BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI CAMPUS

Submitted in partial fulfillment of the
requirements of

CS F469 Information Retrieval

PREDICTIVE TYPING SYSTEM
and
SEMANTIC SIMILARITY BETWEEN QUERIES

Adarsh Sanghai	2014B2A70667P
Amitojdeep Singh	2014B3A70615P
Anirudh Kumar Bansal	2014A7PS0081P
Lakshit Bhutani	2014A7PS0095P

November 14, 2017

Problem Statement

Brief Description:

The project involves building a predictive typing system (for English) using two approaches.
In the first approach, using a large corpus, a language model is constructed using five different smoothing methods - Absolute Discounting, Additive Smoothing, Witten-Bell, Jelinek-Mercer and Good-Turing . The implemented language models with different smoothing techniques are evaluated using measures of perplexity and cross-entropy using a separate test corpus. This approach is suitable for long text.
The second approach involves finding the semantic similarity between two texts and using it to predict the next word. This approach is suitable for short text.
These two approaches are then compared for their prediction abilities.
Finally, a text editor with the predictive typing capabilities is developed. A suitable GUI will be provided to the user for using the text editor.

Module1

a) Corpus data preprocessing and implementation of few smoothing techniques and interface implementation
b) Implementation of remaining smoothing techniques and their comparison.

Module2

a) Preparation of dataset for semantic similarity and finding the semantic similarity between two texts
b) Using semantic similarity for prediction and comparison of both approaches.

Background of the Problem

- **Application Domain**

A language model is a probability distribution p(s) over strings s that describes how often the string s occurs as a sentence in some domain of interest. Language models are employed in many tasks including speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction. In our project, we are using language models to implement a word predictor and use it to make a predictive typing system. The central goal of the most commonly used language models, trigram models, is to determine the probability of a word given the previous two words: $P\ (w_i \mid w_{i-2} w_{i-1})$. The simplest way to approximate this probability is to compute:

$$P\ (w_i \mid w_{i-2}\ w_{i-1}\ )\ = \frac{C\ (w_{i-2}\ w_{i-1}\ w_i\ )}{C(w_{i-2} w_{i-1})}$$

Unfortunately, this estimate (called the maximum likelihood estimate) can lead to poor performance in many applications of language models. Hence smoothing is used to address this problem. The term smoothing describes techniques for adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities. The name comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward.

Siamese LSTM is an adaptation of Long Short-Term Memory (LSTM) network for comparison of variable-length text sequences. LSTM has been found to be very effective in language modelling tasks including NLP and speech recognition due to it's inherent memory which can capture the context arising from sequence of words in a sentence. It uses two identical branches of LSTM network and is generalization of Manhattan LSTM which can have asymmetric structure. General Manhattan LSTM can be more useful for tasks like query - document matching which query and documents are written in a different manner.

In the first module of this project, five different smoothing techniques are used in the predictive text editor and they are compared using two measures of evaluation - perplexity and cross-entropy.

In the second module Siamese LSTM has been used for language modelling to obtain semantic similarity between two short pieces of text. This model can be applied to tasks like comparing two queries, aggregating similar queries, duplicate question detection, query ranking etc. The model is evaluated using the change in mean squared errors and accuracy for validation and training data over the course of training of the model.

- **Motivation of the problem**

The typing speed for most people is generally much slower compared to their thinking or talking speed. However, if users have a list of options for relevant words available to them then simply selecting one of them is much faster than typing that word out and even comparable to their thinking or talking speed. This can be done by using a word predictor that predicts the next word based on previous few words (in our case previous two words since we are using trigrams). Thus, a predictive text-based editor can be very useful to frequent typists.

Identification of similar queries and questions can help optimize many Information Retrieval problems. Some of these include Query Ranking, Duplicate Question Identification, Short Text Clustering and Query - Document Title matching. It is not feasible to use typical methods like tf-idf on small texts like queries or tweets as they don't account for semantics. LSTM has been highly successful in language modelling recently and can be applied to this domain.

- **Technical issues included in the work**

The foremost technical issue was how to compute various conditional probabilities and counts of trigrams,bigrams and unigrams efficiently since it would be inefficient to keep traversing through the training corpus to find these probabilities/counts every time the next word needed to be predicted. Thus we precomputed the counts of these relevant n-grams using dictionary in python indexed by the n-gram (implemented as hash table).

This can result in a large memory requirement, particularly if the order of n is large or if the training corpus is very large. Fortunately, because of the small value of n (3 in our case) and a reasonable size of training corpus that can fit in main memory, we did not have to use too much memory.

The Good-Turing method also required us to know the count of n-grams which have a particular frequency in the training corpus. This was also pre-computed (after computing the counts of all relevant n-grams) and stored using dictionary in python indexed by the n-gram. This increased our memory requirement but it still remained reasonably small while it made the Good-Turing method efficient.

Selection of appropriate labelled database for Semantic Similarity tasks is itself challenging as most query hit databases are proprietary, while others like SICK database are quite small for deep learning and need intensive data augmentation to be useful for deep learning.

Related Work

A lot of work has been done on language modelling using n-grams and it has been applied for various purposes including word prediction, handwriting recognition, machine translation, spelling correction and rank-based query retrieval.

Stanley F. Chen and *Joshua Goodman* in the 1999 paper '*An empirical study of smoothing techniques for language modeling*' surveyed the most widely used algorithms for smoothing methods of n-gram

modelling. They compared algorithms such as Jelinek-Mercer, Witten Bell, Katz, Kneser-Ney etc. investigating how various factors like training data size, training corpus and n-gram order affect their relative performance, which is measured using cross-entropy on a test data.

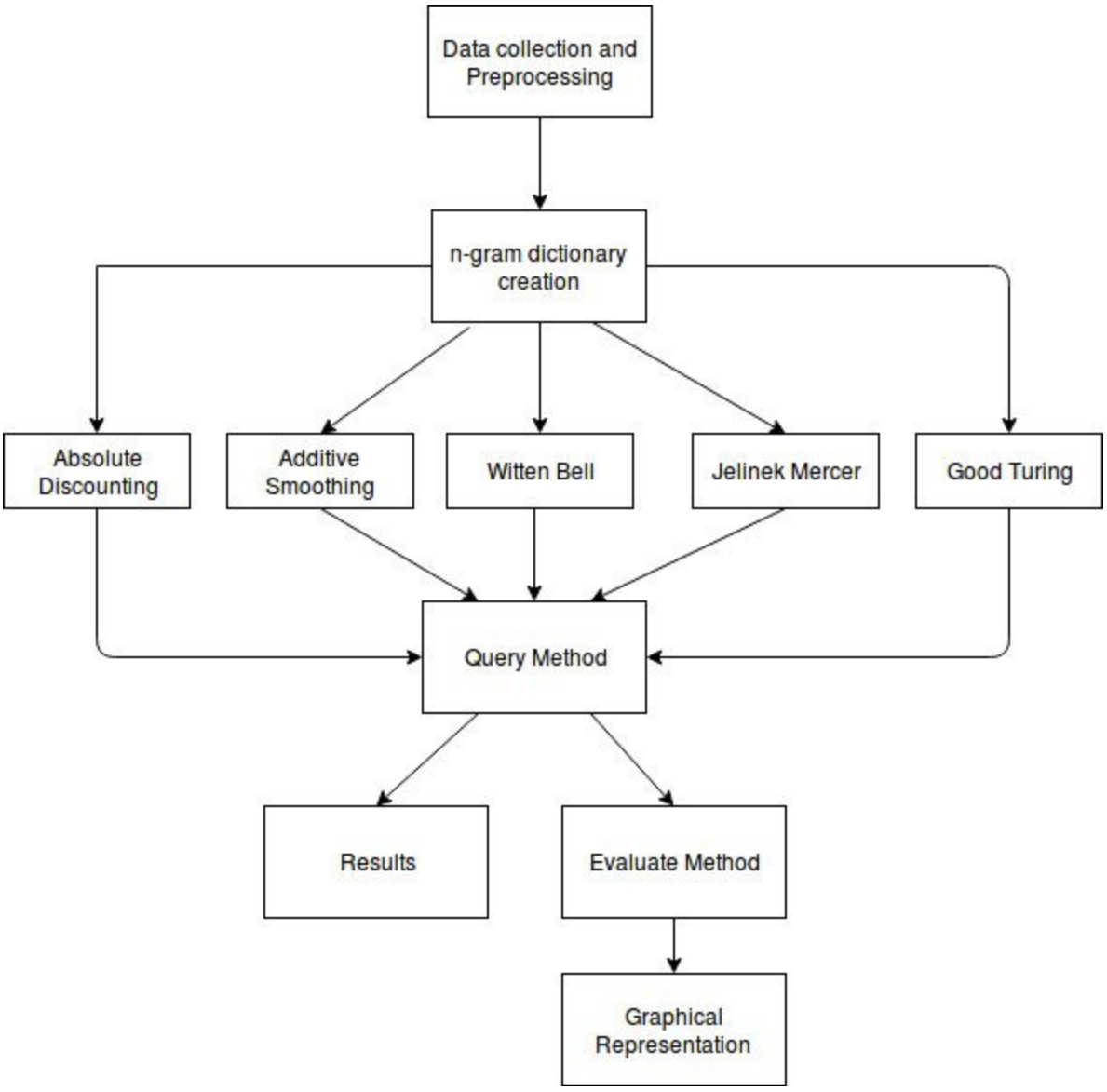
Chengxiang Zhai and *John Laferty* in their paper '*A Study of Smoothing Methods for Language Models Applied to Information Retrieval*' studied the influence of different language modelling methods on query retrieval performance. The basic idea of these approaches is to estimate a language model for each document, and to then rank documents by the likelihood of the query according to the estimated language model. Experimental results show that not only is the retrieval performance generally sensitive to the smoothing parameters, but also the sensitivity pattern is affected by the query type, with performance being more sensitive to smoothing for verbose queries than for keyword queries. Verbose queries also generally require more aggressive smoothing to achieve optimal performance.

Language modelling has many interesting applications including in the fields of speech recognition, machine translation and image captioning. One particular field where it has been used extensively is semantic matching of text. Semantic matching is used for many various purposes, for example modern day search engines use semantic matching for retrieval of web documents.

One such paper that talks about semantic matching and its implementation is "*A latent semantic model with Convolutional-Pooling structure for Information Retrieval*" by *Yelong Shen, Xiaodong He, Jianfeng Gao* and others. The paper talks about developing a LSM (latent semantic model) which is based on convolutional neural network in order to take into account the contextual information. Instead of using a normal bag of words model they use the latent semantic representation to store maximum possible contextual information. They also compare their model by comparing with studies on some other state of the art semantic matching models and find improvements.

The other paper of reference was "*Siamese Recurrent Architectures for learning Sentence similarity*" by *Jonas Mueller* and *Aditya Thyagarajan*. Their paper focuses on implementing a siamese adaptation of the LSTM model also known as the Long short-term Memory model, which is quite widely used for language modelling. They try to show that LSTM may be adapted to a simpler version and used by training on paired sentences, where they ultimately store high structured space of representations which aim to store rich semantics. They focus on using pre trained word-vectors and synonym augmentation as their LSTM inputs and demonstrate good results. They have proposed that other than for scoring semantic scoring, trained MaLSTM sentence representations can produce interesting insights in exploratory data analysis. The model implemented here is based on this paper as LSTMs are typically better at language modelling tasks.

System Description for Word Prediction



The system is implemented as a client-server program where the client request the server to do preprocessing of training data and dictionary creation, then requests it to apply a particular method to get prediction for the next word and also to evaluate results on the test files and display results. The above flowchart is for the server side of the program. The modules and their functions are:

1. **Data collection and preprocessing**

The dataset involved documents which contained sentences on new lines. The sentences contain words in lower case without any punctuations or numbers. No stop word removal or stemming is performed.

The standard datasets used are -

1. A Corpus of Sentence-level Revisions in Academic Writing: *A Step towards Understanding Statement Strength in Communication* : *Chenhao Tan and Lillian Lee* in Proceedings of ACL (short papers) , 2014 [17.7 MB]
Data was in the form of csv and hence was read appropriately and then processed.

2. Supreme Court Dialogs Corpus v1.01 : Initially featured in *Computational analysis of the conversational dynamics of the United States Supreme Court*, *Timothy Hawes*, Master's Thesis, 2009 [11.7 MB]
The instances were of the format :
CASE_ID +++$+++ UTTERANCE_ID +++$+++ AFTER_PREVIOUS +++$+++ SPEAKER +++$+++ IS_JUSTICE +++$+++ JUSTICE_VOTE +++$+++ PRESENTATION_SIDE +++$+++ UTTERANCE
Only the utterance part is extracted and processed.

3. Cornell Movie-Quotes Corpus v1.0 : Extracted from *http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html - moviequotes.memorable_quotes.txt* [672 KB]
Each item contains 3 lines with the following format:
MOVIE_TITLE
MEMORABLE_QUOTE
LINE_ID_MEMORABLE MATCHED_QUOTE
Only the memorable quote part was extracted and processed.

4. Cornell Movie-Quotes Corpus v1.0 : Extracted from *http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html - moviequotes.scripts.tx*t [54.3 MB]
Each item is of the form (the field separator is: "+++$+++")
LINE_ID MOVIE_TITLE MOVIE_LINE_NR CHARACTER REPLY_TO_LINE_ID TEXT
Only the text part is extracted and processed.

5. The extracted datasets used are obtained by extracting the top 10 daily news (title and body) from the following online newspapers :

"abc-news-au", "bbc-news", "al-jazeera-english" , "associated-press" , "bloomberg" , "buzzfeed", "cnbc", "cnn", "daily-mail", "espn", "espn-cric-info", "fortune", "google-news", "hacker-news", "independent", "metro", "mirror", "mtv-news" , "national-geographic", "new-york-magazine", "polygon", "reuters", "the-hindu", "the-economist", "the-guardian-uk", "the-new-york-times", "the-telegraph", "the-times-of-india", "the-wall-street-journal" , "time" and "usa-today" using an API provided by *newsapi.org*.

Retrieval was performed twice on :

1. Thu, Nov 9 2017 12:37:06 [944 KB]
2. Sun, Nov 12 2017 00:04:49 [1.1 MB]

2. N-gram dictionary creation

The sentences in the documents are split into words using delimiters and following counts stored :

- *Unigram*, *bigram* and *trigram* dictionaries which store the frequency of a particular unigram, bigram and trigram respectively.
- *Count_unigram*, *count_bigram* and *count_trigram* which store the count of unigrams, bigrams and trigrams having a particular frequency
- *Count_distinct_uni* and *count_distinct_bi* storing the count of distinct unigrams and bigrams in the dataset.

3. Additive smoothing

It applies the additive smoothing method to get predictions for the next word given the previous two words. We assume each n-gram occurs δ more than it actually does where 0 < δ ≤ 1 . This avoids zero probabilities.

$$p_{\text{add}}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} c(w_{i-n+1}^{i})}$$

V is the vocabulary, or set of all words considered. *Lidstone and Jeffreys* advocate taking δ = 1. Hence we have used δ = 1 in this project.

4. Good-Turing

It applies the good turing method to get predictions for the next word given the previous two words. For any n-gram that occurs r times, this method pretends that it occurs r* times where:

$$r^* = \frac{(r+1) \cdot n_{r+1}}{n_r}$$

where nr is the number of n-grams that occur exactly r times in the training data. This can be written as a probability after normalizing:

$$P_{GT}(w^{i}_{i-n+1}) = \frac{r^*}{N}$$

Where N = $\sum_{r=0}^{\infty} n_r r^*$

5. Jelinek-Mercer

It applies the jelinek-mercer method to get predictions for the next word given the previous two words. It uses linear interpolation to combine the information from lower-order n-gram models in estimating higher-order probabilities.

$$p_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) p_{\text{interp}}(w_i|w_{i-n+2}^{i-1}).$$

The n^{th} - order smoothed model is defined recursively as a linear interpolation between the nth-order maximum likelihood model and the $(n-1)^{th}$ - order smoothed model. We took the value for the base case of n=1 as the maximum likelihood probability.

The values for the constants can either be obtained using the Baum-Welch algorithm which works on maximizing the probability of some data or by using a training method after partitioning the n-grams into buckets depending on their counts. However, either of these approaches were found to be difficult to implement and the constant value for each n-gram (or lower) was taken to be same (though it is not recommended) since we wanted a basic implementation for word prediction. Using evaluation measures like cross-entropy and perplexity on a portion of the training data, we got a value of 0.4 (after trying out different values for the constant).

6. **Witten-Bell**

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})}.$$

It applies the Witten-Bell method to get predictions for the next word given the previous two words. It is a special case of jelinek-mercer where the constant is determined as:

Where $N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1} w_i) > 0\}|.$

7. **Absolute Discounting**

It applies the witten-bell method to get predictions for the next word given the previous two words. It also involves linear interpolation of higher and lower order models like jelinek mercer and witten bell but here a constant D is subtracted from the higher order counts.

$$p_{\text{abs}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + (1 - \lambda_{w_{i-n+1}^{i-1}}) p_{\text{abs}}(w_i|w_{i-n+2}^{i-1}).$$

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet)$$

Ney et al (1994) estimate the constant D as:

$$D = \frac{n_1}{n_1 + 2n_2}$$

Where n_1 and n_2 are the total number of unigrams with exactly one and two counts respectively in the training data.

8. **Query method**

It is given a partial sentence and a method name and it finds the top 10 predictions for the next word as given by the method. It finds the last two words and then calls the appropriate method to get these predictions. The corresponding method iterates through the vocabulary of terms in the training data set, finds the probability of that term appearing next given the previous two terms and adds the tuple (probability,term) to a priority queue. It later pops off 10 items from the queue to get the top 10 terms (i.e top 10 probabilities).

9. **Results**
It graphically displays the results in the corresponding textbox by reading the partial sentence entered in the input textbox and calling query method to get the top 10 predictions.

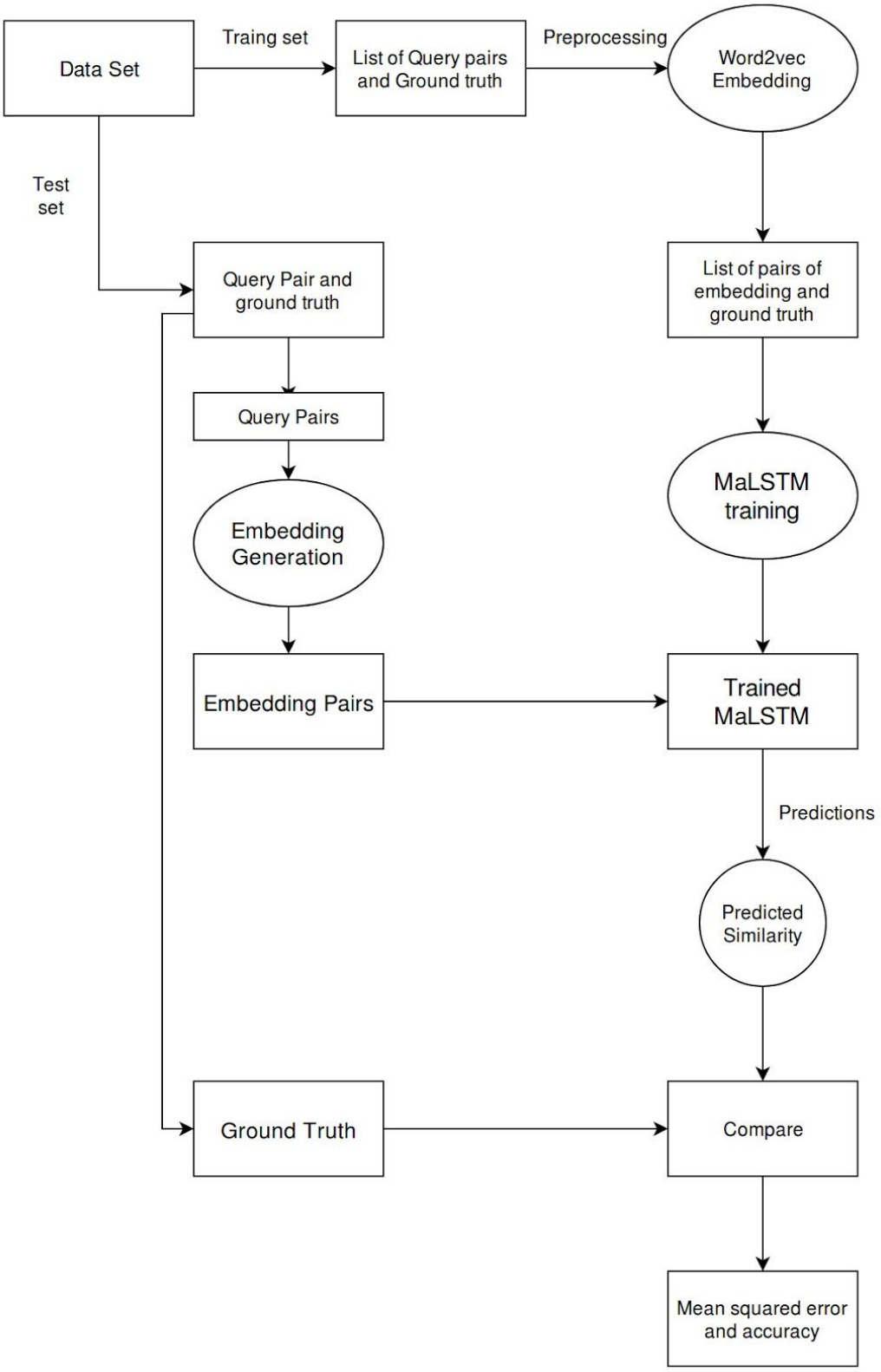
10. **Evaluate method**
It finds two measures of evaluation i.e cross-entropy and perplexity for each of the methods on a test dataset containing four files - two files with sentences taken from the training corpus (in the domain) and two files with arbitrary sentences (out of domain).
For a test set T composed of the sentences $(t_1, t_2, ..., t_{l_T})$ where l_T is the number of sentences in the test set T, the cross-entropy is defined as: $H_p(T) = -(1/W_T) log_2 p(T)$ and the perplexity is defined as : $PP_p(T) = 2^{H_p(T)}$
Where p(T) is the product of the probabilities of all sentences in the set: p(T) =
And W_T is the total number of words in the test set T

The cross-entropy can be interpreted as the average number of bits needed to encode each of the words in the test data using the compression algorithm associated with model. The perplexity of a model is the reciprocal of the (geometric) average probability assigned by the model to each word in the test set T.

Thus lower the values of cross-entropy and perplexity on the test set, better is the smoothing method.

11. **Graphical Representation**
It uses the two evaluation metric values provided by evaluate method on all the four testfiles and with all the five methods to generate two graphs - one for cross-entropy and one for perplexity showing how the values for the in-domain files (testfiles 1 and 2) compare with that of out-of domain files(testfiles 3 and 4) for each of the five smoothing methods.

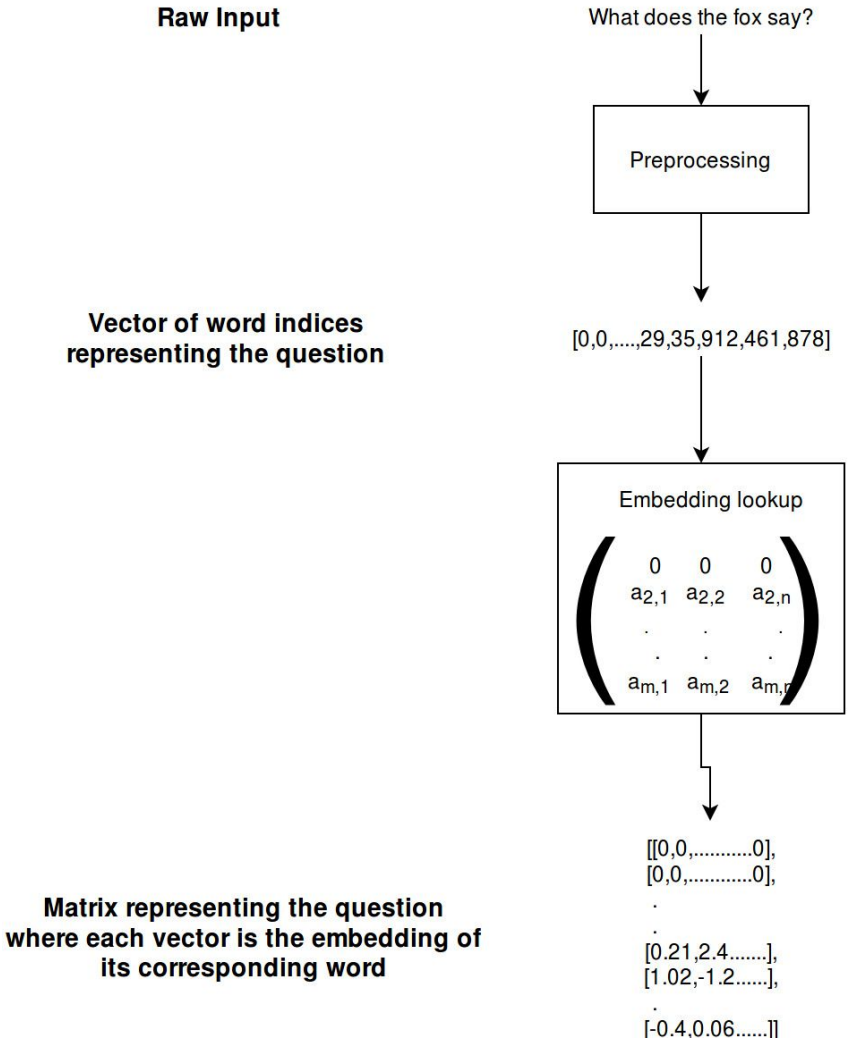
System Description for Semantic Similarity of Queries



1. **Dataset**
   Quora Question Pairs Dataset which is publically available on Kaggle has been used to train the Siamese LSTM Model. It has 400,000 samples of potential question duplicate pairs. Each sample has two questions along with ground truth about their similarity(0 - dissimilar, 1- similar). These are split into test and training dataset. Further 40,000 pairs have been separated from training dataset for validation.

2. **Preprocessing**
   Raw text needs to be converted to list of word indices. A helper function takes a string as input and outputs a list of words from it after some preprocessing like accommodating specific signs.

3. **Embeddings**
   The flowchart below shows the process of embedding generation used here.



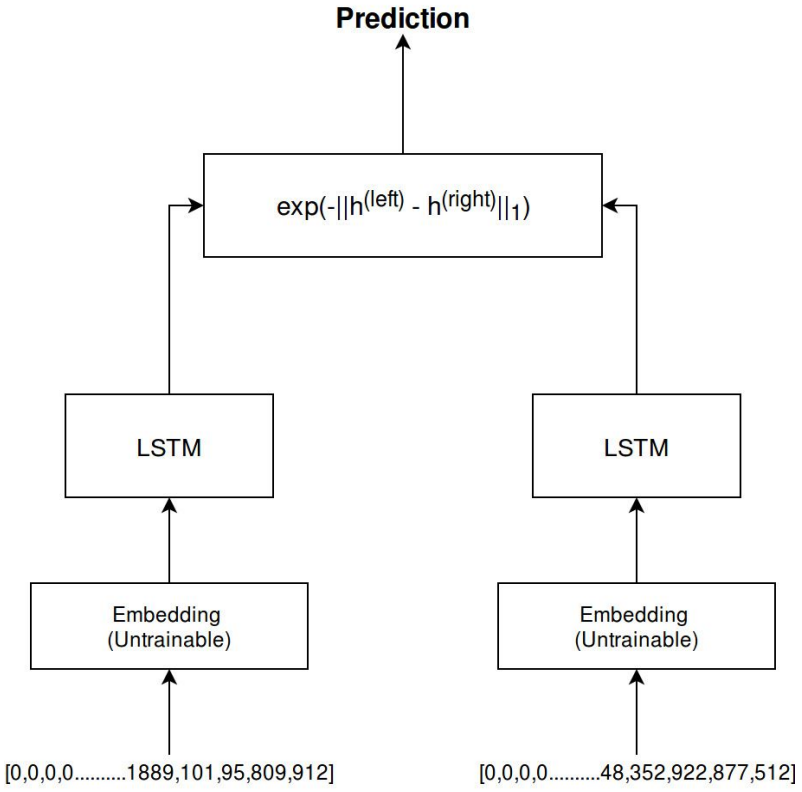
   Google's word2vec has been used to turn words to embeddings. It is a Vector Space Model (VSM) that maps similar words to nearby points. So, similar words are represented by numbers that are closer to each other and this provides the representation a semantic value to it. This is more powerful than representing words as unique ids with no relation between words with closer ids as helps in training models that leverage the semantics of words to create context. Vocabulary

dictionary stores word to index mapping and Inverse Vocabulary dictionary stores index to word mapping.

Embedding list is created for each query pair and it is padded with zeroes to accommodate the sentence with longest representation and make length of all embedding lists equal.

4. **MaLSTM**

Manhattan LSTM models has two networks $LSTM_{left}$ and $LSTM_{right}$ which process one of the sentences in a given pair independently. Siamese LSTM, a version of Manhattan LSTM where both $LSTM_{left}$ and $LSTM_{right}$ have same tied weights such that $LSTM_{left} = LSTM_{right}$. Such a model is useful for tasks like duplicate query detection and query ranking. Here, duplicate detection task is performed to find if two queries and duplicates or not. Similar model can be trained for query ranking using hit data for a given query and it's matching results as a proxy for similarity.



Our model uses an LSTM which reads word-vector representations of two queries and represents it in final hidden state. Similarity between the is used generated from the function below:

$$g(h^{(left)}_{T(left)}, h^{(right)}_{T(right)}) = \exp(-||h^{(left)}_{T(left)} - h^{(right)}_{T(right)}||_1)$$

The model has two visible input layers, one for each side of the MaLSTM. These are followed by two embedding layers on each size and LSTM model of Keras Functional API.

Training is run for 10 epochs due because the gain in accuracy per epoch begins to fall thereafter. Also training this model very resource intensive and it took over 11hrs on a machine with a 960m GPU. For deployment purposes the system can be trained for more epochs until the validation accuracy begin to stabilize (to avoid overfitting). The model is trained and the weights are stored for prediction tasks.

5. Similarity Prediction
Model is initialized and it's training weights from the previous step are loaded. Input pair of queries is converted into embedding lists (using 2 and 3) and are provided to the input layer of MaLSTM model. The representation from final hidden layer of both the sides are compared using Manhattan distance between them to get the similarity. Note that these scores are exponential and should not be inferred as percentages. Rather a value close to zero indicates that the queries are not likely to be duplicates and a value close to 1 means that the queries are very likely to be duplicates. Scores close to 0.5 show that model is not very confident in making the decision.

In case of queries run from GUI, this score is used to generate the output. Whereas to calculate model accuracy on test set the following procedure is applied.

6. Evaluation
Ground truth is compared with the predicted values and mean Squared Errors and Accuracy are obtained as measures of goodness. Confusion matrix is also obtained for the classifier.

Evaluation Strategy

The prediction models are evaluated using two measures of evaluation - cross-entropy and perplexity. The test dataset has four files each having five sentences. Two files have random sentences taken from the training corpus and thus are of the same domain as the training files. Two files have random sentences taken from the internet and hence in general do not have the same domain as the ones in the training corpus. The two measures are calculated for each testfile for each of the five smoothing methods implemented. Lower the values of these two evaluation measures, better is the performance of the smoothing method on the test cases (The meaning of these two measures is already explained in description for the evaluate method module). Hence it is expected that the methods will perform better (or have lower values of these measures) on the in-domain test files 1 and 2 compared to test files 3 and 4 where the values of these measures should be higher.

Semantic similarity is evaluated using Mean Squared Errors(MSE), Accuracy and Confusion Matrix for visualizing the prediction. Mean Squared Error is the average of the square of difference between predicted and actual values and gives the variance of an unbiased estimator. Accuracy tells us that given a pair of queries, how likely is the model to correctly classify it to the class it actually belongs to.

Experimental Results and Evaluation

1. Predictive Typing System

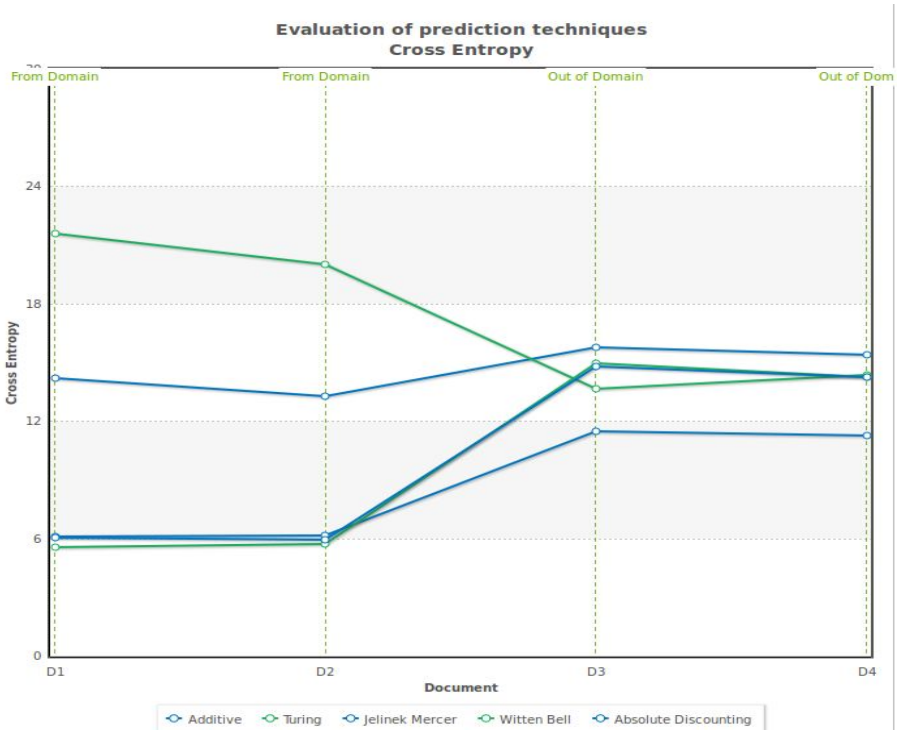
The values of the evaluation measures cross entropy and perplexity are given in the tables below:

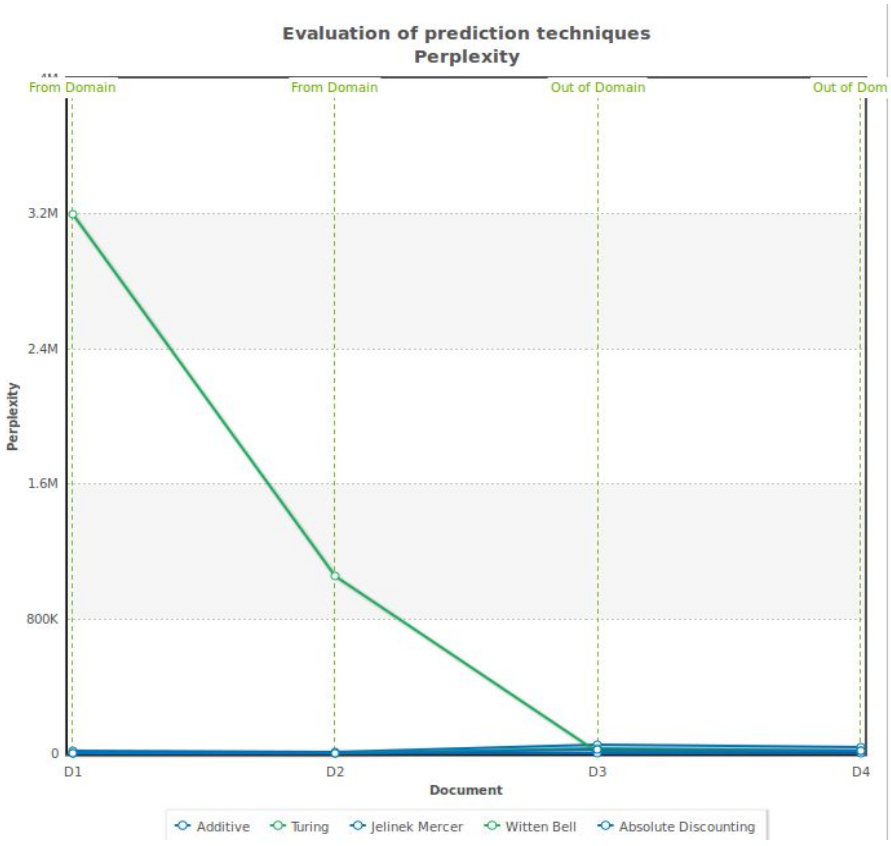
Table 1: Cross-entropy

	Test file 1	Test file 2	Test file 3	Test file 4
Additive smoothing	14.18	13.25	15.77	15.40
Good Turing	21.60	20.00	13.68	14.34
Jelinek-Mercer	6.09	6.16	11.49	11.25
Witten-Bell	5.57	5.74	14.95	14.25
Absolute Discounting	6.06	5.94	14.78	14.23

Table 2: Perplexity

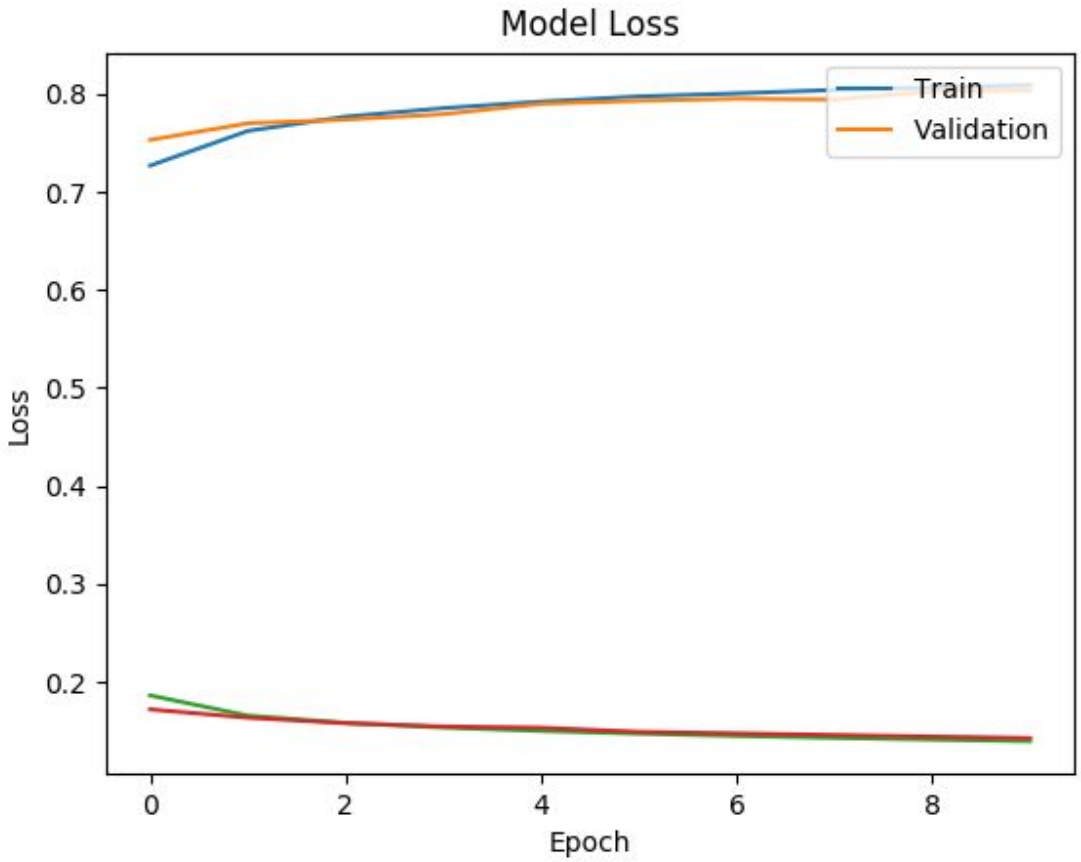
	Test file 1	Test file 2	Test file 3	Test file 4
Additive smoothing	18.68 x 10^3	9.76 x 10^3	55.91 x 10^3	43.26 x 10^3
Good Turing	3.19 x 10^6	1.04 x 10^6	13.13 x 10^3	20.76 x 10^3
Jelinek-Mercer	68.39	71.91	2.88 x 10^3	2.43 x 10^3
Witten-Bell	47.82	53.48	31.81 x 10^3	19.52 x 10^3
Absolute Discounting	67.02	61.66	28.29 x 10^3	19.23 x 10^3


Evaluation of prediction techniques
Cross Entropy
From Domain
From Domain
Out of Domain
Out of Dom
24
18
12
6
0
Cross Entropy
D1
D2
D3
D4
Document
Additive
Turing
Jelinek Mercer
Witten Bell
Absolute Discounting


Evaluation of prediction techniques
Perplexity
From Domain
From Domain
Out of Domain
Out of Dom
3.2M
2.4M
1.6M
800K
0
Perplexity
D1
D2
D3
D4
Document
Additive
Turing
Jelinek Mercer
Witten Bell
Absolute Discounting

2. Semantic Similarity Between Queries

The figure below shows Model Loss (MSE) and Accuracy on y-axis as a function of number of epochs on the x-axis.



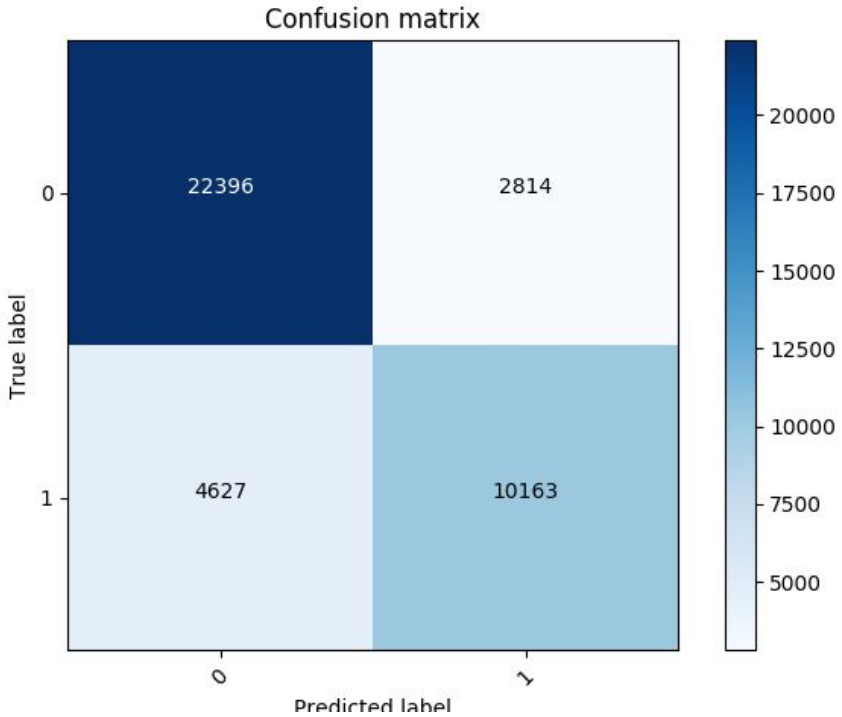
1. **Mean Squared Error (MSE)**
   The mean squared error is the same as loss measure of the model. Here, the lower two plots are for MSE. The best value of training MSE obtained is *0.1395* and it is found to be decreasing at a decreasing rate throughout the training process of the model. Whereas the validation MSE (which is equivalent to test MSE) is found to be *0.1426* at the end of ten epochs and follows same behavior as discussed above.

   It is worth noting that MSE for training set as initially worse off than that for validation set as the model is under fitted then. It approaches Validation MSE and then crosses it slightly indicating that the model is performing equally well on training and validation. Our model seems to be well-fitted as these values are very close.

2. **Accuracy**
   Validation accuracy which indicates the performance of model on unlabelled data is found to be *80.35%* at the end of 10 epochs. It increases at a decreasing rate as the model learns weights better after each epochs. Training accuracy reaches *80.88%* and follows a similar trend. There is

not much disparity between these two indicating a well fitted model.

3. **Confusion Matrix**



Confusion matrix plotted above shows the assignment of classes to sample pairs of questions by the model. Most of the predictions are concentrated in the diagonal classes indicating a good classification. Non duplicate questions is clearly the dominant class in the model.

Conclusion and Future Work

Semantic Similarity classifier based on Siamese LSTM model has given sufficiently good results on the Quora Question Pairs Dataset giving an accuracy of *80.35%* indicating its suitability for the task. This model can be trained on task specific datasets for application in various domains as a part of future research. Also hyperparameter optimization & changing optimizer can be done. Upcoming techniques like Reinforcement Learning, Transfer Learning, Augmentation using synonyms can be accommodated in the model for studying their impact on the model.

Reasonable predictions were obtained using the trigram model. Witten Bell performed the best on the training data followed by Absolute Discounting and Jelinek-Mercer. Additive smoothing and Good Turing did not give as good results. As expected, the evaluation metric values were higher for the out of domain test cases 3 and 4 compared to the in domain test cases 1 and 2, with Good Turing being the unexpected exception.

For future work, the model can be extended to use higher order n-grams like 4-grams or 5-grams and more smoothing techniques can be tested.
