Insincere question identification on Quora using Multi-ratio Sampling, LSTM and Capsules

Asif Iqbal
Department of Computer Science
Ryerson university
asif1.iqbal@ryerson.ca

Abstract

With the emergence of modern social question answering platforms like Quora, one of the headaches is to identify question that are insincere, or mocking in nature. In this age where tons of questions come in everyday, depending on human investigation for an exhaustive detection of such insincere questions is almost next to impossible. In this paper, we attempt to present an automated and scalable approach for detecting insincere questions using LSTM enabled recurrent neural networks. Insincere question also present us with the problem of data imbalance, since the number of such questions is pretty low compared to the sincere ones. We propose and attempt a novel approach for battling data imbalance by using multi-ratio random sampling. Our paper is an outcome of a an ongoing attempt of participating in this recent Kaggle competition Quora Insincere Questions Classification. We also investigate the use of Capsule layers on top of LSTM layers to see whether that might further improve detection performance, based on the fact that Capsules earned quite a bit of a success in one of of the past similar looking Kaggle competition on toxic comment detection. Like the competition suggests, we use F1 score as the success metric of our model.

1. Introduction

An existential problem for any major website today is how to handle toxic and divisive content [5]. *Quora* wants to tackle this problem head-on to keep their platform a place where users can feel safe sharing their knowledge with the world.

Quora [11] is a platform that empowers people to learn from each other. On *Quora*, people can ask questions and connect with others who contribute unique insights and quality answers. A key challenge is to weed out insincere questions – those founded upon false premises, or that intend to make a statement rather than look for helpful answers. As such, Kaggle recently launched a contest where they expect people to develop models that identify and flag insincere questions. To date, *Quora* has employed both machine learning and manual review to address this problem. With the results of this competition, they are looking to develop more scalable methods to detect toxic and misleading content. For the purpose of our term project, we have decided to leverage the problem and data from this competition. However, our main purpose is not to compete but rather to explore and experiment with some ideas.

Technically, the competition is a binary text classification problem where the task is to label a *Quora* question as sincere (0) or insincere (1). As can be easily imagined, the number of insincere questions are pretty low compared to the number of sincere questions in general, implying that this is an imbalanced classification problem [13]. This is also evident from the training data that Kaggle provided, where we see that only just over 6% of the total questions are labelled as insincere, and the rest vast majority are labelled as sincere. Because of this imbalance, mere accuracy is not a good metric for the classification performance. We need to consider metrics like precision, recall and F1 score. Kaggle puts the F1 score (for the insincere class) as their evaluation metric for the competition participants.

Another big issue here is the volume of data. The competition itself provides almost 1.3 million questions for training, in real life which could be bigger by several orders of magnitude. For the work being done in this project, we don't really have access to a very high end machine (especially one with GPU support), and ideally even if we do, we need some scalable approach to tackle such huge volume of data. Given that our goal is to increase F1 score, we propose a novel method that tries to combat the huge volume and data imbalance problem simultaneously - we first divide the training data into a training and validation portion equally and maintaining the class ratios using stratified sampling. Then we produce 3 random samples sampled from 3 different regions of the training data containing 3 different ratios of the sincere over insincere questions. In a real scenario, this number can be easily generalized beyond 3. Our aim is to train our models on these 3 different class-ratio containing training sets separately and then take an ensemble results of the model. We also validate our model on 5 different validation sets taken from 5 different regions of the validation data we had sampled earlier.

As for choosing our training model, a myriad of techniques have been used over the years to classify texts ranging from classical machine learning techniques like logistic regression, SVM, random forests, bagging, boosting to modern deep learning techniques including RNN [4] along with LSTM [3] based networks. As has been seen in many Kaggle competitions and practical applications, an ensemble of various of these techniques can often earn better results. For the purpose of this project, however, due to time and resource constraints, we don't plan to use an ensemble of a lot of these techniques. We limit our approach to exploring relatively shallow neural networks with a bidirectional LSTM layer only. Also, inspired by the recent success of capsule networks [12] in some of the NLP based competitions in the recent past, we also plan to explore it to see whether we can gain any improvement over LSTMs.

The rest of the paper is organized as follows:

2. Related Work

Tons of work have been done in the area of text classification to date. We will not go into a detailed literary review of these works in this term project paper. Recent work like [9], [8], [16] show the effectiveness of RNN and LSTM for text classification. There has not been a significant amount of work in academia with specific emphasis on toxic or insincere content detection. An early competition from Kaggle [6] resulted in lots of people sharing their kernels using various techniques for toxicity level detection from social media comments. Our work is also partly motivated by this competition.

A few works on text classification using capsule networks [12] have also taken place in the meantime. These include work like [15] and [7].

As for dealing with data imbalance, several approaches such as described in [13] have been proposed and used to date, one of the most prominent being SMOTE [1].

However, for the purpose of this term project, we are not trying to implement such sophisticated approaches since we are limited by time and resource constraints. We have come up with a much simpler yet slightly novel approach of dealing with imbalance.

In the next sections, we first give a brief overview of the data, followed by our novel sampling and split strategy.

3. Data and Exploratory Analysis

The training data provided by Kaggle has 1306122 records with 3 columns - **qid**, **question_text** and **target**. Of these only 80810 records (approximately 6%) are labelled insincere and there rest 1225312 are labelled sincere as can be seen in Figure 1.

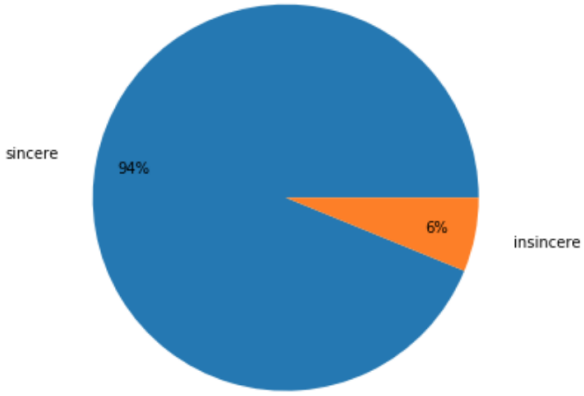


Figure 1: Class Distribution

As per the competition [5], nn insincere question

is defined as a question intended to make a statement rather than look for helpful answers. Some characteristics that can signify that a question is insincere:

- Has a non-neutral tone
- Has an exaggerated tone to underscore a point about a group of people
- Is rhetorical and meant to imply a statement about a group of people
- Is disparaging or inflammatory
- Suggests a discriminatory idea against a protected class of people, or seeks confirmation of a stereotype
- Makes disparaging attacks/insults against a specific person or group of people
- Based on an outlandish premise about a group of people
- Disparages against a characteristic that is not fixable and not measurable
- Isn't grounded in reality
- Based on false information, or contains absurd assumptions
- Uses sexual content (incest, bestiality, paedophilia) for shock value, and not to seek genuine answers

Kaggle also mentions that the distribution of questions in the dataset should not be taken to be representative of the distribution of questions asked on Quora. This is, in part, because of the combination of sampling procedures and sanitization measures that have been applied to the final dataset.

Given that we have over a million records in the training data, our time and resource constraints (especially lack of access to a GPU server) will not allow us to experiment models using RNN and LSTM for the purpose of this project. Along with this, there is also the general problem of scalability. That's why we decide to resort on a slightly novel approach which we call *multi-ratio sampling* for training our model on. In the next section we will in detail how we perform the multi-ratio sampling.

4. Multi-ratio Sampling

We first shuffle and divide our dataset into 2 equal portions - one for using in training and the other for validation and testing purpose. We then divide the training portion into 3 equal segments (each approximately 16.67% of the full data), and from each of these segments we take a random sample. However, the ration of sincere vs insincere instances are different in these samples. The first one mimics the actual distribution ratio. For the second one we take a class-balanced sample maintaining a 1:1 ratio of sincere and insincere instances. Finally for the third one, we take a 2:1 ratio for sincere instances against insincere ones. These 3 samples are the ones that we finally end up using as training sets. The division and sampling have been illustrated in Figure 2. We hope that training on different ratio containing datasets and ensembling the results might get us some better predictions. Notice that in the end, each of our training sets (samples) is approximately 1.67% of the original data, each containing 21768 records.

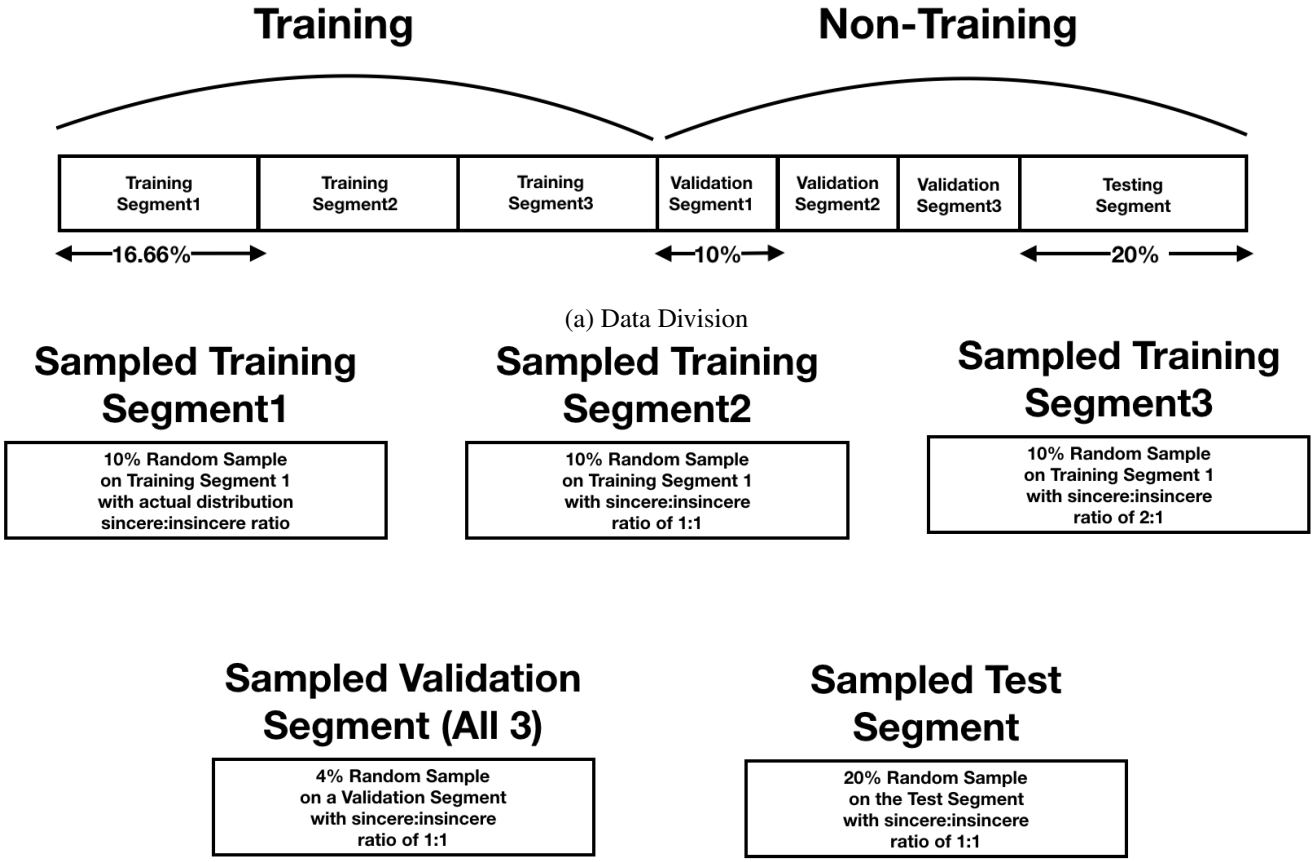
For validation and testing, we divide the other half of the data in 4 segments, the first 3 as validation segments with 20% each (10% of the total data), and the rest as test segment with 40%(20% of the total data). We take a 4% random sample on each of the validation segments, and a 2% random sample on the test segment. So each of the validation sets end up having approximately 5225 records each and the test set end up having 52245 records.

Having discussed the data distribution and split, let's look at some of the preprocessing of the data that we perform.

5. Preprocessing

In a typical NLP problem, the kind of preprocessing that are usually done are stop words removal, tokenization, stemming, lemmatization, word frequency/TF-IDF calculation, n-grams generation and the use of word embeddings.

We found after experimentation that things like stemming and lemmatization don't really help too much in our case, and sometimes rather decreases the performance. Also, for training our LSTM and Capsule models, to keep things relatively simple, we don't make use of word frequency, TF-IDF or n-grams. We,



(a) Data Division

(b) Multi-ratio Sampling

Figure 2: Partitioning into training (multi-ratio), validation and test sets

however, do make use of word embeddings, in particular the Glove [10] embeddings.

5.1. Use of Word Embeddings

The use of embeddings help us represent words in much more compact dimensions [2]. Whereas the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are low dimensional floating-point vectors (that is, dense vectors, as opposed to sparse vectors); as you can see in Figure 3. Unlike the word vectors obtained via one-hot encoding, word embeddings are learned from data. In our case, one-hot encoding words lead to vectors that are 139,962-dimensional or greater, which leads to a gigantic sparse matrix.

Another advantage of using embeddings is that the geometric relationship between words and vectors in a well-built embedding space capture the semantic relationship between these words, and specific *directions* in the embedding space is meaningful. In figure 4 (courtesy of [2]), four words are embedded on a 2D plane: *cat*, *dog*, *wolf*, and *tiger*. With the vector representations shown here, some semantic relationships between these words can be encoded as geometric transformations. For instance, the same vector allows us to go from *cat* to *tiger* and from *dog* to *wolf* : this vector could be interpreted as the ”from pet to wild animal” vector. Similarly, another vector lets us go from dog to cat and from wolf to tiger, which could be interpreted as a ”from canine to feline” vector.

One option for choosing these embeddings is to let our model learn it’s own embedding for our case. While that would be something interesting to explore,

to keep things simple, we decide to use the pretrained Glove [10] 300-dimensional embedding vectors from Stanford.

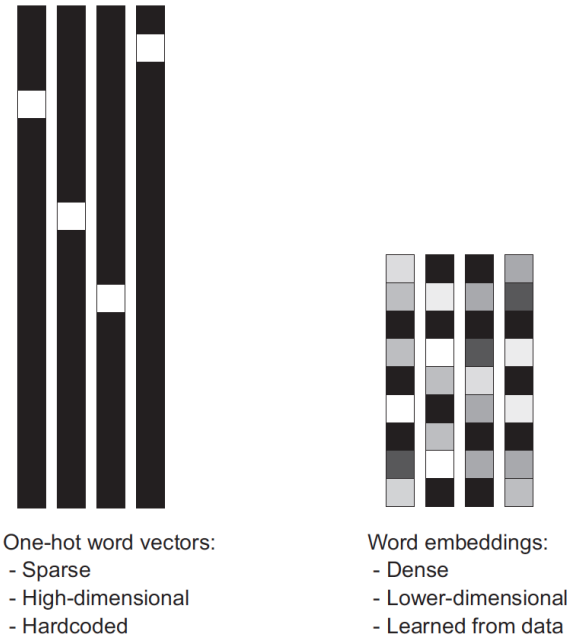


Figure 3: Benefits of using word embeddings

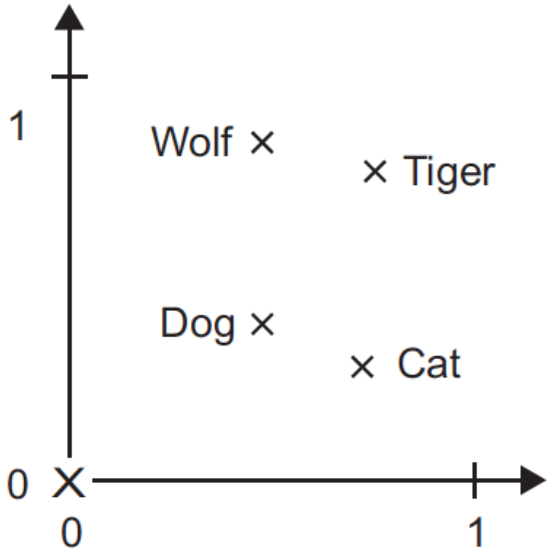


Figure 4: Semantic Relationship between Words in Embedding Space

So to summarize, we go through the following preprocessing steps before feeding the data to our model.

- From each instance we remove stop words
- We tokenize the sentences into words
- At this point each word is represented as a one-hot encoded vector of word indices. Each vector size is (maximum word index + 1)
- We transform the words into reduced dimensioned vectors with semantic relationships. This is where the Glove embeddings comes into play.
- We keep the first 100 words fro each sentence and use padding if required.
- Later we use these embeddings as an Embedding layer in our LSTM and Capsule model so that this layer transforms our sentences into sentence vectors using the word embeddings before feeding into the LSTM layer (discussed later).

In the next section we discuss the architecture of our LSTM based neural predictive model.

6. Model Description

Although ideally we would like to use a reasonably deep architecture for the neural model, for resource and time constraints, we use a relatively shallow network. Since RNNs with LSTMs have had good success over the years in text classification and NLP, we chose to go with these.

6.1. RNN and LSTM

We assume the reader has familiarity with RNNs and LSTMs, so we don't go into the details of how they work. The details can be found in [3]. The motivation is that when we read a sentence, we processing it word by word—while keeping memories of what came before; this gives us a fluid representation of the meaning conveyed by this sentence [2]. Biological intelligence processes information incrementally while maintaining an internal model of what it is processing, built from past information and constantly updated as new information comes in. This is what a recurrent neural network (RNN) tries to accomplish (see Figure 5a). However, because of the vanishing gradient problem the information carry over gradually decays [4], and as such vanilla RNNs are not practically that useful. The LSTM (Long Short Term Memory) [3] solves

this problem by saving information for later, thus preventing older signals from gradually vanishing during processing (see Figure 5b).

We also decide to use bidirectional LSTM [2] since a bidirectional LSTM would exploit the order sensitivity of the input (in our case, the words in a sentence). It consists of using two regular LSTM layers, such as the each of which processes the input sequence in one direction (chronologically and antichronologically), and then merging their representations. By processing a sequence both ways, a bidirectional LSTM layer can catch patterns that may be overlooked by a unidirectional LSTM layer.

6.2. Basic Model Architecture

Having talked about LSTM a bit, let's describe the actual model architecture we use. At a high level our model consists of:

- The embedding layer that has the pretrained Glove embeddings. This layer is not learnt by our process and is responsible for taking the sentence vectors (consisting of the one-hot encoded word vectors) of size 100×139962 and converting them to compact vector of size 100×300, since 300 is the word embedding space dimension. At this point each word is represented as a one-hot encoded vector of word indices. Each vector size is (maximum word index + 1). We transform the words into reduced dimensioned vectors with semantic relationships. This is where the Glove embeddings comes into play. We keep the first 100 words for each sentence and use padding if required. Later we use these embeddings as an Embedding layer in our LSTM and Capsule model so that this layer transforms our sentences into sentence vectors using the word embeddings before feeding into the LSTM layer (discussed later).
- A bidirectional LSTM layer, with 100 steps since we have 100 words in each sentence.
- Three fully connected layers. The first first 2 with 1024 hidden units and Relu activation, and the last one is just 1 output unit with a sigmoid activation, since this is binary classification problem.

The architecture of the model has been shown in Figure 6.

6.3. Capsule Model Architecture

As we mentioned earlier, inspired by works like [15] and also the success of applying capsule enabled networks in recent Kaggle competitions [6], we also decide to experiment with addition a capsule layer to our basic model. However, since the capsule is a relatively new concept, we first go over an overview of how they work.

6.3.1 Capsule Networks

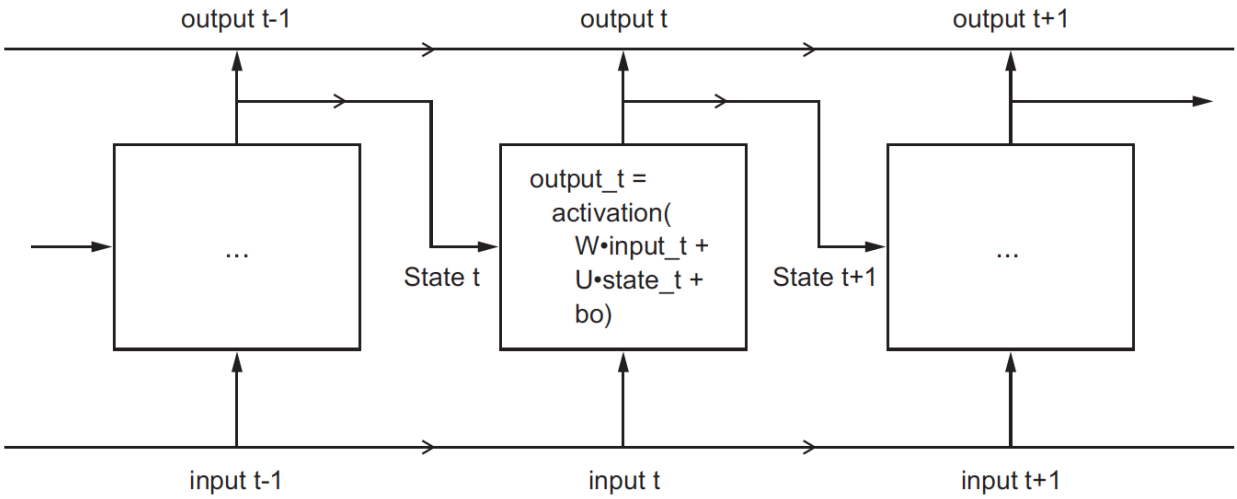
Although we are dealing with a NLP problem in this paper, it is important to understand that the Capsule concept originated in the context of Computer Vision and image classification, supposedly as a better alternative to Convolutional Neural Networks (CNN). We will first elaborate capsules in the context of images just for clarification. The description below has been mostly taken from [14].

Any real object is made up of several smaller objects. For example, a tree consists of a trunk, a crown and roots. These parts form a hierarchy. The crown of a tree further consists of branches and branches have leaves (Figure 7).

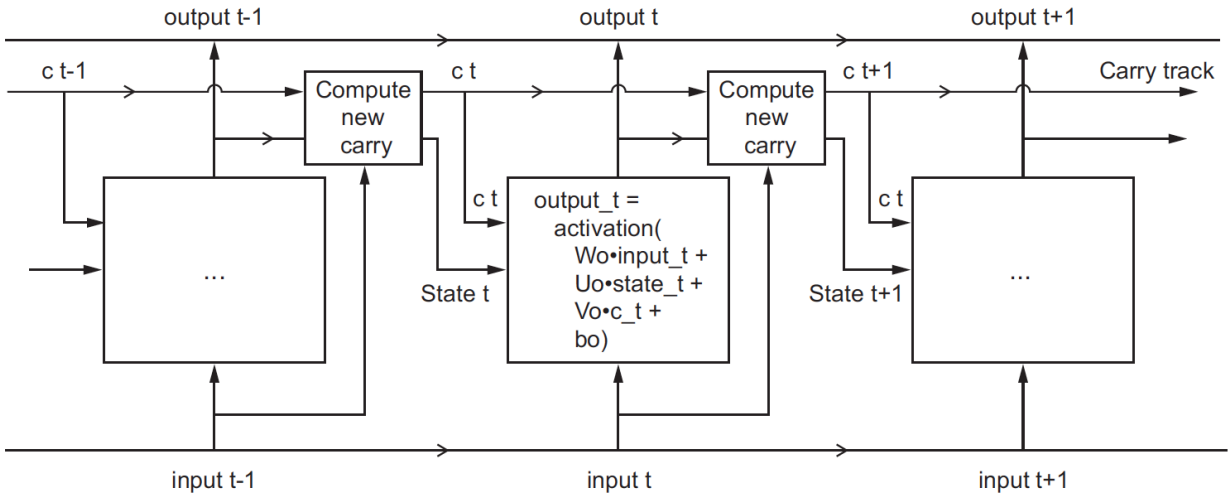
Whenever we see some object, our eyes make some fixation points and the relative positions and natures of these fixation points help our brain in recognizing that object. By doing so, our brain does not have to process every detail. Just by seeing some leaves and branches, our brain recognizes there is a crown of a tree. And the crown is standing on a trunk below which are some roots. Combining this hierarchical information, our brain knows that there is a tree. From now on, we will call the parts of the objects as entities. he assumption behind CapsNet is that there are capsules (groups of neurons) that tell whether certain objects (entities) are present in an image. Corresponding to each entity, there is a capsule which gives:

- The probability that the entity exists
- The **instantiation parameters** of that entity

Instantiation parameters are the properties of that entity in an image (like "position", "size", "position", "hue", etc). For example, a rectangle is a simple geometric object. The capsule corresponding to a rectangle will tell us about its instantiation parameters.



(a) RNN



(b) LSTM

Figure 5: RNN and LSTM

Like from Figure 9, our imaginary capsule consists of 6 neurons each corresponding to some property of the rectangle. The length of this vector will give us the probability of the presence of a rectangle. So, the probability that a rectangle is present will be:

$$\sqrt{1.3^2 + 0.6^2 + 7.4^2 + 6.5^2 + 0.5^2 + 1.4^2} = 10.06$$

However, since this is a probability we are talking about, we need to normalize this value into range between 0 and 1. This is done via a non-linear transformation is called **squashing function** and it serves as an activation function for capsule networks (just like ReLU is used in CNNs).

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

A CapsNet consists of several layers [14]. Capsules in the lower layer correspond to simple entities (like rectangles, triangles, circles, etc). These low-level capsules bet on the presence of more complex entities and their bets are "combined" to get the output of high-level capsules (doors, windows, etc). For ex-

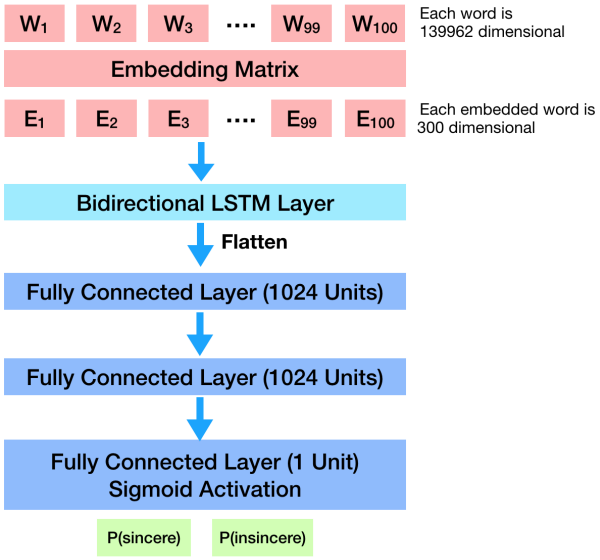


Figure 6: Model Architecture

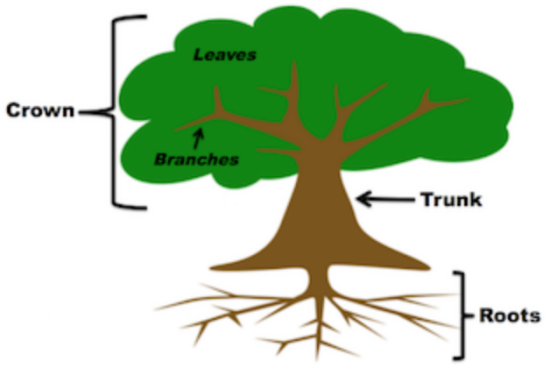


Figure 7: Object Hierarchy

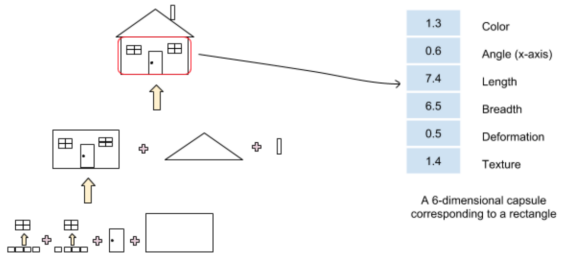


Figure 8: Rectangle Capsule

ample, the presence of a rectangle (angle with x-axis = 0, size = 5, position = 0,...) and a triangle (angle with x-axis = 6, size = 5, position = 5,...) work together to bet on the presence of a house (a higher-level entity).

There is a coupling effect too. When some low-level capsules agree on the presence of a high-level entity, the high-level capsule corresponding to that entity sends a feedback to these low-level capsules which increases their bet on that high-level capsule. To understand this, let's assume we have two levels of capsules:

- Lower level corresponds to rectangles, triangles and circles
- High level corresponds to houses, boats, and cars

If there is an image of a house, the capsules corresponding to rectangles and triangles will have large activation vectors. Their relative positions (coded in their instantiation parameters) will bet on the presence of high-level objects. Since they will agree on the presence of house, the output vector of the house capsule will become large. This, in turn, will make the predictions by the rectangle and the triangle capsules larger. This cycle will repeat 4-5 times after which the bets on the presence of a house will be considerably larger than the bets on the presence of a boat or a car. In the terminology of the original authors [12], this algorithm is called **Dynamic Routing by Agreement**.

We skip the detail mathematical elaboration of Capsule networks and for that we refer the reader to [12] and [14], but in short the advantage of CapsNet compared to CNN is that in a CNN, there are pooling layers. We generally use MaxPool which is a very primitive type of routing mechanism. The most active feature in a local pool (say 4x4 grid) is routed to the higher layer and the higher-level detectors don't have a say in the routing. Compare this with the routing-by-agreement mechanism introduced in the CapsNet. Only those features that agree with high-level detectors are routed. This is the advantage of CapsNet over CNN. It has a superior dynamic routing mechanism (dynamic because the information to be routed is determined in real time) [14].

6.4. Model Architecture with Capsules

Having given an overview of how capsules are motivated for images, we can get a rough idea about how capsules could possibly be used for NLP problems like text classification as well. As we pointed out earlier, this practice has already been started in works like

[15] and [7]. We can hypothesize the use of capsules in text classification as follows: lower level capsules might be representative of various aspect of a word, like its vector position in the embedding space, its relative position within a sentence etc., whereas higher level capsules might be representative of similar features multi-word phrases. However, for the purpose of this project, we are mainly interested in exploring Capsule networks to see whether they gain us better performance, and not particularly in the interpretation of what they might be doing for our texts under the hood.

The architecture with capsule looks very similar to our previous LSTM architecture discussed in 6.3, except that now we will add Capsule layers after the LSTM layer, as shown in . We basically have 2 internal Capsule layers here, one as a lower level capsule layer and the other as a higher level one.

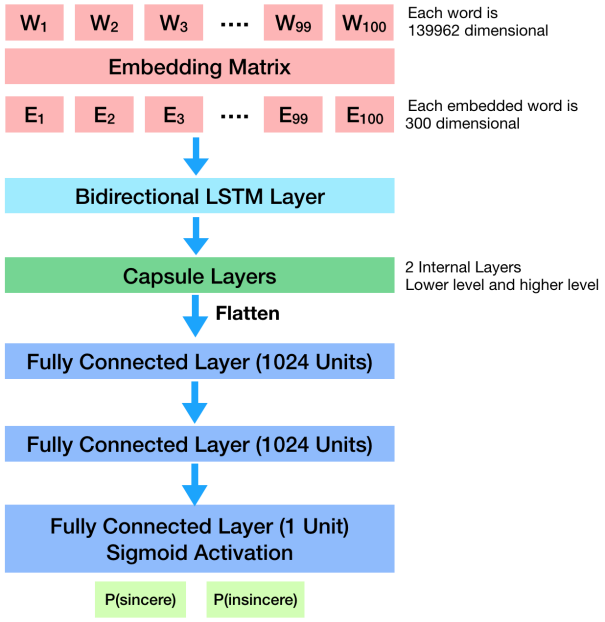


Figure 9: Model Architecture with Capsules

7. Experiments

Although ideally this sort of models should be run on a GPU enabled server and with the full training dataset available, for time constraints and lack of access to a GPU server, we have taken smaller samples from the training data (as discussed in section 4) and we run it on a Quad-core 2.7 GHz Intel i7 machine with 16GB of memory. For our implementation we use **Keras** with **Tensorflow** backend. We choose Keras since it provides high level API classes and methods to quickly prototype and run a model. For both the LSTM-only and LSTM with Capsule models, we use an Adam optimizer with learning rate 0.001, and a batch size of 32.

Before going into the details of our actual experiments, we will discuss in brief a quick pre-experimentation we did with simple Logistic Regression. We did this just to set some sort of baseline for ourselves, and to validate our assumptions before moving forward with the complex LSTM and Capsule models.

7.1. Pre-experiment with Logistic Regression

For this, we build simple logistic regression models based on two sorts of feature vectors - TF-IDF and word counts. Table 1a and 1b show the F1 scores obtained for the validation sets for TF-IDF and word count, respectively.

	Validation Set 1	**Validation Set 2**	**Validation Set 3**
Training Set 1 sincere:insincere=94:6	0.09	0.04	0.04
Training Set 2 sincere:insincere=1:1	0.47	0.44	0.42
Training Set 3 sincere:insincere=2:1	0.51	0.53	0.49

(a) TF-IDF vectors

	Validation Set 1	**Validation Set 2**	**Validation Set 3**
Training Set 1 sincere:insincere=94:6	0.33	0.36	0.36
Training Set 2 sincere:insincere=1:1	0.49	0.48	0.48
Training Set 3 sincere:insincere=2:1	0.56	0.53	0.51

(b) Word count vectors

Table 1: Logistic regression F1 scores on validation sets obtained for 3 different training sets

To summarize, it looks like it looks like for models trained on training set 1 (actual sincere:insicere ratio), the F1 performance of logistic regression is very poor. Training set 3 (with 2:1 ratio) gains us the best performance, while training set 2 (1:1 ratio) gets us better performance than training set 1 but slightly worse than

training set 3. This is the pre-experimentation that motivated us to use the multi-ratio sampling we have been mentioning all along.

Now let's discuss the experimentation of our actual LSTM and Capsule models.

7.2. LSTM-only model

As mentioned earlier, it is the F1 score we are particularly interested in. We train our model for 10 epochs (and once again, ideally this should be run for a larger number of epochs, but for time constraints we are limiting ourselves to just 10) using 3 different training sets as we discussed in 4 and every epoch we monitor the performance on each of the 3 validation sets. Figure 10 shows the F1 scores for the epochs for the training and validation data sets.

It looks like that for the first training set (that has the similar 94:6 sincere-insincere ratio as the full data), upto epoch 7, the validation F1 trend somewhat follows the training F1 trend (although the values are lower), however after that they start diverging wildly, meaning that at that point the model started overfitting. For the second training set (with 1:1 sincere-insincere ratio), the training F1 is pretty high from the beginning, however the validation, even though shows a very slight upward trend, effectively stalls around the 45-50% mark. A somewhat similar behaviour is noticed for the third training set (with 2:1 sincere-insincere ratio), even though the rise in validation F1 is slightly better than the second case. Also, we notice that even though we ran for 10 epochs, the epoch showing the best F1 scores for the validation sets are epochs 7, 2 and 7 for the training sets 1, 2 and 3, respectively. Table 2a shows the F1 scores obtained for the validation sets for these best epochs.

It turns out that the model performance achieved with the insincere-oversampled training sets are not really better than the the performance achieved with the training set that has the actual (94:6) ratio. In fact, in the 1:1 ratio case, the F1 performance is worst, and in the 2:1 ratio case, the performance is slightly better that 1:1 but still worse that the actual ratio performance. So the multi-ratio sampling did not help too much here. This behaviour differs quite a bit from what we saw in the logistic regression case above.

	Validation Set 1	**Validation Set 2**	**Validation Set 3**
Training Set 1 sincere:insincere=94:6	0.60	0.58	0.57
Training Set 2 sincere:insincere=1:1	0.52	0.51	0.48
Training Set 3 sincere:insincere=2:1	0.56	0.55	0.51

(a) LSTM-only model

	Validation Set 1	**Validation Set 2**	**Validation Set 3**
Training Set 1 sincere:insincere=94:6	0.57	0.56	0.55
Training Set 2 sincere:insincere=1:1	0.53	0.51	0.49
Training Set 3 sincere:insincere=2:1	0.59	0.59	0.56

(b) LSTM with Capsule model

Table 2: Best F1 scores on validation sets obtained for 3 different training sets

7.3. LSTM with Capsule model

We perform similar experiments with the Capsule enabled LSTM model. The trends of training F1 and validation F1 are shown in Figure 11. It looks like that although the overall trend of the training vs validation F1 scores is somewhat similar to the trend we saw in the LSTM-only case, the overfitting tendency is a little lower with capsules. For example, for training set 1, we see that the validation trend (for all 3 validation sets) is following the training trend till the last epoch (although with a lower score), meaning that they haven't started diverging till then. For training sets 2 and 3 though, there hasn't been too much improvement in the trend compared to the LSTM-only case, but we do notice that we are getting a slightly higher F1 scores in these cases as discussed below. Table 2b shows the F1 scores obtained for the validation sets for the best epochs, which are epoch number 9, 7 and 9, respectively for the 3 training sets.

We notice that unlike the LSTM only case, now we have a slightly better F1 performance achieved from the 2:1 ratio training sample than what we gain with the actual ratio training sample. Although the 1:1 ratio training sample still gains us slightly lower performance than the actual ratio one, we see that the validation F1 scores achieved with training sets 2 and

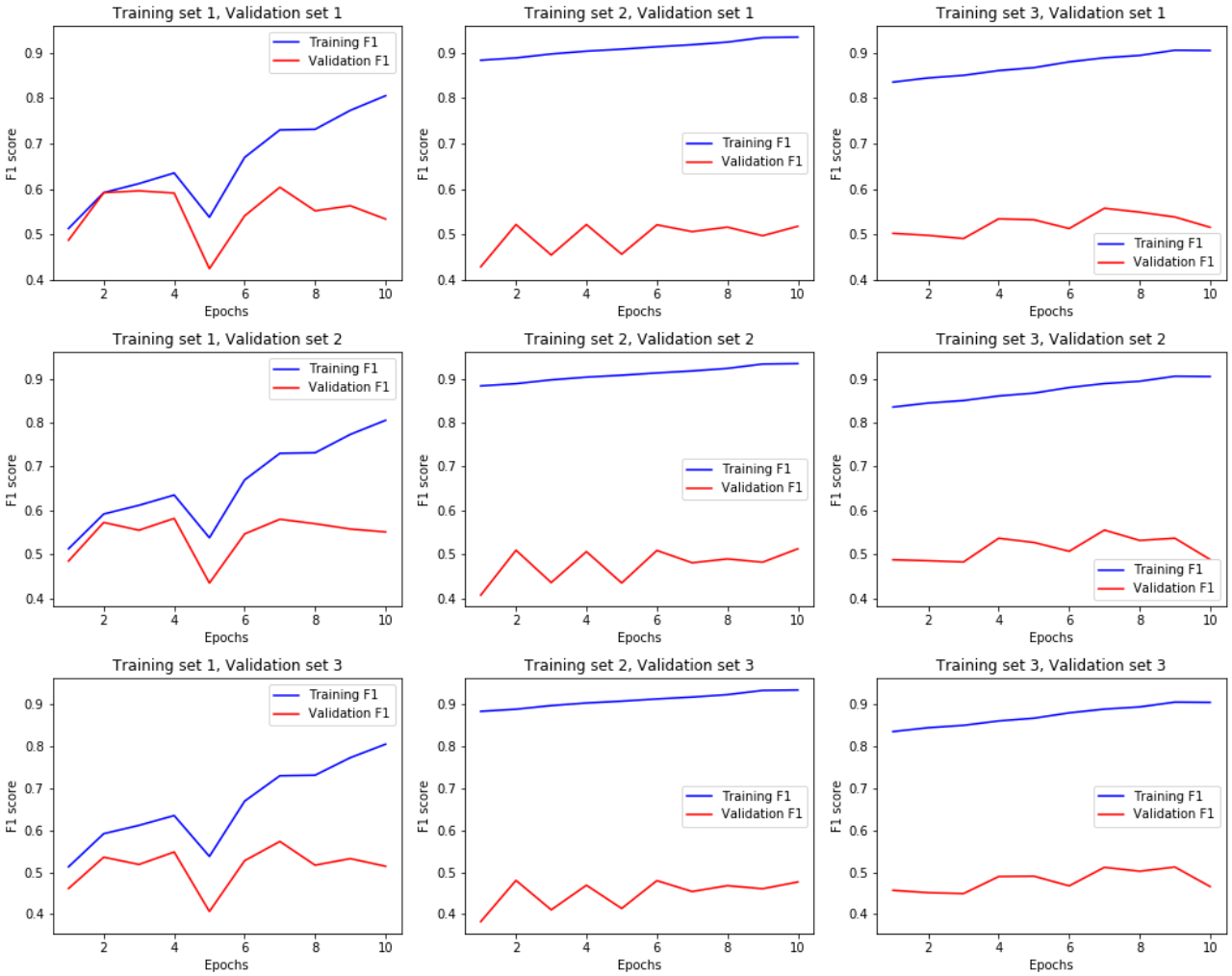


Figure 10: Training and Valdation F1 Score Performance for LSTM-only model

3 have caught up with the validation scorer achieved with training set 1. So unlike the LSTM-only case, the multi-ratio sampling did seem to help here.

7.4. Performance on Test Data

We perform testing from in two different ways - one is on the test dataset (having 52,245 records) we made for ourselves sampling from the training data provided by Kaggle, as discussed in section 4, while the other is in the wild when we submit our work to Kaggle and Kaggle performs the evaluation on their original test data.

For our own testing, we first predict the classes (sincere or insincere) using all 3 LSTM-only models (trained on 3 different training dataset) and then take a maximum-voted ensemble result on those for the final prediction. We do the same for the Capsule enabled models as well.

On our self created test data, we get an F1 score of **0.548** using the LSTM-only models and an F1 score of **0.546** using the Capsule enabled models. We should point out that these scores were obtained with the versions of the models after all 10 epochs, and not with the best epoch versions of the model. Due to time constraints, we avoided saving checkpoints of our models which would allow us to take the model version for the best epoch. We anticipate that if we are able to use the best epoch model version, we could possibly get higher F1 scores.

Figure 12 shows the unnormalized and normalized

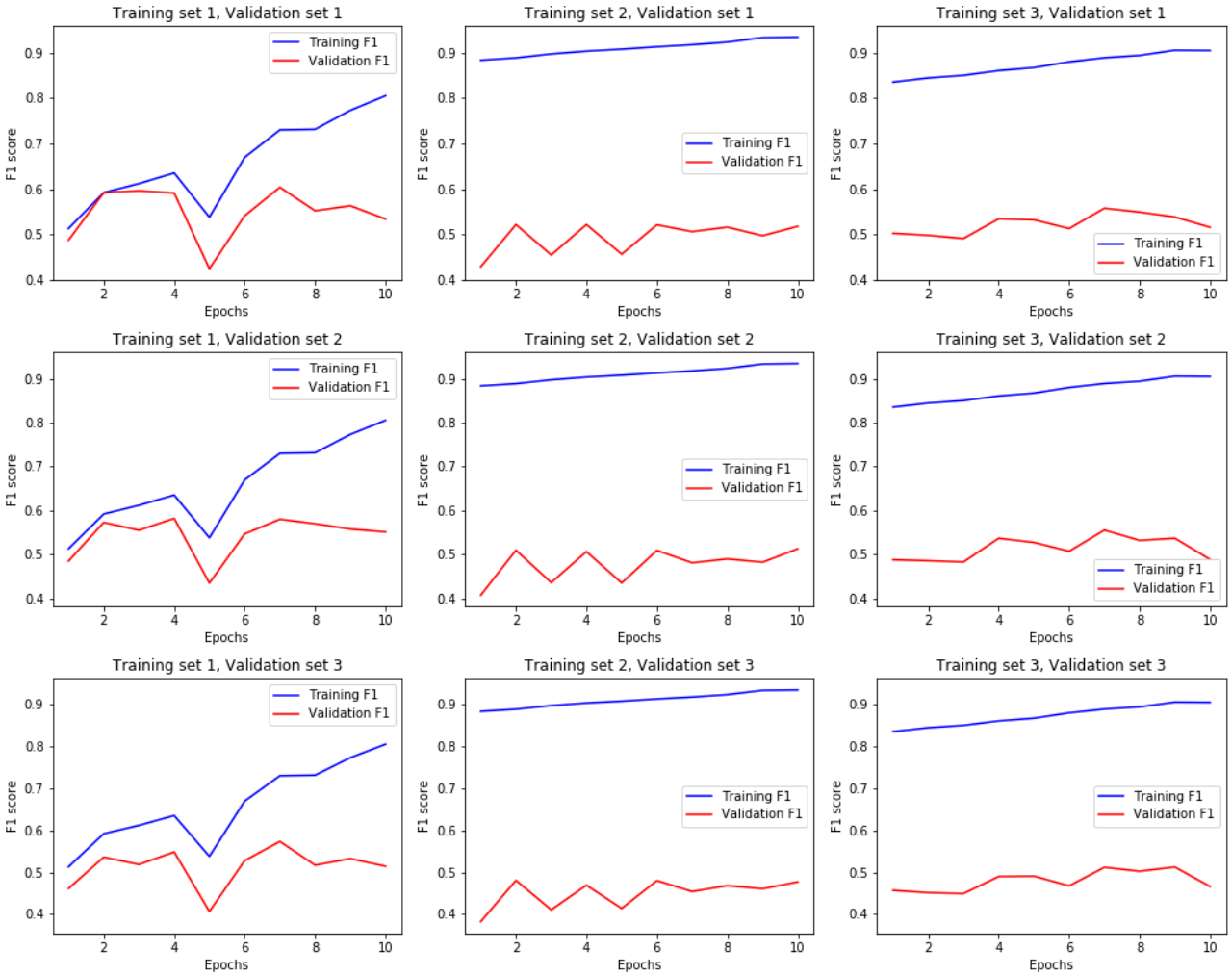


Figure 11: Training and Valdation F1 Score Performance for LSTM with Capsule model

version of the confusion matrix on our test data, while Table 3 shows the classification report. It looks like the main performance bottleneck is caused by the lack of precision of the insincere class. We have a huge number of sincere instances and misclassification of even a small percentage of that (9% in this case) causes the precision of the insincere class to go down drastically and the F1 score follows that as well.

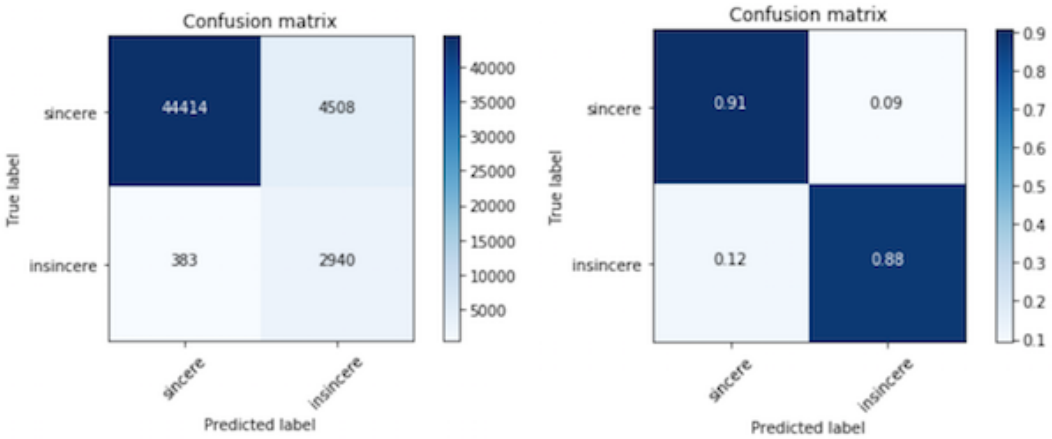
	Precision	**Recall**	**F1**
Sincere	0.99	0.91	0.95
Insincere	0.39	0.88	0.55

Table 3: Classification report on our test data

As for the real Kaggle test data, we have been able to make one submission with LSTM-only model and that got us an F1 score of **0.53** using the LSTM-only model, which is comparable to our own test scores of **0.548**. Also, with the LSTM with Capsule model, we made one submission and got an F1 score of and **0.528** which is also comparable to our own test score of **0.546**. Table 4 summarizes these scores. Note again that these submissions were done with the trained models after 10 epochs and not with the best checkpoints.

	LSTM only	**LSTM with Capsule**
Our Test Set	0.548	0.546
Kaggle Test Set	0.53	0.528

Table 4: F1 scores on test data



(a) Unnormalized (b) Normalized

Figure 12: Confusion Matrix

7.5. Experiment Retrospective

It is to be noted that we performed our training and evaluation on very small subsets of the total data provided by Kaggle, because of time and resource constraints. To recap, we used only just over 4% of the data for training our models. However, we do notice that even using such a small percentage of data for our training, we were able to achieve an F1 score of **0.53** on Kaggle's actual test data which is on par with the F1 scores we got with our small validation sets. This means that our models are showing good generalization property. The highest F1 score on Kaggle to date seems to be **0.707**, so our models are clearly underfitting, but the generalization property hints a promising direction.

We noticed that while multi-ratio sampling looked like a promising direction after doing our pre-experimentation with logistic regression, it didn't seem to be that helpful for the LSTM-only model, then again it proved to be helpful in the LSTM with Capsule model. In short, this sort of sampling is definitely worth a try.

Finally, there comes the question of sample size, model architecture and hyperparameter tuning. Because of our time and resource constraints, we could not perform a whole bunch of tunings and ablations. However, below are some of the tunings and ablations that we could definitely perform in the future to improve performance.

- Increasing training and validation sample size
- Increasing number of layers (depth) in our model
- Stacking more LSTM and Capsule layers one after the other
- Performing more preprocessing on the data, possibly deleting words that are too frequent or too rare
- Adapting learning rate with time,
- Increasing the batch size

8. Conclusion

In this paper, we have shown an attempt to identify insincere questions on *Quora* using LSTM and Capsule based models and using a multi-ratio sampling approach. The applicability of the approach is not limited to *Quora* by any means. It can be applied to any text classification problem where we have a large imbalance between the classes. We have also shown that even using a very small portion of the full training data, we have been able got a F1 score that generalizes well to the actual wild test data. We believe that with proper tuning and ablations, our approach can be extended to perform much better.

Acknowledgements Dr. Neil Bruce and Francois Chollet.

References

[1] N. V. Chawla et al. *SMOTE: Synthetic Minority Over-sampling Technique*. Journal Of Artificial Intelligence Research, Volume 16, pages 321-357. 2002.

[2] Francois Chollet. *Deep Learning With Python*. Manning Publications. 2018.

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[4] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1999. ISBN: 0849371813.

[5] Kaggle. *Quora Insincere Questions Classification*. 2018. URL: https://www.kaggle.com/c/quora-insincere-questions-classification.

[6] Kaggle. *Toxic Comment Classification Challenge*. 2018. URL: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

[7] Jaeyoung Kim et al. *Text Classification using Capsules*. 2018. URL: https://arxiv.org/abs/1808.03976.

[8] Ji Young Lee and Franck Dernoncourt. *Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks*. NAACL. 2016.

[9] SPengfei Liu, Xipeng Qiu, and Xuanjing Huang. *Recurrent Neural Network for Text Classification with Multi-Task Learning*. IJCAI. 2016.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. 2014. URL: https://nlp.stanford.edu/projects/glove/.

[11] *Quora*. URL: https://www.quora.com.

[12] Sara Sabour, Nicholas Frosst, and Geoffrey Hinton. *Dynamic Routing between Capsules*. Computer Vision and Pattern Recognition. 2017.

[13] Upasana. *How to handle Imbalanced Classification Problems in machine learning?* 2017. URL: https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/.

[14] Zafar. *Beginner's Guide to Capsule Networks*. 2018. URL: https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-capsule-networks.

[15] Wei Zhao et al. *Investigating Capsule Networks with Dynamic Routing for Text Classification*. 2018. URL: https://arxiv.org/abs/1804.00538.

[16] Chunting Zhou et al. *A C-LSTM Neural Network for Text Classification*. Computation and Language. 2016. URL: https://arxiv.org/abs/1511.08630.