Identifying Duplicate Questions on Quora

Joanna Olson joannaolson@umass
Jakob Herlitz jherlitz@umass
Sanuj Bhatia sanujbhatia@umass
Parth Goel pgoel@umass

1 Problem statement

Our goal was to predict whether two separate questions are asking the same thing. For instance, if we have the questions "What color is the sky?" and "Is the sky blue or pink?", the answer to both is the same. We aimed to correctly classify when two questions have the same intent and, therefore, share a common answer.

This way, for things like online forums, we can combine these two questions onto one page, instead of two separate pages with potentially differently worded answers. It can help to clean up websites, make it easier for readers to find a single page with answers theyre looking for, and prevent inconsistent answers. If we look at Stack Overflow, for example, there are often questions that are marked as duplicate questions, and the link to the original question is supplied. With the ability to classify questions as duplicates or not, we will avoid having these two separate pages for questions seeking the same answer. Now, users will not waste time on two different pages, and, instead, the information theyre looking for is concentrated into one area. Also, it saves space by combining these two pages into one.

2 What you proposed vs. what you accomplished

Throughout the process, we were able to stay on track with our goals and accomplish almost all of what we set out to do at the beginning of the project.

1. ~~Research current solutions and get an understanding of how they work (Nov. 1)~~
2. ~~Explore data to see trends and preprocessing tasks (Nov. 6)~~
3. ~~Develop and test baseline algorithm (Nov. 8)~~
4. ~~Begin implementation of selected algorithms from research (Nov. 12)~~
5. ~~Finalize progress report (Nov. 16)~~
6. ~~Finalize implementation of both approaches and start testing (Nov. 25)~~
7. ~~Evaluate and compare models to find best solution (Dec. 1)~~
8. ~~Complete poster (Dec. 6)~~
9. *Continue to improve models as much as possible (Dec. 20)*: Hard to say this is ever finished! A strong CPU to handle more question pairs and more time to train for more epochs would result in improvements to the Siamese LSTM model.
10. ~~Complete final report (Dec. 20)~~

3 Related work

Measuring semantic text similarity has been a topic of research for many years. Methods for this challenge have utilized a variety of features, like word overlap and similarity, sentence or phrase composition, and many learning algorithms, like SVM, k-NN, and ensembles. Bjerva et al. (2014) use formal semantics and logical inference, with word2vec features, which we also use in our model. Other approaches have found sparse features useful, like syntax based classifiers, e.g., modeling divergence of dependency syntax between the two sentences (Das and Smith, 2009). Collobert and Weston (2008) used convolutional neural networks, where their model is trained jointly for multiple NLP tasks with shared weights, learning paraphrase relations between sentence and word pairs. Socher et al. (2011) used a recursive neural network to model each sentence, recursively computing the representation for the sentence from the representations of its constituents. In our mode, we use a much simpler sentence representation using a single layer LSTM. He et al. (2015) use detailed architectural engineering on a ConvNet variant, which infers sentence similarity by integrating various differences across many convolutions. Due to the limited availability of labeled data, a fine tuned model was necessary for good performance, but we dont face this issue for our project. Tai et al. (2015) propose Tree-LSTMs using tree-structured network topologies. The hope is that the tree structured network can propagate necessary information more efficiently than a sequentially restricted architecture.

Mueller and Thyagarajan (2016) is our chosen approach. The authors propose a Manhattan LSTM model (MaLSTM) which uses the final hidden state of the network as the sentence representation. A similarity score is assigned based on the Manhattan distance.

Zhang et al. (2014) focuses on our other main approach to the problem, topic modeling using Latent Dirichlet Allocation (LDA). We elected to follow this paper and implement LDA because researchers in the past have struggled with properly assigning topics to questions. Duan et al. (2008) aims to solve a very similar problem to our problem, but lacks the implementation of LDA.

Blei et al. (2003) gives a good background on LDA as a generative, probabilistic topic model. The goal of the model is to generate a topic space in which we will be able to compare questions. Topics, in this case, will be groupings of words all relating to some common theme (usually with a combination of semantic and syntactic similarity). The end goal is that we will be able to compare questions by measuring their similarity in topic space. This is done by using an unsupervised clustering algorithm to generate clusters of questions using some carefully chosen similarity measures. These similarity measures take into account things like the Levenshtein distance factor (minimum number of edit operations necessary to transform one string into another), a part of speech factor, and a bag of words factor (the bag of words feature is the sole feature used in our baseline algorithm).

Implementing this model has proven to be fairly difficult, with topics being slightly too vague to achieve the desired performance. Looking at the literature in the field, it appears that the LSTM is a more reliable approach for what we are trying to accomplish.

4 Your dataset

Our data was taken from a Kaggle competition. The competition can be found here. Our basic data contains five items per row: id (identification of the question pair, int64), qid1 (question 1 identification, int64), qid2 (question 2 identification, int64), question1 (the string of question 1, object), question2 (the string of question 2, object), and is_duplicate (where 0 means the questions are not duplicates and 1 means they are, int64). Below is are examples of rows in the dataset:

id = 7
qid1 = 15
qid2 = 16
question1 = How can I be a good geologist?
question2 = What should I do to be a great geologist?
is_duplicate = 1

id = 26
qid1 = 53
qid2 = 54
question1 = What is web application?
question2 = What is the web application framework?
is_duplicate = 0

Our data was given to us split into a train.csv and test.csv. The train.csv contains 404290 rows of data, and our predicted label, is_duplicate, is provided. The test.csv contains 3563475 rows of data, but it does not provide the applicable is_duplicate label. Since our data is from a Kaggle competition, we are supposed to train our models on train.csv, and then run our models with test.csv and submit the results. Once submitted, we would receive our accuracy results. Before we test on test.csv, we have split train.csv into a train and validation dataset. Therefore, we can see accuracies of different models before testing them on test.csv.

A potential problem with the dataset is that questions can vary a lot in length. Figure 2 shows a box and whisker plot of the variance of the lengths of questions 1 and 2. As we can see, it is centered roughly around 10 words per question for questions 1 and 2. However, we see there are a lot of outliers. There are even questions that have a length of one word. Figure 3 shows the average difference in length between sentences. Looking at the box, we can see that, for the most part, questions vary in length by around 1 to 5 words. Yet there are still instances where questions can vary in length by over 50 words.

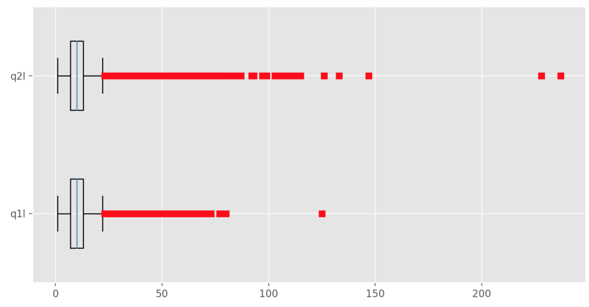


Figure 1: Average number of words in each questions in the dataset.

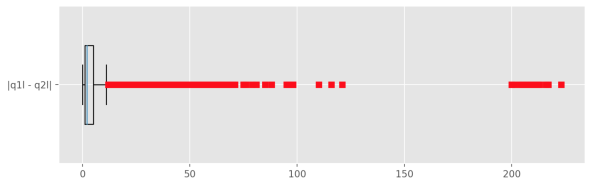


Figure 2: Average difference in number of words in Question 1 and Question 2.

During our exploration of the data, we looked for patterns and features that showed distinctions between the two classes: duplicate and non-duplicate. We confirmed a strong correlation between the ratio of repeated words in the questions and whether or not the questions were duplicates. Figure 4 shows a histogram of this ratio. We will discuss this ratio more in depth in “Baselines”, as we ended up using this feature for our baseline algorithm. We also found a relationship between the difference in length of the two questions and whether or not the questions are duplicates. Figure 5 shows a box and whisker plot of the average difference in length of the two questions for the two different classes. As we can see, the non-duplicates, on average, have a greater difference in length.

4.1 Data preprocessing

For the baseline model, we did not do any special preprocessing - only the sequence of raw tokens retrieved by splitting the sentences on spaces were used as input.

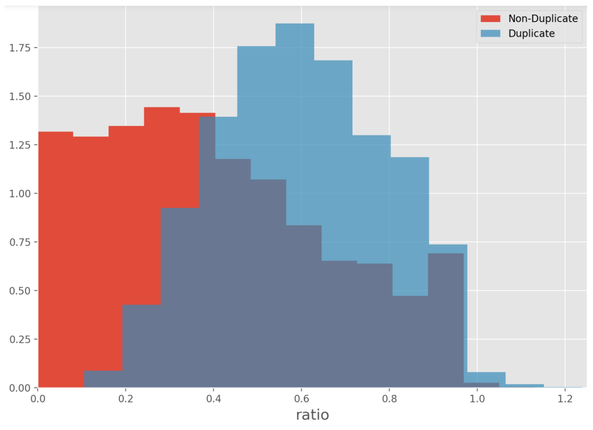


Figure 3: Histogram representation of the feature vector: ratio of repeated words compared to the lengths of the questions.

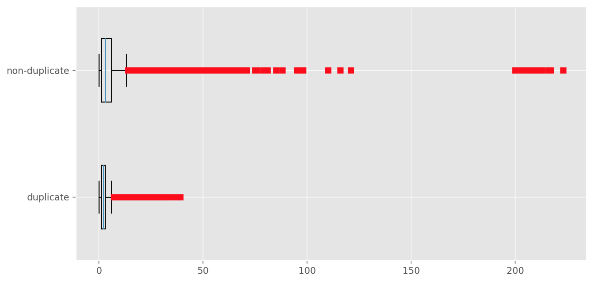


Figure 4: Average difference in length between two the questions by class (duplicate and non-duplicate).

When preparing the data for input to the LSTM, however, some preprocessing steps were taken.

1. First, all non-alphanumeric characters were removed from the sentences. This strips the sentence of punctuation - it would interfere with finding token matches in Word2Vec, and is cleaner to not deal with incorrect/repeated punctuation, something that can lead to worse predictions.

2. All characters lower-cased and then tokenized. Again, this leads to more matches in Word2vec, but sometimes when capitalized words represent named entities, that information is lost.

3. Some common stop words were removed, including most that are not found in the Word2Vec vocabulary. We noticed an improvement in the loss on our test set when removing stop words.

4. In order to get more matches in Word2Vec,

tokens were searched with upper case and title case variations, with some success at finding missing tokens with all lower case characters.

For the LDA Model, In order to implement the approach, from our training data, we created data frames using the Pandas library by importing the data- which was already available to us by the Kaggle competition and already split into train and test sets.

We cleaned the data which involved tokenizing each question pair in the data frame to a list of words, removing stop-words such 'the', 'a', 'for' which tend to duplicate a lot in sentences and hence will skew our results in determining similarity, stemming the words so that all forms of a word have one 'root word', and lemmatization of the data. We traversed through our training set and created a list of lists consisting of all the question pairs.

5 Baselines

For our baseline algorithm, we used a basic decision tree classifier to predict whether or not a question pair contained duplicates. We engineered our own features from the given dataset as well. We parsed the two questions and counted the repeated words in each question. This count of words was labeled as "repeats". We also tracked the length of each question: "q1l" and "q2l" for the length of question 1 and question 2, respectively. From there, we used the results of computation on these features to decide whether or not we would classify a pair of questions as duplicates or not. We compared the number of repeated words in a ratio with the length of the questions and used this number as our only feature. The words are only counted as repeated if they are the same exact word. For example, "dance" and "dances" would not be the same word.

Figure 6 shows a confusion matrix of the best results, pertaining to about a 67% accuracy with a binary threshold of about 0.57 (this was determined by running every possible binary threshold between 0 and 1, incrementing by 0.01). As evident in the figure, the baseline algorithm is better at correctly identifying duplicate questions than non-duplicate questions. It is more likely to predict a duplicate question than a non-duplicate question. Figure 7 shows a graph of the precision and recall of the model as the binary threshold changes. The value of the binary threshold at the crossing point of the precision and recall was used to pick the best threshold value to run the baseline algorithm on.

For splitting our dataset, we used scikit-learns function train_test_split and contributed a third of our data in train.csv to validation and the rest to training.

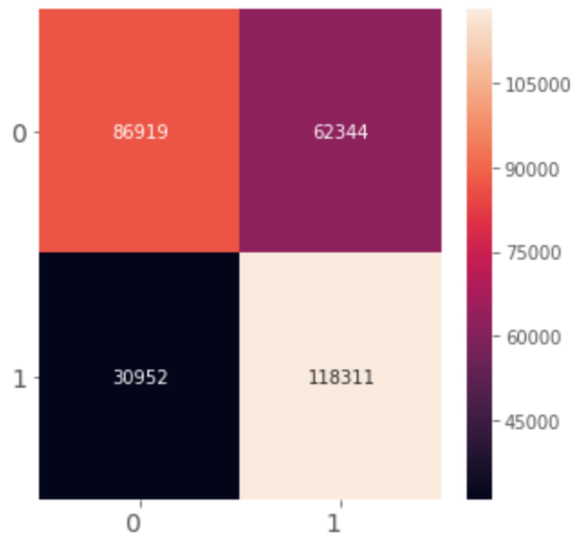


Figure 5: Confusion matrix results of the baseline decision tree classification algorithm with a binary threshold of 0.57.

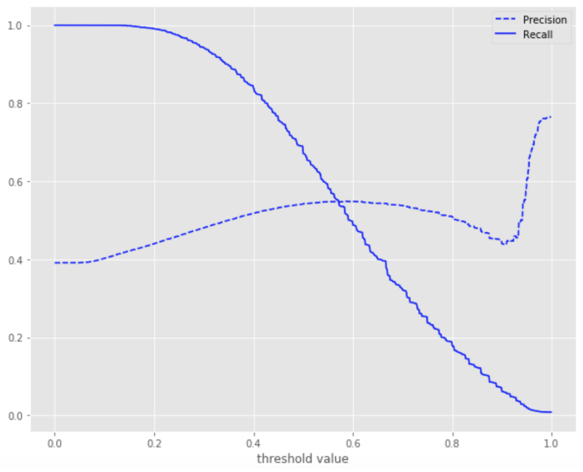
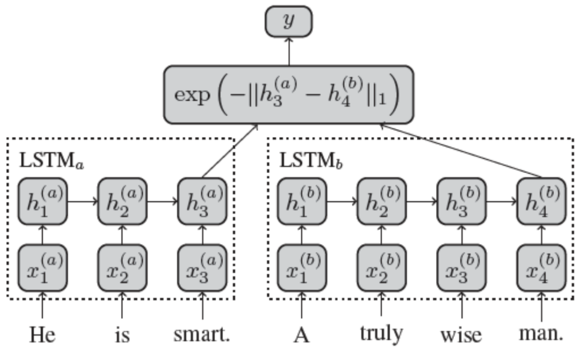


Figure 6: Plot of precision and recall numbers of the baseline as binary threshold changes.

6 Your approach

We tried two main approaches, one using topic modelling, as well as a recurrent neural network that we trained on this task. The two approaches are described in detail below:

SIAMESE LSTM: This is our more successful, RNN approach, where we implement the model described by Mueller and Thyagarajan (2016). The authors propose a Manhattan LSTM model (MaLSTM) with two single-layer LSTMs with tied weights that each process one of the sentences in the pair. The final hidden state is employed as a vector representation of the sentence. Subsequently, the similarity between these representations is used as a predictor of semantic similarity. Figure 8 shows the architecture we use.



y
$\exp\left(-\|h_3^{(a)} - h_4^{(b)}\|_1\right)$
LSTM$_a$
LSTM$_b$
$h_1^{(a)}$ $h_2^{(a)}$ $h_3^{(a)}$
$x_1^{(a)}$ $x_2^{(a)}$ $x_3^{(a)}$
$h_1^{(b)}$ $h_2^{(b)}$ $h_3^{(b)}$ $h_4^{(b)}$
$x_1^{(b)}$ $x_2^{(b)}$ $x_3^{(b)}$ $x_4^{(b)}$
He is smart.
A truly wise man.

Figure 7: The high-level architecture of the siamesenetwork with two LSTMs in it.

The LSTM learns a mapping from the space of variable length sequences (sequence of tokens from each question), where each token uses a 300-dimensional pre-trained Word2Vec word embedding as input to the LSTM [1], to the space of 50-dimensional hidden states. Concretely, each sentence is passed to the LSTM, token-by-token, updating its hidden state at each sequence index, at the end of which we obtain the final hidden state as the sentence representation. This LSTM network functions like an encoder, and the only error signal backpropagating during training is calculated from the similarity between the question pair representations, and how much this predicted similarity deviates from the ground truth label. For two sentences A and B, a simple similarity function is used, defined as $g(A, B) = exp(-\|h_A - h_B\|_1) \in [0, 1]$. This is the exponentiation of the Manhattan distance between the final hidden states, h_A and h_B, obtained for sentences A and B respectively. The reasoning provided in the paper for the choice of the similarity function indicates that it would force the LSTM to capture the semantic differences between question pairs in its entirety during training itself, instead of having to supplement the RNN with a more complex model to resolve its shortcomings. Using the l_2 rather than the l_1 norm can lead to undesirable plateaus in the objective function, said to occur due to vanishing gradients of the Euclidian distance.

This model is expected to capture the intent of the questions and predict similarity, irrespective of length and word similarity differences. Therefore, we do not expect it to fail for the same reasons as our baseline. Error analysis and comparison is provided in the next section.

We were able to implement the architecture and used PyTorch to build and train the LSTM model. For this approach, we only used the first 100k question pairs from the training data, using a 80-20 train-test split. We use a single LSTM to encode each of the sentences in a pair, randomly re-initializing its hidden state after every sentence (zero-initialization and random initialization did not produce any obvious differences during training). Our Siamese LSTM class takes a mini-batch (of size 64 question pairs in the final model) as input and, after encoding each of them using the LSTM, returns the tensor of 64 predicted similarities for that batch. We use binary cross entropy loss, and the Adadelta optimizer. We also use gradient clipping after each backpropagation step, clipping to a value of 0.25 to avoid the exploding gradients problem.

We implemented this model from scratch in Pytorch, taking hints from the Keras implementation for the same problem, linked here. This implementation uses a Siamese architecture in Keras, along with some interesting modifications. First, it uses custom preprocessing rules (Kaggle community contributed rules, developed while the contest was live), some of which we picked up as well. It also extends the training set, copying a (q1, q2, label) tuple as a new (q2, q1, label), to avoid any bias due to symmetry.

To train the network, we used the first 80k data points with randomized mini-batches of size 64. We observed more than a 1.2% increase in accuracy when randomizing batches across epochs, with each epoch first randomly shuffling the 80k pairs, and then generating batches using a generator function. We deployed a VM instance on the Compute Engine in the Google Cloud Platform, with a 6 core CPU and 52 GB of memory (a free trial did not let us use GPUs on it). Training the final model took around 3 hours per epoch, and

[1]Googles Word2Vec, which we obtained here.

here we present results after 15 epochs of training. We took a model snapshot after every 5 epochs. Some issues we faced are detailed here, before we present the results:

- Using more data led to a huge increase in memory usage, even though we used generators for batch creation. Initially, with a 26 GB memory size, we were unable to run more than 50k questions pairs for training, but increasing the batch size helped with that. We also used gc.collect() after every epoch, further reducing memory usage. We settled on a size of 80k to balance the time per epoch and memory usage.

- Each epoch seems to be taking more time (upto 10% increase in time) than the previous one. By the 15th epoch, the time was up to 4.5 hours/epoch. This is highly unreasonable and something we can look into for future work on this project.

- We found a lot of the tokens in our raw vocabulary missing from Word2Vec. Many of these are proper nouns and named entities, but also include spelling errors and around 5% are numbers. Ignoring these missing words, around 20k of them, our vocabulary size is around 85k. This effectively puts an upper limit on the performance we can achieve, since we lose valuable information in ignoring those tokens. This is discussed in more detail in the error analysis and future works section.

Tested on 20k question pairs, we achieved an accuracy of 76.4%, and a negative loss of 0.5. We obtained this accuracy by binarizing the predictions at a variable threshold, and selected the best threshold of 0.419, giving us the best accuracy.

Figure 9 plots the F1 and accuracy scores for varying threshold values from 0 to 1 in steps of 0.01. The dotted line represents the threshold we use for the final predictions, with a 76.4% accuracy and 0.67 F1-score. It is interesting to note that at a binary threshold of 0 and 1, around 38% and 62% accuracy is seen. This means that classifying all examples as non-duplicate gives us a 62% accuracy, and so 62% of the test set consists of non-duplicate data points, and 38% duplicate data points.

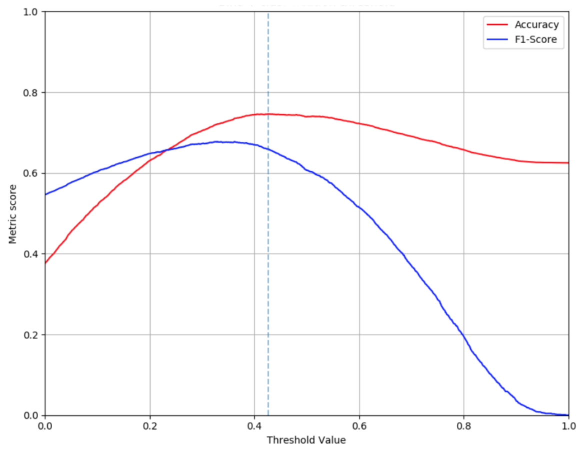


Figure 8: Accuracy and F1 score of Siamese LSTM model. Dotted line indicates most accurate binary threshold.

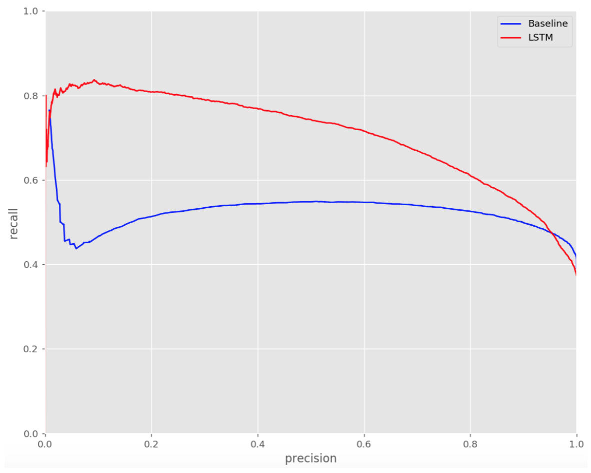


Figure 9: Graph of precision-recall curve over varying binary thresholds for our LSTM and baseline model.

Figure 10 is a precision recall curve over varying binary thresholds for our LSTM and baseline model. As we can see, the precision and recall is higher for the LSTM in general, and we have a much better model for similarity prediction.

LDA: A Topic clustering model using Latent Dirichlet Allocation to develop a topic space consisting of topic vectors and clusters from which we can semantically relate the similarity of the question pairs.

Latent Dirichlet allocation (LDA) is a topic model that generates topics based on word frequency from a set of documents. LDA is particularly useful for finding reasonably accurate mixtures of topics within a given document set. Figure 11 shows a graphical representation of the model.

Where w is a word, z is a topic, Θ is a document, and Φ is the topic distribution matrix. N is

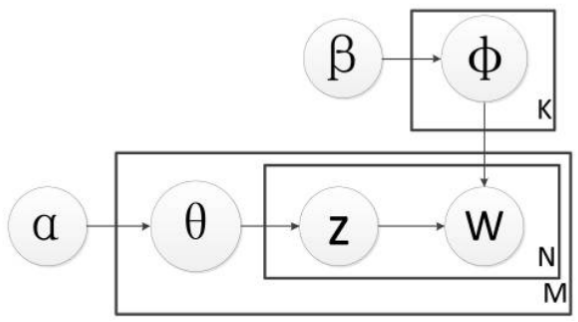


Figure 10: Graphical Representation of LDA architecture.

the number of words in the document, M is the number of documents in the corpus which in our case is question pairs, and K is the number of topics we specify.

LDA has 2 hyperparameters: α and β.

α - A low value for α means that documents have only a low number of topics contributing to them. A high value of α yields the inverse, meaning the documents appear more alike within a corpus.

β - A low value for β means the topics have a low number of contributing words. A high value of β yields the inverse, meaning topics will have word overlap and appear more alike.

To implement this we had to import several libraries and packages:

- NLTK - Used to preprocess the data
- Tom_lib - To identify the optimal number of topics
- Gensim - To create the LDA model
- Pandas - To create and manage the data frame
- Matplot & Bokeh - To display and plot the topic models

Involves:

1. Preprocessing: Removing stop words and stemming.
2. LDA modeling: Transferring the question repository into the corresponding topic vectors.
3. Topic guided clustering: Based on the factors, we utilize an unsupervised machine learning approach to clustering the questions into several clusters.
4. Similar question filtering: Selecting and reranking the similar questions for each of the clusters

We converted our document-term matrix (training set) to a bag-of-words (bow) which is a list of vectors equal to the number of question pairs and generated our LDA model using this bow by passing in our parameters (passes, no. of topics, etc.).

A sample corpus that we generated from our tokenized and preprocessed data is shown below:

[['univers', 'expand', 'expand', 'univers', 'expand'],
['will', 'us', 'perform', 'regim', 'chang', 'philippin', 'restor', 'freedom', 'democraci', 'someon', 'eagerli', 'pursuit', 'freedom', 'want', 'lead', 'chang', 'societi', 'can', 'assum', 'person', 'brainwash', 'freedom', 'democraci'],
['best', 'decis', 'ever', 'made', 'life', 'best', 'decis', 'ever', 'made', 'life']]

We then constructed a document-term matrix using gensim by assigning a unique integer id to each unique token while also collecting word counts and relevant statistics. We converted this dictionary to a bag-of-words(bow) which is a list of vectors equal to the number of question pairs:

[[(0, 3), (1, 2)],
[(2, 1), (3, 1), (4, 1), (5, 2), (6, 2), (7, 1), (8, 3), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1), (20, 1)],
[(21, 2), (22, 2), (23, 2), (24, 2), (25, 2)],
[(26, 2), (27, 2), (28, 1), (29, 2), (30, 1), (31, 1), (32, 1), (33, 1), (34, 1), (35, 2), (36, 1), (37, 2), (38, 1), (39, 1)]

We generated our LDA model using this bow by passing in our parameters (passes, no. of topics, etc.) and printed out the resulting topic model and most popular words per topic for the data set.

The model initializes by assigning every word in every document to a random topic. Then, we iterate through each word, unassign it's current topic, decrement the topic count corpus wide and reassign the word to a new topic based on the local probability of topic assignments to the current document, and the global (corpus wide) probability of the word assignments to the current topic. An example of the topic-ids, along with their words and their respective probabilities regarding that topic is shown in Figures 12 and 13.

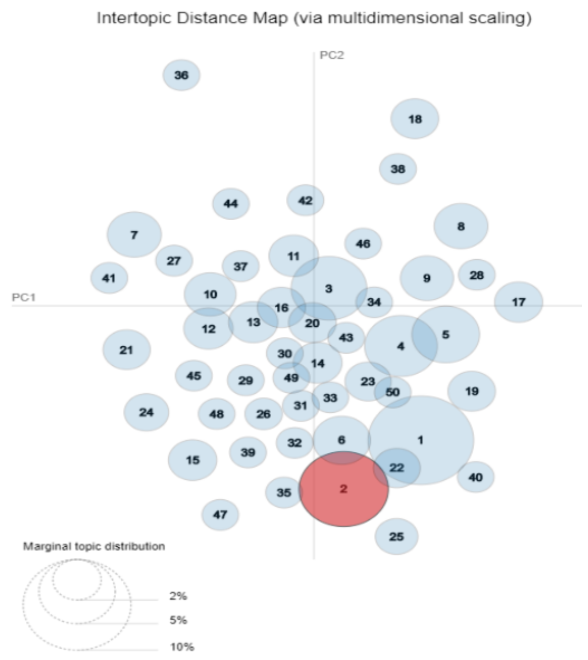


Figure 11: Topic IDs and their respective probabilities

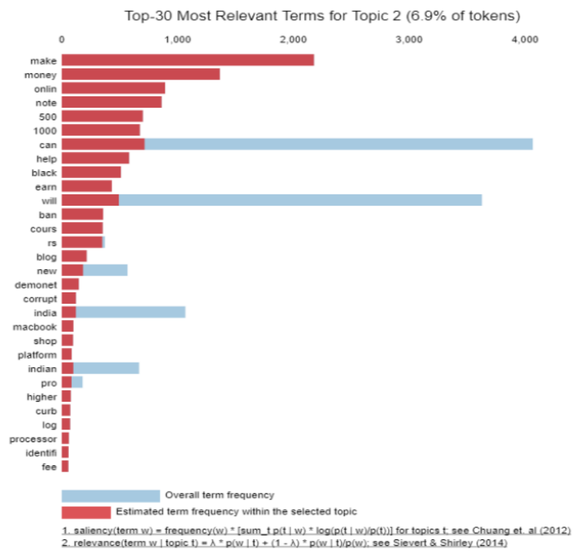


Figure 12: Given a Topic ID, the probabilities of specific words showing up. Expected and observed.

We then used Cosine similarity to calculate the angle between the topic vectors of the respective question pairs and Manhattan distance to calculate the difference between the probability distributions of the topics as our similarity metrics to infer the relationship between the questions.

Some sample values of the Manhattan function along with the respective question pair ids are given below:

	manhattan_dist
392740 →	7.053975
311070 →	8.000953
79083 →	0.009293
74107 →	0.000000
265216 →	3.057778
400748 →	2.398000
55102 →	6.113925
48529 →	3.163333
86102 →	0.003333
79235 →	0.100000
373945 →	4.292444
203213 →	0.000000
236647 →	0.000000
68166 →	3.016706
352792 →	1.470000

We calculated our accuracy based on these similarity measures. Initially, the performance was barely above 50% but after tweaking the thresholds for both the mesures and retraining the LDA model with a more optimal number of topics and increased passes, the accuracy improved to 62% for the Cosine metric and 64% for the Manhattan metric which is a vast increase and considering the paper this approach is based on, also is reaching peak potential (State of the art being around 66-68%). Figure 14 shows a confusion matrix of the Manhattan metric model.

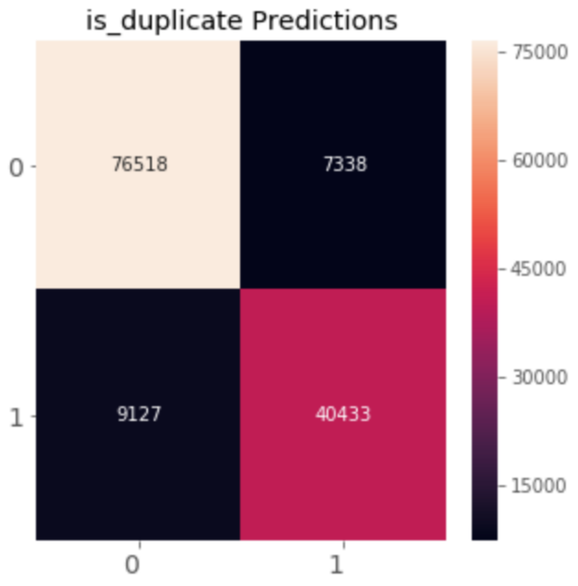


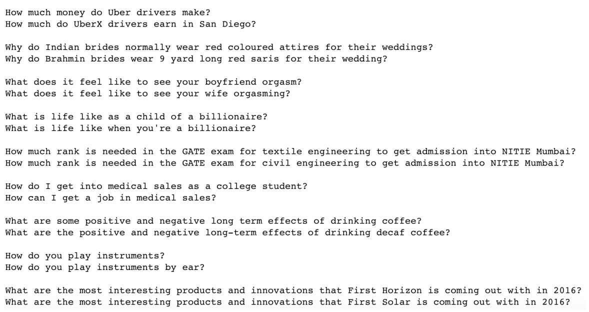
Figure 13: Confusion matrix results of the LDA classification algorithm.

However, this is still performing poorly in relation to our baseline algorithm and our other neural network solution and hence, have to conclude that despite its positives, a topic clustering model is perhaps not to the best approach to tackling this problem.

7 Error analysis

Our two main approaches, plus the baseline algorithm, are analyzed below:

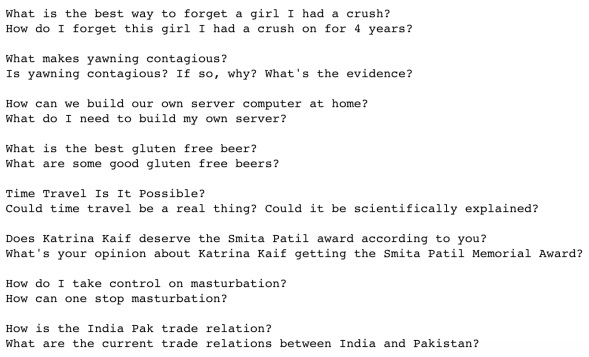
BASELINE: The baseline is more likely to falsely predict a duplicate question pair for multiple scenarios. For instance, when two questions are worded similarly, but ask about different things, the model will predict they are asking the same question. An example of this is when the two questions were “How would you compare the United States’ euthanasia laws to Netherlands?” and “How would you compare the United States’ euthanasia laws to Denmark?”. Clearly, they are not the same question, but they share 10 of their 11 words. The baseline algorithm only looks for similarity based on the number of shared words, so it sees these two questions as the same. When two questions share very similar syntactic commonalities, they are likely to be classified as the duplicate questions. Figure 15 shows some question pair instances where the baseline had false positives. As you can see, many of the questions vary by just one or two words, but these words are key to deciding what exactly it is that the question is asking.



```
How much money do Uber drivers make?
How much do UberX drivers earn in San Diego?

Why do Indian brides normally wear red coloured attires for their weddings?
Why do Brahmin brides wear 9 yard long red saris for their wedding?

What does it feel like to see your boyfriend orgasm?
What does it feel like to see your wife orgasming?

What is life like as a child of a billionaire?
What is life like when you're a billionaire?

How much rank is needed in the GATE exam for textile engineering to get admission into NITIE Mumbai?
How much rank is needed in the GATE exam for civil engineering to get admission into NITIE Mumbai?

How do I get into medical sales as a college student?
How can I get a job in medical sales?

What are some positive and negative long term effects of drinking coffee?
What are the positive and negative long-term effects of drinking decaf coffee?

How do you play instruments?
How do you play instruments by ear?

What are the most interesting products and innovations that First Horizon is coming out with in 2016?
What are the most interesting products and innovations that First Solar is coming out with in 2016?
```

Figure 14: Instances where the baseline model incorrectly predicted that two questions were duplicates (FALSE POSITIVES).

An example of an instance where the baseline incorrectly classified two questions as non-duplicates was these two questions: “How did the Big Bang occur?” and “What caused the Big Bang?”. The statements “What caused x” and “How did x occur” are evidently asking the same thing, but they do not share any words besides “x”. Therefore, the baseline fails to see these as duplicate questions. It does not understand any type of grammar or sentence structure, so it fails to see the correlation between the two questions. Figure 16 shows some question pair instances where the baseline had false negatives.



```
What is the best way to forget a girl I had a crush?
How do I forget this girl I had a crush on for 4 years?

What makes yawning contagious?
Is yawning contagious? If so, why? What's the evidence?

How can we build our own server computer at home?
What do I need to build my own server?

What is the best gluten free beer?
What are some good gluten free beers?

Time Travel Is It Possible?
Could time travel be a real thing? Could it be scientifically explained?

Does Katrina Kaif deserve the Smita Patil award according to you?
What's your opinion about Katrina Kaif getting the Smita Patil Memorial Award?

How do I take control on masturbation?
How can one stop masturbation?

How is the India Pak trade relation?
What are the current trade relations between India and Pakistan?
```

Figure 15: Instances where the baseline model incorrectly predicted that two questions were not duplicates (FALSE NEGATIVES).

In general, the baseline algorithm struggles when the two questions vary in length. If one question has only a view words and the other has 10+, then the algorithm struggles. There are instances in which this difference results in both false positives and false negatives. For example, the question pair (“A car travels 40 kilometers at an average speed of 80 km/h and then travels 40 kilometers at an average speed of 40 km/h. What is the average speed of the car for this 80 km trip?”, “What is the average bike speed?”) resulted in a false positive, and the question pair (“Are atheists more accepting of homosexuality than normal people because they do not follow the Bible, which says it is a sin?”, “Do atheists tend to accept homosexuality more than religious people?”) resulted in a false negative. Since we are only looking at the repeated words between questions, this difference in length makes our main feature a lot less accurate. Longer text usually requires a deeper understand of the context and subject of the question, which is not a capability of our baseline algorithm.

SIAMESE LSTM: The LSTM predictions seem to fail in some very interesting examples. Consider the two questions below, which have a target label of being duplicate.

1: How do I join army after hotel management?
2: How do I join IAF after MBBS?

These sentences, after preprocessing, do not lose any words to the stop words list or due to Word2Vec - theyre represented exactly as show. But, the LSTM predicts these are not being duplicates, leading to a false negative. I think I would agree with this prediction, and the label could as easily have indicated the pair to be non-duplicate. Other such interesting false negatives and false positives are shown below, and they are some of the more ambiguous sentences, where human-provided labels could be wrong, and the LSTM ones more correct. This shows the power of the neural network model.

False Positives (that are actually positives):

- 1: What is the tastiest Pizza in Dominos?
  2: What is the best Pizza in Dominos?

- 1: From where can I start to improve my GK?
  2: How can I improve my GK?

False Negatives (that are actually negatives):

- 1: How good is Great Lakes Gurgaon PGPM?
  2: How is PGPM course of Great Lakes Chennai?

- 1: Is Python good to start learning programming?
  2: How should I start learning Python for Data Science?

Next, we present some incorrectly predicted examples where the exclusion of some tokens from the processed sentence might have been the cause. These include changes in the way words are capitalized, and proper nouns that might be missing from Word2Vec:

False Positives:

- 1: What is instafinancials.com?
  1 (preprocessed): com
  2: What is screenedrenters.com?
  2 (preprocessed): com

- 1: How much does Snapchat pay a new grad software engineer?
  1 (preprocessed): How much does pay new grad software engineer
  2: How much does Twitter pay a new grad software engineer
  2 (preprocessed): How much does Twitter pay a new grad software engineer

False Negatives:

- 1: When will Zomato launch in Germany?
  1 (preprocessed): When will launch in Germany
  2: When will Zomato be launched in Germany?
  2 (preprocessed): When will be launched in Germany

- 1: When will Xiaomi Redmi Note 3 be back on stocks?
  1 (preprocessed): When will Note 3 be back on stocks
  2: When will Xiaomi Redmi Note 3 be Back in Stocks?
  2 (preprocessed): When will Note 3 be Back in Stocks

Note how, in the last example, the word Stocks from Word2Vec could mean financial stocks, and could have changed the meaning of the sentence completely, when looking at the word embeddings.

Lastly, there are some cases where the LSTM obviously fails due to insufficient training:

False Positives:

- 1: Why should I not study?
  2: Why should I study?

- 1: How can I convert Whatsapps text messages into text?
  2: How can I change my voice to text?

False Negatives:

- 1: How much does it cost to manufacture a Tesla car?
  2: How much does the production of a Tesla car cost?

- 1: Which steps should I start a Cafe Business?
  2: How do I start a cafe?

LDA: Although some topics had good connections and semantic relationship between the words, a lot of the groups were quite vague and hence the accuracy of this must be improved. We can see the distinction of topics in the question pairs and their respective frequencies in those topics.

Some inputs it failed in were ones in which a lot of the words were similar or almost exactly the same having one or two words being different. For example, “How do I invest in bitcoin?” and “Should I invest in bitcoin?” are in terms of their meaning semantically different and would have completely different answers but were registered as duplicates in our LDA model.

Other examples include “How effective is Krav Maga for self-defence as compared to other martial arts?” and “Which martial art is better in a real-world scenario?” with both questions having a different answer and question words, yet was marked as duplicate; “How exactly does banning the Rs. 500 and Rs. 1000 note curb the black market?” and “Would banning notes of denominations 500 and 1000 help curb the black market?” however being very similar questions are actually duplicates but were marked as non-duplicate perhaps because of the additional common words present in there which either were not stopped or stemmed out during pre-processing or the topic for one the words ’500’ is very different from ’Rs. 500’ hence resulting in the divergence.

A reason for why these errors crop up is because of the weights that are attached to each of the topics. Since they are based on frequency, some of the words may develop a very high weights when compared to other common words in the same sentence, then affecting the value of the ’document’ (question pair) as seen above with ’martial arts’.

Also since we can only assign one topic to each word some contextual meaning between two words may go missing such as between synonyms which the model may treat as completely different as seen with ’500’ and ’Rs. 500’.

8 Contributions of group members

Below is a list of the contributions of each member of the group. Everyone contributed to the information in the various reports and the poster.

- Joanna Olson: responsible for data exploration, creating visualization of results and data, generating baseline algorithm and results, putting together the various reports, and designing the poster
- Jakob Herlitz: responsible for researching various papers, deciding an approach, and assisting with the implementation of the Siamese LSTM and LDA models
- Sanuj Bhatia: responsible for researching various papers, deciding an approach, and implementing the Siamese LSTM model
- Parth Goel: responsible for researching various papers, deciding an approach, and implementing the LDA model

9 Conclusion

It was really interesting to be a part of a project like this from start to finish, experiencing all the steps of the process, from thinking about potential challenges to work on, reading research papers to gain insights on how it could be done, selecting some reasonable approaches and planning our work in a group of 4.

This project really gave us a working knowledge of how to set up an NLP project, complete with data preprocessing, building models, training and testing them, and also evaluating them properly with visualizations and plots. We realised that data preprocessing is extremely crucial to any machine learning task. It proved surprisingly difficult to tune network parameters of our LSTM, simply due to each train-test cycle requiring an upward of 2 days to complete satisfactorily. We ended with much of the default parameters unchanged, which actually worked pretty well. Again, the LSTM surprised us some of the examples, as shown previously, where it clearly used the meaning of the words through word embeddings and, in essence, gave better similarity predictions than humans.

For potential future work, we could add more features to our LSTM, including the topic models we generated, as well as incorporate the missing words from Word2Vec in the model. This can be done either by randomly initializing and training the missing word embeddings while the network is trained, or by at least noting if the words being removed from the two sentences are similar (maybe even character level similarity) or entirely distinct. Apart from this, we would want to run our model on a GPU, along with training on 400k data, for much more epochs than just 15. We think that putting in attention mechanisms should also provide benefits.

References

Bjerva, J., Bos, J., van der Goot, R., and Nissim, M. (2014). The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. *SemEval*.

Blei, D., Ng, A., and Jordan, M. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res. volume 3*, pages 993–1002.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. *In Proceedings of the 25th International Conference on Machine learning*, pages 160–167.

Das, D. and Smith, N. A. (2009). Paraphrase identification as probabilistic quasi-synchronous recognition. *In Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 468–476.

Duan, H., Cao, Y., Lin, C., and Yu, Y. (2008). Searching questions by identifying question topic and question focus. *Proceedings of the 46rd Annual Meeting on Association for Computational Linguistics*, pages 156–164.

He, H., Gimpel, K., and Lin, J. (2015). Multi-perspective sentence similarity modeling with convolutional neural networks. *EMNLP*, pages 1576–1586.

Mueller, J. and Thyagarajan, A. (2016). Siamese recurrent architectures for learning sentence similarity. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*.

Socher, R., Huang, E. H., Pennington, J., Ng, A. Y., and Manning, C. D. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *Advances in Neural Information Processing Systems*.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *ACL*, pages 1556–1566.

Zhang, W.-N., Liu, T., Yang, Y., Cao, L., Zhang, Y., and Ji, R. (2014). A topic clustering approach to finding similar questions from large question and answer archives. *PLoS ONE 9(3): e71511*.