DIKU NLP PROJECT

Ellia Morse, Federico Fiorio,
Gonzalo Villalba

Abstract

The aim of this project was to create a multilingual question answering system. The model was trained on English, Finnish, and Japanese samples within the publicly available TyDi QA dataset. The dataset contained a set of question-document-answer items.The project built up through the duration of the course following in-class lessons.

Introduction and Work Distribution

This project was completed by Ellia Morse, Federico Fiorio, and Gonzalo Villalba.
The work has been divided between the 3 students in this way:
Federico was responsible for:

- Week 1 (entirely);
- Week 2 (entirely);
- Week 3 (task (a) together with Gonzalo, task b,c,d alone);
- Week 5 (entirely);
- The corresponding report parts regarding the work done on the project.

Gonzalo was responsible for:

- Week 3 (task (a) together with Federico; task (e) alone);
- Week 6 (entirely).
- The corresponding report parts.

Ellia was responsible for:

- Week 4;
- Revisions made on midpoint reviews
- The rest of the report.

Week 1: Introduction to NLP

1.1 Preprocessing and Data Analysis

This section had two goals: to create a preprocessing pipeline to tokenize samples at the word level, and to understand what tokens usually begin and end questions.

A single sample consisted of question text, a document tile, langage, annotations, document plaintext, and the document url. However, not all of the data was relevant for our project.

There were two datasets: training and validation. Samples from both sets were tokenized at the word level by answers, context, and questions. Samples also belonged to several different languages but we only needed those in Japanese, Finnish and English.

Tokenization was performed with the nltk library for English. Janome was selected for Japanese. Google Translate was used to translate words for the purposes of this report.

The next step was to investigate the most and least common words observed in questions and context. A "bag of words" model counted the number of occurrences for each word. The training dataset was included for counting occurrences.

Within the most common "question" tokens for each language, all languages contained the question mark token and the equivalent tokens for "What", "was", and "is". The top ten words differed by the following:

English: ['the', 'When', 'of', 'How', 'in', 'did', 'Who']

Finnish: ['Milloin', 'Kuka', 'oli', 'syntyi', 'kuoli', 'Kuinka', 'tarkoittaa']
['When', 'Where', 'Who', 'What', 'was born', 'died', 'How', 'means']

Japanese: ['の', 'い', 'つ', 'し', 'どこ', 'が', 'か']
['of', 'I', 'one', 'then', 'where', 'is', 'or']

The least common word in the English dataset was peninsula. For Finnish it was *väestölaskennan*, translated to "census" while in Japanese it was ウサギ translated to "hare".

The most common tokens in the 'Context' samples are often punctuation such as periods, apostrophes, and commas. The least common words in each context dataset were:

English: ['the', 'of', 'and', 'in', 'to', 'a']

Finnish: ['ja', 'on', 'oli']
['and', 'is', 'was',]

Japanese: ['の', 'に', 'は', 'を', 'た', 'が', 'で']
['in', 'to', 'is', 'was', 'was', 'is', 'in']

Least common words in context:
English: igniting
Finnish: ylänköalue
Translated to “Highland area”
Japanese: 終わらせよ
Translated to “Finish it off”

1.2 Binary Question Classification
The goal for this task is to train and evaluate a binary classifier on the linguistic features extracted from the data.

Logistic regression is a simple model that guarantees good interpretability for binary classification. Binary classification is based on the probability of a binary event occurring.

While we previously obtained a “bag of words” that could be used to train our classifier, we decided to use a TF-IDF representation to train our classifier. TF-IDF stands for term frequency and inverse document frequency.
The idea behind TF-IDF is that each word will have a weight. The higher the weight is, the better that word describes the specific document in a corpus.

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

This feature is obtained by calculating the term frequency for a word and multiplying it by the inverse document frequency for that word—as seen in the formulas above.

This approach treats the questions and context as two separate features. After TF-IDF is computed on each, we unite the features and obtain a simple vector for each sample.

TF-IDF is computed only on the training set. If the IDF vector is re-utilized for the validation set, unknown words are ignored. Otherwise, data-leakage could have occurred and the model would have obtained some optimistic performance.

Another necessary condition is that the independent variables contributing to the binary answer need to be independent from each other. If not, adjusting one weight will change another weight related to a different class class. This will produce poor classification scores.

Since TF-IDF produces our features by highlighting specific words that represent the document, these independent variables should have a small multicollinearity.

	English	Finnish	Japanese
train_set	89.2%	91.69%	98.33%
vald_set	72.42%	71.23%	57.92%

The scores on the Japanese data highlight that our model has some overfitting behaviour. We may want to try another model, or representation specifically for the Japanese language.

Week 2: Representation Learning

This week we were asked to implement an extension to the classifier that utilises word2vec architecture.

Word2vec is a group of related models that produce word embeddings— real valued vectors that include the meaning of the word. Word2vec is a neural network architecture composed of these three layers: ‘input’, ‘hidden’, and ‘output’.

While there are two types of word2vec architectures, CBOW architecture and skip-gram architecture, we chose CBOW. In the “continuous bag of words” model, a phrase generated by a sliding window on the data is inputted. The model will consider the centre word of the phrase as missing. Then it will predict that word while using the other words in the sliding window as context.

The skip-gram model is more limited since only one word is used as context when making predictions. We used the word2vec model to represent a sentence instead of just words. A vector could be computed based on each word in the question or context. Then, the average of all vectors would be computed, generating a single vector to represent the whole question or context of a sample.

Our model has a vector size of 500, so the hidden layer has a size of 500 neurons. 5 words are considered in the sliding window.

The word2vec model was trained on our training dataset. It created a representation on the validation dataset. Any words present in the validation set but not in the training set were ignored by the model.

Then we re-trained our previous logistic-regression model. This time, a single input sample would contain concatenated vectors produced by TF-IDF representation for question and context as well as the two CBOW representations for question and context. These features should improve the model's understanding of context.

The accuracy with these features are:

	English	Finnish	Japanese
train_set	86.41%	89.03%	88.50%
vald_set	72.02%	72.22%	64.19%

While these results are similar to those from week 1, the Japanese language was more accurate. This confirms that the model reached a better generalisation.

We also tried to re-train the logistic regression model with just the features coming from the word2vec representation, we obtained the following results:

	English	Finnish	Japanese
train_set	69.69%	69.60%	68.22%
vald_set	65.35%	64.70%	63.32%

The model trained only with the continuous vector representation learnt the meaning of the words, but not how they are entangled. Word2vec does not consider the position of the words since the vector of the sentence is only the average for the vector of the words. That model captures the context and the words. This does not translate into an understanding of what composes a good question or whether a question is answerable or not.

The TF-IDF representations are more detailed and perform accordingly better

Week 3: Language Modelling

The scope for this week was to train a language model or fine-tune one. Language models are a way to associate a probability to a sequence of words. We decided to use a model architecture called BERT, particularly, bert-base-cased, and fine-tune it.

Transformers receive the whole input of a sentence at once. However, they understand the position of each word by using a positional embedding.

BERT stands for Bidirectional Encoder Representations from Transformers. Transformers are different from LSTMs (or Long short-term memory) because they prioritise the most important features of a sentence using a mechanism called attention. BERT also ignores non-base forms of words like playing or plays.

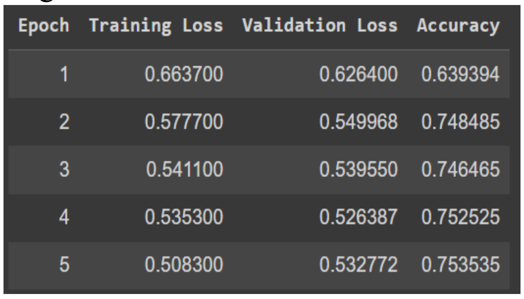
BERT is a model composed of stacked encoders and it is trained using masked-LM. The [CLS] and [SEP] tokens define, respectively, the start of a sentence and the separation between two sentences.After a first tokenization of the input where [CLS] and [SEP] tokens are added to the sentences, the model masks some inputs. During the training phase, BERT will predict those masked words with the help of the surrounding context.

BERT can be fine-tuned for many machine-learning tasks. In our case, we want a model for sequence classification. In order to fine-tune this model, we have decided to give as input to the model, once again, the question + context as a single input sample. Since there were different lengths for the input sentences we used padding for the shorter sentences and truncation for the longer ones, in order to have sentences of the same length.

Two layers were added to the original model taken from HuggingFace. The first layer consists of a pooler layer and the second layer is used for classification. All the layers of the model except the last two used for classification were frozen. Freezing a layer means that during training the gradient for that layer will not be computed. Thus the weights will not be updated and those layers will not change.

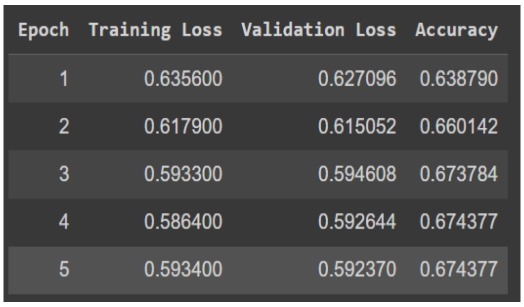
These scores were achieved with 5 epochs:

English:



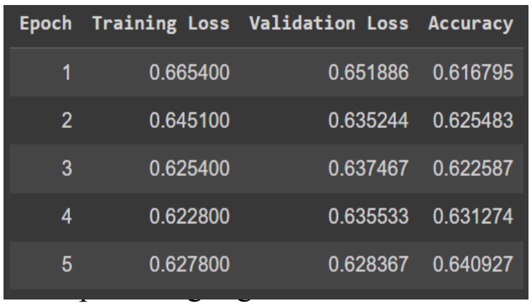
Epoch	Training Loss	Validation Loss	Accuracy
1	0.663700	0.626400	0.639394
2	0.577700	0.549968	0.748485
3	0.541100	0.539550	0.746465
4	0.535300	0.526387	0.752525
5	0.508300	0.532772	0.753535

Finnish:



Epoch	Training Loss	Validation Loss	Accuracy
1	0.635600	0.627096	0.638790
2	0.617900	0.615052	0.660142
3	0.593300	0.594608	0.673784
4	0.586400	0.592644	0.674377
5	0.593400	0.592370	0.674377

Japanese:



Epoch	Training Loss	Validation Loss	Accuracy
1	0.665400	0.651886	0.616795
2	0.645100	0.635244	0.625483
3	0.625400	0.637467	0.622587
4	0.622800	0.635533	0.631274
5	0.627800	0.628367	0.640927

compared with the previous logistic regression model. Some fine-tuning of the hyperparameters will bring these models to better results.

We then tried to sample from the model to understand how well it was understanding the language.

BERT's sampling was bad, the model kept predicting ellipses to complete a sentence, for example:

"Today I will buy" was continued with "Today I will buy ……….."

For this reason we have tried to use another model called GPT-2, Generative Pre-trained Transformer 2.This generative model, which still uses attention and transformers mechanisms, generated far better sentences with respect to BERT.

For example, the previous sentence "Today I will buy", was completed with "Today I would like to buy a new car. I am a big fan of the BMW i3 and I am looking forward to the new car."

We also tried to comprehend the models' understanding of the language using a metric called perplexity. Given a model and an input text sequence, perplexity measures how likely the model is to generate the input text sequence.

A small perplexity indicates that the model understands the language. High perplexity means that the model is, in fact, perplexed and it does not understand a sentence.

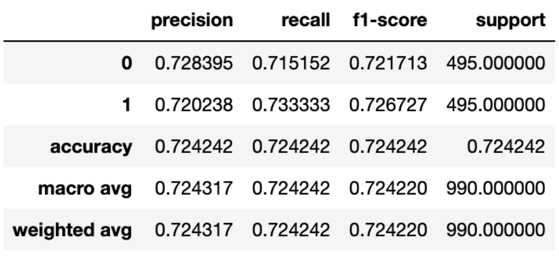
We tried to use as input the entire validation set to the BERT and GPT-2 models. GPT-2 obtained a small perplexity while BERT had a very large perplexity.

Lastly, we defined a mean pooling function to represent the last hidden layers of the language model. We implemented the extracted layers into the datasets with padding.

Week 4: Error Analysis and Interpretability

This week's purpose was to conduct error analysis and to compare the performance of BERT and the logistic regression TF-IDF model on the task of binary question classification. Error analysis is the practice of understanding a model's behaviour. We can examine the classification report and confusion matrix as the first steps, as well as return what the most and least weighted tokens are for each model.

The confusion matrix resulting from the Logistic Regression model is the following array. ([354, 141], [132, 363]). The first two values are the actual values and the others are the model's predictions. Here is the classification report for LR.



	precision	recall	f1-score	support
0	0.728395	0.715152	0.721713	495.000000
1	0.720238	0.733333	0.726727	495.000000
accuracy	0.724242	0.724242	0.724242	0.724242
macro avg	0.724317	0.724242	0.724220	990.000000
weighted avg	0.724317	0.724242	0.724220	990.000000

Both models are very sensitive to wording. The TF-IDF model places more weight on the most frequently occurring words, which happen to be question tokens. The BERT model struggles when questions are inputted with the token "is" before the masked token. For example, upon entering the sentence "The most popular winter holiday is {tokenizer.mask_token}.", the model will return "is" as the masked token with over 25% certainty, and the answer "Christmas" with under 12 percent certainty.

It appears that when the model does not understand a question, it will often return a token already existing within the sentence as an answer, regardless of whether it matches the correct part of speech. An example of a question that returns the wrong class is "Who is going to be at the party".

Week 5: Sequence Labelling

The task for week 5 was to train and validate a sequence labeler. The labels are based on IOB tagging, which stands for "inside, outside, and beginning of an entity". We customised the IOB tagging system to recognise parts of the answers inside the context. The beginning of the answer would be tagged "B-answ", and any remaining part of the answer would be "I-answ". Any elements not belonging to the answer were labelled with O.

For this task we tried with different approaches, first with transformers, then with a Bilstm with CRF and then we tried with an encoder-decoder model with beam search.

Before talking about the models, once again, the input consisted in a concatenation of question and context, only the inputs with answerable questions were taken into consideration, because in the other cases, the labeler could not tag anything and would have just ignored the sample.

In order to perform a supervised learning approach, we had to give to each word in our question + context a label, in order to achieve this result we have tagged every word in the samples using the dataset annotations field which also contains the answer that our labeler is expected to find.

Going deeper into the explanation of the models used: The transformer model used is called distilbert,particularly distilbert-base-uncased.Distilbert is a transformers model, smaller and faster than BERT, the model was fine-tuned on our dataset of generated tagging labels.Since distilbert adds tags like [CLS], [PAD], [SEP], to our input data, we had to re-align our labels for the IOB tagging so that they would only match the corresponding words, and not some added tags.The tags that were not initially part of the input have been labelled with a -100 so that during training, distilbert would have ignored those -100 labels and would have been trained only on the proper words.

The labels used for tagging were heavily unbalanced, the tag ‘O’ was dominant with respect to the others, so we had to define a proper metric which was not accuracy, and we had to give proper loss weights to the loss function.

The metric used was the F1-score because it summarises precision and recall and it is used in case of unbalanced datasets, if we would have used accuracy, that could have been almost 100 but our model in that case would have been useless, because a model that always predicts ‘O’ will have a very high accuracy but 0 meaning.

For the weights of the loss function we have decided to give them with the following formula. This formula assures a big weight for the unbalanced class, and thus it is saying to the model to give more weights/attention to that specific class. The model achieved an F1-score of 0.36, which is very poor. We can explain why this score is poor. Consider the golden labels. We are utilising a method called This BERT model does not use conditional random fields (CRF).

CRF is a discriminative model that considers dependency between the inputs and the previous predicted labels, if instead of considering the predicted labels we would need to first flatten out the output from BERT using a linear layer of size 768 as the BERT output, and then add the CRF at the end of the architecture.

We then decided to try the Bi-lstm + CRF architecture instead of the transformers + CRF architecture. The LSTM part of the architecture is bi-directional in order to better capture the context of the inputs, in bidirectional LSTM we give the input from both the directions from right to left and from left to right. The CRF part captures the best labelling sequence as output, boosting the performances of the labeler. The F1-score produced by this model was far better then the plain transformers, with an F1-score of 58, this proves the effectiveness of CRF.The last model used is an encoder-decoder which uses beam-search in order to find the best sequence of tags.

Beam search is an algorithm used in nlp tasks, specifically in sentence generation, where instead of using a greedy approach and output the highest probability word (in our case tag) at each step, beam search keep a specified number of beams that represents the top choices between the words to predict (the ones with highest probability) and for each one of them, it will iterate the process.

The algorithm will also prune the least appealing sentences, that is, it will prune the sentences with the least sum of log-likelihood (keep the top b sentences, where b is a parameter). The beam-search enc-dec results were disappointing with an F1-score of 0.29 with a beam size of 3 and 2 and with and an F1-score of 0.35 with a greedy search.

It looks like the sequence of tags that are not all ‘O’ will correspond to a lower overall F1-score, so even if beam-search finds more realistic tagging sequences they produce a lower score. With a greedy search the algorithm reaches a higher result, but all the tagging sentences are similar to having all ‘O’ as prediction.

Looking at the confusion matrix of our Bi-lstm with CRF:

```
      ‘O’      ‘B’    ‘I’
‘O’   [61284   151    453]
 ‘B’  [321       137    20]
‘I’   [1517      14    466]
```

These predictions obtained an F1-score of 0.55.As we can see there are a lot of FP and FN, the FN are much higher than the FP for the tags: ‘B-answer’ and ‘I-answ’ , with the class predicted being the ‘O’ instead of the actual class.

This highlights that most of the times, the model predicts ‘O’ instead of the proper tag even with the Bi-lstm model, so the adjustments for the loss weights were not enough, and the model still prefers to predict the common class ‘O’ most of the times. This behaviour suggests that some more preprocessing could be done to the dataset in order to make it more balanced.

Week 6: Multilingual QA

This week, we were tasked with implementing the Answerable TyDiQA from week 3 and the IOB tagging system from week 5. Here, we still worked with the English, Finnish, and Japanese languages.

For the purposes of this week’s assignment, we chose the multilingual “XLM RoBERTa” model. This model was suggested by the teaching team of this NLP course as a widely accepted multilingual model, ideal for question answering.

The goal was to train the model with a dataset of only one language at a time, and recording the difference in output when trying to answer questions in other languages. For instance, we first trained the model with the English dataset, and predicted answers with English, Finnish, and Japanese questions. Then, we did the same after training the model with the Finnish and Japanese datasets, respectively.

Before that, we defined the functions for the evaluation tools to compare outputs for abovementioned processes and outputs. At the end, we saw that the tasks where best performed in Finnish when testing the zero-shot cross-lingual evaluation. On the other hand, in Japanese, we observed that the sequence labeling did not perform as well as Finnish. Finally, in the English language, both tasks achieved good performances.