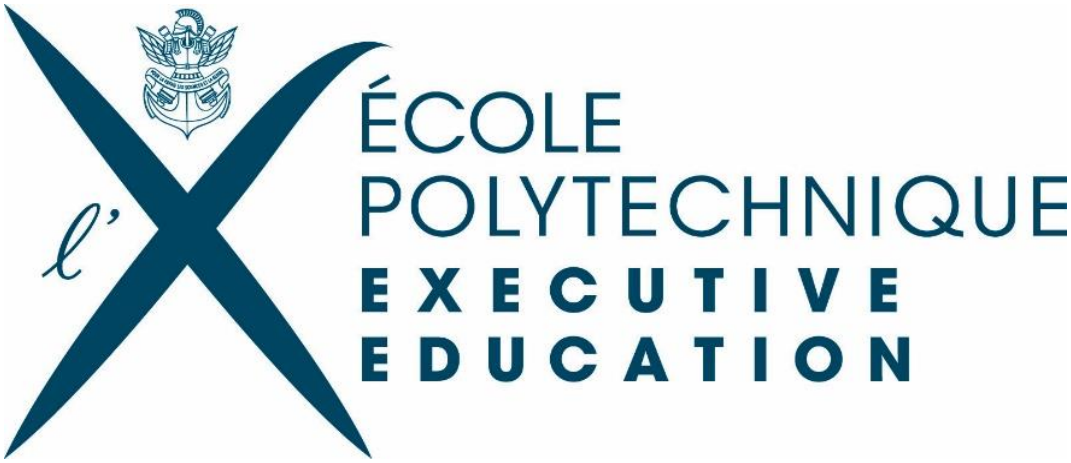
2018



Conduire un projet de sciences de données

CNCP 1527

Pair of questions similarities

NEZZARI, ABDELHAK - SAP

Contents

Introduction:

Unstructured data represent 88% of the existing data in 2015[*]

And most of these data has a very high business value, 80-90%[*]

And the unstructured data will increase very significantly in the future.

In the project, we will use texts as part of unstructured data, and we will apply some NLP and deep learning algorithms.

Long term objective (out of scope):

At SAP, we have many information in the text format that we can be exploited, we have for example:

- Manual text cases
- Manual configuration

In both we have huge amount of texts, and the user must read the text and in parallel need to perform actions in the system by following the written steps.

This need to be automated, by extracting necessary information from the test cases and manual configuration and map it to some automated test scripts, the data of training should be the format:

Description what to do:	Action Script:	Matching
'Open Transaction SOLAR01'	ScriptXYZ	1
'Open Transaction SOLAR01'	ScriptXYF	0

Extracting such data is not easy and might very effort, so let's keep the scope small and focus more on the part of machine learning.

Short term objective and project:

For the project, I have chosen sematic comparison between two short sentences, and find out if the sentences are close to each other semantically, by measuring similarity between two texts, and this is a key topic in NLP and very important statistic problems, as the short text are widely used today in the internet, in:

- Product descriptions
- News headlines
- Forum questions
- Image and video descriptions

Finding or measuring the similarity has many applications in:

- Text similarities
- Question / answering
- Machine translation
- Data organization, and avoid saving duplicate data
- Test text to test script translator
- Classification, clustering

All these problems share the part of machine learning, as we you have two entries to the machine learning, and the machine will measure or predict the similarity.

The similarity of the texts comes from the fact, that, the people can express differently the same thing,

For example:

Sentence 1: "The plane landed at 12 pm"

Sentence 2: “the flight arrived at noon”

These two sentences are semantically similar, so we will train the model with entry like:

Sentence1Vect	Sentence2Vect	Duplicate 0:1
The plane landed at 12 pm	The flight arrived at noon”	1

And, the part of machine learning should similar for my long-term objective of test case and configuration automation, as it will be mainly:

I will use the data set from Quora:

http://qim.ec.quoracdn.net/quora_duplicate_questions.tsv

Where they provide TSV file to train the machine to find out if given question was already existing in the data base, and avoid storing the same question,

Question1	**Question2**	**Is duplicate?**
How can I be a good geologist?	What should I do to be a great geologist?	1

and train a model that can predict if we they are duplicate or not.

The data science question:

What are the features that we can extract from data, given that the data is just two text sentences?

What are the most influencing feature for the prediction?

I will use my intuition in identifying these features, from the first sight the features can be in two categories:

- The features for the delta and difference between the two sentences
- The sequence of the words in every sentence

We will the two classical ways to study the problem:

- Identify the features manually and give them to the machine learning
- Give everything to the machine learning, and the machine will find the features itselfe

In the first way, we will use classical machine learning.

In the second way we will use neural network and deep learning layers.

Data exploration

Original data set:

	id	qid1	qid2	question1	question2	is_duplicate
1	0	1	2	what is the step by step guide to invest in share market in india	what is the step by step guide to invest in share market	0
2	1	3	4	what is the story of kohinoor diamond	what would happen if the indian government stole the kohinoor diamond back	0
3	2	5	6	how can i increase the speed of my internet connection while using a vpn	how can internet speed be increased by hacking through dns	0
4	3	7	8	why am i mentally very lonely how can i solve it	find the remainder when is divided by	0
5	4	9	10	which one dissolve in water quikly sugar salt methane and carbon di oxide	which fish would survive in salt water	0
6	5	11	12	astrology i am a capricorn sun cap moon and cap risingwhat does that say about me	i am a triple capricorn what does this say about me	1

The original data consists mainly of three columns: question1, and question2 which is pair of questions, in addition, to the label column ‘is_duplicate’ which tells if the two questions are semantically identical or not.

Label 0, means non-duplicated question, and label 1 means duplicated question.

In total we have 404 290 lines in the data set.

Number of label with 0 is greater than with 1, almost we 50 % lines more in the non-duplicate questions than in duplicated questions.

We need to consider this in the training of machine learning, so that we give the same proportion of data example with 0, and with 1, so it should be 50% for both.

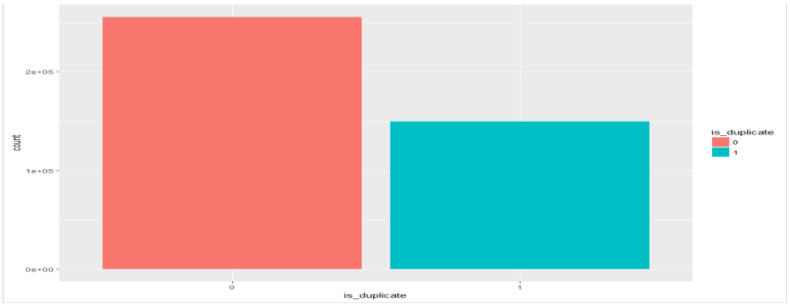


Fig1. Proportion of label categories (duplicated, non-duplicated)

Otherwise the prediction of the machine learning will be biased toward the non-duplicated category as it contains more data examples.

Words in common between two question columns:

The words are extracted from the two sentences of both question columns, then we count their usage, we sort the words according to the number of usage, and finally we join the common words from both columns and we show the result in pyramid plot.

In picture we take the subset of [500:530] only, as showing all the words will make the graphics not readable.

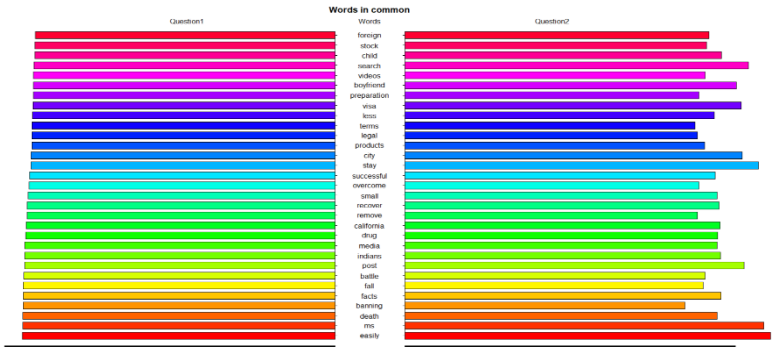


Fig2. Comoun words usage comaparaison with pyramid plot

R code:

In section '# pyramid_commoun_words' of appendix.

Word cloud:

We are interested in all the words that are used in all questions of both columns: question1, question2, the world cloud can show the words that are in common, it can show also the difference.

We put all the questions of columns question1 and question2 into two big text, and we send it to the world cloud for visualization.

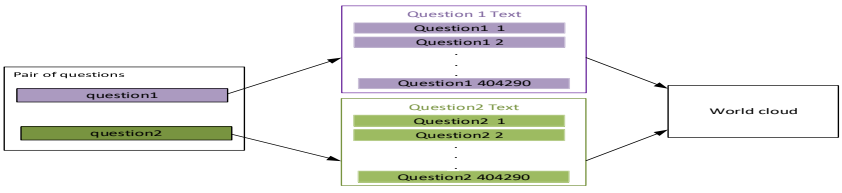


Fig3. Word cloud from question1 and question2 corpuses

With this kind of visualizations, someone can see the words that are mostly used in the questions, so it shows the hot topics in a given text corpus.

Word cloud common:

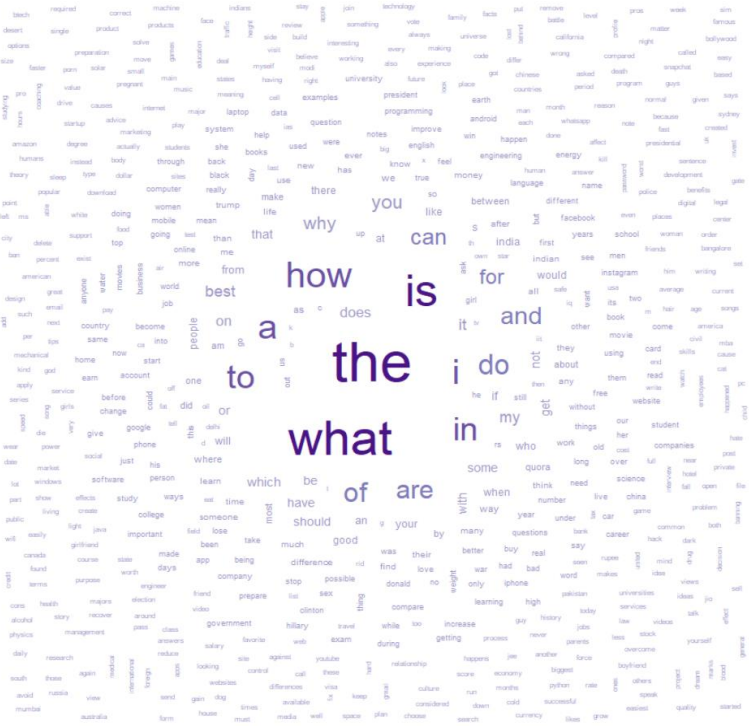


Fig4. Word cloud showing common words

These show the words which are common, it shows also the words which mostly in common by displaying the words more bolded.

As you can see in the picture, words like: ‘how’, ‘what’, ‘the’, ‘is’, ‘to’, I didn’t apply any filter on the words, someone can introduce with shiny UI scroll bar that can zoom or filter the words.

Word cloud comparison:

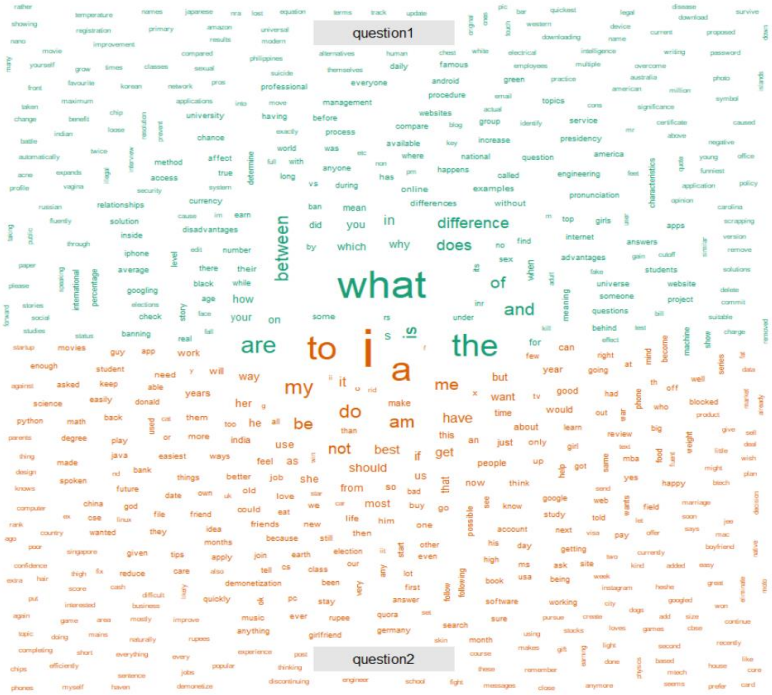


Fig5. Word cloud showing difference in frequencies of words for corpus of: question1, question2

This cloud picture compares the words frequency between the two documents: ‘question1’ and ‘question’, and the words which are shown more bolded in given color orange or blue-green, means that the words are more frequent in the text corpus of question 1 if it is blue-green, or more frequent in text corpus of question 2 if it is shown with orange color.

R code:

The R code is in section ‘# words.cloud’ of appendix

Clean the text of questions data:

Before using the text from the two sentence columns question1, and question2

We need to clean the texts and perform some text operations, like:

- tolower(): I like -> i like
- split_non_ascii(): 形から入る -> 形 か ら 入 る
- replace_appostrophs(): what's -> what is, isn’t -> is not
- remove_brackets(): "spiritual" -> spiritual
- remove_punctuations(): . ?

We decided not to remove non-asci characters, as some of them correspond to Chinese words in an English question, so we keep them to keep the sentence as complete as possible, but this is not a problem for our problem of finding similar questions, because every word will get a vector, and the learning machine will not really care.
All the cleaning operation is consisting of unifying the words in both columns, to be sure they use the same format, and prepare the data to the tokenizer, so that it gives the best result, and avoid removing the words as much as possible, as we are dealing with the text semantic, as one word only can change completely the sense of given sentence.

R code:

The code is in the section '# Clean.data' appendix.

Feature extraction from question sentences:

The first step will be extracting some features form our pair of questions, and we will send the features to the learning machine to decide how to classify, and decide from the calculated features:

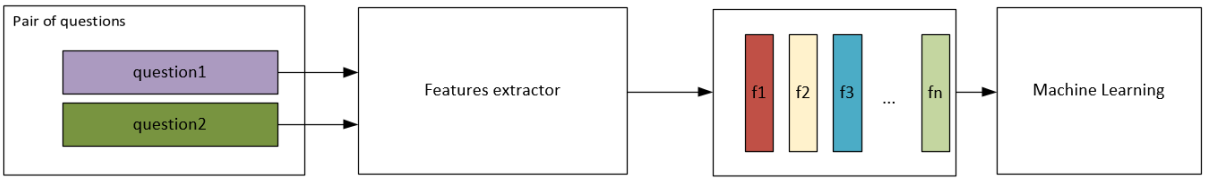


Fig6. Feature extraction from the two questions

We will extract the features f1, … fn in column formats, and at the end we merge them to one table which we send the different machine learning models.

We calculate the feature by ourselves using R capabilities, and they are based on:

- Number of words
- Grammatical entities
- Sequence of grammatical entities
- Word vector representation
- Cosine distance between two words

Feature based on number of words:

This is the simplest feature that someone can think about, we split the sentences into words, and we count the number of words which are in,

Need to get some features form the shot sentences of questions:

- Number of words in sentence 1
- Number of words in sentence 2
- Number of stop words sentence 1
- Number of stop words sentence 2
- Number of words difference between sentence 1 and 2
- Number of words difference between sentence 2 and 1

Number of words in sentence:

Question 1 and question2 are tokenized, we extract the list of used words from both questions, we count the number of words and store them in fields:' q1_length', 'q2_length'

Number of stop words sentence

We use: stopwords("english") to get this list,

Like: "i","me","my","myself","we","our","ours", …

And the occurrence of these words in both question sentences, we count them in the fields: 'q1_stopwords', 'q2_stopwords'

Number difference words between sentence 1 and 2:

We use the words from both questions, and we get the difference of words with setdiff, we count the number in fields: ‘q1_q2’, ‘q2_q1’.

We plot the density distribution for the calculated attributes: ‘delta_q1_q2’, ‘delta_q2_q1’, ‘question1_no_stop_words’, ‘question2_no_stop_words’, ‘question1_no_words’, and ‘question2_no_words’, they are shown with different colors indicated in the legend of graphs, all the graphs are showing discrete values as they are numbers, some graphs looks similar in both categories (duplicate with label ‘1’ and non-duplicate with label ‘0’) .

We have divided the graph by facet grid, the first grid is showing the densities of calculated attributes for questions which duplicate, and have label ‘1’, and the other grid is for non-duplicated questions with label ‘0’

In the category of duplicate questions, the blue/yellow, pink/green graphs are correlated somehow, as they have the same shape but slight different values, as one graph is bit above the other, but the follow the same variation.

In the category of non-duplicated questions, green/blue, blue-green/pink, yellow/red looks correlated in shape and slight different values.

In summary, there are some attribute densities which are correlated, and the correlated attributes are not the same in both cases.

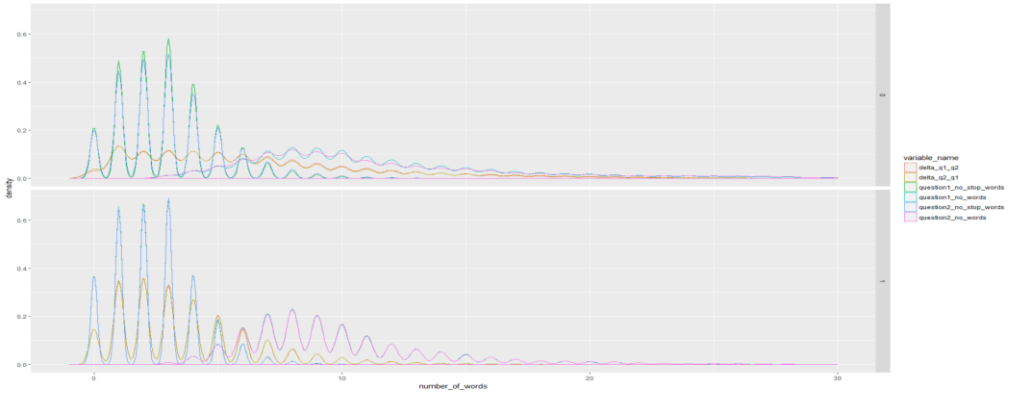


Fig7. Density distribution of features based on word numbers

Now we plot the attributes as pairs, in X axe will have one attribute, and in Y axe with another attribute. And we plot the questions as cloud of points, in red non-duplictae and blue-green the duplicate.

We combine all the calucated attribute names, and we plot then questions as cloud of points.

As we can see in the graphs, that almost all the blue-green (duplicated) points are near the (0,0) coordinate, except for some popints, this can help to separate lineary the two categories, if we play all the attributes together.

And the red cloud occupy more space, then the blue-green, this can be also explained by the difference between the number of the two categories, as the non-duplicated questions is higher than the duplicated ones.

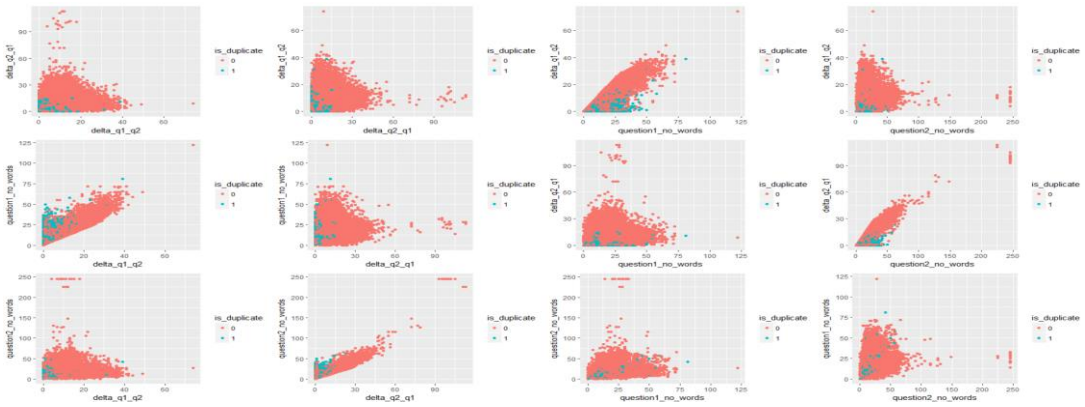


Fig8. Scatter plot for duplicated and non-duplicated categories using different feature combinations

We can check the redundancy amongst the calculated attributes, with corrplot shown below, this confirm what we have noticed with the density plots, where some attributes are correlated.

The following attributes are slightly correlated, as seen shown in density, they have the same shape, but values differ little bit:

- question2_no_stop_words / question2_no_words and delta_q2_q1
- question1_no_stop_words / question1_no_words and delta_q1_q2

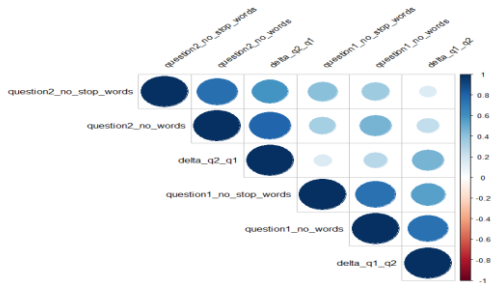


Fig9. Correlation of the features by word counters

We must consider these correlations when we build the machine learning models, by reducing the used attributes in the prediction, as this will keep the accuracy, make model learning faster and reduce the validation error.

To verify the range variation of the calculated attributes, we plot the Bar boxes, and we show in different colors the two categories, we can see that with non-duplicated category, the attributes have larger variation as the Bar box is wider, the top values are shown with many vertical points, the graph is not showing the highest value for attribute 'question2_no_words' which is 250 as we resized the taken screen shot to take less place

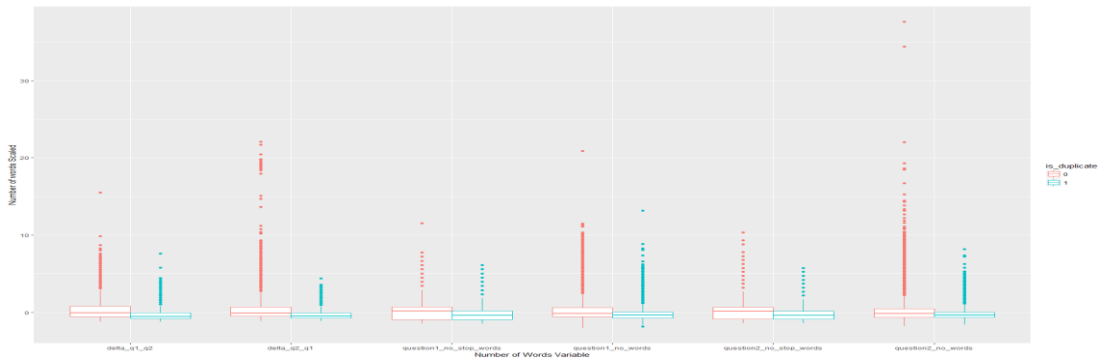


Fig10. Barbox plot for the word counters features

Another plot that show the attributes variation in addition to their density distributions, is Violin plot, we plot it similar way to the previous Bar box plot, we can see the difference between the two categories of questions in the plotted attributes.

In the non-duplicated questions category in red, the plots have higher values as the vertical line on every attribute is longer, and the shape of the violin is slimmer.

In the duplicated questions category, the shape of violins looks like pyramidal trees, and the top values are lower comparing to the other category.

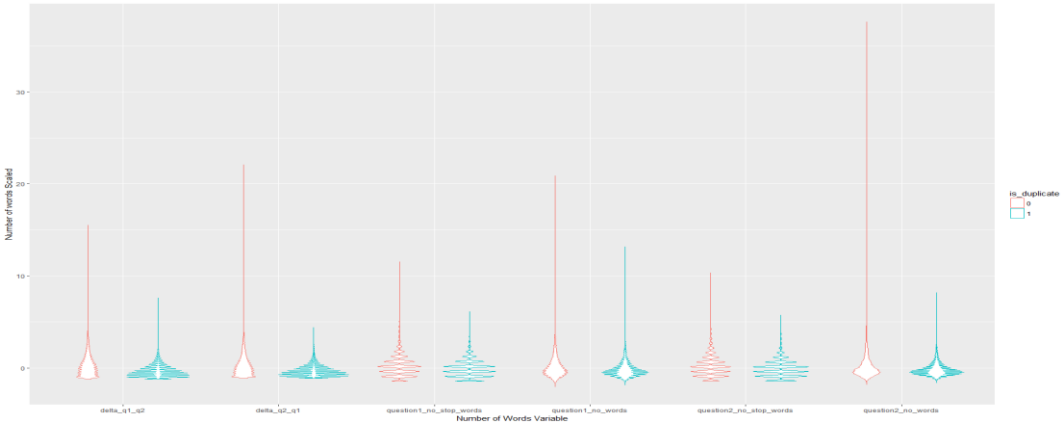


Fig11. Violin plot for features of word counter for both classification categories

Let's hope that these differences between the attributes will help the machine learning to predict both categories correctly, as the more difference found, the easiest will be the job of machine learning.

R code:
The different attributes are calculated using our R functions mentioned in the appendix part:' ***Feat.Calc.Part1.*** '

The results are saved in different 'RData' files, all the files are merged together to form one table for these group of features, the merging of attributes are done in R function 'build_table_of_features' mentioned as well in the appendix

Grammatical entities statistics:

We calculate the number of grammatical entities like:

- Number of verbs
- Number of adjectives
- Number of proper nouns

In NLP they call these entities: 'Part-of-Speech taggers' abbreviation 'POS taggers'.

The complete list of these entities can be found in website:

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

This is an example of some of them:

- CC Coordinating conjunction
- CD Cardinal number
- DT Determiner
- EX Existential there
- FW Foreign word
- IN Preposition or subordinating

- JJ Adjective
- UH Interjection
- VB Verb, base form
- VBD Verb, past tense
- WDT Wh-determiner
- WP Wh-pronoun

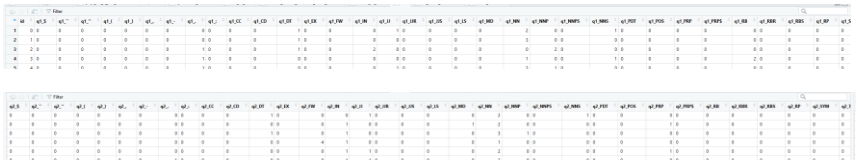
We use openNLP library to parse the two sentences and we extract the different Tages used in both.

We store them in table of the following format:

	$	``	''	(	)	,	-	.	:	CC	CD	DT	EX	FW	IN	JJ	JJR	JJS	LS	MD	NN	NNP	NNPS	NNS	PDT	POS	PRP	PRP$	RB	RBR	RBS	RP	SYM	TO	UH	VB	VBD	VBG	VBN	VBP	VBZ	WDT	WP	WP$	WRB
1:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3:	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Every column field contain the number of time a grammatical entity is used, so it is a number, and if it is 0, then it was not found.

We do this twice for column question1 and question2.



As we have 45 x2 = 90 columns which is too much, so we will apply PCA to reduce the number of columns to just 4 x 2 = 8.

We can also reduce the dimensionality by:

- Either, calculating the cosine distance between the vectors of both questions
- Or, applying PCA to reduce the number of columns to just 4 x 2 = 8.

We use the calculated vectors of question1 and question2 to calculate the cosine distance, between the two, and we store this in field: '`cosin_distance_bag_pos`'

So we calculate the PCA components in addition to the cosine distance, and we can plot the different attributes in Box plot to see the range of variations in both categories: duplicated and non-duplicated to see if there are any difference in the variation behavior and compare the different attributes between themselves, as we can see that the variation of given attribute is almost the same for both categories, with duplicated category, the attribute Bar box is slightly higher and the median looks as well higher, the first PCA components are have larger boxes, and the last PCAs looks smaller, this is the main feature of PCA decomposition, as the first components represent more pair of questions in our data set that's why we have more variation in first PCAs.

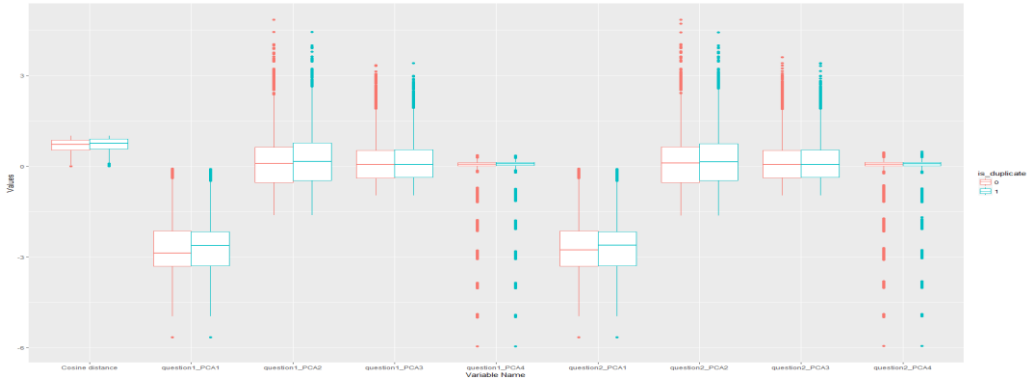


Fig12. Barbox Plot for the PCA components of POS taggers statistics and cosine distance

If we combine the Bar box with the density distribution, we get the following violin graphs

the red color and blue-green represent also the two categories of pair of questions, the shape of PCAs component looks almost the same, even it is not varying in the same range of values, except for PCA4 for both question1 and question 2, the shape also of cosine distance looks different.

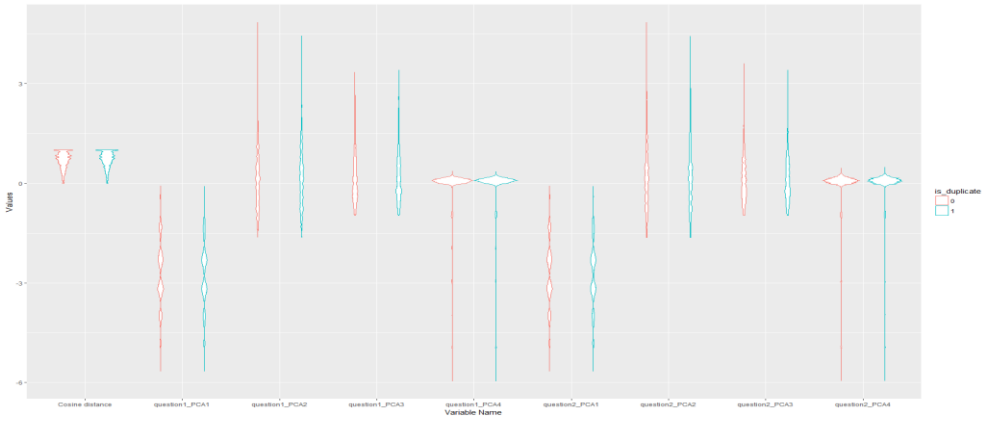


Fig12. Violin Plot for the PCA components of POS taggers statistics and cosine distance

We plot the correlations matrix and show that almost there are no correlation between all the PCA components, as well the cosine distance between the vectors of POS taggers statistics of both questions.

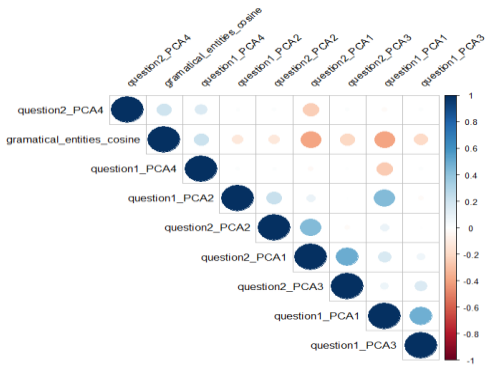


Fig13. Correlation between different PCA comonents of POS taggers and cosine distance

R code:

The features are saved also in RData table format, the R code that extract them is mentioned in the appendix section '***Feat.Gramatical.Entities.Statistics***' there are mainly 5 parts

- The first part is initializing the tag POS parser
- The second part is parsing the tag POS entities
- The third part is applying the parser to columns question1, and question2 and save the results in RData table
- The fourth part is calculating the cosine distance and save the result
- The fifth part is calculating the PCA component and saving the result

Sequence distance for grammatical entities:

I wanted the first time to get information about the sequence of words in given sentence, so I googled little bit, and found the R package 'TraminR' which primary aim is to analysis of biographical longitudinal data in the social sciences, such as data describing careers or family trajectories.

Applying it to word sequences will need a huge calculation time, given the huge amount of words in all our questions corpus, we have around 95000 words, also the length of the sentences where we have 20 words in mean.

But we can apply it to the sequence of grammatical entities, and we can measure some metrics provided by the library, and that what we are looking for, to feed our machine learnings.

First, we need to extract the sequence of grammatical entities, we parse the sequence of POS tags (part of speech taggers) and we get sequences like:

NNP-VBD-RB-VB-DT-NN for question 1
NNP-IN-JJ-NNS-VBP-VBG for question 2

We use openNLP library to get this sequences for all the questions.

Then we use library traminR to measures some distances, there a nice user guid help in the following link:

http://www.ruxizhang.com/uploads/4/4/0/2/44023465/traminer-1.4-users-guide.pdf

The package 'traminer' offer some distances to quantify the distances
seqLLCS: length of the longest common subsequence of two sequences.

seqLLCP: the length of the longest common prefix of two sequences.

seqmpos: the number of common elements.

You give to these functions, two sequences of:

- POS tages
- Word sequences of question

They return numbers, and this is what we are looking for, as we can use these distance functions calculate new features that gives some information about the sequence in our questions.

Then we store the calculated distances, in new fields:
- traminr_seqLLCP
- traminr_seqLLCS
- traminr_seqmpos

We plot the density distribution of the three features, and we get the following picture, we can see that there a re 6 distinguish values for all the features.
The density 'traminr_seqLLCP' in red color is starting high at 0, and then decreasing for the values after, like a b all bouncing.
The density of 'traminr_seqLLCS' in 'green' color, is increasing and then decreasing.
From graph it is obvious that the different attributes are correlated.

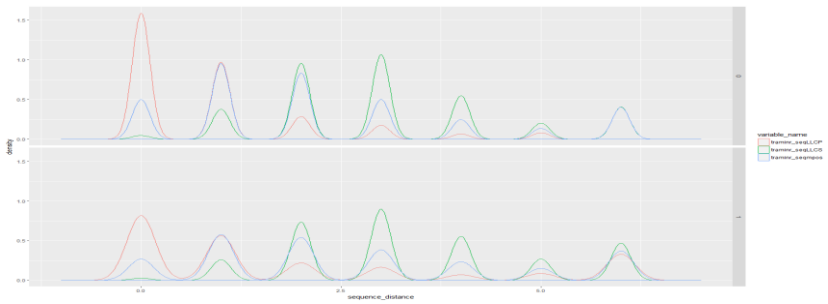


Fig14. Desity of sequence distances between POS taggers

We plot now the pair relation between the combination of the three features, to see if we can visually separate the duplicate questions from the non-duplicated.

There different features they have a limited variation values, and as they are integers, we can see the two categories are easily separated by the pair of calculated attributes.

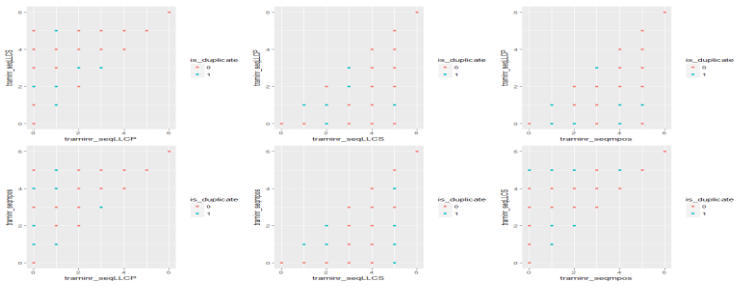


Fig15. Scatter plot to separate the two categories using sequence disntances of POS taggers

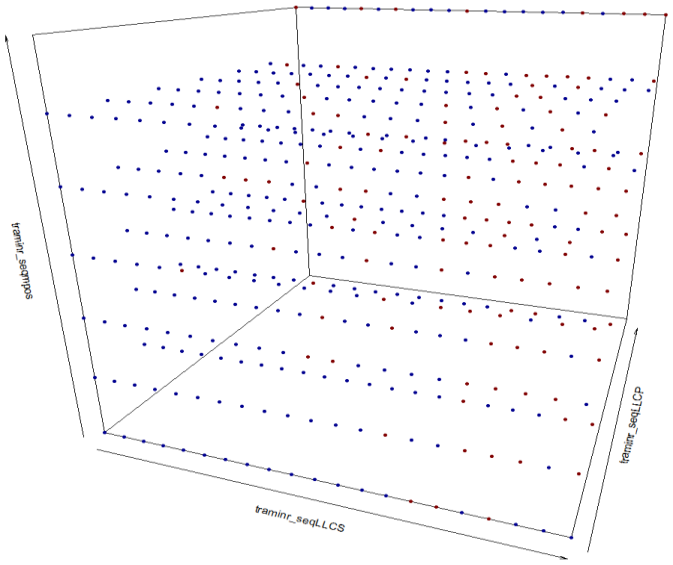


Fig16. 3D Scatter plot for the sequence distances of POS taggers

We plot the Bar box plot the see the variation of calculated attributes

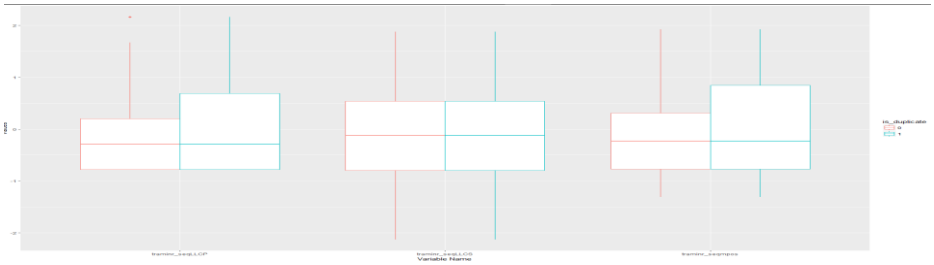


Fig17. Bar Box plot of the sequence distances of POS taggers

And the Violin plot to combine both range of variation and density of distribution.

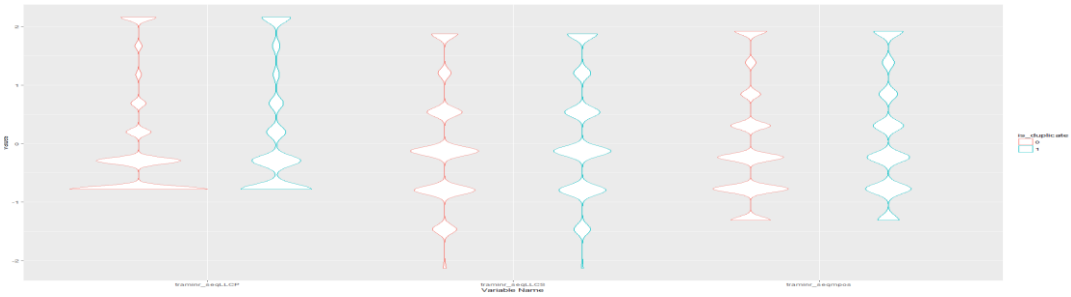


Fig18. Violin plot of the sequence distances of POS taggers

let's see if there are any correlation between the calculated three features

We can see that all the features are correlated.

So we can take at least one for the machine learning.

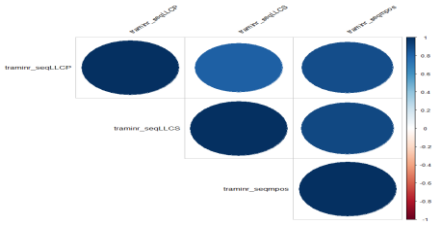


Fig18. Correlation matrix between the seuquence distances of POS taggers

R code:
The results of calculated attributes are saved in RData table, the code that extract the features is the appendix section: '**# Feat.Sequence.distances.By.TramineR** ' the attributes are saved to RData table.

Word sequences:

We apply the sequence distances that can be calculated by 'traminR' library to the sequence of words in the questions, we first extract the vocabulary of words, where we give every word a number and we replace the words in the question sentences with these number indexes, we calculate the three distances: seqLLCS,seqLLCP,seqmpos as we have done in the previous section.

We do the calculation in NVIDIA machine to boost the run time.

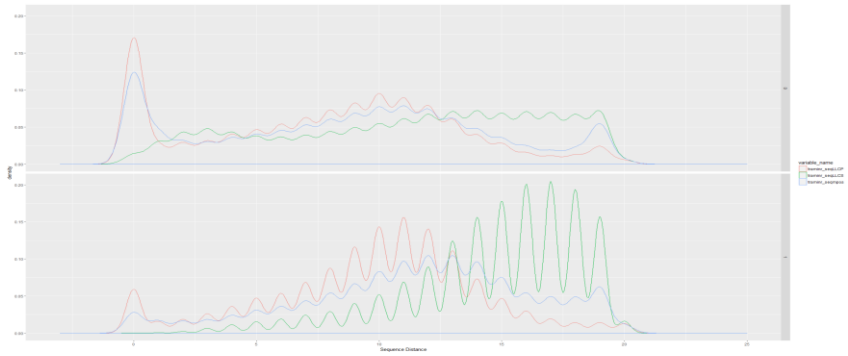


Fig19. Density of word sequence distances

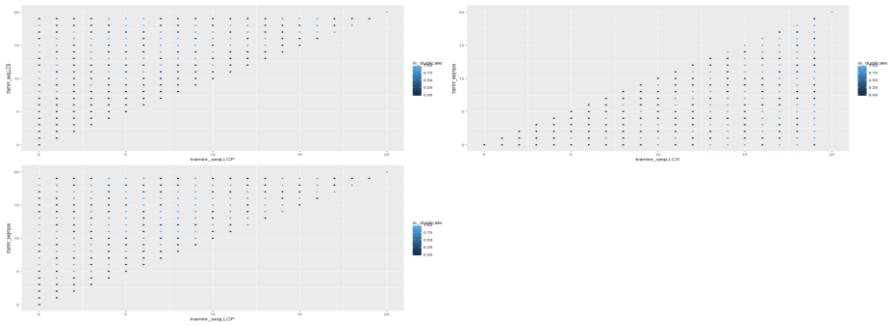


Fig20. Scatter plots of the word sequences distances

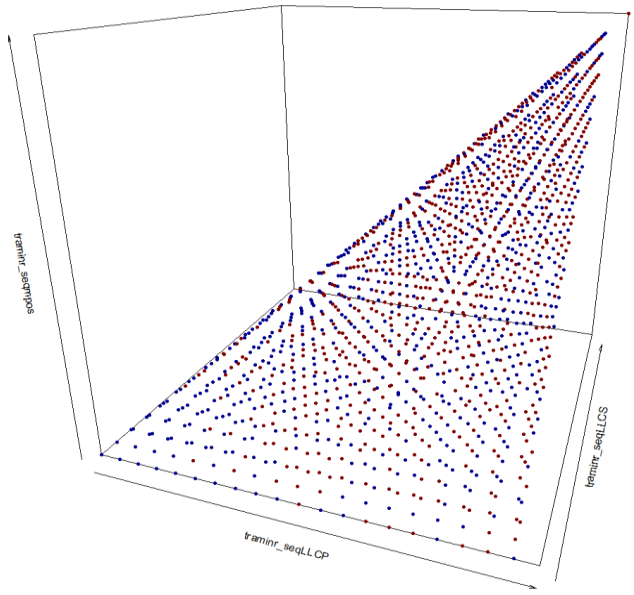


Fig21. 3D Scatter plot of the word sequences distances

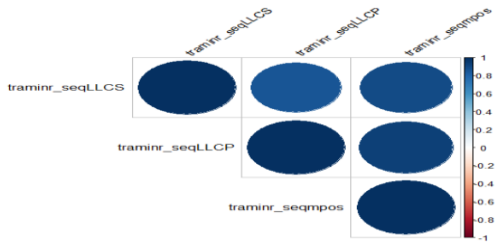


Fig22. Correlation between the word sequence distances

Cosine distances by Word vectors:

Word vectors:

Word vector algorithms build dense vector for each word, chosen so that it is good at predicting other words appearing with its context.

$$\text{Linguistics} = \begin{bmatrix} 0.266 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.258 \\ 0.548 \end{bmatrix}$$

These other words also being represented by vectors, you can get a lot of value by representing a word by means of its neighbors.

Glove algorithm defines a model that aims to predict between center word w_t and context words in terms of word vectors.

It looks at many positions t in a big language corpus, and it keeps adjusting the vector representation of words to minimize the loss function.

Word2vec main idea is to predict between every word and its context words.

There are two main algorithms:

- Skip-grams (SG): predict context words given target (position independent)
- Continuous bag of words(CBOW): predict target word from bag of words context

In the skip-grams algorithm, they slide a window of given radius over the text corpus, in our case the questions, and in the example below we set the radius to 5 for the sliding windows, we will go at every position in the questions corpus and calculate probability for surrounding words given the center word, they use the Softmax function to calculate such probability which can be written in its simplest form:

$$P(o/c) = \frac{\exp\,(U_0^t V_c)}{\sum_{w=1}^{V} \exp\,(U_0^t V_c)}$$

Where o is the outside word index or (output), and c is the center word index, V_c and U_0 are center and outside vectors of indices c and o.

$$U^t V = U.V = \sum_{i=1}^{n} u_i v_i$$

This is the dot product, and the bigger it is, the more U and V are similar.

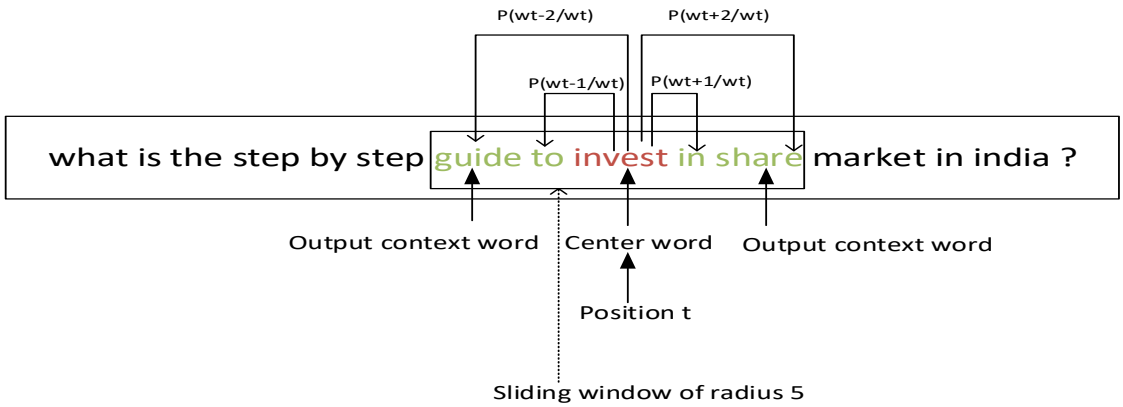


Fig23. Sliding widow to calculate word-word co-occurences

Get the word vector using Glove (global vector):

It is model for word representation, the model is unsupervised learning algorithm for obtaining vector representation for words, the training is performed on aggregated global word-word cooccurrence statistics from a corpus and the resulting representation showcase interesting linear substructures of the word vector space. It is developed as an open source project at Stanford. wikipedia

Glove is Log-bilinear model based on ratios of word co-occurrences frequency. The training objective is that the dot product of the vectors learned for words is equal to the logarithm of their co-occurrence frequencies, it regresses over a weighted least square to get optimal vector presentation of words.

$$J = \sum_{i,j=1}^{V} f(X_{ij})\,(w_i^T \widetilde{w_j} + b_i + \widetilde{b_j} - log(X_{ij}))^2$$

V is the size of vocabulary

f is the weighting function

$$\mathrm{f(x)} = \begin{cases} (x/x_{max})^{\alpha} & if\ x < x_{max} \\ 1 & otherwise \end{cases}$$

According to what I have read from the internet, Glove looks like to be the best algorithm for words vectorization, so we will use to generate dense vectors of

dimension 50, and glove can also give the words which are similar in meaning vectors where the cosine distance is small.

To feed glove word vector, the question1 and question2 column sentences need to be transformed, in the following way:

- Put all the questions in column question1 and question2 in one table of 808580 entries questions.
- Get the tokens
- Create vocabulary using the tokens
- Vectorize the vocabulary
- Term co-occurrence matrix: TCM
- Pass TCM and vocabulary tables to glove algorithm to build the word vectors of dimension 100

The pipelines for generating Glove word vector is summarizing in the following diagram:

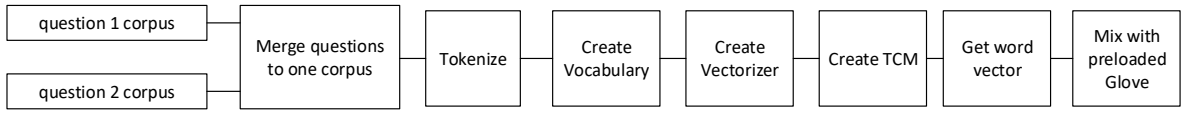


Fig24. Pipline of word vectorization with Glove Algorithm

Create TCM:

term co-occurrence matrix

skip_grams_window = 5

94559

94559

	nontelgu	guantity	findsearch	.	.	.	i	is	what	the
nontelgu	0	0	0				0	0	0	0
guantity	0	0	0				0	0	0	0
findsearch	0	0	0				1	0	0	0,2
.										
.										
.										
i							3187.28	2067.37	9510.80	8456.15
is							0	1329.87	132774.8	119553.6
what								0	518.60	76219.36
the	0	0	0					0	0	10867.13

Fig25. TCM Matrix

The vocabulary words will be used to build a square matrix, where the columns and lines of this matrix will be the vocabulary word, and window will slide the questions corpus to find the statistics words used in the context of other words of the vocabulary as described above in the skip_gram algorithm windows.

Get word vector:

The Glove algorithm that we described above is used to vectorize and reduce the dimension of words, the parameters are as follow:

- x_max = 10
- vector size = 100
- number of iteration = 20

Mix with preloaded Glove:

We download a pre-calculated word vectors from: 'http://nlp.stanford.edu/data/'

In this link, you can find word vectors with different dimension formats: 50,100,200 and 300.

We first get all our vocabulary of words for the two-word corpus of the two columns: 'question1' and 'question2', we apply the 'Glove' algorithm to get word vectors of dimension 100.

We loop on the vocabulary of words, if we find that the word exist in the pre-calculated 'Stanford' vocabulary we use it, otherwise we use the vector we have calculated, we believe that these will increase the accuracy of word vectors.

R code:

The extraction of word vectors using Glove algorithm, can be found in the section 'Glove.word2vec' of appendix.

Projection to 2 D of word vectors using t-sne:

If we reduce the dimension of the generated word vector by glove, using t-sne two axes, we obtain the following cloud of points, every point in the cloud represent a word from the pair questions corpus.

We have used the following parameters for t-sne algorithm: Theta = 0.5 ,PCA = True,Dims = 2

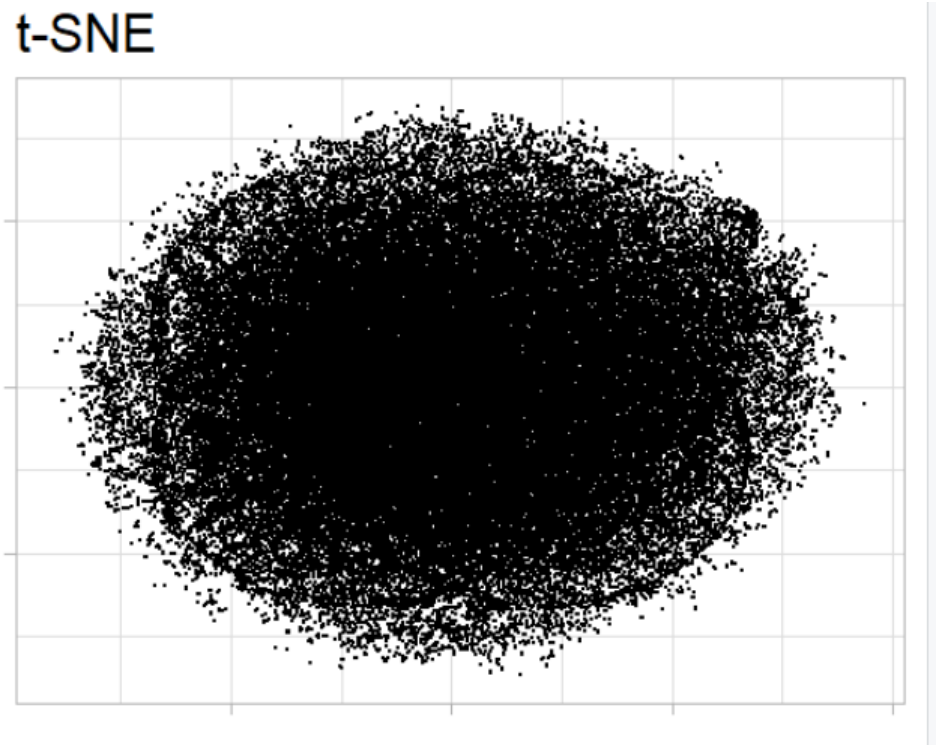


Fig26. Words cloud of word vectors in 2D using T-SNE

R Code:

In section '*T-SNE.WORD.VEC.2D.PLOT*'

Words of interest:

Let's see only some words, and see their neighbors, to check if the semantic of the words are respected geometrically.

"children", "family", "health", "safety", "happy" we find the 10 other neighbors words of these words and plot them in XY plane

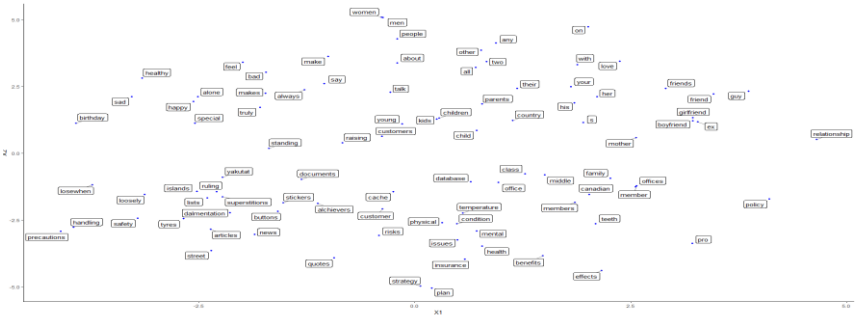


Fig27. Some example of words in 2D to check the neighbor words

R code:

Appendix section: '**word.interest.plot**'

Cosine distance between pair of questions:

We have every sentence matrix in matrix format:

Q[number of words in sentence x dimension of every word]

Example:

The first question has 14 words of 100 vector for each word

If we multiply this matrix of Q[14 * 100] by Qt[100*14] using the cosine distance we will obtain a smaller matrix in dimension [14 X 14], you can see the diagonal is one vector.

We can plot the dot product between the words matrix, let's see how they look like.

	what	is	the	step	by	step	guide	to	invest	in	share	market	in	india
what	1.00000000	0.69093311	0.72722591	0.17053491	0.48792876	0.17053491	0.02224997	0.5624913	0.22175854	0.60435341	0.2894672	0.30674748	0.60435341	0.52013969
is	0.69093311	1.00000000	0.59108398	0.16223068	0.47112124	0.16223068	-0.01027738	0.4527965	0.10894902	0.49761820	0.2247630	0.28731800	0.49761820	0.49553672
the	0.72722591	0.59108398	1.00000000	0.25295319	0.51739732	0.25295319	0.09191601	0.5345194	0.13052505	0.68621369	0.2885964	0.38414386	0.68621369	0.49001709
step	0.17053491	0.16223068	0.25295319	1.00000000	0.36857326	1.00000000	0.22976397	0.1644976	-0.13505212	0.12589616	-0.1716888	-0.01895668	0.12589616	0.06716778
by	0.48792876	0.47112124	0.51739732	0.36857326	1.00000000	0.36857326	0.01400204	0.4538640	0.11873008	0.44326469	0.1410891	0.16514995	0.44326469	0.47906138
step	0.17053491	0.16223068	0.25295319	1.00000000	0.36857326	1.00000000	0.22976397	0.1644976	-0.13505212	0.12589616	-0.1716888	-0.01895668	0.12589616	0.06716778
guide	0.02224997	-0.01027738	0.09191601	0.22976397	0.01400204	0.22976397	1.00000000	0.2264952	0.08747211	0.01270618	0.2070969	0.07038095	0.01270618	0.10016322
to	0.56249134	0.45279653	0.53451942	0.16449761	0.45386400	0.16449761	0.22649517	1.0000000	0.36986723	0.48142646	0.3762303	0.25145500	0.48142646	0.45873242
invest	0.22175854	0.10894902	0.13052505	-0.13505212	0.11873008	-0.13505212	0.08747211	0.3698672	1.00000000	0.36180568	0.4143516	0.25707386	0.36180568	0.31407600
in	0.60435341	0.49761820	0.68621369	0.12589616	0.44326469	0.12589616	0.01270618	0.4814265	0.36180568	1.00000000	0.2604351	0.31066759	1.00000000	0.65680273
share	0.28946722	0.22476300	0.28859643	-0.17168880	0.14108907	-0.17168880	0.20709689	0.3762303	0.41435160	0.26043513	1.0000000	0.54280190	0.26043513	0.25116830
market	0.30674748	0.28731800	0.38414386	-0.01895668	0.16514995	-0.01895668	0.07038095	0.2514550	0.25707386	0.31066759	0.5428019	1.0000000	0.31066759	0.25748059
in	0.60435341	0.49761820	0.68621369	0.12589616	0.44326469	0.12589616	0.01270618	0.4814265	0.36180568	1.00000000	0.2604351	0.31066759	1.00000000	0.65680273
india	0.52013969	0.49553672	0.49001709	0.06716778	0.47906138	0.06716778	0.10016322	0.4587324	0.31407600	0.65680273	0.2511683	0.25748059	0.65680273	1.00000000

Fig28. Example of dot product between words of same question

Examples of duplicated questions:

We perform the dot product between the two matrixes of some pair of questions, and we plot the matrix as heatmap, as you can notice, the diagonal tends to be all red, it means that it gets ones along the diagonal.

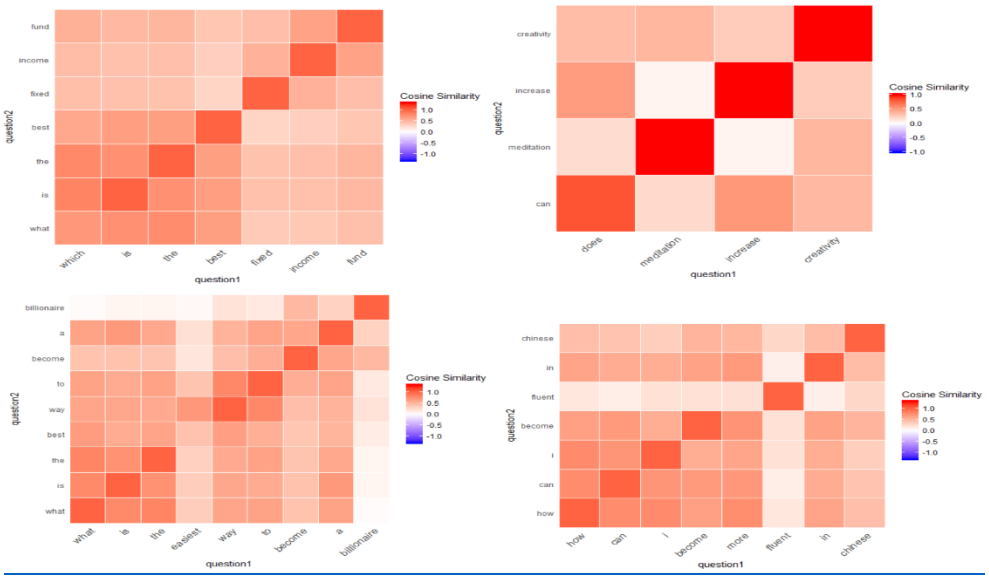


Fig29. Some examples of heatmaps of dot product between pair of questions of category 'duplicated'

Examples of non-duplicated questions:

Do product for pair of questions which are non-duplicated, in the following heatmap plots, the diagonal of ones has disappeared.

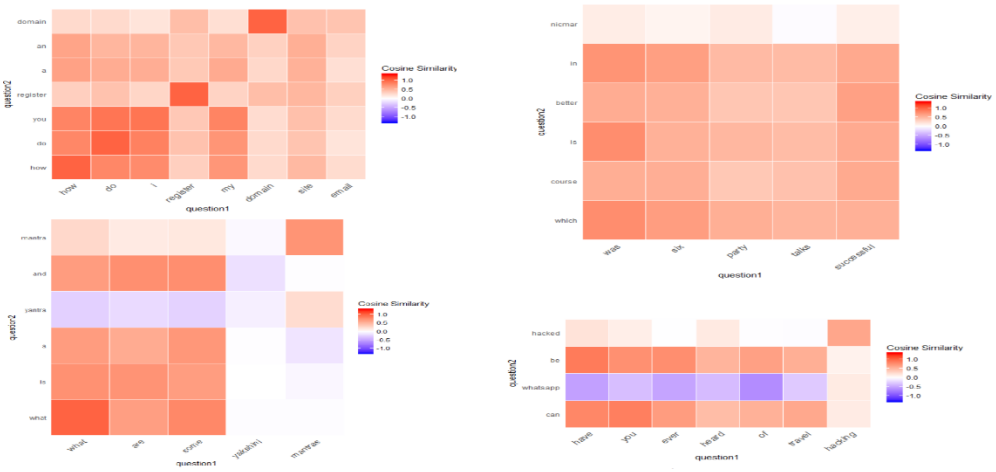


Fig30. Some examples of heatmaps of dot product between pair of questions of category ‘non-duplicated’

Normally we should get matrix with diagonal as vector of ones when the pair of question are similar.

To get good results, I will try to use a pretrained Wordvec.

I want to retrieve the diagonal of these matrixes and give it to the machine learnings.

The following diagram summarize what we wanted to achieve:

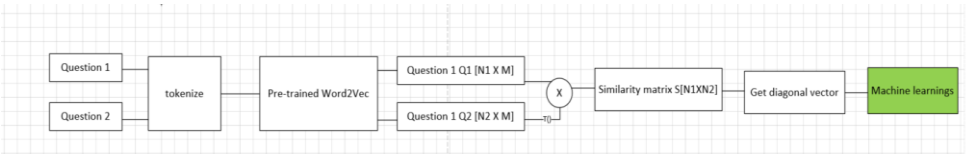


Fig31. Pipline of needed calculation to get the similarity matrix

R code:

The heatmap plots code is in the section ‘HEATMAP_QUESTIONS_COSINE_DISTANCE’ of appendix.

Conclusion:

By displaying the similarity matrix of two sentences, we have seen that the similar sentences are showing diagonal with highest value, which near to 1.

But in our data set we have seen that there are many sentence question, with long matrix diagonal equal to vector of ones, but the label is_diplicate is indicating ‘0’ which is not similar.

Until now we are trying to extract features for the question sentences, and we have used mostly: cosine distance between word vectors, the delta of words, the cosine distance of delta words to words in the other sentence question, and some sequence distances of grammatical object in both sentence, we get some satisfying result,

The problem with approach is that you need to find such features, and extract them, which need special knowledge, but at least we get feeling about the problem to solve, also this has generated us data we could practice some data science technics, like displaying graphs and applying some classical machine learning.

Cosine distance between two sentences difference:

Instead of doing dot product between the whole word matrices of the two sentences, we do it between the word difference of the two questions against the words of the other question.

So, if we do first, the word difference between question1 and question2, the obtained word difference we multiply the corresponding matrix against the matrix of question2 words.

In addition, we calculate the opposite word difference between question2 and question1, and we multiply the obtained matrix against the words in question1.

In summary we are looking to find the closest words in question1 missing in question2, and the other way, closest words in question2 missing in question1.

Let's explain it with the following example:

We have two sentences:

Question1: "What is the step by step guide to invest in share market in india?"

Question2: "What is the step by step guide to invest in share market?"

We need to get the set of words in the two questions, we do that by the mean of tokenization:

Tokenize Question1:

"what" "is" "the" "step" "by" "step" "guide" "to" "invest" "in" "share" "market" "in" "india"

Tokenize Question2:"what" "is" "the" "step" "by" "step" "guide" "to" "invest" "in" "share" "market"

Let's calculate the word difference, first we do question1- question2, and it is the same process for question2 – question1.

Token difference: "india" (question1 – question 2)

After getting the difference of words, we can measure the distance of found words with words in second question, we calculate the cosine distance to all the words and we take the biggest value, as this correspond to the word that is close in meaning, as the angle between the vectors is smaller.

Let's plot the cosine function just to get the feeling about the maximum and minimum value, we execute R code: seq(-pi,+pi,0.01) %>% plot(.,cos(.))

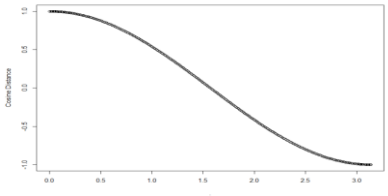


Fig32. Plot of cosine function

For angle 0, cosine is maximum, equal 1.

For angle 180 degree = pi, the cosine equal 0.

We measure the biggest distance for all the delta words, and we sum the obtained distances, and we store this in fields: q2_q1_cosine_distance, q1_q2_cosine_distance

For q2_q1_cosine_distance: we calculate the delta of words 'words of question 2' – 'words of question1'

The result we calculate the lowest cosine distance to words in question1.

For q1_q2_cosine_distance: we do the other way

In the picture bellow we plot the vector of the words for the pair of questions, in red the vector of word ‘india’ which the delta between the question2 is and question1.

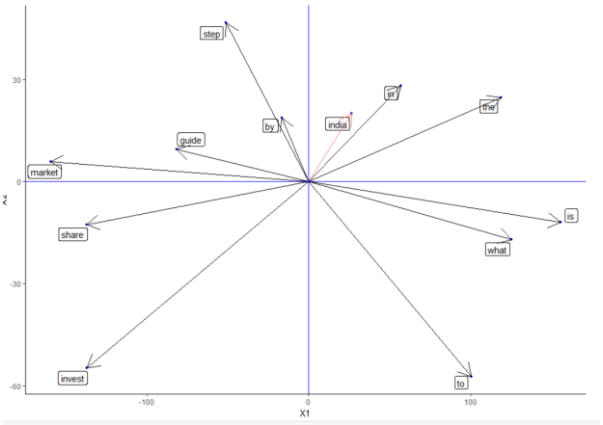


Fig33. Vectors of words of pair of questions

The above picture shows the angle between delta word “india” against all the words in question 2, we will choose the smallest value, which correspond to the word with the closest meaning.

Filter

	what	is	the	step	by	step	guide	to	invest	in	share	market
india	0.5350236	0.4702103	0.5216656	0.07907497	0.481437	0.07907497	0.07630398	0.4863769	0.2303866	0.6567631	0.2140418	0.3478025

Fig34. Similarity matrix for the delta words (only one word and one vector)

Let’s take an example of question pair where we have multiple words in the delta, in the example shown bellow we have three vectors shown in red color.

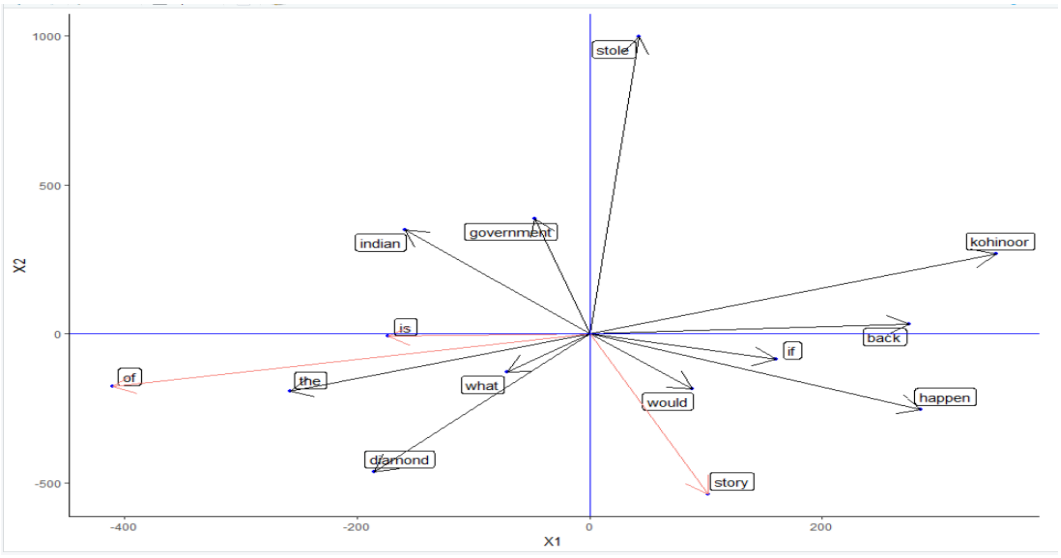


Fig35. Another example of word vectors, the delta contains 3-word vectors

Bellow, we show the similarity matrix between the red word vectors and the blue word vectors.

Filter

	what	would	happen	if	the	indian	government	stole	the	kohinoor	diamond	back
is	0.7206533	0.4081466	0.22698145	0.5565331	0.6723171	0.3676531	0.23330232	-0.008828230	0.6723171	-0.16413986	0.14048458	0.2364797
story	0.2787424	0.2665153	0.08628019	0.2161932	0.3749129	0.1297572	-0.01922223	0.112121870	0.3749129	0.03085583	0.01993476	0.1637620
of	0.6111187	0.3698290	0.09322592	0.4007239	0.6860542	0.3968665	0.37586458	-0.006037593	0.6860542	-0.01356021	0.06612470	0.2608385

Fig36. Similarity matrix between delta word vectors and the word vectors of the other question

R code:

Section '[PAIR_QUESTION_WORD_VECT_PLOT'](#) in appendix, to plot the above vectors of words for the pair of questions.

Generalization:

The following picture summarize how we calculate the similarity matrixes, we get the two questions to compare, and we tokenize every question to extract their different words, then we calculate the difference of words between question1 against question2, and the other way, question2 against question1,

We vectorize the sentences:

- Set difference of question1 – question2
- Set difference of question2 – question1
- question1
- question2

Every word in the above 4 sentences is a dense vector of given dimension, this dimension is a parameter specified when applying the Glove algorithm.

If we replace the different words by their unique vector, then we get 4 matrixes, which we can multiply, to calculate the similarity distances.

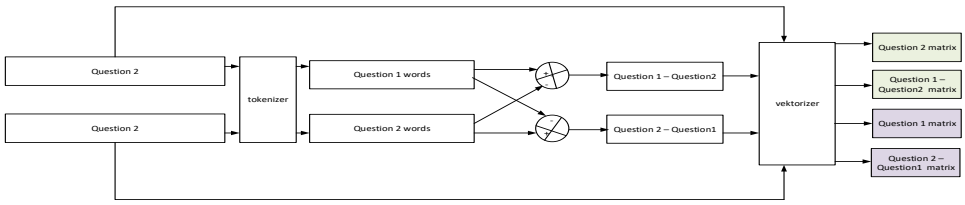


Fig37. Pipline of how to calculate the similarity matrixes

we get the matrix of difference words question1 – question2, which represent all the words in question1 that are not in question2, and we multiply it by the matrix of question2, and we get similarity matrix between question1 and question2.

The similarity matrixes are not big, as we can reduce the dimension of Glove dimension, by using it as the multiplication dimension, by transposing the matrix we multiply by.

At the end, the similarity matrixes will have the number of words of both sentences I'm comparing.

In the bellow picture, you can see the multiplication of the question matrixes, and how I calculate the features.

With the similarity matrix we can calculate some statistic features, like: minimum, maximum and mean for every: line and column of the matrix, and we calculate the sum of calculate features.

I have experimented with the mentioned statistic features, but I could also use other statistic features of matrixes, like median, standard deviation, skewness and may be others.

I will continue experimenting outside the current project.

Instead of summing the found features by line and by column, I can replace the sum by: min, max, mean as for the features, so instead of summing the features max, min, and mean, I can just find the max of the maximums, the min of the minimums, and the mean of means.

There are wide possibilities to experiment, I can measure every possibility of features by applying them to the machine learning and find the ones that are giving best of accuracy and error for the validation data.

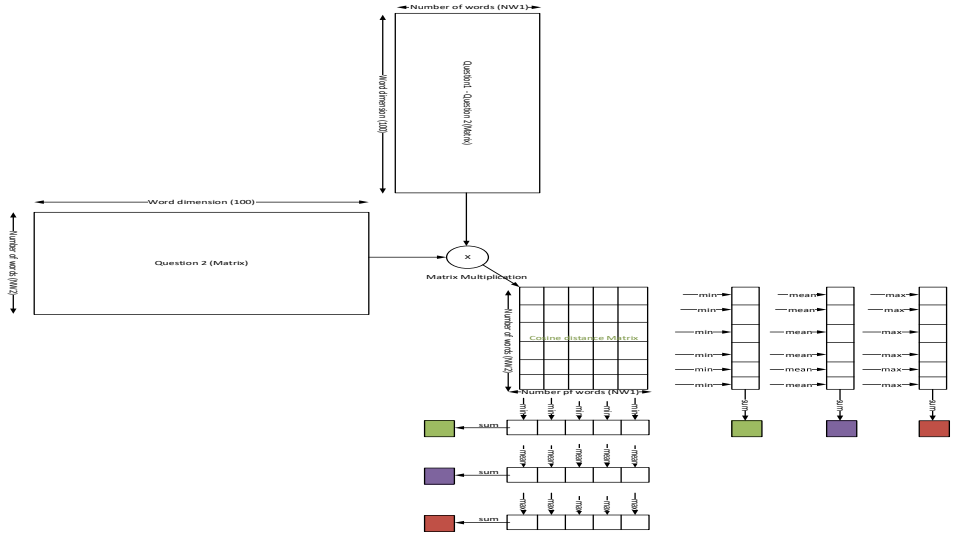


Fig38. Calculation of similarity matrix and the features

We get the following features, as columns in our data:

- q1_q2_cos_dist_max_c
- q1_q2_cos_dist_max_l
- q1_q2_cos_dist_min_c
- q1_q2_cos_dist_min_l
- q1_q2_cos_dist_mean_c
- q1_q2_cos_dist_mean_l
- q2_q1_cos_dist_max_c
- q2_q1_cos_dist_max_l
- q2_q1_cos_dist_min_c
- q2_q1_cos_dist_min_l
- q2_q1_cos_dist_mean_c
- q2_q1_cos_dist_mean_l

This is the resulting of combining the different possibilities of:

- column / line
- set difference q1_q2, q2_1
- min, max and mean

after calculating the features, let's plot the correlations, to see if any of the features is related to each other, this can allow us to reduce the features this will boost the efficiency of our machine learning algorithms, and improve the calculation time:

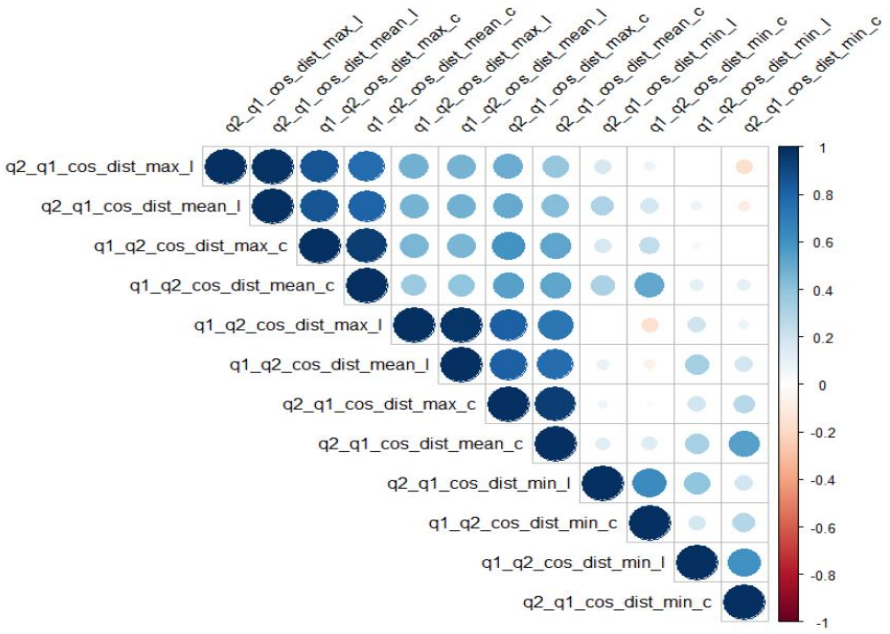


Fig39. Correlation between the calculated features

As some of the features are correlated according to the above plot, we reduce the list of features to:

- q2_q1_cos_dist_max_l
- q1_q2_cos_dist_max_c
- q1_q2_cos_dist_max_l
- q2_q1_cos_dist_max_c
- q2_q1_cos_dist_min_l
- q1_q2_cos_dist_min_c
- q1_q2_cos_dist_min_l
- q2_q1_cos_dist_min_c

let's plot the density of the different features, the different plots shows the range variation of the features, in all the plots there is a pic at 0, and the range of values is between [-10,+10]

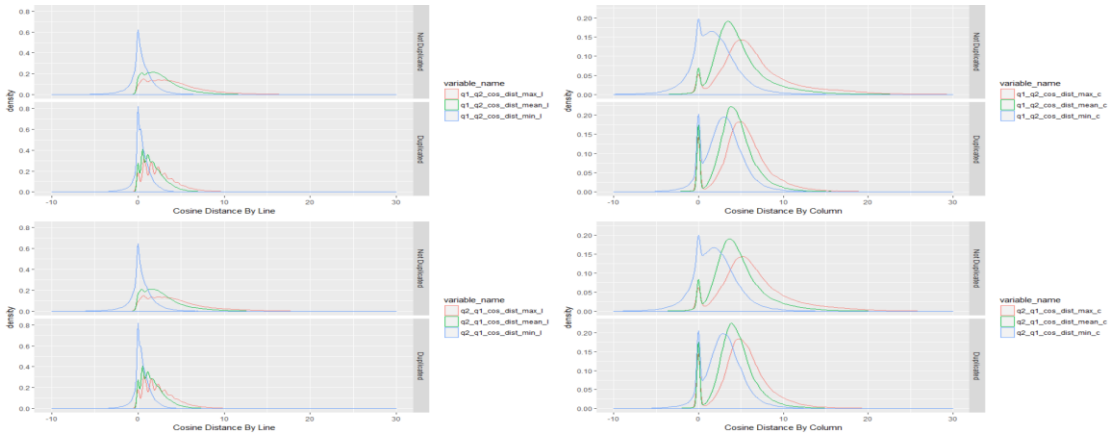


Fig40. Density of calculated features

Let's plot the different features in pairs and see if we can separate the questions which are duplicated visually.

From the pictures, the duplicated questions which are shown with the blue color are closer to the origin (0,0), the pictures in the diagonal is showing location of duplicated question with just one feature in the both X,Y.

The pictures are showing the different mean feature pairs:

- q1_q2_cos_dist_mean_l
- q1_q2_cos_dist_mean_c
- q2_q1_cos_dist_mean_l
- q2_q1_cos_dist_mean_c

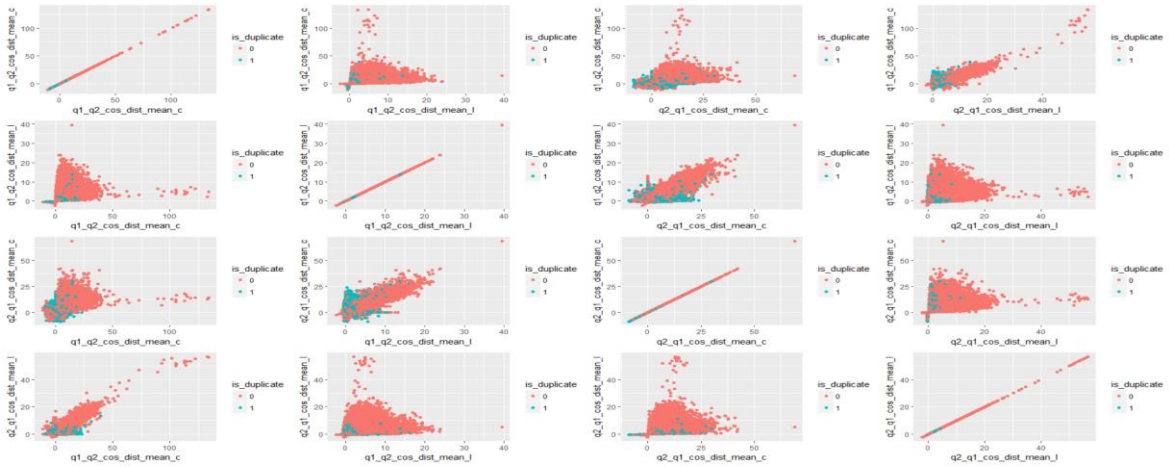


Fig41. Scatter plots of the cosine distance mean features to distinguish duplicated and non-duplicated classes

The above pair plots show the combination of the following column features:

- q1_q2_cos_dist_max_l
- q1_q2_cos_dist_max_c
- q2_q1_cos_dist_max_l
- q2_q1_cos_dist_max_c

again, the duplicated questions are close to coordinate (0,0)

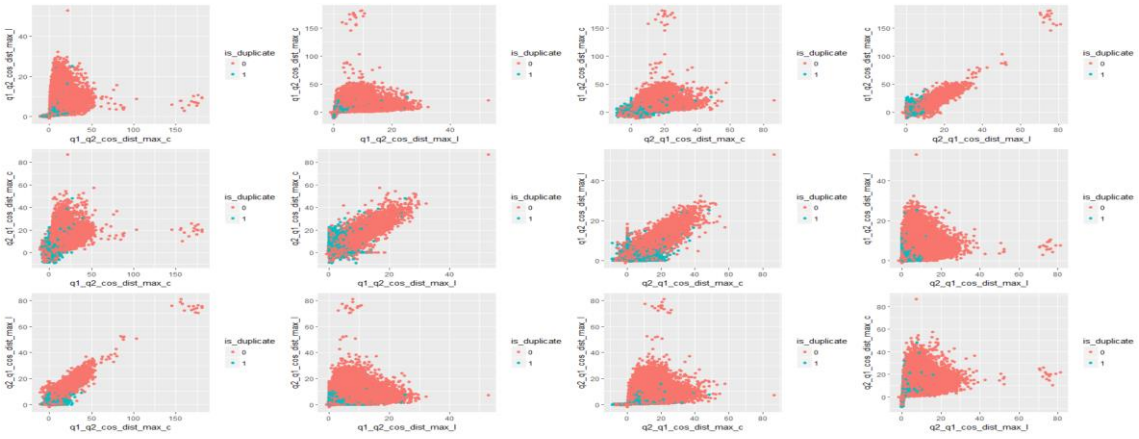


Fig42. Scatter plots of the cosine distance max features to distinguish duplicated and non-duplicated classes

Let's plot the pairs for min features:

- q1_q2_cos_dist_min_l
- q1_q2_cos_dist_min_c
- q2_q1_cos_dist_min_l
- q2_q1_cos_dist_min_c

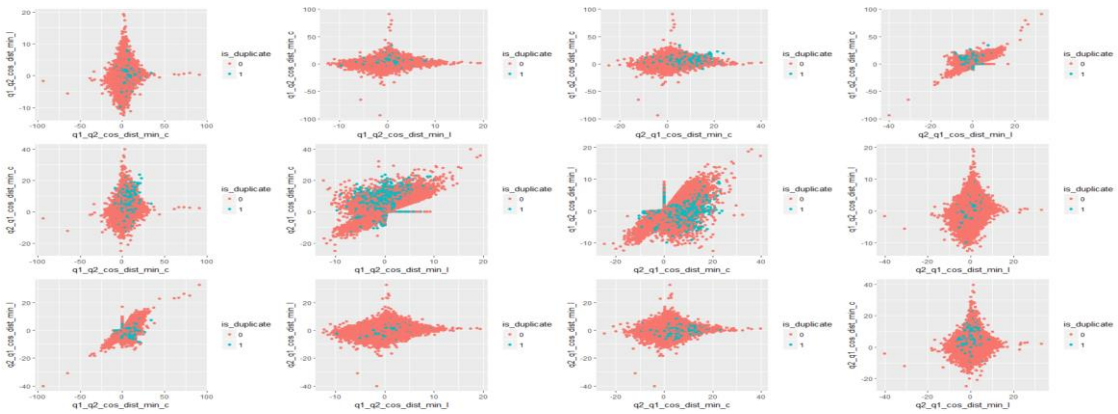


Fig43. Scatter plots of the cosine distance min features to distinguish duplicated and non-duplicated classes

R code:

For the calculation of the features: 'COSINE_DIST.CALC.ATTR' appendix section, result saved in RData file 'question_similarity_distances.RData'

Density, pair attributes and correlation plots: 'SIMILARITY.DISTANCES.PLOTS' appendix section

More features: DTM, TFIDF, LSA, similarity distances, Jaccard:

DTM:

Document term matrix DTM is calculated for the pairs of questions, DTM1 and DTM2 for question1 and question2 columns respectively, both matrices have the dimension of '404290 x 94559' which correspond to 'number of questions in data set X number of words in vocabulary'

Every line of the matrix is one question sentence, and every cell of the matrix contain 0 or 1, 1 means that the word is used in the question, and 0 not.

94559

404290

	nontelgu	guantity	findsearch	.	.	.	how	i	is	the
1	0	0	0				1	0	1	1
2	0	0	0				0	0	0	0
3	0	0	0							
.										
.										
.										
404290	0	0	0				1	0	0	0

Fig44. Document term matrix

The following pipelines describe how the DTMs are calculated.

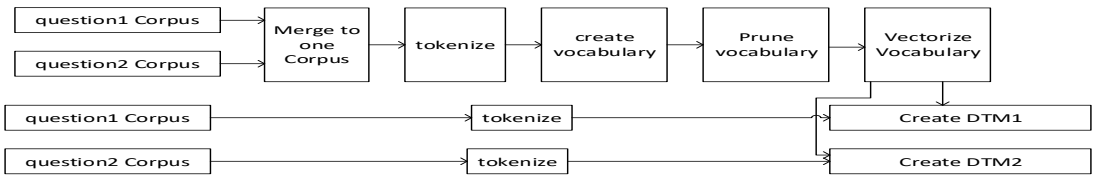


Fig45. Pipline for calculating the DTM matrixes

Now after having calculated the DTMs, we have the statistics of the words usages in all our questions corpus, so we can use it for algorithms like: LSA, and TF-IDF

For the pruning of vocabulary, we set the parameter of minimum counting of terms to 1, so that it does not filter any term according to its frequency usage, we want to consider all the terms

TF-IDF:

Term frequency inverse document frequency, reflects how important is word in a document,

It consists of multiplying two entities:

Frequency of a word appears in a document X Information that word provide to document,

The mathematic formula is:

$$w_{i,j} = tf_{i,j} X \log(\frac{N}{df_i})$$

$tf_{i,j}$ number of occurrence of term i in question j

df_i number of questions containing term i

N number of questions

Every line of our data set will be considered as two documents for the question pair.

So, words like: 'the' will have lower values, and rare words like 'cowboy' will have higher value

We transform the DTM data to TF-IDF format, and get a matrix which will look like as follow

94559

404290

	nontelgu	guantity	findsearch	.	.	.	how	i	is	the
1	0	0	0				0.06678	0	0.0734	0.0695
2	0	0	0				0	0	0	0
3	0	0	0							
.										
.										
.										
404290	0	0	0				0.1335	0	0	0

Fig46. DTM matrix transformed by TF-IDF

We get the TF-IDF matrices as follow:

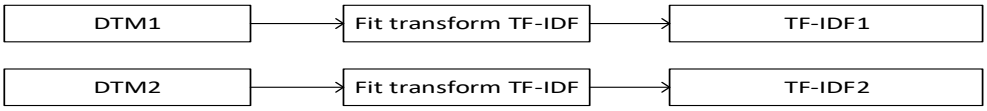


Fig47. Calculation of TF-IDF matrixes

LSA: Latent sematic analytics

The main idea of LSA is to reduce the dimensionality of TF-IDF or DTM matrices, it extracts the relation between the words and questions (documents), we assume that similar words in meaning appears in similar pieces of texts.

It uses the matrix factorization SVD (singular value decomposition)

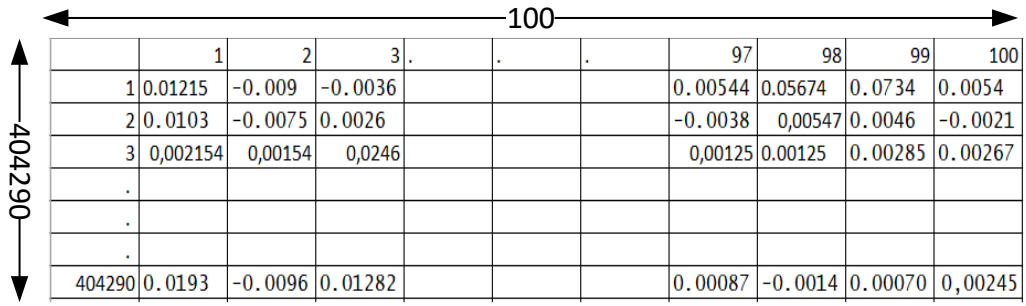
$$X = U\Sigma V^T$$

U: contains eigenvectors of term correlation

Σ: contains singular values of factorization

V: eigenvectors of document correlation

We apply the LSA on the TF-IDF matrix, and we reduce the dimension of the matrix to 100 columns instead of 94559, we will get two matrices, for question pairs columns.



	1	2	3	.	.	.	97	98	99	100
1	0.01215	-0.009	-0.0036				0.00544	0.05674	0.0734	0.0054
2	0.0103	-0.0075	0.0026				-0.0038	0,00547	0.0046	-0.0021
3	0,002154	0,00154	0,0246				0,00125	0.00125	0.00285	0.00267
.										
.										
.										
404290	0.0193	-0.0096	0.01282				0.00087	-0.0014	0.00070	0,00245

Fig48. Matrix of LSA applied to TF-IDF

The pipelines to calculate the two LSA matrices:

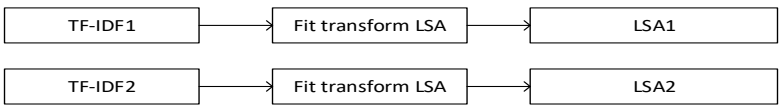


Fig49. LSA pipeline

Apply the machine learnings on DTM, TF-IDF and LSA:

Now that we have calculated the three matrices: DTM, TF-IDF and LSA, there are two ways to apply the machine learning:

- Concatenate the matrices of features
- Cosine distance between pair lines of matrices

Concatenate the features:

The matrices of DTM and TF-IDF are very large, they have the dimensions of '404290 x 94559', if we concatenate the features for the two pair of questions, we will get '94559 X 2' features to feed the machine learning.

We will not proceed with this way as:

- High dimensions of the features, this might be a problem for memory and speed
- Cannot work with the previous we calculated until now

Similarity distances:

We can calculate different cosine distances between every line of both matrices, and the obtained value can be stored as new feature.

There are many distances that can be calculated from the three matrices:

Jaccard distance

$$J(doc1, doc2) = \frac{doc1 \cap doc2}{doc1 \cup doc2}$$

It measures the proportion of common words between pairs of questions

Cosine distance

$$cosine\ similarity(doc1, doc2) = \cos(\theta) = \frac{doc1\ doc2}{\|doc1\|\|doc2\|}$$

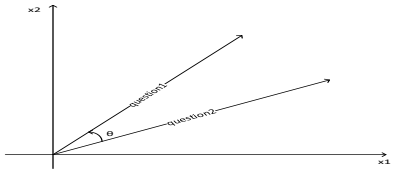


Fig50. Cosine distance between two vectors

It measures how the vectors are pointing to the same direction and tells how the questions are similar.

in the pictures we project the features to 2D dimension to show the idea.

Euclidian distance

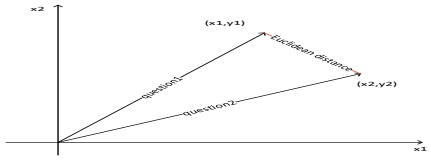


Fig51. Euclidian distance between two vectors

$$euclidean\ distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

Measures the physical distance between two vectors of features

Relaxed Word Mover's Distance(WMD)

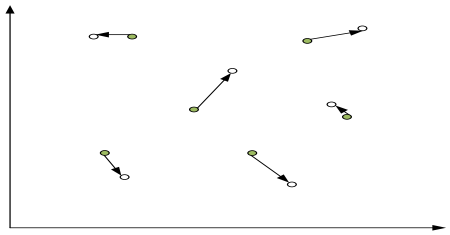


Fig52. WMD distance between words of two sentences

If we have two document A and B, and in every document, cotains a set of words with a vector for each word, every word in document A can move to the closest word in document A, and can have a measured distance, WMD calculate the sum of these distances.

Claculation distances features:

To calculate the different distances, we split the data to many small matrices and then we perform the calculation on these small matrices, this avoid memory issues, as the calculation cannot be performed at once for the whole matrices.

The distance features will be calculated between the pairs of DTM, pairs of TF-IDF and pairs of LSA, after splitting the data.

We choose the packet size to 10 000, this is fine for the memory, and for the calculation speed.

After the calculation, we gather the results in one table, which will consist of multiple columns, where every column represents a calculated distance.

In the bellow diagram, the processing units are shown with green color, the uncolored part are the data matrices.

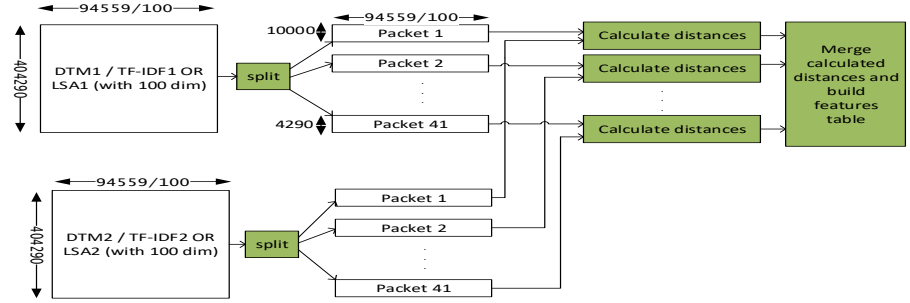


Fig53. Calculation of distances

In the picture bellow, we show in detail the calculation of different features, for every feature we need as input the matrixes mentioned in the picture, and we calculate the similarity matrix using different distance function, the resulted matrix will have the same raws and columns and we take then the diagonal of this matrix as values for our feature, because it will correspond for the pair of questions of every raw in our data set.

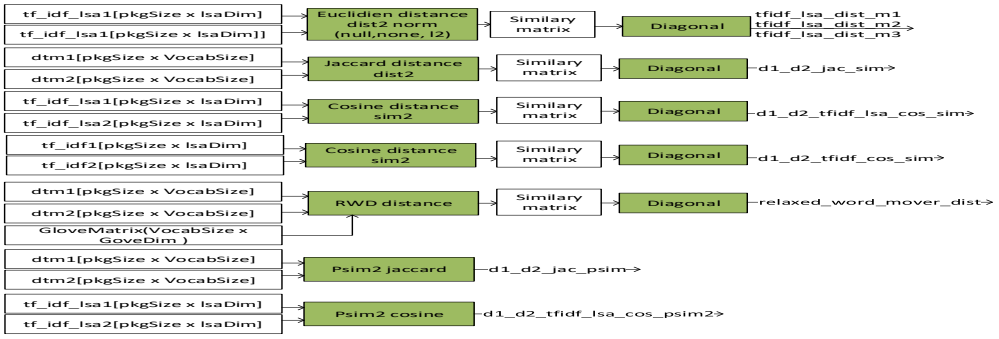


Fig54. More detail how every individual feature is calculated

The calculated data table has the following columns:

- d1_d2_tfidf_lsa_cos_sim: cosine distance between the two LSA matrices
- d1_d2_tfidf_cos_sim: cosine distances between the TF-IDF pair matrices
- tfidf_lsa_dist_m1: Euclidean distances between LSA matrix pairs using no norm
- tfidf_lsa_dist_m2: Euclidean distances between LSA matrix pairs using l1 norm
- tfidf_lsa_dist_m3: Euclidean distances between LSA matrix pairs using ‘none’ norm
- d1_d2_jac_sim: Jaccard distances between DTM matrix pairs
- d1_d2_cosine_sim: Cosine distances between DTM matrix pairs
- d1_d2_jac_psim: Jaccard parallel distances between DTM matrix pairs
- d1_d2_tfidf_lsa_cos_psim2: Cosine parallel distances between LSA matrix pairs

I continue to calculate more additional features related to the original data set of pair of question, so I calculate the following additional fields, I have used the library(text2vec), and the functions: sim2, dist2 and psim2

Check text2vec documentation.

- d1_d2_jac_sim:

we use function dim2 to calculate it, we set the method to “jaccard”, and norm to “none”, dtm

$J(doc1,doc2)=doc1 \cap doc2 / doc1 \cup doc2$

- d1_d2_cosine_sim

function sim2, we set parameters: method = "cosine", norm = "l2", dtm

- d1_d2_jac_psim:

function psim2, parameters: method = "jaccard", norm = "none", dtm

- d1_d2_tfidf_lsa_cos_psim2:

function psim2, parameters: method = "jaccard", norm = "none", dtm_tfidf_lsa

- tfidf_cos_sim:

function sim2, we set parameters: method = "cosine", norm = "l2", dtm_tfidf

- tfidf_lsa_cos_sim

function sim2, we set parameters: method = "cosine", norm = "l2", dtm_tfidf_lsa

- tfidf_lsa_dist_m1

function dist2, we set parameters: method = " euclidean", dtm_tfidf_lsa

- tfidf_lsa_dist_m2

function dist2, we set parameters: method = "euclidean",norm= "l2", dtm_tfidf_lsa

- tfidf_lsa_dist_m3

function dist2, we set parameter method to different values "euclidean",norm= "none", or we supply nothing

We save the calculated attributes in two tables tables: 'df_simelarities.RData', 'Relaxed_Word_Mover_dist.RData'

let's see if we can reduce the features by applying the correlation plot:

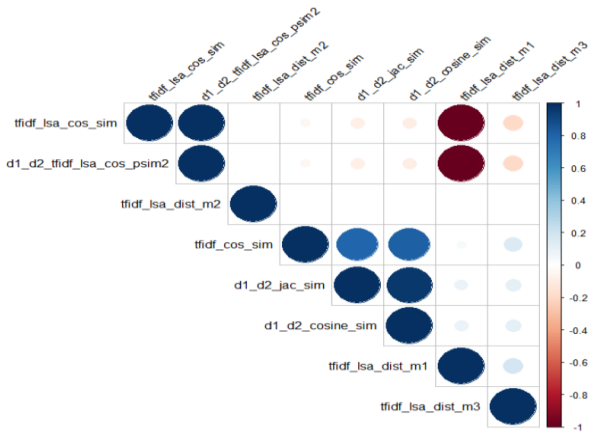


Fig55. Correlation between features before reduction

the following features:

- tfidf_lsa_cos_sim, d1_d2_tfidf_lsa_cos_psim2, and tfidf_lsa_dist_m1
- d1_d2_jac_sim and d1_d2_cosine_sim

Are correlated, so we reduce the list of features to

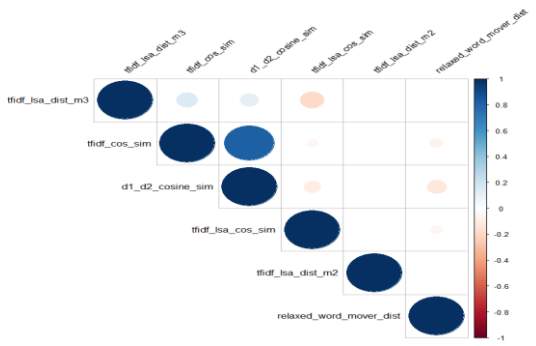


Fig56. Correlation between fetures after reduction

We will take the following features:

- tfidf_lsa_dist_m3
- tfidf_cos_sim
- d1_d2_cosine_sim
- tfidf_lsa_cos_sim
- tfidf_lsa_dist_m2
- relaxed_word_mover_dist

let's plot now, the density distributions for the features, in both situations of duplicate questions and not duplicate questions,

the distribution of tfidf_lsa_dist_m3 have the same shape, but it has higher values above 10, in case of not duplicate.

relaxed_word_mover_dist has higher values in duplicate questions, and it looks thicker.

tfidf_cos_sim, d1_d2_cosine_sim looks different, in both duplicate and non-duplicate situations.

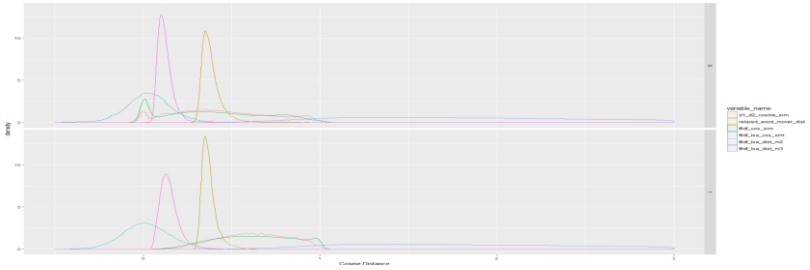


Fig57. Density distribution of calculated features

let's plot the pairs relation between the features and see if we can visually separate, the duplicate questions by the calculated features

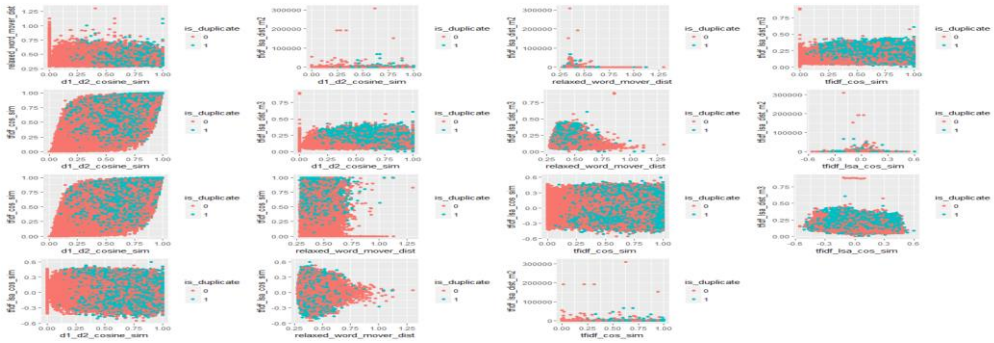


Fig58. Scatter plot for the different pair of features to distinguish the label classes

The new columns can be added to the previous calculated columns as additional features, or we can independently use them in another machine learnings as done previously.

R Code:

Calculate the features in section 'FEAT.DTM.TFIDF.LSA.SIM.DIST.JACC' of the appendix

Plots of: density, correlation and attribute pairs in section 'PLOTS.DTM.TFIDF.LSA.SIM.DIST'

Machine learning:

Merge calculated features to one table:

We collect all the features that we have calculated in the previous sections, as we have saved them in an RData files, we can read them all and put them in the same data set table, that we can feed to the machine learning, the following picture shows all the categories of calculated features.

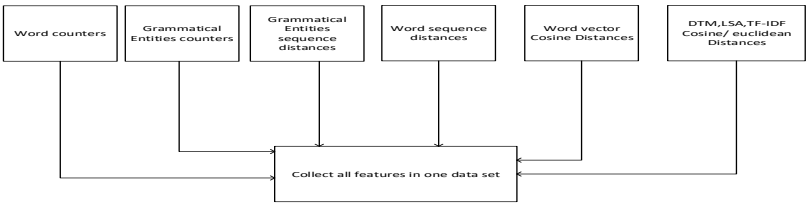


Fig59. Merging all the feature to one data table for machine learning models

For word counters: we have saved them in the following RData files:

- question1_number_of_words.RData
- question2_number_of_words.RData
- question_delta_words.RData
- question_no_stop_words.RData

Grammatical entities statistic: (POS tags) question_gramatical_entities_cosine.RData, question_gramatical_entities_pca.RData

Sequence distances of grammatical entities: questions_Tag_sequences_distances.RData

Sequence distances of words: questions_word_sequence_distances.RData

Word vector cosine distances: question_similarity_distances.RData

DTM, LSA, TF-IDF Cosine/Euclidian distances: df_simelarities.RData,Relaxed_Word_Mover_dist.RData

Every RData file contains a table of the calculated features, it can have one or many columns of data, we provide R function that can return any of these features, together or alone, in addition we add to the returned table, the label column of 'is_duplicate', the name of the function is 'build_table_of_features' it can be found in the appendix.

Reduce features and check redundancies:

We apply the correlation plot to verify if there are still features which depends on other features:

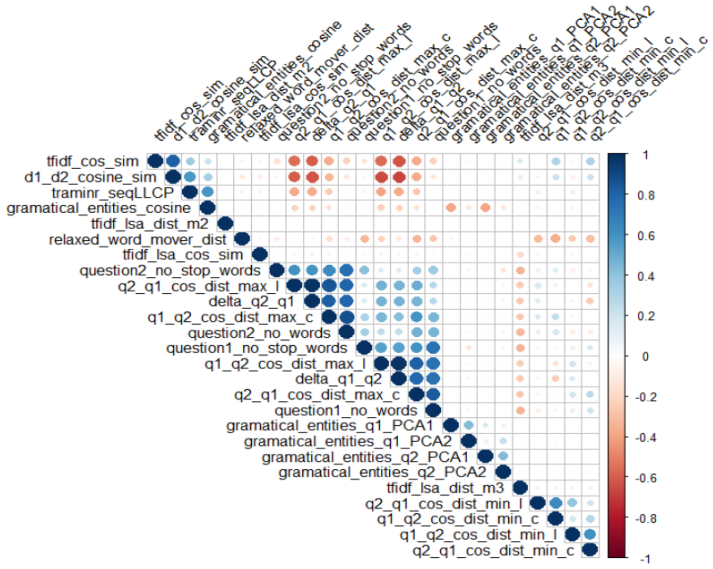


Fig60. Correlation between all the features before reduction

We can remove some of these features, and we reduce the correlation matrix to the following:

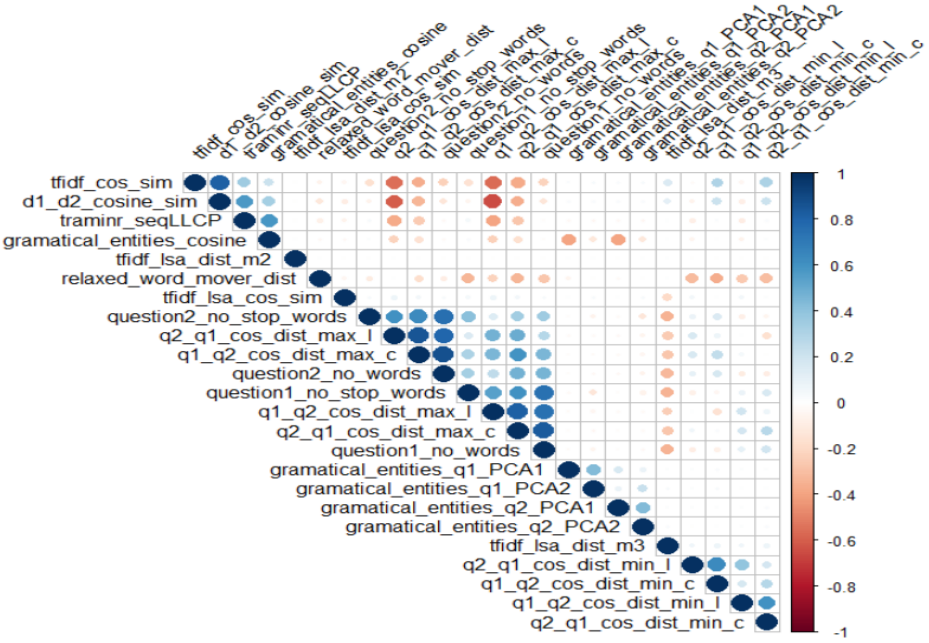


Fig61. Correlation between all the features after reduction

We have removed the correlated attributes: delta_q1_q2 and delta_q2_q1

Learning models:

We apply the following learning models:

GLM: General linear regression

binomial(link = "logit")

LDA: Linear discriminant analysis

QDA: Quadratic discriminant analysis

RPART: Decision tree

Naïve Bayes:

Based on the Bayes theorem, we want to predict if the pair of questions is duplicated or not given the vector of calculated features, the Bayes formula is used to calculate these probability, and with predefined threshold we can classify either 1 or 0.

NNET: Neural Network

It is neural network where the number of input corresponds to the number features, and the number of hidden neurons can be specified with the parameter 'size', we set it to 30

SVM: Support vector machine

Tried SVM but it is taking ages to get results, so I waited for two days, and then stopped it.

Random Forest:

It combines many decisions trees, and aggregates the results to get the average of all the trees, it has mainly two parameters:

ntree = 201, it is the number of trees to grow.

mtry = 10, number of variables to consider in each tree selected randomly.

Ensemble:

We train all the models with the training data (80 %), and then perform the prediction on the remaining 20%, and we measure the accuracy of all models,

After, we sort the models according to the obtained accuracy and select the three best models, with these best models we can build our ensemble model.

The prediction of ensemble for these best model, will be:

$$\frac{1}{3}\ model1\ prediction + \frac{model2\ accuracy}{3*model1\ accuracy}\ model2\ prediction + \frac{model3\ accuracy}{3*model\ 1\ accuracy}\ model3\ prediction$$

The following diagram show how the prediction is calculated for the ensemble model:

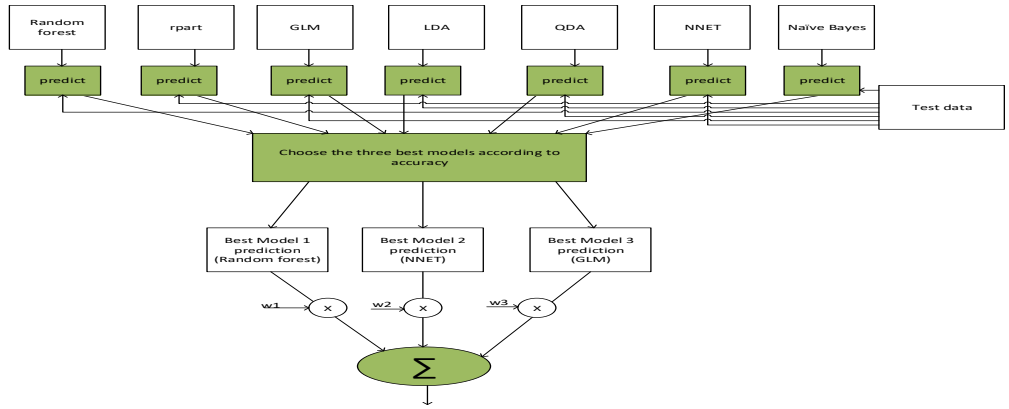


Fig62. Ensemble model based on three best obtained models

The w1, w2 and w3 are the weights of the best models, and their proportion in the prediction.

$$w1 = \frac{1}{3}, w2 = \frac{Model2\ accuracy}{3*Model1\ accuracy}, w3 = \frac{model3\ accuracy}{3*model1\ accuracy}$$

Cross validation:

For the cross validation we split the data to training data where we train different models that we mentioned, we took 80 % of the data for this training, the rest of the data which is 20%, we used them to test the obtained models.

We plot get ROC data, and plot the following curves:

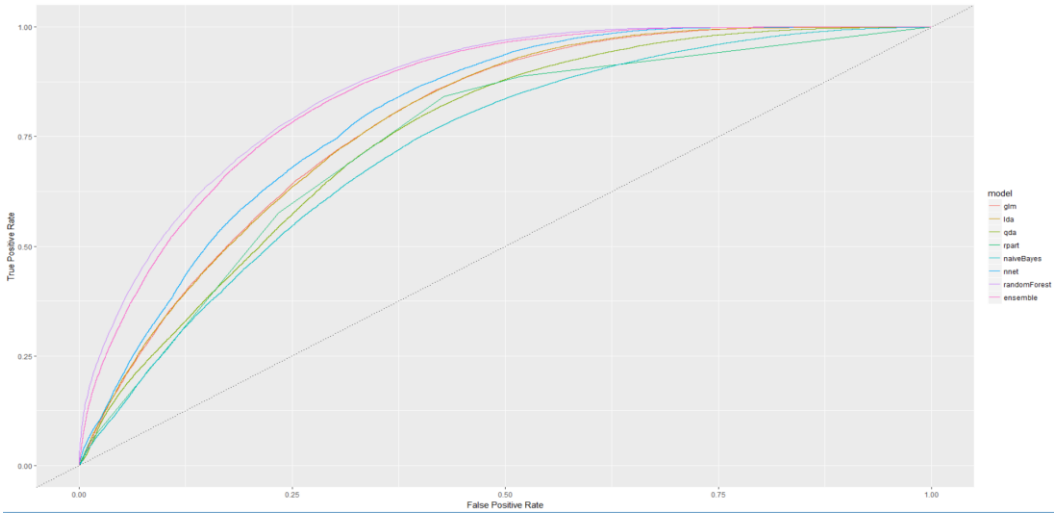


Fig63. ROC curves for all models obtained from cross validation

Which shows that Random forest and Ensemble models are the best

The error of the different models are shown bellow:

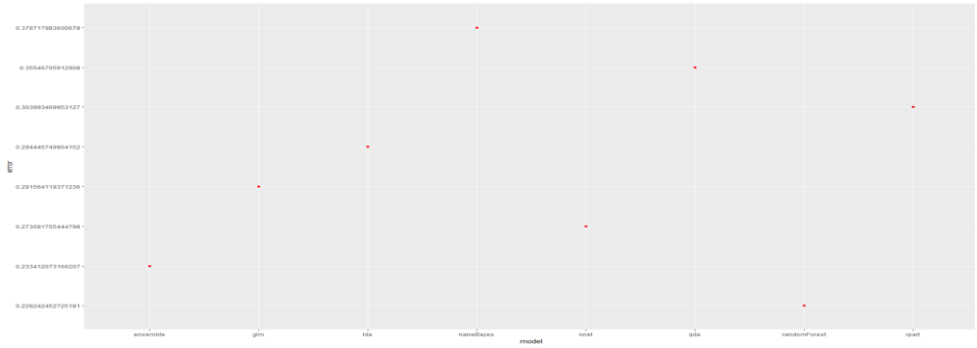


Fig64. Value of errors obtained for all models with cross validation

The accuracy of the different models is plotted bellow:

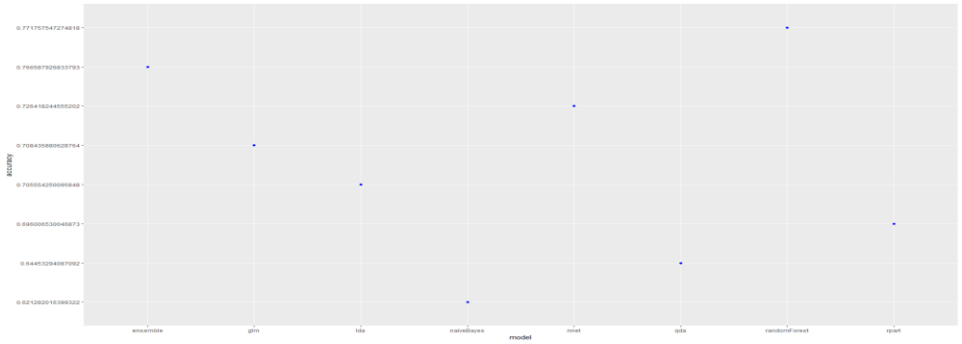


Fig65. Value of accuracy obtained for all models with cross validation

The confusion matrices for the different models, showing the number of pair of questions in the 4 categories of: false posive, false negative, true positive and true negative, these last categeories are displayed as the diagonal of the matrix.

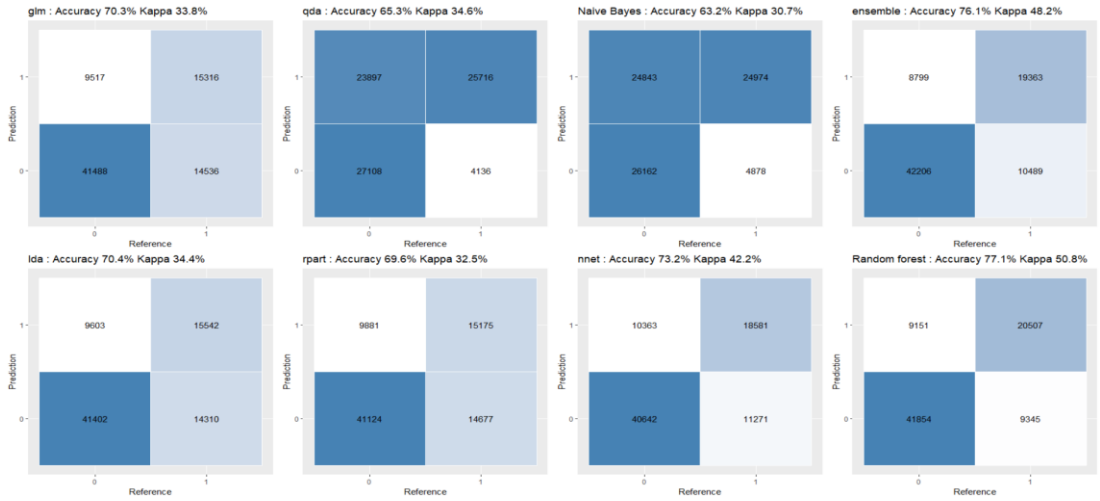


Fig66. Confusion matrixes obtained for all models with cross validation

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-2.627508e+00	6.317262e-02	-41.592509	0.000000e+00
tfidf_cos_sim	1.769133e+00	2.971240e-02	59.541919	0.000000e+00
tfidf_lsa_dist_m3	8.788354e+00	1.024943e-01	85.744835	0.000000e+00
d1_d2_cosine_sim	1.904827e+00	5.166142e-02	36.871370	1.329578e-297
q2_q1_cos_dist_max_c	1.020895e-01	2.959444e-03	34.496167	9.156482e-261
q1_q2_cos_dist_max_c	6.752258e-02	2.867707e-03	23.545839	1.384827e-122
q1_q2_cos_dist_min_c	5.720073e-02	2.621569e-03	21.819268	1.522760e-105
traminr_seqLLCP	-6.704190e-02	3.168925e-03	-21.156039	2.427842e-99
relaxed_word_mover_dist	-2.402042e+00	1.148107e-01	-20.921765	3.393213e-97
question1_no_words	-5.313057e-02	3.541201e-03	-15.003547	6.959874e-51
question2_no_words	-5.124370e-02	3.452775e-03	-14.841310	7.919425e-50
gramatical_entities_q1_PCA1	-7.047375e-02	5.156293e-03	-13.667522	1.587079e-42
gramatical_entities_q1_PCA2	7.310244e-02	5.603884e-03	13.044961	6.789160e-39
gramatical_entities_q2_PCA1	-6.464375e-02	5.144279e-03	-12.566144	3.241512e-36
q2_q1_cos_dist_min_c	2.670131e-02	2.619684e-03	10.192570	2.140303e-24
gramatical_entities_q2_PCA2	5.437299e-02	5.572985e-03	9.756529	1.729747e-22
q2_q1_cos_dist_max_l	-6.212929e-02	6.577015e-03	-9.446427	3.505956e-21
q1_q2_cos_dist_min_l	-4.820262e-02	5.323106e-03	-9.055356	1.361229e-19
q2_q1_cos_dist_min_l	-4.539417e-02	5.259461e-03	-8.630955	6.084165e-18
question1_no_stop_words	-2.989623e-02	4.140743e-03	-7.220017	5.198112e-13
tfidf_lsa_cos_sim	2.380093e-01	3.301287e-02	7.209590	5.612055e-13
gramatical_entities_cosine	-1.739241e-01	2.763661e-02	-6.293252	3.108831e-10
q1_q2_cos_dist_max_l	-3.889829e-02	6.672904e-03	-5.829289	5.566411e-09
question2_no_stop_words	1.871980e-02	4.053191e-03	4.618534	3.864611e-06
tfidf_lsa_dist_m2	-1.686982e-05	9.184795e-06	-1.836712	6.625247e-02

Fig67. Some metrics with P-Value column for GLM model showing all the features

For the GLM model, the above table show some metrics for the different attributes, the last column is showing the P-Values, where Most of them are very small, which means that they confirm the H1 hypothesis and reject H0 hypothesis, in other words, the different columns of the data set are affecting label prediction.

Only the column 'tfidf_lsa_dist_m2' has relatively a high value, we can experiment removing it from the data set, and check if we are still getting similar accuracy for the different models.

Cross validation with K fold:

We set the K fold = 5,

We split the data into 5 folds, and we build the models based on the data of $5 - 1 = 4$ folds, and we do the tests on the remaining 1 fold, we repeat the process 5 times, and in every time we choose another fold for testing data, and the rest for training the model, at the end of the process we take the averages of:

Accuracy, error, recall, precision, and ROC data.

With the resulted ROC data, we can plot the following ROC curve for every model.

Random Forest has the best ROC curve, as it is the farthest and widest from the diagonal line, and it is followed by ensemble model which not far.

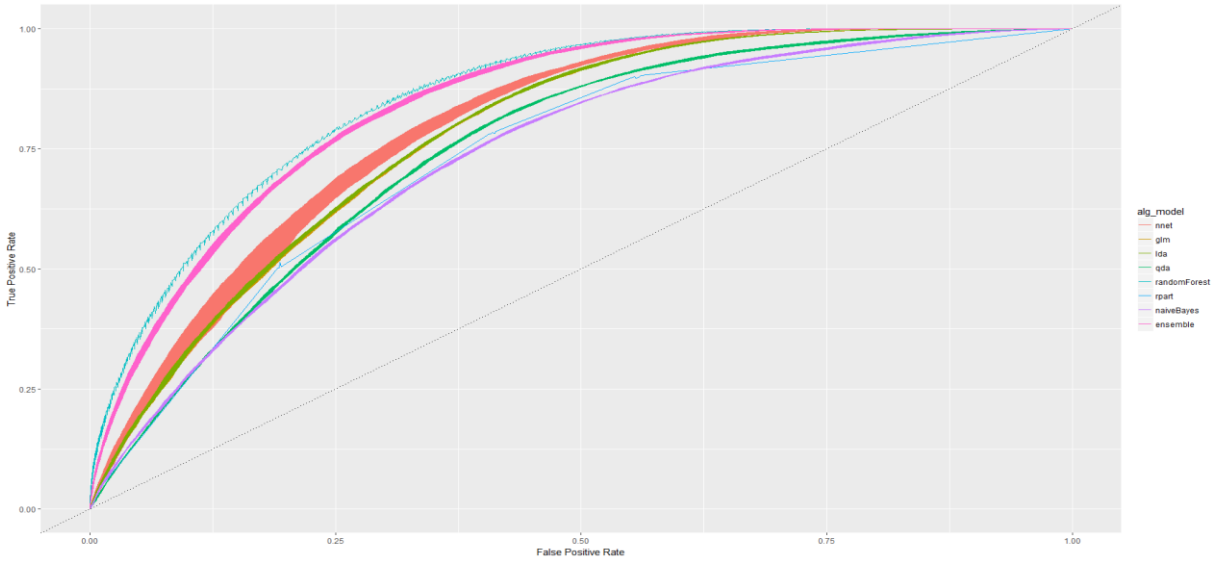


Fig68. ROC curves for all models obtained with K-fold cross validation

The following table summurize the perforomnce of the different models that we trained, where we sorted the models by the accuracy, and the winner is always Random Forest model.

alg_model	accuracy	error	Recall	precision
randomForest	0.7702046	0.2297954	0.6890705	0.6880603
ensemble	0.7604813	0.2395187	0.6226111	0.5434812
nnet	0.7199833	0.2800167	0.6232876	0.6104028
lda	0.7027826	0.2972174	0.6158431	0.5182572
glm	0.7011378	0.2988622	0.6145994	0.5108737
rpart	0.6956665	0.3043335	0.6053713	0.5046903
qda	0.6434342	0.3565658	0.5099755	0.8744352
naiveBayes	0.6265972	0.3734028	0.4966680	0.8491984

Fig69. Some performance metrics for all models obtained with K-fold cross validation

R code:

It can be found in the appendix section '# MACHINE.LEARNING.MODELS',

It contains several parts: building the models and cross validation, build ensemble model, partition the data, k fold cross validation, plot ROC curves, and plot confusions matrixes.

Conclusion:

We started with original data set which consists on only texts, and we calculated some features from these texts, and that's what they call feature engineering, it is somehow projecting of the original data to other dimensions, or the dimension of calculated features, where the features are numbers and real values, where it is very easy to apply machine learning and all to apply all visualization technics, the drawbacks are:

- The calculation of the features, needs special knowledge, to know what to calculate and how to calculate.
- The calculations need a huge amount of time

The first time we used our intuition to find the features to calculate, and we also found some very helpful resources from the internet.

For the calculation time, we didn't calculate all the attributes at once, so calculated them by groups, and grouped the ones that look similar in the same function, and after getting the result we save them temporarily in 'RData' format, until we get them all.

We applied different machine learnings to the obtained features, and the random Forest model is giving the best results in the cross validation and has the best ROC curve, on the other hand ensemble model also is doing very well even it gets slightly under random forest, it can give a better result than random forest, this will depend on the other three best models how they are doing.

Some models are time consuming, so didn't get results for them, like SVM model, this is because of the size of the data set as we have '404290' records.

We have experienced that the combination of all features has increase the accuracy, the possible future improvements are to find other new feature fields that can improve the accuracy to more than 77% that we get, and we need to tune the parameters of the best models like random forest and compare the accuracy.

Neural Network:

In the previous chapter we were extracting features from texts, and then we applied different machine learning models, in this chapter we will apply deep learning models implemented in Keras libray, we will let the task of extracting the features to the neural network layers, and we will just transform the data set to the right format so that it can be consumed by the deep learning models.

Auto-encoders:

We will start with the autoencoders, as they are very important in all domains of data science and artificial intelligence, the main object for us is to transform the pair questions texts, to a lower dimension vector, these vectors should be full of the information concerning the sequence of words in sentence in addition to the word semantic meaning.

We need an embedding layer that act like a loockup table where we store the word vectors, we need LSTM, GRU or Convolution layers to catch the sequence of words.

The following diagram summarize the different pipelines and layers needed.

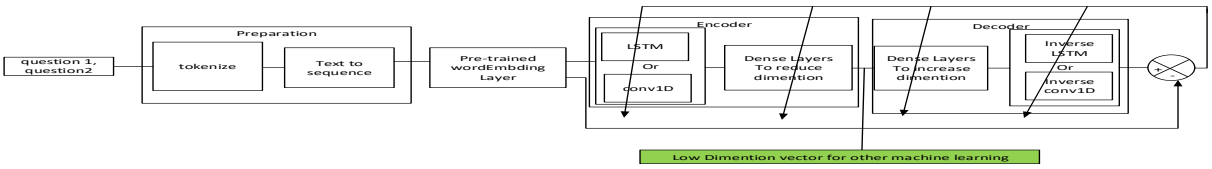


Fig70. Pipline of needed layers for auto-encoders

We use a pretrained Word embedding layer.

The pipeline in the autoencoder will consist of three parts:

- Preparation of data to neural network block: tokenization, text to sequence, pretrained Word Embedding
- Encoder: LSTM, GRU or conv1D, Dense Layers to reduce the dimension of every question
- Decoder: Dense layer to increase the dimension, inverse LSTM, GRU or conv1D

We will train the autoencoder with the output of Embedding layer, and the output of neural network, the error will be as shown in the picture = Output of Neural Network – Output Preparation Layer

We will use the output the encoder, as a vector with reduced dimension, and the vector will present our question, and we use it as input for other machine learnings.

The embedding layer will be outside the autoencoder, and it will have its own model, which will convert the sequence of words which numbers, to a matrix of sentence, it is not possible to reverse the operation of embedding layer, by giving it matrix and return the word sequences, this is the main reason why it is outside the autoencoder model.

Use LSTM or conv1D to consider the sequence of the words.

We use keras library, as it is high level and ease to use comparing to tenserflow, also it is a wrapper that can use tensorflow, theano or CNTK to Microsoft, in the backend, it has very clear API that make very easy to use.

With the pipe concept of R, it will be very easy to build a sequence of deep learning layers, quickly, and which can be understandable just in one sight.

Layer1 %>% Layer2 %>% Layer3 %>% Layer4

We will need one layer that can vectorize the words of the question sentences, we will use embedding layer.

To train the sequence of the words in sentence, we add LSTM, or GRU layer.

What is embedding layer:

The embedding layer will vectorize a word into a vector of given dimension, like 100, depending on the embedding dimension:

for example:

if we take a sentence, than this will be presented with a matrix, with format: [number of words x 100]

as the number of words will be different in every question sentence, and to all the word vectors in the same matrix, then we measure the largest value of number of words, and we take as the second dimension of our word matrix.

[Max number of words(= 250) x embedding dimension(= 100)]

When the sentence has less words than 250, we fill the rest of words with dummy zero vector.

In our data set, we have 495 650 pair of questions, so 495 650 questions will be presented by a three-dimension matrix, which will be the output of embedding layer:

[Number of questions (= 495 650) x Max Number of Words (= 500) x Embedding Dimension (= 100)]

To produce such matrix, we need to load a pretrained words vector matrix to the embedding layer

Prepare the data for neural network:

Every word in sentence can have an index number, which correspond to the index of the word in the matrix of word vectors.

If we give the index of the word to the embedding layer, it will return the vector of the word.

If we give a vector of word indexes, like: 45564,5465,8898 …, the embedding layer will return a matrix with all vectors.

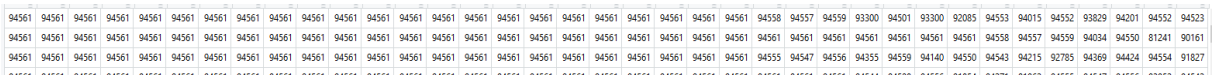


Fig71. Word sequence of the question sentences

94561 correspond to empty word, as you can see in the picture above, the first numbers in the sequence vector are set to this number, as we are constructing the sequence of words in the sentences using a vector of length 512, the words of the sentence are filled at the end of the vector.
We need to convert all the words in given sentence to their corresponding sequence of the word indexes.

Preload word vector to embedding layer:

We can load word vectors from the intenet, where the word vectors exist and already calculated, they are stored in text file, as shown in the bellow picture, the dimension of each word vector is 100, and they exists in other dimention formations like: ‘50’,’200’,’300’.

Fig72. Precalculated word vectors sotred in text document

These vecotors cann be obtained at site: ‘http://nlp.stanford.edu/data/glove.6B.zip’

On the other hand, we take the question sentences from the two columns of our data set, question1, and question2, we put them all in the same table, and then we apply to them the glove algorithm with parameters:

- Word_vector_size = 100
- Number of iteration = 20

We obtain our word vector of 100 dimension

To understand how Glove Algorithm work, you can read some more documentation at the internet, and we have already mentioned it in the first section.

The main idea is to give to every word a vector of chosen dimension, and the words which are close emetically, will get vectors close geometrically, and this can be shown with projection in 2D space using T-SNE.

After getting the word vectors with Glove, we look inside the pre-calculated word vector of file that we get from file ‘glove.6B.100d.txt’, if the word is already inside then we take its vector, otherwise we take the vector that we have just calculated.

We put all the vectors in one matrix, and we load it to the embedding layer.

We add zero vector at the end of this matrix, we use the index of this vector when we build the word sequence when we build the input of neural network.

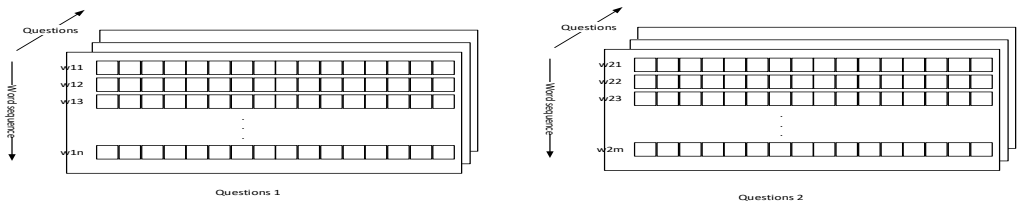


Fig72. Pair of question transformed for the deep learning models

Build sentence word sequence and input of embedding layer:

We feed the input of embedding layer with the word sequence indexes, every sentence will consist of indexes of the words which are in.

The index of word, is the line number of the word vector.

Sentence: word1 word2 word3 … word10

We get the index of every word, and we construct the sequence:

Index1, index2, index3 … index10

Every index is number between:1 to vocabulary length(94562)

We put all these sequence to the same matrix, and as the number of words is different in every sentence, we choose 20 or 30 as the maximum length for the word sequences.

In case the sentence has less words then the max we set, then we fill the rest with dummy value of 94561.

So our sentence will looks like:

94561 94561 94561 94561 94561 … 94561 Index1, index2, index3 … index10

Or

Index1, index2, index3 … index10 94561 .. 94561 94561 94561 94561 94561

Here is a real example:

404232	94561	94561	94561	94561	94561	94561	94561	94561	94561	94561	94561	94561	94561	94561	94557	67810	68955	94527	92893	93452
404233	94561	94561	94561	94561	94561	94561	94561	94561	94556	94538	94220	92483	94552	94150	94537	93861	94528	94556	94166	94553

Fig73. Word sequences for two question sentences

We obtain the matrix input of the embedding layer, with number of lines corresponding to the number of examples, and number of columns equal to 20 or 30.

The output of the pretrained embedding layer will be a three-dimensional matrix, as every index is a vector in the embedding layer.

Used deep learning Layers

LSTM layer:

We add an LSTM layer to our autoencoder, to take account of word sequence in a sentence, we will also experiment with GRU layer as it needs less computing operations than LSTM.

GRU Layer:

Dense Layers:

Conv1D:

Maxpooling1D:

Upsampling1D:

Autoencoder complete model:

In the autoencoders in general we are trying to predict the input, so the prediction and input should be the same.

Transforming the input by applying it to different deep learning layers, will need inverse transformations so that the output will look the same, for this reason in the autoencoder we find the layers duplicated, so that the first in transforming and the second similar layer is doing the inverse.

For our problem we need an embedding layer, which is a must for semantic text, as it vectorize our text to the right format, and it seems that we cannot invert the embedding layer, because its input and output are different format, the input is sequence of numbers, and output matrix of vectors.

So we will separate between embedding layer, and the rest of the autoencoder, and the output of embedding layer will feed the autoencoder, this is fine, because we are not training the embedding layer, as we preload it with a pre-trained weights.

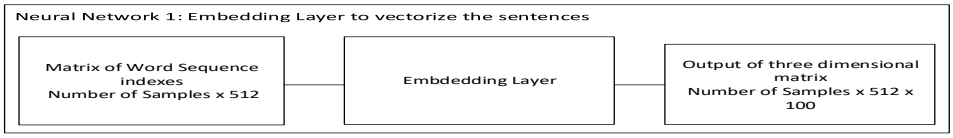


Fig74. Pipline of Embedding Layer

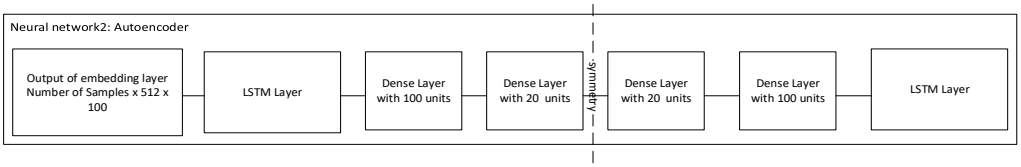


Fig75. Pipeline of auto-encoder using LSTM and Dense Layers

How to reduce the memory usage:

The output of embedding layer is three-dimensional matrix and applying it to all the data sets pair of question, will be very expensive in memory.

So instead of transforming all the data at once with embedding layer, we will do the training and validation of deep learning model with chunks.

We divide our data to packages of size '2000', and we take 1% of it for the validation which mean 200, the rest of 1800 will be for training.

We do this in iteration for all the packages, and we store the history, to draw some graphs about mean square error, and accuracy.

Optimizer = 'addadelta'

Loss = 'mse'

Metrics = 'acc'

How to choose the data for training and tests

As we cannot hold all the three dimensional output of embedding layer in the memory, so we will not train the autoencoder at once with all the data, we split the data in packages of 2000, and we take them randomly form our data set, after that we split it randomly to the 10% and 90% for training and validation data, as shown below:

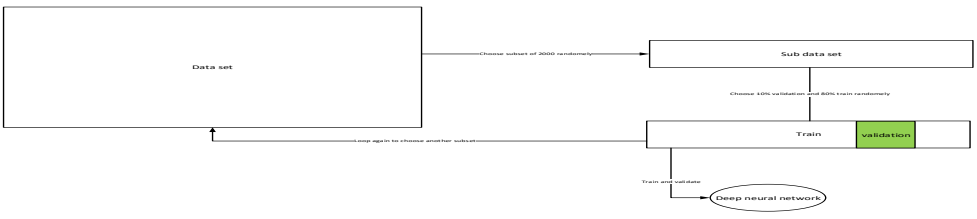


Fig76. Train and validate the auto-encoder with sliced data

we send the training and validation data to the embedding layer, this will give us the input data of our autoencoder.

The inputs of the embedding layer will be matrixes of training and validation data set, and the output will be three-dimentional matrixes, where the new dimention will be the word vectors.

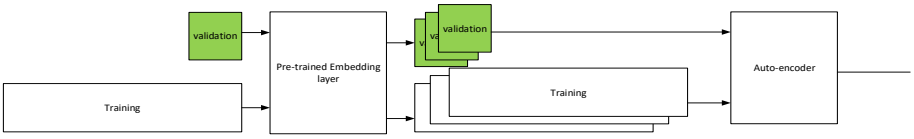


Fig77. Pipline for train and validate the auto-encoder

We repeat again by looping and chose another data subset, and we do the same process, many times, at least 404290/2000 = 202 times.

We can also use another way to train our autoencoders, by splitting the data to packages of size 2000 not randomly and we split then to 10%/90% validation/training randomly:

Fig78. Slicing the data set into smaller packages to avoid memory problems

The 202 iterations will a time to finishes in both training methods, so I prefer the first method because it will give a better autoencoder if we stop the learning and validation earlier.

Different models:

Model1:

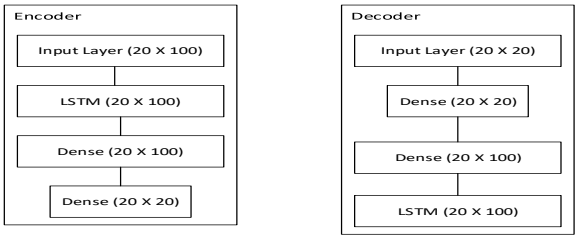


Fig79. Model1 of auto-encoder

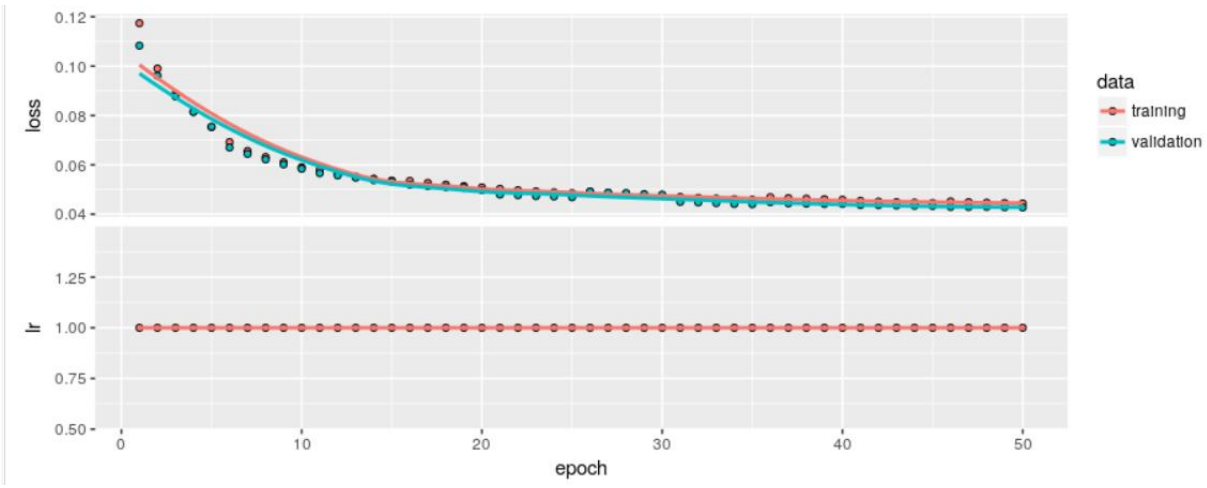


Fig80. Loss curves for training and validation in addition to learning rate

Model2:

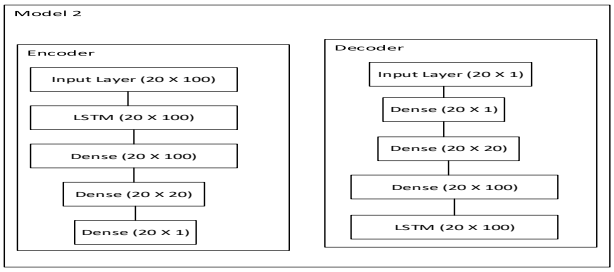


Fig81. Model2 of auto-encoder

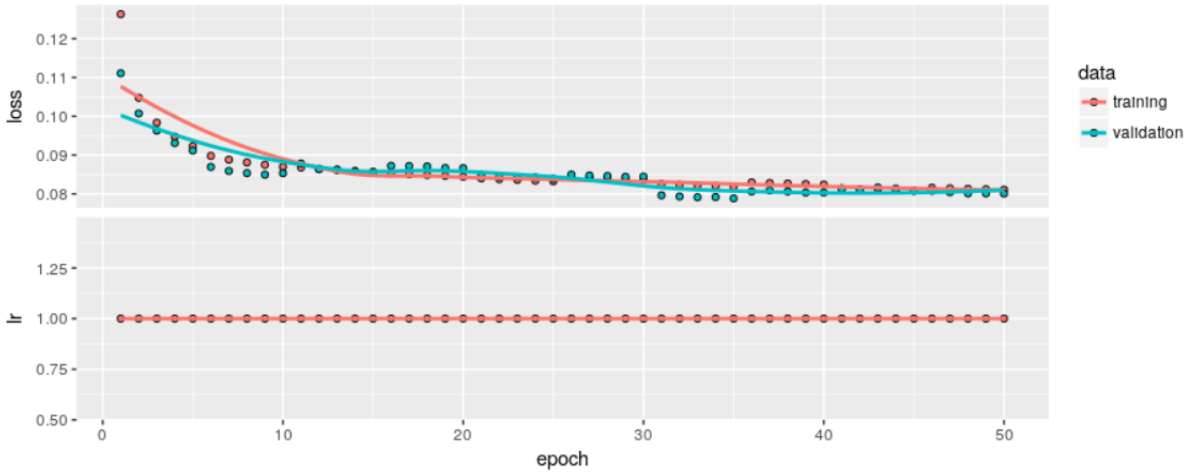


Fig82. Loss curves for training and validation in addition to learning rate

Model3:

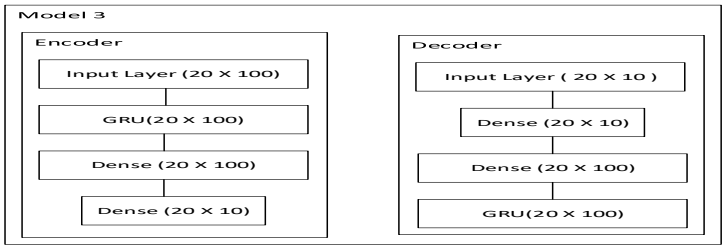


Fig83. Model3 of auto-encoder

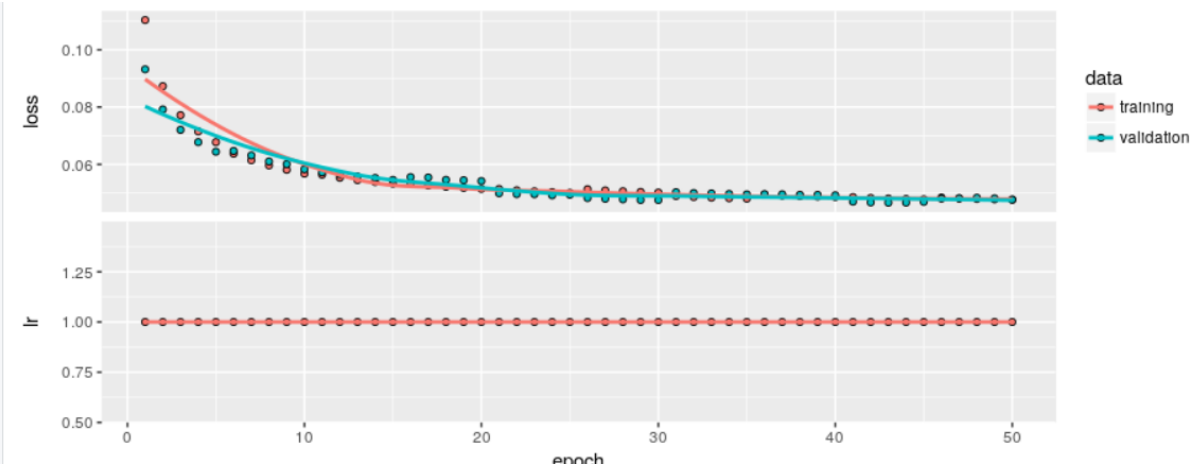


Fig84. Loss curves for training and validation in addition to learning rate

Model4 :

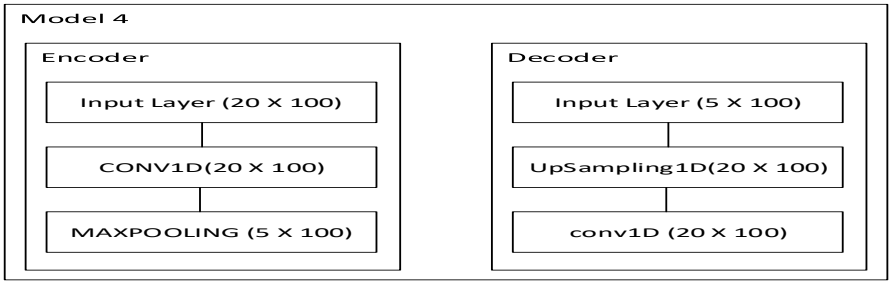


Fig85. Model4 of auto-encoder

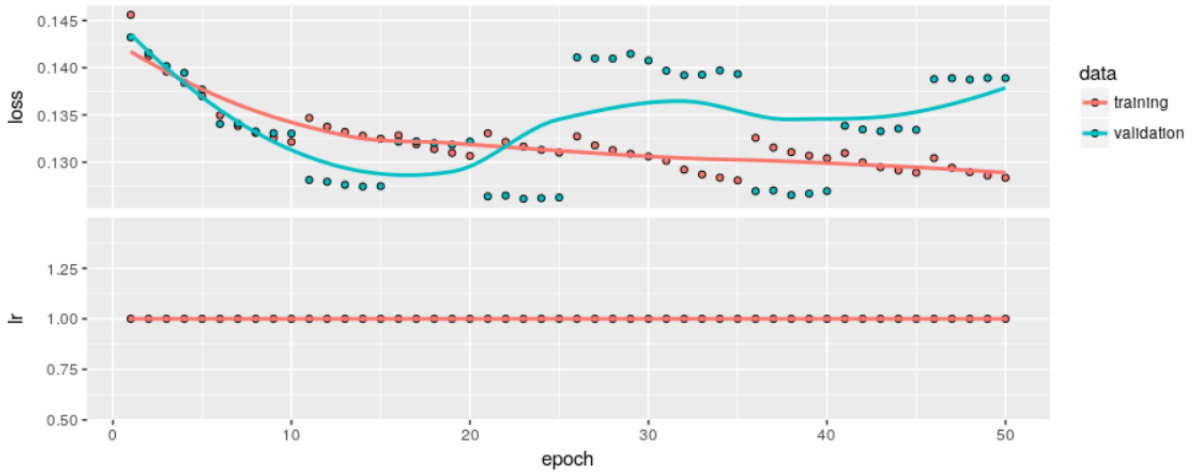


Fig86. Loss curves for training and validation in addition to learning rate

Model5 :

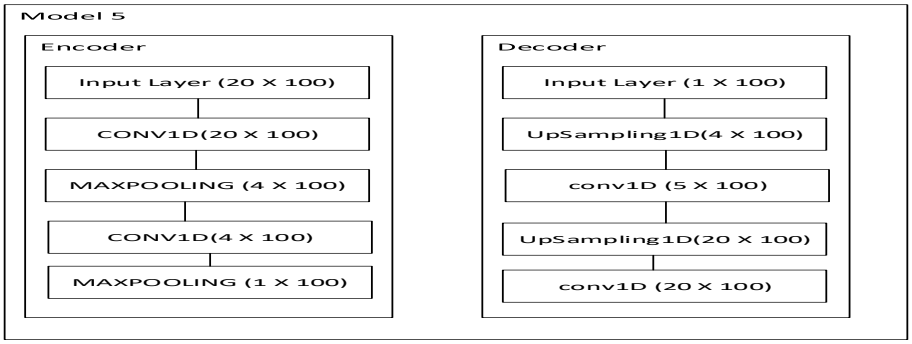


Fig87. Model5 of auto-encoder

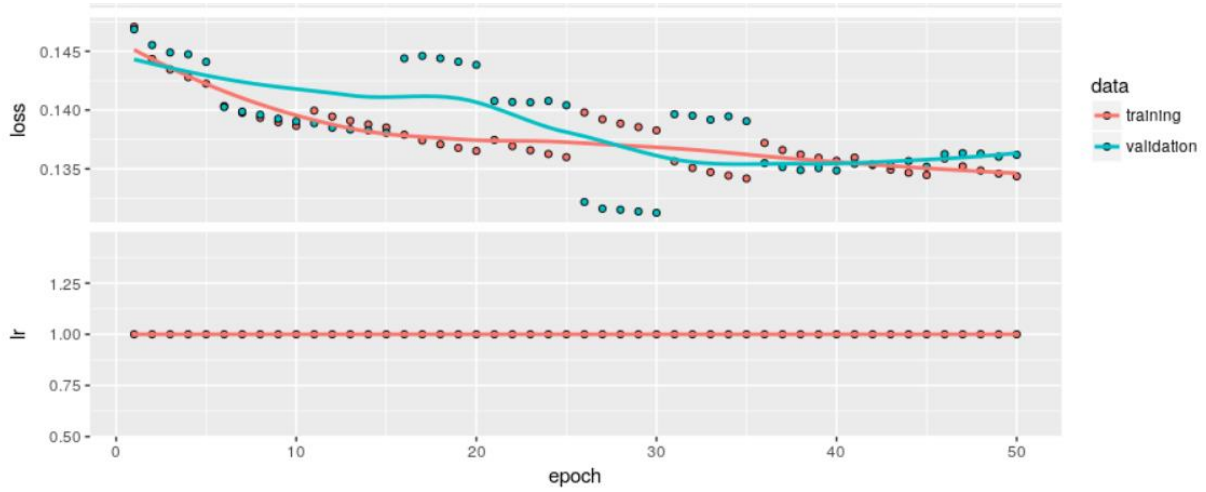


Fig88. Loss curves for training and validation in addition to learning rate

Remove embedding layer:

We have seen that embedding layer need a lot of memory

30,6 Megabytes for 2000 entry

6.2 Gigabytes for all field question1 sentences

So, I could not hold all these data into the memory, so instead of doing the autoencoder at level of sentences, we can do it at level of words, so that we reduce the dimension representation of every word, and let's choose the target dimension one, so that every word will be represented with one decimal value, these can be achieved by two ways:

- Deep learning autoencoder
- Apply T-SNE to the Glove vectors (we develop these at the end)

With only one decimal value for every word, then we do not need to have an embedding layer

R Code:

The R Code of the different models and their cross validation is in appendix section: '# Autoencoders'

Conclusion:

Model 1 is the best

Memory problem to calculate with embedding layer the input of the autoencoders for all data sets

Improve run time with pretrained embedding layer.

Need to check the autoencoders at level of words, to reduce the dimension vector of one word from 100 to 1, by doing this we can remove the embedding layer, and we can train on all the data set at once.

Auto-encoder for word vectors:

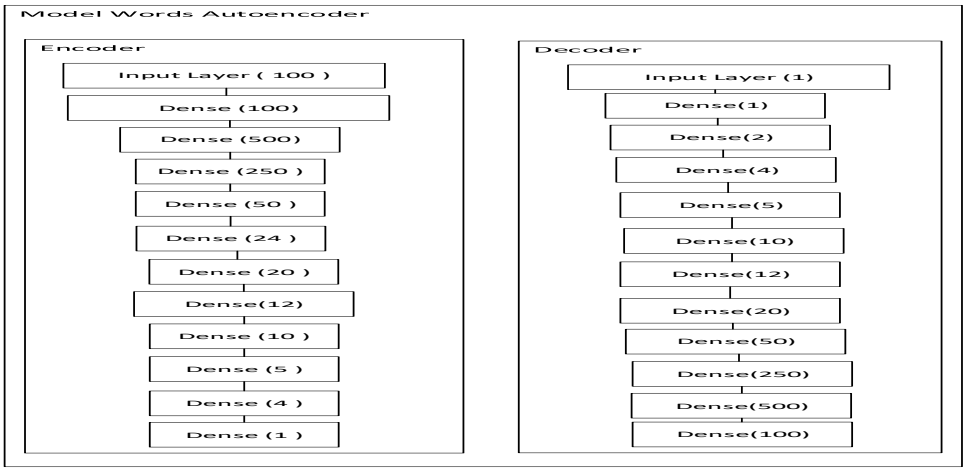


Fig89. Auto-encoder for word vectors

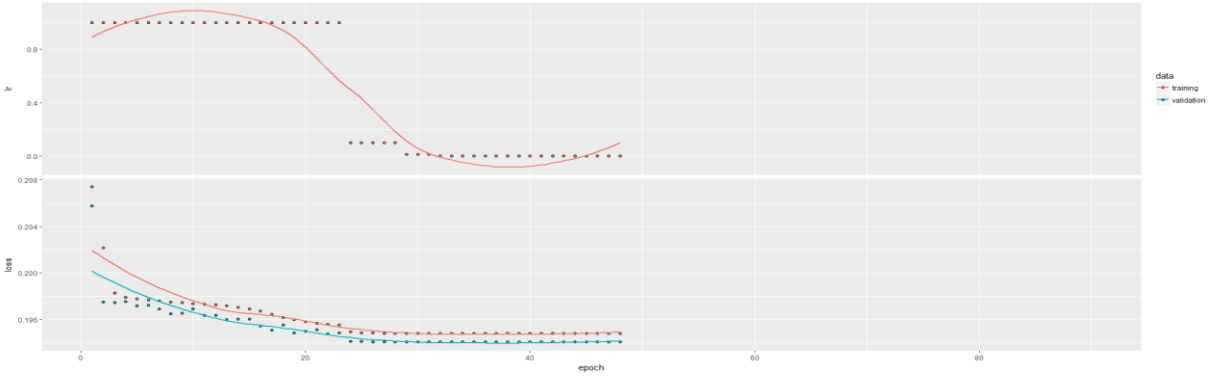


Fig90. Loss for validation and training data and learning rate curves

The loss is not three zeros after the virgule, so there will still losses after training the model.

Which mean that we will get duplicate values for the words, I have counted 176 values over

‘94562,

The other issue is that the generated decimal value will not reflect the semantic distance between the words.

Deep Learning classifiers:

We will try different deep learning architecture models, and we will compare them to choose the best, by using metrics like ROC curves.

In the models: model1 until model5 we set the embedding layers as trainable, and we train them during the training of the whole model.

In the modeles: model6 until model10, we will load a pre-trained embedding layer, it will be shown with green color just to make the difference with the trainable layers.

All the models will have two inputs, every input enters a question, and two questions can be entered at the same time.

After the enty layer, comes the embedding layer.

We will describe the model architectures with visio flow digrams, and we will plot the accuracy and losss of every model before introducing 'early_stopping' and 'drop_out' layers, and at the end we plot all together the accuracy and loss after adding the 'early_stopping' and 'drop_out' layer.

Every input of the models will be feeded with matrix of 404290 X 20 which contain the word sequences in every question sentence, this was described in previous section.

We compile the models with the following parameters:

- optimizer = "adam"
- loss = "binary_crossentropy"
- metrics = metric_binary_accuracy

For the training we use the following parameters:

- batch_size = 128,
- epochs = 30,

The first time we trained the models with early stopping, and we let them run all the 30 epochs.

After we have introduced the following callback functions:

- callback_early_stopping(patience = 5)
- callback_reduce_lr_on_plateau(patience = 3)

The first function will stop the training of the models when the accuracy didn't improve after certain epoch times, controlled by the patience parameter.

The second function, will do the same and will have the effect on the learning rate parameter, it will stop when it does not improve.

Both functions will help to fight against the overfitting.

The validation data represent 10% and the training 90%.

Model1:

We show the different layers of the model.

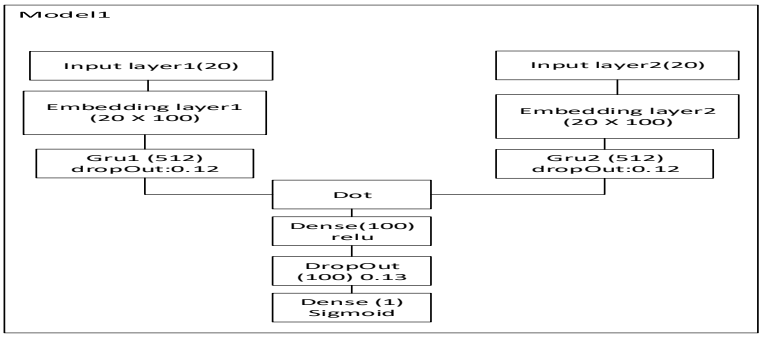


Fig91. Model1 for classifier

The following graph shows the accuracy and the loss graphes during the different epochs, we can see that it continues until 30 epochs as there are no early stopping, and there is no dropout Layers.

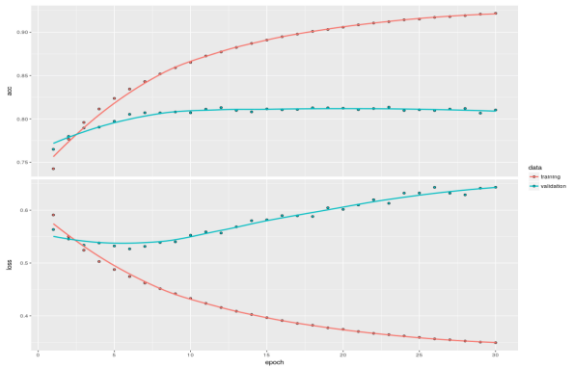


Fig92. Accuracy and loss curves for validation and training

Model 2 :

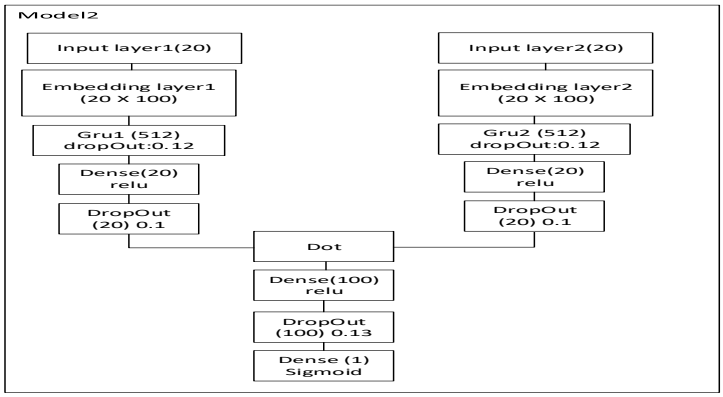


Fig93. Model2 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

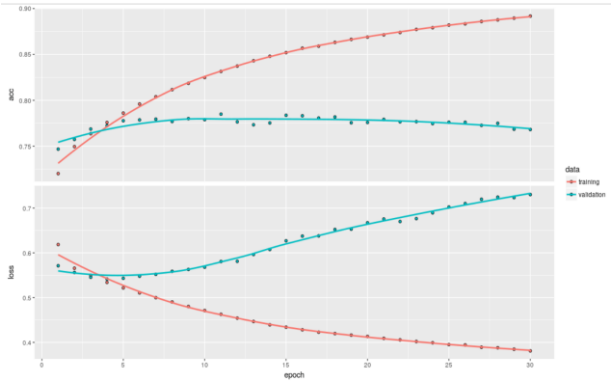


Fig94. Accuracy and loss curves for validation and training

Model 3 :

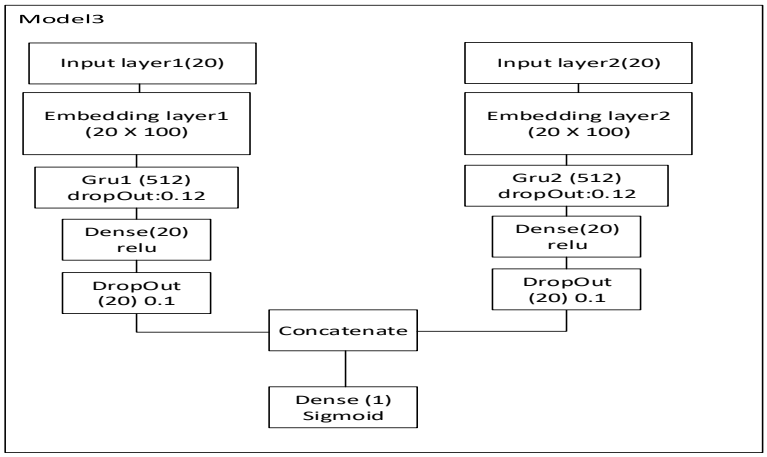


Fig95. Model3 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

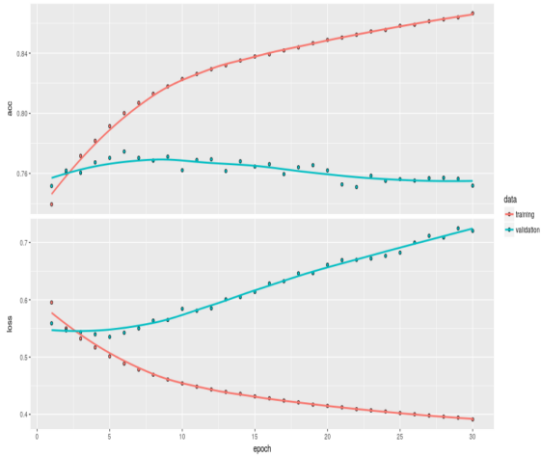


Fig96. Accuracy and loss curves for validation and training

Model 4 :

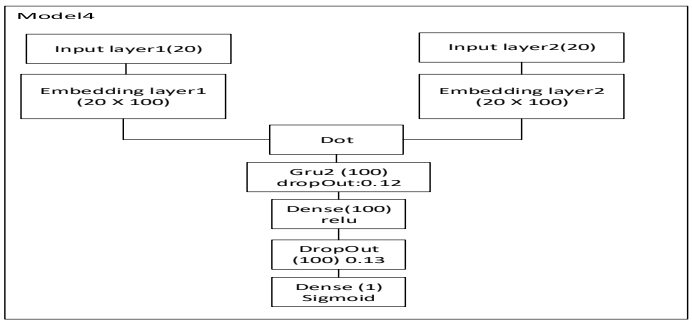


Fig97. Model4 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

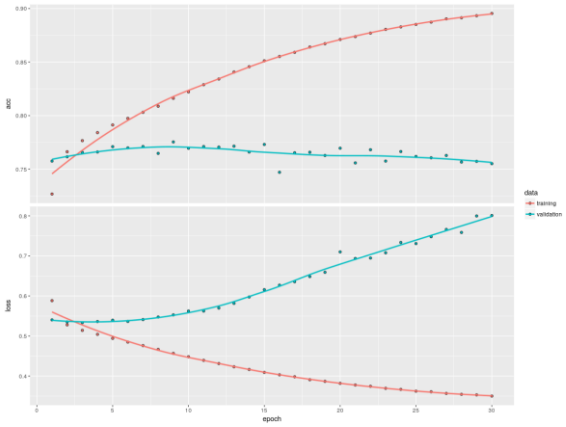


Fig98. Accuracy and loss curves for validation and training

Model 5 :

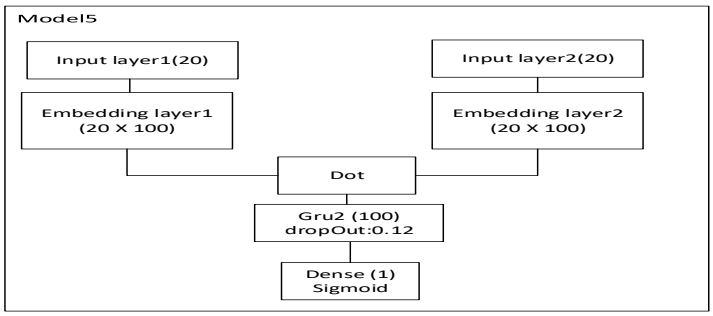


Fig99. Model5 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

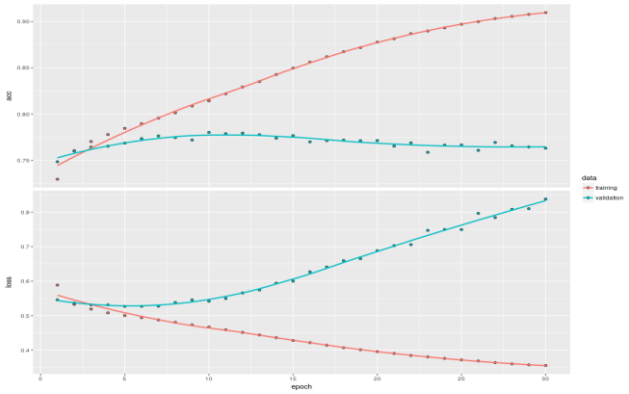


Fig100. Accuracy and loss curves for validation and training

Model 6 :

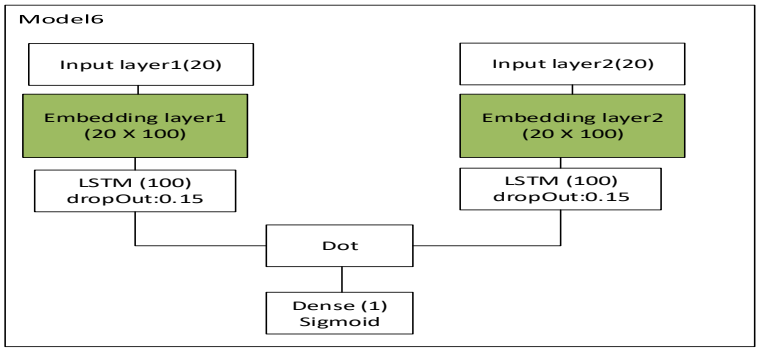


Fig101. Model6 for classifier

The embeding layers are loaded with a pretrained word vector, trained by Glove algorithm.

The speed of the training has accelerated, as we have less parameters to tune with the backpropagation algorithm.

Accuracy and loss grpahs without early stopping or dropout layer.

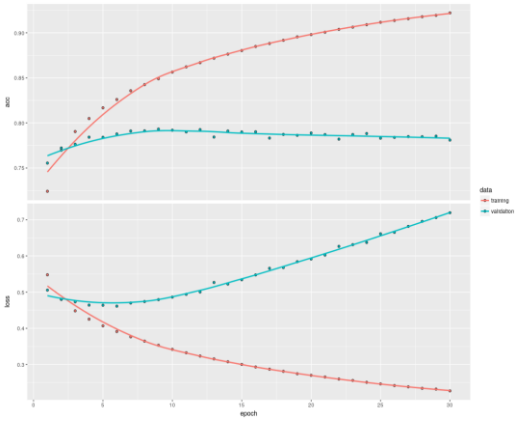


Fig102. Accuracy and loss curves for validation and training

Model7 :

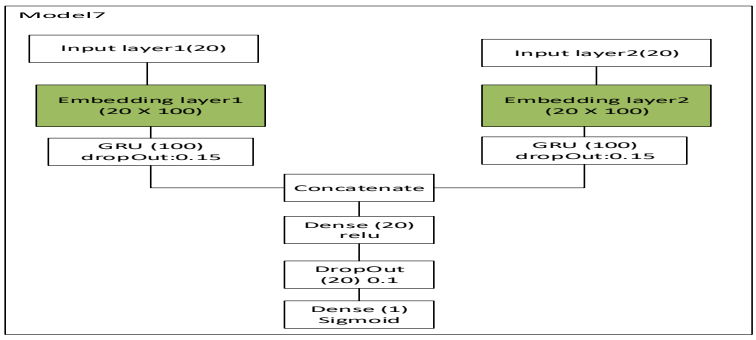


Fig103. Model7 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

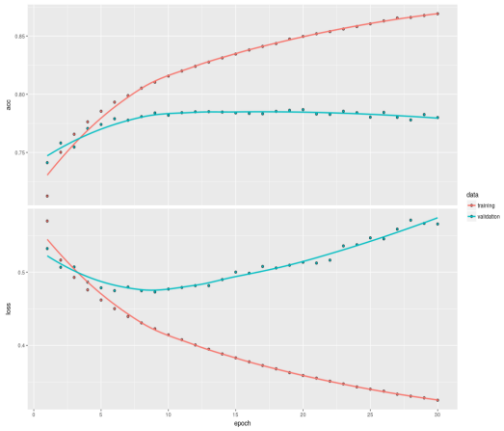


Fig104. Accuracy and loss curves for validation and training

Model 8 :

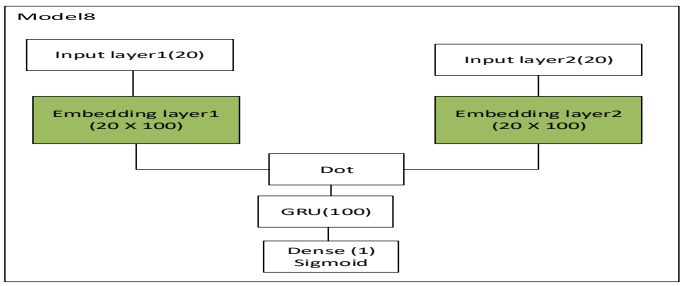


Fig105. Model8 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

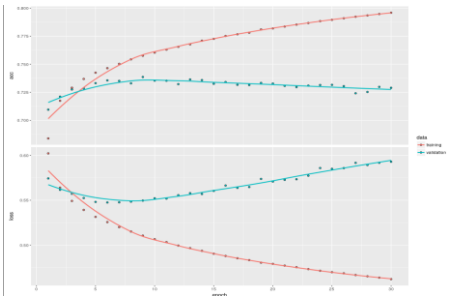


Fig106. Accuracy and loss curves for validation and training

Model9 :

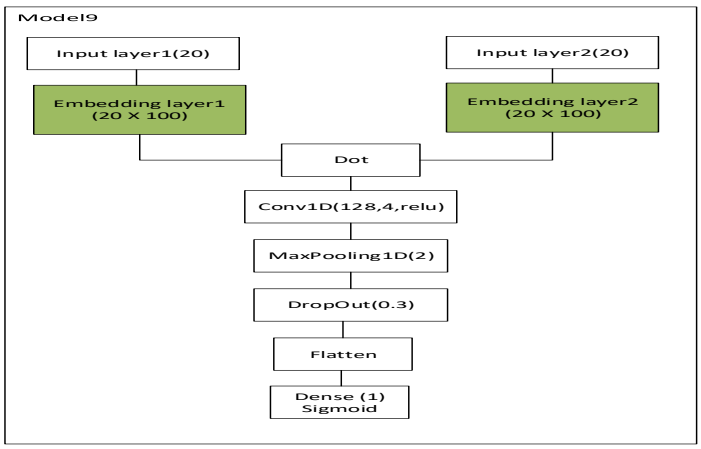


Fig107. Model9 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

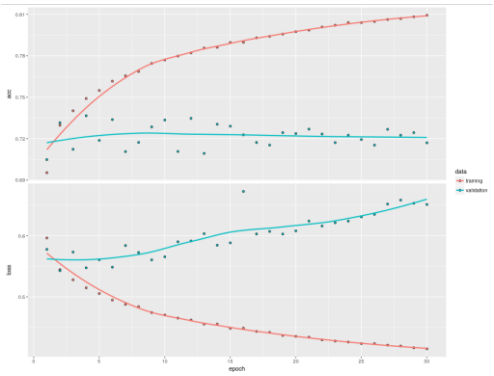


Fig108. Accuracy and loss curves for validation and training

Model10 :

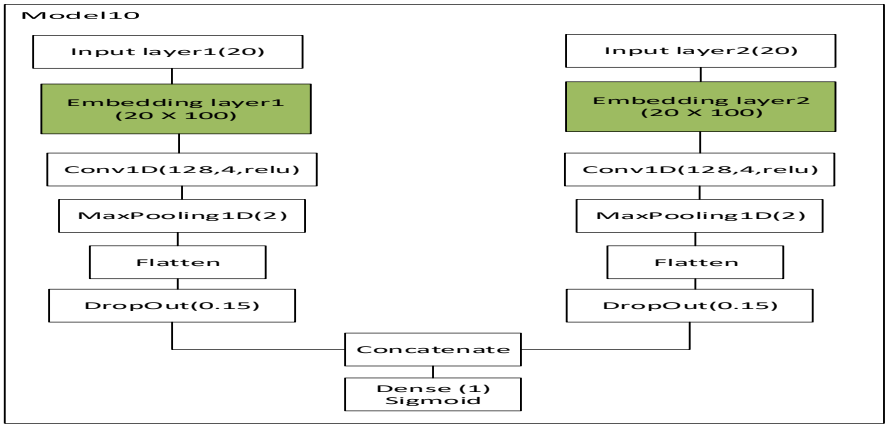


Fig109. Model10 for classifier

Accuracy and loss grpahs without early stopping or dropout layer.

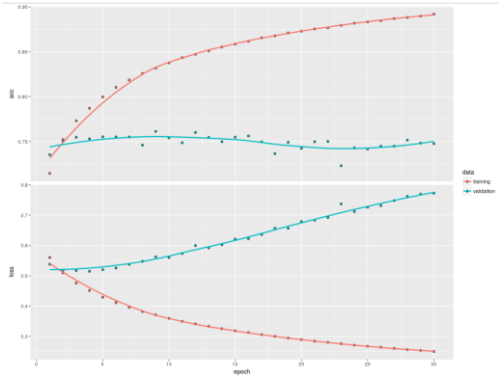


Fig110. Accuracy and loss curves for validation and training

Improving the overfitting:

We introduced also dropout layer to the different models, and we introduce the early stopping functions for the accuracy and for the learning rate, and we plot the accuracy loss curves in addition to the learning rate variation during the different epochs.

We plot all the models all together with multiplot function.

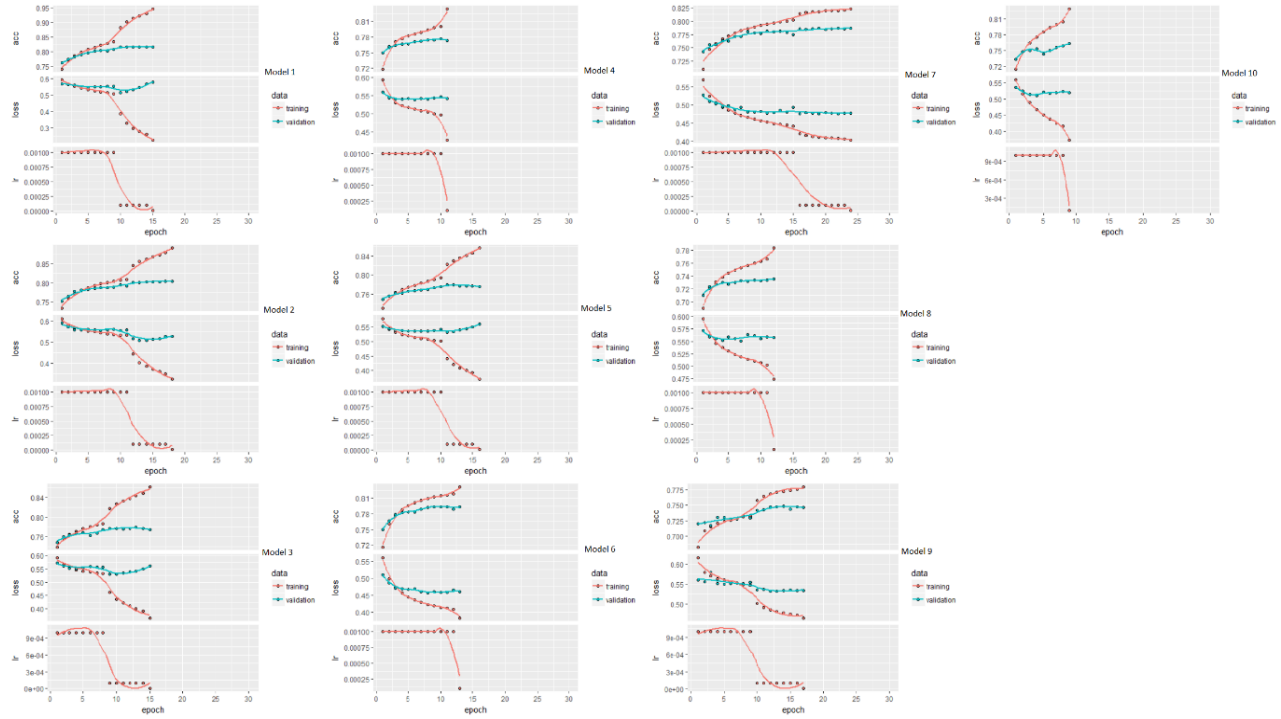


Fig111. Accuracy and loss curves for validation and training for all models after adding drop-out layers

Compare the models:

We plot the ROC curves for the 10 models, and we obtain the following plots, where every model curve is shown with a different color described in the legend.

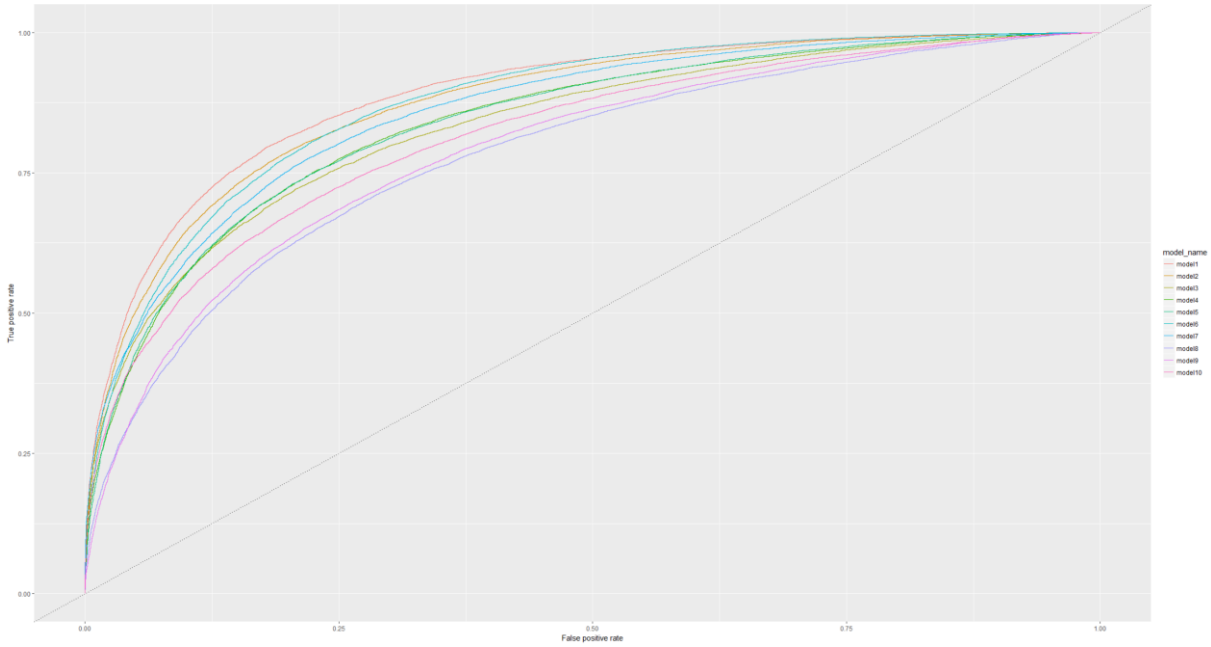


Fig112. ROC curves for all models

R Code :

The R code can be found in appendix section: 'Classifiers'.

It cotains the 10 models and their training function, in addition to the ROC curve comparaison.

Reduce word vector to 1 decimal value:

We project the 100 dimension of the words we calculated earlier to 1, by using T-SNE, so every word will have one decimal value between -1 and 1, and the similarity distance between words is still respecting the semantic meaning of the words:

	value	term
49538	0.05685434	quesadillas
45806	0.05685678	scallions
65400	0.05685812	cookbook
18961	0.05685814	enchiladas
51507	0.05685823	quarts
34045	0.05686423	caramelized
81448	0.05686467	pastry
68748	0.05686555	cookbooks
65386	0.05686824	thyme
77400	0.05687066	burrito
73616	0.05687124	brownie
72672	0.05687369	tortilla
65906	0.05687703	quart
78803	0.05687808	pudding
57381	0.05688339	oregano
89299	0.05688353	menu
58652	0.05688454	custard
83800	0.05688477	almonds

Fig113. Check word nighbours related to kitchen

Let's check the neighbor words of 'health'

	value	term
3550	0.05181737	ottoman
3551	0.05181229	sites
3552	0.05180761	counseling
3553	0.05180027	site
3554	0.05178984	alpine
3555	0.05178960	health
3556	0.05178922	pediatric
3557	0.05176950	care
3558	0.05175770	cardiology
3559	0.05175483	medical
3560	0.05175313	medicine

Fig114. Word nighbours of ‘health’

Word family :

Filter

	value	term
4299	0.04978820	prairie
4300	0.04978778	father
4301	0.04978473	abb
4302	0.04977872	dug
4303	0.04977862	ehime
4304	0.04977683	family
4305	0.04977597	valleys
4306	0.04977584	kursk
4307	0.04976710	wineries
4308	0.04976461	born
4309	0.04976337	ridges

Fig115. Word nighbours of ‘family’

In the table displayed above, I have sorted by the word values, and it looks like the words with similar values are also similar in meaning.

As every word has a value, and a sentence is a sequence of words, then we can plot the graph of these word sequences, to see if the graphs are also similar in case of similar questions.

Plot grpah of question with is_duplicate = 1 :

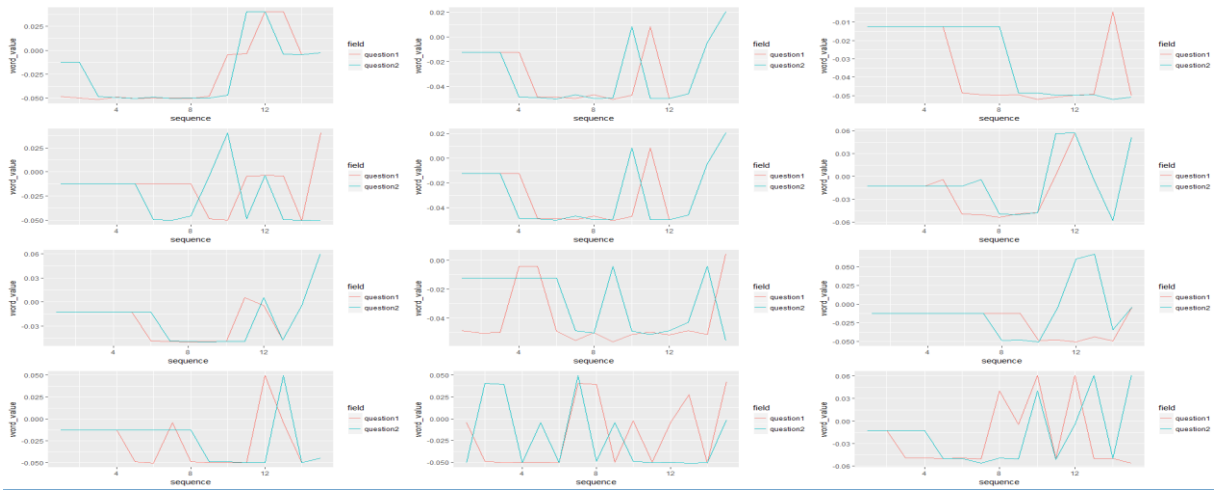


Fig116. Graph of pair question for class ‘duplicated’

Some question graphs with is_duplcate = 0

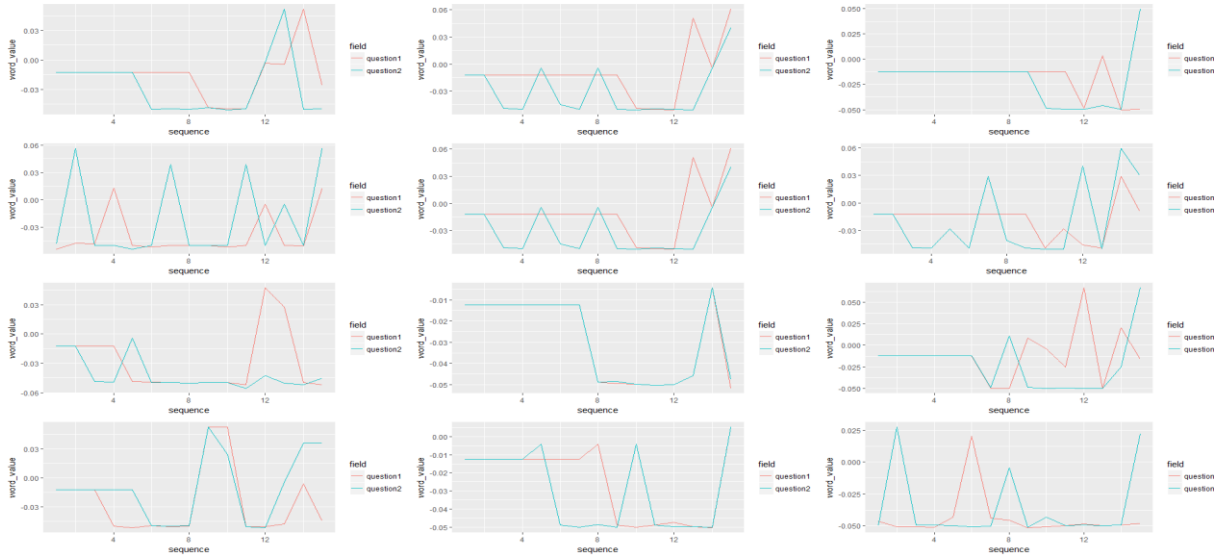


Fig117. Graph of pair question for class ‘non-duplicated’

We have tried the following deep learning models:

Model1:

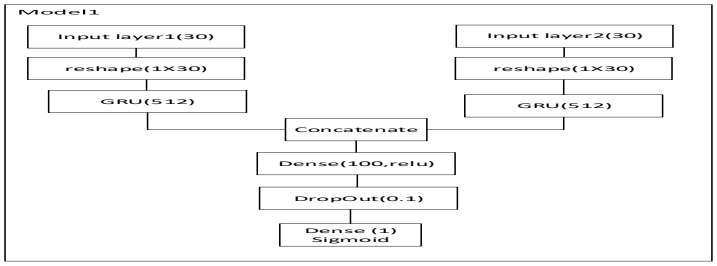


Fig118. Model1 with 1d WordVec classifier

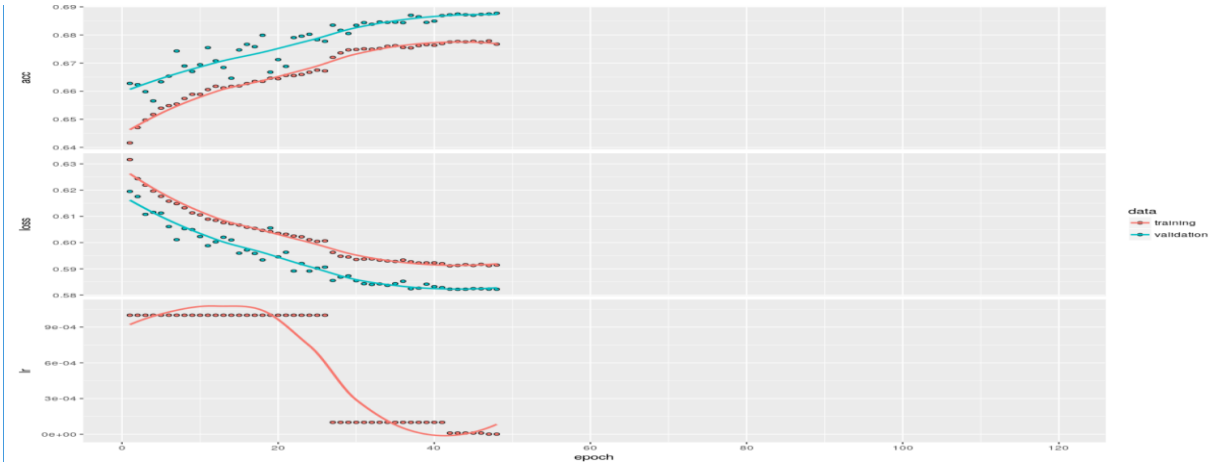


Fig119. Accuracy, Loss, Learning rate curves for model1 with training and validation data

Model 2:

We have just replaced the concatenation layer with dot layer

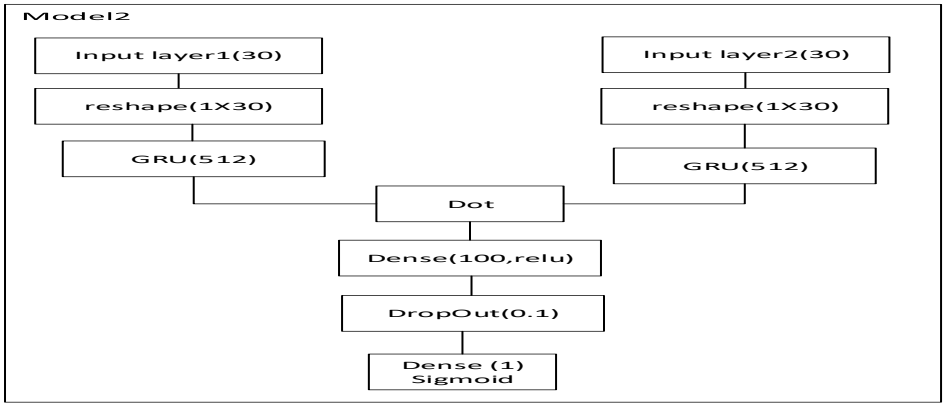


Fig120. Model2 with 1d WordVec classifier

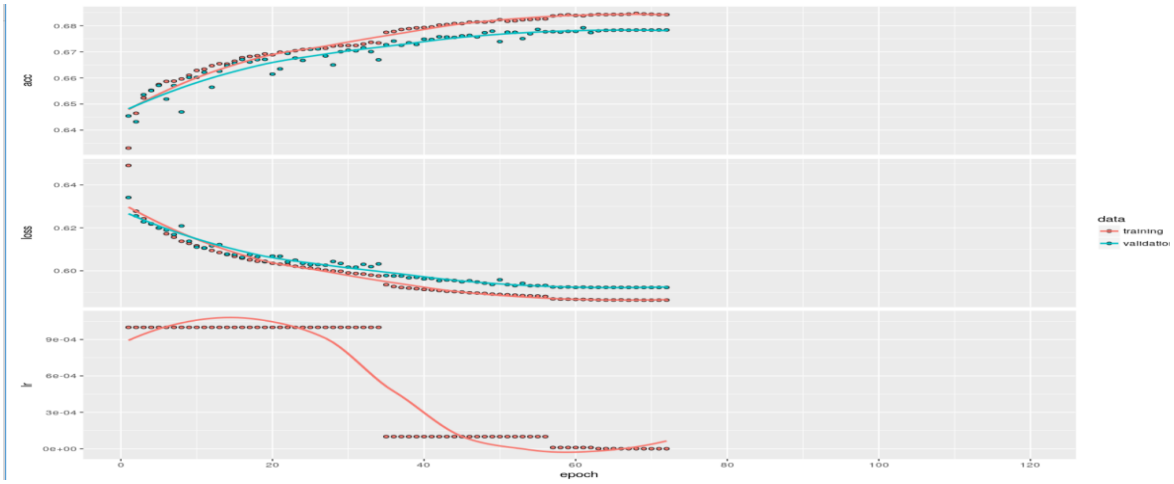


Fig121. Accuracy, Loss, Learning rate curves for model1 with training and validation data

In validation and training graphs, the accuracy and loss are correlated which is good, because all what the model learn is applied correctly to the validation data, which mean that there is no learning loss, as all what is learned can be applied.

R Code:

The R Code can be found in the following section 'Classifier with 1D word vectors'

Conclusion:

We have seen that keras library and deep learning in general offers a tremndous of possibility, given the number implemented layers, it is very easy combine them and play with different model architectures, to find the best one for the problem.

It is adviced to start always with the simple models, as they might solve the problem, and it is faster in cacluclation time.

We have tried to preload the embedding layer, and this helped in gaining the execution time, but it didn't give a better accuracy.

Most of the model suffers from overfitting, that's why we introduced:

- Early stopping when the validation loss increasing
- Introduce noise to the textes

- Adding noise as drop out layers, and early stopping, the models learn better as it reduced the gaps between training and validation accuracy and loss.

After comparing the models with ROC curves, Model 1 looks the best.

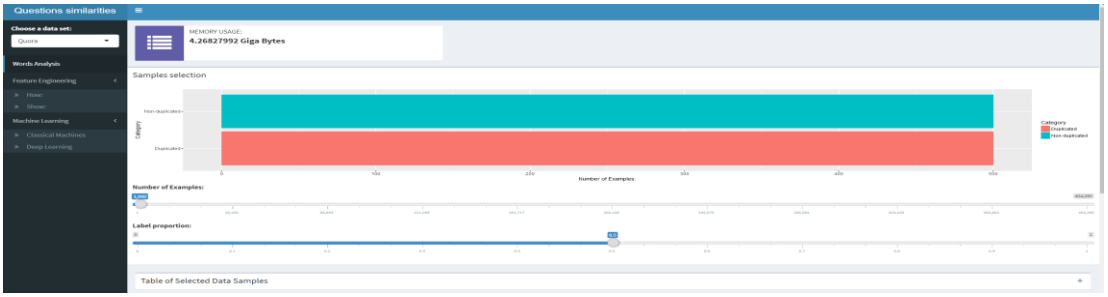
Shiny Dashboard:

The different R code functionalitions are integrated to shinay dashboard application, this will make our work:

- As stand-alone application
- Shiny has very nice visualization and smart events machanizims like zooming in graphics.
- Sihny allow to have some parameters in UI like sliders, to set dynamically the different parametes of machine learning
- It can be a good summary of the work that we have done
- Easy to demo
- Easy to deploy in the cloud

Navigation panel:

The structure of our shiny will have the following tree displayed in the navigation panel:



Data selection:

By default, the select data are 1000 pair of questions, they are taken randomly from the original data set, we take only 1000 to avoid load problems when the dashboard is launched the first time.

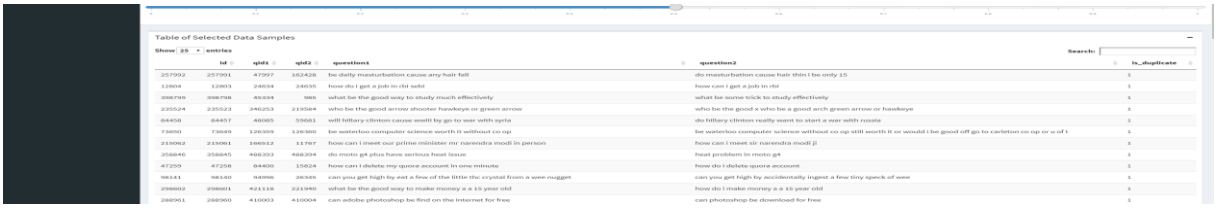
We have slider to change the number of examples, in addition the slider bellow which control ratio of the two categories, by default it is 0.5, which mean 500 duplicated and 500 non-duplicated.

The selected data are then used in the different parts of the application, this allow exprementing first with smaller amount of data to see the effect.

The selection area is displayed in all views and it contains the following parts:

- We show on the top info box, the memory usage at run time, there is a timer flashing evry second, and read the used memory, this will help future improvements of the application until having a good memery cleaning from unused data, to avoid accumulation of data in memory.
- After we display two horizonal bars showing the number of data set exampls for two categories of ‘duplicated’ and ‘Non-duplicated’.

The table of selected Data samples is expandbale, and show the select data by the sliders:

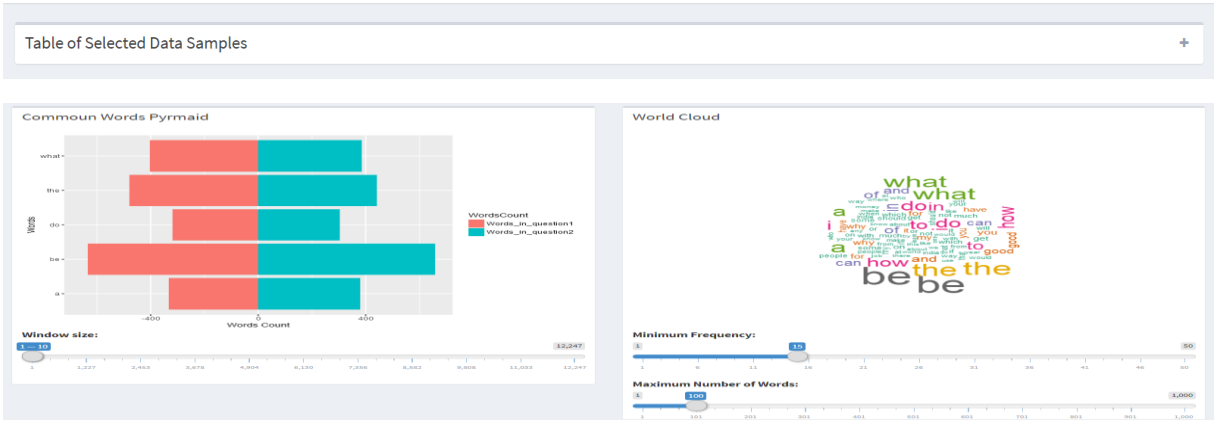


Words analysis:

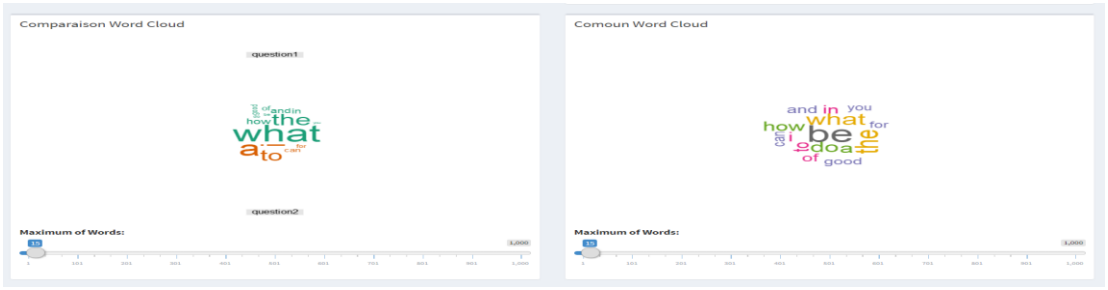
The first node is ‘Words analysis’, and it is displayed by default as the first detail view,

We show the word pyramide in the text corpus of question1 and question2 columns, and we show the word cloud,

With the sliders you can select the number of words to be displayed.



We show ‘comparaison word cloud’ and cmmoun word cloud,



Feature Engeneering:

How:

In the sub-node ‘Feature engeneering -> How’:

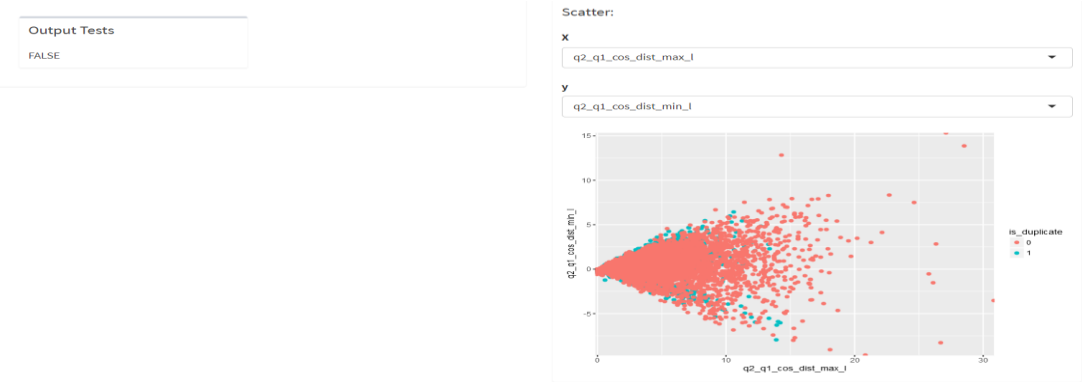
We show how the different features are calculated.

Show:

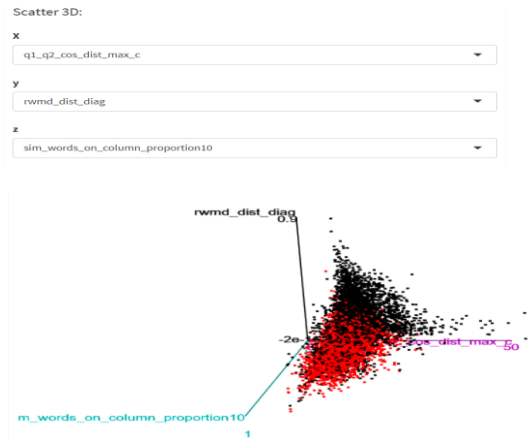
In the sub-node ‘Feature engeneering -> Show’, we show:

- The correlation plot between the features, the user can add as he likes the features to check the correlation
- The densities of different features, the user can zoom in graphic to see more details, and he can add as many feature he likes

With the scatter plot, you can choose x and y axis from the features and plot the cloud points of the two categories, the graph has also the possibly to zoom in



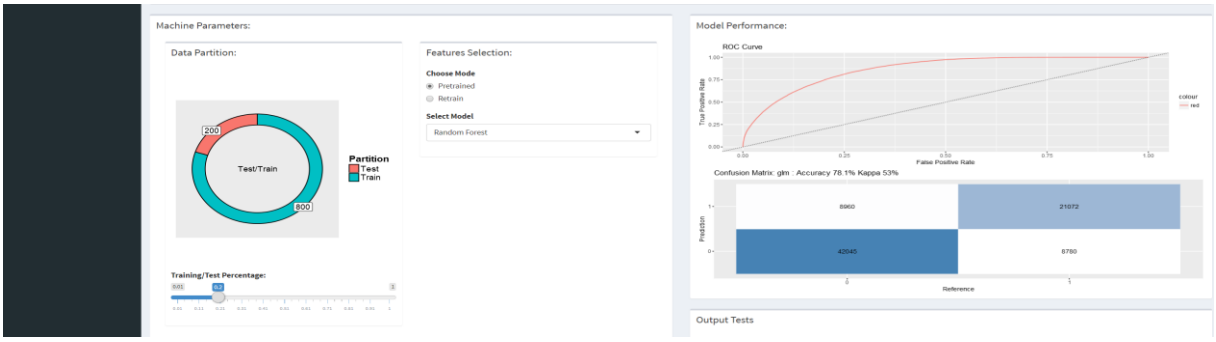
You can also choose the three axis features, to see the scatter plot in 3D, you zoom in and change the angle to see cloud from different angle.



Classical Machine Learning:

Pretrained mode:

In this mode you can select any model with the drop-down list, and we get from the stored RDS data format, the data concerning the model, and we display the ROC curve in addition to confusion matrix.

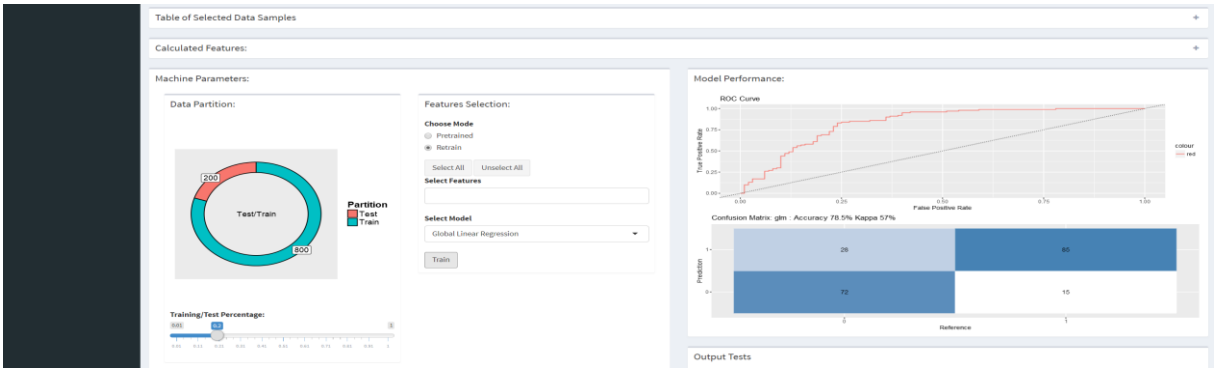


Retrain Mode:

In the reatrain mode, we use the selected model, and the user can select with slider bar the ration between test and training data, by default it is 20% for test data.

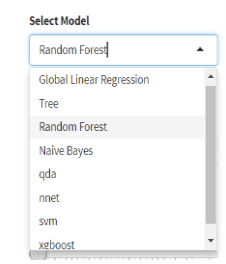
The user can choose the features to which the machine learning is applied, by default it is taking all if nothing is selected.

After the training and validation on the selected data, the ROC curve and confusion matrix are ploted.

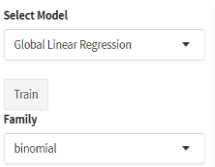


List of available model:

This is the list of all models, we have in addition ‘SVM’ and ‘XGBOOST’ machines that we didn’t look in the machine learning section.

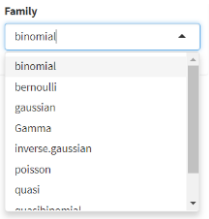


Prameters of GLM model:

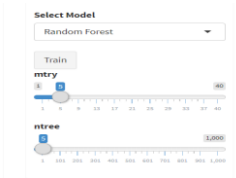


Pair of questions similarities 04-09-2018

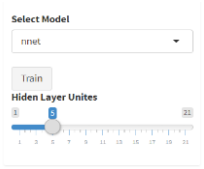
Famlilies of GLM:



Parameters of Random Forest:

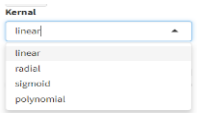


Parameters of nnet model:

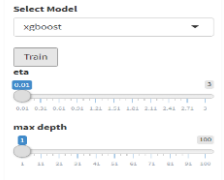


Parameters of SVM model:

Kernals of SVM model:

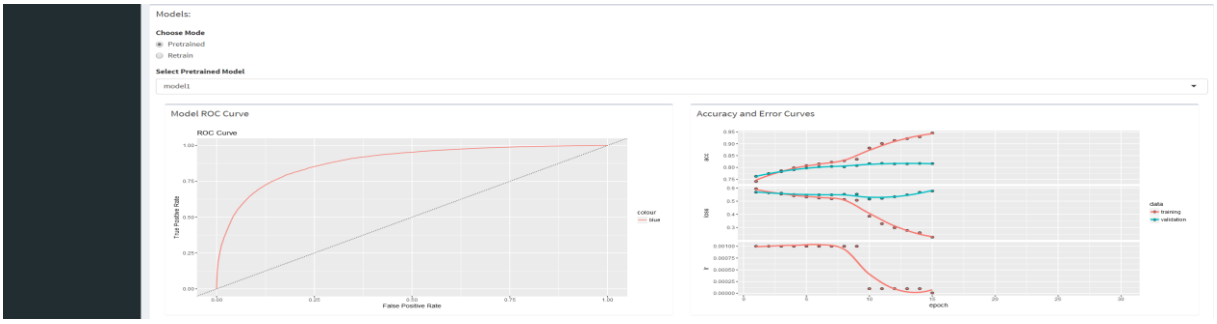


Xgboost parameters:

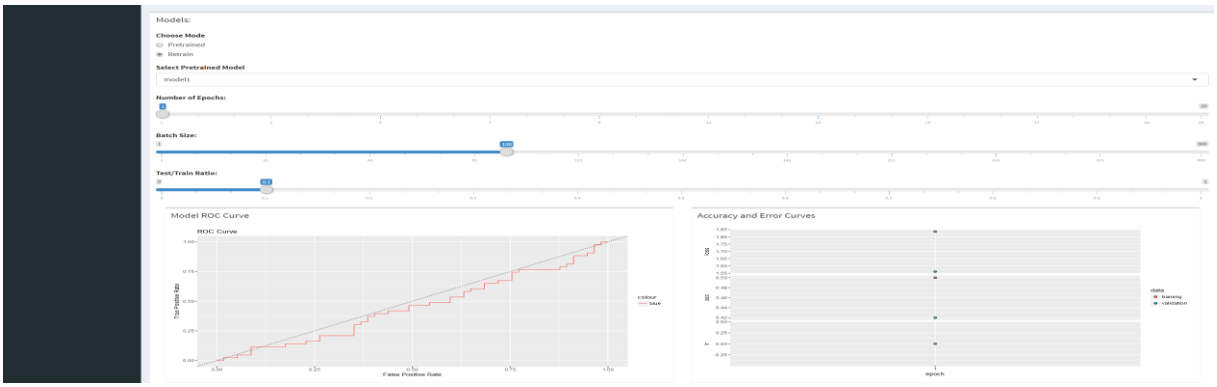


Deep Learning:

Pretrained mode:



Retrain mode:



Topology of deep learning model:

Model Topology

Model

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 20)	0	
input_2 (InputLayer)	(None, 20)	0	
embedding_1 (Embedding)	(None, 20, 100)	7602900	input_1[0][0]
embedding_2 (Embedding)	(None, 20, 100)	7602900	input_2[0][0]
gru_1 (GRU)	(None, 512)	941568	embedding_1[0][0]
gru_2 (GRU)	(None, 512)	941568	embedding_2[0][0]
dot_1 (Dot)	(None, 1)	0	gru_1[0][0] gru_2[0][0]
dense_1 (Dense)	(None, 100)	200	dot_1[0][0]
dropout_1 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_3 (Dense)	(None, 1)	101	dropout_1[0][0]

Total params: 17,089,237 Trainable params: 17,089,237 Non-trainable params: 0

Conclusion:

In this project we started with the first element of sentence which is word, we presented some visual graphs like word clouds, word pyramid that gives some statistics about the used words.

To produce a data science project, we need numerical values, and we have calculated some features like:

- Number of words in sentences
- Number of different words between two sentences
- Grammatical entities
- Sequence of words and sequence of grammatical entities

To use some more advanced features which are based on cosine distances between vectors, we vectorized all the words of the vocabulary, we used GLOVE (Global Vector) algorithm which produce dense vectors which reflect the semantics between the vocabulary words.

And then we calculated:

- the similarity matrixs between the pair of question, and from the matrix we calculated some statistical features like: sum of mins, sum of means, and sum of maxs for the lines and the cloumns of the matrix

We transformed our data set of pair of questions to the matrixes of DTM, TF-IDF and LSA, and then we calculated the following distances as additional features:

- Jaccard, euclidienne, cosine and Relaxed word mover

We showed some graphics based on the calculated features, like the scatter in 2D and 3D to see the visual separability between the the two categories of 'Duplcated' and 'Non-duplicated' which we want to predict.

We used the calculated features to train some learning machines, we applied the cross validation with K-Fold and the random forest get the best accuracy of 78%.

The feature engeneering is cosuming lotof calculation ressources, as we passed around 2 weeks just in the calculations, despite this, it helped us to get more knowledge on the NLP field, in addition, it produced us numerical values which are a must for machine learning and to any data science project, and we wanted to check other alternatives than the deep learning.

In the deep learning part, we don't have to deal with the features as the features will be contained in the wights of the layers as they auto-adapt themselves with the backpropgation algorithm, until the learning objective is reached.

We started with autoencoders where we wanted to encode the sentences to lower dimension dense vectors proruced by the last layer of the encoder, we have tried different topologies of models where we mixed the different keras layer types.

We faced a problem of inversing or decoding the embeding layer, that's why we used a pretrained embeding layer where we downloaded the word vectors from the intenet, and we took than the embeding layer outside the autoencoder chain.

We faced memory problems, as the embeding layer output is giving every word in the word sequences its vector, this produces a matrix for every sentence, which cost a lotof memory when it is calculated for the whole data set, so we split the data to smaller packages, and we trained the auto-encoder in many iteration, we start every iteration with the wights of the last one.

We didn't use the obtained encoder to test it with other machine learning, to check if the obtained low dimention vectors represent the sentences, their word sequences and the semantic meaning.

We continued with deep learning classifiers, and we impelemted 10 different topologies, where we varied the layer types, we used:

- Trainable and pretrained embedding layers for vectorizing the words
- GRU and LSTM layers to model word sequence of sentences
- Dot and concatenate layer to join the two branches where the two sentences are comming

- Normal dense layers with different activation functions, and different number of whights to glue the different parts
- Drop-out layer for the overfiting and to avoid local minimas as it has randomization effects

To combat the overfitting, we used some early stopping technics.

We obtained the best accuracy of 81 % with model number one, this model uses a trainable embeding, GRU, dot, dense, drop-out layers.

Without the help of NVIDIA machine, it will be difficult experement with the deep learning, as it is very hungry for CPU and memory.

For the implementation in R, we started with normal R functions for the different parts, and we then tried to integrate them in a shiny dashboard, which has a very nice visualization features, and we can have global control and vision of all the implemted functions.

For the future of this project, we need to develop the following points:

- We need to know which features from the extracted features contribute mostely on the increasing of accuracy of the learning models, it is impossible to use ‘LIME’ method given the amount of data we have , we need to create a new data set with all the calculated features, and where the label will be: ‘good prediction’ and ‘bad prediction’ which will come from the model if it predict correctely or not, and we train new learning models with this data, hopfully by doing this we will get some ideas in which area of feature engeering we can search more features that can additionaly improve the accuracy.
- Experiment with reinforcement learning to replace the labels by a generic reward function which will be based on the cosine distance
- Instead of using pair of texts as data set, we need to experiment with other data sets where we have mapping of short texts with code or script statements like : script of automatic test, or line of language code, the objective will be, which of code scripts refelect mostely the user request which is expressed in natural language, this can help a lot in the autaumation tasks ,the best starting point for this challenge will the autoencoders, as every autoencoder will focus on its data, one for the short texts, and one for the scripts, after having lower dimention representation of both data, we can map them easily with deep learning layers,let’s find first a good data set for this challenge.

Appendix:

Read References:

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger, FromWord Embeddings To Document Distances, Washington University in St. Louis, 1 Brookings Dr., St. Louis, MO 63130

http://mkusner.github.io/publications/WMD.pdf

Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors forWord Representation

https://nlp.stanford.edu/pubs/glove.pdf

Sunipa Dev, Safia Hassan, Jeff M. Phillips, Absolute Orientation for Word Embedding Alignment
https://arxiv.org/pdf/1806.01330.pdf

Francois Chollet, Deep learning with python

Lectures collections, natural language processing with deep learning (winter 2017) Harvard,
https://www.youtube.com/playlist?list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6

Machine learning course by andrew ng, coursura, youtube
https://www.youtube.com/watch?v=Hxm4ERsDv5U&list=PLBAGcD3siRDghsFtvJH9HjWSq9DHk1fTJ

Environment of executing R code:

Most of the code was executing in my laptop which is i5-6300U with 16Gb Ram.

The parts which are demanding more calculation resources like deep learning are done in NVIDIA machine, and I have installed the two R rockers:

- ‘alperyilmaz/tidyverse-keras’
- ‘rocker/verse’

The first one contains keras library, where I have technical problems to install other libraries.

And the seconde rocker, is for machine learning library like ‘caret’ and other packages, it helped sometimes to accelerate the calculations and recheck results obtained with my laptop.

R Code:

Clean.data

```
# Downloading Data ----------------------------------------------------------
quora_data <- get_file(
  "quora_duplicate_questions.tsv",
  "http://qim.ec.quoracdn.net/quora_duplicate_questions.tsv" )


clean_data_set <- function(Df)
```

```
{
   Df <- Df %>% mutate( question1 = (question1 %>% replace_contraction() %>% replace_symbol()
%>% tolower()),
                        question2 = (question2 %>% replace_contraction() %>% replace_symbol()
%>% tolower()))

   Df <- Df %>% mutate( question1 = (question1 %>% str_replace_all( "`","") %>%
str_replace_all( "’","") %>%  str_replace_all( "…","")),
                        question2 = (question2 %>% str_replace_all( "`","") %>%
str_replace_all( "’","") %>%  str_replace_all( "…","")))


   Df <- Df %>% mutate( question1 = ( question1 %>% rm_bracket(pattern = "all",trim=T,clean =
T ) ),
                        question2 = ( question2 %>% rm_bracket(pattern = "all",trim=T,clean =
T ) ) )

   Df


}

clean appostrophes <- function(Df)
{
  patterns     <- c("it's"   ,"he's"   ,"she's"  ,"i'm" ,"'re "    )
  replacements <- c("it is"  ,"he is"  ,"she is" ,"i am"," are "  )


  patterns    <- cbind(patterns     %>% t(),"won't"   ,"ain't"  ,"n't " )
  replacements<- cbind(replacements %>% t(),"will not","are not"," not ")


  patterns    <- cbind(patterns    ,"'ve "    , "'d "     , "'ll "     )
  replacements<- cbind(replacements," have "  , " would " , " will "   )


  pattern_replace <- data.frame( patterns %>% t(), replacements %>% t())

  apply(X = pattern_replace,MARGIN = 1,FUN = function(x){

    Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
x[[1]],replacement = x[[2]]) ),
                          question2 = ( question2 %>%  str_replace_all(pattern =
x[[1]],replacement = x[[2]]) ) )

    })

  Df
}



clean_non_ascii <- function(Df,Missing_words)
{

  patterns     <- c( "bigdata" , "onmyoji"  ,"heigou"  , "tarteist"  ,"delhi" ,"company" ,
"flatland" ,
                     "？","'" ,"`" ,"〞" ,"？" ,"–" ,"—" ,
                     stri_unescape_unicode(paste0("\U2206")) ,
stri_unescape_unicode(paste0("\U222B")),
                     "flatland"  , "onmyoji"   , "ſ"         ,"iṡtanbul", "iṽe")
  replacements <- c("bigdata ", "onmyoji ","heigou " , "tarteist " ,"delhi ","company ",
"flatland ",
                     " " ," " ," "," "  ,""   ," " ," " ,
                     "d"                                  ,"integral"
,
                     "flatland " , "onmyoji "   , "integral ","istanbul", "ive")

  pattern_replace <- data.frame( patterns , replacements,stringsAsFactors = F )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  rm_bracket(pattern = "round")  ),
                        question2 = ( question2 %>%  rm_bracket(pattern = "round") ) )
```

```
  Df <- Df %>% mutate(  question1 = ( question1 %>%  rm_stopwords(c(".","?"),separate =
F,strip = T)  ),
                        question2 = ( question2 %>%  rm_stopwords(c(".","?") ,separate =
F,strip = T) ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern = "'",replacement
= " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "'",replacement
= " ") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern = "`",replacement
= " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "`",replacement
= " ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern = "´",replacement
= " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "´",replacement
= " ") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str replace all(pattern = "—",replacement
= " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "—",replacement
= " ") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern = "-",replacement
= " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "-",replacement
= " ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern = "？
",replacement = " ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern = "？
",replacement = " ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"flatland",replacement = "flatland ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"flatland",replacement = "flatland ") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"company",replacement = "company ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"company",replacement = "company ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"delhi",replacement = "delhi ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"delhi",replacement = "delhi ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"tarteist",replacement = "tarteist ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"tarteist",replacement = "tarteist ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"heigou",replacement = " heigou ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"heigou",replacement = " heigou ") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"onmyoji",replacement = "onmyoji ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"onmyoji",replacement = "onmyoji ") ) )

  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
"bigdata",replacement = "bigdata ") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
"bigdata",replacement = "bigdata ") ) )
```

```
  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
stri_unescape_unicode(paste0("\U2206")),replacement = "d") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
stri_unescape_unicode(paste0("\U2206")),replacement = "d") ) )


  Df <- Df %>% mutate(  question1 = ( question1 %>%  str_replace_all(pattern =
stri_unescape_unicode(paste0("\U222B")),replacement = "integral") ),
                        question2 = ( question2 %>%  str_replace_all(pattern =
stri_unescape_unicode(paste0("\U222B")),replacement = "integral") ) )

  Df
}

 split_non_ascii_words <- function(OneRaw)
{
  #v = tibble(question1 = OneRaw["question1"],question2 = OneRaw["question2"] )

  list(  id              = OneRaw["id"]    %>% as.numeric() ,
         qid1            = OneRaw["qid1"]  %>% as.numeric() ,
         qid2            = OneRaw["qid2"]  %>% as.numeric() ,
         question1       = OneRaw["question1"] %>% split non ascii(),
         question2       = OneRaw["question2"] %>% split_non_ascii(),
         is_duplicate    = OneRaw["is_duplicate"] %>% as.character()  )


}

question_has_non_asccii <- function(question)
{
  non_ascii_q = question %>% ex_non_ascii() %>% unlist()

  logic_nonacii <- non_ascii_q %>% is_empty()
  if (!logic_nonacii )  logic_nonacii <- ! ( non_ascii_q %>% is.na() %>% mean() >0 )
  logic_nonacii
}

split_non_ascii <- function(question)
{
  ifelse( question %>% question_has_non_asccii(),split_q(question) ,question)

}
split_q <- function(question)
{
  v = tibble(question = question )
  Q <-   v %>% select(question)  %>% unnest_tokens(word,question)
  Q <- Q %>% as.vector()  %>% t() %>% paste(collapse = ' ')
  Q
}

split_non_ascii_words_df <- function(Df)
{
  Df <- Df %>% apply(MARGIN = 1,FUN = split_non_ascii_words) %>% rbindlist( fill = TRUE)
  Df
}

df <- read_tsv(quora_data)
df <- clean_data_set(df)
df <- clean_appostrophes(df)
df <- split_non_ascii_words_df(df)
df <- clean_non_ascii(df, Missing_Q)
save(df,file="df.RData")
```

Pyramid_plot_commoun_words

```
library(plotrix)
load("df.RData")

vocab1 <- df$question1 %>% space_tokenizer( ) %>% itoken( progressbar = FALSE ) %>%
create_vocabulary( )
vocab2 <- df$question2 %>% space_tokenizer( ) %>% itoken( progressbar = FALSE)  %>%
create_vocabulary( )

colnames(vocab2) = c("term","term_count2","doc_count2" )
```

```
colnames(vocab1) = c("term","term_count1","doc_count1" )

comoun_vocab_1_2 <- vocab1 %>% inner_join(vocab2 ) %>% arrange(term_count1 %>% desc() ,
term_count2 %>% desc())

set.seed(1817328)
val_sample <- sample.int(nrow(comoun_vocab_1_2), size = 20)

comoun_vocab_1_2_subset <- comoun_vocab_1_2[500:530,]

pyramid.plot(comoun_vocab_1_2_subset$term_count1,comoun_vocab_1_2_subset$term_count2, labels =
comoun_vocab_1_2_subset$term,
             gap = 14, top.labels = c("Question1","Words","Question2") ,main = "Words in
common", laxlab = NULL, raxlab = NULL, unit = NULL)
```

Word.cloud

```
row.names(comoun_vocab_1_2) = comoun_vocab_1_2$term
comoun_vocab_1_2 <- comoun_vocab_1_2 %>% select(term_count1,term_count2)
colnames(comoun_vocab_1_2)  = c("question1", "question2")
library(RColorBrewer)
library(wordcloud)
display.brewer.all()
pal <- brewer.pal(8,"Purples")
pal <- pal[-(1:4)]

comoun_vocab_1_2 %>% commonality.cloud(max.words = 750, random.order = FALSE, colors = pal)
comoun_vocab_1_2 %>% comparison.cloud( max.words = 750, random.order = FALSE, title.size =
1.0, colors = brewer.pal( ncol(comoun_vocab_1_2) , "Dark2"))
```

Features of word numbers in pair questions

```
# Feat.Calc.Part1.  Features of word numbers in pair questions
# Feature 1: number of words --
feature_number_of_words <- function(Question)
{
  v = tibble(Question)
  token_q1 <- v  %>% select(Question)  %>% unnest_tokens(word,Question) %>% t()
  length(token_q1)
}
# Feature 2: delta of words -------
feature_delta_of_words  <- function(OneRaw)
{
  question1 <- OneRaw["question1"] %>% as.character()
  question2 <- OneRaw["question2"]  %>% as.character()
  v = tibble(question1,question2)
   token_q1 <- v  %>% select(question1)  %>% unnest_tokens(word,question1) %>% t()
   token_q2 <- v  %>% select(question2)  %>% unnest_tokens(word,question2) %>% t()
   v1 <- setdiff(token_q1, token_q2 ) %>% length()
   v2 <- setdiff(token_q2, token_q1 ) %>% length()
   list(delta_q1_q2 = v1, delta_q2_q1 = v2)
}
# Feature 3: no of stop words  -------
feature_no_stop_words <- function(OneRaw,soptwords_en)
{
  question1 <- OneRaw["question1"]%>% as.character()
  question2 <- OneRaw["question2"] %>% as.character()
   v = tibble(question1,question2)
  token_q1 <- v  %>% select(question1)  %>% unnest_tokens(word,question1) %>% t()
  token_q2 <- v  %>% select(question2)  %>% unnest_tokens(word,question2) %>% t()
  v1 <- intersect(token_q1, soptwords_en ) %>% length()
  v2 <- intersect(token_q2, soptwords_en ) %>% length()
  list(question1_no_stop_words = v1, question2_no_stop_words = v2)
  }
# Call the functions to calculate the attribute and save to 'RData' files the results
question no stop words = df %>% apply(MARGIN = 1,feature no stop words,soptwords en)%>%
rbindlist(fill =  F)
save(question_no_stop_words,file='question_no_stop_words.RData')
question_delta_words = df %>%   apply( MARGIN = 1,feature_delta_of_words) %>% rbindlist(fill =
F)
save(question_delta_words,file='question_delta_words.RData')
```

```
question1_number_of_words = df$question1 %>% as.data.frame() %>%  apply(MARGIN = 1 , FUN =
feature_number_of_words)
save(question1_number_of_words,file='question1_number_of_words.RData')
question1_number_of_words =question1_number_of_words   %>% as.data.frame()
colnames(question1_number_of_words) = c('question1_no_words')
save(question1_number_of_words,file='question1_number_of_words.RData')
question2_number_of_words = df$question2 %>% as.data.frame() %>%  apply(MARGIN = 1 , FUN =
feature_number_of_words)
question2_number_of_words = question2_number_of_words %>% as.data.frame()
colnames(question2_number_of_words) = c('question2_no_words')
save(question2_number_of_words, file='question2_number_of_words.RData')
```

POS tags statistics

```
# Feat.Gramatical.Entities.Statistics:
# 1 Initialization of POS tags
pos_token_annotator_model <- Maxent_POS_Tag_Annotator(language = "en",
                                                      probs = TRUE, model =
system.file("models", "en-pos-perceptron.bin",
                                                      package = "openNLPmodels.en"))
wordAnnotator     <- Maxent_Word_Token_Annotator(language = "en", probs = TRUE, model =NULL)
sentAnnotator     <- Maxent_Sent_Token_Annotator(language = "en", probs = TRUE, model =NULL)

pos_tag_annotator <- Maxent_POS_Tag_Annotator(language = "en", probs =TRUE, model =NULL)
pos_tag_annotator <- Maxent_POS_Tag_Annotator(language = "en", probs =TRUE, model =NULL)
s= " the of system is not ok"
chunkAnnotator <- Maxent_Chunk_Annotator(language = "en", probs =        FALSE, model = NULL)
annotate(s,chunkAnnotator,posTaggedSentence)
annotated_sentence <- annotate(s,sentAnnotator)
posTaggedSentence <- annotate(s, pos_tag_annotator, annotated_word)
annotated_word<- annotate(s,wordAnnotator,annotated_sentence)
# 2 POS tags parsing
```

POS tags sequences:

```
# Get the sequence POS Tags -----------
POS_tags_seq <- function(dataRaw)
{
  posTaggedSentence <-  as.String(dataRaw) %>% annotate( pos_tag_annotator, annotated_word)
  posTaggedWords <- posTaggedSentence %>% subset(type == "word")
  posTaggedWords
  tags <- posTaggedWords$features %>% sapply( `[[`, "POS")
   tags %>% paste(collapse = "-")
}
POS_tags <- function(dataRaw)
{
  posTaggedSentence <-  as.String(dataRaw) %>% annotate( pos_tag_annotator, annotated_word)
  posTaggedWords <- posTaggedSentence %>% subset(type == "word")
  posTaggedWords
  tags <- posTaggedWords$features %>% sapply( `[[`, "POS")
  table_tags <- table(tags) %>% as.data.frame() %>% t()
  colnames(table_tags) = table_tags[1,]
  row.names(table_tags) = NULL
  table_tags
}
# Create New structure with all fields of tags
Create_data_frame_for_tags <- function(Nrow)
{
  Vect_Tags = NLP::Penn_Treebank_POS_tags$entry[1:45]
  length(Vect_Tags)
  y <- data.frame(matrix(0,ncol = length(Vect_Tags), nrow = Nrow))
  colnames(y) = Vect_Tags
  y
  }
# Get the tags and fill one-line table of features
MAP_POS_tags_to_Data <- function(  Question_sentence )
{
  tag_line = Create_data_frame_for_tags(1)
  question_tag      <- Question_sentence %>% POS_tags()
  question_columns <- question_tag %>% colnames()
  tag_colunms <- tag_line %>% colnames( )
  tag_diff <- setdiff(question_columns,tag_colunms)
  tag_diff
  question_columns <- setdiff(question_columns,tag_diff)
  question_columns
```

```
  tag_line[1,question_columns] <- question_tag[2,question_columns] %>% as.character()
  return(tag_line)
}
# 3 extract POS tags statistics
# grammatical statistics for question1 column
question1_gramatical_entities_no <- df$question1 %>% lapply(FUN = MAP_POS_tags_to_Data) %>%
rbindlist( fill = TRUE)
colnames(question1_gramatical_entities_no) =
paste('question1',colnames(question1_gramatical_entities_no),sep = "_" )
save(question1_gramatical_entities_no,file='question1_gramatical_entities_no.RData')
# grammatical statistics for question2 column
question2_gramatical_entities_no <- df$question2 %>% lapply(FUN = MAP_POS_tags_to_Data) %>%
rbindlist( fill = TRUE)
colnames(question2_gramatical_entities_no) =
paste('question2',colnames(question2_gramatical_entities_no),sep = "_" )
save(question2_gramatical_entities_no,file='question2_gramatical_entities_no.RData')
# merge both and save
question_gramatical_entities_no =
cbind(question1_gramatical_entities_no,question2_gramatical_entities_no)
question_gramatical_entities_no <- question_gramatical_entities_no %>% mutate_all(as.numeric)
save(question_gramatical_entities_no,file='question_gramatical_entities_no.RData')

# 4 : Gramatical entities cosine distance ---
load('question_gramatical_entities_no.RData')
question_gramatical_entities_no %>% head() %>% View()
cosin_distance_pos_tags <- function(OneRaw)
{
  x = OneRaw[1:45] %>% as.numeric()
  y = OneRaw[46:(46+44)] %>% as.numeric()
  return(cosine(x ,y  ))
}
question_gramatical_entities_no[1,] %>% cosin_distance_pos_tags()
question_gramatical_entities_cosine <- question_gramatical_entities_no %>% apply(MARGIN =
1,FUN = cosin_distance_pos_tags) %>% as.data.frame()
colnames(question_gramatical_entities_cosine) = c('gramatical_entities_cosine')
save(question_gramatical_entities_cosine,file='question_gramatical_entities_cosine.RData')
```

PCA on grammatical entties statistics

```
# 5 Gramatical entities PCA
load('question2_gramatical_entities_no.RData')
load('question1 gramatical entities no.RData')
question1_gramatical_entities_no = question1_gramatical_entities_no %>% mutate_all(as.numeric)
question2_gramatical_entities_no = question2_gramatical_entities_no %>% mutate_all(as.numeric)
save(question2_gramatical_entities_no,file='question2_gramatical_entities_no.RData')
save(question1_gramatical_entities_no,file='question1_gramatical_entities_no.RData')
question1 gramatical entities pca <- question1 gramatical entities no %>% prcomp( scale. = F,
center = F)
question2_gramatical_entities_pca <- question2_gramatical_entities_no %>% prcomp( scale. = F,
center = F)
question1_gramatical_entities_pca <- question1_gramatical_entities_pca$x[,1:4]
question2_gramatical_entities_pca <- question2_gramatical_entities_pca$x[,1:4]
save(question1_gramatical_entities_pca,file='question1_gramatical_entities_pca.RData')
save(question2_gramatical_entities_pca,file='question2_gramatical_entities_pca.RData')
question_gramatical_entities_pca =
cbind(question1_gramatical_entities_pca,question2_gramatical_entities_pca)
save(question_gramatical_entities_pca,file='question_gramatical_entities_pca.RData')

Sequence distances by TraminR for POS tags:
# Feat.Sequence.distances.By.TramineR
Get_tag_seq <- function(Data_)
{
  Q1_tags <- Data_ %>% select(question1)%>% as.matrix() %>% apply(MARGIN = 1,POS_tags_seq )
%>% as.matrix()
  Q2_tags <- Data_ %>% select(question2)%>% as.matrix() %>% apply(MARGIN = 1,POS_tags_seq )
%>% as.matrix()
  rbind(Q1_tags,Q2_tags) %>% seqdef()
}
questions_Tag_sequences_Q1Q2 <- df %>% Get_tag_seq()
questions_Tag_sequences_Q1Q2 = Tag_sequences_Q1Q2
save(questions_Tag_sequences_Q1Q2, file = "questions_Tag_sequences_Q1Q2.RData")
feature_gramatical_entity_sequence <- function(OneRaw,Tag_sequences_Q1Q2,df)
{
  idx = OneRaw["id"] %>% as.integer() + 1
```

```
  list(  traminr_seqLLCS = seqLLCS(Tag_sequences_Q1Q2[idx,], Tag_sequences_Q1Q2[(nrow(df)+
idx),]),
         traminr_seqLLCP = seqLLCP(Tag_sequences_Q1Q2[idx,], Tag_sequences_Q1Q2[(nrow(df)+
idx),]),
         traminr_seqmpos = seqmpos(Tag_sequences_Q1Q2[idx,], Tag_sequences_Q1Q2[(nrow(df)+
idx),]) )
}
questions_Tag_sequences_distances <- df %>%
     apply(MARGIN = 1,feature_gramatical_entity_sequence,questions_Tag_sequences_Q1Q2,df ) %>%
     rbindlist( fill = TRUE)
save(questions_Tag_sequences_distances, file= 'questions_Tag_sequences_distances.RData')
```

Merge the features:

```
# Build table of all extracted features and merge the calculated features in one table
build_table_of_features <- function( no_cosine_distance_word_vec,
no_cosine_distance_by_dtm,no_traminr_seq,no_gramatic,no_word_numbers)
{
  load('question_similarity_distances.RData')
  load('df.RData')
  load('questions_Tag_sequences_distances.RData')
  load('question_gramatical_entities_pca.RData')
  question_gramatical_entities_pca = question_gramatical_entities_pca %>% as.data.frame()
  load('question_gramatical_entities_cosine.RData')
  if (!no_cosine_distance_word_vec){
    df_more_features1 = data.frame(q2_q1_cos_dist_max_l =
question_similarity_distances$q2_q1_cos_dist_max_l,
                                   q1_q2_cos_dist_max_c =
question_similarity_distances$q1_q2_cos_dist_max_c,
                                   q1_q2_cos_dist_max_l =
question_similarity_distances$q1_q2_cos_dist_max_l,
                                   q2_q1_cos_dist_max_c =
question_similarity_distances$q2_q1_cos_dist_max_c,
                                   q2 q1 cos dist min l =
question_similarity_distances$q2_q1_cos_dist_min_l,
                                   q1_q2_cos_dist_min_c =
question_similarity_distances$q1_q2_cos_dist_min_c,
                                   q1_q2_cos_dist_min_l =
question_similarity_distances$q1_q2_cos_dist_min_l,
                                   q2 q1 cos dist min c =
question_similarity_distances$q2_q1_cos_dist_min_c
                                   # is_duplicate = factor(df$is_duplicate
,levels=c(0,1),labels=c('Not Duplicated','Duplicated'))
                                   )
      df more features= df more features1
  }
  if(!no_cosine_distance_by_dtm ) {
    load('df_simelarities.RData')
    load('Relaxed_Word_Mover_dist.RData')
  df_more_features1 = data.frame(tfidf_lsa_cos_sim   = df_simelarities$tfidf_lsa_cos_sim,
                                 tfidf_cos_sim       = df_simelarities$tfidf_cos_sim,
                                 tfidf_lsa_dist_m2   = df_simelarities$tfidf_lsa_dist_m2,
                                 tfidf_lsa_dist_m3   = df_simelarities$tfidf_lsa_dist_m3,
                                d1_d2_cosine_sim    = df_simelarities$d1_d2_cosine_sim,
                                 rwmd_dist_colmean =
Relaxed_Word_Mover_dist$rwmd_dist_colmean,
                                 rwmd_dist_rowmean =
Relaxed_Word_Mover_dist$rwmd_dist_rowmean,
                                 rwmd_dist_diag    = Relaxed_Word_Mover_dist$rwmd_dist_diag
                               #  d1_d2_jac_sim       = df_simelarities$d1_d2_jac_sim,
                                  #  tfidf_lsa_dist_m1   = df_simelarities$tfidf_lsa_dist_m1,
                                  # d1_d2_jac_psim      = df_simelarities$d1_d2_jac_psim,
                                  # d1_d2_tfidf_lsa_cos_psim2 =
df_simelarities$d1_d2_tfidf_lsa_cos_psim2
                                 #  is_duplicate = factor(df$is_duplicate
,levels=c(0,1),labels=c('Not Duplicated','Duplicated'))
           )
    # df_more_features1 = df_more_features1 %>% mutate (d1_d2_jac_psim  = ifelse(
d1_d2_jac_psim %>% is.na() , 0, d1_d2_jac_psim ) )
   if (no_cosine_distance_word_vec ) {df_more_features = df_more_features1
    } else { df_more_features = cbind(df_more_features,df_more_features1) }
  }
  if(!no_traminr_seq)
```

```
  {
    df_more_features1 = data.frame(
      # traminr_seqLLCS = questions_Tag_sequences_distances$traminr_seqLLCS,
      traminr_seqLLCP = questions_Tag_sequences_distances$traminr_seqLLCP
     # traminr_seqmpos = questions_Tag_sequences_distances$traminr_seqmpos
    )
    if (no_cosine_distance_word_vec | no_cosine_distance_by_dtm ) {
      df_more_features = df_more_features1
    } else { df_more_features = cbind(df_more_features,df_more_features1) }
  }
 if(! no_gramatic )
  {
    df_more_features1 = data.frame(
      gramatical_entities_cosine  =
question_gramatical_entities_cosine$gramatical_entities_cosine,
      gramatical_entities_q1_PCA1 = question_gramatical_entities_pca$question1_PCA1,
      gramatical_entities_q1_PCA2 = question_gramatical_entities_pca$question1_PCA2,
     # gramatical_entities_q1_PCA3 = question_gramatical_entities_pca$question1_PCA3,
      #gramatical_entities_q1_PCA4 = question_gramatical_entities_pca$question1_PCA4,
      gramatical_entities_q2_PCA1 = question_gramatical_entities_pca$question2_PCA1,
      gramatical_entities_q2_PCA2 = question_gramatical_entities_pca$question2_PCA2
      #gramatical_entities_q2_PCA3 = question_gramatical_entities_pca$question2_PCA3,
      #gramatical_entities_q2_PCA4 = question_gramatical_entities_pca$question2_PCA4
      )
    if (no_cosine_distance_word_vec | no_cosine_distance_by_dtm | no_traminr_seq) {
      df_more_features = df_more_features1
    } else { df_more_features = cbind(df_more_features,df_more_features1) }
  }
  if(! no_word_numbers)
  {
    load('question1_number_of_words.RData')
    load('question2_number_of_words.RData')
    load('question_delta_words.RData')
    load('question_no_stop_words.RData')
    df_more_features1 = data.frame(
      question1_no_words = question1_number_of_words$question1_no_words,
      question2_no_words = question2_number_of_words$question2_no_words,
      # delta_q1_q2 = question_delta_words$delta_q1_q2,
      # delta_q2_q1 = question_delta_words$delta_q2_q1,
      question1_no_stop_words = question_no_stop_words$question1_no_stop_words,
      question2_no_stop_words = question_no_stop_words$question2_no_stop_words )
    if (no_cosine_distance_word_vec | no_cosine_distance_by_dtm | no_traminr_seq |
no_word_numbers) {
      df_more_features = df_more_features1
    } else { df_more_features = cbind(df_more_features,df_more_features1) }
    question1_number_of_words %>% colnames()
  }
   df_more_features = cbind(df_more_features, is_duplicate = df$is_duplicate %>% as.factor( )
)
  df_more_features
  }
df_more_features = build_table_of_features(F,F,F,F,F)
```

World Vector by Glove

```
# Glove.word2vec -----------------
GLOVE_DIR <- 'glove.6B'
EMBEDDING_DIM <- 100
get_text_for_text2vec <- function(Path)
{
  texts <- character()  # text samples
  fpath_table <- Path %>% file.path(., list.files(.))
  pb <- txtProgressBar( style = 3, title = "Read",min = 0,max =fpath_table %>% length, initial
= 0 )
  for (fpath in fpath_table)
  {
    t <- readLines(fpath, encoding = "latin1")
    t <- paste(t, collapse = "\n")
    i <- regexpr(pattern = '\n\n', t, fixed = TRUE)[[1]]
    if (i != -1L)
      t <- substring(t, i)
    texts <- c(texts, t)
  }
  texts
}
```

```
tokenize_text_ <- function(texts_)
{
  # Create iterator over tokens
  it <- texts_ %>% space_tokenizer( ) %>% itoken( progressbar = FALSE)
  it
}

glove_create_vocab<- function(itokens)
{
  vocab <- itokens %>% create_vocabulary( )
  # vocab <- prune_vocabulary(vocab, term_count_min = 1L)
  list(vocab=vocab, itokens = itokens)
}

glove_create_vectorizer <- function(parameters)
{
  # Use our filtered vocabulary
  vectorizer <- parameters$vocab %>% vocab_vectorizer( )
  list( vectorizer = vectorizer,
        vocab= parameters$vocab,
        itokens = parameters$itokens)
}

glove_create_tcm <- function(parameters)
{
  # use window of 5 for context words
  tcm <- parameters$itokens %>% create_tcm(parameters$vectorizer,
                                           skip_grams_window = 5L)
  list(tcm = tcm,
       vocab= parameters$vocab)
}

glove_get_word_vector <- function(parameters)
{
  glove = GlobalVectors$new( word_vectors_size = 100,
                             vocabulary = parameters$vocab, x_max = 10)
  wv_main <- glove$fit_transform(parameters$tcm, n_iter = 20)
  wv_context = glove$components
  word_vectors <- wv_main + t(wv_context)
   word_vectors
}

merge_with_pretrained_word_vector <- function( word_vectors_glove,embeddings_index) {
  vect_words <- word_vectors_glove %>% row.names() %>% as.character() %>% as.data.frame( )
  colnames(vect_words) = c('word')
  row.names(vect_words )= vect_words$word
  vect_words <- vect_words %>% mutate(word = as.character(word))
  i = 1
  for( word in vect_words$word)
  {
    wvec = embeddings_index[[word]]
    if( ! wvec %>% is.null() ) word_vectors_glove[i,] = wvec
    i = i + 1
  }
#   Add two empty vectors at the end
   v <- rep_len(0.0,length.out = 100)
   word_vectors_glove <-  rbind(word_vectors_glove,v)
   word_vectors_glove <-  rbind(word_vectors_glove,v)
   word_vectors_glove <-  rbind(word_vectors_glove,v)
   word_vectors_glove
  }
download_data <- function(data_dir, url_path, data_file) {
  if (!dir.exists(data_dir)) {
    download.file(paste0(url_path, data_file), data_file, mode = "wb")
    if (tools::file_ext(data_file) == "zip")
      unzip(data_file, exdir = tools::file_path_sans_ext(data_file))
    else
      untar(data_file)
    unlink(data_file)
  }
}
download_data(GLOVE_DIR, 'http://nlp.stanford.edu/data/', 'glove.6B.zip')

word_vectors_glove <- rbind(df$question1,df$question2) %>%
```

```
  tokenize_text_() %>%
  glove_create_vocab() %>%
  glove_create_vectorizer() %>%
  glove_create_tcm() %>%
  glove_get_word_vector() %>%
  merge_with_pretrained_word_vector(embeddings_index)
 save(word_vectors_glove,file="word_vectors_glove.RData")
```

T-SNE for word vector and projection in 2D

T-SNE.WORD.VEC.2D.PLOT

```
tsne_word_of_interest <- function(words_of_interst1) {
set.seed(9)
tsne_model_1 = Rtsne(as.matrix(words_of_interst1), check_duplicates=FALSE, pca=TRUE,
perplexity=3,      theta=0.5, dims=2)
## getting the two dimensions matrix
d tsne 1 = data.frame(tsne model 1$Y, term =row.names(words of interst1) )
row.names(d_tsne_1) =  row.names(words_of_interst1)

  ggplot(d_tsne_1, aes(x= X1, y = X2)) +
    geom_point(color = "blue", size = 1) +
    geom_label_repel(aes(label = term),
                     segment.color = "grey50") +
    theme_classic()
}
word_vectors[ , , drop = FALSE] %>% tsne_word_of_interest()
```

Words of interest:

word.interest.plot

```
get_similar_words <- function( word ) {
  word_vec = word_vectors[word, , drop = FALSE]
  cos_sim = sim2(x = word_vectors, y = word_vec, method = "cosine", norm = "l2")
  word_nighbours_ <- head(sort(cos_sim[,1], decreasing = TRUE), 10)
  word_nighbours_ %>% names()
}
get_some_word_of_interst <- function() {
  word_of_interst <- c("children","family","health","safety","happy")
  word nighbours <- word of interst %>% lapply(FUN = get similar words)
  word_nighbours <- word_nighbours %>% unlist() %>%sort() %>%  unique()
   word_nighbours %>% unlist()
}
get_similar_words("family")
get_some_word_of_interst() %>% word_vectors[., , drop = FALSE] %>% tsne_word_of_interest()
```

Pair questions word vectors plot:

PAIR_QUESTION_WORD_VECT_PLOT

```
plot_word_vector_questions_vectors <- function(OnRaw)
{
   tokenQ1 =   tokenize_words( OnRaw["question1"] ) %>% unlist() %>% sort() %>%  unique()
   tokenQ2 =   tokenize_words( OnRaw["question2"] ) %>% unlist() %>% sort() %>%  unique()
  d1 <- setdiff(tokenQ1, tokenQ2 ) %>% sort() %>%  unique()
   d2 <- setdiff(tokenQ2, tokenQ1 )  %>% sort() %>%  unique()
    words_of_d1  <- d1 %>% word_vectors[., , drop = FALSE]
   words_of_q2  <- tokenQ2 %>% word_vectors[., , drop = FALSE]
    word_interest <- rbind(d1, tokenQ2)  %>% sort() %>%  unique()
   word_interest_v <- word_interest %>%  word_vectors[., , drop = FALSE]
    set.seed(50)
   tsne_model_1 = word_interest_v %>%  as.matrix( ) %>%  Rtsne(. , check_duplicates=FALSE,
pca=TRUE, perplexity=3, theta=0.5, dims=2)
   ## getting the two dimension matrix
   d_tsne_1 = data.frame(tsne_model_1$Y, term =row.names(word_interest_v) )
   row.names(d_tsne_1) =  row.names(word_interest_v)
      ggplot(d_tsne_1, aes(x= X1, y = X2)) +
     geom_point(color = "blue", size = 1) +
     geom_label_repel(aes(label = term),
                      segment.color = "grey50") +
     geom_segment(aes(x = 0, y = 0, xend = X1, yend = X2), data = d_tsne_1[tokenQ2,-3],arrow =
arrow())+
```

```
     geom_segment(aes(x = 0, y = 0, xend = X1, yend = X2, color="red"), data = d_tsne_1[d1,-
3],arrow = arrow())+
     geom_vline(xintercept = 0,color = "blue")+
     geom_hline(yintercept = 0,color = "blue")+
     theme_classic()

}
df[2,] %>% plot_word_vector_questions_vectors()
```

Cosine distances between pair of questions:

HEATMAP_QUESTIONS_COSINE_DISTANCE

```
plot_conv_matrix_q1q2 <- function(OneRaw,word_vectors) {
  question1 <- OneRaw["question1"] %>% as.character()
  question2 <- OneRaw["question2"] %>% as.character()
  v = tibble(question1,question2)
  v_q1 <- v %>% select(question1)   %>% unnest_tokens(word,question1) %>% as.vector() %>% t()
  v_q2 <- v %>% select(question2)   %>% unnest_tokens(word,question2) %>% as.vector() %>% t()
  token_q1 <- v_q1  %>%  word_vectors[.,,drop=FALSE]
  token_q2 <- v_q2  %>%  word_vectors[.,,drop=FALSE]
  covMatrixQ1Q2 = token_q1 %>% sim2( y = token_q2, method = "cosine", norm = "l2")
  covMatrixQ1Q2_melt = melt(covMatrixQ1Q2)
  colnames(covMatrixQ1Q2_melt) = c('question1','question2','value')
  p <- ggplot(covMatrixQ1Q2 melt , aes(question1,question2)) +
    geom_tile(aes(fill = value),colour = "white") +
    # scale_fill_gradient(low = "white", high = "steelblue") +
    scale_fill_gradient2(low = "blue", high = "red", mid = "white",  midpoint = 0, limit = c(-
1.3,1.3), space = "Lab", name="Cosine Similarity")+
    theme grey(base size = 10) + labs(x = "question1",y = "question2") +
    scale_x_discrete(expand = c(0, 0)) +
    scale_y_discrete(expand = c(0, 0) ) +
    theme_minimal()+
    theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                     size = 12, hjust = 1))+
    coord fixed()
    # theme(legend.position = "none",strip.text.x= element_text(size = 6, angle = 90, hjust =
0, colour = "grey50"))
   p
}
```

multiplot

multiplot copied from: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/

```
# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
```

```
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}


v1 = which(df$is_duplicate == 1)[100:200] %>% df[.,] %>%   apply(MARGIN=1,FUN =
plot_conv_matrix_q1q2,word_vectors)

v1 %>% multiplot(cols = 3)

v <- cbind( question1_number_of_words,question2_number_of_words, is_duplicate = df$is_duplicate,id=
df$id )

v <- v %>% filter( question1_no_words < 10 & question2_no_words <10 & is_duplicate == 0 ) %>%
select(id) %>% mutate(id= id+1)

v1 = v[1:40,] %>% df[.,] %>%   apply(MARGIN=1,FUN = plot_conv_matrix_q1q2,word_vectors)

v1 %>% multiplot(cols = 3)
```

COSINE_DIST.CALC.ATTR

```
calculate_delta_distance_min_max <- function(OneRaw,word_vectors)
{
  question1 <- OneRaw["question1"] %>% as.character()
  question2 <- OneRaw["question2"] %>% as.character()

  v = tibble(question1 = question1,question2 = question2)
  #v = tibble(question1,question2)

  Q1 <- v  %>% select(question1)  %>% unnest_tokens(word,question1)
  Q2 <- v  %>% select(question2)  %>% unnest_tokens(word,question2)

  token_q1 = Q1 %>% as.vector() %>% t()
  token_q2 = Q2 %>% as.vector() %>% t()

  token_q1_q2 <- setdiff(token_q1, token_q2 )
  v1 = token_q1_q2 %>% as.vector() %>% t() %>% word_vectors[.,,drop=FALSE]
  v2 = token_q2    %>% as.vector() %>% t() %>% word_vectors[.,,drop=FALSE]
  covMatrixQ1Q2 = v1 %>% sim2( y = v2, method = "cosine", norm = "l2")
  # covMatrixQ1Q2 = ifelse(token_q2 %>% length() == 0 | token_q1_q2 %>% length() == 0 ,0,v1
%>% sim2( y = v2, method = "cosine", norm = "l2"))
  minimum_cov_distQ1Q2_l <- apply(covMatrixQ1Q2, 1, min)  %>% sum()
  maximum_cov_distQ1Q2_l <- apply(covMatrixQ1Q2, 1, max)  %>% sum()
  moyenne_cov_distQ1Q2_l <- apply(covMatrixQ1Q2, 1, mean) %>% sum()
  minimum_cov_distQ1Q2_c <- apply(covMatrixQ1Q2, 2, min)  %>% sum()
  maximum_cov_distQ1Q2_c <- apply(covMatrixQ1Q2, 2, max)  %>% sum()
  moyenne_cov_distQ1Q2_c <- apply(covMatrixQ1Q2, 2, mean) %>% sum()
  token_q2_q1 <- setdiff(token_q2, token_q1 )
  v1 = token_q2_q1 %>% as.vector() %>% t() %>% word_vectors[.,,drop=FALSE]
  v2 = token_q1    %>% as.vector() %>% t() %>% word_vectors[.,,drop=FALSE]
  # covMatrixQ2Q1 = ifelse(token_q1 %>% length() == 0 | token_q2_q1 %>% length() == 0 ,0,v1
%>% sim2( y = v2, method = "cosine", norm = "l2"))
  covMatrixQ2Q1 = v1 %>% sim2( y = v2, method = "cosine", norm = "l2")
  minimum_cov_distQ2Q1_l <- apply(covMatrixQ2Q1, 1, min)  %>% sum()
  maximum_cov_distQ2Q1_l <- apply(covMatrixQ2Q1, 1, max)  %>% sum()
  moyenne_cov_distQ2Q1_l <- apply(covMatrixQ2Q1, 1, mean) %>% sum()
```

```
  minimum_cov_distQ2Q1_c <- apply(covMatrixQ2Q1, 2, min)  %>% sum()
  maximum_cov_distQ2Q1_c <- apply(covMatrixQ2Q1, 2, max)  %>% sum()
  moyenne_cov_distQ2Q1_c <- apply(covMatrixQ2Q1, 2, mean) %>% sum()
 list(  q1_q2_cos_dist_min_l  = ifelse(minimum_cov_distQ1Q2_l %>%
is.infinite(),0,minimum_cov_distQ1Q2_l),
         q1_q2_cos_dist_max_l  = ifelse(maximum_cov_distQ1Q2_l %>%
is.infinite(),0,maximum_cov_distQ1Q2_l),
         q1_q2_cos_dist_mean_l = ifelse(moyenne_cov_distQ1Q2_l %>%
is.infinite()|moyenne_cov_distQ1Q2_l %>% is.nan,0,moyenne_cov_distQ1Q2_l),
         q2_q1_cos_dist_min_l  = ifelse(minimum_cov_distQ2Q1_l %>%
is.infinite(),0,minimum_cov_distQ2Q1_l),
         q2_q1_cos_dist_max_l  = ifelse(maximum_cov_distQ2Q1_l %>%
is.infinite(),0,maximum_cov_distQ2Q1_l),
         q2_q1_cos_dist_mean_l = ifelse(moyenne_cov_distQ2Q1_l %>%
is.infinite()|moyenne_cov_distQ2Q1_l %>% is.nan,0,moyenne_cov_distQ2Q1_l),
         q1_q2_cos_dist_min_c  = ifelse(minimum_cov_distQ1Q2_c %>% is.infinite() |
minimum_cov_distQ1Q2_c %>% is.nan,0,minimum_cov_distQ1Q2_c),
         q1_q2_cos_dist_max_c  = ifelse(maximum_cov_distQ1Q2_c %>% is.infinite() |
maximum_cov_distQ1Q2_c %>% is.nan,0,maximum_cov_distQ1Q2_c),
         q1_q2_cos_dist_mean_c = ifelse(moyenne_cov_distQ1Q2_c %>% is.infinite() |
moyenne_cov_distQ1Q2_c %>% is.nan,0,moyenne_cov_distQ1Q2_c),
         q2_q1_cos_dist_min_c  = ifelse(minimum_cov_distQ2Q1_c %>% is.infinite() |
minimum cov distQ2Q1 c %>% is.nan,0,minimum cov distQ2Q1 c),
         q2_q1_cos_dist_max_c  = ifelse(maximum_cov_distQ2Q1_c %>% is.infinite() |
maximum_cov_distQ2Q1_c %>% is.nan,0,maximum_cov_distQ2Q1_c),
         q2_q1_cos_dist_mean_c = ifelse(moyenne_cov_distQ2Q1_c %>% is.infinite() |
moyenne_cov_distQ2Q1_c %>% is.nan,0,moyenne_cov_distQ2Q1_c))
}
load('word_vectors_glove.RData')
df[1,] %>% calculate_delta_distance_min_max(word_vectors_glove )
question_similarity_distances <- df %>% apply(MARGIN =
1,calculate_delta_distance_min_max,word_vectors_glove )%>% rbindlist(fill =  F)
save(question_similarity_distances,file='question_similarity_distances.RData')
```

SIMILARITY.DISTANCES.PLOTS

 # Plot similarity distances

```
load('question_similarity_distances.RData')
colnames(question_similarity_distances)
plot_data_similarity_distances = cbind(question_similarity_distances,is_duplicate =
df$is_duplicate)
splom(~ plot_data_similarity_distances)
```

 # Plot densities density of similarity distances

```
plot_data_similarity_distances %>% View()
plot_data_similarity_distances = question_similarity_distances
plot_data_similarity_distances <- plot_data_similarity_distances %>% gather( key =
variable_name, value= distance_value  )
plot_data_similarity_distances <- cbind(plot_data_similarity_distances,is_duplicate =
df$is_duplicate)

ggplot(plot_data_similarity_distances, aes(distance_value,  colour=variable_name)) +
 geom_density() + xlim(-10,+30) +facet_grid(is_duplicate ~ .)

plot_data_similarity_distances = plot_data_similarity_distances %>%
  mutate(is_duplicate =  as.factor(is_duplicate,labels = c("Not Duplicated","Duplicated")  ) )

plot_data_similarity_distances$is_duplicate =
factor(plot_data_similarity_distances$is_duplicate,labels = c("Not Duplicated","Duplicated"))
plot_data_similarity_distances$is_duplicate %>% labels()
p1 <- plot data similarity distances %>% filter(variable name == "q1 q2 cos dist min l"
                                        | variable_name == "q1_q2_cos_dist_max_l"
                                        | variable_name == "q1_q2_cos_dist_mean_l") %>%
ggplot( aes(distance_value,  colour=variable_name)) +
  xlab("Cosine Distance By Line")+
  geom_density() + xlim(-10,+30) +facet_grid(is_duplicate ~ .)

p2 <- plot_data_similarity_distances %>% filter(variable_name == "q2_q1_cos_dist_min_l"
                                                | variable_name == "q2_q1_cos_dist_max_l"
                                                | variable_name == "q2_q1_cos_dist_mean_l")
%>%
  ggplot( aes(distance_value,  colour=variable_name)) +
  xlab("Cosine Distance By Line")+
```

```
  geom_density() + xlim(-10,+30) +facet_grid(is_duplicate ~ .)
p3 <- plot_data_similarity_distances %>% filter(variable_name == "q1_q2_cos_dist_min_c"
                                                | variable_name == "q1_q2_cos_dist_max_c"
                                                | variable_name == "q1_q2_cos_dist_mean_c")
%>%
  ggplot( aes(distance_value,  colour=variable_name)) +
  xlab("Cosine Distance By Column")+
  geom_density() + xlim(-10,+30) +facet_grid(is_duplicate ~ .)
p4 <- plot_data_similarity_distances %>% filter(variable_name == "q2_q1_cos_dist_min_c"
                                                | variable_name == "q2_q1_cos_dist_max_c"
                                                | variable_name == "q2_q1_cos_dist_mean_c")
%>%
  ggplot( aes(distance_value,  colour=variable_name)) +
  xlab("Cosine Distance By Column")+
  geom_density() + xlim(-10,+30) +facet_grid(is_duplicate ~ .)
multiplot(p1,p2,p3,p4,cols=2)

# Plot pair attributs
col_comb <-
c('q2_q1_cos_dist_min_c','q2_q1_cos_dist_min_1','q1_q2_cos_dist_min_c','q1_q2_cos_dist_min_1')
col_comb = crossing(col_comb, col_comb) %>% filter(col_comb !=col_comb1 ) %>% as.matrix()
col_comb
plot similarity pair variables <- function(v1)
{
  cbind(question_similarity_distances,is_duplicate=df$is_duplicate) %>%
    ggplot(aes(get(v1[1]), get(v1[2]),colour= is_duplicate))+
    xlab(v1[1]) + ylab(v1[2])+
    geom_point()
}
plots_similariy_pairs <- col_comb %>%  apply(MARGIN = 1,FUN = plot_similarity_pair_variables)
multiplot(plots_similariy_pairs[[1]],plots_similariy_pairs[[2]],plots_similariy_pairs[[3]],plo
ts_similariy_pairs[[4]],
          plots_similariy_pairs[[5]],plots_similariy_pairs[[6]],plots_similariy_pairs[[7]],
          plots_similariy_pairs[[8]],plots_similariy_pairs[[9]],plots_similariy_pairs[[10]],
          plots_similariy_pairs[[11]],plots_similariy_pairs[[12]],cols = 4)
```

#More features: DTM, TFIDF, LSA, similarity distances, Jaccard:

```
# FEAT.DTM.TFIDF.LSA.SIM.DIST.JACC
get_Relaxed_Word_Mover_dist <- function(dtm1,dtm2,word_vectors_glove)
{
  start = 0
  end_   = 0
  nrow_df = nrow(dtm1 )
  pkg_size = 1000
  V = (nrow_df/pkg_size) %>% round() + 1
  rwmd_model = RWMD$new(word_vectors_glove,method = "cosine",progressbar = F )
  data_frame_  = data.frame()
  for (i in 1:V)
  {
    start = ( i - 1 ) * pkg_size + 1
    end_ = ifelse( end_ >= ( nrow_df - pkg_size ), nrow_df, (i * pkg_size))
   rwmd_dist_mat = dist2(dtm1[start:end_,],dtm2[start:end_,], method = rwmd_model, norm =
'none') %>%  as.matrix()
     rwmd_dist_colmean =  rwmd_dist_mat %>%
      colMeans()
    rwmd_dist_rowmean =  rwmd_dist_mat %>%
      rowMeans()
    rwmd_dist_diag =  rwmd_dist_mat %>%
      diag()
     data_frame_ = rbind(data_frame_,data.frame( rwmd_dist_colmean = rwmd_dist_colmean,
                                                 rwmd_dist_rowmean = rwmd_dist_rowmean,
                                                 rwmd_dist_diag    = rwmd_dist_diag)  )
    }
  data_frame_
}
Relaxed_Word_Mover_dist =
get_Relaxed_Word_Mover_dist(dtm_tfidf_lsa$dtm1,dtm_tfidf_lsa$dtm2,word_vectors_glove)
save(Relaxed_Word_Mover_dist,file='Relaxed_Word_Mover_dist.RData')
get_vocab_vectorizer <- function(Data)
```

```
  {
      vectorizer <- rbind(Data$question1 ,Data$question2) %>%
      itoken()  %>%
      create_vocabulary( ) %>%
      prune_vocabulary(term_count_min = 1L) %>%
      vocab_vectorizer( )
    list(vectorizer = vectorizer, df = Data)
}
get_dtm_ <- function(Input_params) {
  dtm1 = Input_params$df$question1 %>% itoken(progressbar = FALSE) %>% create_dtm(
Input_params$vectorizer)
  dtm2 = Input_params$df$question2 %>% itoken(progressbar = FALSE) %>% create_dtm(
Input_params$vectorizer)
  list(dtm1 = dtm1, dtm2 = dtm2,df = Input_params$df)
}
get_tfidf_ <- function(dtm) {
  tfidf = TfIdf$new()
  dtm1_tfidf = fit_transform(dtm$dtm1, tfidf)
  dtm2_tfidf = fit_transform(dtm$dtm2, tfidf)
  list(dtm1_tfidf = dtm1_tfidf, dtm2_tfidf = dtm2_tfidf, df = dtm$df, dtm1 = dtm$dtm1, dtm2 =
dtm$dtm2 )
}
get_lsa  <- function( dtm tfidf) {
  lsa = LSA$new( n_topics = 100)
  dtm1_tfidf_lsa = fit_transform(dtm_tfidf$dtm1_tfidf, lsa)
  dtm2_tfidf_lsa = fit_transform(dtm_tfidf$dtm2_tfidf, lsa)
  list(dtm1_tfidf_lsa = dtm1_tfidf_lsa,
       dtm2_tfidf_lsa = dtm2_tfidf_lsa,
       dtm1_tfidf     = dtm_tfidf$dtm1_tfidf,
       dtm2_tfidf     = dtm_tfidf$dtm2_tfidf,
       dtm1 = dtm_tfidf$dtm1,
       dtm2 = dtm_tfidf$dtm2,
       df = dtm_tfidf$df )
}
calc_similarities <- function(dtm_tfidf_lsa)
{
  start = 0
  end   = 0
  nrow_df = nrow(dtm_tfidf_lsa$df )
  pkg_size = 10000
  V = (nrow_df/pkg_size) %>% round() + 1
  d1_d2_tfidf_lsa_sim = data.frame()
  d1_d2_tfidf_cos_sim = data.frame()
  tfidf_lsa_dist_m1    = data.frame()
  tfidf_lsa_dist_m2    = data.frame()
  tfidf_lsa_dist_m3    = data.frame()
  d1_d2_tfidf_lsa_cos_sim  = data.frame()
  d1_d2_jac_sim    = data.frame()
  d1_d2_cosine_sim = data.frame()
  for (i in 1:V)
  {
    start = ( i - 1 ) * pkg_size + 1
    end = ifelse( end >= ( nrow_df - pkg_size ), nrow_df, (i * pkg_size))

    m1 = dist2(dtm_tfidf_lsa$dtm1_tfidf_lsa[start:end,],
dtm_tfidf_lsa$dtm2_tfidf_lsa[start:end,], method = "euclidean") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()

    m2 = dist2(dtm_tfidf_lsa$dtm1_tfidf_lsa[start:end,],
dtm_tfidf_lsa$dtm2_tfidf_lsa[start:end,], method = "euclidean",norm="l1") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()

    m3 = dist2(dtm tfidf lsa$dtm1 tfidf lsa[start:end,],
dtm_tfidf_lsa$dtm2_tfidf_lsa[start:end,], method = "euclidean",norm="none") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()
    d1_d2_jac_sim1 = sim2(dtm_tfidf_lsa$dtm1[start:end,], dtm_tfidf_lsa$dtm2[start:end,],
method = "jaccard", norm = "none") %>%
      as.matrix()%>%
      diag(.) %>%
```

```
      as.data.frame()
    d1_d2_cosine_sim1 = sim2(dtm_tfidf_lsa$dtm1[start:end,], dtm_tfidf_lsa$dtm2[start:end,],
method = "cosine", norm = "l2") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()

    d1_d2_tfidf_cos_sim1 = sim2(dtm_tfidf_lsa$dtm1_tfidf[start:end,],
dtm_tfidf_lsa$dtm2_tfidf[start:end,], method = "cosine", norm = "l2") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()

    d1_d2_tfidf_lsa_cos_sim1 = sim2(dtm_tfidf_lsa$dtm1_tfidf_lsa[start:end,],
dtm_tfidf_lsa$dtm2_tfidf_lsa[start:end,], method = "cosine", norm = "l2") %>%
      as.matrix()%>%
      diag(.) %>%
      as.data.frame()
    d1_d2_jac_sim    = rbind(d1_d2_jac_sim,d1_d2_jac_sim1)
    d1_d2_cosine_sim = rbind(d1_d2_cosine_sim,d1_d2_cosine_sim1)
     d1_d2_tfidf_lsa_cos_sim = rbind(d1_d2_tfidf_lsa_cos_sim,d1_d2_tfidf_lsa_cos_sim1)
    d1_d2_tfidf_cos_sim     = rbind(d1_d2_tfidf_cos_sim,d1_d2_tfidf_cos_sim1)
    tfidf lsa dist m1        = rbind(tfidf lsa dist m1, m1 )
    tfidf_lsa_dist_m2        = rbind(tfidf_lsa_dist_m2, m2 )
    tfidf_lsa_dist_m3        = rbind(tfidf_lsa_dist_m3, m3 )
     if( end >= nrow_df) break
      }
  d1_d2_jac_psim = psim2(dtm_tfidf_lsa$dtm1, dtm_tfidf_lsa$dtm2, method = "jaccard", norm =
"none") %>% as.data.frame()
  d1_d2_tfidf_lsa_cos_psim2 = psim2(dtm_tfidf_lsa$dtm1_tfidf_lsa,
dtm_tfidf_lsa$dtm2_tfidf_lsa, method = "cosine", norm = "l2") %>% as.data.frame()
    data_frame_ = data.frame(d1_d2_tfidf_lsa_cos_sim,
                             d1_d2_tfidf_cos_sim,
                             tfidf_lsa_dist_m1,
                             tfidf_lsa_dist_m2 ,
                             tfidf_lsa_dist_m3,
                             d1_d2_jac_sim,
                             d1_d2_cosine_sim,
                             d1_d2_jac_psim,
                             d1_d2_tfidf_lsa_cos_psim2,
                             dtm_tfidf_lsa$df$is_duplicate)
  colnames(data_frame_) = c("tfidf_lsa_cos_sim",
                            "tfidf_cos_sim",
                            "tfidf_lsa_dist_m1",
                            "tfidf_lsa_dist_m2",
                            "tfidf_lsa_dist_m3",
                            "d1_d2_jac_sim",
                            "d1_d2_cosine_sim",
                            "d1_d2_jac_psim",
                            "d1_d2_tfidf_lsa_cos_psim2",
                            "is_duplicate")
  data_frame_
}
dtm_tfidf_lsa <- df %>% get_vocab_vectorizer()  %>%  get_dtm_( ) %>% get_tfidf_() %>%
get_lsa_()
df_simelarities <- dtm_tfidf_lsa %>% calc_similarities()
save(df_simelarities, file = "df_simelarities.RData")
```

PLOTS.DTM.TFIDF.LSA.SIM.DIST

 # Densities

```
plot_data_sim_dist_by_tfidf <- df_more_features[,-7] %>% gather( key = variable_name, value=
distance_value  )
plot_data_sim_dist_by_tfidf <- cbind(plot_data_sim_dist_by_tfidf,is_duplicate =
df$is_duplicate)
p1 <- plot_data_sim_dist_by_tfidf %>%
  ggplot( aes(distance_value,  colour=variable_name)) +
  xlab("Cosine Distance")+
  geom_density() + xlim(-.5,+3) +facet_grid(is_duplicate ~ .)

p1
```

 # Pair of attributes

```
col_comb <- df_more_features[,-7] %>% colnames()
col_comb = crossing(col_comb, col_comb ) %>% filter(col_comb !=col_comb1 ) %>% as.data.frame()
```

```
col_comb <- col_comb[!duplicated(t(apply(col_comb, 1, sort))),]

plot_similarity_pair_variables <- function(v1,df_more_features)
{
  cbind(df_more_features,is_duplicate=df$is_duplicate) %>%
    ggplot(aes(get(v1[1]), get(v1[2]),colour= is_duplicate))+
    xlab(v1[1]) + ylab(v1[2])+
    geom_point()
}
plots_similariy_pairs <- col_comb %>%  apply(MARGIN = 1,FUN =
plot_similarity_pair_variables,df_more_features)
plots_similariy_pairs %>% length
multiplot(plots_similariy_pairs[[1]],plots_similariy_pairs[[2]],plots_similariy_pairs[[2]],plo
ts_similariy_pairs[[3]],plots_similariy_pairs[[4]],plots_similariy_pairs[[5]],

plots_similariy_pairs[[6]],plots_similariy_pairs[[7]],plots_similariy_pairs[[8]],plots_similar
iy_pairs[[9]],plots_similariy_pairs[[10]],plots_similariy_pairs[[11]],plots_similariy_pairs[[1
2]],plots_similariy_pairs[[13]],plots_similariy_pairs[[14]],plots_similariy_pairs[[15]],
cols = 5)
```

 # Correlations

```
df_more_features[,-c(7)] %>% cor(method = c("pearson", "kendall", "spearman")) %>%
  corrplot( type="upper", order="hclust", tl.col="black", tl.srt=45)
```

MACHINE.LEARNING.MODELS

Cross Validation

```
get_model_train.predict_metrics <- function(data.partition,Model_Alg)
{
  require(class)
  require(MASS)
  require(randomForest)
  require(e1071)
  require(nnet)
  require(ROCR)

  if (Model_Alg == 'glm')
  {
     model.fits = glm(is_duplicate~. , data = data.partition$train ,family = binomial(link =
"logit")  )
  } else if(Model_Alg == 'lda') {
    model.fits = lda(is_duplicate~. , data = data.partition$train  )

  } else if(Model_Alg == 'qda') {

    model.fits = qda(is_duplicate~. , data = data.partition$train  )
  } else if(Model_Alg == 'knn') {
    model.fits = knn( train = data.partition$train , test = data.partition$test ,
                      cl = data.partition$train$is_duplicate, k=20)
   } else if(Model_Alg == 'rpart')
   {
     model.fits = rpart(is_duplicate ~.  , data = data.partition$train )

   } else if(Model_Alg == 'randomForest')
   {
     model.fits = randomForest(is_duplicate ~. , data = data.partition$train, ntree = 201,
mtry = 10)
   } else if(Model_Alg == 'naiveBayes'){
     model.fits <- naiveBayes(is_duplicate ~. , data = data.partition$train)

   } else if(Model_Alg == 'nnet') {
     # maxs <- apply(data.partition$train[,-28] , 2, max)
     # mins <- apply(data.partition$train[,-28] , 2, min)
     # train_data <- data.partition$train[,-28] %>% scale(center = mins, scale = maxs-mins)
%>% as.data.frame()
     # train_data = cbind( train_data , is_duplicate = data.partition$train[,28] %>%
as.numeric)
     #
     # maxs <- apply(data.partition$test[,-28] , 2, max)
     # mins <- apply(data.partition$test[,-28] , 2, min)
     # test_data <- data.partition$test[,-28] %>% scale(center = mins, scale = maxs-mins) %>%
as.data.frame()
     # test_data = cbind( test_data , is_duplicate = data.partition$test[,28] %>% as.numeric()
)
```

```
     #
    model.fits <- nnet(is_duplicate ~., data = data.partition$train, size = 25)
   } else if(Model_Alg == 'svm' )
   {
     model.fits <- svm(is_duplicate ~. , data =
data.partition$train[1:10000,],kernel="linear", probability = T)
   } else if(Model_Alg == 'neuralnet'){
     library(neuralnet)
     collabel = 25
     maxs <- apply(data.partition$train[,-collabel] , 2, max)
     mins <- apply(data.partition$train[,-collabel] , 2, min)
     train_data <- data.partition$train[,-collabel] %>% scale(center = mins, scale = maxs-
mins) %>% as.data.frame()
     train_data = cbind( train_data, is_duplicate = data.partition$train[,collabel] %>%
as.numeric)
     maxs <- apply(data.partition$test[,-collabel] , 2, max)
     mins <- apply(data.partition$test[,-collabel] , 2, min)
     test_data <- data.partition$test[,-collabel] %>% scale(center = mins, scale = maxs-mins)
%>% as.data.frame()
     test_data = cbind( test_data , is_duplicate = data.partition$test[,collabel] %>%
as.numeric() )

     test data[,-6 ] %>% colnames
     formula1 = test_data[,-collabel] %>% colnames %>%  paste0(collapse = ' + ')
     formula1 = paste0('is_duplicate ~ ' ,formula1) %>%  as.formula()

     model.deepnn <- neuralnet(formula1 , data=train_data , hidden=c(50,50,50,50),act.fct =
"logistic",
                               linear.output = FALSE)
   }


  if(Model_Alg == 'rpart'  | Model_Alg == 'randomForest' ){
    model.probs = predict(model.fits,data.partition$test, type = "prob")[,2] %>% as.numeric()
   } else if(Model_Alg == 'lda' | Model_Alg == 'qda' )
  {
    model.probs = predict(model.fits,data.partition$test, type="response")$posterior[,2]%>%
as.numeric()
  } else if(Model_Alg == 'naiveBayes') {
    model.probs <- predict(model.fits, newdata = data.partition$test,type ="raw")[,2] %>%
as.numeric()

  } else if(Model_Alg == 'nnet'){


    model.probs <- predict(model.fits, newdata = data.partition$test, type = "raw")%>%
as.numeric()

  } else if(Model_Alg == 'glm')
    {
      model.probs = predict(model.fits,data.partition$test, type="response") %>% as.numeric()
  } else if(Model_Alg == 'svm')
  {
    model.probs = predict(model.fits,data.partition$test, type = 'prob') %>% as.numeric()
    model.probs = ifelse(model.probs == 2,1,0)
  } else if(Model_ALg == 'neuralnet' )
  {

    model.probs = compute(model.deepnn,test_data[,-27])$net.result[,1]


  }


  model.pred = ifelse(model.probs > .5,1,0)

  # model.pred = model.pred %>% as.numeric()
  cross.classify.table = table(model.pred ,data.partition$test$is_duplicate)

  conf_matr = confusionMatrix(model.pred ,data.partition$test$is_duplicate)

  precision <- cross.classify.table[2,2] / sum(cross.classify.table[,2])
  # Recall
  Recall <-  cross.classify.table[2,2] / sum(cross.classify.table[2,])
```

```
  accuracy             = mean(model.pred  == data.partition$test$is_duplicate )
  error                = mean(model.pred  != data.partition$test$is_duplicate )
  pred <- ROCR::prediction(model.probs, data.partition$test$is_duplicate)
  perf <- ROCR::performance(pred, 'tpr', 'fpr')
  perf_df <- data.frame(perf@x.values, perf@y.values)
  names(perf_df) <- c("fpr", "tpr")

  list( roc_data=perf_df,
        accuracy = accuracy,
        error= error,
        pred.prob=model.probs,
        precision = precision,
        Recall = Recall,
        conf.table = cross.classify.table,
        model = model.fits,
        conf_matr = conf_matr)

}
```

Build Ensemble Model

```
build_ensemble_model <- function(data.partition,
model.glm,model.lda,model.qda,model.rpart,model.naiveBayes,model.nnet,model.randomForest)
{
  model.probs <- (  model.nnet$pred.prob * model.nnet$accuracy+
                    model.randomForest$pred.prob * model.randomForest$accuracy) / (2 *
model.randomForest$accuracy)


  # model.probs <- (  model.glm$pred.prob *model.glm$accuracy  +  model.nnet$pred.prob *
model.nnet$accuracy+
  #                   model.randomForest$pred.prob * model.randomForest$accuracy) / (3 *
model.randomForest$accuracy)
  #


  # model.probs <- ( model.glm$pred.prob *model.glm$accuracy  + model.lda$pred.prob
*model.lda$accuracy  +
  #                   model.qda$pred.prob * model.qda$accuracy + model.rpart$pred.prob *
model.rpart$accuracy +
  #                model.naiveBayes$pred.prob * model.naiveBayes$accuracy   +
model.nnet$pred.prob * model.nnet$accuracy+
  #                  model.randomForest$pred.prob * model.randomForest$accuracy) / (7 *
model.randomForest$accuracy)
  model.pred = ifelse(model.probs > .5,1,0)


  conf_matr = confusionMatrix(model.pred ,data.partition$test$is_duplicate)

  precision <- cross.classify.table[2,2] / sum(cross.classify.table[,2])
  # Recall
  Recall <-  cross.classify.table[2,2] / sum(cross.classify.table[2,])


  # model.pred = model.pred %>% as.numeric()
  cross.classify.table = table(model.pred ,data.partition$test$is_duplicate)
  accuracy             = mean(model.pred  == data.partition$test$is_duplicate )
  error                = mean(model.pred  != data.partition$test$is_duplicate )
  pred <- ROCR::prediction(model.probs, data.partition$test$is_duplicate)
  perf <- ROCR::performance(pred, 'tpr', 'fpr')
  perf_df <- data.frame(perf@x.values, perf@y.values)
  names(perf df) <- c("fpr", "tpr")

  list( roc_data=perf_df,
        accuracy = accuracy,
        error= error,
        pred.prob=model.probs,
        precision = precision,
        Recall    = Recall,
        conf_matr = conf_matr )

}
```

Partition the data for cross validation

```
create_data_partition <- function(Data)
```

```
{
  set.seed(504023)
  trainIndex <- createDataPartition(Data$is_duplicate, p = .8, list = FALSE, times = 1)
  df_more_features_train_ <- Data[ trainIndex,]
  df_more_features_test_  <- Data[-trainIndex,]
  list( train = df_more_features_train_, test = df_more_features_test_)
}

data.partition <- create_data_partition(Data = df_more_features )
# Call the different models
data.partition %>% get_model_train.predict_metrics('glm') -> roc.data.glm
roc.data.lda   <- data.partition %>% get_model_train.predict_metrics('lda')
roc.data.qda   <- data.partition %>% get_model_train.predict_metrics('qda')
roc.data.rpart <- data.partition %>% get_model_train.predict_metrics('rpart')
roc.data.naivb <- data.partition %>% get_model_train.predict_metrics('naiveBayes')
roc.data.nnet  <- data.partition %>% get_model_train.predict_metrics('nnet')
roc.data.rfrst <- data.partition %>% get_model_train.predict_metrics('randomForest')
roc.data.ensemble <- data.partition %>% build_ensemble_model(roc.data.glm,
                                                             roc.data.lda,
                                                             roc.data.qda,
                                                             roc.data.rpart,
                                                             roc.data.rpart,
                                                             roc.data.nnet,
                                                             roc.data.rfrst)

roc.data.all.models = cbind(roc.data.glm$roc_data, model = 'glm')
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.lda$roc_data, model = 'lda'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.qda$roc_data, model = 'qda'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.rpart$roc_data, model =
'rpart'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.naivb$roc_data, model =
'naiveBayes'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.nnet$roc_data, model =
'nnet'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.rfrst$roc_data, model =
'randomForest'))
roc.data.all.models = rbind(roc.data.all.models, cbind(roc.data.ensemble$roc_data, model =
'ensemble'))
```

Plot ROC Curves for CV

```
roc.data.ensemble$accuracy


roc <- ggplot(data = roc.data.all.models, aes(x = fpr, y = tpr)) +
  geom_line(aes(color = model)) + geom_abline(intercept=0, slope=1, lty=3) +
  ylab('True Positive Rate') + xlab('False Positive Rate')

roc

model.accuracy.error = cbind( accuracy = roc.data.lda$accuracy, error = roc.data.lda$error,
model = 'lda')
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.qda$accuracy,error = roc.data.qda$error, model = 'qda'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.glm$accuracy,error = roc.data.glm$error, model = 'glm'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.rpart$accuracy,error= roc.data.rpart$error, model = 'rpart'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.naivb$accuracy,error= roc.data.naivb$error, model = 'naiveBayes'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy = roc.data.nnet$accuracy,
error = roc.data.nnet$error,model = 'nnet'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.rfrst$accuracy,error = roc.data.rfrst$error ,model = 'randomForest'))
model.accuracy.error = rbind( model.accuracy.error , cbind(accuracy =
roc.data.ensemble$accuracy,error = roc.data.ensemble$error ,model = 'ensemble'))


model.accuracy.error = model.accuracy.error %>% as.data.frame()

ggplot(data = model.accuracy.error , mapping = aes(x=model, y = accuracy  ) ) +
geom point(color = 'blue')
ggplot(data = model.accuracy.error , mapping = aes(x=model, y = error  ) ) + geom_point(color
= 'red')
```

K fold validation -----

```
k_fold = 5
get_ROC_cross_kfold_validation <- function(df_more_features,k_fold)
{
  model_metrics_kfold = data.frame()
  n <- nrow(df_more_features)
  Vf <- sample(n) %% k_fold + 1
  for (v in 1:k_fold) {
    df.train <- df more features[Vf != v, ]
    df.test  <- df_more_features[Vf == v, ]


    model_metrics_cv_nt =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'nnet')
    model metrics kfold = rbind(model metrics kfold,data.frame(roc data =
model_metrics_cv_nt$roc_data,
                                                               fold = v,
                                                               alg_model = 'nnet',
                                                               accuracy =
model_metrics_cv_nt$accuracy,
                                                               error    =
model_metrics_cv_nt$error,
                                                               Recall   =
model_metrics_cv_nt$Recall,
                                                               precision=
model_metrics_cv_nt$precision))


    model_metrics_cv_lm =  list(train = df.train, test = df.test) %>%
                get_model_train.predict_metrics(Model_Alg = 'glm')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_lm$roc_data,
                                                              fold = v,
                                                              alg_model = 'glm',
                                                              accuracy =
model_metrics_cv_lm$accuracy,
                                                              error    =
model_metrics_cv_lm$error,
                                                              Recall   =
model_metrics_cv_lm$Recall,
                                                              precision=
model_metrics_cv_lm$precision))

    model_metrics_cv_ld =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'lda')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_ld$roc_data,
                                                               fold = v,
                                                               alg_model = 'lda',
                                                               accuracy =
model_metrics_cv_ld$accuracy,
                                                               error    =
model_metrics_cv_ld$error,
                                                               Recall   =
model_metrics_cv_ld$Recall,
                                                               precision=
model_metrics_cv_ld$precision))


    model_metrics_cv_qd =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'qda')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_qd$roc_data,
                                                               fold = v,
                                                               alg model = 'qda',
                                                               accuracy =
model_metrics_cv_qd$accuracy,
                                                               error    =
model_metrics_cv_qd$error,
                                                               Recall   =
model_metrics_cv_qd$Recall,
                                                               precision=
model_metrics_cv_qd$precision))
```

```
    model_metrics_cv_rf =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'randomForest')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_rf$roc_data,
                                                               fold = v,
                                                               alg_model = 'randomForest',
                                                               accuracy =
model_metrics_cv_rf$accuracy,
                                                               error    =
model_metrics_cv_rf$error,
                                                               Recall   =
model_metrics_cv_rf$Recall,
                                                               precision=
model_metrics_cv_rf$precision))

    model_metrics_cv_rp =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'rpart')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_rp$roc_data,
                                                               fold = v,
                                                               alg model = 'rpart',
                                                               accuracy =
model_metrics_cv_rp$accuracy,
                                                               error    =
model_metrics_cv_rp$error,
                                                               Recall   =
model_metrics_cv_rp$Recall,
                                                               precision=
model_metrics_cv_rp$precision))



    model_metrics_cv_nv =  list(train = df.train, test = df.test) %>%
      get_model_train.predict_metrics(Model_Alg = 'naiveBayes')
    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_nv$roc_data,
                                                               fold = v,
                                                               alg_model = 'naiveBayes',
                                                               accuracy =
model_metrics_cv_nv$accuracy,
                                                               error    =
model_metrics_cv_nv$error,
                                                               Recall   =
model_metrics_cv_nv$Recall,
                                                               precision=
model_metrics_cv_nv$precision))

    model_metrics_cv_en = list(train = df.train, test = df.test) %>%
      build_ensemble_model( model.glm = model_metrics_cv_lm,
                            model.lda = model_metrics_cv_ld,
                            model.qda = model_metrics_cv_qd,
                            model.rpart = model_metrics_cv_rp,
                            model.naiveBayes = model_metrics_cv_nv,
                            model.nnet = model_metrics_cv_nt,
                            model.randomForest = model_metrics_cv_rf)

    model_metrics_kfold = rbind(model_metrics_kfold,data.frame(roc_data =
model_metrics_cv_en$roc_data,
                                                               fold = v,
                                                               alg_model = 'ensemble',
                                                               accuracy =
model_metrics_cv_en$accuracy,
                                                               error    =
model_metrics_cv_en$error,
                                                               Recall   =
model_metrics_cv_en$Recall,
                                                               precision=
model_metrics_cv_en$precision))


 }
  model_metrics_kfold
```

```
}

models_metrics_kfold <- df_more_features %>% get_ROC_cross_kfold_validation(5)
models_metrics_kfold_mean= models_metrics_kfold %>%
  group_by(roc_data.fpr,alg_model) %>%
   summarise( roc_data.tpr = mean(roc_data.tpr))

models_metrics_kfold_mean1= models_metrics_kfold %>%
  group_by(alg_model) %>%
  summarise( accuracy = mean(accuracy),
             error = mean(error),
             Recall = mean(Recall),
             precision = mean(precision) )
```

Plot ROC Curves for K Fold

```
ggplot(data = models_metrics_kfold_mean, aes( x=roc_data.fpr, y= roc_data.tpr)) +
  geom line(aes(color = alg model)) + geom abline(intercept=0, slope=1, lty=3) +
  ylab('True Positive Rate') + xlab('False Positive Rate')
```

Confusion matrix plot ----

```
library(ggplot2)
library(scales)

ggplotConfusionMatrix <- function(m,model_){
  mytitle <- paste( model_, ':',
                    "Accuracy", percent_format()(m$overall[1]),
                    "Kappa", percent_format()(m$overall[2]))
  p <-
    ggplot(data = as.data.frame(m$table) ,
           aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = log(Freq)), colour = "white") +
    scale_fill_gradient(low = "white", high = "steelblue") +
    geom_text(aes(x = Reference, y = Prediction, label = Freq)) +
    theme(legend.position = "none") +
    ggtitle(mytitle)
  return(p)
}
# Plot confusion matrixes
p1 = ggplotConfusionMatrix(roc.data.glm$conf_matr,'glm')
p2 = ggplotConfusionMatrix(roc.data.lda$conf_matr,'lda')
p3 = ggplotConfusionMatrix(roc.data.qda$conf_matr,'qda')
p4 = ggplotConfusionMatrix(roc.data.rpart$conf_matr,'rpart')
p5 = ggplotConfusionMatrix(roc.data.naivb$conf_matr,'Naive Bayes')
p6 = ggplotConfusionMatrix(roc.data.nnet$conf_matr,'nnet')
p7 = ggplotConfusionMatrix(roc.data.ensemble$conf_matr,'ensemble')
p8 = ggplotConfusionMatrix(roc.data.rfrst$conf_matr,'Random forest')
multiplot(p1,p2,p3,p4,p5,p6,p7,p8,cols = 4)
```

Deep Learning with Keras

Autoencoders

```
FLAGS <- flags(
  vocab_size = 94561,
  max_len_padding=  20,
  embedding_size =  100,
  regularization =  0.0001,
  seq_embedding_size = 512)
# Train Autoencoders and cross validation
train_autoencoder <- function(autoencoder,question ) {
  start = 0
  end   = 0
  nrow_df = nrow(question )
  pkg_size = 2000
  V = (nrow_df/pkg_size) %>% round() + 1

  histories = list()

  for( i in seq_len(10))
  {
    start = ( i - 1 ) * pkg_size + 1
    end = ifelse( end >= ( nrow_df - pkg_size ), nrow_df, (i * pkg_size))
```

```
    nnet_data_input         = question[start:end,]

    random_samples <- sample.int(pkg_size,size= 0.1 * pkg_size)

    nnet_data_input_train    = nnet_data_input[-random_samples,]
    nnet_data_input_valid    = nnet_data_input[random_samples,]

    # output_embedding <- model_embeding %>% predict(nnet_data_input)

    data_train <- autoencoder$embeding_model %>% predict(nnet_data_input_train)
    data_valid <- autoencoder$embeding_model %>% predict(nnet_data_input_valid)


    history = autoencoder$autoencoder_model %>%
      fit(
        data_train,
        data_train,
        batch_size = 1,
        epochs = 5,
        callbacks = list(
          callback_early_stopping(patience = 5),
          callback_reduce_lr_on_plateau(patience = 3)

        ),
        validation_data = list(data_valid,data_valid)
      )
    if( exists('history_prev') )
    {
      history_prev$params$epochs         = history_prev$params$epochs + history$params$epochs
      history_prev$metrics$val_acc       = cbind(history_prev$metrics$val_acc  ,
history$metrics$val_acc)
      history_prev$metrics$acc           = cbind(history_prev$metrics$acc      ,
history$metrics$acc)
      history_prev$metrics$lr            = cbind(history_prev$metrics$lr       ,
history$metrics$lr)
      history_prev$metrics$val_loss      = cbind(history_prev$metrics$val_loss ,
history$metrics$val_loss)
      history_prev$metrics$loss          = cbind(history_prev$metrics$loss     ,
history$metrics$loss)
    } else history_prev <- history

  }
  history_prev %>% plot()
}


# Model 1
autoencoder1 <- function(FLAGS) {
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding))

  # embedding model
  embedding <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  preds_embedding  <- sequence_input %>% embedding
  embeding_model   <- keras_model(sequence_input, preds_embedding)


  seq_emb1 <- layer_lstm(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  seq_emb2 <- layer_lstm(
    units =100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T  )
```

```
  dense_100_tanh1 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_100_tanh2 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_020_relu1 <- layer_dense( units = 20, activation = 'tanh')
  dense_020_relu2 <- layer_dense( units = 20, activation = 'tanh')

  # Encoder Model
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding,FLAGS$embedding_size))

  encoder <- sequence_input %>% seq_emb1 %>% dense_100_tanh1 %>%  dense_020_relu1
  encoder_model <- keras_model(sequence_input, encoder)

  # Decoder Model
  input_decoder <- layer_input(shape = c(20,20) )
  decoder <- input_decoder %>% dense_020_relu2 %>% dense_100_tanh2 %>% seq_emb2
  decoder_model <- keras_model(input_decoder , decoder)


  # Autoencoder Model
  autoencoder <- sequence_input %>% encoder_model %>%   decoder_model
  autoencoder_model <- keras_model(sequence_input , autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )
  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
       encoder_model=encoder_model,
       embeding_model = embeding_model)
}


# Model 2
autoencoder2 <- function(FLAGS) {
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding))

  # embedding model
  embedding <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  preds_embedding  <- sequence_input %>% embedding
  embeding_model   <- keras_model(sequence_input, preds_embedding)


  seq_emb1 <- layer_lstm(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  seq_emb2 <- layer_lstm(
    units =100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  dense_100_tanh1 <- layer_dense( units = 100, activation = 'relu')
  dense_100_tanh2 <- layer_dense( units = 100, activation = 'relu')
  dense_020_relu1 <- layer_dense( units = 20, activation = 'tanh')
  dense_020_relu2 <- layer_dense( units = 20, activation = 'tanh')
  dense_001_tanh1  <- layer_dense( units = 1 , activation = 'tanh')
  dense_001_tanh2  <- layer_dense( units = 1 , activation = 'tanh')

  # Encoder Model
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding,FLAGS$embedding_size))
  encoder <- sequence_input %>% seq_emb1 %>% dense_100_tanh1 %>%  dense_020_relu1 %>%
dense_001_tanh1
  encoder_model <- keras_model(sequence_input, encoder)
```

```
  # Decoder Model
  input_decoder <- layer_input(shape = c(20,1) )
  decoder <- input_decoder %>% dense_001_tanh2 %>% dense_020_relu2 %>% dense_100_tanh2  %>%
seq_emb2
  decoder_model <- keras_model(input_decoder , decoder)


  # v1 = embeding_model %>% predict(question1_[1:2,])
  # v1 %>% dim
  #
  # v2 = encoder_model %>% predict(v1)
  # v2 %>% dim
  #
  # v2 %>% dim()
  #
  # v3 = decoder_model %>% predict(v2)
  # v3 %>% dim()

  # Autoencoder Model
  autoencoder <- sequence_input %>% encoder_model %>%   decoder_model
  autoencoder model <- keras model(sequence input , autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )
  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
       encoder_model=encoder_model,
       embeding_model = embeding_model)
}


# Model 3
autoencoder3 <- function(FLAGS) {
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding))

  # embedding model
  embedding <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  preds_embedding  <- sequence_input %>% embedding
  embeding_model   <- keras_model(sequence_input, preds_embedding)


  seq_emb1 <- layer_lstm(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  seq_emb2 <- layer_lstm(
    units =100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T  )


  seq_gru1 <- layer_gru(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  seq_gru2 <- layer_gru(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
```

```
  )



  dense_100_tanh1 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_100_tanh2 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_020_relu1 <- layer_dense( units = 10, activation = 'tanh')
  dense_020_relu2 <- layer_dense( units = 10, activation = 'tanh')


  # Encoder Model
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding,FLAGS$embedding_size))
  encoder <- sequence_input %>% seq_gru1 %>% dense_100_tanh1 %>%  dense_020_relu1
  encoder_model <- keras_model(sequence_input, encoder)

  # Decoder Model
  input_decoder <- layer_input(shape = c(FLAGS$max_len_padding,10) )
  decoder <- input_decoder %>% dense_020_relu2 %>% dense_100_tanh2 %>% seq_gru2
  decoder_model <- keras_model(input_decoder , decoder)


  # Autoencoder Model
  autoencoder <- sequence input %>% encoder model  %>% decoder model
  autoencoder_model <- keras_model(sequence_input , autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )

  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
       encoder_model=encoder_model,
       embeding_model = embeding_model)
}



# Model 4
autoencoder4 <- function(FLAGS) {

  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding))

  # embedding model
  embedding <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  preds_embedding  <- sequence_input %>% embedding
  embeding_model   <- keras_model(sequence_input, preds_embedding)


  seq_emb1 <- layer_lstm(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  seq_emb2 <- layer_lstm(
    units =100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return sequences=T  )


  seq_gru <- layer_gru(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return sequences=T
  )
```

```
  conv1d_100_1         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )
  conv1d_100_2         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )
  max_pooling_1d       <- layer_max_pooling_1d(pool_size = 4)
  upsampling           <- layer_upsampling_1d( size = 4)

  dense_100_tanh1 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_100_tanh2 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_020_relu  <- layer_dense( units = 20, activation = 'tanh')


  # Encoder Model
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding,FLAGS$embedding_size))
  encoder        <- sequence_input %>% conv1d_100_1 %>% max_pooling_1d
  encoder_model  <- keras_model(sequence_input, encoder)

  # Decoder Model
  input_decoder <- layer_input(shape = c(5,100) )
  decoder <- input_decoder %>% upsampling %>% conv1d_100_2
  decoder_model <- keras_model(input_decoder , decoder)


  # Autoencoder Model
  autoencoder <- sequence_input %>% encoder_model %>%   decoder_model
  autoencoder_model <- sequence_input  %>% keras_model(autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )


  #
  #  v1 = embeding_model %>% predict(question1_[1:2,])
  #  v1 %>% dim
  #
  #  v2 = encoder_model %>% predict(v1)
  #  v2 %>% dim
  #
  #  v3 = decoder_model %>% predict(v2)
  #  v3 %>% dim()
  #
  # v4 = autoencoder_model %>% predict(v1)
  #
  # v4 %>% dim()

  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
       encoder_model=encoder_model,
       embeding_model = embeding_model)
}

# Model 5
autoencoder5 <- function(FLAGS) {

  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding))

  # embedding model
  embedding <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  preds_embedding  <- sequence_input %>% embedding
  embeding_model   <- keras_model(sequence_input, preds_embedding)


  seq emb1 <- layer lstm(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
```

```
    return_sequences=T
  )

  seq_emb2 <- layer_lstm(
    units =100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T  )


  seq_gru <- layer_gru(
    units = 100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    return_sequences=T
  )

  conv1d_100_1         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )
  conv1d_100_2         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )
  conv1d_100_3         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )
  conv1d_100_4         <- layer_conv_1d(filters = 100, kernel_size =  4, activation='relu',
padding='same' )

  max_pooling_1d     <- layer_max_pooling_1d(pool_size = 4)
  upsampling         <- layer_upsampling_1d( size = 4)



  max_pooling_1d_5     <- layer_max_pooling_1d(pool_size = 5)
  upsampling_5         <- layer_upsampling_1d( size = 5)


  dense_100_tanh1 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_100_tanh2 <- layer_dense( units = FLAGS$embedding_size, activation = 'relu')
  dense_020_relu  <- layer_dense( units = 20, activation = 'tanh')

  # Encoder Model
  sequence_input <- layer_input(shape = list(FLAGS$max_len_padding,FLAGS$embedding_size))
  encoder        <- sequence_input %>% conv1d_100_1 %>% max_pooling_1d_5 %>% conv1d_100_2 %>%
max_pooling_1d
  encoder_model  <- keras_model(sequence_input, encoder)

  # Decoder Model
  input_decoder <- layer_input(shape = c(1,100) )
  decoder <- input_decoder%>%    upsampling_5  %>% conv1d_100_3%>%    upsampling     %>%
conv1d_100_4
  decoder_model <- keras_model(input_decoder , decoder)


  # Autoencoder Model
  autoencoder <- sequence_input %>% encoder_model %>%   decoder_model
  autoencoder_model <- sequence_input  %>% keras_model(autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )



  #  v1 = embeding_model %>% predict(question1_[1:2,])
  #  v1 %>% dim
  #
  #  v2 = encoder_model %>% predict(v1)
  #  v2 %>% dim
  #
  #  v3 = decoder_model %>% predict(v2)
  #  v3 %>% dim()
  #
  # v4 = autoencoder_model %>% predict(v1)
  # v4 %>% dim()

  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
```

```
       encoder_model=encoder_model,
       embeding_model = embeding_model)
}


# Tests  Auto-encoders -----
autoencoder = FLAGS %>% autoencoder1()
question1_ = question1[,493:512]
question2_ = question2[,493:512]

FLAGS %>% autoencoder1()%>% train_autoencoder(question1_)
FLAGS %>% autoencoder2()%>% train_autoencoder(question1_)
FLAGS %>% autoencoder3()%>% train_autoencoder(question1_)
FLAGS %>% autoencoder4()%>% train_autoencoder(question1_)
FLAGS %>% autoencoder5()%>% train_autoencoder(question1_)
```

Autoencoder for word vectors ----

```
# Autoencoder for wrod vektor, and replacment of embedding layer

train_word_vec_model<- function(model)
{
  set.seed(1817328)
  val_sample <- sample.int(nrow(model$word_vectors_glove), size =
0.1*nrow(model$word_vectors_glove))

  history <- model$autoencoder_model  %>%
    fit(
      model$word_vectors_glove[-val_sample,],
      model$word_vectors_glove[-val_sample,],
      batch_size = 1000,
      epochs = 90,
      validation_data = list(
        model$word_vectors_glove[val_sample,], model$word_vectors_glove[val_sample,]
      ),
      callbacks = list(
       callback_early_stopping(patience = 5),
        callback_reduce_lr_on_plateau(patience = 3)
      )
    )

  list(history= history, model= model)

}


word_vect_autoencoder1 <- function(word_vectors_glove,config_params)
{
  dim_word_vec = word_vectors_glove %>% ncol
  sequence_input <- layer_input(shape = list( dim_word_vec ))


  dense_500_tanh1 <- layer_dense( units = 500, activation = 'tanh')
  dense_500_tanh2 <- layer_dense( units = 500, activation = 'tanh')

  dense_250_tanh1 <- layer_dense( units = 250, activation = 'tanh')
  dense_250_tanh2 <- layer_dense( units = 250, activation = 'tanh')

  dense_100_tanh1 <- layer_dense( units = dim_word_vec, activation = 'tanh')
  dense_100_tanh2 <- layer_dense( units = dim_word_vec, activation = 'tanh')

  dense_050_tanh1 <- layer_dense( units = 50, activation = 'tanh')
  dense_050_tanh2 <- layer_dense( units = 50, activation = 'tanh')


  dense_024_tanh1 <- layer_dense( units = 24, activation = 'tanh')
  dense_024_tanh2 <- layer_dense( units = 24, activation = 'tanh')

  dense 020 tanh1 <- layer dense( units = 20, activation = 'tanh')
  dense_020_tanh2 <- layer_dense( units = 20, activation = 'tanh')

  dense_012_tanh1 <- layer_dense( units = 12, activation = 'tanh')
```

```
  dense_012_tanh2 <- layer_dense( units = 12, activation = 'tanh')


  dense_010_tanh1 <- layer_dense( units = 10, activation = 'tanh')
  dense_010_tanh2 <- layer_dense( units = 10, activation = 'tanh')

  dense_005_tanh1 <- layer_dense( units = 5, activation = 'tanh')
  dense_005_tanh2 <- layer_dense( units = 5, activation = 'tanh')


  dense_004_tanh1 <- layer_dense( units = 4, activation = 'tanh')
  dense_004_tanh2 <- layer_dense( units = 4, activation = 'tanh')

  dense_002_tanh1 <- layer_dense( units = 2, activation = 'tanh')
  dense_002_tanh2 <- layer_dense( units = 2, activation = 'tanh')

  dense_001_tanh1 <- layer_dense( units = 1, activation = 'tanh')
  dense_001_tanh2 <- layer_dense( units = 1, activation = 'tanh')


  # Encoder Model
  encoder <- sequence_input %>%
    dense_100_tanh1 %>% dense_500_tanh1 %>% dense_250_tanh1 %>%
    dense_050_tanh1 %>% dense_024_tanh1 %>%
    dense_020_tanh1 %>% dense_012_tanh1 %>% dense_010_tanh1 %>%
    dense_005_tanh1 %>% dense_004_tanh1 %>% dense_001_tanh1
    encoder_model <- sequence_input %>% keras_model(encoder)
    # Decoder Model
  input_decoder <- layer_input(shape = 1 )
  decoder <- input_decoder %>%
    dense_001_tanh2 %>% dense_002_tanh2 %>% dense_004_tanh2 %>%
    dense_005_tanh2 %>% dense_010_tanh2 %>% dense_012_tanh2 %>%
    dense_020_tanh2 %>% dense_024_tanh2 %>% dense_050_tanh2 %>%
    dense_250_tanh2 %>% dense_500_tanh2 %>% dense_100_tanh2
  decoder_model <- input_decoder %>%  keras_model( decoder)
      # Autoencoder Model
  autoencoder <- sequence_input %>% encoder_model  %>% decoder_model
  autoencoder_model <- sequence_input %>%  keras_model(autoencoder)
  autoencoder_model %>%
    compile(
      optimizer='adadelta',
      loss='mse',
      metrics = c('accuracy')  )

  list(autoencoder_model=autoencoder_model,
       decoder_model=decoder_model,
       encoder_model=encoder_model,
       word_vectors_glove = word_vectors_glove)

   # v1 = encoder_model %>% predict(word_vectors_glove[1:2,1:50])
   # v1 %>% dim
   # v2 = decoder_model %>% predict(v1)
   # v2 %>% dim

  # v3 = autoencoder %>% predict(word_vectors_glove[1:2,])
  # v3 %>% dim
}

# one dimention word vector with encoder ----
d_tsne_1 %>% word_vect_autoencoder1() %>% train_word_vec_model()
autoencoder_hist <- word_vectors_glove %>% word_vect_autoencoder1() %>% train_word_vec_model()
plot(autoencoder_hist$history)
word_vec1d_encoder = autoencoder_hist$model$encoder_model %>% predict(word_vectors_glove )
word_vec1d_encoder <- word_vec1d_encoder %>% as.data.frame %>% mutate(term =
row.names(word_vectors_glove ))
colnames(word_vec1d_encoder ) = c('value','term')
serach similar words 1d <- function(Word ) {
  sorted_word_vecs <- word_vec1d_encoder %>% arrange( desc(value))
  word_indx = which(sorted_word_vecs$term == Word_)%>% as.numeric()

  sorted_word_vecs[(word_indx -10):(word_indx+ 10 ), ]
}
serach similar words 1d('france') %>% View()
which(duplicated(word_vec1d_encoder$term))
```

Classifiers

```
# Classification Model Definition ------------------------------------------------------

model1 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  embedding1 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  embedding2 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )


  seq emb <- layer lstm(
    units = FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq gru1 <- layer gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq embedding size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)

  vector1 <- input1 %>% embedding1() %>%
    seq_gru1()
  vector2 <- input2 %>% embedding2() %>%
    seq_gru2()

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- layer_dot(list(vector1, vector2), axes = 1) %>% dense_100_relu %>%
    layer_dropout( rate = 0.13)%>%
    layer_dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )

  model
}


model2 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  dense_020_relu1     <- layer_dense( units = 20, activation = 'relu')
  dense_020_relu2     <- layer_dense( units = 20, activation = 'relu')

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)
```

```
  embedding1 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  embedding2 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )


  seq_emb <- layer_lstm(
    units = FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru1 <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)

  seq_gru2 <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)


  vector1 <- input1 %>% embedding1() %>%
    seq_gru1() %>%
    dense_020_relu1 %>%
    layer_dropout( rate = 0.1)

  vector2 <- input2 %>% embedding2() %>%
    seq_gru2() %>%
    dense_020_relu2 %>%
    layer_dropout( rate = 0.1)


  out <- list(vector1, vector2) %>% layer_dot( axes = 1) %>%
    layer_dense(1, activation = "sigmoid")

  model <- list(input1, input2) %>% keras_model( out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )

  model
}

model3 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  dense_020_relu1     <- layer_dense( units = 20, activation = 'relu')
  dense_020_relu2     <- layer_dense( units = 20, activation = 'relu')

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)

  embedding1 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )
```

```
  embedding2 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )


  seq_emb <- layer_lstm(
    units = FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru1 <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)

  seq_gru2 <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.12)


  vector1 <- input1 %>%
    embedding1() %>%
    seq_gru1() %>%
    dense_020_relu1 %>%
    layer_dropout( rate = 0.13)

  vector2 <-input2 %>%
    embedding2() %>%
    seq_gru2() %>%
    dense_020_relu2 %>%
    layer_dropout( rate = 0.13)


  out <- layer_concatenate(list(vector1, vector2)) %>%
    layer_dense(1, activation = "sigmoid")

  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )

  model
}


model4 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)

  embedding1 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input length = FLAGS$max len padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  embedding2 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output dim = FLAGS$embedding size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
```

```
  )

  seq_emb <- layer_lstm(
    units = FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.15)

  vector1 <- input1 %>% embedding1()
  vector2 <- input2 %>% embedding2()


  out <- layer_dot(list(vector1, vector2),axes = 1) %>% seq_gru %>%  dense_100_relu %>%
layer_dropout(rate = 0.12) %>%
    layer_dense(1, activation = "sigmoid")

  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )


  model
}

model5 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)

  embedding1 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  embedding2 <- layer_embedding(
    input_dim = FLAGS$vocab_size + 2,
    output_dim = FLAGS$embedding_size,
    input_length = FLAGS$max_len_padding,
    embeddings_regularizer = regularizer_l2(l = FLAGS$regularization)
  )


  seq_emb <- layer_lstm(
    units = FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru <- layer_gru(
    units =  100,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.15)

  vector1 <- input1 %>% embedding1()
  vector2 <- input2 %>% embedding2()


  out <- list(vector1, vector2) %>% layer_dot(axes = 1) %>% seq_gru %>%
    layer dense(1, activation = "sigmoid")

  model <- list(input1, input2) %>% keras_model( out)
```

```
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )


  model
}

model6 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))


  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)
  drop_out_0.1       <- layer_dropout(rate = 0.1)



  embedding1 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  embedding2 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )


  seq_emb1 <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.15
  )

  seq_emb2 <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.15
  )


  seq_gru <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1 %>% embedding1() %>%
    seq_emb1()
  vector2 <- input2 %>% embedding2() %>%
    seq_emb2()

  out <- list(vector1, vector2) %>% layer_dot( axes = 1) %>%
    layer_dense(1, activation = "sigmoid")

  model <- list(input1, input2) %>% keras_model( out)


  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
```

```
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  # v1 = model %>% predict(list(question1_[1:2,],question2_[1:2,]))
  # v1 %>% dim
  model
}


model7 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))


  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)
  drop_out_0.1       <- layer_dropout(rate = 0.1)



  embedding1 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )


  embedding2 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  seq_emb <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru1 <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.1)
  seq_gru2 <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.1)

  vector1 <- input1 %>% embedding1() %>%
    seq_gru1()
  vector2 <- input2 %>% embedding2() %>%
    seq_gru2()

  out <- list(vector1, vector2) %>% layer_concatenate() %>% dense_020_relu %>% drop_out_0.1
%>%
    layer_dense(1, activation = "sigmoid")

  model <- list(input1, input2) %>% keras_model( out)


  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric binary accuracy
    )
  )
```

```
  # v1 = model %>% predict(list(question1_[1:2,],question2_[1:2,]))
  # v1 %>% dim
  model
}
model8 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))


  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)
  drop_out_0.1       <- layer_dropout(rate = 0.1)



  embedding1 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input length = FLAGS$max len padding,
    trainable = FALSE
  )

  embedding2 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  seq_emb <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1 %>%  embedding1()
  vector2 <- input2 %>%  embedding2()

  out <- list(vector1, vector2) %>% layer_dot( axes = 1) %>% seq_gru %>%
    layer_dense(1, activation = "sigmoid")

  model <- keras_model(list(input1, input2), out)


  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  # v1 = model %>% predict(list(question1_[1:2,],question2_[1:2,]))
  # v1 %>% dim
  model
}

model9 <- function(FLAGS) {
  input1 <- layer input(shape = c(FLAGS$max len padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))


  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense 001 softmax  <- layer dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten()
  drop_out_0.1       <- layer_dropout(rate = 0.3)
```

```
  conv2d_out_0.1     <- layer_conv_1d(filters = 128, kernel_size =  4, activation='relu',
padding='same', input_shape = c(100,100) )
  max_pooling_2d     <- layer_max_pooling_1d(pool_size = 2)


  embedding1 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  embedding2 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  seq_emb <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization)
  )

  seq_gru <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1 %>% embedding1()
  vector2 <- input2 %>% embedding2()

  out <- list(vector1, vector2) %>% layer_dot( axes = 1) %>% conv2d_out_0.1 %>%
max_pooling_2d %>% drop_out_0.1 %>%  flatten_layer_ %>%
    layer_dense(1, activation = "sigmoid")

  model <- keras_model(list(input1, input2), out)


  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  # v1 = model %>% predict(list(question1_[1:2,],question2_[1:2,]))
  # v1 %>% dim
  model
}


model10 <- function(FLAGS) {
  input1 <- layer_input(shape = c(FLAGS$max_len_padding))
  input2 <- layer_input(shape = c(FLAGS$max_len_padding))

  drop_out_0.1       <- layer_dropout(rate = 0.1)
  dense_020_relu     <- layer_dense( units = 20, activation = 'relu')
  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")

  flatten_layer_1     <- layer_flatten()
  conv1d_128_1    <- layer_conv_1d(filters = 128, kernel_size =  4, activation='relu',
padding='same', input_shape = 128 )
  max_pooling_1d1     <- layer_max_pooling_1d(pool_size = 2)

  flatten_layer_2     <- layer_flatten()
  conv1d_128_2    <- layer_conv_1d(filters = 128, kernel_size =  4, activation='relu',
padding='same', input_shape = 128 )
  max_pooling_1d2     <- layer_max_pooling_1d(pool_size = 2)
```

```
  embedding1 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )

  embedding2 <- layer_embedding(
    input_dim = word_vectors_glove %>% nrow() ,
    # regularizer_l2(l = FLAGS$regularization)
    output_dim = FLAGS$embedding_size,
    weights = list(word_vectors_glove),
    input_length = FLAGS$max_len_padding,
    trainable = FALSE
  )


  seq_emb <- layer_lstm(
    units = FLAGS$embedding_size,
    recurrent regularizer = regularizer l2(l = FLAGS$regularization)
  )

  seq_gru <- layer_gru(
    units =  FLAGS$embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1 %>% embedding1() %>% conv1d_128_1() %>% max_pooling_1d1 %>%
flatten_layer_1 %>% layer_dropout(rate = 0.15)
  vector2 <- input2 %>% embedding2() %>% conv1d_128_2() %>% max_pooling_1d2 %>%
flatten_layer_2 %>% layer_dropout(rate = 0.15)

  out <- list(vector1, vector2) %>% layer_concatenate( )  %>%
    layer_dense(1, activation = "sigmoid")

  model <- keras_model(list(input1, input2), out)


  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  # v1 = model %>% predict(list(question1_[1:2,],question2_[1:2,]))
  # v1 %>% dim
  model
}

model <- model1(FLAGS)
print(model)
model <- model2(FLAGS)
model <- model3(FLAGS) # 1
model <- model4(FLAGS)
model <- model5(FLAGS)
model <- model6(FLAGS) ## 1
model <- model7(FLAGS)
model <- model8(FLAGS)
model <- model9(FLAGS) ##
model <- model10(FLAGS) ##

# Model Fitting ------------------------------------------------------------

train model <- function(model,question1 ,question2 )
{
  set.seed(1817328)
  val_sample <- sample.int(nrow(question1_), size = 0.1*nrow(question1_))

 history <- model %>%
   keras:: fit(
      list(question1_[-val_sample,], question2_[-val_sample,]),
      df$is_duplicate[-val_sample],
```

```
      batch_size = 128,
      epochs = 30,
      validation_data = list(
        list(question1_[val_sample,], question2_[val_sample,]), df$is_duplicate[val_sample]
      ),
      callbacks = list(
        callback_early_stopping(patience = 5),
        callback_reduce_lr_on_plateau(patience = 3)
      )
    )
 prediction_ = list(question1_[val_sample,], question2_[val_sample,]) %>% predict(model,.)


 list(history= history, model = model, validation = data.frame(prediction = prediction_ ,
val_sample = val_sample ))


}

model_hist <- FLAGS %>% model10() %>% train_model(question1_,question2_)
model_hist$model %>% save_model_hdf5('model10.hdf5',overwrite = TRUE,include_optimizer = TRUE)
saveRDS(model_hist$history,file='histmodel10.rds')
histmodel10 = readRDS('histmodel10.rds')
plot(histmodel10)
```

Classifier with 1 Dimension word vector ----

```
#  one decimal value word with R-TSNE ---
library(Rtsne)
get_word_vec_1d <- function(word_vectors_glove)
{
  tsne_model_1 = word_vectors_glove[-c(94561:94562),]  %>%  as.matrix( ) %>%  Rtsne(. ,
check_duplicates=F, pca=TRUE, perplexity=3, theta=0.5, dims=1)
  d_tsne_1d     = data.frame(value = tsne_model_1$Y, term =row.names(word_vectors_glove[-
c(94561:94562),])  )
  colnames(d_tsne_1d) = c('value','term')
  d_tsne_1d$term = as.character(d_tsne_1d$term)
  # row.names(d_tsne_1d) =  row.names(word_vectors_glove[-c(94561:94562),])
  save(d_tsne_1d,file='d_tsne_1d.RData')
  save(tsne_model_1, file = 'tsne_model_1.RData')
  d_tsne_1d$value = scale(d_tsne_1d$value )/max(d_tsne_1d$value )
  d_tsne_1d
}

word_vectors_glove1D <- word_vectors_glove[-c(94561:94562),]  %>% get_word_vec_1d()


get_sentence_word_sequence_1d <- function(Sentence,word_vec_1d)
{
  v_q <- Sentence %>% as.character() %>% tibble(v=.) %>% select(v)   %>% unnest_tokens(word,v)
  v <- v_q$word %>%   sapply(function(x){ which( word_vec_1d$term == x)[[1]] } ) %>% na.omit
  names(v) = c()
  v
}
df[1,]$question1 %>% get_sentence_word_sequence_1d(word_vec_1d)


prepare questions for keras 1d <- function(Questions,word vec 1d)
{
  n = word_vec_1d %>% nrow()
  v <- Questions %>% sapply(FUN = get_sentence_word_sequence_1d,word_vec_1d) %>% unname()
  v <- v %>% pad_sequences( maxlen = 30, value = n )
  dimv <- v %>% dim
  v   <- matrix(v %>% word_vec_1d[.,'value'] ,dimv[1],dimv[2])
  v
}

question1_1d = df$question1 %>% prepare_questions_for_keras_1d(word_vec_1d)
question2_1d = df$question2 %>% prepare_questions_for_keras_1d(word_vec_1d)


covq1q2 = question1_1d %>%  apply(1,cov(.,question2_1d))

save(question2_1d ,file='question2_1d.RData')
```

```
save(question1_1d ,file='question1_1d.RData')
```

```
plot_gama_ddensity <- function(numb = 1000000) {
  x <- rgamma(numb, shape = 3, scale = 0.2)
  den <- density(x)
  dat <- data.frame(x = den$x, y = den$y)
  # Plot density as points
  ggplot(data = dat, aes(x = x, y = y)) +
    geom_point(size = 3) +
    theme_classic()
}

plot_gama_ddensity(numb = 50)
```

```
model1d_1 <- function(FLAGS,question1,question2)
{
  input1 <- layer_input(shape = c(30))
  input2 <- layer_input(shape = c(30))

  seq_reshape1 <- layer_reshape(target_shape = c(1,30))
  seq_reshape2 <- layer_reshape(target_shape = c(1,30))

  seq_gru1 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1  %>% seq_reshape1 %>%
    seq_gru1()
  vector2 <- input2  %>% seq_reshape2 %>%
    seq_gru2()

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- layer_dot(list(vector1, vector2), axes = 1) %>% dense_100_relu %>%
    layer_dropout( rate = 0.1)%>%
    layer_dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  model

  # v1 = model %>% predict(list(question1_1d[1:2,],question2_1d[1:2,]))
  # v1 %>% dim

}

model1d_2 <- function(FLAGS,question1,question2)
{
  input1 <- layer_input(shape = c(30))
  input2 <- layer_input(shape = c(30))

  seq reshape1 <- layer reshape(target shape = c(30,1))
  seq_reshape2 <- layer_reshape(target_shape = c(30,1))
```

```
  seq_gru1 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1  %>% seq_reshape1 %>%
    seq_gru1()
  vector2 <- input2  %>% seq_reshape2 %>%
    seq_gru2()

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- layer_dot(list(vector1, vector2), axes = 1) %>% dense_100_relu %>%
    layer_dropout( rate = 0.1)%>%
    layer_dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  model

  # v1 = model %>% predict(list(question1_1d[1:2,],question2_1d[1:2,]))
  # v1 %>% dim

}



model1d_3 <- function(FLAGS,question1,question2)
{
  input1 <- layer_input(shape = c(30))
  input2 <- layer_input(shape = c(30))

  seq_reshape1 <- layer_reshape(target_shape = c(1,30))
  seq_reshape2 <- layer_reshape(target_shape = c(1,30))

  seq_gru1 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.4)

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.4)

  vector1 <- input1  %>% seq_reshape1 %>%
    seq_gru1()
  vector2 <- input2  %>% seq_reshape2 %>%
    seq_gru2()

  dense_100_relu     <- layer_dense( units = 100, activation = 'relu')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- list(vector1, vector2) %>% layer_concatenate( ) %>% dense_100_relu %>%
    layer_dropout( rate = 0.1)%>%
    layer_dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
```

```
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  model

  # v1 = model %>% predict(list(question1_1d[1:2,],question2_1d[1:2,]))
  # v1 %>% dim

}


model1d_4 <- function(FLAGS,question1,question2)
{
  input1 <- layer_input(shape = c(30))
  input2 <- layer_input(shape = c(30))

  seq_reshape1 <- layer_reshape(target_shape = c(1,30))
  seq_reshape2 <- layer_reshape(target_shape = c(1,30))

  seq gru1 <- layer gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.2)

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization),
    dropout = 0.2)

  vector1 <- input1  %>% seq_reshape1 %>%
    seq_gru1()
  vector2 <- input2  %>% seq_reshape2 %>%
    seq_gru2()

  dense_100_relu1     <- layer_dense( units = 100, activation = 'tanh')
  dense_100_relu2     <- layer_dense( units = 100, activation = 'relu')
  dense_100_relu3     <- layer_dense( units = 100, activation = 'tanh')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- list(vector1, vector2) %>% layer_concatenate( ) %>% dense_100_relu1 %>%
    layer_dropout( rate = 0.2) %>% dense_100_relu2 %>% layer_dropout( rate = 0.2) %>%
dense_100_relu3 %>%
    layer_dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  model

  # v1 = model %>% predict(list(question1_1d[1:2,],question2_1d[1:2,]))
  # v1 %>% dim

}


model1d 5 <- function(FLAGS,question1,question2)
{
  input1 <- layer_input(shape = c(30))
  input2 <- layer_input(shape = c(30))

  seq_reshape1 <- layer_reshape(target_shape = c(30,1))
  seq reshape2 <- layer reshape(target shape = c(30,1))

  seq_gru1 <- layer_gru(
```

```
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  seq_gru2 <- layer_gru(
    units =  FLAGS$seq_embedding_size,
    recurrent_regularizer = regularizer_l2(l = FLAGS$regularization))

  vector1 <- input1  %>% seq_reshape1 %>%
    seq_gru1()
  vector2 <- input2  %>% seq_reshape2 %>%
    seq_gru2()

  dense_100_relu1     <- layer_dense( units = 100, activation = 'tanh')
  dense_100_relu2     <- layer_dense( units = 100, activation = 'relu')
  dense_100_relu3     <- layer_dense( units = 100, activation = 'tanh')
  dense_001_softmax  <- layer_dense(units = 1 , activation = "sigmoid")
  flatten_layer_     <- layer_flatten(input_shape = 512)


  out <- layer_dot(list(vector1, vector2), axes = 1) %>% dense_100_relu1 %>%
    layer_dropout( rate = 0.1) %>% dense_100_relu2 %>% layer_dropout( rate = 0.1) %>%
dense_100_relu3 %>%
    layer dense(1, activation = "sigmoid")


  model <- keras_model(list(input1, input2), out)
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = list(
      acc = metric_binary_accuracy
    )
  )
  model

  # v1 = model %>% predict(list(question1_1d[1:2,],question2_1d[1:2,]))
  # v1 %>% dim

}



question1_1d[1,] %>% length()

model <- FLAGS %>% model1d_4(question1_1d,question2_1d) %>%
train_model(question1_1d,question2_1d)

x <- c(2,8,11,19)
x <- data.frame(x,1) ## 1 is your "height"
plot(seq(1:30),question1_1d[2453,], type='l')
plot(seq(1:30),question2_1d[2453,], type='l')



plot_questions_1d_project <- function(idx) {
  q1d_graph_data = data.frame(word_value= question1_1d[idx,16:30],sequence= seq(15),field =
'question1')
  q1d_graph_data = q1d_graph_data %>% rbind(data.frame(word_value=
question2_1d[idx,16:30],sequence= seq(15),field = 'question2'))
  p= q1d_graph_data %>% ggplot( aes(x=sequence, y=word_value, colour=field)) + geom_line()
  p
  # q1d_graph_data
}

v1 = which(df$is_duplicate == 0 )[100:111] %>% lapply(plot_questions_1d_project)
multiplot(v1[[1]],v1[[2]],v1[[3]],v1[[4]],v1[[5]],v1[[5]],v1[[6]],v1[[7]],v1[[8]],v1[[9]],v1[[
10]],v1[[11]],cols=3)



serach_similar_words_1d <- function(Word_) {
  sorted_word_vecs <- word_vec_1d %>% arrange( desc(value))
  word indx = which(sorted word vecs$term == Word )%>% as.numeric()

  sorted_word_vecs[(word_indx -10):(word_indx+ 10 ), ]
```

```
}

serach_similar_words_1d('safety') %>% View()

heatmap(question1_1d[idx,16:30], question2_1d[idx,16:30])
```

Compare the models

```
# Compare keras models with ROC curves -----
set.seed(3645789)
n <- nrow(question1_)
test_idx <- sample.int(n, size = round(0.2 * n))
question1_train <- question1_[-test_idx, ]
question1_test <- question1_[test_idx, ]
question2_train <- question2_[-test_idx, ]
question2_test <- question2_[test_idx, ]

model10 <- load_model_hdf5('model10.hdf5')
model7 <- load_model_hdf5('model7.hdf5')
model8 <- load_model_hdf5('model8.hdf5')
model9 <- load_model_hdf5('model9.hdf5')

get_roc_data_for_keras_model <- function( model, model_name) {
  predction1 = list(question1_test,question2_test) %>% predict(model,.)
  pred <- ROCR::prediction(predction1, df$is_duplicate[test_idx])
  perf <- ROCR::performance(pred, 'tpr', 'fpr')
  perf_df <- data.frame(perf@x.values, perf@y.values,model_name = model_name)
  names(perf_df) <- c("fpr", "tpr","model_name")
  perf_df
}

get_roc_data_for_keras_model <- function( validation, model_name) {

  pred <- ROCR::prediction(validation$prediction, df$is_duplicate[validation$val_sample])
  perf <- ROCR::performance(pred, 'tpr', 'fpr')
  perf_df <- data.frame(perf@x.values, perf@y.values,model_name = model_name)
  names(perf_df) <- c("fpr", "tpr","model_name")
  perf_df
}


roc_keras_models <- function( ) {

 load('validation10.RData')
 load('validation9.RData')
 load('validation8.RData')
 load('validation7.RData')
 load('validation6.RData')
 load('validation5.RData')
 load('validation4.RData')
 load('validation3.RData')
 load('validation2.RData')
 load('validation1.RData')



 roc10  = get_roc_data_for_keras_model(validation10,'model10')
 roc9  = get_roc_data_for_keras_model(validation_data9,'model9')
 roc8  = get_roc_data_for_keras_model(validation_data8,'model8')
 roc7  = get_roc_data_for_keras_model(validation_data7,'model7')
 roc6  = get_roc_data_for_keras_model(validation_data6,'model6')
 roc5  = get_roc_data_for_keras_model(validation_data5,'model5')
 roc4  = get_roc_data_for_keras_model(validation_data4,'model4')
 roc3  = get_roc_data_for_keras_model(validation_data3,'model3')
 roc2  = get_roc_data_for_keras_model(validation_data2,'model2')
 roc1  = get_roc_data_for_keras_model(validation_data1,'model1')


 roc_data = rbind(roc1,roc2,roc3,roc4,roc5,roc6,roc7,roc8,roc9,roc10)

 roc <- ggplot(data = roc_data, aes(x = fpr, y = tpr)) +
    geom_line(aes(color=model_name)) + geom_abline(intercept=0, slope=1, lty=3) +
    ylab(perf@y.name) + xlab(perf@x.name)
  roc
}
```

```
roc_keras_models()


multiplot(readRDS('histmodel1.rds') %>% plot(),readRDS('histmodel2.rds') %>%
plot(),readRDS('histmodel3.rds') %>% plot(),
          readRDS('histmodel4.rds') %>% plot(),readRDS('histmodel5.rds') %>%
plot(),readRDS('histmodel6.rds') %>% plot(),
          readRDS('histmodel7.rds') %>% plot(),readRDS('histmodel8.rds') %>%
plot(),readRDS('histmodel9.rds') %>% plot(),
          readRDS('histmodel10.rds') %>% plot()  ,cols=4)
```