Quora Question Pairs

Amit Kanwar (akanwar2)

Sudipto Biswas (sbiswas4)

Github Link: https://github.ncsu.edu/akanwar2/quora-question-pairs

Overview

Quora is a question-answer platform where questions are organized, edited, and answered by its community of users (“Quora”, n.d. [1]). It was founded by Adam D’ Angelo and Charles Cheever in June 2010. Quora had an estimated 100 million monthly unique visitors, as of March 2016 (“Quora”, n.d. [1]) . On this platform many people ask similarly worded questions, because of this seekers may find it very difficult to find the best answer to a question and makes writer answer multiple versions of same question (Quora Question Pairs, 2017 [2]). “Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term” (Quora Question Pairs, 2017 [2]). Right now, Quora uses Random Forest Model to find similar questions. But it is not that effective as we can see a number of duplicate questions still being asked on Quora which has a negative effect on the site visits by active seekers and writers. Our Proposal is to create a more complex and advanced method to tackle this NLP problem so as to provide Quora seekers and writers with an improved experience. .

Problem Statement

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

We want to tackle this natural language processing problem by applying advanced techniques to classify whether question pairs are duplicates or not. Doing so will make it easier to find high quality answers to questions resulting in an improved experience for Quora writers, seekers, and readers.

Technical Description

Data Description

Quora is providing us with a training and test dataset. The train dataset contains the id of the pair, ids of the two questions in the pair, two questions, and a field target variable set to 1 in case they are duplicate and 0 in case they are not. All these questions are actual examples from Quora. The ground-truth labels in training data are not 100% accurate and should be taken as "informed". Test data contains some of Kaggle's automatically generated pairs as an anti-cheating measure which are not counted in scoring.

Techniques Used

We first used Jaccard Coefficient, Cosine Similarity, Pearson's Coefficient and TF-IDF based Cosine Similarity individually to get the accuracy (log-loss in case of Kaggle). Finally we used these coefficients to train a model and significantly reduced the log-loss value.

I. Jaccard Coefficient

Jaccard similarity coefficient is a statistic used for comparing the similarity and diversity of sample sets. It is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(X,Y) = |X \cap Y| \ / \ |X \cup Y|$$

It ranges from 0 to 1 with higher the number the closer the sets are.
We propose to calculate the Jaccard coefficient between keywords of two questions to find out how similar they are.

II. Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity

of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1].

The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos\theta$$

Given two vectors of attributes, A and B, the cosine similarity, cos(θ), is represented using a dot product and magnitude as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$, where A_i and B_i are components of vector A and B respectively.

Image Taken as is from [17]

The resulting similarity ranges from −1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality (decorrelation), and in-between values indicating intermediate similarity or dissimilarity.

For text matching, the attribute vectors A and B are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison.

III. Pearson's Coefficient

Pearson coefficient [7] is a measure of the linear correlation between two variables and possess values between -1 and +1. From the perspective of text similarity, given a term set T = {t1, . . . , tm}, the Pearson Coefficient can be written as (Taken from [6]):

$$SIM_P(\overrightarrow{t_a}, \overrightarrow{t_b}) = \frac{m \sum_{t=1}^{m} w_{t,a} \times w_{t,b} - TF_a \times TF_b}{\sqrt{[m \sum_{t=1}^{m} w_{t,a}^2 - TF_a^2][m \sum_{t=1}^{m} w_{t,b}^2 - TF_b^2]}}$$

where $TF_a = \sum_{t=1}^{m} w_{t,a}$ and $TF_b = \sum_{t=1}^{m} w_{t,b}$.

IV. TF-IDF Based Cosine Similarity

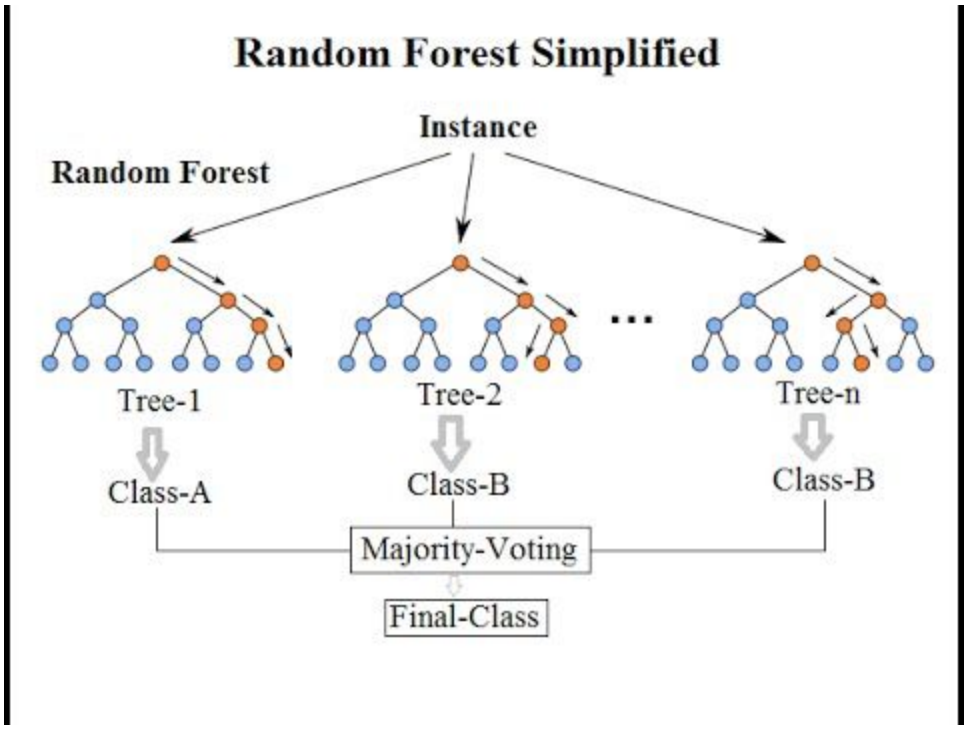
A commonly used term weighting method is tf-idf, which assigns a high weight to a term, if it occurs frequently in the document but rarely in the whole document collection. Contrarily, a term that occurs in nearly all documents has hardly any discriminative power and is given a low weight, which is usually true for stop-words like determiners, but also for collection-specific terms; think of apparatus, method or claim in a corpus of patent documents. For calculating the tf-idf weight of a term in a particular document, it is necessary to know two things: how often does it occur in the document (term frequency = tf), and in how many documents of the collection does it appear (document frequency = df). Take the inverse of the document frequency (= idf) and you have both components to calculate the weight by multiplying tf by idf. In practice, the inverse document frequency is calculated as the logarithm (base 10) of the quotient of the total number of documents (N) and the document frequency in order to scale the values. [4]

$$idf = \log\left(\frac{N}{df}\right)$$

V. Combining them all

Since the log-loss value was coming out to be very high in all of these used individually, we combined these techniques to create a more complex model which can better fit the test data. We used the following methods for this:

1. Random Forests: Random Forests is very powerful learning model for any kind of classification problem. It belongs to the ensemble class of the prediction models which combine the classification power of multiple classifiers. In this case, those classifiers are Decision Trees. It is very easy and doesn't require that much tuning as compared to other models. It is useful for both Regression and Classification but it tends to be overfitting in case of Classification. (Random Forests in Python, n.d. []).



An instance of Random Forest (Taken as is from [16])

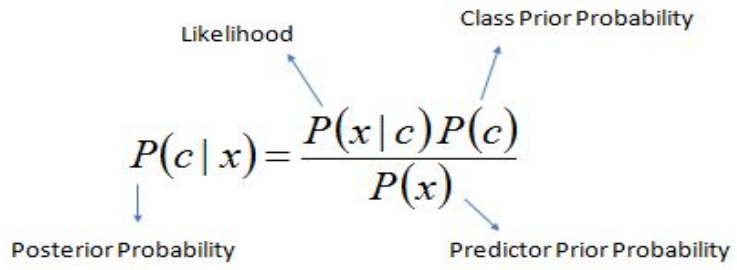
2. Logistic Regression: "Logistic Regression is a categorical model where the dependent variable is a categorical variable" ("Logistic Regression, n.d. [12]). Logistic function F(x) is defined by:

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

(Taken as is from "Logistic Regression", n.d. [12])

The independent variables include Jaccard Coefficient, Cosine Similarity, Pearson Coefficient and TF-IDF based Cosine Similarity. We trained this model on training data, we have been provided. This model significantly reduced the log-loss value for the Kaggle Competition.

3. Naive Bayes: Naive Bayes is a supervised learning method which combines the probabilities of each attribute to make a prediction that those belong to a particular class. It is based on the assumption that the attributes are independent of each other. Most of the times, this assumption holds and it gives us very fast and robust solutions. "To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability" (Naive Bayes Classifier from Scratch, n.d. [14]).



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Bayes Equation (Taken as is from [15])

4. Probabilistic Voting Ensemble: We were seeing a remarkable decrease in the log-loss value by using Ensemble Classifiers as above. Since adding complexity in the model seemed to be bumping us up the leaderboard, we thought of creating an ensemble of all the above ensembles. We used Probabilistic Voting Ensemble for this. We took the results from the ensemble classifiers (Random Forests, Logistic Regression, Naive Bayes) and took the average of the probabilities to get the final probability value. While testing, we gave each of them same weights as [1, 1, 1] which gave us the lowest log-loss value among all.

Research Papers

While working on the project we read many interesting papers. The three most influential among them are

[1] **Pedersen, T, Patwardhan, S and Michelizzi, J. "WordNet:: Similarity: measuring the relatedness of concepts." Demonstration papers at HLT- ... dl.acm.org, 2004 -** Wordnet makes it possible to measure the semantic similarity and relatedness between a pair of concepts (or synsets). It provides six measures of similarity, and three measures of relatedness, all of which are based on the lexical database WordNet. These measures are implemented as Perl modules which take as input two concepts, and return a numeric value that represents the degree to which they are similar or related.

[2] **Islam, A and Inkpen, D. "Semantic text similarity using corpus-based word similarity and string similarity." ACM Transactions on Knowledge Discovery from ... dl.acm.org, 2008 -** The paper proposes a method for measuring the semantic similarity of texts using a corpus-based measure of semantic word similarity and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm. Existing methods for computing text similarity have focused mainly on either large documents or individual words. This technique focus on computing the similarity between two sentences or two short paragraphs. The proposed method can be exploited in a variety of applications involving textual

knowledge representation and knowledge discovery. Evaluation results on two different data sets show that the method outperforms several competing methods.

[3] **Aggarwal, CC and Zhai, CX. "A survey of text clustering algorithms." Mining text data Springer, 2012 -** Clustering is a widely studied data mining problem in the text domains. The problem finds numerous applications in customer segmentation, classification, collaborative filtering, visualization, document organization, and indexing. In this chapter, we will provide a detailed survey of the problem of text clustering. We will study the key challenges of the clustering problem, as it applies to the text domain. We will discuss the key methods used for text clustering, and their relative advantages. We will also discuss a number of recent advances in the area in the context of social network and linked data.

Implementation and Results

We implemented this project using python 2.7. Data pre-processing was one of the chief components in the implementation. We had to get rid of special characters and stopwords to prevent any bias they might create while calculating the coefficients. We used nltk library for the same. For the classifiers, we used sklearn library. To handle data, we used numpy and pandas libraries. We first implemented Jaccard Coefficient, Pearson's Coefficient, Cosine Similarity and TF-IDF based Cosine Similarity and used sklearn to create models out of those values.

Log-Loss Measure

Kaggle uses a log-loss measure to evaluate the performance of the model. Rather than assigning a class to the set of attributes, we calculate the probability given by the model for that class. The log-loss is known to penalize the low probabilities or the wrong values more. It is calculated as:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

Taken as is from [17]

Results

First of, we simply ran these on the test dataset without any training. Jaccard Coefficient gave us a log-loss of **1.05962**, Cosine Similarity a log-loss of **0.98322**, and TF-IDF **0.96163.** Using Ensemble Classifiers, we were able to significantly reduce the log-loss value. Random Forests gave us a log-loss of **0.58670**, Logistic Regression **0.60782**, and Naive Bayes

0.61589. Combining all these ensemble classifiers using Probabilistic Voting Ensemble gave us a much lower log-loss value of **0.40167.** For now, this is our best result which is better than all the classifiers trained alone. (**Note:** Due to large train and test datasets, it might take a while to train and get the prediction values)

Future Plans

Since, the competition will commence in 2 months, we have sufficient time to try to improve our model. The next step that we are thinking is to fine tune the weights assigned to the ensemble of classifiers in Probabilistic Voting Ensemble. We plan on using k-fold cross-validation for this and hope that this will more reduce the log-loss values.

References

1. **Quora, n.d. Retrieved from https://en.wikipedia.org/wiki/Quora**
2. **Quora Question Pairs, 2017. Retrieved from https://www.kaggle.com/c/quora-question-pairs**
3. **http://www.iaeng.org/publication/IMECS2013/IMECS2013_pp380-384.pdf**
4. **http://www.ir-facility.org/scoring-and-ranking-techniques-tf-idf-term-weighting-and-cosine-similarity**
5. **Cosine similarity, Pearson correlation, and OLS coefficient. [URL]**
6. **Huang, Anna. "Similarity measures for text document clustering." Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. 2008.**
7. **Pearson correlation coefficient. [URL]**
8. **Pedersen, T, Patwardhan, S and Michelizzi, J. "WordNet:: Similarity: measuring the relatedness of concepts." Demonstration papers at HLT- ... dl.acm.org, 2004.**
9. **Gomaa, WH and Fahmy, AA. "A survey of text similarity approaches." International Journal of Computer ... search.proquest.com, 2013.**
10. **Aggarwal, CC and Zhai, CX. "A survey of text clustering algorithms." Mining text data Springer, 2012.**
11. **Islam, A and Inkpen, D. "Semantic text similarity using corpus-based word similarity and string similarity." ACM Transactions on Knowledge Discovery from ... dl.acm.org, 2008.**
12. **"Logistic Regression", n.d., Retrieved from https://en.wikipedia.org/wiki/Logistic_regression**

13. Random Forests in Python, n.d., Retrieved from http://blog.yhat.com/posts/random-forests-in-python.html
14. Naive Bayes Classifier from Scratch, n.d., Retrieved from http://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
15. Bayes Equation, n.d., Retrieved from http://www.saedsayad.com/images/Bayes_rule.png
16. Random Forest Image, Retrieved from https://www.youtube.com/watch?v=ajTc5y3OqSQ
17. Kaggle Log-loss Wiki, Retrieved from https://www.kaggle.com/wiki/LogarithmicLoss
18. Cosine Similarity, Wikipedia, n.d., Retrieved from https://en.wikipedia.org/wiki/Cosine_similarity