DIKU NLP Course 2022/2023: Group Project

Rikke Rhode Nissen and **Killian Steunou**
Group 105

Abstract

The aim of this project is to create a multilingual question answering system.

1 Introduction to NLP

1.1 Preprocessing and Dataset Analysis

We have implemented a preprocessing pipeline that tokenises the instances (question and document) for each of the languages. The pipeline is based on pretrained `SpaCy` tokenisers for each of the languages: `en_core_web_sm`, `ja_core_web_sm`, and `fi_core_web_sm`. For at start, we remove all punctuation tokens. The three most common words for each of the languages follows in table 1.

English	Finnish	Japanese
when	milloin (when)	日本 (Japan)
what	mikä (what)	アメリカ (America)
how	missä (where)	世界(World)

Table 1: Three most common first tokens (excluding punctuation) for the three languages.

And the three most common last tokens for the languages follows in table 2.

English	Finnish	Japanese
born	syntyi (born in)	た (rice field)
founded	on (is)	か (mosquito)
die	kuoli (died)	何 (what)

Table 2: Three most common last tokens (excluding punctuation) for the three languages.

The tables above illustrates the differences in how sentences are constructed in different languages: English and Finnish questions are constructed similar, where the "question-word" like *what* is placed in the beginning of the question. Japanese questions are constructed almost in the opposite direction, where questions begin with a question mark and "question-words" are placed in the end of the sentence. For the further analysis, we also remove stopwords from the tokens.

1.2 Binary Question Classification

1.2.a Design classifier

We design a binary classifier to predict whether a question is answerable (1) or non-answerable (0).

The classifier takes the following features based on the question and the context of the document:

Occurrences: We count how many times the a single words appears in both the question and the document and report the total number of "occurrences". This is a relative simple approach to measure similarity between the question and text. The intuition is that the more times a word appears in both question and text, the greater is the probability that the answer can be found in the text. The number of occurrences for each question is scaled with the maximum number of occurrences.

Bag-of-words: We define bag-of-words vectors with use of `CountVectorizer` to fit and transform question and document into two equal sized vectors, where positive numbers in the same position indicates a common word between the question and document. We use two components of the the BoW-vectors in the model.

First we compute the mean of the BoW-vector for the question, as an indication of how many times a word in the question appears in the document relative to the documents length. A larger mean value corresponds to words in the question are often represented in the text.

Second we compute the cosine similarity between the two BoW-vectors and use this in the model. The cosine similarity measures the cosine of the angle between the BoW-vectors, which in our case indicates if the same words are often in the question and document. One strength of using cosine similarity, relative to other distance measures, is that it compares the direction of the vectors and not the length, which accounts for the text being much longer than the question. Cosine similarity close to 1 indicates that question and text consists of the almost same words (tokens).

Term frequency-inverse document frequency, TF-IDF: Another feature similar to the Bag-of-words, is the TF-IDF, which quantifies the relevance of the words. The feature calculates how often a word appears in the text scaled with the relative rarity of the word. This makes the feature more sophisticated than the BoW. We use `tfidfvectorizer` to compute TF-IDF vectors for both the question and the text. Following similar arguments as for the BoW-implementation, we use the mean values for the TF-IDF question vector and the cosine similarity between the question and text TF-IDF vectors.

We define the binary classification model as a neural network with three layers, the two hidden layers are wrapped in a ReLU and the output layer is wrapped with a sigmoid. The two hidden layers is linear layers with input dimension five and eight (and output dimension eight and 24). The last layer, the output layer, is also a linear layer with input dimension 24 and output dimension 1.

1.2.b Train classifier for each language

When training the model we use the binary loss function `BCELoss` as we have a equal distribution of answerable/non-answerable question, and the `Adam` optimizer. We keep track of the accuracy during training and break the training process if it does not improve over 10 training instances.[1]

1.2.c Evaluate classifiers

We obtain the following accuracy and loss:

- English classifier accuracy: 68.9 %
- Finnish classifier accuracy: 71.9 %
- Japanese classifier accuracy: 66.5 %

The results shows that the Finnish classifier performs better than the two classifiers for the two other languages.

2 Representation Learning

In this section we extend the binary classifier as we defined above with features based on continuous vector representations. To extend the model, we implement Byte-Pair encoding which tokenise words and transform the tokens into vectors. For each of the languages we load pre-trained subword embeddings with dimension 100 and vocabulary size 25.000 from the library `bpemb`. For this assignment Byte-Pair encoding have especially two strengths. First, we will not get any OOV (out-of-vocabulary) tokens, as unknown words are splitted until the remaining part exists in the vocabulary. Also different representations of the same word like *bird* and *birds* will be interpreted as similar. Second, the vector representation of tokens ensures words that are alike (e.g. *apple* and *pear*) are also close in the vector space.

2.a Model with previous features and features based on continuous representation

We use the pre-trained subword embedding BPEmb to create the vectors (dimension 100) for the question and the context (combined into one text string) in the dataset. We include the additional 100 BPE-features in the binary classifier. We use the same model and evaluation method as specified in the previous assignment. This gives the following results:

- English classifier accuracy: 73.3 %
- Finnish classifier accuracy: 77.5 %
- Japanese classifier accuracy: 75.1 %

Including Byte-Pair embeddings of the question and text clearly improves the accuracy of all three classifiers. Losses for the best classifiers in about the same as for the previous classifiers, however during training, the reported losses were in general lower. Especially, the Japanese classifier improves a lot.

2.b Model with only the continuous representation

Now we include only the 100 BPE-features to train the model, which results in the following:

- English classifier accuracy: 71.4 %
- Finnish classifier accuracy: 71.4 %
- Japanese classifier accuracy: 72.2 %

All of the three classifiers have a lower predictive power compared to the previous classifiers in 2.a. The accuracy of the Finnish classifier is about the same as the classifier in 1.2, which is somewhat

[1]The model is trained with the hyper parameters: epochs=300, learning rate=0.001, batch size=64.

surprising. We would have expected features based on BPE to be more predictive as it not only captures the same word but also words that are similar to each other.

3 Language modelling

In this section we implement an extension to the classifiers in Assignment 1.2 and Assignment 2, where we extract word/sentence representations from neural language models.

3.a Pre-trained language models

For all of the three languages we will use pre-trained monolingual Transformer language models and fine-tune on the TyDi QA training sets for each of the languages. For the three models we will use the following pre-trained language models `distilgpt2` (English), `Finnish-NLP/gpt2-finnish` (Finnish), and `rinna/japanese-gpt2-small` (Japanese). Each of the models are fine-tuned using the training document text for the corresponding language.[2]

3.b Sampling from the models

We sample from the three fine-tuned language models to get a understanding of the model. For an experiment we pass "What is" as a start of the sentence and let the model generate the rest. In the first approach we simply use *greedy search* which returns the following:

What is the first time that a person has been arrested for a crime in the United States. The first time that a person has been arrested for a crime in the United States. The first time that a person has been arrested for a crime in the...

The model quickly starts repeating itself. Furthermore, greedy search returns the word with the highest probability and might hereby miss high probability words hidden behind a word with lower probability. To avoid this we can instead use *beam search* which generates words based on the probability for the next `num_beams` words, in this case we look at the next five beams, which generates the following:

What is the first time in the history of the United States of America that the United States of America has become the first country in the world to become the first country in the world to become the first country in the world to become the first country in...

[2]The models are fine-tuned with the hyper parameters: learning rate = 2e-9, weight decay = 0.01, number of training ephocs = 1, maximum steps = 300, with evaluation at the end of each epoch.

The model still repeats itself. By including N-gram penalty we can stop the repeating by reducing the probability of words that have already been generated. This results in the following sentence:

What is the first time that a person has been arrested for a crime in the United States. The first is a case of a man being arrested in a state of emergency, and the second is an arrest of an individual who is being held in custody...

The sentence make a slightly more sense. To generate more sensible sentences we further include Top-K Sampling with K=50, where the model picks the next word from distribution of the K most likely next words. Including all of the above mentioned options in the text generation we obtain sentences which *almost* make sense:

What is a video game developed by Microsoft and developed for the Xbox 360 and PlayStation 3. The game was first released in the United States in 1995 and was the first game to be released on the PlayStation 4 and Xbox One. It was released as a...

3.c

Optional

3.d Evaluate the language models

We evaluate the language models by computing the perplexity score both on the raw pre-trained Transformer model and after the models are fine-tuned using TiDy QA. The results are stated in Table 3.

Language	Perplexity (raw)	Perplexity (fine-tuned)
English	50.51	44.41
Finnish	19,789,662,094.37	90.08
Japanese	62.38	44.39

Table 3: Perplexity for pre-trained Transformer language models before and after fine-tuning.

As we would expect, all language models performs on the specific task as we fine-tune on the training data. The Finnish language model performs extremely bad before fine-tuning, which shows the need for fine-tuning on relevant data.

3.e Pooled representation of the last hidden layer

In this section we will use a mean pooling representation of the last hidden layer in the language models as input in the classifier in 2, replacing the continuous representation by this input.

To create the feature, we used the language model to get the last hidden layer, and then we apply a mean pooling function. This gives us a tensor of size (1, 768) that we can use as a feature in the model. After training, here are the results:

- English classifier accuracy: 70.3 %
- Finnish classifier accuracy: 50.0 %
- Japanese classifier accuracy: 66.2 %

For the English and Japanese classifier, the models are more accurate than the ones in 1, but worse than those using the byte-pair embedding as a feature (2). The Finnish classifier is performing poorly, with an accuracy stuck at 50%.

4 Error Analysis and Interpretability

In this section we will perform an error analysis to try and see how a model performs and what kind of errors it makes.
For this, we have selected two models, with different features, achieving different accuracy, and performing the same task. For the contrast to be the higher, we chose the English version of the model trained in section 1, which we will call `model_1_2b`, which has an accuracy of 69.9 %, and the English version of the model trained in 2.a, which we will call `model_2a` which has an accuracy of 73.3 %. They are the farthest models we have in term of accuracy.

4.a Confusion Matrix

First, let us have a look at the confusions matrices obtained on the test set:

	N-A	A	Total
N-A	278	217	495
A	98	397	495

Table 4: Confusion Matrix of the `model_1_2b` on the Test set. N-A = Non-answerable and A = Answerable

The confusion matrix of the first model (figure 4) shows that the model performs quite well on the answerable questions (80.2% of correct predictions), but struggles on the non-answerable questions with only 56.16% of the questions correctly predicted.

	Non-answerable	Answerable	Total
Non-answerable	338	157	495
Answerable	103	392	495

Table 5: Confusion Matrix of the `model_2a` on the Test set

On the other hand, `model_2a` has a similar accuracy for the answerable question (79.19%), but a way better accuracy for the non-answerable questions of 68.28%.
It seems that both models tend to give too much importance to the co-occurrence of words in the question and the context, since it tends to be wrong and predict 'Answerable' when one word of the question is found in the context.

4.b Fooling the model

Here's an example. Given this context: *"The iPhone is a line of smartphones designed and marketed by Apple Inc. These devices use Apple's iOS mobile operating system. The first-generation iPhone was announced by then-Apple CEO Steve Jobs on January 9, 2007. Since then, Apple has annually released new iPhone models and iOS updates. As of November 1, 2018, more than 2.2 billion iPhones had been sold. As of 2022, the iPhone accounts for 15.6% of global smartphone market share."*, we first predict for this question *"Who is the world leading phone seller?"*. This question has no answer from the context, and the models predict correctly, "Non-answerable". But if we slightly change the question, to *"Is Apple the world leading phone seller?"*, which has almost the same meaning and is still not answerable, the models predict that it is, just because of the word "Apple" in the question.
A difference between the models appears when asking *"How to find jobs?"*, `model_1_2b` gets fooled by the word "jobs" that is the same as "Jobs" from "Steve Jobs", without the capital letter, whereas `model_2a` gets the correct prediction, thanks to the continuous representation of the question used as a feature.

5 Sequence Labelling

In this section we will implement a sequence labeller, which predicts which parts of the paragraph (document text) are the likely part of the answer to the corresponding question.

5.a Convert to IOB format

First, we combine the English, Japanese, and Finnish datasets into one dataset and combine each question and context into one text[3]. We define a function which first tokenise the text and then indicate the position of the start and end token of the answer. We can use the combination of the indicators for start and end tokens and obtain a vector with the same dimensionality as the tokenised text where 1 and 0 indicates if the token is part of the answer or not. We use the same language model to tokenise the text as we will use for the next subsequent questions.

5.b Implement sequence labeller

We implement a sequence labeller based on the pre-trained multilingual Transformer language model `xlm-roberta-base`. Since the task is question answering, we use the `AutoModelForQuestionAnswering` class to load the model. The scope of the model is to predict the span of the answer given the question and context, which is equvivalent to predicting the 0/1-indicator vector of the answers position. Then we map these positions back on the original text and outputs the answer span to the question. We fine-tune the pre-trained model on the training sample to obtain better performance for the labelling task.[4]

As we fine-tune the model on the TyDi QA dataset, we loop over the number of epochs and for each epoch, the batches in the data. After fine-tuning the model outputs logits for start and end position of the answer span. If we do a greedy search for the best answer span, we first select the start position with the highest logits and then loop over the end positions to find the one with the highest score.[5]

5.c Add beam search extension to sequence labeller

We extent the model by adding beam search. The model still outputs the same logits, however instead of selecting the start position as the one with the highest logits, we select the k highest scoring start positions and for each we again loop over the end positions, sums the logits and then selects the answer span based on the highest score. This should increase the performance of the model (or at least not make it worse). To illustrate this argument we consider the following text *...get a coffee and a cookie....* If *cookie* have the highest probability for start position, then by greedy search the predicted answer would be cookie. However, it might be that cookie have very low end position probability. By using beam search we might then find that the *coffee* gives a higher sum of logits and are a more likely answer.

5.d Analyse performance

We analyse the differences in performance of the greedy search and the beam search, we train the model on a sample of 4,000. The greedy search gives an F1- score at 67.9 % and performs slightly worse than the beam search (beam size=20), which performs an F1-score at 68.5 %. For the further analysis, we evaluate the model performance using beam search.

To evaluate the performance of the implemented sequence labeller, we compute the percentage of exact matches and the F1 scores. Before fine-tuning the model we get 0.16 % exact matches and a F1-score at 2.6%. After fine-tuning the model we obtain a 71.9 % exact matches and a F1-score at 75.9 %. This result clearly demonstrates the necessity of fine-tuning pre-trained language models before using them on a specific task.

To get a better understanding on the performance across the languages, we have validated the fine-tuned model on each of the three languages. The results are stated in Table 6. We find that the fine-tuned model in general performs best on Finnish and English data and worst on Japanese data. In Assignment 1 we found that the sentence construction differs from Japanese and the two other languages, as we fine-tune on all three languages, English and Finnish data are overrepresented relative to Japanese. This can be an explanation to the differences in performance across languages.

5.e Qualitative investigation of predicted answer spans

We find that when the predicted answer span is wrong, it is typically for one of the three following reasons. First, the model predicts the question is

[3]In case the combined question and context is longer than 384 tokens, we split it into several features, where each feature contains the question and the remainer part of the text (overlapping 128 tokens). For this specific task, the specified length should be sufficient.

[4]The model is fine-tuned with the weighted Adam optimiser and the hyperparameters: learning rate=2e-5, number of epochs=3, weight decay=0.01 and warmup steps=200.

[5]When looping, we do not consider spans of answers that are either out of scope, too long or have a negative length (end position before start position).

Language	Exact matches	F1-score
English	70.9 %	77.3 %
Finnish	74.9 %	79.9 %
Japanese	68.0 %	68.1 %

Table 6: Exact matches and F1-scores for pre-trained XLM-RoBERTa-base question answering model fine-tuned on Answerble TiDy QA dataset for English, Finnish, and Japanese.

unanswerable i.e. no answer. Second, the predicted answer span is either longer or shorter than the true answer span, but contains (at least part of) the true answer. For instance the true answer is *the 20th century* and the model predicts *beginning of the 20th century*. Third, the model predicts wrong dates as an answer. It seems that the model sometimes get confused about "when was..."-question and predicts the wrong date from the text.

6 Multilingual QA

In this section we implement multilingual QA models based on multilingual text representation.

6.a Implement binary question system and IOB tagging system with a multilingual encoder

The IOB tagging system from Assignment 5 have already been implemented with a multilingual encoder.

6.b Cross-lingual evaluation

IOB tagging

By: Rikke Rhode Nissen

We perform a cross-lingual evaluation by fine-tuning on one language and validating on the other two languages. To give a "fair" evaluation, we fine-tune on the same sample size across the languages.[6] The validation results are reported in Table 7 and 8. If we consider the reported exact matches we do not observe large differences across the languages, other that models trained on Finnish in general have more exact matches than the other languages.

If we instead consider the F1-scores, it is notable that models trained on Finnish and validated on English (and vice versa) performs better than models that are trained on Japanese and validated on either Finnish or English (or vice versa). As argued in Assignment 1, the sentence construction in Japanese differs from the two other languages. This might be an explanation for the differences in the cross-lingual performance.

[6]We fine-tune on a random sample of size 1000, as the task is only to compare evaluation across languages rather than predicting the answer span.

Train \ Test	English	Finnish	Japanese
English	-	52.0 %	52.1 %
Finnish	52.8 %	-	54.3 %
Japanese	50.7 %	49.1 %	-

Table 7: Cross-lingual evaluation: Exact matches for pre-trained XLM-RoBERTa-base question answering model fine-tuned on Answerble TiDy QA dataset for one language and testes on the other two.

Train \ Test	English	Finnish	Japanese
English	-	56.8 %	52.3 %
Finnish	59.0 %	-	54.6 %
Japanese	52.3 %	51.7 %	-

Table 8: Cross-lingual evaluation: F1-scores for pre-trained XLM-RoBERTa-base question answering model fine-tuned on Answerble TiDy QA dataset for one language and testes on the other two.

6.c Compare performance

Let's compare the models for the tasks of binary classification (answerable and not-answerable) and the task of span-based QA.

6.c.1 Binary Classification

We use the model trained on the features that has proven to perform better, that is to say the ones defined in section 2.a, and compare the accuracy.
Below are ranked the three languages according to their accuracy.

1. Finnish: 0.7782
2. Japanese: 0.7413
3. English: 0.7394

6.c.2 Sequence labeller

We use the span-based sequence labeller defined in section 5.c. We use the f1 score to compare each language.
Below are ranked the three languages according to their accuracy.

1. Finnish: 79.935

2. Japanese: 77.324

3. English: 68.095

6.c.3 Conclusion

We can see that both settings achieve the best performance in Finnish, whereas in Japanese, even though the classifier has a performance close to Finnish, the sequence labeller does not perform very well, with a f1 score almost 10 points under the one in Finnish.
Regarding the English language, both settings achieve good performances.