Group 28 Coursework 1 - Question Classification

Abdullah Imam - *s09062ai*
Edward Davies - *p03167ed*
Rolando Fernandez - *y64738rf*
Sam Garlick - *j74268sg*

1 Introduction

The project aims to build a question classifier that can classify questions into a certain set of predefined classes. This is accomplished through training a neural network model. Question classifier is particularly useful in interfacing between human and machine. For example, it can be used in chat-bots and voice assistants as a primary step in comprehending user queries.

This paper will seek to decide what is the best setup for a model to classify these questions along three key metrics, Word Embedding, Sentence Embedding and Fine Tuning. The Word Embedding will be created from an imported pre-trained set of vectors or randomly created vectors. The Sentence Embedding that uses either the Bidirectional Long Short-Term Dependencies (BiLSTM) and Bag of Words (BOW) models, explained in further in the methodology. The tine-tuning is whether or not we freeze the Word Embedding or allow them to also be trained along with the rest of the words in the model. Effectively this final stage is testing how good the imported word embedding is. This task is difficult requires knowledge of multiple words in the sentence to derive an understanding of what the question is asking. It also requires real-world knowledge about a wide range of concept that a rule-based system would likely struggle to cover.

2 Methodology

2.1 Configuration

The configuration is fed from a config.ini file using the config-parser library. We also decided to wrap its handling of configuration into our own utility class that parses the configuration dictionary and asserts the presence of critical configuration parameters, as well as the presence of certain parameters conditional on the presence of others e.g when BiLSTM is specified as the sentence encoder, we expect the desired hidden-dimension size to also be supplied. Particular attention was paid to parsing and type safety of the configuration along with due attention to simple and clear error messages.

Our config-parsing utilities support ensemble models by means of nested .ini files referenced from the main config.ini, which are recursively loaded, parsed and validated to provide configuration for sub-models that make up the overall ensemble.

Our config utilities provide functions for building models by piping configuration through to our Model.Builder class (which follows the "builder-pattern") for validating and constructing the desired model.

2.2 Dataset

The data set is the 'Experimental data for Question Classification' [1] comprised of a set of questions and their corresponding question-topic classification. This dataset allows the use of machine learning classifiers to create a Natural Language Processing (NLP) pipeline which allows a user to ask a question and the classifier predict what type of answer the user is expecting. The objective of the classifier is not that of answering the question but allowing the NLP pipeline to choose a method that yields the best response and use a predetermined method to answer the given question, for example, knowing whether to gather data about a person on Wikipedia or search for a specific location using some global mapping software.

The training set was split up into two sets, training and validation, with a 90:10 split ratio respectively. It is important to ensure that all hyperparameters are tuned against a validation set, as tuning it against the test set will cause data leakage, introduce a bias, and potentially over-fit the model to work only against the test set and causing the model to perform poorly for unseen data. By tuning the network against the validation set, we can, with relative accuracy, predict the performance of the network by testing it against the unseen test set, hence our use of the validation set.

2.3 Tokenisation

The questions in the dataset are easy to parse as each sentence is broken down into individual tokens. However, this data is potentially exhaustive, containing excessive tokens which may be unnecessary for sentence classification. To reduce the token set, and therefore reduce complexity, we parse the tokens in each question to minimise and therefore simplify all the input data.

Numerous questions in the data-set contain URLs which contain a very specific order of characters, unlikely to be found in a word embedding. For the purpose of question classification, the specific URL in question is not important but it is important to know that a specific token represents a URL. For this reason, we parse and replace all URL with the token 'URL' and numerous other changes to reduce the token set. This is beneficial as the tokens are able to maintain semantic meaning even if the original specific token isn't found in the embedding set.

2.4 Vectorisation

Once we have the initial stream of words tokenised into meaningful tokens then we need to convert these word into a set of vectors that can hold information about their semantic meaning. These will start in two forms, either vectors imported from a pre-trained embedding or completely random vectors. Regardless each token will be represented by a unique vector that the classification algorithm can learn holds within it the notion of how the given word relates to all other words in the corpus. The pre-trained embedding is generated using an algorithm external to our model, the GloVe [4] as the one we are using but word2vec [2] another with nearly as good performance.

Vectors within these embedding have been exposed to lots of text where the words are used to in context and the semantic meaning has been stored in the vectors through a machine learning model. These vectors can be imagined plotted in multi-dimensional space, with more dimensions than a human brain can visualise but with distance on each of these dimensions representing an aspect of the semantic meaning. For example all noun tokens are similar to each other on a given dimension than they are to verbs with words that can be used as both noun or verb falling somewhere in between. With vectors of words that can be substituted into the same sentence such as 'Cat' and 'Dog' or 'Orange' and 'Pear' being close on some other dimensions. 'Orange' also being close to 'Blue' on a dimension that 'Pear' is not. Important to note that although we know this information is encoded into the vectors based on the results achieved when using them, the exact way is not explainable.

2.5 Hyper-parameter Tuning

A hyper-parameter is a control-able feature that may define some aspect of the model or how the model trains. Often one hyper-parameter value may be more suitable than another, for example, a higher learning rate is more likely to allow a neural network to learn faster albeit often at the expense of accuracy. The coursework specification outlines a limit of ten epochs, therefore, it is beneficial that we choose hyper-parameters that reach the highest accuracy after the limit number of epochs.

The network has numerous hyper-parameters which could be optimised, such as, batch sizes, layers sizes and loss functions. Whilst there are many different hyper-parameters about the network that we can tune, we only optimised a few due to time constraints. We tuned the network's learning rate and optimiser as we deemed these two variables to have the greatest effect on the network's classification performance. Some optimisers may perform better with a higher learning rate compared to other optimisers, meaning, there is not one learning rate that outperforms other learning rates across all optimisers. Therefore, in order to find an optimal learning rate-optimiser combination, we must repeat a learning rate search for each optimiser we test.

As previously mentioned we split up our data into three sets, training, validation and test to help ensure representative, non-biased results. To ensure that the results we obtained were reliable we repeated each experiment multiple times unseeded. The use of repeated runs allows us to gain a more representative view of how our learning rates and optimisers perform as some random initialisations may skew the results.

3 Results and Discussion

3.1 Evaluation Metrics

The coursework assigned is a task of classifying the type of question given. This type of prediction can either be right or wrong. A standard approach to this is to simply calculate the number of correct predictions. This approach is applicable, however, it does not depict the full ability of the network.

Rather than using accuracy, we use ROC analysis as a means to understand how the network is behaving when it doesn't predict the correct class. Particularly for working with multi-class data, ROC analysis allows us to understand and benchmark the classifier with greater accuracy and understanding. There are two main types of F1 average, micro and macro averaging where the micro-F1 score is far less sensitive to class imbalance. The dataset provided is very imbalanced with a clear majority of class 'HUM:ind', therefore, it was important to use the micro-F1 score to gain a representative view of the classifiers.

Word Embedding	Sentence Embedding	Fine Tuning	Result (%)
Random	BOW	No	34.6
Random	BiLSTM	No	52
Pre-trained	BOW	No	60.2
Pre-trained	BiLSTM	No	71.8
Pre-trained	BOW	Yes	55.0
Pre-trained	BiLSTM	Yes	73.1

Table 1: Model Classification Accuracies

Due to this class imbalance, there are a relatively small number of instances of classes such as, ”ENTY:religion”, ”ENTY:currency” and ”NUM:ord” - the three classes with the fewest number of training examples. This means the network may have a harder time discerning between the different labels. This is lack of training examples may be exemplified as we separate out the training set into the training and validation sets, further diminishing the number of these rare training examples. The method to which we separate our training set may cause an increased class imbalance, however, to ensure representative results, the examples used in the validation set were sampled stochastically to help ensure no bias created in either the training or validation sets.

3.2 Word Embedding

Through our testing we have shown that the worse performing is the random word embedding. For both models this shows worse performance than the pre-trained GloVe embedding vectors. This is expected as the random vectors are not encoded with any of the semantic meaning of the words that they represent. We also showed that with further fine tuning of the GloVe embedding vectors the improvement are very minimal. This shows us that the GloVe embedding vectors are already a very good representation of the words semantic meaning that our training did not have much meaning .

3.3 Sentence Embedding

We found that using BiLSTM to generate sentence embeddings achieved 10% greater accuracy overall regardless of the choice of word embedding, and indeed our best overall result came from a model incorporating BiLSTM sentence embeddings.

Bag-of-Words in comparison exhibited relatively poorer performance but still >50% accuracy. In addition, we found that a model incorporating Bag-of-Words trains a lot faster, due to the Bag-of-Words sentence embedding layer requiring no parameter tuning, unlike the BiLSTM’s neural network weights.

3.4 Hyper-parameter Tuning

Results comparing different hyper parameters. As previously mentioned, numerous hyper-parameters were tested, we found the following results to be optimal when compared using different optimisers[3] and learning rates.

- BOW: Adam with learning rate 0.002
- BiLSTM: Adma with learning rate 0.001

(When using Negative Loss Likelihood Loss)

3.5 Ablation Study

Fine-tuning word embeddings have a disproportionately greater positive effect in the case of randomly-initialised word embeddings vs. pre-trained word embedding. This is expected as the pre-trained word embeddings in GloVe provide useful properties such as an interpretable distance metric (cosine similarity) which groups similar words together in the embedding vector space. This means the GloVe embeddings contain more semantic meaning in the latent space which means the network doesn’t need to relate words together and the embeddings essentially have that property.

4 Conclusion

In conclusion, we found that the BiLSTM model outperforms the Bag of Words model by roughly 10% with an accuracy of roughly 70% - far greater than random guessing. Furthermore, we found the pre-trained glove embeddings outperformed random embeddings - which is to be expected. The Bag of Words model both trained and predicted far faster than the BiLSTM model meaning it may be more suitable for edge and low power devices whereas the BiLSTM model may be better for high-end server operations. We used Bag of Words as a means to make an ensemble model which was able to further outperform previous models with an accuracy of 70% outperforming all previous Bag of Word models in our work.

Future work for this coursework could include further hyper-parameter tuning and improved tokenisation. Additionally, we believe that we would be able to improve performance by creating a custom loss function that reduces the disparity between similar classes. For example, the network should be punished more for predicting the class ”LOC:state” when the answer is ”DESC:def” compared to the network predicting NUM:speed” to ”NUM:money” as these classes are both numeric. This should help improve the network's accuracy as it should learn that certain labels are semantically related although perhaps not 100% correct.

References

[1] Xin Li and Dan Roth. “Experimental data for question classification”. In: *Cognitive Computation Group, Department of Computer Science, University of Illinois at Urbana-Champaign,(URL: http://l2r. cs. uiuc. edu/% 7Ecogcomp/Data/QA/QC/)* (2002).

[2] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: `1301.3781 [cs.CL]`.

[3] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[4] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.