CS4248 Assignment 2

Leow Wen Bin

February 2021

1 Dependency Installation

To install dependencies, run both of the following commands, in sequence:

1. `pip install -r requirements.txt`
2. `python -m spacy download en_core_web_lg`

2 Model

Our choice of model was a traditional feed-forward neural network. The main reason for this was that there are many existing libraries in place that implement such neural networks, allowing us to save valuable time on creating the model. More resources are thus available to focus on the main focus of the assignment (feature engineering). Neural networks are also very general in nature - the architecture of the model can be made as complex as necessary to fit the problem, or it can likewise be simplified to the level of regression.

For this assignment, we have tried a variety of different architectures and algorithms. The most noteworthy differences are discussed below.

2.1 Output Layer

The neurons in the output layer implicitly encode the nature of the problem that we wish to solve. We have tried two different configurations, corresponding to two different ways to model the problem:

1. An output layer with 1 *tanh* neuron, modeling the problem as a regression problem on the range $(-1, 1)$
2. An output layer with 3 softmax neurons, modeling the problem as a 3-class classification problem

Option (1) above was our first attempt. It appeared to make the most sense as the labels were all in the range $[-1, 1]$, thus seeming suitable for a *tanh* function. However, it was observed that, unlike a typical regression problem, the labels here are discrete and fall into three classes. In addition, from a linguistic and semantic standpoint, it is not immediately apparent why "unimportant statement" (label 0) should be in the middle of the continuum between "important fact" (label 1) and "non-factual statement" (label -1). This was what led to the consideration of option (2).

The transition from option (1) to option (2) resulted in a massive 0.27 increase in F1-score. This justifies our earlier observation that the sentences belonging to the three types of labels do not, in fact, exist on a one-dimensional continuum as assumed by option (1).

Following this transition, another 0.12 increase in F1-score came about when the cross-entropy loss calculation was changed slightly, to give each class the same overall weight in the loss function. This change was brought about due to the obvious imbalance between the three classes, where there are about six times as many non-factual statements as there are unimportant statements.

2.2 Hidden Layers

As mentioned above, neural networks can be made as general or specific as necessary to model the problem. We have tried two hidden layer architectures, one of which is much more complex than the other:

1. Two hidden layers of size 500 and 75
2. One hidden layer of size 10

Of the above, option (1) came about due to the observation that the input vectors (section 4.1) were extremely large, with thousands of dimensions. This was how our earlier models were structured - later models changed to option (2), which was observed to converge several times faster than the alternative, while still offering comparable validation performance at convergence.

3 Pre-processing

For this assignment, we have tried various preprocessing steps on the string inputs before performing feature extraction.

3.1 Case-folding

The input strings contain text in proper sentence-case English. This does not lend well to learning as each word will then be represented by multiple types, thus greatly increasing the number of types that exist in the training set. Case-folding was our solution to this - by assuming that the case of the text does not encode any useful information, we converted all the text to lowercase and were thus able to reduce the number of types.

Case-folding was initially used with our original feature representation in section 4.1. This allowed us to reduce the number of types by about 500, giving us the same reduction in length of the feature vector and a slight improvement in F1 of about 0.06. In our final model, using feature representation 4.2, this step was not explicitly performed in our code - this is because our call to the spaCy library already performs case-folding by default.

3.2 Stopword Removal

The English language contains a large number of stopwords which carry very little useful meaning. The NLTK library contains a list of these stopwords - these words were removed from the training set as they seemed to serve little to no semantic purpose. Our own list of stopwords was added on to this, consisting of interjections like "uh" and speech-related contractions like "n't".

Stopword removal was used in our initial attempts at a model. However, in our final model, it was found to result in an F1-score that was about 0.05 lower - implying that some of the stopwords in the set are actually useful for predicting the label. Using NLTK's list alone did not make any improvement either. As such, this preprocessing step did not make it to the final model.

4 Feature Engineering

4.1 Sum of One-Hot Vectors

Our first attempt at feature engineering was to split the sentence into its constituent tokens - each token would be one-hot encoded, then the vectors would be summed up to obtain the representation for the original sentence. This constitutes a bag-of-words paradigm which operates under the assumption that the choice and frequency of words in a sentence is enough to predict its label.

The nature of this encoding scheme means that each dimension directly corresponds to one type in the vocabulary. Under such an encoding scheme, it would make sense that unimportant types in the vocabulary, such as stopwords, would also result in entire dimensions that are not useful for classification. It was this intuition that led to the usage of stopword removal (section 3.2) in our earlier models. The same observation also meant that the model would not have much of an opportunity to train on words that are rare in the training corpus, as the dimensions representing these words would almost always be 0, thus effectively preventing backpropagation. Because of this, we also chose to remove words that occurred rarely in the corpus (less than 5 times total).

Our best attempt using this encoding scheme resulted in a validation F1-score of approximately 0.55, which was much lower than our final model. This would indicate that many of the words are likely not useful in predicting the label. Additionally, the large dimensionality of the input vector meant that there were more weights to be trained as well, thus hindering the learning process.

4.2 Sentence Embedding

One major issue with the previous choice of feature representation in section 4.1 is the large dimensionality and the sparsity of each input vector. As a solution to this, we tried using the vector embeddings provided by the spaCy library. The embedding of each sentence is formed by the average of the embeddings of its constituent words - these give rise to vectors of length 300, which are far smaller and denser than those of 4.1.

Unsurprisingly, this feature representation led to an improvement over the previous representation, with the F1-score increasing by about 0.05. This was the choice of feature representation in our final model.

4.3 TF-IDF

An attempt to further improve our model's performance led to the usage of TF-IDF vectors. In order to reduce the dimensionality of the vectors without compromising too much on performance, these vectors would be generated after

stopwords were removed (3.2). Unsurprisingly, this representation did not perform as well as 4.2, causing F1-score to fall by about 0.03. This is likely due to many of the same problems as the original representation in 4.1.

5 Analysis and Comparison of Attempts

In the process of training a performant classifier, we have gone through many iterations of models. Each iteration varies from the previous in some way - in this section, we will list a few noteworthy iterations and compare them to their predecessors/successors. These comparisons will primarily focus on the areas of feature representations and preprocessing.

5.1 Second Iteration

- Model: Two hidden layers of size 500 and 75; output layer of size 3
- Preprocessing: Case-folding, removal of rare words
- Feature Representation: Sum of one-hot vectors of constituent words
- Validation Macro-F1: 0.55

It was observed that the training and validation cross-entropy losses were still falling even after 25 epochs. This likely indicates a slow rate of learning, caused by a large number of parameters. The solution was to reduce the number of parameters - this led us to look into an alternative feature representation (section 4.2) and smaller hidden layers.

5.2 Third Iteration

- Model: One hidden layer of size 10; output layer of size 3
- Preprocessing: Case-folding (implicit; part of external library call)
- Feature Representation: Vector embedding of length 300
- Validation Macro-F1: 0.60

The amount of time taken to train this model was greatly reduced - convergence was reached in about 5 epochs. This is owed to the much smaller feature representation, which allowed us to reduce the number of parameters between the input and first hidden layers. There was a small improvement in F1 score as well. The next two iterations primarily involved modifications to the optimizer and model hyperparameters - as such, we will not go into them in detail.

5.3 Fifth Iteration

- Model: One hidden layer of size 10; output layer of size 3
- Preprocessing: Case-folding (implicit; part of external library call)
- Feature Representation: Vector embedding of length 300
- Validation Macro-F1: 0.74

The fifth iteration improved from the third mostly through changes in the optimizer and hyperparameters. As such, we will not go into detail - this subsection exists solely for comparison to the following two.

5.4 Sixth Iteration

- Model: One hidden layer of size 10; output layer of size 3
- Preprocessing: Stopword-removal; Case-folding (implicit; part of external library call)
- Feature Representation: Vector embedding of length 300
- Validation Macro-F1: 0.71

The only difference between this and the fifth iteration was that in this iteration, stopwords were removed from the text before being converted to a vector embedding. Given the semantic nature of stopwords, it was expected that their presence would "dilute" the meaning of the sentences - the removal of stopwords should thus result in better performance. Interestingly, this was not the case - removing the stopwords resulted in a slight fall in F1, indicating that some of the stopwords are actually useful in predicting the labels. The decision was thus made to leave stopwords as part of the sentences to be classified.

5.5 Seventh Iteration

- Model: One hidden layer of size 10; output layer of size 3
- Preprocessing: Stopword-removal; case-folding
- Feature Representation: TF-IDF vector
- Validation Macro-F1: 0.72

Our seventh attempt involved creating a model which would use a TF-IDF vector representation of sentences. This was mainly to investigate if it could improve over the fifth attempt, which used dense vector embeddings. Perhaps unsurprisingly, the macro-F1 fell slightly. This is likely due to the sparsity of TF-IDF vectors, which makes learning more inefficient. The decision was thus made to stick with dense vector embeddings.

6 Declaration of Original Work

By entering my Student ID below, I certify that I completed my assignment independently of all others (except where sanctioned during in-class sessions), obeying the class policy outlined in the introductory lecture. In particular, I am allowed to discuss the problems and solutions in this assignment, but have waited at least 30 minutes by doing other activities unrelated to class before attempting to complete or modify my answers as per the Pokémon Go rule.

Signed, A0184415E

7 References

I give credit where credit is due. I acknowledge that I used the following websites or contacts to complete this assignment.

1. Natural Language Toolkit Documentation (https://www.nltk.org/)
2. PyTorch Documentation (https://pytorch.org/docs/stable/index.html)
3. Scikit-Learn Documentation (https://scikit-learn.org/stable/modules/classes.html)
4. spaCy Documentation (https://spacy.io/api/doc/)