Quora question pair similarity problem

Tirth Kothari
Dept. of Computer Science
UMass Lowell
TirthKamlesh_Kothari@Student.uml.edu

Soham Prasad Kulakrni
Dept. of Computer Science
UMass Lowell
SohamPrasad_Kulkani@student.uml.edu

***Abstract*— In this project, we examine the problem of text similarity to get rid of the same questions asked over the question-and-answer platform known as 'Quora'. Typical text contents encountered in real applications are massive and what makes it huge is the redundancy of the content over the internet. We design a systematic way to find out the text-similarity between n questions using several models such as Linear Support Vector Machine (SVM), Logistic Regression, XG-Boost model.**

I. INTRODUCTION

The basic concept in determining the text-similarity is to find if the two pieces of text means the same. If we can do so, this is great in many ways. Solving this issue solves a couple of problems in other ways such as query suggestion, information retrieval system like search, similar image search. In this project, we check the similarity of the n-number of questions.

It's not just about 'Quora'; if there is any other firm with such a question-and-answer platform, this concept applies to them as well. Let us suppose, someone developed the application initially, and as soon as it goes live in production, one may remain busy in getting other stuff done such as marketing of the application, application development, maintenance, and many more, it may be hard to focus on this kind of stuff of text-similarity for the application. So, it remains unnoticed in the production. Then if the application is live in production, google or any other search engine does their work with efficiency for whatever the user asks the question. It fetches similar results, and the results might be from that app, but that makes no sense to the owner of that application because the application is not on the search engine based. However, internally in the applications, there might be thousands of questions that might have redundant things. So, whatever is being done in this project entirely focuses on finding similar meaning questions, or we can say text-similarity between n-number of questions with a good amount of accuracy.

This project focuses on how a pair of two questions are similar; In the upcoming sections, we will come across the background behind this project, methodology, results, future work and conclusion.

II. BACKGROUND

Various works and experiments are done text similarity till now. Some of the works are very noticeable. in [1], the basic topics covered are text-level and distributional semantics without external semantic knowledge. The author also deals with the Text level semantics with external knowledge and some of the features. Features like: SemEval STS and Meta level features.

In text-level semantics without using any external semantic knowledge, paragraph vectors were calculated based on a variation of word2vec. Along with this, other basic string features and semantics were also implemented by the authors.

In distributional semantics, it says the word appearing in a similar context have same meaning. So the author in[10] build a word document co-occurrence matrix using Latent Semantic Analysis Algorithms. after that, it performs Singular value decomposition (SVD) to get the words in vector form. Here the author used word2vec for getting the word embeddings. In word2vec, there are two architectures called Continuous bag of words (CBOW) and skip-gram.

Skip-gram: from the one input word, multiple surrounding words are predicted.

CBOW: in this architecture, the word is predicted on the basis of its surroundings.

As word2vec gives high-quality word embeddings, it is used by the author[8] in experiment In text-level semantics with using external knowledge, the author basically uses POS tagging for measuring semantic similarity.

The features used are SemEvalSTS, which was used by the best team in 2013, to calculate the similarity score and Meta-level features, which gets the characteristic between vectors and the distribution of word embedding.

Seeing the background works of different authors, we decided to do word embedding using pre-trained GloVe model which comes free with spacy and also used TF-IDF. according to the [4] the author had described that the king of speed is TF-IDF and also accuracy is good, so we decided to implement it in our experiment and compare their results on our models. The models used by us are Linear (Support Vector Machine (SVM), Logistic Regression and XG Boost.

III. METHODOLOGY

We utilize the dataset available from the 'Quora' data repository [1]. This dataset is made publicly available by Shankar Iyer, Nikhil Dandekar, and Kornel Csernai to allow training and testing models of semantic equivalence based on genuine Quora data.

The dataset attributes include an 'id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate' and the dataset has 404290 instances, the details of attributes are as follows:

- 'id': each question pair is given unique id this also implies each row has unique id.
- 'qid1': this attribute implies for the corresponding question number 1 in the row there is unique id and if in further going down the dataset if there is copy of this question with corresponding to another question then it will be assigned with same 'qid1' if it is there in place of 'question1'.

- ‘qid2’: this attribute is same as ‘qid1’ just in this case the question going down should be present in ‘question2’ to put the same ‘qid2’.
- ‘question1’: this attribute contains only question in textual format this and ‘question2’ attribute details are same as they both have questions in textual format.
- ‘is_duplicate’: this attribute gives the classification that weather the ‘question1’ is similar to ‘question2’ with two numerical values i.e., 0 and 1, where ‘0’ states that the questions are not similar and ‘1’ states that the questions are similar.

This is on higher level looks like a text-similarity problem by taking deep dive we know that this is a binary classification problem but this in not a simple one it is the which needs the use of Natural Language Processing with various other concepts such as data preprocessing, analyzing the textual data, conversion of textual data to numerical data, the stop-word concept, longest common substring, dimensionality reduction, and mainly the concepts related to the algorithms that we are using for the classification of the data i.e. Linear Support Vector Machine, Logistic Regression, and XG-Boost.

The task at hand for us is to do classification using the classification algorithm mentioned above to do the textual comparison and classification and to know how two questions are similar even though we have a label i.e., ‘is_duplicate’ but for comparison of the textual data we have to use NLP and to use NLP we have to build the dataset of numerical values to compare two question and then we can check that are the two questions are similar or not in the upcoming content you will be able to see the full methodology of the project in detail manner.

It all started like, we read the dataset then we started checking the information and parameters of dataset next looked for the null values in the dataset as we all know that the noise or outliers in the dataset creates disturbance in the results as this is a textual dataset, we decided filling the null values with one common value for all the null values with a common numerical value as a text. Further we looked up for the duplicate and non-duplicate values (Fig.1), later looked for the number of unique questions that are there in the dataset and by looking for more information created a plot for questions which are unique and repeated (Fig.2). Also checked for the repeated pair of question so when we checked the repeated pair of question there were no repeated pair. Checked for the information about the single question which is repeated and got that a single question which was maximumly repeated 157 times this implies that there is a question which has the maximum frequency of repetition in the dataset.

Calculated few features before going for the cleaning of the data.

- ‘freq_qid1’ and ‘freq_id2’: are two feature which provides with the information about the frequency or count of ‘quid1’ and ‘quid2’ respectively.
- ‘q1len’ and ‘q2len’: are the features which gives you the numerical value about the length of the ‘question1’ and ‘question2’ respectively.
- ‘q1_n_words’ and ‘q2_n_words’: are the feature provides a numerical value of the count of the words in the ‘question1’ and ‘question2’ respectively.
- ‘word_Common’: Number of common unique words in ‘question1’ and ‘question2’
- ‘word_Total’: The Total num of words in ‘question1’ + The Total num of words in ‘question2’
- word_share: (word_common)/(word_Total)
- freq_q1 + freq_q2: the sum total of frequency of qid1 and qid2
- freq_q1 - freq_q2: the absolute difference of frequency of qid1 and qid2

Further for understanding the relationship between 'word_share' and the questions 'is_duplicate' or not plotted a violin plot Fig. 3 which shows that how two features are related with each other if you see at the plot in Fig. 3 then you will be able to know that as the word share is less the is duplicate chances are high you can see it in both blue violin and orange violin if you see blue violin it's representing 0 implies not duplicate and hence if word share is less there are fewer chances of being duplicate. Hence, the violin is thick when word share is less and gets thinner as the word share increases. The same concept goes with orange violin in Fig. 3 as orange violin states 1, which means yes, its duplicate, and as the word share is less, there are fewer chances of questions being duplicated, and hence the orange violin is thinner in starting when the word share is less and gets thicker when word share increases.

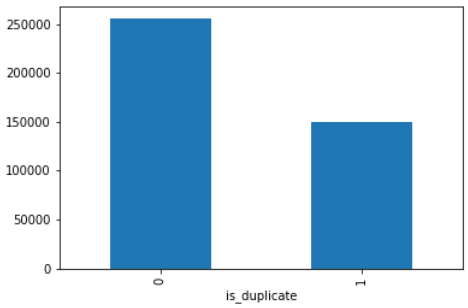

Fig. 1. Result for analysing how many duplicate and non-duplicate values are there in the dataset.

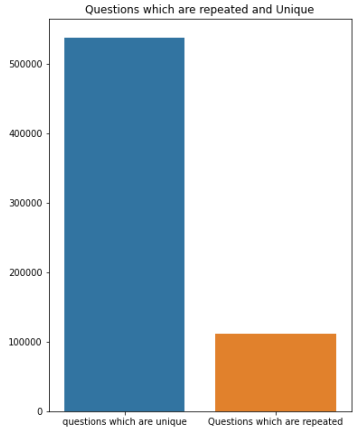

Fig. 2. Plot result for gathering the information regarding question present in the data are unique and questions out of unique are repeated.

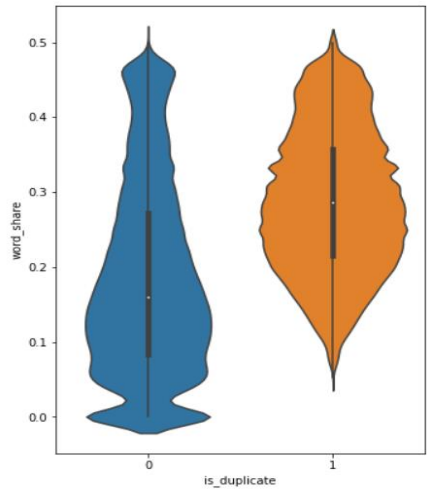

Fig. 3. Result of Violin-Plot which gives relation between word_share and is_duplicate

After performing a little preprocessing on the data with the feature that we generated. We used stemming technique on the data we got and also removed the stop words from the data the functions used for doing these functionalities are part of NLTK module of python. Performed advance feature extraction for getting the relevant numerical data and generated several other features.

The dataset the we have generated out from the original dataset to get the better understanding of the data we created a word cloud plot for duplicate question pairs and we can analyze from the word cloud that in the dataset the highest frequency of question is related to ‘Donald trump’, ‘Hillary Clinton’, ‘best way’, ‘1k’, ‘rupee’ whatever is in large text has the highest frequency of these words Fig.4 are found in the duplicate question pairs and vice versa for the Fig. 5 which shows the word cloud plot for the non-duplicate question

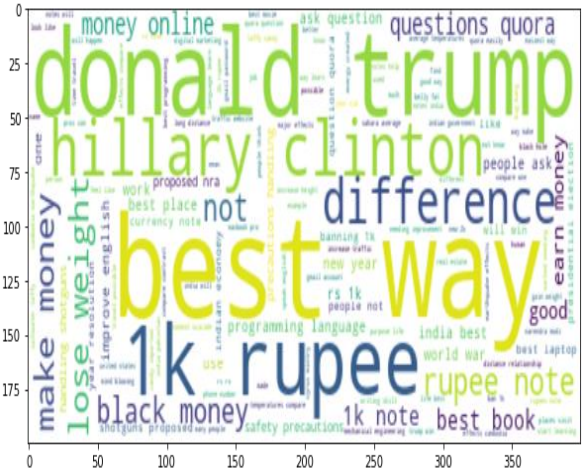

Fig. 4. Word cloud for duplicate question pair.

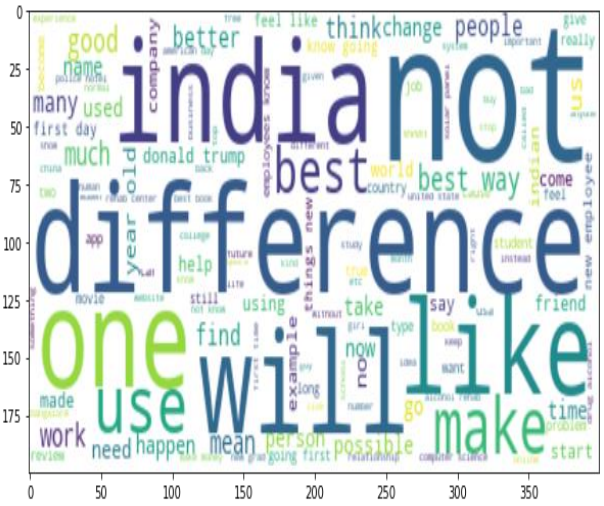

Fig. 5. Word cloud plot for non-duplicate question

We used TSNE as a part of dimensionality reduction, TSNE (T-distributed Stochastic Neighbor Embedding). t-SNE [2] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e., with different initializations we can get different results.

After completing all the preprocessing of the new data, we analyzed the new data and separated the target column and further we used Tfidf-Vectorizer for the weighted word vectors the Tfidf converts the collection of the raw documents to a matrix TF-IDF features it is almost same as Count vectorizer [3]. vectorizer on the two columns i.e., ‘question1’ and ‘question2’ finally after playing really long with data and doing all the necessary operations we split the data into train 70% and test data of 30% after this we applied our algorithms on the data.

A. Logistic Regression

Logistic regression is an uncomplicated and easy-to-understand classification algorithm, and Logistic regression can be easily generalized to various classes.

The linear regression here in logistic regression tries to find the slope and the intercept. Hence, the equation of the plane is similar here "y = mx+c," But the way we use the equation in the case of logistic regression, it is different. Rather than fitting the line to the points, we try to find the line/plane that correctly separates the two data points.

Sigmoid Function, we usually apply the Sigmoid Function on our equation "y=mx + c," i.e., Sigmoid(y=mx + c); this is what Logistic Regression at its core is. Exactly what this sigmoid Function effecting inside.

$$sigmoid(z) = \frac{1}{1+exp(-z)}$$

Fig. 6. Sigmod function applied on z here z = mx +c

This function is used to predict the value of either class 0 or class 1. But before that, we need to generalize values of 'm' and 'c' to predict the new data points. we used an optimization algorithm to find the optimum values of 'm' and 'c.' We used the Stochastic Gradient Descent (SGD) algorithm to perform optimization of parameters.

$$logloss = \frac{-1}{n}\Sigma_{foreach Y_{true}, Y_{pred}}(Y_{true} * log * (Y_{pred}) + (1 - Y_{true}) * log * (1 - Y_{pred}))$$

Fig. 7. Logloss equation

here, 'Ytrue' is a value that is true, and 'Ypred' is a predicted value, Ypred = sigmoid(mx+c) We differentiate this loss function concerning the parameters we wanted to optimize. So here we have to differentiate with respect to. 'm' and 'c'.

$$dm = x_n(y_n - \sigma((m)^T x_n + c))$$

$$dc = y_n - \sigma((m)^T x_n + c))$$

Fig. 8. Logloss differnitaed equation

This is how we compare the text similarity between two questions using the Logistic Regression with Stochastic Gradient Descent Algorithm The results of the applied algorithm will be displayed in results section.

B. Linear Support Vector Machine (SVM)

Support Vector Machine in hard margin classification and linearly separable data. There is minimal discussion of soft margins and kernel tricks, the dual formulation, or more advanced solving techniques. Have prior knowledge of linear algebra, calculus, and machine learning objectives.

The first thing to learn about SVMs is precisely a "support vector." To understand this, it is also essential to understand the goal of SVM, as it has slightly differed from the Logistic Regression and other non-parameter techniques.

SVM aims to draw a decision boundary through linear separable classes such that the border is as robust as possible. This means that the position of the edge is determined by those data points which lie nearest to it, and the decision boundary is a line or hyperplane with as considerable distance as possible from the closest training instance of either class, as shown below in the plot Fig. 9.

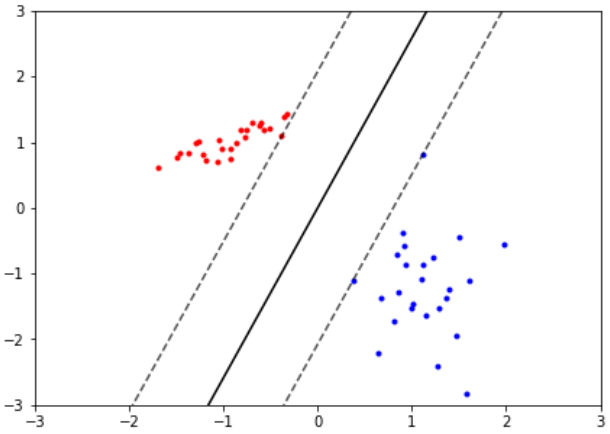

Fig. 9. Support Vector Machine Plot

Using Stochastic Gradient Descent (SGD) on Support Vector Machines (SVM), we must find the gradient of the hinge-loss function. The equation for this can be represented as below:

$$h(\beta) = \frac{1}{2}\|\beta\|^2 + C\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\beta x_i))\right]$$

$$\downarrow$$

$$h(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}\|\beta\|^2 + C * \max(0, 1 - y_i(\beta x_i))\right]$$

Fig. 10. Equation of SVM

By taking the partial derivative of the equation 2nd equation in the above from Fig. 10 with respect to $\boldsymbol{\beta}$ yields the gradient:

$$\nabla_\beta h(\beta) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\beta, & 1 - y_i(\beta x_i) \le 0\\ \beta - C * y_i x_i, & \text{else}\end{cases}$$

Fig. 11. Equation after yeiding gradient to Fig 10 equation

As we calculating single instance at a time with SGD (Stochastic Gradient Descent) the gradient used in this is:

$$\nabla_\beta h(\beta) = \begin{cases}\beta, & 1 - y_i(\beta x_i) \le 0\\ \beta - C * y_i x_i, & \text{else}\end{cases}$$

Fig. 12. SGD equation

The application of Linear SVM with SGD was a great idea for tunning the parameters as it really helped in performance of the model. The results of this will also be shown in the results section.

C. XGBoost

It is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the years[5].

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.[5]

There is nothing for free that we can achieve in Machine Learning (or even life, for that matter). As Data Scientists, we must test all possible algorithms for data at hand to identify the champion algorithm. Besides, picking the correct algorithm is not enough. We must also choose the correct configuration of the algorithm for a dataset by tuning the hyper-parameters. Furthermore, several other considerations for selecting the winning algorithm include computational complexity, explain ability, and ease of implementation. This is precisely the point where Machine Learning starts drifting

away from science towards art, but honestly, that's where the magic happens!

Implementation of XGBoost

Algorithm:

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta).$$

2. For m = 1 to M:

   1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2.$$

$$\hat{f}_m(x) = \alpha \hat{\phi}_m(x).$$

   3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$

Fig. 13. XGBoost: Implementation of Algorithm

This is all about the theoretical and implementation part of XGBoost in practical it is now considered as to be the finest algorithm for most of the problem statements the results of our results will be shown in results section.

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

IV. Results

We predicted the values using the model and converted them to a specific grade level. We generated a log loss and hinge loss as per the given Fig.

A. *Model 1: Logistic Regression*

```
For alpha =  1e-05 The log loss is: 0.3757259506295624
For alpha =  0.0001 The log loss is: 0.3867683906140073
For alpha =  0.001 The log loss is: 0.4155543503526951
For alpha =  0.01 The log loss is: 0.4465394535292481
For alpha =  0.1 The log loss is: 0.46075642499166447
For alpha =  1 The log loss is: 0.48186916065055896
For alpha =  10 The log loss is: 0.5287824258455822
```

Fig. 14. Logloss for Logistic Regression with different values of alpha

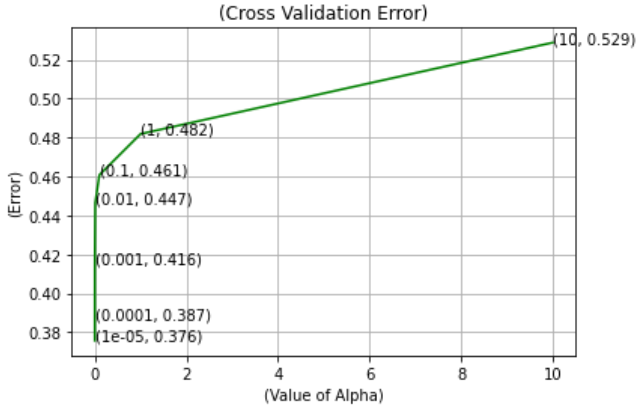

Fig. 15. Cross Validation Error for Logistic Regression

```
For the best alpha among all =  1e-05 The value of log loss for train is: 0.3733158407992717
For the best alpha among all =  1e-05 The value of log loss for test is: 0.3757259506295624
```

Fig. 16. Logloss value for the best value of alpha

B. *Model 2: Linear Support Vector Machine (SVM)*

Asda

```
For alpha =  1e-05 The log loss is: 0.3757259506295624
For alpha =  0.0001 The log loss is: 0.3867683906140073
For alpha =  0.001 The log loss is: 0.4155543503526951
For alpha =  0.01 The log loss is: 0.4465394535292481
For alpha =  0.1 The log loss is: 0.46075642499166447
For alpha =  1 The log loss is: 0.48186916065055896
For alpha =  10 The log loss is: 0.5287824258455822
```

Fig. 17. Logloss for Linear SVM with different values of alpha

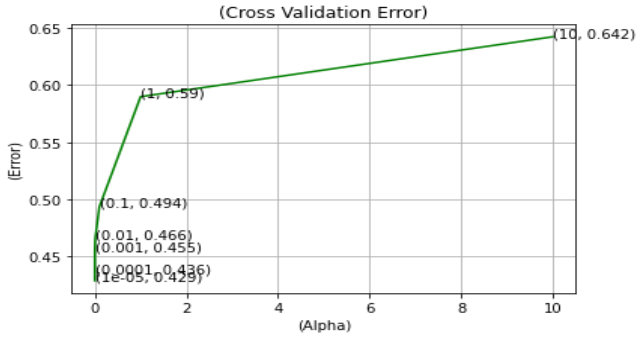

Fig. 18. Cross Validation Error for Linear SVM

```
For the best alpha among all =  1e-05 The value of log loss for train is: 0.3733158407992717
For the best alpha among all =  1e-05 The value of log loss for test is: 0.3757259506295624
```

Fig. 19. . Logloss value for the best value of alpha

C. *Model 3: XGBoost*

For our dataset among all the models XGBoost prove to be one of the best models compared to other two models here is the log loss value for XGBoost Model in the given Fig. 20.

```
The test log loss is: 0.3482895103792922
```

Fig. 20. Logloss value for the best value of alpha for the XGBoost

V. Conclusion

From the experiments we performed, we have concluded that among the all the models implemented, XG-Boost gives very less log loss value and then Linear SVM and Logistic regression gives almost the same values for our dataset.

VI. References

[1] Tom Kenter and Maarten de Rijke. Short Text Similarity with Word Embeddings. In University of Amsterdam, The Netherlands 2015.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[4] https://towardsdatascience.com/the-best-document-similarity-algorithm-in-2020-a-beginners-guide-a01b9ef8cf05.

[5] https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

[6] E. Agirre, D. Cer, M. Diab, A. Gonzalez-Agirre, and W. Guo. sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In *SEM 2013, 2013.

[7] C. Banea, D. Chen, R. Mihalcea, C. Cardie, and J. Wiebe. Simcompass: Using deep learning word embeddings to assess cross-level similarity. SemEval 2014, 2014.

[8] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. In ECIR 2007, 2007

[9] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitasks learning. In ICML 2008, 2008

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[11] L. Han, A. Kashyap, T. Finin, J. Mayfield, and J. Weese. UMBC EBIQUITY-CORE: Semantic textual similarity systems. In *SEM-2013, 2013