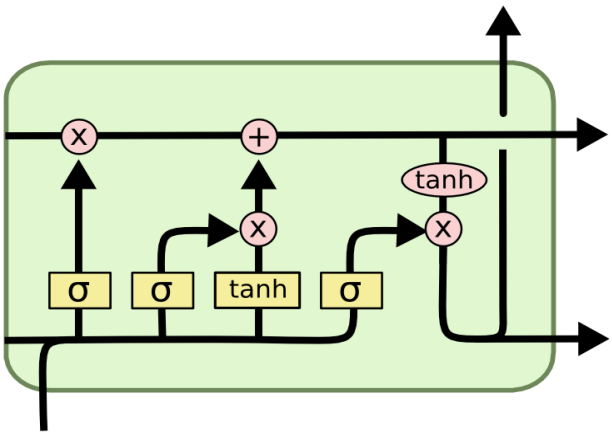

Figure: LSTM Network

The output of the LSTM Layers of both the questions were used to calculate the vectorized distance. This vectorized distance is fed into Dense Neural Networks and the output layer is a Dense layer with one hidden unit and activation as sigmoid function. This way the model would give out a binary response for every training instance. This model can then be further used for predictions. Next, in order to predict the topics for the questions we use Topic Modelling.

Part II - Topic Modelling

Topic modelling is an unsupervised class of text analysis approach that revolves around the process of capturing latent patterns in data. It is otherwise called as cluster analysis or latent sematic analysis.

The main difference between Topic modelling and the conventional cluster analysis in machine learning (k-Means) is that Topic Modelling is a more sophisticated iterative Bayesian Technique which assigns a probability for each document to be associated with a theme or topic. Here 'document' refers to a sequence of text.

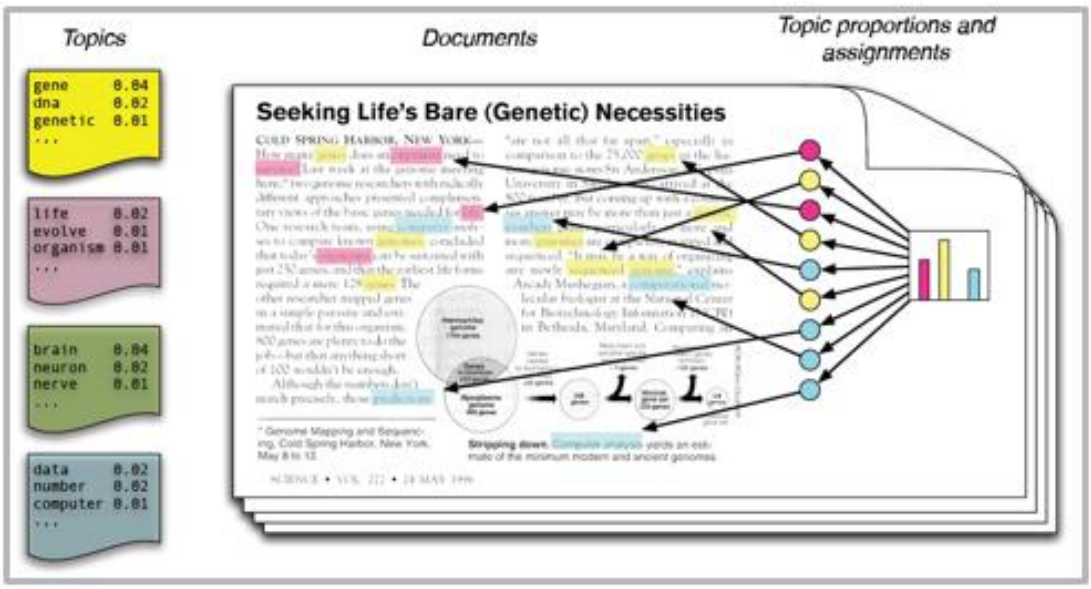

Figure: An example of LDA Topic Model

The above example illustrates how each chunk of text in a document is assigned to a topic. The coloured boxes on the left contain the topics that the model has identified and the words in each box have an educated guess about how they are coherent with the theme. The numbers next to the words correspond to the probability that the word belongs to the theme identified.

One of the commonly used techniques for Topic Modelling is 'Latent Dirichlet Allocation' aka LDA.

Latent Dirichlet Allocation(LDA) is a generative probabilistic model wherein the algorithm the algorithm assumes that each document(sequence of text) consists of a combination of several topics and each token (or word) can be associated to a particular topic with a probability. LDA is chiefly a matrix factorization technique. Matrix Factorization means decomposing a matrix into a product of two lower dimensional rectangular matrices. In case of text processing, the matrix that we will be decomposing is the 'Document-term Matrix".

Once the pre-processing is done, it is important to convert the text into machine-readable numerical representation. In our model, we have used a 'bag-of-words' representation for the questions corpus which resulted in a sparse representation of word counts i.e. how many times a given word in a question appears in the entire question corpus. This is essential because while using the LDA model, the algorithm will evaluate in how many different topics can a word be associated based on the bag-of-words and document-term matrix.

The LDA algorithm is summarized below in context of our problem statement:

Suppose we initialize the number of topics as 10.

1) For each document, randomly initialize each token to one of the K topics that are coming from the topic distribution.
2) For each token (or word) in the document, we compute P (Topic |Document) which is the proportion of words in Document D that is assigned to Topic T.
3) The algorithm then computes P (Word | Topic) which is the proportion of assignments to Topic T over all the documents having the word W.
4) Re-assign word W to topic T with probability P(T|D) *P(W|T) after considering all other words and their topic assignments.

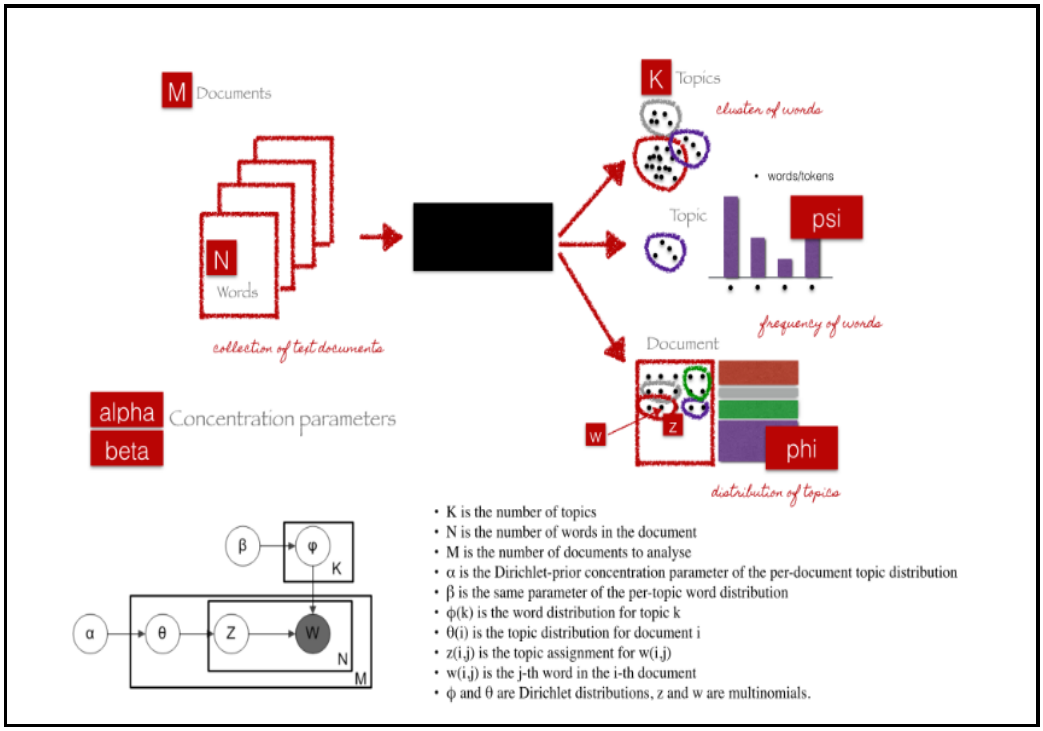

Figure: LDA Topic Modelling Flow

Experiment and Results

In order to conduct the experiment for Question Similarity, we used *Keras/TensorFlow* to setup the environment. The data set was loaded into the environment, and then pre-processed. The GloVe embedding was sourced, and the LSTM layers were built in order to learn the similarity of questions. For Topic Modeling a plot was analysed for coherence score vs number of topics. From the plot, the number of topics was taken as 10 to train the model. To execute the LDA model, we used the *gensim* package and its dependencies.

Architecture

Given below is the model architecture of the LSTM RNN.

Figure 1: LSTM layers

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 25)           0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 25)           0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 25, 300)      28949400    input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 128)          219648      embedding_1[0][0]
                                                                 embedding_1[1][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 1)            0           lstm_1[0][0]
                                                                 lstm_1[1][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 16)           32          lambda_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 16)           0           dense_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 16)           64          dropout_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            17          batch_normalization_1[0][0]
==================================================================================================
Total params: 29,169,161
Trainable params: 219,729
Non-trainable params: 28,949,432
```

Model Parameters

The model was run for 10 epochs, with the loss function as “binary_crossentropy” and the optimizer as “adam”. The scoring metric was “accuracy”.

Model Evaluation

We achieved a training accuracy of 89% and validation accuracy of **84%.** Below are the plots showing the pattern of accuracy across the epochs.

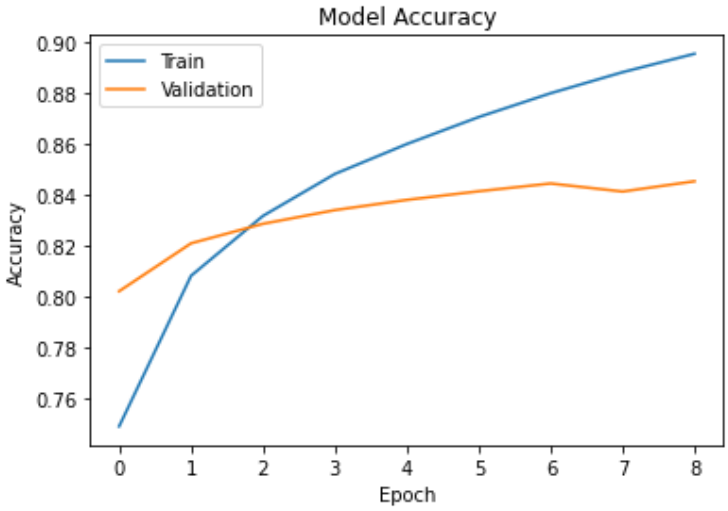

Figure: Model Accuracy

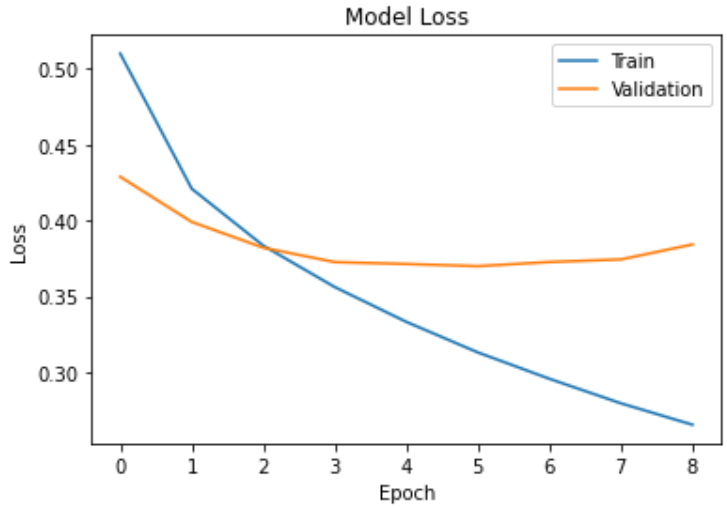

Figure: Model Loss

Now, having predicted the question similarity using Deep Learning, we used LDA to predict the Topics for the Questions.

The model extracted topics in form of the following output –

```
Topic: 0
Words: 0.039*"earn" + 0.033*"invest" + 0.033*"cost" + 0.028*"averag" + 0.024*"earth" + 0.024*"got" + 0.023*"talk" +
0.021*"drink" + 0.021*"degre" + 0.020*"america"
Topic: 1
Words: 0.054*"500" + 0.036*"black" + 0.034*"clinton" + 0.034*"math" + 0.032*"rupe" + 0.032*"hillari" + 0.029*"ban" +
0.027*"date" + 0.025*"unit" + 0.022*"affect"
Topic: 2
Words: 0.054*"facebook" + 0.033*"hair" + 0.031*"food" + 0.027*"email" + 0.025*"password" + 0.022*"move" + 0.022*"bod
i" + 0.020*"face" + 0.019*"gmail" + 0.019*"charg"
Topic: 3
Words: 0.044*"car" + 0.028*"chines" + 0.023*"review" + 0.022*"base" + 0.022*"seri" + 0.021*"someth" + 0.021*"part" +
0.020*"network" + 0.020*"never" + 0.020*"mind"
Topic: 4
Words: 0.077*"trump" + 0.049*"univers" + 0.049*"donald" + 0.035*"win" + 0.033*"major" + 0.028*"com" + 0.022*"stori" +
0.020*"open" + 0.019*"project" + 0.018*"star"
Topic: 5
Words: 0.044*"interest" + 0.039*"watch" + 0.024*"home" + 0.021*"avail" + 0.021*"sentenc" + 0.019*"wrong" + 0.019*"ind
ustri" + 0.019*"relat" + 0.019*"bangalor" + 0.018*"pro"
Topic: 6
Words: 0.143*"quora" + 0.055*"stop" + 0.030*"even" + 0.028*"type" + 0.028*"die" + 0.023*"parent" + 0.022*"post" + 0.0
19*"user" + 0.019*"today" + 0.019*"appl"
Topic: 7
Words: 0.059*"lose" + 0.034*"men" + 0.026*"attract" + 0.025*"everi" + 0.023*"reduc" + 0.022*"fat" + 0.022*"salari" +
0.021*"mechan" + 0.018*"machin" + 0.018*"fast"
Topic: 8
Words: 0.042*"android" + 0.024*"remov" + 0.023*"space" + 0.022*"photo" + 0.021*"wear" + 0.020*"fact" + 0.018*"well" +
0.015*"charact" + 0.014*"origin" + 0.014*"care"
Topic: 9
Words: 0.044*"war" + 0.042*"sex" + 0.032*"top" + 0.028*"idea" + 0.028*"social" + 0.023*"current" + 0.022*"pakistan" +
0.021*"favorit" + 0.019*"girlfriend" + 0.018*"dream"
Topic: 10
Words: 0.030*"build" + 0.026*"god" + 0.024*"file" + 0.024*"polit" + 0.022*"alway" + 0.020*"gain" + 0.019*"allow" + 0.
018*"hate" + 0.017*"group" + 0.016*"calcul"
```

Figure: Topics Extracted from the model

An interactive visual was created to observe the most relevant terms in each of the 10 topics which looks as follows –

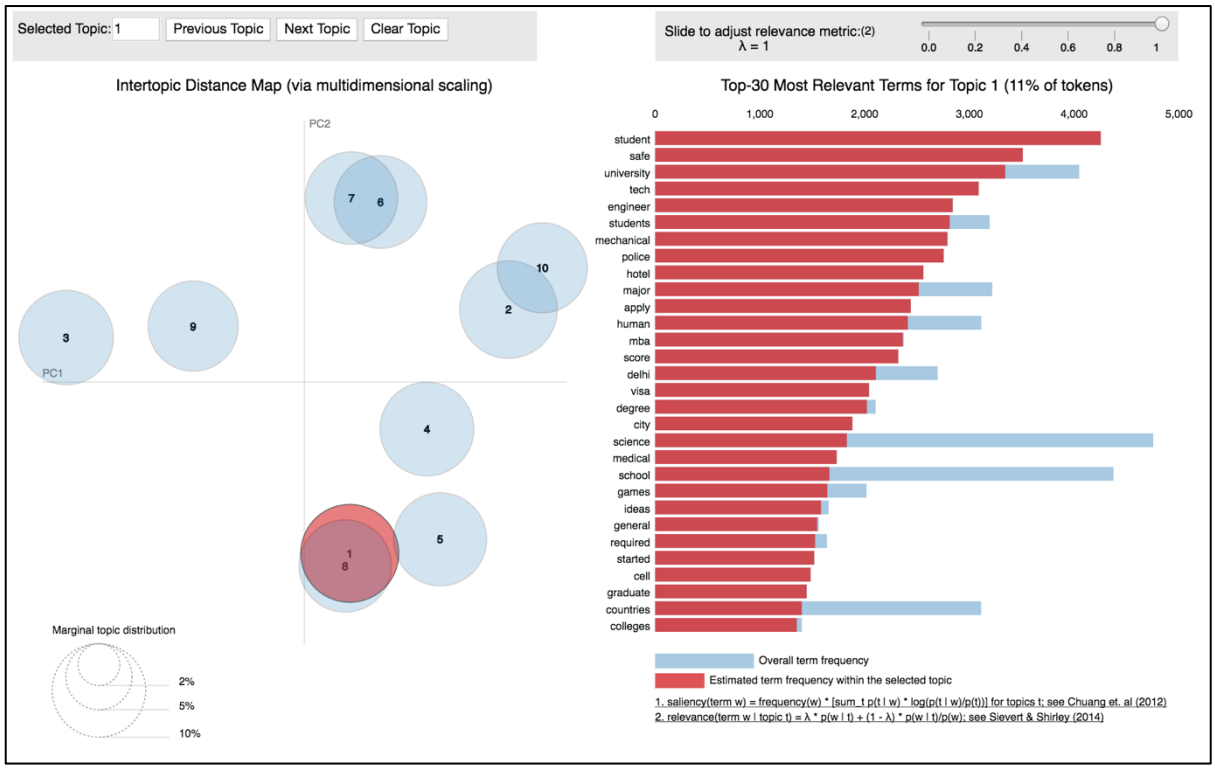

Figure: Interactive Visual to observe the extracted topics

The plot on the left is an inter-topic plot showing the categories of various topics using Multidimensional Scaling, and the plot on the right shows the 30 most relevant terms for topics selected (e.g.: in this case, Topic 1) in the decreasing order. We can decipher from the words that the probable topic is related to 'education'.

Conclusion

Through this work, we used word embeddings through LSTM and dense neural networks to predict the question similarity. As well, we used LDA to assign topics to the questions. These state of art models can be used in customer service platforms to quickly help customer care executives to find user queries previously answered and assign topics/tags for proper request categorization.

Bibliography

- Doig, C. (n.d.). Github. Retrieved from http://chdoig.github.io/pytexas2015-topic-modeling/#/.
- Dataset retreived from https://www.kaggle.com/c/quora-question-pairs
- Mael, F. (n.d.). Github. Retrieved from • https://maelfabien.github.io/machinelearning/NLP_2/#2-term-frequency-inverse-document-frequency-tf-idf.
- Olah, C. (n.d.). Github. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
- GloVe Embeddings Retreived from https://nlp.stanford.edu/projects/glove/

- Question Similarity

  https://pdfs.semanticscholar.org/ff89/c3925581c9551323e9fd7deff699fa149dcb.pdf

- Duplicate Quora Questions Detection by Lei Guo, Chong Li and Haiming Tian

  https://pdfs.semanticscholar.org/4c19/2b8f45b1e913ee7da32624cd7559eccb0890.pdf