**DIKU NLP PROJECT**

**Ellia Morse, Federico Fiorio,**

**Gonzalo Villalba**

**Abstract**

The aim of this project is to create a multilingual question answering system. The dataset used is the publicly available TyDi QA dataset containing a set of question-document-answer items. This project covered English, Finnish, and Japanese. 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.

Ellia was responsible for:

* Week 4;
* The rest of the report.

1. **Introduction to NLP (5-11 Sept)**

**1.1 Preprocessing and Data Analysis**

The goal of this section was to create a preprocessing pipeline to tokenize instances of English, Finnish, and Japanese at the word level, and familiarise ourselves with the data by understanding what tokens usually begin and end questions.

A first step in the preprocessing pipeline was to familiarise ourselves with the data, understand which are the parts that make up a single sample and which parts to use for the other tasks of the project.

A single sample is composed by these attributes:

* Question\_text;
* Document\_tile;
* Language;
* Annotations;
* Document\_plaintext;
* Document\_url.

The samples belong to different languages but we only need Japanese, Finnish and English for the scope of the project, so the first step was to retrieve only those specific samples related to these languages.

Then we tokenized at word level the answers, context and questions for each sample, because these attributes will be the only important ones for the rest of the project.

Tokenization was performed with nltk library for english and finnish and with janome for japanese.

To further familiarise with the data, we investigated the most common words in questions and context, we also observed some of the most uncommon words in questions.

For languages that we do not speak we used google translate as a machine translation tool in order to translate the words and write them in the report.

In order to find common and uncommon words we use a bag of words model where basically, we count the number of repetitions for each word, the word that has the highest count is the most common word.

The 10 most common words for questions are (considering training set):

English: ['?', 'the', 'When', 'was', 'What', 'is', 'of', 'How', 'in', 'did']

Finnish: ['?', 'on', 'Milloin', 'Mikä', 'Missä', 'Kuka', 'oli', 'Mitä', 'syntyi', 'kuoli']

Translated to:

['?', 'is', 'When', 'What', 'Where', 'Who', 'was', 'What', 'was born', 'died']

Japanese: ['は', 'の', '？', 'た', 'い', 'つ', '何', 'し', 'どこ', 'が']

Translated to:

['is', 'of', '?', 'was', ‘I’, 'one', 'what', 'then', 'where', 'is']

The least common words in questions are:

English: peninsula,

Finnish: väestölaskennan translated to census,

Japanese: ウサギ translated to hare.

The top 10 common words in context are (considering training set):

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

Finnish: ['.', ',', 'ja', 'on', '[', ']', '(', ')', 'oli', ':']

Translated to:

['.', ',', 'ja', 'on', '[', ']', '(', ')', 'was', ':']

Japanese: ['の', '、', 'に', '。', 'は', ' ', 'を', 'た', 'が', 'で']

Translated to:

['of', ',', 'to', '.'. ', 'is', ', ' 'was', 'is', 'in'].

As we can see, while in questions the words are representative for questions, in the context the most common words are represented by stop words.

Least common words:

English: igniting

Finnish: ylänköalue

Translated to Highland area

Japanese: 終わらせよ

Translated to Finish it off

Take into consideration that we have two datasets, training and validation.

Each step described regarding tokenization was performed on both datasets.

**1.2 Binary Question Classification**

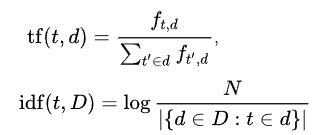
The goal for this task is to train and evaluate a binary classifier on the linguistic features extracted from the data.

From 1.1 we already have a bag of words that could be used as a linguistic feature, however we decided to use a TF-IDF representation to train our classifier.

TF-IDF stands for term frequency and inverse document frequency.

This feature is obtained by calculating the term frequency for a word and multiplying it by the inverse document frequency for that word.

In the following image we can see a simple way to compute both.



The idea behind TF-IDF is that each word will have a weight, the higher the weight the better that word describes the specific document in a corpus.

The approach used is to treat the corpus of questions and the corpus of context as two separated corpuses and then after the computation of TF-IDF on both, we have united the features and obtained a simple vector for each sample, this vector represented the words that better describe the question and context of that sample.

TF-IDF is computed only on the training set and then, the IDF vector has been re-utilized for the validation set, unknown words are ignored.

Computing TF-IDF with training and testing combined would have brought some data leakage, which means that the model would have obtained some optimistic performance.

The binary classification used is a logistic regression, we have decided to use it because it is one of the simplest models which guarantees a good interpretability.

Logistic regression is essentially used to calculate the probability of a binary event occurring.

This classification algorithm has a binary output coming from a set of independent variables.

A problem that can occur is that the independent variables into to have a small multicollinearity or in other words they need to be independent between each other, if that is not the case, it will become difficult to adjust the weights of the model, because changing a single weight will result in changing also another weight related to another class, because of the multicollinearity and this will result in poor classification scores.

In this case our features come from TF-IDF, while it is true that words are somehow related between each other, using TF-IDF should highlight only specific words that represent the document, so in some sense these independent variables should have a small multicollinearity.

The scores obtained with logistic regression are:

English: training 89.2%, validation 72.42%

Finnish: training 91.69%, validation 71.23%

Japanese:training 98.33%, validation 57.92%

The scores on the Japanese data highlight an overfitting behaviour and thus, it could be a good idea to try to use another model, or to try to use different representation for the features, specifically for the Japanese language.

1. **Representation Learning (12-18 Sept)**

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

Word2vec is a group of related models that are used to produce word embeddings.

Word embeddings are real valued vectors that include the meaning of the word.

Word2vec is a neural network architecture composed by 3 layers:

* 1 input layer;
* 1 hidden layer;
* 1 output layer.

There are two types of word2vec architectures, the CBOW architecture and the skip-gram architecture.

In the case of CBOW or “continuous bag of words”, the model has as input a phrase generated by a sliding window on the data. The model will consider the centre word of the phrase as missing, and will try to predict that word based on the context, that is, on the other words present in the sliding window.

In the skip-gram model, only one word is taken into consideration and based on this word, the model tries to predict the other missing words in the sliding window.

We used a word2vec model based on the CBOW architecture.

The model has a vector size of 500, so the hidden layer has a size of 500 neurons.

The sliding window is 5, so only 5 words are considered at the time.

For the continuous vector representation we used the word2vec model trained on the train set and then we used the same model to perform the representation also for the validation set, the words that were in the validation set but not in the training set were ignored by the model.

We used the word2vec model to represent a sentence instead of just words, in this way we could use the model to represent with a single vector, the whole question or the whole context of a specific sample.

We basically computed the word embedding for each word presented in the question/context and then we computed the average vector for them (summing all vectors and then dividing by the number of words).

We re-trained our previous logistic-regression model, this time, as input features we had the previously computed TF-IDF features for question + context (the vectors were concatenated) and then we also added the CBOW representation for the question concatenated to the CBOW representation for the context.

So to recap a single input sample for this model has the question + context TF-IDF representation concatenated with the CBOW question + context representation.

These features should give the model a wider understanding of the context.

The results with these features are:

English:

train\_set accuracy

86.41

vald\_set accuracy

72.02

Finnish:

train\_set accuracy

89.03

vald\_set accuracy

70.22

Japanese:

train\_set accuracy

88.50

vald\_set accuracy

64.19

The results are similar to the previously computed ones for week 1, however the Japanese language obtained a higher accuracy bringing the model to 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:

train\_set accuracy

69.69

vald\_set accuracy

65.35

Finnish:

train\_set accuracy

69.60

vald\_set accuracy

64.70

Japanese:

train\_set accuracy

68.22

vald\_set accuracy

63.32

We can say that 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, and our representation for the vector of the sentence is just the average for the vector of the words.

So this model captures the context and the words but doesn’t really understand what is a good question and what is not, it does not really capture if a question is answerable or not.

With also the TF-IDF representation we were able to give a more detailed representation of the data and thus the performances were a little bit better.

1. **Language Modelling (18-25 Sept)**

Language models are a way to associate a probability to a sequence of words, they associate a probability to a word given n others preceding words.

The scope for this week was to train a language model or fine-tune one.

We decided to focus on a model architecture called BERT, particularly, bert-base-cased, and we tried to fine-tune it.

BERT stands for Bidirectional Encoder Representations from Transformers.

Transformers are different from LSTMs because they are able to focus on the most important parts of a sentence using a mechanism called attention.

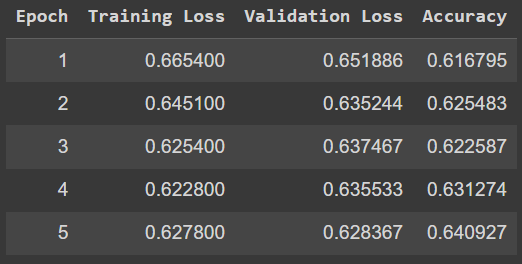
Transformers do not process sentences sequentially but they receive the whole input at the same time, they are still able to understand the position of each word by using a positional embedding.

BERT is a model composed of stacked encoders and it is trained using masked-LM.

Basically, after a first tokenization of the input where [CLS] and [SEP] tokens are added to the sentences, the model will mask some inputs and during the training phase will try to predict those masked words with the help of the context (other words) around the masked word.

The [CLS] and [SEP] tokens are used to define respectively the start of a sentence and the separation between two sentences.

BERT also ignores non-base forms of words like playing or plays.

BERT can be fine-tuned for many different tasks, in our case we used a BERT model used for sequence classification, it means that from the original model, two layers were added, the first one consisting of a pooler layer and the second one used for the classification task.

The model was taken from [HuggingFace](https://huggingface.co/).

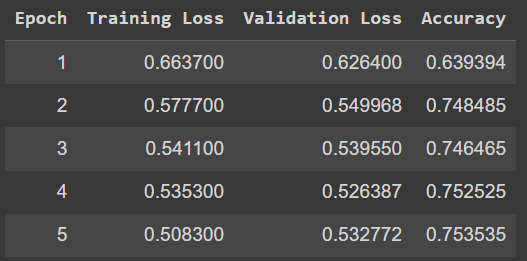
In order to fine-tune this model, we have decided to give as input to the model, once again, the question + context as a single input sample.

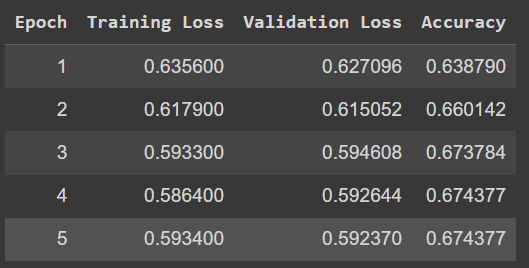
Since there were different lengths for the input sentences we used padding for the shorter sentences and truncation for the longer ones, in order to have sentences of the same length.

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 and thus, the weights will not be updated, meaning that even after training those layers will stay the same.

Scores achieved with 5 epochs:

English:

Finnish:

Japanese:

The accuracy measure computed highlights a better model compared with the previous model (logistic regression).

We only trained the models for 5 epochs, 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 dots as words to finish off a sentence, for example:

We have let the model continue the phrase:

“Today I will buy” and the model 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 dig deeper into the models and tried to understand what their understanding of the language was by using a metric called perplexity.

Without going too much into details, 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, and high perplexity means that the model is, in fact, perplexed and it does not really understand what is going on.

We tried to use as input the entire validation set to the BERT and GPT-2 models.

GPT-2 obtained a small perplexity, meaning that the model is understanding the language and it does not have problems generating similar sentences.

BERT had a very big perplexity and it had no idea on what was going on.

ADD TASK (E) ???

1. **Error Analysis and Interpretability (26 Sept - 2 Oct)**

This week’s purpose was to conduct error analysis and to compare the performance of BERT (or GPT2) and the logistic regression model.

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 examine what the most and least weighted tokens are for each model.

Confusion matrices:

Logistic regression vs BERT

[[273,222],

[120, 375]]



| Figure 1: Chart depicting the confusion report for the logistic regression model |
| --- |

What are common mistakes of the weak model?

Are there any common mistakes of the stronger model that the weak model predicts correctly?

What are the similarities of the two models?

We can examine the better model using the local interpretation tool SHAP (SHaply Additive exPlanations).

What input tokens do the model consider to be the most important? What are the common patterns of each of the classes?

Write adversarial instances that fool the model into predicting the wrong class. Any differences in model predictions on newly created data?

1. **Sequence Labelling (3-9 Oct)**

The task for week 5 was to train and validate a sequence labeler.

The labels are based on IOB tagging.

IOB stands for: inside, outside, beginning of an entity, in our case we had to customise the IOB tagging to our specific need.

The labeler had to recognise parts of the answers inside the context, specifically it had to label the beginning of the answer with a tag B-answ, then tag the rest of the answer (if there is) with I-answ and the rest of the elements not belonging to the answer had to be 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 samples.

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:

*1- (Count for the tag class / total number of elements in all classes).*

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 F-1 score of 0.36

The result is pretty bad, and there is a reason for it.

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

Consider the golden labels, then we are doing what is called “teacher forcing”.

In order to add CRF to BERT 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 following the lab5 instructions.

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 F-1 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 predictions.

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-answ’ 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.

1. **Multilingual QA (10 Oct +)**

