Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Active Learning for NLP

Anton Kretov

Supervisor: Ing. Jan Drchal, Ph.D.
Field of study: Open Informatics
January 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kretov** Jméno: **Anton** Osobní číslo: **474672**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Datové vědy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Aktivní učení pro metody zpracování přirozeného jazyka

Název diplomové práce anglicky:

Active Learning for NLP

Pokyny pro vypracování:

The aim of this project is to assess various methods of active learning in the NLP domain. Give an overview of state-of-the-art methods and evaluate them on various available NLP datasets. Focus on applicability to Transformer neural models.
1) Explore the state-of-the-art methods of active learning applicable to NLP tasks.
2) Select appropriate public datasets. Focus on Natural Language Inference including the public Czech CTKFactsNLI dataset.
3) Design a methodology to compare selected active learning methods.
4) Implement (if needed) the selected methods, perform experiments and evaluate.
5) Publish the code as an open-source package.

Seznam doporučené literatury:

[1] Dor, Liat Ein, et al. "Active learning for BERT: An empirical study." Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020.
[2] Schröder, Christopher, and Andreas Niekler. "A survey of active learning for text classification using deep neural networks." arXiv preprint arXiv:2008.07267 (2020).
[3] Wang, Zijie J., et al. "Putting humans in the natural language processing loop: A survey." arXiv preprint arXiv:2103.04044 (2021).
[4] Ren, Pengzhen, et al. "A survey of deep active learning." arXiv preprint arXiv:2009.00236 (2020)

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Drchal, Ph.D. centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **29.08.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové práce: **19.02.2024**

Ing. Jan Drchal, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank CTU and my supervisor Ing. Jan Drchal, Ph. D. for giving me an opportunity to work on this project and to contribute to the whole Natural Language Processing (NLP) community with my work. I would also like to thank my colleagues who listened to my stories about this project and who gave me valuable consultations and provided lots of advice. I would also like to express my gratitude to my parents and to my friends who supported me during my studies and during this research project.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 10 January 2023

Abstract

Natural language processing (NLP) is a field that attracts lots of artificial intelligence researches who attempt to solve various tasks of processing text and extracting knowledge from it. NLP research is centered around several renowned technologies and methodologies which have proven to be promising and that are already helping in various aspects of life. However, the problem of data collection and data labelling is currently playing one of the most important roles in many machine learning problems, especially in natural language processing, where it is even more pronounced. In this thesis I examine active learning - a methodology for smarter training of machine learning models with less data by favoring those data entries that presumably contribute more to model's convergence. I also introduce an implementation of an open-source library for further experiments in this field. I conduct many experiments showing the impact of active learning compared to conventional models training and revealing the best performing active learning strategies. Finally, a state-of-the-art result on Czech NLI dataset CTKFactsNLI was achieved during experiments.

Keywords: NLP, machine learning, active learning, BADGE, uncertainty sampling, BERT, RobeCzech, NLI, CTKFactsNLI, RCI cluster, Weights & Biases, state-of-the-art

Supervisor: Ing. Jan Drchal, Ph.D.

Abstrakt

Zpracování přirozeného jazyka (anglicky *Natural Language Processing*) je oblastí výzkumu umělé inteligence, která si klade za cíl zpracování textu a extrakci znalostí z něj. Výzkum NLP se v současné době soustřeďuje kolem standardních přístupů, které jsou v praxi dobře osvědčeny. Nicméně problém sběru a anotace dat je stále velmi palčivý a hraje velmi důležitou roli ve mnoha úlohách strojového učení, obzvlášť ve zpracování přirozeného jazyka. V této práci studuji metody aktivního učení (anglicky *Active Learning*) - metodiku chytřejšího trénování modelů strojového učení, která si klade za cíl použítí menšího množství kvalitních dat, potřebných k natrénování modelů na požadovanou úroveň přesnosti. V této práci zároveň představuji svoji implementaci open-source knihovny pro další experimenty v této oblasti. Pomocí této knihovny jsem provedl několik experimentů, jež prokazují efektivitu metod aktivního učení a ukazují, které metody jsou nejslibnější. Nakonec se mi povedlo dosáhnout tzv. state-of-the-art výsledků v úloze ověření faktů (anglicky *fact checking*) na českém datasetu ČTKFactsNLI za pomocí jedné z metod aktivního učení.

Klíčová slova: NLP, machine learning, active learning, BADGE, uncertainty sampling, BERT, RobeCzech, NLI, CTKFactsNLI, RCI cluster, Weights & Biases, state-of-the-art

Contents

Chapter 1

Introduction

1.1 Preface

The project I have been working on is my contribution to the Natural Language Processing (NLP) field of artificial intelligence (AI) research conducted by many data scientists across the world. NLP is a vast field with different tasks, approaches, and methodologies that are being explored for many decades. Most of major breakthroughs in Deep Learning beyond NLP, including the most recent State-of-the-Art (SOTA) methods and architectures, were possible thanks to the last several years of major improvement and standardization in NLP sphere, where Transformer [VSP$^+$17] architecture with Attention Mechanism inside [BCB14] which I was studying deeply in my Bachelor's thesis [Kre20], played an essential role in making NLP more widespread, more democratized, less sophisticated from one point of view. Nevertheless, the invention of Transformer and the first foundational model [BHA$^+$21] called BERT [DCLT18a] made it possible to solve various types of tasks, starting from the well-known and well-established tasks of text classification and words tagging, to more the complex ones requiring deeper language understanding, such as fact checking [GSV22], machine translation [YWC20] and text summarization [BR20].

This project is also a contribution to the vast NLP research conducted by NLP research group lead by Ing. Jan Drchal Ph.D., who is also my supervisor, which is a part of Artificial Intelligence Center (AIC) at CTU.

Active Learning (AL) is a framework which originally comes from generic machine learning. However, it is currently undergoing its own renaissance and hence has been receiving considerably more attention as of late. In order to formulate my motivation in this field, I first provide a short outline of NLP as a research sphere and describe the way it works these days. By doing this, I will formulate the problem which is hypothesised to be solved by AL framework.

Active learning is a branch of artificial intelligence research, which aims for the optimalisation and reduction of the volume of data needed for training machine learning models to reach required accuracy. Since many machine learning models belong to the so-called statistical machine learning sphere, data form a vital part of a successful model. Data collection, versioning, annotation, validation are tedious and time-consuming processes. It is therefore extremely expensive and makes up a considerable part of a project budget. Some machine learning tasks allow for the so-called data augmentation [FGW$^+$21], when the new synthesized data entry is generated given a genuine data entry, e.g. a rotated or distorted photo given an original picture. Data augmentation is a well-established method of improving data magnitude and model robustness [RGC$^+$21], however in NLP it is rarely the method that we can safely use - often the result of augmentation applied to text is a new piece of text which does not make any sense. Hence, data has to be collected and treated with caution. Generally, the majority of data found in the world is represented in a textual form [HPZC07], however in order to train an efficient model, this data has to be of good quality and high variability. Active learning has emerged to address this issue and to provide a methodology and instruments for training models with less effort put into data collection and labelling. Active learning tries to find only those data entries which can provide more information and thus give more benefit to the model, by inherently bypassing duplicate entries and promoting those that are distinct from the ones already seen. The idea of active learning is to collect as little data as possible for reaching required performance. In other words, AL states the hypothesis that only a fraction of data of high quality is enough to train the model for the same accuracy as with the whole dataset.

1.2 NLP - brief outline

Natural Language Processing (NLP) is a field of artificial intelligence (AI) study aiming to solve tasks which deal with text. Textual information is considered as unstructured data. Often the task is formulated in a way that we need to extract some information from the text: either to classify it (is e-mail spam or not?), or to find names of people there (so-called Named

Entity Recognition task). Overall, there were different approaches to NLP throughout its history:

- At the very beginning, there were basic rule-based algorithms for basic language modelling and simple morphological and semantic NLP tasks. In this era, the first chatbot model ELIZA [Wei66] was built, which was based on pattern matching and substitution methodology - a simple algorithm found precise string entries in the text and matched them against a predefined list of known words and phrases.

- In the early 2000s, a boom of using statistical methods with n-grams (a sequence of n tokens: characters or words) for language models emerged. Several different neural approaches were also developed for language modelling [BBEZ00, MSC$^+$13] and downstream tasks. Statistical approaches required more data to be collected, which is in principle a complete change in paradigm compared to the earlier ways of solving NLP tasks, where the emphasis was put on rules and common knowledge converted into code. Statistical methods provided a toolset for generalization and patterns learning: more frequent n-grams received higher probability of occurrence in a random text.

- The focus has then changed to language models. Language models are literally models possessing Markov property - memory-less property of a stochastic process of language generation. The task of a language model is to assign a probability of the next token given the current one (there are also models which pay attention to more than one token). This task is accomplished mostly by training of a neural network (black-box model). Language models started to become larger and larger with far more parameters to learn, thus putting more strain on data needed to train better models. People transitioned from feature engineering to an automated feature extraction with the help of neural networks [HS97], mainly with the help of recurrent neural networks [MKB$^+$11] or convolutional neural networks [KJSR15], thus treating text with less granularity and working with data as time-series rather than independent tokens. Since language is a complex mechanism, the models started to be more complex and therefore required more data to be trained on.

- The era of large language models like ELMo [PNI$^+$18], ULMFiT [HR18] came with BERT [DCLT18b] finally conquering the whole NLP community and becoming the SOTA solution. These language models have huge number of parameters, need a lot of computational power and are trained for many days with the help of extremely expensive hardware [RNKC22].

- These language models have spurred the rise of *Transfer Learning*. The idea behind Transfer Learning is simple yet very powerful: a general and large model presumably captures the high-level understanding of how language works during the so-called pre-training. Then, this model is used for solving some particular task (e.g. Named Entity Recognition,Part of Speech tagging Summarisation). Hence, the model is trained on a specific dataset for the particular task. This dataset is considerably smaller than the amount of data required for training a language model. This step is called *fine-tuning*.

Nowadays, fine-tuning is considered the standard to solve NLP tasks. Fine-tuned models usually outperform those models trained from scratch. Moreover, training from scratch means that a network has to firstly understand the language structure before solving a language task. Thanks to this approach, the amount of data required to train a domain-specific end-to-end model is significantly reduced. Furthermore, **Active learning** (AL) may become the next logical step aiming for choosing only those data helping model to learn underlying problem structure, thus reducing the amount of required training data even more. It introduces the mechanism of assessing the quality of data used to train the model and favors data quality instead of quantity. There are also several more benefits which might be introduced by AL framework:

- The amount of work performed by people is reduced, thus saving time and money.
- The data selected by AL framework might result in a dataset of higher quality. AL can presumably be applied to an already existing dataset in order to investigate it and to find inconsistencies.

I give a detailed introduction to active learning methodology in the following section.

Chapter 2

Active Learning

Active learning (AL) is an approach to train machine learning models in iterations by gradually obtaining training data by utilising model's current knowledge, unlike passive learning, where all labelled data is provided beforehand. It introduces the mechanism of assessing the quality of data used to train the model and favors data quality instead of quantity. The main goal of this framework is to reduce the amount of data and effort needed by the model to reach required performance.

Currently, the models are trained in the following way:

- The dataset is prepared. It can be for example a database dump of incoming e-mails, or a collection of news articles.
- The dataset is annotated by a domain expert. For example, a news article is given a label whether it is a fake-news article or not; or an e-mail is given a classification whether it is spam or not.

  The annotation is usually conducted by several annotators by following a concrete methodology.
- An annotated dataset is divided into training, validation and testing sets. The training set is used for adjusting neural network's weights, the validation set is used together with training set to assure that the model is not overfitting or underfitting. The test set is used to predict the model's performance on previously unseen data.

Traditionally, a vast collection of unlabelled text is collected and then annotated. As a result, a huge dataset of labelled data is produced. Annotator's job is mostly monotonous and tedious, which is prone to errors, e.g. the same text can be mistakenly put into different categories. Then, annotation of a large dataset is an extremely lengthy process, which is consequently expensive. Another pitfall of such approach is abundance of similar data. For instance, when sentiment analysis dataset consisting of movie reviews is collected and annotated, these sentences express the same idea:

1. *This film is good.*
2. *This film is very good.*
3. *The film I have recently seen is good.*
4. *This is a very good film.*

From the perspective of a contextual language model [BGD+21], these texts have very similar representation, i.e. their vectors are very close to each other. Hence, labelling of all these sentences will not give adequate amount of diverse information to the model. In case when texts are completely different and describe the same category from different language perspective (by different expressions and collocations, lengths, etc.), the model is able to learn much broader space of phrases that cover the topic better. In real-life scenarios though the amount of literally duplicate texts in training datasets is enormous and thus more courteous annotation is needed.

Active learning is a framework which changes the methodology of collecting data for neural models. It favors data quality over quantity: a dataset is collected from the very beginning and the framework determines examples for annotation. The decision is made by a so-called **strategy**. An active learning strategy is responsible for gradual collection of future inputs to the model from an unlabelled set of data. These texts are then sent to an annotator who gives them the correct labels. After the labels are received, the texts are put into training set, which then becomes larger. By this approach we let the model decide itself, what data needs to be collected and labelled.

Active learning works in the following way. We assume that we have a large set of unlabelled data to choose from. The initial chunk of data is chosen and labelled by an expert, so as to give some basic understanding of the problem in question. For example, it might consist of several instances of different categories. Then, the model is trained on this chunk and evaluated on validation and test sets, which are already prepared and are fixed for the whole process. These datasets are necessary for training efficiency control. Then, the next chunk of unlabelled data is chosen, which is then passed to the annotator. After the chunk is annotated, it is appended to the training dataset, and the process starts once again from the beginning.

A simple AL framework consists of the following four main steps, which are also depicted in Figure 2.1. This sequence is repeated until the desired accuracy evaluated on the test set is reached.

1. Creation of a training dataset based on labelled data.
2. Training of a classifier or other model based on this data.
3. Selection of the new batch of unlabelled data (based on an AL strategy).
4. Labelling of the new data batch before adding it to the training set.

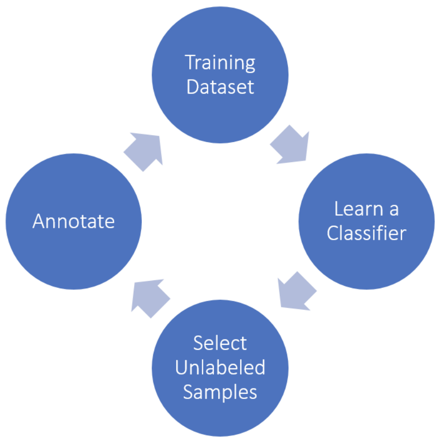


Figure 2.1: AL pipeline [Set09]

Seed data creation

Initial dataset is created by a traditional approach to label some amount of collected data. It is needed for the initialisation of the model. The dataset has to be small enough for new batches to influence model's behaviour. Correct sizes of the training dataset and the additional batch of newly annotated data is currently a research question. During AL experiments, when a suitable strategy is unknown, an initial dataset might be a random sample of a small size from the original training dataset. This sample is then considered "labelled" part, whereas the rest of the dataset is for experimental phase considered "unlabelled" - so that the active learning scenario is simulated.

Model training

The model is trained in the same way as if it is trained with a full training dataset.

New batch selection

This is the novel part in the whole training process and is the key part of active learning. The whole idea of active learning lies in a way (or a *strategy*) the new batch of previously unseen data is selected. It can be done in many different ways. Data can be selected from the vast collection of unlabelled data or it can be synthesised. More on that in 2.3.1.

New batch labelling

This phase is a logical continuation of the selection phase. Data chosen for labelling is passed to domain experts for annotation. This batch can also be supported by the model's guess - candidate categories, however during the first several iterations the model cannot guarantee stable behaviour. The selected batch is labelled and then the decision maker has to assess this batch from the domain perspective - is the batch well-balanced, is the batch descriptive enough or whether it has some dominant topic with minor ones not present (data imbalance)? These are the questions to be answered before adding the batch to the training set, since the model's performance is highly influenced by each added batch, especially during the first several iterations when there are not much data present.

2.1 Related Work

Active learning holds a long and successful history in the field of machine learning [FSST97], and many different AL strategies have already been proposed over the years (e.g., [AZK+19], [CDG+21a]). Very recently, [HGD19] argued that AL strategies choose similar examples, which do not significantly contribute to the learning process. They propose an approach that actively adapts to the deep learning model being trained to eliminate these redundant examples. With this method, they are able to reduce data requirements of AL strategies by around 2 - 16 % on multiple NLP tasks while achieving the same performance.

Moreover, [SYL+17] claim that while being sample-efficient, AL tends to be computationally expensive since it requires iterative retraining. They propose a more lightweight architecture that requires less computational power for a Named Entity Recognition (NER) task and achieve nearly state-of-the-art performance on standard datasets. During AL, they were able to match that performance with only 25% of the original full training data.

Furthermore, [SPK+21] decided to investigate the combination of transfer and active learning for a sequence tagging task. For the CoNLL-2003 corpus, the combination of their best performing pre-trained model and AL strategy achieved 99% of the score that can be obtained with training on the full corpus, while using only 20% of the full dataset. Besides, they also demonstrate that acquiring instances during AL, a full-size Transformer can be substituted with a distilled version to achieve better computational performance. Therefore, in my project I have also focused on working with smaller Transformer models to save computational power while still achieving good performance. Then, I have received access to higher computation power and thus conducted more experiments that are also documented in this report. I believe that research in the field of AL should be further accelerated and I we want to contribute to it by proposing a generic framework and therefore facilitating access to AL.

2.2 Active Learning - scenarios

Before we study state-of-the-art active learning strategies, we have to classify the ways where the data for labelling might come from. There are generally several sources for unannotated data:

1. **Membership query synthesis** - data generation from the range of all possible values. An algorithm generates data which are seemingly correct and which can pass through the model. For instance, for large NLP language models it can be any combination of token indices from vocabulary up to the maximum length of the sequence. If the maximum sequence length is 256 tokens and there are 30000 tokens (including PAD token for padding sequences to the maximum length), then the range of possible values is 256^{30000}. It is obvious that although there are roughly speaking infinite combinations of tokens, only a small fraction is semantically correct, i.e. represent a real text, not a collection of letters. That's why a simple data generation process is not applicable for NLP. However, when dealing with other tasks, for example, robot orientation where data represent its location, this approach might give more sense.

2. **Stream-based selective sampling** - this method works in the way that the new data entry is selected randomly from a sampled distribution of the input space and then the decision whether it is worth labelling is made by an annotator. This approach is also not well suited for NLP tasks, since we do not know the distribution the data comes from.

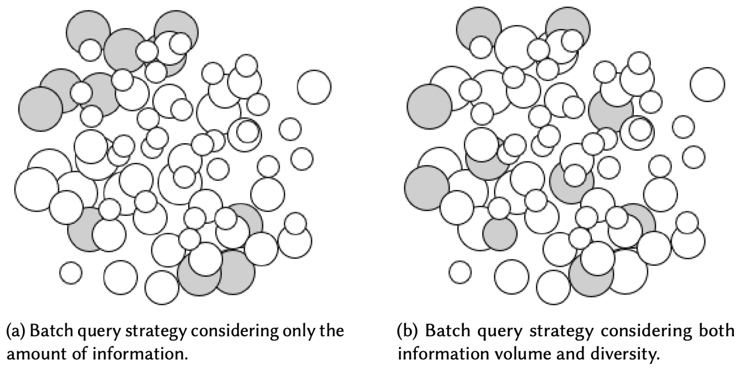
3. **Pool-based sampling** - this is the most widespread way of choosing data for labeling. Its principal difference from the former two methods is that the new examples are not generated, but drawn from a pool of unlabelled data. The assumption is then that such collection exists. For example, it might be a news agency with numerous articles that have to be annotated, or a repository of source codes for analysis. The method works with the whole unlabelled set of data that are at researcher's disposal. A strategy selects examples that are worth labelling basing on some informativeness criteria and returns data back to the annotator. The annotator knows that the data are not generated, but drawn from the dataset. The informativeness criteria is mostly a heuristic function that guesses how useful a data entry for the model training is. The annotator either agrees with the selection or finds better-suited examples ordered by informativeness metric function.

The methods described above traditionally work in the following way: a new data entry is either generated or drawn from an unlabelled set. An annotator decides whether to add this entry to the training dataset and then the model is retrained with the new data entry. This routine is valid mostly for models with low number of trainable parameters, where a single additional example is capable of causing change in model's parameters. However, in tasks involving large language models like BERT [DCLT18a], it is inadequate to increase training set by only one data entry, which won't affect it much. These models are trained with larger data batches, therefore it gives more sense to draw a batch of new unlabelled data and update training set with more data at once. One exception from this idea might be a large foundation model with fixed weights, where the only trainable part is a classification head, which does not have so much parameters as the whole model [TMC+21].

However, batch selection instead of mere one data entry poses a significant challenge on AL strategy design.

2.2.1 Batch Mode active learning

Many AL strategies that are not designed specifically for NLP tasks deal with single data entry, not a batch. The first obvious solution might be to use these strategies (known as *one-by-one strategy*) more than once and collect a batch of a specified size by repeating strategy on more data entries. As a result, a batch of data is returned, however this approach does not take into consideration a possibility of collecting a batch with elements which are highly correlated. An algorithm might select individual instances which might be useful for the model, however they might be all the same or close to each other. By collecting a batch of data to put into the model we rather want this batch to consist of a representative sample of the target distribution and be as diverse as possible. An example of two possible outcomes of collecting batches for further training is depicted in figure 2.2.


(a) Batch query strategy considering only the amount of information.

(b) Batch query strategy considering both information volume and diversity.

Figure 2.2: Batch AL vs naive approach, source: [RXC+21]

It can be clearly seen from the left part of the picture that a naive approach selects a batch with elements of highest individual informativeness. These data points are located in the same place of the concept space and the studied space is not covered well. That's why the compound effect of such batch on the model performance might be suboptimal. Hence, it is crucial to study not only AL strategies for single element selection, but also so-called *batched active learning strategies* which attempt to find better coverage of the studied concept space and thus stabilise and improve training.

2.3 State-of-the-art strategies

In this section I present several state-of-the-art active learning strategies that are actively used in related literature and that show promising results in empirical studies. The list is not exhaustive, however these strategies many times are found superior to others.

2.3.1 One-by-one strategies

This is the first class of active learning strategies. One-by-one strategies were originally developed to serve the purpose of selection of only one unlabelled example with further annotation and addition to the training dataset. These methods are usually based on the principle of assigning each unlabelled example some qualitative score, which determines, to what extent this example is valuable for the model. The example with the best score is selected.

Random sampling

Random sampling is a trivial method. It is easy to implement by taking a random sample of indices uniformly from the list of indices from 1 to N, where N is length of a list of items to choose from. Each element is sampled with the same probability from the uniform distribution. Hence, each element from the unlabelled set has the same chance of being added to the next training batch. This method does not use any information about the dataset, about the model, thus it can act as a baseline for comparison with other more sophisticated methods. Moreover, it is the fastest method due to its simplicity. Random sampling has nice characteristics for large NLP models and active learning: it mostly chooses diverse set of data and its sample can be proven to be similarly distributed as the target population, i.e. elements from distribution of classes in the target population is more or less kept the same in the sample as well - thus less worries about "forgetting" less frequent categories.

Uncertainty Sampling

Uncertainty sampling is a class of methods based on the principle of finding elements from unlabelled set for which the model provides the lowest certainty score. The assumption is that the model generates scores that define preference scores for each output category. The higher the score, the more certain the model about this category is. This assumption is also supported by the way how most neural classification and tagging models work - for the decision making they choose a category with the highest softmax score. That's why the category with the lowest softmax score is the least likely.

The strategy works in a way that it finds those entries of the unlabelled dataset, for which the probability distribution among possible classes is the closest to the uniform. For instance, in binary classification (where there is usually only one number denoting the likelihood of element being from class 0) this would be the probability close to 0.5. It means that the model is uncertain where this element belongs to. Hence, this element is worth labelling, since knowing the true category for this entry would help the model to discriminate better along the classes borderline.

For more categories this method works in the following way: a dataset entry is considered the least certain if the maximum value among all probability scores (or other kinds of scores not necessarily summing up to 1) is the lowest among all other dataset entries. Then, such entry is said to be the least certain and is sent for labelling. Generally, for N categories ($N \geq 2$) the formula for finding the least certain element is"

$$x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x)$$

where $\hat{y} = \arg\max_y P_\theta(y|x)$, or the label of the category with the highest posterior probability under the model θ.

The most widespread metric function for most uncertain element finding though is entropy score, which expresses the amount of information needed for description of data distribution. Thus, the scoring function for this AL strategy is:

$$x^*_H = \arg\max_x - \sum_i P_\theta(y_i|x) \log P_\theta(y_i|x)$$

,

where y_i ranges over all possible labellings. Overall, both methods are equivalent to querying the instance with a class posterior probability closest to 0.5.

Overall, in my project I have implemented two strategies from this class of methods - least confidence and entropy sampling.

Monte Carlo Dropout Sampling

This method is an improvement of the previous strategy. The difference is in the way how the uncertainty is calculated - here it is implemented with the help of Monte Carlo Dropout with n inference cycles [GG16]. Sources like [GG16] reference number 10 as a good start for experimentation with this parameter. Authors [EDHG$^+$20] claim that this method shows better results and consistently generates better batches of unlabelled examples in terms of representativeness and diversity, see figure 2.3. However, this method is considerably more complex and lasts much longer than uncertainty sampling discussed above, see figure 2.4.

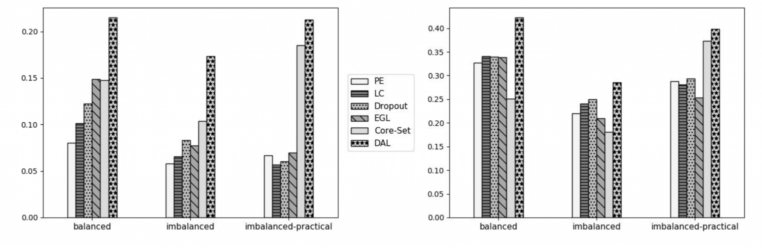
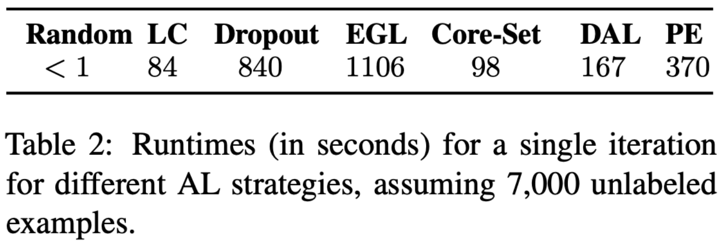


Figure 2.3: The difference of active learning strategies from the perspective of diversity (left) and representativeness (right) of sampled unlabelled examples, selected by different AL strategies. In this figure uncertainty sample is shown as LC, Monte Carlo Dropout Sampling is shown as Dropout. Source: [EDHG$^+$20]

There also other smart strategies which are based on different calculations of expected contribution of an unlabelled example to model's weights, for example *Expected Gradient Length* [HCR$^+$16]. This method calculates an expected change of a loss function with the help of the norm of its gradient for each example and chooses the one with the largest expected change. The idea behind this method lies in the assumption that inputs that cause larger changes in network's parameters (and thus loss function gradients are used as the first input to the back-propagation algorithm) will make the model converge faster. Such approach according by newer studies ([SN20], [RXC+21]) are not significantly better than uncertainty sampling. Moreover, they are extremely time-consuming compared to easier methods, see again 2.4.



Random	LC	Dropout	EGL	Core-Set	DAL	PE
< 1	84	840	1106	98	167	370

Table 2: Runtimes (in seconds) for a single iteration for different AL strategies, assuming 7,000 unlabeled examples.

Figure 2.4: Illustration on difference of time needed for a single iteration of different AL strategies. Uncertainty sampling is labelled as LC, Monte Carlo Dropout Sampling as Dropout, Expected gradient Length is shown as EGL. Source: [EDHG+20]

2.3.2 Batch methods

Batch methods are built on the idea of finding a collection of unlabelled examples instead of a single example. The focus is then switched towards the aim of finding as diverse batch of data as possible. In the light of large language models this concept is more attractive and gives more sense when dealing with small datasets in the beginning of labelling training datasets. Newest sources [ZWH+22] state that these methods are currently achieving state-of-the-art results.

CoreSet

CoreSet [SS18] is one of the earliest studied methods of batch strategies class. It is based on analysis of vector representation of unlabelled examples and choice of n elements (where n is the size of unlabelled batch to draw from existing pool) which forms a solid representation of the whole unlabelled collection. The selection is done by application of core-set method [Wik22a]. The whole problem of active learning is redefined by this method to the core-set selection problem. Authors of [AZK+20] state based on their own empirical studies that this method is much more accurate than strategies that attempt to assess individual characteristics of unlabelled examples.

BADGE

BADGE [AZK+20] is an acronym for *Batch Active learning by Diverse Gradient Embeddings*. This method is different from others in a way of how it attempts to find the most diverse batch of unlabelled data. It does not use model output like it is usually done in K-means 2.3.2 or CoreSet method 2.3.2, but it computes gradient of the predicted category, with respect to the parameters of the last layer. As a result, a vector of gradients is returned. The method is then inspired by expected gradient length strategy and the assumption here is pretty similar - when the norm of the gradient vector is large, the parameters of the network are needed to change more drastically in order to be "more sure" about the true category.

As soon as so-called *gradient embeddings* are collected, they are clustered. However, they are not clustered by K-means or any other algorithm. They are approximately "clustered" by an easy method of smart K-means algorithm centroids initialisation - K-means++ [AV07a], [AV07b], which attempts to overcome known issues with K-means centroids initialisation instability.

Authors of BADGE claim that the strategy does not have weaknesses of those methods that served as inspiration to this one - uncertainty sampling, expected gradient length, K-means. These methods, judged by empirical studies, work consistently throughout different neural architectures and are therefore reliable. It is empirically known that least confidence method sometimes gives the most accurate results and sometimes might perform worse than random strategy - BADGE is however designed in a way that eliminates this weakness, which is shown in figure 2.6. The pseudo code of the algorithm is depicted in figure 2.5.

```
Algorithm 1 BADGE: Batch Active learning by Diverse Gradient Embeddings
Require: Neural network $f(x;\theta)$, unlabeled pool of examples $U$, initial number of examples $M$, number of
    iterations $T$, number of examples in a batch $B$.
1: Labeled dataset $S \leftarrow M$ examples drawn uniformly at random from $U$ together with queried labels.
2: Train an initial model $\theta_1$ on $S$ by minimizing $\mathbb{E}_S[\ell_{\mathrm{CE}}(f(x;\theta),y)]$.
3: for $t = 1, 2, \ldots, T$: do
4:    For all examples $x$ in $U \setminus S$:
         1.  Compute its hypothetical label $\hat{y}(x) = h_{\theta_t}(x)$.
         2.  Compute gradient embedding $g_x = \frac{\partial}{\partial \theta_{\mathrm{out}}} \ell_{\mathrm{CE}}(f(x;\theta), \hat{y}(x))|_{\theta=\theta_t}$, where $\theta_{\mathrm{out}}$ refers to param-
             eters of the final (output) layer.
5:    Compute $S_t$, a random subset of $U \setminus S$, using the $k$-MEANS++ seeding algorithm on $\{g_x : x \in U \setminus S\}$
      and query for their labels.
6:    $S \leftarrow S \cup S_t$.
7:    Train a model $\theta_{t+1}$ on $S$ by minimizing $\mathbb{E}_S[\ell_{\mathrm{CE}}(f(x;\theta),y)]$.
8: end for
9: return Final model $\theta_{T+1}$.
```

Figure 2.5: BADGE strategy pseudo code, source: [AZK+20]

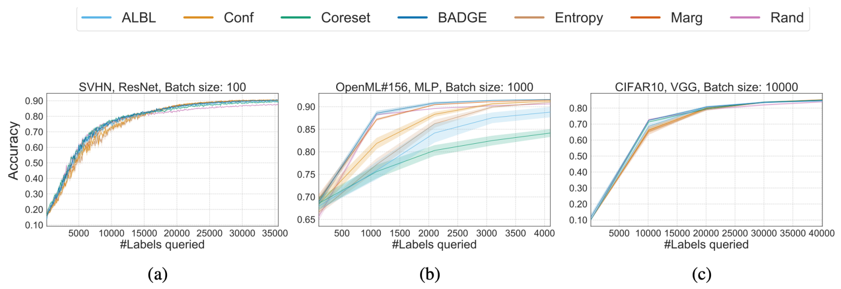


Figure 2.6: BADGE algorithm comparison with simpler methods. Uncertainty sampling is labelled here as Entropy. Source: [AZK$^+$20]

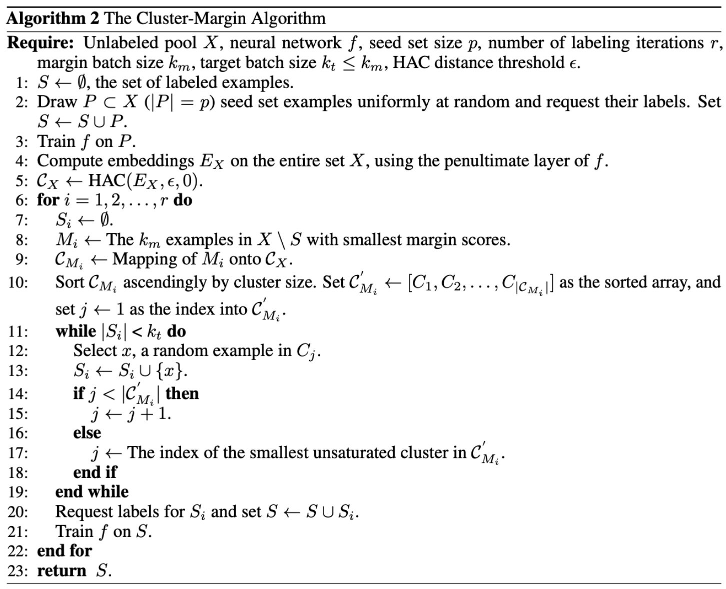
K-means

One of the most straightforward methods which shows great performance [Zhd19] is a K-means approach of finding K clusters of data close to each other [JH10] in terms of Euclidean distance, but applied for finding K clusters in an unlabelled dataset. Since this dataset is often enormous, setting parameter k to higher values (10-100) is justified. The method firstly finds centroids for each cluster, then for each centroid the closest input unlabelled example is found and selected for the next batch. Hence, the method makes sure the sampled texts are diverse.

K-means is a great method, which combines well with one-by-one strategies. For example, [Zhd19] incorporates uncertainty sampling and introduces weighted K-means method, where each input image is assigned a weight from an uncertainty calculation function. Weighted K-means can be further improved in terms of runtime when pre-filtered to leave only the most complicated (uncertain) examples. Basing on the empirical results from [Zhd19] I have decided not to implement both weighted and classical K-means approach and decided for the latter for the experimental part.

Cluster Margin

Cluster Margin [CDG$^+$21b] is one of the newest methods, which compared to BADGE method is not that computationally expensive. Its implementation is based on leveraging Hierarchical Agglomerative Clustering (HAC) [Wik22b] to diversify batches of examples that the model is least confident on. This algorithm is executed only once as a preprocessing step before running AL experiment. While sampling, the algorithm utilises computed HAC clusters and with the help of round robin scheme [Wik22d] chooses the best candidates. This algorithm is more appropriate for problems of large unlabelled batch selection. Empirical results from [CDG+21b] show that this method has almost identical performance compared to BADGE and benefits just from the runtime perspective. Its pseudo code is however much more complicated, see 2.7.



```
**Algorithm 2** The Cluster-Margin Algorithm
**Require:** Unlabeled pool $X$, neural network $f$, seed set size $p$, number of labeling iterations $r$,
    margin batch size $k_m$, target batch size $k_t \leq k_m$, HAC distance threshold $\epsilon$.
$S \leftarrow \emptyset$, the set of labeled examples.
Draw $P \subset X$ ($|P| = p$) seed set examples uniformly at random and request their labels. Set
    $S \leftarrow S \cup P$.
Train $f$ on $P$.
Compute embeddings $E_X$ on the entire set $X$, using the penultimate layer of $f$.
$\mathcal{C}_X \leftarrow$ HAC($E_X, \epsilon, 0$).
**for** $i = 1, 2, \ldots, r$ **do**
  $S_i \leftarrow \emptyset$.
  $M_i \leftarrow$ The $k_m$ examples in $X \setminus S$ with smallest margin scores.
  $\mathcal{C}_{M_i} \leftarrow$ Mapping of $M_i$ onto $\mathcal{C}_X$.
  Sort $\mathcal{C}_{M_i}$ ascendingly by cluster size. Set $\mathcal{C}'_{M_i} \leftarrow [C_1, C_2, \ldots, C_{|\mathcal{C}_{M_i}|}]$ as the sorted array, and
  set $j \leftarrow 1$ as the index into $\mathcal{C}'_{M_i}$.
  **while** $|S_i| < k_t$ **do**
    Select $x$, a random example in $C_j$.
    $S_i \leftarrow S_i \cup \{x\}$.
    **if** $j < |\mathcal{C}'_{M_i}|$ **then**
      $j \leftarrow j + 1$.
    **else**
      $j \leftarrow$ The index of the smallest unsaturated cluster in $\mathcal{C}'_{M_i}$.
    **end if**
  **end while**
  Request labels for $S_i$ and set $S \leftarrow S \cup S_i$.
  Train $f$ on $S$.
**end for**
**return** $S$.
```

Figure 2.7: Cluster Margin algorithm pseudo code. Source: [CDG+21b]

Query-by-Committee approach

The approach worth mentioning in this chapter is a so-called Query-by-Committee (QBC) algorithm [MM04], which is merely an ensemble method for new unlabelled example sampling. This method works as a classical ensemble of learners: there is a set of independent machine learning models (with several differently formulated hypotheses) which are trained on the same dataset. An ensemble chooses unlabelled example for adding to the training dataset basing on some criteria of maximum disagreement. A disagreement metric can be calculated in many ways, for example *vote entropy* method is formulated as:

$$x^*_{VE} = \arg\max_x - \sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C}$$

,

where y_i all possible labels of the input, $V(y_i)$ is a number of "votes" for this label among all estimators, C is a number of estimators in the ensemble.

Unfortunately, sources like [ZWH$^+$22] claim that this method is infeasible for deep learning models, as these models are themselves too complex and require lots of computation to train, thus creating a committee of such models is an ultimate strain for the hardware this strategy will be tested. Authors suggest using this method with simpler models rather than deep learning models, which in case of language models act like an ensemble themselves thanks to the architecture of attention heads.

2.4 Aim of the project

The reason why I chose this project was my motivation to explore the framework of active learning and make it possible to be re-used in other projects and further research. I have built a prototype for an active learning framework in the form of a Python package, which will complement existing libraries for NLP model training. My ambitions were the following:

1. Implement an active learning pipeline depicted in the figure 2.1. More on that in section 3.
2. Create a framework which deals mainly with *transformers* models implemented in PyTorch [1].
3. Implement and compare five different active learning strategies. Is there a difference on final performance between repetitive uncertainty sampling methods and advanced batch sampling methods?
4. Compare the active learning strategies for a common classification NLP task. Then, repeat the experiment on a tagging task. Finally, implement a pipeline for training NLI tasks and test them on a Czech NLI dataset.
5. Implement reporting and visualizations for better understanding of active learning impact on model performance.
6. My wish was to implement the whole framework in a modular way, so that it consists of logical parts put together in a pipeline. Each active learning strategy has its own implementation, so that it can be easily plugged into the pipeline to enhance user-friendliness.

[1]For more information please visit https://huggingface.co/docs/transformers/index.

Chapter 3

Proposed solution

The aim of this project was to study the active learning framework and design a system of running active learning experiments.

I decided to implement a library for seamless active learning experiments management with comparison and tracking capabilities. The aim of this library is to streamline the experimental process by abstracting all necessary AL steps and giving the user only those parts of the implementation which are important not only for experiments and comparisons, but also for further development of the system. This package has to be a baseline solution with further possible improvements which can make active learning methodology more accessible to research community. It is flexible, configurable and extendable.

This project consists of three logical parts: research, system design and implementation, and experimentation. The system which is then used for experimentation and which is actually available as an open-source package [1], is the main output of this thesis. To the best of my knowledge, there are a few solutions available in the internet nowadays which somehow deal with active learning methodology ([EDHG^{+}20], [DH]), but none of them are ready to be used in real world scenarios and are not easily extendable. That's why I find my package utterly essential.

The proposed solution is a Python open-source library, which can be used for experimenting with different AL strategies and neural models found in

[1]The package can be found at https://github.com/DevKretov/ntu_nlp_al.

transformers ecosystem. It can be seen as a platform to build new models, new architectures, new strategies and experiment on. The package allows for running complete active learning pipeline for experimental purposes or for real-life training with the help of active learning. It implements the following things:

1. Data load and data preparation. The datasets in tabular format are imported into the system and textual information is preprocessed and prepared for training models.
2. Model administration. A pre-trained model is imported, a classification (or any other) head is added upon the model graph. These models can be downloaded from hosted environments or loaded from local storage.
3. Strategies addition. Active learning strategies can be easily added and included into training.
4. Logging. All the essential information for a data scientist is saved for further investigation and decision making.
5. Training pipeline. A pipeline which combines everything above and orchestrates the whole idea of active learning. It prepares data, a model, selects a strategy and runs training routine with further logging and testing and new batch selection basing on AL strategy. Then, it also checkpoints all models trained during the experiment and tracks the best one. A training pipeline can be run either on CPU or GPU interchangeably.
6. Configuration. The whole library is highly configurable and has a large choice of variables to tweak in the configuration file.
7. User-friendliness. The package can be used out of the box and is accessible to a wide range of researchers who work with NLP models and want to improve their performance. The library is modular and thus it is highly extendable, so that others can improve it in the future.

3.1 Used libraries

The platform is built upon these renowned libraries:

1. **transformers** [1] - an open-source library and an ecosystem which implements various Transformer-based models primarily in NLP sphere.

[1]For more information please visit https://huggingface.co/docs/transformers/index.

Nowadays it is one of the most popular and widespread frameworks for NLP research. It has a deep integration with two most favorable deep learning engines - PyTorch and TensorFlow, thus the models built in Transformers are easily trainable on GPUs, TPUs, they can be easily deployed almost everywhere. Implementation of all necessary steps for neural models training and testing is straightforward thanks to the deep level of abstraction given by its API. Additionally, thanks to the growing community, it is now the standard place to publish new models and architectures so that others can replicate the results found in various research papers. That's why my solution is also deeply based on this library and implements all active learning steps in transformers ecosystem. This decision enables user to configure the whole framework as he wishes without the need for extensive code rewriting. It is almost trivial to change the pre-trained model architecture, configure hyperparameters, track the model's performance and deploy the best one to HuggingFace Hub of models [1].

Moreover, since this library is now actively enhanced by models from Computer Vision, audio processing, multi-modal architectures, my solution is easily transferable to other fields of AI research and in perspective is simply applicable across AI domains.

2. **Datasets** [2] - a library made by creators of transformers which is fully integrated with transformers. It simplifies the way the data are obtained, preprocessed, utilised and published. It is in a way a Git for datasets, not only for usage in transformers library. The framework has numerous features that help manipulate large datasets which cannot fit into memory at once by batching them, thus enabling off-the-shelf usage of data with large models like ROBERTA [LOG$^+$19]. I use this library to implement all steps needed for data to be prepared for training and evaluation and since AL is a framework which works primarily with data sources, the choice of this library helped me a lot to implement a robust data processing pipeline throughout AL iterations.

3. **Google Colab** [3] - a cloud-based solution from Google for running experiments in notebooks hosted on Google's virtual machines.

4. **Weights and Biases** [4] - a MLOps library for tracking and comparing experiments, a so-called Git for machine learning runs. It is a game-changer for those who experiment a lot. It provides a rich API with vast capabilities of saving essential information about training models, including hyperparameters tracking, logging, evaluation and metrics, models artifacts and so on.

[1]Models hub can be found on https://huggingface.co/models.
[2]For more information please visit https://huggingface.co/docs/datasets/index.
[3]For more information please visit https://colab.research.google.com/.
[4]For more information please visit https://wandb.ai/.

I have integrated all the necessary steps of active learning with this library, so that experimentation with different strategies and their comparison is drastically simplified.

3.2 Architecture

It consists of four main components:

1. **Model class:** an abstraction of the neural model I use to train. Currently it supports all types of classification models (namely all models with a suffix *"ForSequenceClassification"*) implemented in transformers library. Additionally, I have also implemented models for tagging (namely all models with a suffix *"ForTokenClassification"*). This class implements model initialisation, reinitialisation and storing.
2. **Dataset class:** an abstraction of the dataset and all data-related operations: reading dataset from local storage or from the cloud (currently from Huggingface datasets Hub), preprocessing text and getting it ready for training with the help of tokenizers library, preparing different dataset slices and postprocessing. Technically, there is an abstract class *Dataset* which contains generic methods shared among all its implementations (like downloading dataset, truncating or shuffling datasets, etc). Then, there are two classes *ClassificationDataset* and *TokenClassificationDataset*, which implement all task-specific functions (labels encoding, updates, dataloaders preparation).
3. **Strategy class:** an abstraction of an AL strategy. The particular implementation of a strategy is written as a separate inherited class. I provide several strategies for classification and two strategies for tagging tasks. Each strategy has to implement *query* method, which accepts a dataloader with unlabelled texts and returns indices of selected texts. All data shuffling is performed outside of a strategy instance. Several strategies also store reference to the currently trained model, so that they can access model's weights for best candidates selection.
4. **Trainer class:** an orchestrator of the whole AL pipeline. It is responsible for everything in the platform and is used for communication and coordination of three other main classes in the framework. The detailed logic of a trainer is described in the section below. In a nutshell, the whole experiment takes place and is logged in this class.

3.3 Process description

The proposed solution is created to serve two main purposes: to evaluate feasibility of applying active learning framework on a particular task and to run active learning pipeline in a real-life scenario. Here is the possible process:

Feasibility evaluation

My library is designed to abide by following process:

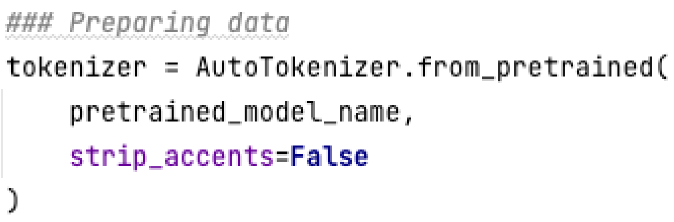
1. **Tokenizer.** The first thing that has to be initialised is a tokenizer. It is loaded with the help of *transformers* API and is downloaded either from HuggingFace (HF) models hub or from a locally stored model. My solution utilises loading HF API, hence it is capable of loading tokenizer from extremely large possible storages and databases, including cloud storages.

   The tokenizer is initialised with the help of a dictionary file, where all trained tokens are stored. For more detailed implementation of tokenizers please read in detail about WordPiece [WSC$^+$16] or SentencePiece [KR18] algorithms.

   The tokenizer has to be carefully loaded in case when Czech data is going to be processed: many tokenizers do not have their corresponding configuration files (stored in the same folder as a tokenizer). Most Czech models (like RobeCzech [SNSS21] or Czert [SPP$^+$21]) have carefully designed tokenizers' configs, however it must be taken into account while loading own tokenizers. For Czech language it is crucial to load tokenizers without stripping accents (shown in figure 3.1), otherwise the results might be suboptimal.

   The tokenizer is loaded first, since it is then used to preprocess input data and convert string tokens to integer indices of the words in its dictionary.

2. **Data load.** Firstly, the dataset in the CSV format is downloaded. It can be a dataset stored locally in a file system or a dataset stored at HuggingFace Datasets hub. Each dataset must have its own configuration file in which the system will read out all necessary information: the names of columns for both input and target, a flag whether the dataset can be found on the HF Hub, paths to train-validation-test splits in case of a



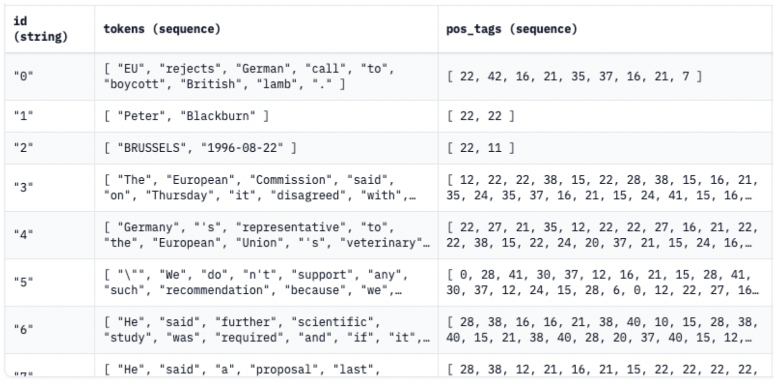
```
### Preparing data
tokenizer = AutoTokenizer.from_pretrained(
    pretrained_model_name,
    strip_accents=False
)
```

Figure 3.1: Correct way of loading HuggingFace tokenizer for correct Czech language handling.

locally stored dataset, and names of the splits. Only datasets for these tasks can be processed at the moment:

a. Classification - a dataset is typically a two-column CSV file with the first column intended for the input text and the second column intended for the category of the text.

b. Tokens tagging - a dataset is typically a two-column CSV file with the first column with a serialized Python-like list of string tokens (words) and the second column with a serialized Python-like list of integer tags labels.
   A tagging dataset example is depicted in figure 3.2.



id (string)	tokens (sequence)	pos_tags (sequence)
"0"	["EU", "rejects", "German", "call", "to", "boycott", "British", "lamb", "."]	[22, 42, 16, 21, 35, 37, 16, 21, 7]
"1"	["Peter", "Blackburn"]	[22, 22]
"2"	["BRUSSELS", "1996-08-22"]	[22, 11]
"3"	["The", "European", "Commission", "said", "on", "Thursday", "it", "disagreed", "with",…	[12, 22, 22, 38, 15, 22, 28, 38, 15, 16, 21, 35, 24, 35, 37, 16, 21, 15, 24, 41, 15, 16,…
"4"	["Germany", "'s", "representative", "to", "the", "European", "Union", "'s", "veterinary"…	[22, 27, 21, 35, 12, 22, 22, 27, 16, 21, 22, 22, 38, 15, 22, 24, 20, 37, 21, 15, 24, 16,…
"5"	["\"", "We", "do", "n't", "support", "any", "such", "recommendation", "because", "we",…	[0, 28, 41, 30, 37, 12, 16, 21, 15, 28, 41, 30, 37, 12, 24, 15, 28, 6, 0, 12, 22, 27, 16…
"6"	["He", "said", "further", "scientific", "study", "was", "required", "and", "if", "it",…	[28, 38, 16, 16, 21, 38, 40, 10, 15, 28, 38, 40, 15, 21, 38, 40, 28, 20, 37, 40, 15, 12,…
"7"	["He", "said", "a", "proposal", "last",	[28, 38, 12, 21, 16, 21, 15, 22, 22, 22, 22,

Figure 3.2: An example of a tagging dataset. There are two important columns that must be present in the table - one with tokenized text ("*tokens*" in the picture) and one with target labels for each token ("*pos_tags*" in the picture). For each row the lengths of these two lists have to be the same.

c. Fact checking - a dataset is typically a three-column CSV file with the first column being a **claim** in string format, the second column - **evidence** in string format and the third column - a **target** label stating if the evidence supports the claim fully or partially, or neither. In fact, this type of data is inherently converted into a classification dataset by appending claim string to the evidence string. These two strings are additionally separated by a technical token [SEP], which tells the neural model that there are two sequences in the input. The visualisation of this approach is given in figure 3.3.

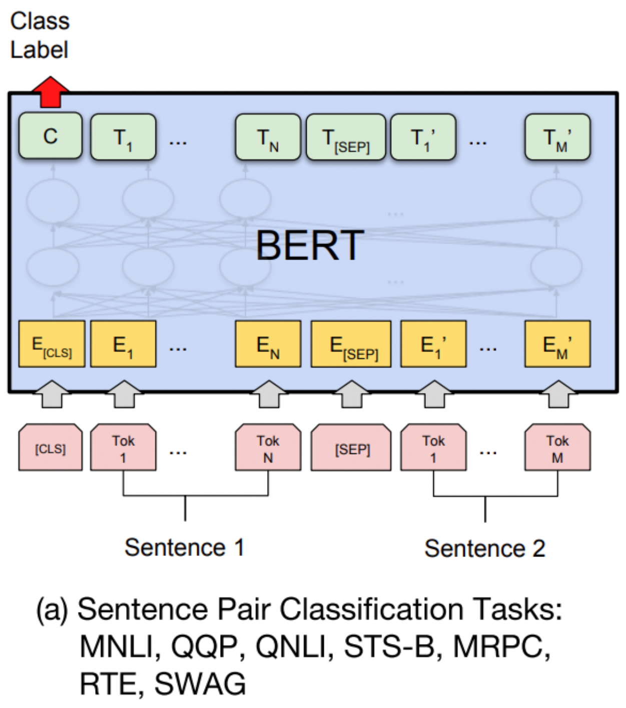
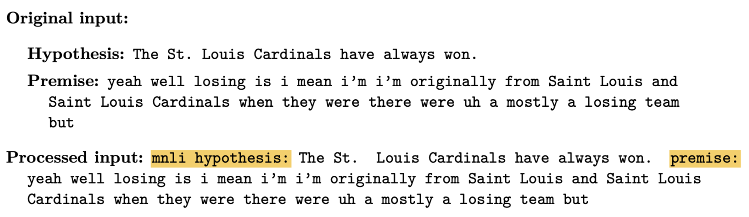


Figure 3.3: Illustration on Fine-tuning BERT-like models on NLI task. Source: [DCLT18a].

This is however just one possible way of dealing with NLI datasets. Another possibility is to use cross-encoders [RG19] and then compute cosine similarity between two inputs. However, I decided to stick to the implementation supported by original paper [DCLT18a] and many others [WFK$^+$21], [RG19] and is one of possible ways of treating this task. For example, figure 3.4 shows how T5 developers from Google preprocess NLI inputs for the classification model.


Original input:

Hypothesis: `The St. Louis Cardinals have always won.`

Premise: `yeah well losing is i mean i'm i'm originally from Saint Louis and Saint Louis Cardinals when they were there were uh a mostly a losing team but`

Processed input: `mnli hypothesis: The St.  Louis Cardinals have always won.  premise: yeah well losing is i mean i'm i'm originally from Saint Louis and Saint Louis Cardinals when they were there were uh a mostly a losing team but`

Figure 3.4: Illustration on NLI inputs preprocessing. Highlighted texts act as separator for the model, so that attention mechanism knows that there are two logical parts in the input. Source: [WFK$^+$21].

Overall, we have two different ways of treating input data, that's why

Dataset class, which acts here rather as an interface, has two implementations - *ClassificationDataset*, *TokenClassificationDataset*. Both these datasets implement all data-related tasks of reading, preprocessing, dataloaders preparation and datasets actualization in each AL iteration.

A *Dataset* class instance is initialized with fully functional tokenizer. After the data are loaded either from a local CSV file or from a remotely hosted environment, the textual data is converted into lists of token IDs, target categories are converted into their corresponding numerical representations. Conversion of text into lists of integers is conducted by the tokenizer with the same set of parameters: maximum sequence length, truncation flag, padding parameter. After the preparation there will be a standardized list of lists of equal length representing each input text. This format is already prepared to be used in training routine.

In order to conduct active learning experiments, the original training dataset, which is fully labelled, has to be divided into two parts - into a labelled and presumably unlabelled parts. The one that is considered labelled is always used in training, while the rest of the data is used for querying and obtaining new data for adding to the training dataset. Examples drawn from presumably unseen dataset are enhanced with their labels and then passed to the training dataset. By gradually repeating this process we simulate the real process of active learning when there is no labelled data available beforehand.

3. **Model initialisation.** The model is initialised with the help of its identifier in case when the model has to be downloaded from HF Models hub or with the model folder path found on either local or cloud storage. The model is loaded with the help of AutoModel subclass, which resolves the particular architecture basing on configuration content - model initialisation process is therefore fully dependable on its configuration file.

4. **Trainer initialisation.** A trainer is an orchestrator and the central part of the whole process. It has the complete training and testing logic implemented inside.

   A trainer instance is equipped with initialised model instance and a dataset instance. Then, data loaders for training loop are prepared:

   a. In case when active learning scenario is enabled, the training dataset is created from the "labelled" part of the original training dataset. Additionally, a dataloader for the "unlabelled" part of the training dataset is created for active learning selection procedure. Otherwise the dataloader for training is initialised with the full training dataset.

   b. In case when the user knows that the dataset is imbalanced (there are strong groups of labels that form a dominant part of the dataset, whereas other labels are represented by a small - considerably smaller - number of examples), the dataloader can be configured in a way that enables data resampling basing on labels' probability distribution. Resampling can sometimes stabilise training process and make model more attentive to less frequent classes [HBFF22]. Resampler is a natively implemented mechanism in PyTorch, it is a part of *data* package and is called **WeightedRandomSampler**. Resampling is enabled by default.

c. Training dataloader is initialised with shuffling set to True [NTD+22].
d. Dataloaders for validation and test splits of the dataset are also prepared.

After the dataloaders for the training loop are prepared, the rest necessary parts of training routine are configured. In my solution I use AdamW [LH17] optimizer with learning rate set from configuration file. I also implemented learning rate scheduler, but since fine-tuning of BERT-like models is conducted on just several epochs (up to 5 mostly), there is no much need for its introduction.

After that, the target training device is selected. Currently the solution is capable of running on GPUs and CPUs. GPU support is inherenty implemented in PyTorch, there is no need for additional tweaking. In case when no GPU is available, training and inference are run on a CPU.

Then, evaluation metrics are added to the list of all metrics that are needed to be tracked during training and testing. These metrics have to be compatible with HF Datasets library, where there have to be several functional methods for each metrics instance. Traditionally, these metrics are added: accuracy, precision, recall, F1 score. All metrics must be initialisable with the help of *load_metric* function. In case of such metrics as F1 one needs to bear in mind that they are more flexible than standard accuracy metric, thus it is needed to be configured thereafter. Nevertheless, as for the beginning of 2023 metrics are **deprecated** in *datasets* library and now it is a separate library called Evaluate [1].

In case when local visualisation of AL strategies performance is required, visualisation module is initialised.

It is possible to run a so-called "full training", where the complete training dataset is used to train the model. Full training mode is important for the initial comparison of the effectiveness of AL pipeline. It gives metric scores which we then try to reproduce or even beat during active learning phase. Full training serves as a baseline for an AL experiment.

Finally, the orchestrator starts AL experiment with selected AL strategy. An AL experiment is configured with the following parameters:

a. Number of AL iterations.
b. Initial training dataset size.
c. Number of training entries to add after AL strategy application.

[1]The library can be found at https://huggingface.co/docs/evaluate/index.

d. Number of training epochs in each AL step.
e. Selected AL strategy.
f. Training batch size, validation batch size and test batch size.
g. Debug mode - whether to run the whole AL experiment in debugging mode - only 5 batches for training, validation and testing and then AL strategy application. Overall, there will be 5 AL iterations. I use this mode while testing the integrity of the whole process.
h. Path to save the best performing model.

The lines below describe the mechanism of initialisation of AL Trainer. As soon as all preparatory steps are finished, an experiment can start. An experiment consists of N AL iterations defined by the user. In each iteration we execute the same routine: model training and evaluation, AL strategies querying and new "unlabelled" batch of data selection, datasets actualisation, snapshots creation:

a. Model training is implemented by following the best practices for PyTorch models training and specifically *transformers* models fine-tuning [PB18], [SQXH19]. In each training method call the model is reinitialised and learned weights during previous AL iterations are reset. The optimizer is reinitialised. Then, a standard PyTorch training routine is run: forward pass with the next batch, loss calculation, back-propagation, optimizer step, loss function value logging. After the whole training dataset is passed, average loss is printed. Then, a new epoch is started. Currently calculated average loss is printed together with a progress bar with the help of TQDM library [dCLLA+22].

   After training, the model is evaluated on a validation set, then on a test set. The decision of running evaluation on a validation set only after all training epochs is made due to high computational cost of running validation after each training epoch in each AL iteration. It is however required for best model selection before its evaluation on previously unseen data. Hence, the results that are reported in this work could be possibly lower than if the evaluation on a validation set with further best model selection was run after each training epoch. Evaluation on the test set is done in the end of AL iteration, so that it is possible to compare the results of a model trained on a part of the dataset during each active learning step and the model trained on the full dataset.

   Evaluation is implemented in a similar way as training, however the difference is in the way the metrics are collected and reported. During evaluation, AL trainer calculates all user-specified metric scores depending on the type of a training task: in case of classification, a flat list of predicted texts labels is collected for the whole dataset and then the metrics are calculated, whereas for a tagging task the list of masked labels is collected both for predictions and for true values, then they are transformed into labels in string format due to BIO tagging scheme [Wik22c] requirements and sequential evaluation specifics. After the list of tags is collected for each input text, specific tagging task metrics are calculated with the help of *seqeval* package [Nak18]. Then, these metrics scores are printed out, so that the user sees how the model is performing.

A model with the best F1 score for validation set is saved on disk as an artifact. F1 score metric could be changed in the configuration file if necessary.

b. After the model is trained, an AL strategy instance is given actual dataloader with "unlabelled" data to query from. This is done by an implementation of *query* method of a base AL strategy interface, which gives the whole solution great flexibility. Basing on a selected AL strategy, the list of indices of rows found in the "unlabelled" dataset is returned.

c. Both "labelled" and "unlabelled" datasets are updated: rows selected by an AL strategy move from the "unlabelled" set to the current "labelled" dataset and enlarge it. Then, integrity validations are run so that we are sure that no data are lost during this change. After all, the part of the "unlabelled" dataset which was moved to the "labelled" dataset is saved as a separate CSV file for AL strategy selections tracking and reverse-engineering. Current "labelled" dataset is also saved for further reproduction of the results reported by the solution. A reproduction might be done by simply fine-tuning a language model with the help of this dataset, given that other parameters of training are left unchanged.

After all the steps above, a new AL iteration starts. The whole process lasts until the last AL iteration is finished. After that, all metrics scores collected during AL experiment are returned and stored for further visualisation and monitoring. However, in a real-life scenario a stopping criteria might also be some evaluation metric threshold that has to be surpassed, not the number of AL iterations, which guarantee that the algorithm will terminate.

Real-life AL training

The real life part of the solution lies in correct configuration of all parameters present in a configuration file. As soon as the target AL strategy is known and the training scenario is determined, the real-life AL scenario can start. It consists of the very same steps as during experimentation part, but after the new chunk of unlabelled data is chosen, it has to be manually labelled.

The chunk is saved as a CSV file, which has to be then opened either in an annotation tool like Prodigy [MH] or in Excel for tagging.

The process diagram of the library can be found in appendix: D.2.

3.4 MLOps integration

There are lots of experiments involved into active learning research. In order to be able to tell which AL strategy suits best for the particular task, a thorough process of experimentation, parameters search and comparison has to be conducted. One of the main topic in massive models training is how to track all the experiments and how to be able to systematically trace and compare particular runs. Thankfully, there are several possibilities how to keep experimental data in a clean and sustainable way. The solution that I found promising is the library called Weights & Biases (W&B) [Bie20].

Weights & Biases is a machine learning platform acting like Git for software development. It has broad functionality supporting machine learning engineers and researchers by automation of experiment tracking, model's weights management, datasets versioning and reports preparation. Its aim is to operationalise the lifecycle of ML models, input parameters and weights, trained models, datasets and metrics. It's an API with a web-based UI prepared for tracking experiments and it's free even for small research groups. It enables collaborative work and experiments recreation with the help of artifacts.

Weights & Biases are easily integrated into existing project. I made the following steps in order to fully integrate my project with Weights & Biases ecosystem:

1. I have registered on wandb.ai and received a personal access token (PAT), which is currently considered a standard in cloud-based SaaS solutions.
2. I used this PAT as a value of an environment variable for the conda environment I used for this project. By only setting up this environment variable the project is capable of communicating with Weights & Biases servers and can already log data.
3. For each AL experiment I create a separate "run" in W&B ecosystem and pass the following information:

a. The name of the run - a string with a timestamp of the moment when the experiment is run with base information: device name, task, model name.
b. The whole content of a configuration file converted to JSON format, so that it is accessible from W&B run UI.
c. The whole content of a dataset configuration file.
d. A model instance for tracking gradients and model's topology.
e. Datasets' content grouped by splits as run's artifacts. This is done for ease of control from the UI what kind of data the solution works with. The artifact is saved as a CSV file.

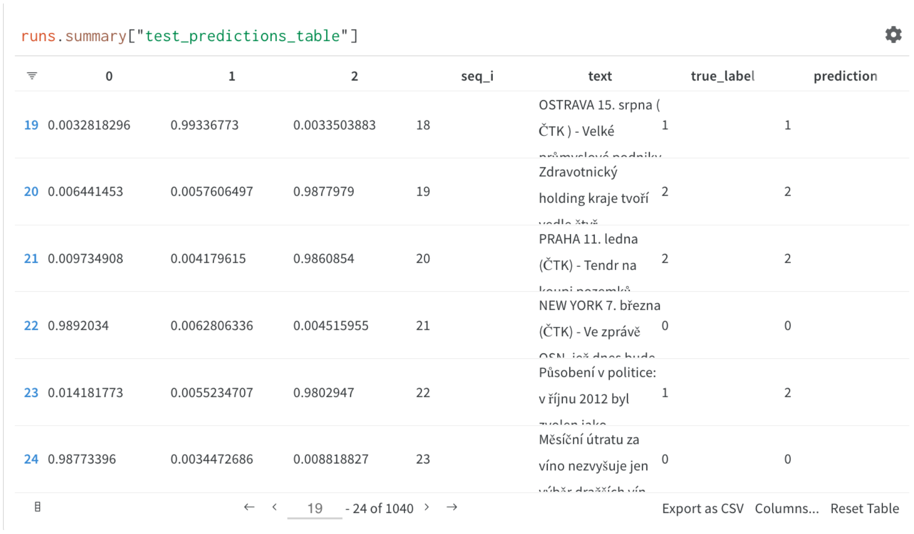
For each experiment I also create a table for tracking the model's performance on a test set with the following columns:

a. Index - the unique identifier of the input text
b. Text - unprocessed raw text which is put into the model.
c. True label - a correct label for the respective text given by an annotator. In case of tagging task there will be a serialised list of tags.
d. Prediction - a predicted label by the model. In case of tagging task there will be a serialised list of predicted tags.

The table is then enlarged by additional columns representing each category from the training dataset. The content of these columns will be a probability distribution (softmax scores), denoting the likelihood of the corresponding category. The model's decision is by default set to the category with maximum value of softmax score. An example of such table in Weights & Biases environment is present in figure 3.5.

During AL experiments, in each iteration the current training dataset is saved. After the selection of the new batch of data move from unlabelled collection to the training set, this batch is also saved, so that it is possible for a researcher to track increments in each step. For each of these datasets - a current one and an increment - there is a separate artifact in Weights & Biases for easier navigation in the UI. An example of different artifacts collected during one AL experiment is shown in figure 3.6.

During training I store all training-related data in logging dictionary that is passed to Weights & Biases logging API. For the full training (if enabled) the value of the mean training loss is stored in AL run configuration file (since it is done only once, there is no need for continuous logging of this value). After all training epochs are finished, the mean loss for each AL iteration is stored in a logging dictionary. The same rule is applied for validation and testing phase. Firstly, a table with both predictions (and their respective softmax scores) and true labels for test dataset is created. This table is then sent to Weights & Biases

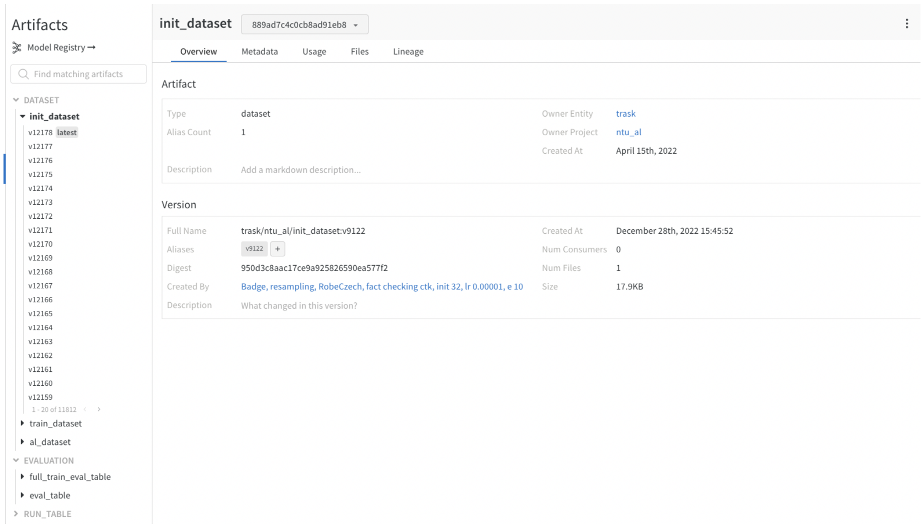


runs.summary["test_predictions_table"]

	0	1	2	seq_i	text	true_label	prediction
19	0.0032818296	0.99336773	0.0033503883	18	OSTRAVA 15. srpna (ČTK) - Velké	1	1
20	0.006441453	0.0057606497	0.9877979	19	Zdravotnický holding kraje tvoří	2	2
21	0.009734908	0.004179615	0.9860854	20	PRAHA 11. ledna (ČTK) - Tendr na	2	2
22	0.9892034	0.0062806336	0.004515955	21	NEW YORK 7. března (ČTK) - Ve zprávě	0	0
23	0.014181773	0.0055234707	0.9802947	22	Působení v politice: v říjnu 2012 byl	1	2
24	0.98773396	0.0034472686	0.008818827	23	Měsíční útratu za víno nezvyšuje jen	0	0

19 - 24 of 1040 Export as CSV Columns... Reset Table

Figure 3.5: Illustration of a prediction table for a test set calculated after each active learning iteration. The table contains columns with the same names as categories' names, then the index of the example (seq_i column)), then input text, correct category and predicted category (in my experiments - a name of the label column with maximum likelihood score).



Artifacts
Model Registry →
Find matching artifacts

DATASET
init_dataset
v12178 latest
v12177
v12176
v12175
v12174
v12173
v12172
v12171
v12170
v12169
v12168
v12167
v12166
v12165
v12164
v12163
v12162
v12161
v12160
v12159
1 - 20 of 11812
train_dataset
al_dataset
EVALUATION
full_train_eval_table
eval_table
RUN_TABLE

init_dataset 889ad7c4c0cb8ad91eb8

Overview Metadata Usage Files Lineage

Artifact

Type	dataset	Owner Entity	trask
Alias Count	1	Owner Project	ntu_al
		Created At	April 15th, 2022
Description	Add a markdown description...		

Version

Full Name	trask/ntu_al/init_dataset:v9122	Created At	December 28th, 2022 15:45:52
Aliases	v9122 +	Num Consumers	0
Digest	950d3c8aac17ce9a925826590ea577f2	Num Files	1
Created By	Badge, resampling, RobeCzech, fact checking ctk, init 32, lr 0.00001, e 10	Size	17.9KB
Description	What changed in this version?		

Figure 3.6: Illustration of artifacts page with all versioned datasets that are collected during one AL experiment.

so that user can either study the results for each test dataset row in a web-based UI or download this table as a CSV table and study it locally. Additionally, a confusion matrix is calculated and sent to W&B, which natively supports confusion matrices and visualise several different experiments with their statistics and confusion matrices very clearly, see figure 3.7. I use this visualisation very often for deciding which model is superior. All evaluation metrics defined by a user are also saved and sent to the UI for comparison with other experiments.

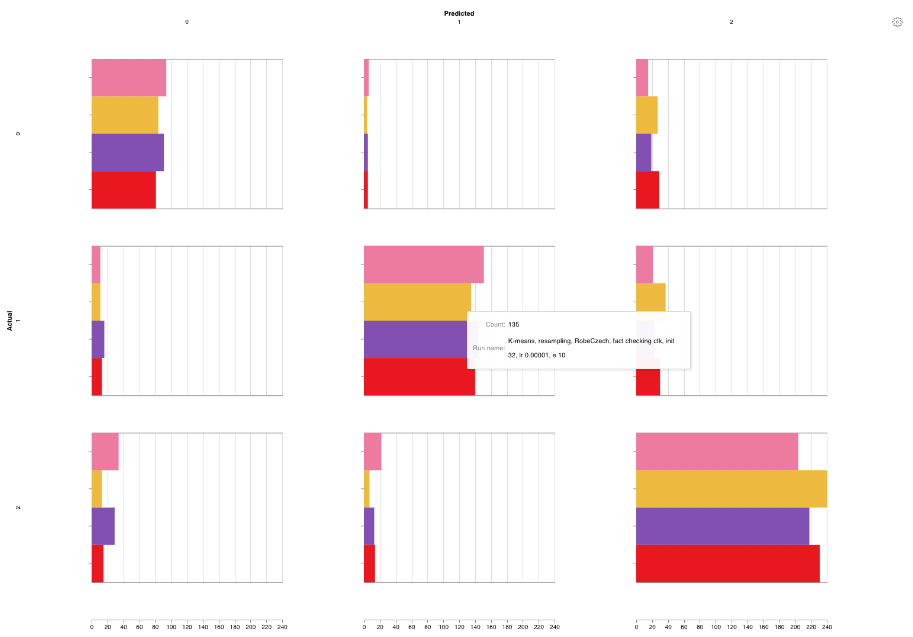


Figure 3.7: Illustration of a confusion matrix representing classification statistics for four different experiments (purple, yellow, violet, red bars). Weights & Biases have nicely implemented visualisation for comparison of different runs. It can be seen from this figure that purple experiment had more mistakes predicting label 0 and 1, while a yellow experiment shows superiority in correctly predicting label 2.

Weights & Biases is a user-friendly web-based interface for machine learning experiments tracking. It is divided into several parts, see figure 3.8. The first part is a list of all experiments (or runs) conducted by a user. A user can click on each run and see all the information that is tracked about particular experiment. Tracking is done either by manually adding some variables to logging dictionary as I stated above, or natively by the library which internally monitors the state of the hardware the experiment is run at, the state of GPUs if present, the whole Python environment and the way the experiment was triggered (all command line arguments).

The second part of the interface is a grid with charts. Each chart is drawn basing on variables logged during each experiment step. These variables can be technically everything: time taken for execution of one step, validation metric value, loss value, etc. Traditionally, training a neural network is done in so-called epochs. One epoch consists of several steps, which many times cover the whole training dataset. In my solution, an epoch is defined in the same way: the number of steps per epochs is calculated basing on the training batch size with the last batch being a residual batch (can be of a smaller size than others). Weights & Biases track variables in so-called steps. The responsibility for the definition of a step lies on the designer of the experiment: a step can be an epoch, a real step in training - one batch of data, or even the

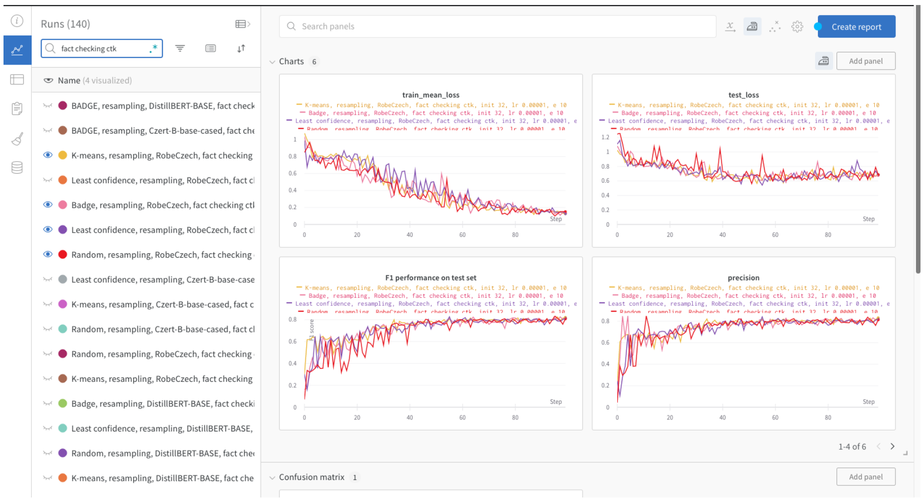

Runs (140)
fact checking ctk
Name (4 visualized)
BADGE, resampling, DistillBERT-BASE, fact check
BADGE, resampling, Czert-B-base-cased, fact che
K-means, resampling, RobeCzech, fact checking
Least confidence, resampling, RobeCzech, fact cl
Badge, resampling, RobeCzech, fact checking ctl
Least confidence, resampling, RobeCzech, fact cl
Random, resampling, RobeCzech, fact checking
Least confidence, resampling, Czert-B-base-case
K-means, resampling, Czert-B-base-cased, fact c
Random, resampling, Czert-B-base-cased, fact cl
Random, resampling, RobeCzech, fact checking
K-means, resampling, RobeCzech, fact checking
Badge, resampling, DistillBERT-BASE, fact checki
Least confidence, resampling, DistillBERT-BASE,
Random, resampling, DistillBERT-BASE, fact che
K-means, resampling, DistillBERT-BASE, fact che
Search panels
Create report
Charts 6
Add panel
train_mean_loss
test_loss
F1 performance on test set
precision
Step
1-4 of 6
Confusion matrix 1
Add panel

Figure 3.8: Illustration of Weights & Biases experiment UI. There is a list of experiments on the side bar and charts with tracked variables in the main window.

whole experiment. Thanks to such flexibility I was able to integrate Weights & Biases charts into my AL library in the following way:

1. One tracked step is one iteration of AL strategy. The monitored variable is a metric score or loss value for training/testing part of an AL iteration. At the moment the library tracks mean training loss, test loss, test accuracy, test precision, test recall, test F1 score.

2. Tracking of the value technically means to create a Python dictionary, where keys resemble the name of the variable we track and its value is the value we want to send in this step to the cloud. The dictionary is gradually filled with values and then is submitted to the server once all values are present.

3. In charts drawn in the UI a horizontal axis denotes each AL iteration of the corresponding experiment. Hence, it is possible to observe how the metrics values are evolving throughout dataset filling. Losses are traditionally expected to decrease and F1, precision, recall, accuracy are supposed to increase. However, since we are dealing with small datasets in the beginning, oscillations might be present due to highly unstable training, which has to be studied separately.

   Weights & Biases cope with oscillation in charts by smoothing them (see C.1). It is a useful feature for comparing different AL strategies - by smoothing we observe trend line, which is more significant than single values in each step.

One more feature worth mentioning is out-of-the-box plotting of confusion matrix not only for one experiment, but also for multiple experiments. Confusion matrix is calculated natively by this library, a researcher only has to provide predicted and true labels for it. All is done by the library. One pitfall is that the confusion matrix in my case is plotted for the last step of AL strategy, since originally Weights & Biases dashboard is designed to track one full neural network training, where confusion matrix is mostly calculated only once on a test set after the training is finished. However, I calculate it after each AL step and store them as artifacts in cloud.

Weights & Biases enable to store tables and show them in a readable format. A table is technically a CSV file with header, which is then parsed in the UI. I use this feature for the following purposes:

1. For storing the source table of confusion matrix calculation - for each pair of true and predicted labels I see the frequency of such situation. It helps me to find the problematic pairs of possibly overlapping categories in input data.
2. For storing softmax scores distribution among possible categories for the whole test dataset together with true and predicted label. This table helps me to see how the model behaves on the test set specifically for particular inputs, not in a summarised way as with test metrics, and to study the scores which show uncertainties about each category: if a falsely predicted category has softmax score close to 1, it might signalize some systematic error in labelling or training. Thus, the data entry might be needed to study deeper.

There are many other features that I did not use in my project, although they are highly valuable: Weights & Biases are capable of creating automated reports for each training and even for a set of experiments, so that a researcher can automate comparison runs and AL strategies selection.

3.5 RCI cluster integration

A part of my thesis was the integration with Research Center for Informatics (RCI) cluster, which is the centre of computer science and artificial intelligence research at Czech Technical University. This integration was necessary for getting access to HPC (High Performance Computing) infrastructure, consisting on CPU, GPU amd SMP nodes - for my active learning experiments. RCI cluster is equipped with nodes with NVIDIA Tesla V100 graphic cards with 32GB graphing memory, which is enough for fine-tuning of large NLP language models (around 100-400M parameters) without freezing layers.

In order to run experiments at RCI cluster, one needs to request for the access to RCI servers. An access is granted for RCI or AIC researchers and for those using GPUs for their individual projects at CTU. I have been granted access and was able to use the whole infrastructure for the project.

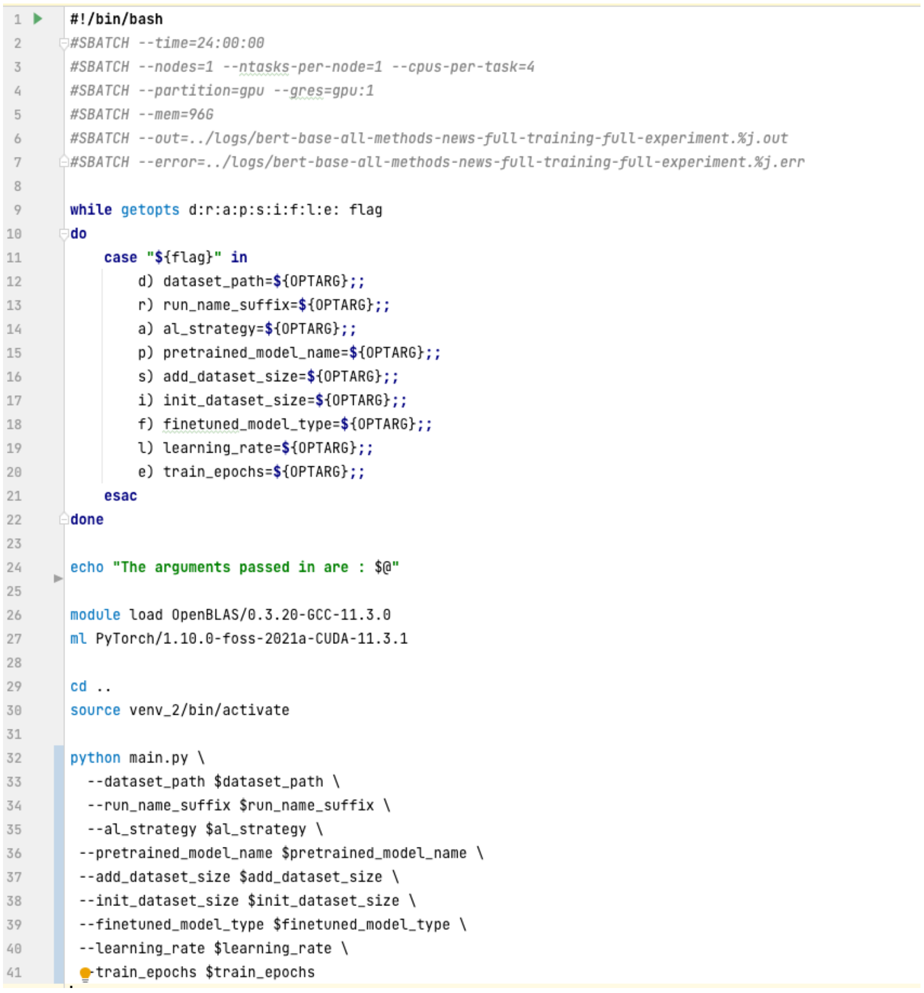
A cluster is a remote server which I connected via UNIX ssh CLI program. Each user has its own private folder where he can run his code. First of all, I had to set up Python environment with all modules required by the solution - *transformers*, *wandb*, *seaborn*, etc. I managed to do it with the help of RCI cluster tutorial for running jobs with *sbatch* utility.

Then, I used a popular Python environments manager - *virtualenv*, which helped me to create an isolated environment (just like conda) for this project. I created the modules list with their corresponding versions (which is vital in Python world) in a *requirements.txt* file which I put to the project directory. The whole source code is hosted on my GitHub repository, and the process of integration and experimentation later was implemented via a dedicated Git branch where I developed all necessary steps for correct RCI cluster usage.

Finally, I stuck to the following workflow:

1. I develop locally on my laptop in PyCharm IDE.
2. I commit and push all changes to RCI branch on my remote repository on GitHub.
3. I log in my RCI account via ssh.
4. I pull changes from this branch to the local file storage on the cluster. This approach frees me from mapping cluster's storage on my own laptop, which I found more elegant.

5. I run a prepared bash script which specifies everything needed for the run, see 3.9.
6. The whole experiment is logged to Weights & Biases web UI, so that I do not need to do anything else on a cluster.
7. I monitor workload, elapsed time of my jobs and availability of nodes via commands:
   a. *squeue | grep <username>* - to monitor all jobs triggered by me.
   b. *scancel <job id>* - to cancel a mistakenly run job.
   c. *showpartitions* - to check if some GPUs are available.



```
1  #!/bin/bash
2  #SBATCH --time=24:00:00
3  #SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=4
4  #SBATCH --partition=gpu --gres=gpu:1
5  #SBATCH --mem=96G
6  #SBATCH --out=../logs/bert-base-all-methods-news-full-training-full-experiment.%j.out
7  #SBATCH --error=../logs/bert-base-all-methods-news-full-training-full-experiment.%j.err
8
9  while getopts d:r:a:p:s:i:f:l:e: flag
10 do
11     case "${flag}" in
12         d) dataset_path=${OPTARG};;
13         r) run_name_suffix=${OPTARG};;
14         a) al_strategy=${OPTARG};;
15         p) pretrained_model_name=${OPTARG};;
16         s) add_dataset_size=${OPTARG};;
17         i) init_dataset_size=${OPTARG};;
18         f) finetuned_model_type=${OPTARG};;
19         l) learning_rate=${OPTARG};;
20         e) train_epochs=${OPTARG};;
21     esac
22 done
23
24 echo "The arguments passed in are : $@"
25
26 module load OpenBLAS/0.3.20-GCC-11.3.0
27 ml PyTorch/1.10.0-foss-2021a-CUDA-11.3.1
28
29 cd ..
30 source venv_2/bin/activate
31
32 python main.py \
33   --dataset_path $dataset_path \
34   --run_name_suffix $run_name_suffix \
35   --al_strategy $al_strategy \
36  --pretrained_model_name $pretrained_model_name \
37  --add_dataset_size $add_dataset_size \
38  --init_dataset_size $init_dataset_size \
39  --finetuned_model_type $finetuned_model_type \
40  --learning_rate $learning_rate \
41  -train_epochs $train_epochs
```

Figure 3.9: RCI cluster run script. Lines 2-7 define variables used by job executor. Lines 9-22 parse input command line arguments. Lines 26-30 initialise active learning environment. Lines 32-41 trigger run on a GPU node with all parameters.

3.6 Google Colab integration

The first part of my experiments with small models was executed before I was given access to RCI cluster, so I wrote several Jupyter Notebooks that I ran in Google Colab environment. It is an online version of Jupyter Notebooks, which are widely used for experimenting in data science field. Google Colab is more advantageous than Jupyter notebooks as it is run on Google's virtual machines dedicated for Python experimenting and the notebooks are accessible via Google Drive. It means that a developer is freed from a painful process of packages installation and incompatibility issues. Almost everything is preinstalled and since it is a product from Google, there is no need to additionally install any other Google's frameworks. Nevertheless, the most important feature Google Colab comes with is GPUs and TPUs availability free of charge. Everyone can simply connect to a cloud accelerator and run experiments on them. This is the main reason why I chose Google Colab as my primal training and analysis environment. TPUs and GPUs are however preemptive, meaning that they are reserved for a short period of time (12 hours) and are primarily assigned to those who utilize them less. The reason for such a policy is to make accelerators available to everybody so that nobody can monopolise them.

The main notebooks, which is literally a *.ipynb* version of the *main.py* script, is *AL_Experiment.ipynb*. This notebook is also integrated with Weights & Biases platform. Although the integration process was not seamless and by the time of development this framework had several synchronisation issues, I managed to find solutions to them, so that no experiment is missing. In the end of the notebook there is also a local visualisation example, which is a lightweight version of charts powered by Weights & Biases and which are designed for a fast comparison of several AL strategies. An example of such visualisation is shown in figure 3.10. The visualisation routine (which is a separate class which collects all data and then produces the grid with curves) is implemented in a native *matplotlib* environment from scratch - to demonstrate how such chart can be implemented. As a result, this visualisation was never used due to lack of flexibility and the need for programming each change in plots compared to straightforward toolset of Weights & Biases. Such visualisation may be useful when there is no possibility to connect to the W&B cloud and to synchronise metrics with it.

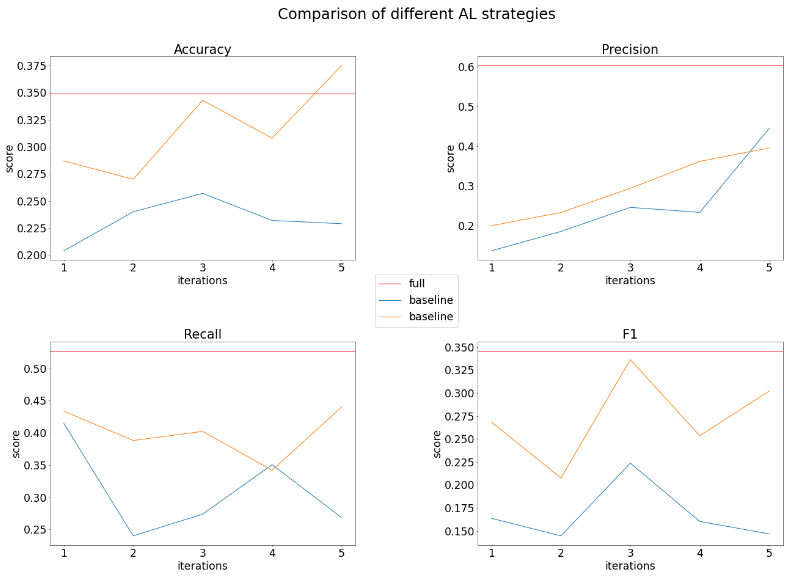


Figure 3.10: A chart used for comparing active learning strategies with the full training result (a horizontal red line labelled "full" in the legend) and the results of several AL strategies. The library is capable of running as many experiments with different strategies as needed, hence two "baseline" curves in plots. Each visualisation is divided into four independent plots: for accuracy score, precision, recall and F1 (weighted) score.

Chapter 4

Experimental part

4.1 Active learning experiments

In this chapter I present the results of the experimental part of my thesis, which I completed with the help of the library described thoroughly in chapter 3. The aim of these experiments was:

1. To evaluate feasibility of active learning approach on NLP tasks which are now solved with the help of fine-tuning of large language models. The evaluation had to answer the question whether it is worth plugging in active learning in such scenario.
2. To compare state-of-the-art active learning strategies discussed in 2.3.1, to observe their behaviour, explore their capabilities and limitations, their peculiarities while using them with NLP models.
3. To experiment with different configurations: different models, different strategies, hyperparameters setups, etc.
4. To debug the library and make its functionality worth publishing, so that other researchers of NLP community would be able to work with this library on their own in their own experiments.

The topic of active learning in NLP is extremely broad and thus there are lots of experimental setups that are possible to design. Here I present a possible classification of the most important parts in each AL experiment and then cover the experiments I conducted.

4.1.1 Tasks

Overall, there are two main tasks that are implemented in my project - classification and tagging.

Classification task is the most studied task in active learning literature. It is generally the most widespread objective of neural networks and is a generalisation of many other tasks. State-of-the-art active learning strategies are mostly implemented and studied in the context of classification, thus the solution would be incomplete in case it lacked this task.

However, I have also integrated Natural language inference (NLI) task like fact checking via translation of two textual inputs and their corresponding label into a classical classification task. Fact checking task is then implemented as a classification task on a textual input consisting of two concatenated strings divided by a technical [SEP] separator, which is known by a tokenizer.

The second task - tagging - is implemented in order to demonstrate the capability of the library to be simply extended by the next application. Tagging task is not extensively covered by active learning NLP literature and thus I hope that this implementation may spur interest in this topic as well.

4.1.2 Datasets

I have taken four datasets for my experiments, two for classification tasks, one for tagging, and one for fact checking:

1. **Newsgroup dataset** - a collection of approximately 20000 documents (13104 in a train set) divided into 7 different categories. Authors claim that the dataset is nearly uniformly divided. It is a popular dataset for text classification tasks, where there are considerably more categories than in most datasets. This is exactly the reason why I chose this dataset for my experiments. The dataset has a hierarchical division firstly into so-called newsgroups (which are 7), then into different corresponding topics.

For example, a newsgroup *comp* contains topics "graphics", "os", "sys", "windows", etc. For the purposes of testing AL approach I have chosen to use newsgroups as target categories for the model. I downloaded this dataset from Kaggle competition page and read it locally from the corresponding configuration file, where I provide paths to all three splits and target columns to process.

2. **Twitter US Airline Sentiment dataset** - a domain-specific dataset of US airlines data with reactions towards major American airlines. This dataset is designed for sentiment analysis. The input for our deep learning model would thus be a tweet about an airline and the output is the predicted sentiment (positive, neutral, or negative). There are 14640 tweets mentioning airline companies. It is a highly imbalanced dataset (63% of commentaries are obviously negative, 21% are neural and only 16% of commentaries are positive). The reason why I chose this dataset is to look at how AL strategies feel themselves with imbalanced data and also to equip my solution with a handler of class imbalance.

3. **CoNLL2003 dataset**, [TKSDM03] - a dataset with shared natural language tagging tasks: Part-Of-Speech (POS) tagging, chunking and Named Entity Recognition (NER) task. Among these three tasks the most complicated one is the latter, that's why I chose it. The dataset consists of 20744 entries and is the most popular benchmark of measuring quality of neural models solving NER task. This dataset is used in my experiments for demonstrational purposes, so that it can be seen that AL is also applicable to tagging. This dataset is hosted on HF Datasets hub.

4. **CTKFactsNLI dataset**, [UDR$^+$22] - a dataset with data in Czech language with claims and evidence. The dataset is acquired by AIC researchers in cooperation with other faculties and to the best of my knowledge is the largest and the most carefully collected dataset for fact checking task. The choice of this dataset is to run experiments with solely Czech language models and to reassure that the library is capable of handling Czech language as well as English. This dataset is hosted on HF Datasets hub.

4.1.3 Models

The selection of language models is given by historical development of this project: originally I did not have access to RCI cluster and thus I had to train smaller models. Lately I was granted access to more powerful GPUs and had opportunity to run experiments on larger architectures. By taking this into consideration, I was experimenting with the following models:

1. For English tasks - the choice was made in favor of well-documented and widely used large language models from BERT family: BERT-tiny [TCLT19], BERT-small [TCLT19], BERT-base [DCLT18a], BERT-large [DCLT18a]. All these models were taken in an uncased version. Then, I also experimented with ROBERTA model [LOG+19], which is the largest model among BERT-like models. All these models are stored at

2. For Czech tasks - I chose the only models publicly available at HuggingFace transformers models hub - RobeCzech [SNSS21], Czert [SPP+21], Small-e-Czech [SPP+21]. These models have comparable performance and create a small ecosystem of purely Czech-data trained models. For comparison, I also include DistilBERT-base-multilingual uncased model [SDCW19], so that the effect of pretraining on isolated Czech data via correctly instantiated dataset (without stripping accents) is clearer.

All the models are initialised with the help of AutoModel interface, thus the solution relies heavily on a configuration file of a corresponding model. Fine-tuning conditions and parameters are then equal. More on model initialisation can be found in on the official guide by HuggingFace.

4.1.4 Implemented strategies

I have implemented several strategies that I then used for my experiments with different datasets. From one-by-one strategies presented in 2.3.1 I have chosen least confidence strategy (finding an unlabelled example whose maximum probability score among possible labels is the lowest) and entropy strategy (choosing an unlabelled examples whose classes probability distribution has the highest entropy, i.e. the distribution is the closest to uniform). From batch methods I have selected two strategies: K-means strategy from 2.3.2 and BADGE strategy from 2.3.2 as two representatives of logically different approaches to candidates selection.

I have also included random strategy in the list of implemented strategies as a good baseline method to compare with.

4.1.5 Evaluation method

First of all, I analysed the performance of the trained model with the help of standard ML metrics: loss on the test set, accuracy, recall, precision and F1 score. Since I do specific research, I compared the performance based on two different schemes.

a. The first one compares the classical performance of the model trained on the fully labelled dataset with the performance of the models trained with the help of an AL strategy. The goal is to determine how much data is needed for an AL scenario for reaching the same performance as of the model trained on the full dataset.

b. The second scheme is targeted towards AL strategies comparison. For that we introduce a random strategy, which simply chooses the prescribed number of unseen data points without replacement and adds them to the training set. A random strategy emulates the stochastic process of data collection. It approximates the performance of the model trained on the partly full dataset in each step, while approaching the full dataset during the steps when the size of the AL dataset becomes equal to the full dataset.

   Then, each AL strategy is compared with the random strategy. I define a good strategy as one which surpasses the random strategy and achieves better results in the long run. A bad strategy is the one that performs worse in the long run. By saying "*in the long run*", I mean the trend line of the monitored metric score, which is achievable by smart smoothing of oscillating metric scores after each AL iteration. In terms of strategies comparison a better strategy will have its trend line higher than a strategy that performs worse. Hence, if a strategy achieves F1 metric score at test set faster and converges to some high F1 score, it is better than a strategy with lower F1 score in the result (in the end of the experiment) or with the same F1 score, but with slower rise of this metric score during the experiment.

AL steps can disrupt model performance - there is no guarantee that the model's performance reported at step $i + 1$ will be better or at least the same as at step i. There is a couple of reasons why it might happen. First, it could be a simple perturbation of the distribution of classes in the train dataset, when a class with larger number of examples will make model be biased towards this class. Then, there can be influence of stochastic nature of neural networks training, which could also disrupt model's behaviour. Since we cannot rely on precise numbers achieved during AL steps, a better evaluation method should be introduced. One possible way is to use smoothing techniques and to look at active learning experiments as a whole. The one that I found promising is **exponential moving average** (EMA) [1] technique which is actively used in trading and estimation of assets long-term price [GS13]. This technique favors those values closer to the actual point in time and discounts those far in the past, that's why it is a possible candidate for logging a test set metric. The values that are produced by EMA method are often characterised as smoothed.

Overall, a good strategy is either the one which achieves a better smoothed value in the end of the experiment or the one which is capable

[1]Explanation of trend line estimation using exponential moving average (EMA) could be found in https://www.investopedia.com/terms/e/ema.asp.

of achieving better score faster (if the speed at which the monitored metric gets to better values). Graphically it means that if there are two curves of smoothed values for two different active learning strategies, the one which is "higher" (when higher metric score is better, lower otherwise) is better. Graphic method is also important for assessment of strategy's behaviour - there are strategies that show faster development of monitored metric, while others grow slower but at some point they surpass the other. In this case, the decision of which one is better depends on the aim of the experiment - whether to achieve overall better performance or find a feasible (and possibly suboptimal) solution, but at a lower price. In case of the former, a strategy which achieves a better smoothed score in the end of the experiment is better, whereas in case of the latter it is the one with a faster growth.

4.1.6 Experimental details

The experimental part consists of three logical parts. The first one deals with smaller language models (BERT-tiny, BERT-small) in order to promptly see the impact of different strategies and to experiment more. The second one contains experiments with larger models (BERT-base, BERT-large). The third part is a part dedicated to Czech NLI task tested on CTKFactsNLI dataset, where the choice of the models is completely different. Here is the list of all experiments that I conducted in my project:

Part 1. Small language models

a. First of all, I compared all five AL strategies on the newsgroup dataset. The goal was to find out which strategy performs best on this particular classification task. To ensure that the result was not dataset-driven, this experiment was again performed on the Tweet US Airline dataset.

b. Secondly, I picked the best strategy from the first experiment for each task and ran it with even more steps to investigate how much data is actually needed to achieve the same F-Measure as the one achieved with training on the full dataset.

c. Then, I compared the performance of the two models, namely BERT-tiny and BERT-small, to understand the impact of the difference in model size. I studied effect of larger models in more detail in a separate section, where I trained larger models on RCI cluster. All experiments in this part were run on Google Colab with a randomly assigned GPU accelerator.

Furthermore, the same wass done for the tagging task on the CoNLL-2003 dataset, with the difference that until now only two strategies for tagging are implemented and compared: *least confidence* and *random*. *Least confidence* strategy is inspired by [CLM$^+$15], with the only difference that I did not include a CRF layer [HXY15] and I computed confidence in a Naive Bayes style simply by taking the sum of posterior log-probabilities of each token in the sequence, then I reweighted the score by the length of the sequence so that the strategy did not favor longer sequences. The final score was used to approximate the probability of such tags sequence. However, I have to admit that it is only an approximation of real probabilities, since tokens in sequences are not independent. For better calculation, Markov Chain Models have to be used). Additionally, I compared the influence of resampling on the tagging task. Resampling is done in order to facilitate training with imbalanced data. PyTorch implements *RandomWeightSampler* for doing oversampling and undersampling [MRA20].

Part 2. Large language models

Here I experimented with both classification and tagging tasks. For classification, four selected strategies (*least confidence*, *random*, *K-means*, *BADGE*) were selected for this part as a result of the previous part with smaller models. For tagging, only two strategies were compared.

Here is the plan for classification task I stuck to:

a. Firstly, I trained BERT-base model with all four strategies on Newsgroup dataset to assess active learning scenario on larger models.

b. Secondly, a similar experiment was conducted with BERT-large model.

c. Then, due to the instability of BERT-large models with a standard hyperparameters setup, a better set of parameters was found and was tested and compared.

d. Finally, all four models: BERT-tiny, BERT-small, BERT-base, BERT-large were compared on a selected AL strategy.

e. BERT-base and BERT-large models were trained on a complete training dataset and all four models were compared. Here I also compared AL scenario with full training - the impact of active learning on dataset size, metric scores.

f. In order to reassure myself that the results were not dataset-driven, I trained BERT-base model on Twitter dataset and compared all four AL strategies.

g. For completeness, I conducted a full training with BERT-base and BERT-large models on tweets dataset. Here I also compared impact of active learning in contrast to full training.

For the tagging part, the list of experiments was considerably smaller:

a. I trained BERT-base and BERT-large models for both *random* and *least confidence* strategies. For each model size I compared the impact of *least confidence* strategy in contrast to *random strategy*.

b. All model sizes were compared so that the best performing model was found and the F1 score development across models was analysed. Full training was conducted - again, the impact of active learning scenario was assessed compared to full training.

Part 3. Czech NLI

This task was translated to classification task, so I compared the same set of strategies for classification used in the previous part.

a. RobeCzech model with all four strategies was trained and compared.

b. Czert model with all four strategies was trained and compared.

c. DistillBERT model with all four strategies was trained and compared.

d. Small-e-Czech model with all four strategies was trained and compared.

e. RobeCzech model with better hyperparameters was again trained and analysed, comparison with the first experiment with RobeCzech was done, the impact of better hyperparameters set was assessed. Analysis of the results of this model and the new **state-of-the-art** result report.

f. More experiments with hyperparameters for RobeCzech.

g. All trained models in this part were compared with all four active learning strategies.

h. RobeCzech model, trained on a complete dataset, was compared to the best performing RobeCzech model, trained in AL setup.

4.1.7 Experiments parameters

As for fixed parameter settings, I set the number of fine-tuning epochs on 5 [DCLT18a], learning rate was 5e-5 [DCLT18a] and I used AdamW optimizer [LH17]. The train batch size was 32 while the validation and test batch size was 64. I set the seed AL dataset size to 32 and also added 32 entries in each AL iteration. In total I performed 100 AL iterations per run.

All the experiments were run with all parameters fixed and only experiments-specific parameters were modified:

a. AL strategy - see the main impact on model performance.

b. Class weight resampling - see the impact of better batching scheme for imbalanced datasets.

c. Model size - see the impact of different models capacities.

d. Datasets - reassure that the results we get are not data-driven.

The experiments from the first part were run in Google Colab with GPU acceleration. Several BERT-tiny models were trained locally on a Macbook Air 2020 M1. Training on GPU took around 60-90 minutes, whereas training on M1 processor took around 250-300 minutes. The experiments from the second and the third part were run on RCI cluster and varied in time needed for execution of experiments: for smaller models an experiment lasted for 30 to 60 minutes, while for larger models there were runs that utilised GPUs for more than 24 hours. The average time needed for one experiment ranges from 2 to 4 hours.

4.2 Part 1 - small models

I have run 18 experiments which tested different aspects of AL model training. Here I provide the results of my experiments.

Comparison of AL strategies

First of all, I compare all four strategies I implemented on a classification task (newsgroup dataset) to investigate which one performs better. I provide the maximum performance score for every AL strategy throughout all AL iterations in Table 4.1. It can be seen that *K-means* strategy mostly performs best. I also pick two winning strategies (*K-means* for batched sampling and *least confidence* for uncertainty sampling) and compare them on another dataset (Twitter) to reassure that my results are consistent across different data sources. The results are shown in Table 4.2 and once again prove that *K-means* strategy performs better.

Metric name	full	random	least confid.	K-means	BADGE
Test Loss	0.423	0.808	0.933	**0.801**	0.817
F1	0.836	0.797	0.796	**0.802**	0.798
Recall	0.85	0.799	0.793	**0.8**	0.795
Precision	0.845	0.806	0.807	**0.811**	**0.811**

Table 4.1: Results on the classification task (newsgroup dataset) with class weight resampling, achieved by experimenting with BERT-tiny model. Maximum performance metric score throughout all AL iterations is reported with the minimum loss function score.

metrics	least confidence	K-means
Test Loss	0.837	**0.71**
F1	0.812	**0.827**
Recall	0.815	**0.868**
Precision	**0.819**	0.799

Table 4.2: Results on the classification task (Twitter dataset) with no resampling, achieved by experimenting with BERT-tiny model. Maximum performance at any given AL step is reported with the minimum loss function score.

I have then also conducted experiments to compare AL strategies for tagging. As can be seen in figure 4.1, changing the resampling parameter did not have a great effect on the performance. However, what can be observed is that *least confidence* strategy performs much better than *random* strategy for tagging.

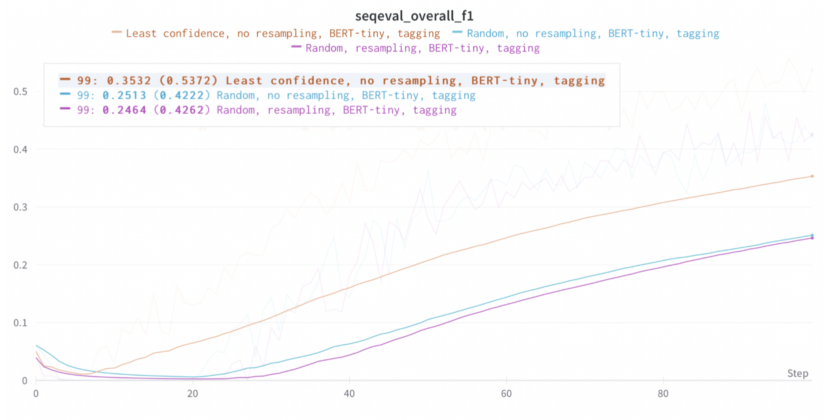


Figure 4.1: F1 of *least confidence* with (beige) and without resampling (orange) and *random* strategy with (purple) and without resampling (blue) of the tagging task (CoNLL-2003 dataset). Smoothed (dark brown) and non-smoothed (light brown). BERT-tiny model.

Best AL strategy vs. full training

The best performing strategy *K-means* from the first classification experiment (on newsgroup dataset) has been run again with more AL steps to explore when the F1 metric score of the fully trained model can be achieved (see Figure 4.2). The F1 of the fully trained model is **0.8362**. At step 167 the non-smoothed F1 score of the model trained with active learning strategy is **0.839** and therefore even higher than the F1 score of the model trained on the full dataset. As in each step, 32 data points were added, only 4384 training instances were needed to achieve the same performance when training with 13104 data entries. 33.4% of the training data can thus be reduced while still achieving good performance.

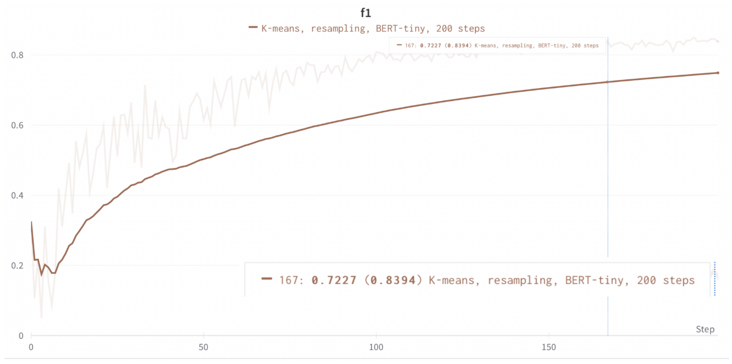


Figure 4.2: F1 of the best strategy *K-means* of the classification task (newsgroups) with more AL steps reported for BERT-tiny model. Smoothed (dark brown) and non-smoothed (light brown). Full training F1 score for this dataset is **0.836**. At step 167 the exact value of F1 test set score is 0.839. One step of the AL experiment adds 32 more labelled examples from the unlabelled set to the training dataset.

I have also compared the best performing active learning strategy score on Twitter dataset with full training on the same dataset. The results are depicted in table 4.3 and show that *K-means* strategy was successful in choosing more informative examples that led to higher overall performance compared to full training dataset, which could have incorrect labels or class imbalance.

metrics	**Full**	**K-means**
Test Loss	**0.516**	0.71
F1	0.788	**0.827**
Recall	0.797	**0.868**
Precision	0.79	**0.799**

Table 4.3: Results on the classification task (Twitter dataset) with no resampling. Evaluation result shown for test set for both full training and *K-means* strategy for comparison.

Impact of model size

Besides, I have also trained a larger model (BERT-small instead of BERT-tiny) and compared its performance with its smaller counterpart. Both experiments consist of 100 AL iterations, the selected strategy for classification on newsgroup dataset was *K-means* sampling. The progression of F1 test set metric of both models can be seen in figure 4.3. The results for both classification (with *K-means* strategy) and tagging task (with *least confidence* strategy) are shown in Table 4.4. It can be seen that BERT-small outperforms BERT-tiny, especially for the tagging task, which is by default considered a harder task.

	classification		**tagging**	
metric	BERT-tiny	BERT-small	BERT-tiny	BERT-small
Test loss	0.71	**0.511**	0.357	**0.124**
F1	0.816	**0.855**	0.558	**0.878**
Recall	**0.858**	0.856	0.566	**0.886**
Precision	0.792	**0.858**	0.566	**0.875**

Table 4.4: Results of models BERT-tiny and BERT-small on the classification task (newsgroup dataset) on the left side with *K-means scoring* strategy with no resampling. On the right side tagging task (CoNLL-2003 dataset) with *least confidence* strategy and no resampling are depicted. Maximum performance at any given AL step is reported.

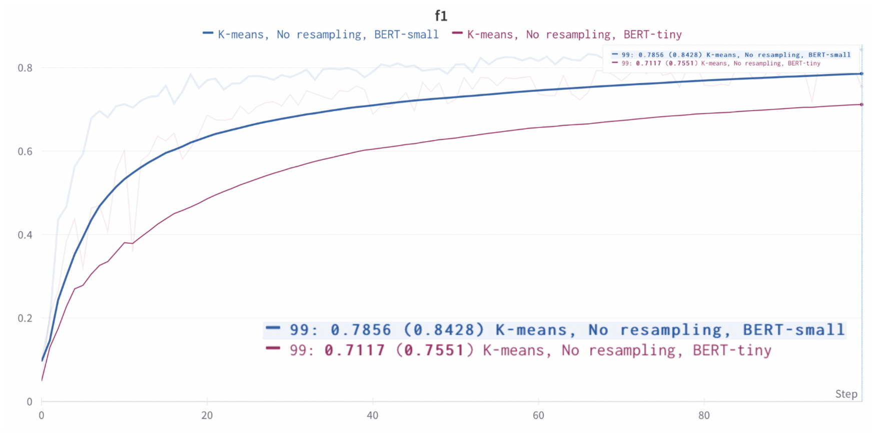


Figure 4.3: F1 of the best strategy *K-means* of the classification task (newsgroups), two models comparison: BERT-small and BERT-tiny. Smoothed (dark blue, dark violet) and non-smoothed (light blue, light violet) for both models (BERT-tiny, BERT-small respectively).

4.2.1 Analysis

I have compared several AL strategies in order to determine which one performs better. I had two classes of AL strategies: simple uncertainty strategies which were designed to pick only one data point, but are here implemented in a way that chooses a batch of the least uncertain data points; and batch sampling strategies designed for acquiring the most diverse batch of unlabelled data in each AL iteration. I also provide *random* strategy as a baseline. I have observed that uncertainty strategies perform worse on both classification tasks than random sampling, whereas batch sampling strategies surpass random sampling. This is evident from the smoothed metrics curves I observed in the Weights & Biases dashboard. My findings support the idea that the most important part of active learning is to provide as diverse data batch as possible. That might be why batch sampling methods behaved better, as expected.

Next, to recall, my initial hypothesis was the following: AL is capable of reaching the same performance as with the full training, but only by using a fraction of all training data. I have conducted several experiments and I have managed to achieve this goal by using only 54% of all training data on news classification dataset: my F1 metric even surpassed the fully trained model.

Moreover, I expected that AL works better with larger models, since they have more capacity. As deep learning tends to utilise more parameters and computational power, I argue that applying AL on larger models has more distinct impact. Hence, I found the justification for my claims in the experiment where I have compared different BERT sizes on *K-means* strategy where I saw that the larger model is capable of training faster. In the end, using AL with larger models makes more sense and brings more savings in terms of data needed to train the model.

Finally, I have also observed that larger models perform significantly better on more sophisticated tasks. We see that BERT-small was able to learn significantly faster than BERT-tiny and thus was able to achieve much better overall F1 score.

4.3 Part 2 - Active learning and large models (classification and tagging)

Following the results reported above, it is clear that active learning helps reduce dataset size and achieves comparable performance as models trained on the whole dataset. In this section I discuss the compound effect of active learning strategies and larger models.

Models studied before were of a small size in terms of number of trainable parameters. Here I study these models:

a. BERT-base uncased - a 12-layer model, embedding size 768, 12 heads, 110M parameters. This model was trained on lower-cased English text.

b. BERT-large uncased - a 24-layer, embedding size 1024, 12 heads, 340M parameters. This model was trained on lower-cased English text.

The objective of this study is to see the effect of active learning strategies when applied on larger models. We have already seen how active learning behaves with smaller models, here I compare this behaviour with models of larger magnitude. The experiments were conducted on both classification and tagging tasks for complete comparison.

4.3.1 Text classification

BERT-base

Text classification is again divided into two parts: newsgroup dataset and Twitter sentiment dataset. The results here present a comparative behaviour of all strategies. For the sake of completeness the result for *random* strategy is presented as well. In figure 4.4 there is the result of the experiment. It can be seen that the *random* strategy is again between batch-based strategies (*K-means* and *BADGE*) and *least confidence* strategy. This behaviour is similar to the one reported above. We also see that *random*, *K-means* and *BADGE* strategies give better results at earlier steps of AL experiments, whereas *least confidence* strategy rises more slowly, but at some point (precisely at 34th step) surpasses both *random* and *K-means* strategies and holds almost always above by a margin of 3% on average. *BADGE* strategy however is almost always above others.

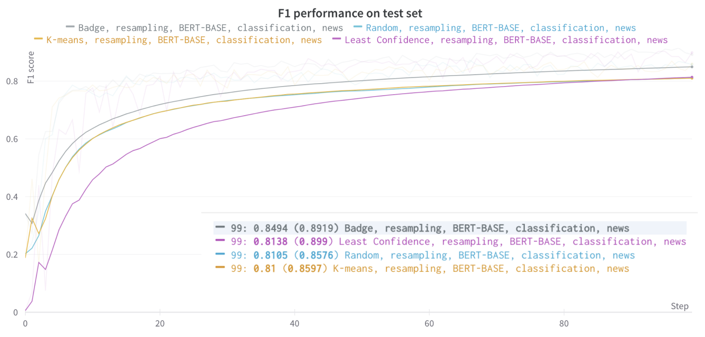


Figure 4.4: F1 development on a test set for newsgroup classification task for all four AL strategies, BERT-base model. Smoothed (dark-colored lines) and non-smoothed (light-colored lines). Smoothed line denotes a trend line, whereas non-smoothed line shows the exact values for each AL step.

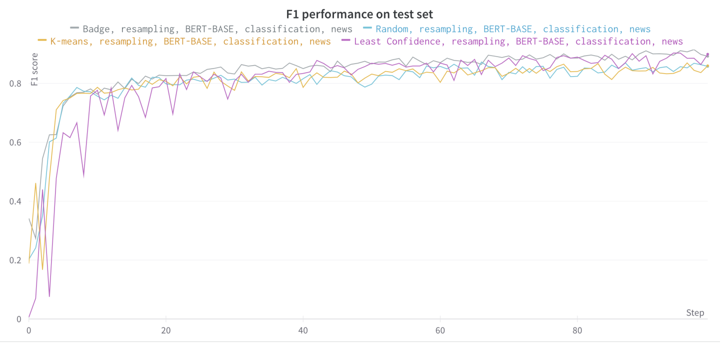


Figure 4.5: F1 development on a test set for newsgroup classification task for four strategies: *random*, *least confidence*, *K-means*, and *BADGE*, BERT-base model. Only non-smoothed curves are illustrated for better visualisation of the first part of training (first 20 steps) and how *least confidence* strategy acts similarly to *BADGE* strategy in the end.

Detailed statistics with best scores are given in table 4.5. *BADGE* method showed the best performance among all other strategies. The best score (0.915) was attained at 98th step - thus requiring 3136 training examples. It is however important to state that comparable F1 metric score (0.894) was registered by the *least confidence* strategy at step 73 (2336 examples).

Due to the fact that in each AL step the training dataset is enlarged by 32 labelled examples, it can be seen that we needed just a fraction (for the winning strategy - 24%, since the full dataset has 13104 examples) of data for reaching such F1 metric score. It is also important to tell that *least confidence* strategy behaves unstably during first 20 steps, whereas *random*, *K-means*, and *BADGE* strategies have more consistent development. This is one more sign of poor design of this strategy at first steps of AL training. However, when the dataset is already large enough (around 50th step or 1632 examples), this strategy acts almost equally like *BADGE*. These observations are clear from non-smoothed

metrics	Random	Least confidence	K-means	BADGE
Test Loss	0.503	**0.349**	0.459	0.353
F1	0.872	0.903	0.869	**0.915**
Recall	0.873	0.903	0.868	**0.914**
Precision	0.876	0.907	0.874	**0.918**

Table 4.5: Results on the newsgroup classification task for BERT-base-uncased model. Maximum performance among all AL steps is reported.

graph - see figure 4.5.

From the smoothed graph 4.4 it is also evident that *least confidence* strategy has the worst trend line than other strategies. It is however given by the first steps of active learning experiment, where the strategy underperformed heavily and was not stable. Nevertheless, non-smoothed graph shows the real situation, which is completely different. This is also the reason why I decided to put both charts and why it is important to assess both ways of interpretation of AL experiments. The fastest gain is provided by *BADGE* strategy, while *least confidence* strategy successes at finding unlabelled examples of higher quality in later steps.

BERT-large

Spectacular behaviour can be seen when BERT-large model is plugged instead of its base counterpart. Smoothed trend lines as in the case above are depicted in figure 4.6, however they do not show the reality that we face - the behaviour of all three experiments is not stable, it is chaotic and unexpected. Non-smoothed variant of the same graph is shown in figure 4.7. The experiments were not finished as usual (till step 100) due to lack of justification - such behaviour implies that something is malfunctioning - either strategies, hyperparameters or the dataset.

I have analysed this behaviour in deeper detail on many other experiments left out of scope of this written report (but left in Weights & Biases environment with all artifacts) and have drawn several conclusions:

a. There is no flaw in AL strategies implementation or their logic: *random* strategy shows similar behaviour as others. Hence, oscillations are not caused by active learning approach.

b. The dataset used for training cannot be blamed as well, since it is consistent and can be trained on by other models.

c. The last piece of the puzzle is the model - BERT-large is three times larger than BERT-base and thus needs to be trained with another set of hyperparameters. In this experiment (shown in figure 4.7)

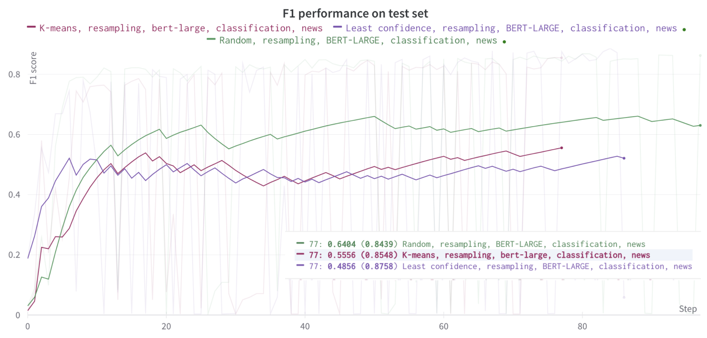


Figure 4.6: F1 development on a test set for newsgroup classification task for three strategies: *random*, *least confidence*, and *K-means*; BERT-large model. Smoothed (dark-colored) trend line and non-smoothed (light-colored) exact values line. In this figure, lines were smoothed by EMA technique with a coefficient of 0.5. Several experiments were not finished due to massive GPU overload by BERT-large model and unsatisfactory behaviour of metric scores.

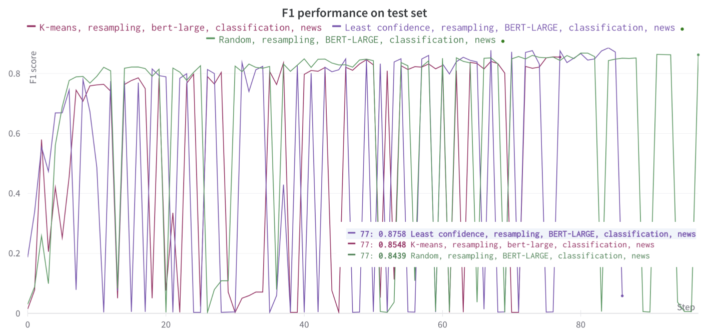


Figure 4.7: A F1 score development graph for the same experiment as in figure 4.6, but in only non-smoothed variant for depiction of oscillations of F1 metric scores.

the set of hyperparameters is the same as for BERT-base model. However, multiple sources ([DCLT18a], [SQXH19], [JHC+20]) claim that larger models have to be trained with lower learning rate and thus larger batch size.

The result of the analysis is a change in implementation compared to the one used for earlier experiments. The library was then enlarged by more flexible input arguments handling and override parameters given in the configuration file by those provided through command line arguments.

The result of the changed implementation are present in the next section.

BERT-large - improved hyperparameters

There are two most important hyperparameters that affect training of models of different parameters number in my case - learning rate and number of training epochs. Table 4.6 summarizes the final setup that I have tested in this project. We see that for a larger model the learning rate is lower and thus the number of training epochs is higher, so that the model receives more discounted response.

hyperparameter	**BERT-base**	**BERT-large**
Learning rate	5e-5	2e-5
Training epochs	5	10

Table 4.6: Key hyperparameters setups for stable fine-tuning of BERT-base and BERT-large models

The effect on training stability is shown in figure 4.8. Although there were still situations during training with better hyperparameters where we see a clear spike in both loss and metric score, we see significant improvement in experiment development.

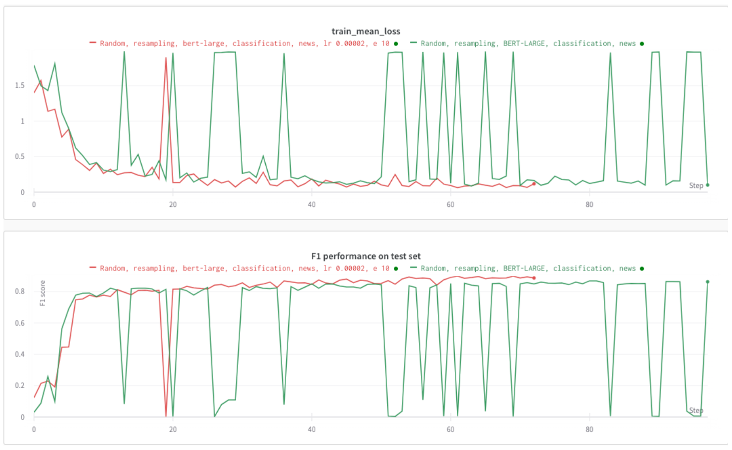


Figure 4.8: The effect of better hyperparameters choice for BERT-large model, Newsgroup dataset. Training loss and test F1 score reported. A green line denotes an experiment with hyperparameters equal to those used for BERT-base model. A red line shows an effect of the fixed setup of hyperparameters. The experiment with new hyperparameters was not finished due to high overload of GPU resource.

The results of the experiments with better hyperparameters setup are depicted in figure 4.9. We observe several spikes for *random* strategy and *least confidence* strategy, whereas *K-means* strategy behaves better. It is not clear which strategy performs better though: both *K-means* and *least confidence* strategies show steeper improvement of F1 score, but then *K-means* is almost always below *random* strategy by a margin on 3-4%, while *least confidence* shows almost the same behaviour as *random* one. My assumption is that there has to be another parameter better tuned, e.g. initial dataset size or number of elements to add after each AL step. Obviously, stability issue has to be resolved as well with more experimental setups.

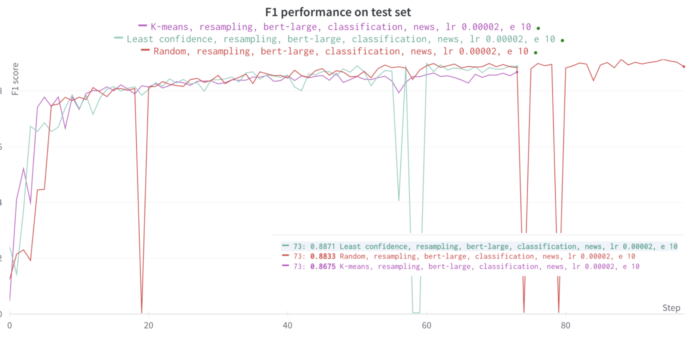


Figure 4.9: News classification. BERT-large - the effect of better hyperparameters choice. Test F1 score is reported.

All models comparison: tiny, small, base, large

Here I present the comparative analysis of all four BERT models in terms of models' sizes. I have chosen to visualise *BADGE* strategy for all four models. Figure 4.10 depicts smoothed trend lines of four models that show that the model's capacity plays an important role both in model's final performance and the speed of training. Surprisingly, BERT-base outperformed BERT-large model and converged faster, even though the final F1 score is almost the same. The result is even not given by a smoothing factor and a little instability of BERT-large model - a non-smoothed version of this graph (shown in figure 4.11) draws BERT-large result that oscillates around BERT-base curve, otherwise the models perform almost identically.

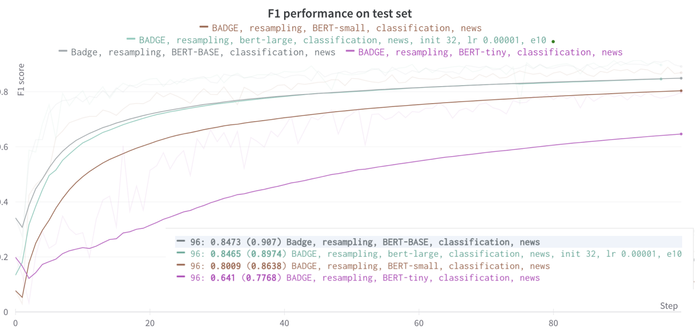


Figure 4.10: F1 development on a test set for newsgroup classification task for four models and *BADGE* strategy. Smoothed (dark-colored) trend lines and non-smoothed (light-colored) exact values lines. The experiment on BERT-large was not completed (stopped at step 97) due to massive GPU overload by BERT-large model and long running time for the very last steps.

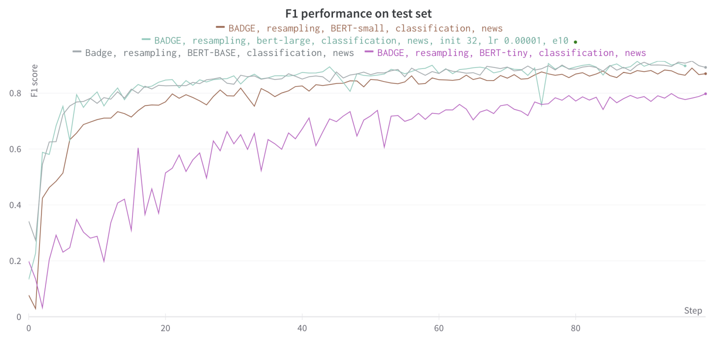


Figure 4.11: A F1 score development graph for the same experiment as in figure 4.10, but in only non-smoothed variant for depiction of oscillations of F1 metric scores.

Newsgroup classification - full training

I have also executed full training for all BERT models so that I see the difference between active learning best score and the traditional approach score. All results are shown in table 4.7. We see that BERT-base model showed the best performance, while BERT-large 2 (with optimized hyperparameters setup) has slightly better test set loss function value. Overall, BERT-base model is the best performing model, which is also true for active learning experiments, where BERT-base model showed superiority over other models. The active learning version of BERT-base model with the best F1 performance is reported for *BADGE* strategy is **0.915** and shows that this strategy was able to surpass full model, but at a small margin (0.005), which is insignificant. Another fact is significant - the number of examples needed by *BADGE* strategy to achieve this score - **3136** examples, which is just around 24% of full training dataset. It proves that it is enough to have only a quarter of the whole newsgroup dataset in order to achieve the same score that can be achieved by training on full dataset with higher hardware strain.

Model	**F1 test score**	**Precision**	**Recall**	Test loss
BERT-large 2	0.891	0.893	0.89	**0.312**
BERT-large 1	0.101	0.064	0.252	1.734
BERT-base	**0.91**	**0.915**	**0.91**	0.352
BERT-small	0.901	0.903	0.9	0.373
BERT-tiny	0.856	0.859	0.857	0.479

Table 4.7: Statistics for the full training dataset experiment for all BERT models studied in this section for newsgroup dataset. BERT-large 2 model denotes the version which has optimized hyperparameters setup, whereas BERT-large 1 is the model that has the same hyperparameters as others.

Tweets classification - BERT-base

In order to reassure myself that AL strategies' performance is not dataset-driven for larger models (for BERT-tiny and BERT-small this evaluation has already been done above), I have also experimented on BERT-base models and compared all strategies with each other and then the winning strategy with full training setup. F1 metric score chart for all strategies is depicted in figure 4.12. It can be clearly seen from the trend lines that *K-means* and *BADGE* strategies show superior behaviour and overperform both *random* and *least confidence* strategies. If we study exact values for this experiment (shown in figure 4.13), we see that all strategies showed similar performance and in the end all were close to each other in terms of F1 metric score. However, *BADGE* strategy kept staying consistently higher than others with top F1 score **0.840** attained at step 81 (2592 examples - around 25% of complete training dataset).

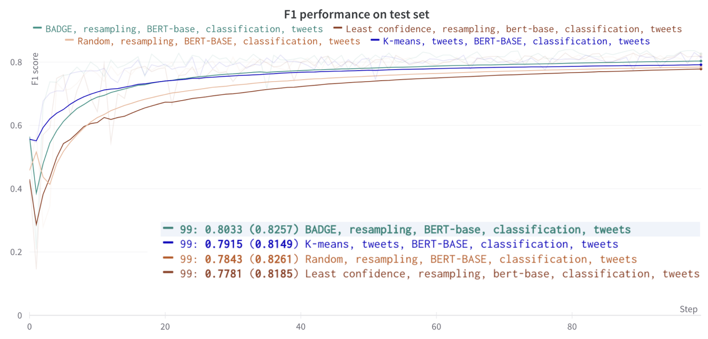


Figure 4.12: A F1 score metric chart for AL experiments on BERT-base model on tweets classification dataset. Dark lines show trend lines after EMA smoothing, light lines show exact values.

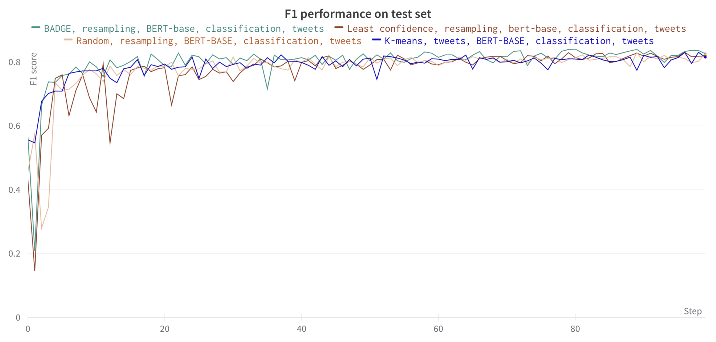


Figure 4.13: A F1 score metric chart for AL experiments on BERT-base model on tweets classification dataset. Exact values are reported.

Tweets classification - full training

Full training on tweets dataset was conducted for the sake of completeness of classification experimental part. The results which are depicted in table 4.8 show that the best performing model is BERT-large 2, which is the model with adjusted hyperparameters set. However, if we compare BERT-base model's result (F1 score 0.839) with the result from 4.3.1, it is almost the same (difference is less than 0.001 of F1 metric score). The result supports the idea of application of active learning for reaching the same result with only a fraction of data.

Model	F1 test score	Precision	Recall	Test loss
BERT-large 2	**0.891**	**0.893**	**0.89**	**0.312**
BERT-large 1	0.101	0.064	0.252	1.734
BERT-base	0.839	0.837	0.841	0.65
BERT-small	0.837	0.838	0.838	0.519
BERT-tiny	0.788	0.79	0.797	0.516

Table 4.8: Tweets classification task. Statistics for the full training dataset experiment for all BERT models studied in this section. BERT-large 2 model denotes the version which has optimized hyperparameters setup, whereas BERT-large 1 is the model that has the same hyperparameters as others.

4.3.2 Tagging

Tagging task is the easiest from the number of strategies perspective. Here I compare a *random* strategy against *adjusted least confidence* strategy. First of all, I present the comparative results of both strategies for BERT-base model. In figure 4.14 it can be seen that *least confidence* strategy is superior to *random sampling* in a trend line, while the exact values sometimes are equal.

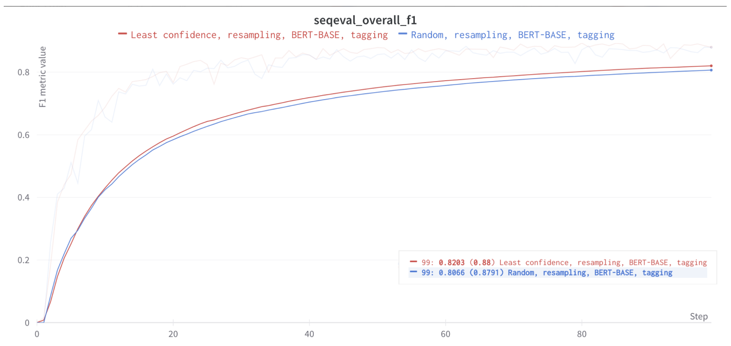


Figure 4.14: Seqeval F1 development on a test set for tagging task for two strategies: *random* and *least confidence*, BERT-base model. Dark lines show trend lines after EMA smoothing, light lines show exact values.

More scores reported in table 4.9. It is worth mentioning that the best F1 score result for *least confidence* strategy was reached at 81st step, thus with a training set consisting of 2592 samples. *Random* strategy reached peak F1 performance one step later - at 82nd step.

metrics	**Least confidence**	**Random**
Test Loss	0.128	**0.126**
F1	**0.893**	0.884
Recall	**0.905**	0.900
Precision	**0.885**	0.874

Table 4.9: Results on the tagging task (CoNLL2003) for BERT-base-uncased model. Maximum performance among all AL steps is reported.

Next, I present the results for BERT-large-uncased model, which is roughly three times larger than BERT-base model. In figure 4.15 we see a different picture than for BERT-base model: a *random* strategy performs better than a *least confidence* strategy, when speaking about trend lines. Figure 4.16 shows the exact development of a F1 score for the test set. It can be seen that *random* strategy behaves more stably compared to the *least confidence* strategy. It is also obvious that the trend line is highly affected by steps where F1 score falls to 0, otherwise *least confidence* strategy is still superior to random one by a margin of 2-3%. We also see that at the beginning the *random* strategy shows better behaviour with steeper F1 curve, however in the end *least confidence* surpasses it and almost always stays above the *random* strategy. The reason why it happened possibly lies in the design of the *least confidence* strategy - it is poorly adapted for diverse batches selection, hence longer time needed for F1 score to reach higher values. It works however extremely accurately on later steps, since when the dataset is considerably large, it picks those entries that are more uncertain for the model, and thus increases its performance, while preserving dataset variability.

Instability introduced by BERT-large compared to BERT-base may be caused by a hyperparameters setup: both models were trained with the same hyperparameters. However, larger models usually require smaller learning rate and thus more epochs for more stable training, which is also subject to further research.

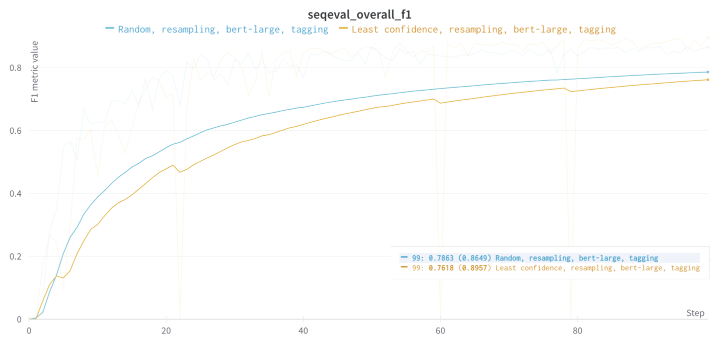


Figure 4.15: Seqeval F1 development on a test set for tagging task for two strategies: *random* and *least confidence*. Both experiments were conducted on BERT-large model. Dark lines show trend lines after EMA smoothing, light lines show exact values.

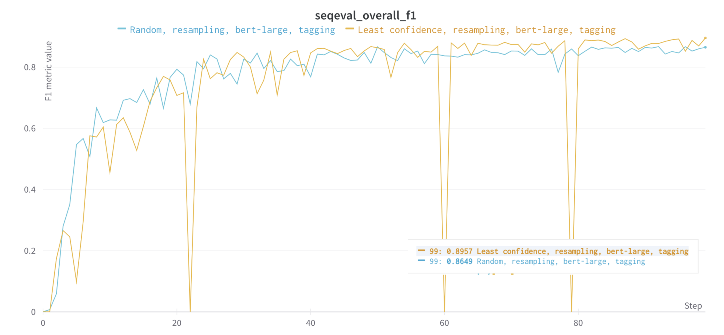


Figure 4.16: Seqeval F1 development on a test set for tagging task for two strategies: *random* and *least confidence*. Both experiments were conducted on BERT-large model. Non-smoothed variant. Numbers in legend show F1 score for the last AL step.

For completeness, in table 4.10 I present more precise results for BERT-large. We see that the model is slightly more accurate than BERT-base, however it has three times more parameters and the performance gain compared to the amount of computational power needed is negligible.

metrics	Least confidence	Random
Test Loss	**0.117**	0.136
F1	**0.896**	0.867
Recall	**0.910**	0.877
Precision	**0.888**	0.873

Table 4.10: Results on the tagging task (CoNLL2003) for BERT-large-uncased model. Maximum performance among all AL steps is reported.

All models comparison: tiny, small, base, large

I have also compared the behaviour of *least confidence* strategy for all models' sizes: tiny, small, base, and large. Smoothed and non-smoothed F1 scores can be seen in figure 4.17. Surprisingly, BERT-base model is showing better performance than BERT-large. Moreover, BERT-large model shows slightly better performance than BERT-small, while having 29.1M parameters compared to 340M (almost 12 times smaller model). However, it is clear from the plot that larger models learn faster and achieve higher F1 scores earlier than smaller models. BERT-tiny model underperforms all larger models, thus it is not suited well for this kind of task.

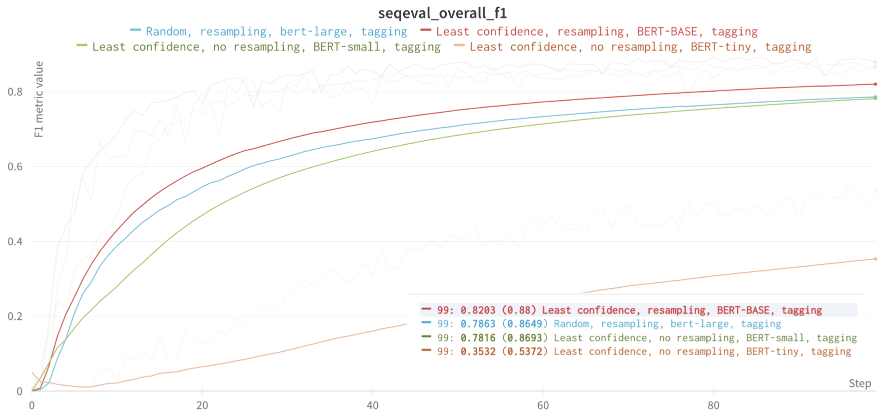


Figure 4.17: Seqeval F1 development on a test set for tagging task for all four BERT models. Each model except for the large one shows F1 score trend development, whereas the large one shows *random* strategy trend development due to inconsistency issue in *least confidence* strategy. Numbers in the legend show F1 score for the last AL step.

Full training - comparison

Full training results are presented in table 4.11. We see that BERT-large 2 model showed the best performance. BERT-tiny model is obviously the weakest among all and it is apparent that it does not have much capacity for such a hard task. It is also notable that BERT-small is just 2% worse than BERT-base model and then BERT-base model is 2% worse than BERT-large-2 model. It is clear that larger models are more capable and have larger capacity for memorization, that's why they have gained additional points. However, the best performing model is negligibly better than the model trained with *least confidence* active learning strategy, which achieved score of 0.896 at 100th step (thus requiring 3200 examples). As a result, only 23% of full training dataset was necessary to reach almost the same result as in the case of full training.

Model	F1 test score	Precision	Recall	Test loss
BERT-large 2	**0.90**	**0.895**	**0.90**	0.133
BERT-large 1	0.898	0.894	0.901	**0.117**
BERT-base	0.872	0.862	0.882	0.153
BERT-small	0.84	0.835	0.845	0.17
BERT-tiny	0.657	0.656	0.658	0.24

Table 4.11: CoNLL2003 tagging task. Statistics for the full training dataset experiment for all BERT models studied in this section. BERT-large 2 model denotes the version which has optimized hyperparameters setup, whereas BERT-large 1 is the model that has the same hyperparameters as others. All metric scores are depicted for corresponding seqeval *overall* metric.

4.4 Part 3 - Czech NLI

NLI task is a new one in this research. It poses higher challenge on neural models, since the task of language inference requires deeper understanding of the language. Presumably, larger models will yield better results. Here application of active learning might be advantageous due to complexity of the problem and higher costs of data annotation and preparation.

In this section I deal with CTKFactsNLI dataset, which consists of an evidence sentence, a claim sentence and a label denoting whether the claim is supported by the evidence, refuted or it is impossible to tell due to lack of information. A task of NLI consists of two textual inputs, which are usually concatenated into one string separated by a token known to the system as a separator: in our case it is a "[SEP]" token. Hence, any dataset for NLI can be transformed into a classification dataset with only one textual input consisting of two original strings. CTKFactsNLI dataset is thus transposed into a classification task with three categories. Then, active learning strategies used for classification task can be used here as well.

The task of Natural Language Inference presented by CTK fact checking dataset was then processed with these models:

a. RobeCzech [SNSS21] - a model from UFAL UK, a Czech version of RoBERTa model trained on a large collection of Czech textual data. It has 125M parameters in total.

b. DistillBERT Multilingual cased [SDCW19] - a distilled version of Multilingual BERT model. It has 6 layers, embedding size of 768, 12 heads, 66M parameters in total.

c. Small-e-Czech [KN21] - an Electra-based language model (small version) from Seznam.cz. This model has 13M parameters and is trained only on Czech textual data. It is the smallest model which I trained for fact checking task and which is partly capable of Czech language processing.

d. Czert [SPP+21] - an ALBERT-based [LCG+19] language model from ZCU university in Pilsen, Czech Republic. It has 110M parameters - the same as ALBERT base model.

A brief comparison table is presented in 4.12.

Firstly, I tested all active learning strategies that showed decent performance previously for each model from table 4.12.

Model	Parameters number	Original model
RobeCzech	125M	RoBERTa
Czert	110M	ALBERT
DistillBERT	66M	BERT
Small-e-Czech	13M	Electra

Table 4.12: Brief description of models used for Czech NLI task.

RobeCzech

The results of RobeCzech model on all AL strategies are shown in figure 4.18. The trend line of *BADGE* strategy seems to be the most superior among others, while *least confidence* strategy is inferior. There is also a *K-means* strategy which finally gets above the *random* one and the whole picture is similar to the one presented in classification task with smaller models. However, if we look at non-smoothed lines, which are presented more clearly in figure D.1, it is obvious that all models were improperly trained and experience from BERT-large model from classification step shows that hyperparameters have to be tuned better. It is done in subsection 4.4. Hence, I do not provide comparative analysis of all strategies for this experiment. However, it is also important to remember that the problem of unstable training can be caused by an inconsistent dataset, which has to be reviewed after sudden plunges of metric scores.

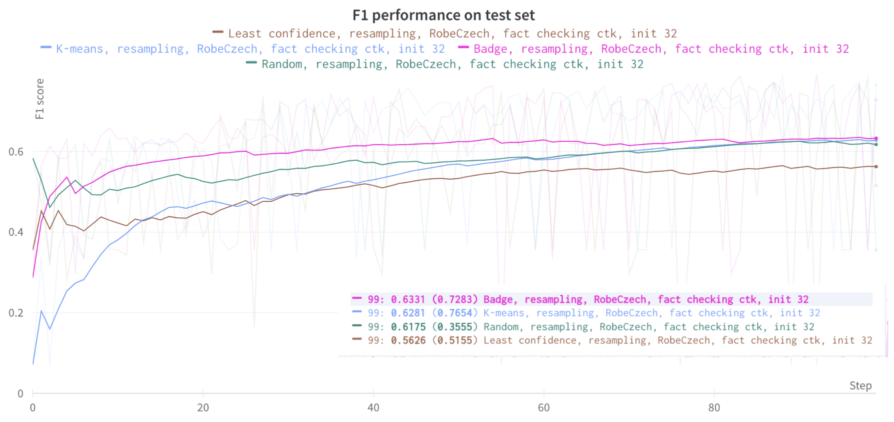


Figure 4.18: Czech NLI task on CTKFactsNLI dataset. RobeCzech-based model. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Dark lines show trend lines after EMA smoothing, light lines show exact values.

Czert

The performance of all active learning strategies on a Czert-based model is reported in figure 4.19. We see that during the first part of active learning steps, all models show poor stability. *Random* strategy is performing a little bit better than others. However, the maximum result is for all four strategies is 0.626 for a *random* strategy at step 40.

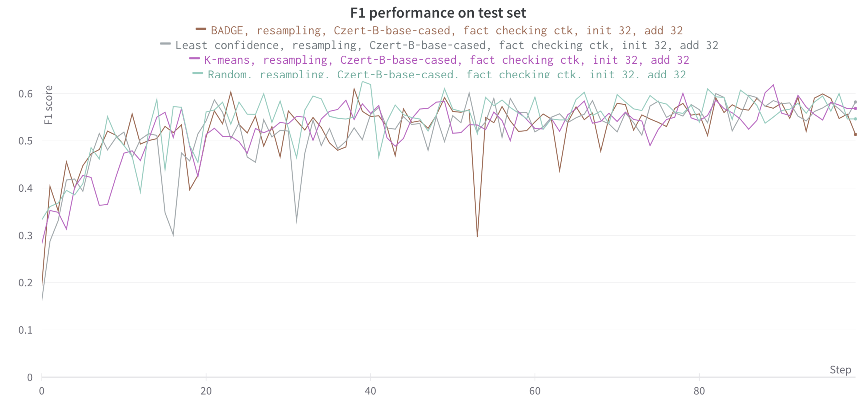


Figure 4.19: Czech NLI task on CTKFactsNLI dataset. Czert-based model. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Non-smoothed version with real F1 scores reported.

DistillBERT

DistillBERT model behaves similarly to Czert model (see fig. 4.20). Even though the number of parameters of the former is lower than of the latter, it is still unstable and this instability is possibly caused by the dataset itself.

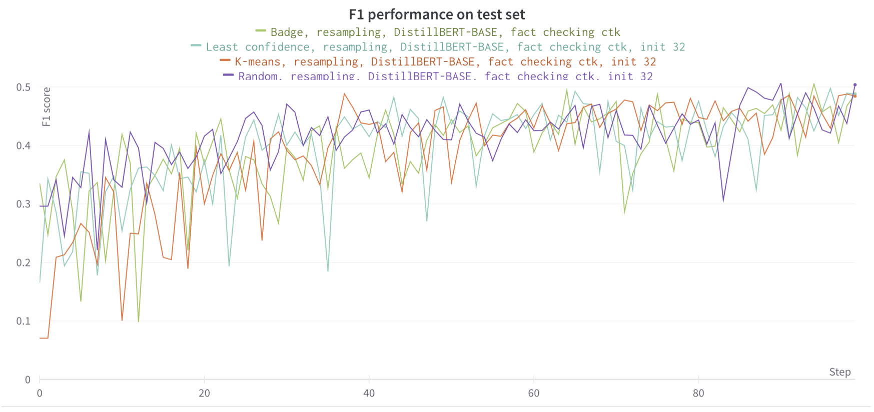


Figure 4.20: Czech NLI task on CTKFactsNLI dataset. DistillBERT-based model. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Non-smoothed version with real F1 scores reported.

Small-e-Czech

Small-e-Czech is the smallest model among all used for NLI experiments. Figure 4.21 shows smoothed trend lines and exact values. Again, we see that *random* strategy performs better than others, it's a clear signal that something is malfunctioning. Again, it is clear from non-smoothed graph depicted in figure 4.22 that F1 metric score is moving between 0.17 to 0.5, which is unreliable. All strategies were however capable of scoring relatively high F1 metric:

a. Random sampling - **0.5** at step 87.
b. Least confidence - 0.48 at step 68.
c. BADGE - 0.493 at step **62**.
d. K-means - 0.478 at step 90.

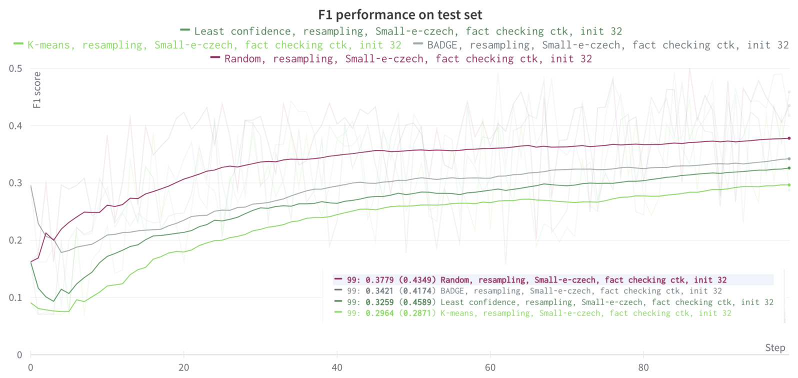


Figure 4.21: Czech NLI task on CTKFactsNLI dataset. Small-e-Czech-based model. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Dark lines show trend lines after EMA smoothing, light lines show exact values.

Nevertheless, *BADGE* strategy was capable of finding better training dataset, using only 1984 examples instead of 2784 as in case of *random* strategy. To recall, at each step I add 32 new examples and step 1 is initialised with only 32 examples. However, I still suspect that either all models had incorrect hyperparameters setups or the dataset itself has inconsistencies or is hard to process with small training set, given that NLI tasks are generally harder than classical classification. It is also possible that since I translated this task to a classification task via concatenation of two strings, the resulting string was too long for the model to process, since Small-e-Czech has embedding size of only 128 tokens, all sequences longer than 128 tokens get truncated. After a quick analysis of encoded sequences lengths I have discovered that out of 3626 train texts there are 933 (or 25.7%) that have an encoded

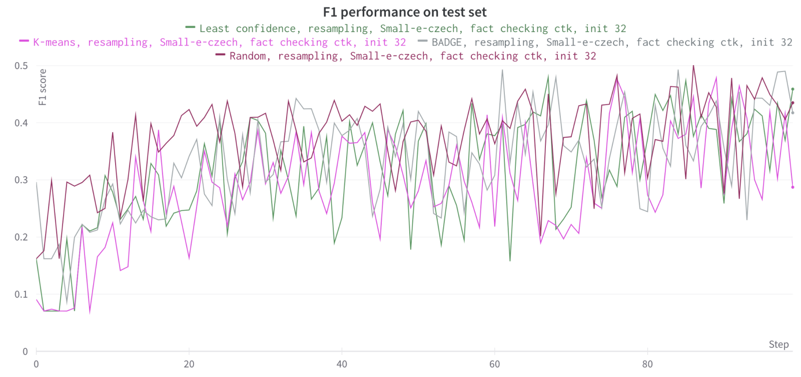


Figure 4.22: Czech NLI task on CTKFactsNLI dataset. Small-e-Czech-based model. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Non-smoothed version with real F1 scores reported.

representation longer than 128 tokens, thus a quarter of all training data gets truncated and thus the information about the claim is partly or fully missing, which means that Small-e-Czech is not the best model for CTKFactsNLI dataset. As for validation set, there are 54 entries out of 482 (or 11.2%) of texts with a representation longer than 128 tokens and for the test set it is 98 out of 558 (or 17.5%) texts.

RobeCzech - better hyperparameters

From previous sections it is evident that the behaviour of all four models is not stable and possibly there is a data consistency issue in the dataset. However, I have also attempted to apply the same hyperparameters trick on the best performing model - RobeCzech - and to look at the result. The only difference is that here the learning rate was set to 1e-5, not 2e-5. Figure 4.23 shows smoothed trend line of four strategies with new hyperparameters, while figure 4.24 shows exact values during AL experiments for the sake of completeness.

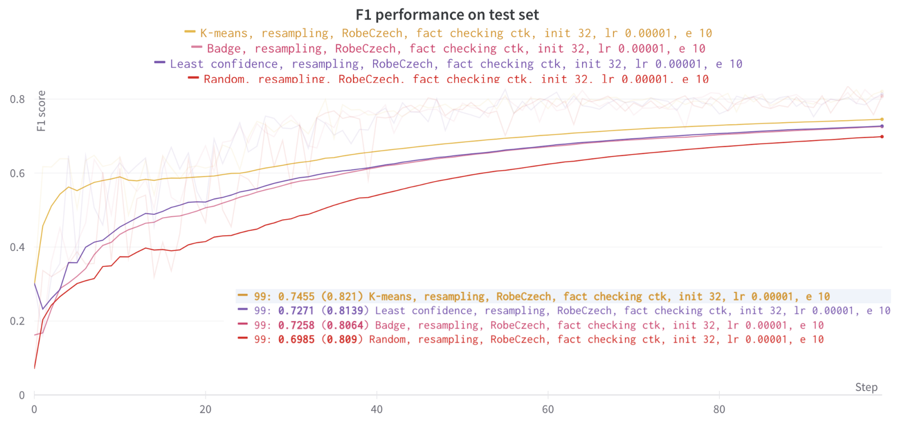


Figure 4.23: RobeCzech with new hyperparameters. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Dark lines show trend lines after EMA smoothing, light lines show exact values.

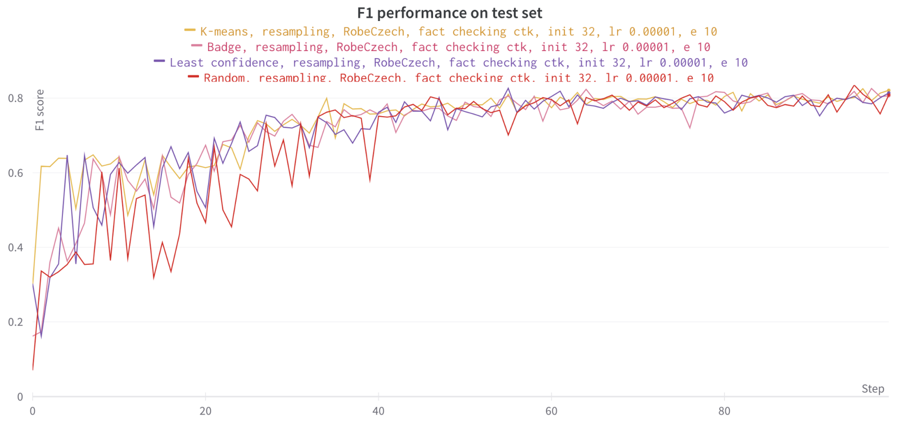


Figure 4.24: RobeCzech with new hyperparameters. F1 test set score development of four strategies: *random*, *least confidence*, *K-means*, *BADGE*. Non-smoothed version with real F1 scores reported.

First of all, the effect of new hyperparameters can be clearly seen: exact values of F1 metric have stabilised after approximately 40th step. Before we see large oscillations of a *random* strategy, while other strategies are more stable after first 15-20 steps. Batch strategies - *K-means* and *BADGE* are almost stable after 15th step and are consistently higher than a *random* strategy, which is clear from smoothed trend lines. Here *K-means* showed much higher capacity from the beginning and achieved high F1 scores earlier than others (0.8 at step 35). However, at later steps all four strategies show the same values and there is no dominant one. The best attained F1 score together with corresponding step are:

a. Random sampling - **0.835** at step 96.

b. Least confidence - 0.827 at step **56**.

c. BADGE - 0.824 at step 65.

d. K-means - 0.825 at step 97.

metrics	Random	Least confidence	K-means	BADGE
Test Loss	0.5625	**0.542**	0.566	0.579
F1	**0.835**	0.827	0.825	0.824
Recall	**0.835**	0.83	0.826	0.828
Precision	**0.849**	0.84	0.831	**0.849**

Table 4.13: Results on CTKFactsNLI NLI task for RobeCzech model for all four AL strategies. Maximum performance among all AL steps is reported.

More precise values for these experiments are presented in table 4.13.

Formally, random sampling was capable of scoring the highest value among all strategies, though with an extremely low margin (only 0.011 compared to *BADGE*). However, it is noteworthy that *BADGE* strategy reached its top value at 65th step, thus requiring only 2080 examples (compared to 3072 examples needed by *random* strategy), which is 68% of *random* strategy's dataset size or 32% reduction of required samples.

K-means strategy reached F1 score of 0.8 at step 35, thus requiring only 1120 examples. Although it is almost 4% less than the result of *random* sampling, it needed 36.5% of data compared to *random* strategy. From the perspective of costs of annotation by a domain expert, both *BADGE* and *K-means* strategies offer great saving of funds, which is an incredible result.

It is also important to say that the best F1 score is always computed with parameter *average* set to *"weighted"*. For comparison I have taken the evaluation table (with predictions on test dataset) for the best performing step (65) of *BADGE* strategy and then also recalculated this value to "macro" score so that I stick to the same evaluation scheme as in

[UDR+22], and I have received **0.807** or **80.7%**, which is **a new state-of-the-art result** for RobeCzech model on this dataset (RobeCzech model in [UDR+22] achieved 67.7%). I also provide classification report with detailed statistics for each category in figure 4.14 and corresponding confusion matrix for this active learning step, shown in figure 4.25.

	precision	recall	f1-score	support
0	0.806	0.652	0.721	115
1	0.874	0.831	0.852	183
2	0.804	0.900	0.849	260
accuracy			0.826	558
macro avg	0.828	0.794	0.807	558
weighted avg	0.827	0.826	0.824	558

Table 4.14: RobeCzech with new hyperparameters. Czech NLI task on CTK-FactsNLI dataset. Classification report for the active learning experiment with *BADGE* strategy (step 65 - 2080 examples - 58.4% of full dataset) achieving the best F1 metric score on test dataset. F1 score is shown with *average* parameter set to "macro".

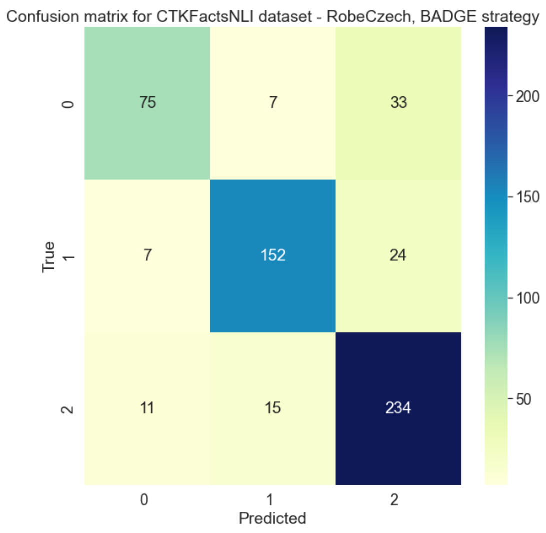


Figure 4.25: RobeCzech with new hyperparameters. Czech NLI task on CTK-FactsNLI dataset. Confusion matrix for the active learning experiment with *BADGE* strategy (step 65 - 2080 examples - 58.4% of full dataset) achieving the best F1 metric score on test dataset.

It is also noteworthy that the minimum values of test loss score (shown in figure 4.26) is achieved around 50th-60th steps, after which the score started to grow. Hence, the idea that the best performance is achieved around these steps is again supported.

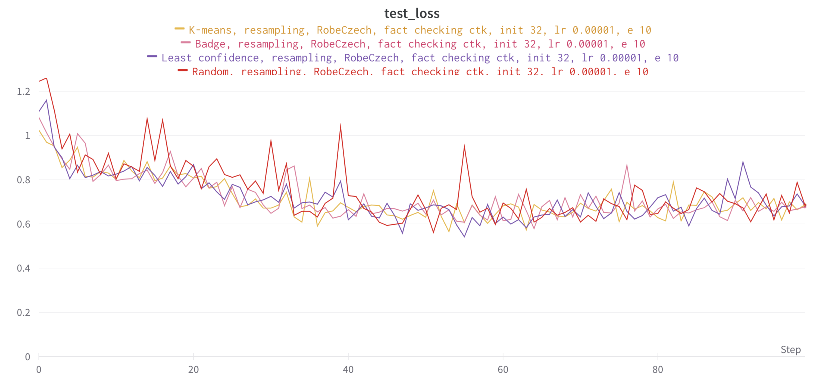


Figure 4.26: RobeCzech with new hyperparameters. Czech NLI task on CTK-FactsNLI dataset. Test set loss function exact values for all strategies reported.

All models comparison

In the light of the aforementioned it is clear that RobeCzech model performed the best even without hyperparameters adjustment. Thanks to the deeper analysis of the hyperparameters I am sure that there is no big issue in the dataset itself, however its inconsistencies can be studied by the spikes in F1 score or by spikes in test loss values. Overall, table 4.15 summarizes all NLI experiments and shows performance of each model. It puts together all four models and all four active learning strategies.

Model	random	least confid.	K-means	BADGE
RobeCzech	**0.835**	0.827	0.825	0.824
Czert	**0.626**	0.601	0.619	0.610
DistillBERT	0.507	0.498	0.485	**0.518**
Small-e-Czech	**0.500**	0.48	0.478	0.493

Table 4.15: Results on the NLI task (CTKFactsNLI dataset) with class weight resampling. Maximum test set F1 score is reported for each model and each AL strategy.

Surprisingly, a *random* strategy achieved the best F1 metric score for all four models, even though with a small margin from any other active learning strategy. It does not necessarily mean that AL strategies are useless. Recall that in case of RobeCzech with better hyperparameters setup *K-means* and *BADGE* strategies showed a trend line which were higher by a margin of 2-3% than *random* strategy - and in my opinion it is the most valuable information. Even the table 4.16 with the corresponding AL step with the number of examples needed for achieving the best strategy score is not informative enough. It is clear that AL strategies are capable of reaching maximum metric score earlier than a random one, but these values are many times unreliable due to a stochastic nature of this experiment. Hence, the only trustworthy way of assessing active learning strategies is to look at the trend line which averages oscillations that happen during AL steps and visualises the margin of target metric score, not the particular value. By seeing which active learning strategy is behaving better, a researcher can make an informed decision of choosing the best performing strategy and use it in real life scenario, thus achieving faster results with the help of less data. Trend lines often converge to the same value, thus they do not necessarily introduce better overall performance (even though we have observed it in a classification and tagging tasks), but help accelerate the whole training process.

Model	**random**	**least confid.**	**K-means**	**BADGE**
RobeCzech	96 (3072)	**56 (1792)**	96 (3072)	65 (2080)
Czert	40 (1280)	53 (1696)	90 (2880)	**38 (1216)**
DistillBERT	91 (2912)	97 (3104)	**38 (1216)**	74 (2368)
Small-e-Czech	87 (2784)	68 (2176)	90 (2880)	**62 (1984)**

Table 4.16: Results on the Czech NLI task (CTKFactsNLI dataset) with class weight resampling. A step number with number of examples in parentheses for the best score are reported for each model and each AL strategy. RobeCzech model shown here is the one with better hyperparameters set (often labelled as RobeCzech 2).

Czech NLI - full training

The last thing to assess - active learning effect compared to full dataset training. I have trained all four models (plus an optimized RobeCzech) on CTKFactsNLI dataset with the very same parameters as in corresponding AL experiments - the only change was in the dataset size. The results are reported in table 4.17. It is evident that RobeCzech 2 model performed the best. However, F1 score on test set (0.811) was surpassed by the same model trained with *BADGE* strategy at step 65 (with the help of 2080 examples) with value of 0.824. The result shows that it was enough for the model to train on 57.4% of the full dataset to reach the same result. Overall, the active learning experiment showed that the model is capable of reaching almost 1.5% rise in F1 score, which is a success.

Model	**F1 test score**	**Precision**	**Recall**	Test loss
RobeCzech 2	**0.811**	**0.818**	**0.812**	**0.632**
RobeCzech 1	0.747	0.755	0.744	0.933
Czert	0.56	0.594	0.579	1.51
DistillBERT	0.42	0.407	0.507	1.131
Small-e-Czech	0.463	0.464	0.468	1.6

Table 4.17: Results on the NLI task (CTKFactsNLI dataset) on full dataset for each assessed model. F1 test set score reported. RobeCzech 1 denotes the initial setup of RobeCzech model. RobeCzech 2 denotes the adjusted setup with more stable performance.

Among the models presented in figure table 4.17 it can be seen that the order of models in terms of test set F1 score is almost the same as the number of parameters in the corresponding models - Small-e-Czech has the least number of parameters, but it performed better than DistillBERT, which is not a pure Czech model. Otherwise the results once again prove that for such a complex task like natural language inference the model size plays the most important role. However, together with active learning this mix is capable of achieving decent performance with a fraction of original dataset, bringing annotation cost reduction and as a result a dataset of higher quality.

Chapter 5

Future considerations

Active learning is one of the so-called Human-in-the-loop computer interaction techniques which is capable of reducing annotation costs for labelled dataset collection and model training, but it poses several limitations that have to be discussed. Most of this chapter comes from [ZSH22] as a relatively new study which I found while finishing my work.

5.1 Tasks complexity

Active learning is mostly studied with simpler tasks, e.g. for text classification in NLP. However, the advantage of such approach lies in shrinking annotation costs by better utilisation of an abundance of unannotated data. In my project I have studied active learning framework on the Czech NLI task, which is a task of natural language inference, which requires deeper language understanding. Datasets used for NLI are harder to collect and annotate consistently, it is therefore logical that active learning might come into play for harder tasks accomplished by larger language models. I am pleased to make a contribution to the active learning study with this work and I hope that there will be more experiments with harder tasks in the future with more clever strategies leveraging large language models' inner knowledge for cost-efficient and swift unlabelled examples retrieval. Preliminary work has shown that AL can be helpful for data collection for such tasks like NLI [MJL20].

5.2 Starting and stopping AL

It is an open question when to start active learning - either completely from scratch or after collection of *reasonably large* seed data. What is meant by "reasonably large" is also unknown. It can be seen from my work that I speak about instability of target metric score during first steps of AL experiments when I started with 32 labelled examples and added 32 examples after each AL step. Some might consider this approach inappropriate and argue that a larger seed sample had to be collected. Active learning might be started for the improvement of an already achieved *good* score, however it is still unclear and presumably task-specific when the dataset is already large enough for active learning to start (so that there is no further improvement provided by a small batch of data to a large initial dataset) [TLHS09].

One more question to consider is when to stop active learning. A simple solution might be when the target metric score is reached on a development set, however there are many other aspects to take care of. A general stopping criterion consists of three main aspects: *metric*, *dataset* and *condition*:

a. **Metric** - a development set metric performance might be a good choice, but this set has to be large enough and also has to be labelled. Sometimes it is simply impractical to assume a large development set.

b. **Dataset** - a dataset requires careful choosing so that it is not biased and represents the problem which is solved by the model. A large dataset is also time-consuming to evaluate during active learning experiments.

c. **Condition** - the first idea that comes into consideration is when a model surpasses a predefined threshold during evaluation. A threshold for a metric score has to be specified. This threshold could be a business requirement. However, a single metric score could be not enough for an optimal stopping criterion. There can be found an inspiration from early stopping [Pre00]: when a model shows consistent better scores than the predefined threshold for some period of time (or steps), then the whole process can be stopped. There are several works that study this question: [Vla08], [LS08], [AP11].

5.3 Real-life application

There are several considerations for the application of active learning in real life. A decision to utilise active learning framework has to be done based on several aspects: annotation cost, efficiency and wait time, data reuse, and starting and stopping criteria. These questions are not present while in simulation, however they play an important role in real life scenarios. The selection of a query strategy and other hyperparameters remains a great challenge as well. We have seen in this work that there is no one superior strategy and each of them has their strengths and weaknesses, however in real life one has to choose a specific strategy for the given application. There can be a hybrid approach of utilising a best performing strategy in the beginning of AL and then an uncertainty sampling method in the end, when the diversity of new examples plays minor role. Another possible consideration is to evaluate several setups on a related task (for instance, for Twitter sentiment analysis there could be a parallel research conducted), which could reveal the best performing model and strategy which could be then used for the target task.

Chapter 6

Conclusion

In this work I have studied the problem of active learning and designed not only the library for experiments and new strategies development, but also an experimental setup for strategies assessment and introduced different kinds of ways how to evaluate active learning strategies. I have shown on two datasets the capability of active learning framework of reaching comparable or even better performance on selected tasks and compared performance of different strategies and different models. I have deeply integrated my library with Weights & Biases MLOps platform and demonstrated the way how I utilised it for my experiments and for finding the best setup for all tasks.

For the task of Czech NLI represented by CTKFactsNLI dataset I was able to reach a **state-of-the-art** result with F1 metric score on a test set equal to **0.807**, which was achieved by a combination of RobeCzech model and *BADGE* active learning strategy.

Finally, I have provided a thorough analysis of the results I had during my experiments with several suggestions for performance assessment. I have run 161 experiments in total with just a fraction of them described in this work, compared four common and often claimed as state-of-the-art active learning strategies and experimented with popular English models represented by well-studied BERT models and also with many popular Czech models and compared them on a harder task of natural language inference - fact checking.

To conclude, I hope this work will inspire other researchers to continue studying various active learning scenarios addressing open questions from chapter 5 and finding better ways of unlabelled data selection. I am also happy to contribute to this research with the library that I developed and used for this thesis and which is hosted on my public GitHub repository [1] together with Weights & Biases integration and RCI integration. Feel free to carry on working on it and expanding for further tasks and for

[1]The source code for this project could be found at https://github.com/DevKretov/ntu_nlp_al.

further strategies, I am looking forward to seeing the new breakthroughs in this topic.

Appendix A

Bibliography

[AP11] Josh Attenberg and Foster Provost, *Inactive learning? difficulties employing active learning in practice*, SIGKDD Explor. Newsl. **12** (2011), no. 2, 36–41.

[AV07a] David Arthur and Sergei Vassilvitskii, *K-means++: The advantages of careful seeding*, Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (USA), SODA '07, Society for Industrial and Applied Mathematics, 2007, p. 1027–1035.

[AV07b] David Arthur and Sergei Vassilvitskii, *k-means++: the advantages of careful seeding*, SODA '07, 2007.

[AZK$^+$19] Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal, *Deep batch active learning by diverse, uncertain gradient lower bounds*, CoRR **abs/1906.03671** (2019).

[AZK$^+$20] ———, *Deep batch active learning by diverse, uncertain gradient lower bounds*, 2020.

[BBEZ00] Y. Bengio, Joachim Buhmann, M. Embrechts, and Jacek Zurada, *Introduction to the special issue on neural networks for data mining and knowledge discovery*, IEEE Transactions on Neural Networks **11** (2000), 545–549.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, *Neural machine translation by jointly learning to align and translate*, 2014.

[BGD$^+$21] Shaily Bhatt, Poonam Goyal, Sandipan Dandapat, Monojit Choudhury, and Sunayana Sitaram, *On the universality of deep contextual language models*, 2021.

[BHA$^+$21] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang, *On the opportunities and risks of foundation models*, 2021.

[Bie20] Lukas Biewald, *Experiment tracking with weights and biases*, 2020, Software available from wandb.com.

[BR20] Ravali Boorugu and G. Ramesh, *A survey on nlp based text summarization for summarizing product reviews*, 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), 2020, pp. 352–356.

[CDG$^+$21a] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar, *Batch active learning at scale*, CoRR **abs/2107.14263** (2021).

[CDG$^+$21b] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar, *Batch active learning at scale*, 2021.

[CLM$^+$15] Yukun Chen, Thomas A. Lasko, Qiaozhu Mei, Joshua C. Denny, and Hua Xu, *A study of active learning methods for named entity recognition in clinical text*, Journal of Biomedical Informatics **58** (2015), 11–18.

[dCLLA$^+$22] Casper da Costa-Luis, Stephen Karl Larroque, Kyle Altendorf, Hadrien Mary, richardsheridan, Mikhail Korobov, Noam Raphael, Ivan Ivanov, Marcel Bargull, Nishant Rodrigues, Guangshuo Chen, Antony Lee, Charles Newey, CrazyPython, JC, Martin Zugnoni, Matthew D. Pagel, mjstevens777, Mikhail Dektyarev, Alex Rothberg, Alexander Plavin, Daniel Panteleit, Fabian Dill, FichteFoll, Gregor Sturm, HeoHeo, Hugo van Kemenade, Jack McCracken, MapleCCC, and Max Nordlund, *tqdm: A fast, Extensible Progress Bar for Python and CLI*, September 2022.

[DCLT18a] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018.

[DCLT18b] ______, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805 (2018).

[DH] Tivadar Danka and Peter Horvath, *modAL: A modular active learning framework for Python*, available on arXiv at `https://arxiv.org/abs/1805.00979`.

[EDHG$^+$20] Liat Ein-Dor, Alon Halfon, Ariel Gera, Eyal Shnarch, Lena Dankin, Leshem Choshen, Marina Danilevsky, Ranit Aharonov, Yoav Katz, and Noam Slonim, *Active Learning for BERT: An Empirical Study*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Online), Association for Computational Linguistics, November 2020, pp. 7949–7962.

[FGW$^+$21] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy, *A survey of data augmentation approaches for nlp*, 2021.

[FSST97] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby, *Selective sampling using the query by committee algorithm*, Mach. Learn. **28** (1997), no. 2–3, 133–168.

[GG16] Yarin Gal and Zoubin Ghahramani, *Dropout as a bayesian approximation: Representing model uncertainty in deep learning*, 2016.

[GS13] Denis Grebenkov and Jeremy Serror, *Following a trend with an exponential moving average: Analytical results for* *a gaussian model*, Physica A: Statistical Mechanics and its Applications **394** (2013).

[GSV22] Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos, *A survey on automated fact-checking*, Transactions of the Association for Computational Linguistics **10** (2022), 178–206.

[HBFF22] Sophie Henning, William H. Beluch, Alexander Fraser, and Annemarie Friedrich, *A survey of methods for addressing class imbalance in deep-learning based natural language processing*, 2022.

[HCR$^+$16] Jiaji Huang, Rewon Child, Vinay Rao, Hairong Liu, Sanjeev Satheesh, and Adam Coates, *Active learning for speech recognition: the power of gradients*, 2016.

[HGD19] Rishi Hazra, Shubham Gupta, and Ambedkar Dukkipati, *Active learning with siamese twins for sequence tagging*, CoRR **abs/1911.00234** (2019).

[HPZC07] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chang, *Accessing the deep web: A survey*, Commun. ACM **50** (2007), 94–101.

[HR18] Jeremy Howard and Sebastian Ruder, *Universal language model fine-tuning for text classification*, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Melbourne, Australia), Association for Computational Linguistics, July 2018, pp. 328–339.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber, *Long Short-Term Memory*, November 1997.

[HXY15] Zhiheng Huang, Wei Xu, and Kai Yu, *Bidirectional lstm-crf models for sequence tagging*, 2015.

[JH10] Xin Jin and Jiawei Han, *K-means clustering*, pp. 563–564, Springer US, Boston, MA, 2010.

[JHC$^+$20] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao, *SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization*, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020.

[KJSR15] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush, *Character-aware neural language models*, 2015.

[KN21] Matěj Kocián, Jakub Náplava, Daniel Štancl, and Vladimír Kadlec, *Siamese bert-based model for web search relevance ranking evaluated on a new czech dataset*, 2021.

[KR18] Taku Kudo and John Richardson, *Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing*, 2018.

[Kre20] Anton Kretov, *Attention mechanism in natural language processing*, 2020.

[LCG+19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut, *Albert: A lite bert for self-supervised learning of language representations*, 2019.

[LH17] Ilya Loshchilov and Frank Hutter, *Decoupled weight decay regularization*, 2017.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, 2019.

[LS08] Florian Laws and Hinrich Schütze, *Stopping criteria for active learning of named entity recognition*, Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008) (Manchester, UK), Coling 2008 Organizing Committee, August 2008, pp. 465–472.

[MH] Ines Montani and Matthew Honnibal, *Prodigy: A modern and scriptable annotation tool for creating training data for machine learning models.*

[MJL20] Stephen Mussmann, Robin Jia, and Percy Liang, *On the Importance of Adaptive Data Collection for Extremely Imbalanced Pairwise Tasks*, Findings of the Association for Computational Linguistics: EMNLP 2020 (Online), Association for Computational Linguistics, November 2020, pp. 3400–3413.

[MKB+11] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur, *Extensions of recurrent neural network language model*, 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2011, pp. 5528–5531.

[MM04] Prem Melville and Raymond J. Mooney, *Diverse ensembles for active learning*, Proceedings of 21st International Conference on Machine Learning (ICML-2004) (Banff, Canada), July 2004, pp. 584–591.

[MRA20] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah, *Machine learning with oversampling and undersampling techniques: Overview study and experimental results*, 2020 11th International Conference on Information and Communication Systems (ICICS), 2020, pp. 243–248.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, *Distributed Representations of Words and Phrases and their Compositionality*, arXiv:1310.4546 [cs, stat] (2013), arXiv: 1310.4546.

[Nak18] Hiroki Nakayama, *seqeval: A python framework for sequence labeling evaluation*, 2018, Software available from https://github.com/chakki-works/seqeval.

[NTD$^+$22] Truong Thao Nguyen, François Trahay, Jens Domke, Aleksandr Drozd, Emil Vatai, Jianwei Liao, Mohamed Wahib, and Balazs Gerofi, *Why globally re-shuffle? revisiting data shuffling in large scale deep learning*, 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022, pp. 1085–1096.

[PB18] Martin Popel and Ondř ej Bojar, *Training tips for the transformer model*, The Prague Bulletin of Mathematical Linguistics **110** (2018), no. 1, 43–70.

[PNI$^+$18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, *Deep contextualized word representations*, 2018.

[Pre00] Lutz Prechelt, *Early stopping - but when?*

[RG19] Nils Reimers and Iryna Gurevych, *Sentence-bert: Sentence embeddings using siamese bert-networks*, Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 11 2019.

[RGC$^+$21] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A. Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann, *Data augmentation can improve robustness*, 2021.

[RNKC22] Omid Rohanian, Mohammadmahdi Nouriborji, Samaneh Kouchaki, and David A. Clifton, *On the effectiveness of compact biomedical transformers*, 2022.

[RXC$^+$21] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang, *A survey of deep active learning*, 2021.

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf, *Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter*, 2019.

[Set09] Burr Settles, *Active learning literature survey.*

[SN20] Christopher Schröder and Andreas Niekler, *A survey of active learning for text classification using deep neural networks*, 2020.

[SNSS21] Milan Straka, Jakub Ná plava, Jana Straková, and David Samuel, *RobeCzech: Czech RoBERTa, a monolingual contextualized language representation model*, Text, Speech, and Dialogue, Springer International Publishing, 2021, pp. 197–209.

[SPK+21] Artem Shelmanov, Dmitry Puzyrev, Lyubov Kupriyanova, Denis Belyakov, Daniil Larionov, Nikita Khromov, Olga Kozlova, Ekaterina Artemova, Dmitry V. Dylov, and Alexander Panchenko, *Active learning for sequence tagging with deep pre-trained models and bayesian uncertainty estimates*, CoRR **abs/2101.08133** (2021).

[SPP+21] Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík, *Czert – czech bert-like model for language representation*, 2021.

[SQXH19] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang, *How to fine-tune bert for text classification?*, 2019.

[SS18] Ozan Sener and Silvio Savarese, *Active learning for convolutional neural networks: A core-set approach*, 2018.

[SYL+17] Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar, *Deep active learning for named entity recognition*, CoRR **abs/1707.05928** (2017).

[TCLT19] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *Well-read students learn better: On the importance of pre-training compact models*, 2019.

[TKSDM03] Erik F. Tjong Kim Sang and Fien De Meulder, *Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition*, Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, 2003, pp. 142–147.

[TLHS09] Katrin Tomanek, Florian Laws, Udo Hahn, and Hinrich Schütze, *On proper unit selection in active learning: Co-selection effects for named entity recognition*, Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing (Boulder, Colorado), Association for Computational Linguistics, June 2009, pp. 9–17.

[TMC+21] Maria Tsimpoukelli, Jacob Menick, Serkan Cabi, S. M. Ali Eslami, Oriol Vinyals, and Felix Hill, *Multimodal few-shot learning with frozen language models*, 2021.

[UDR+22] Herbert Ullrich, Jan Drchal, Martin Rýpar, Hana Vincourová, and Václav Moravec, *Csfever and ctkfacts: Acquiring czech data for fact verification*, 2022.

[Vla08] Andreas Vlachos, *A stopping criterion for active learning*, Computer Speech Language **22** (2008), no. 3, 295–312.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2017.

[Wei66] Joseph Weizenbaum, *Eliza—a computer program for the study of natural language communication between man and machine*, Commun. ACM **9** (1966), no. 1, 36–45.

[WFK+21] Sinong Wang, Han Fang, Madian Khabsa, Hanzi Mao, and Hao Ma, *Entailment as few-shot learner*, 2021.

[Wik22a] Wikipedia contributors, *Coreset — Wikipedia, the free encyclopedia*, 2022, [Online; accessed 12-January-2022].

[Wik22b] ______, *Hierarchical clustering — Wikipedia, the free encyclopedia*, 2022, [Online; accessed 5-January-2023].

[Wik22c] ______, *Inside–outside–beginning (tagging) — Wikipedia, the free encyclopedia*, 2022, [Online; accessed 2-January-2023].

[Wik22d] ______, *Round-robin scheduling — Wikipedia, the free encyclopedia*, 2022, [Online; accessed 5-January-2023].

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean, *Google's neural machine translation system: Bridging the gap between human and machine translation*, 2016.

[YWC20] Shuoheng Yang, Yuxin Wang, and Xiaowen Chu, *A survey of deep learning techniques for neural machine translation*, 2020.

[Zhd19] Fedor Zhdanov, *Diverse mini-batch active learning*, 2019.

[ZSH22] Zhisong Zhang, Emma Strubell, and Eduard Hovy, *A survey of active learning for natural language processing*, 2022.

[ZWH$^+$22] Xueying Zhan, Qingzhong Wang, Kuan-hao Huang, Haoyi Xiong, Dejing Dou, and Antoni B. Chan, *A comparative survey of deep active learning*, 2022.

Appendix B

Technical description of active learning labelling procedure

During active learning experiments the process of manual labelling is simulated by this algorithm:

a. The new index column with values from 0 to the length of the training dataset is added to the train split. Hence, whole train split is indexed.

b. An empty list is created, which will store indices of those rows that are treated as "labelled".

c. The train split is divided into two splits: labelled - for rows from the list defined above, and unlabelled - for the remaining rows. Unlabelled split will be used as source for the selection of the the next batch of data "to label".

d. The split created for training is considered as training dataset and will be updated in the next AL iteration.

Appendix C

Weights & Biases - illustrations

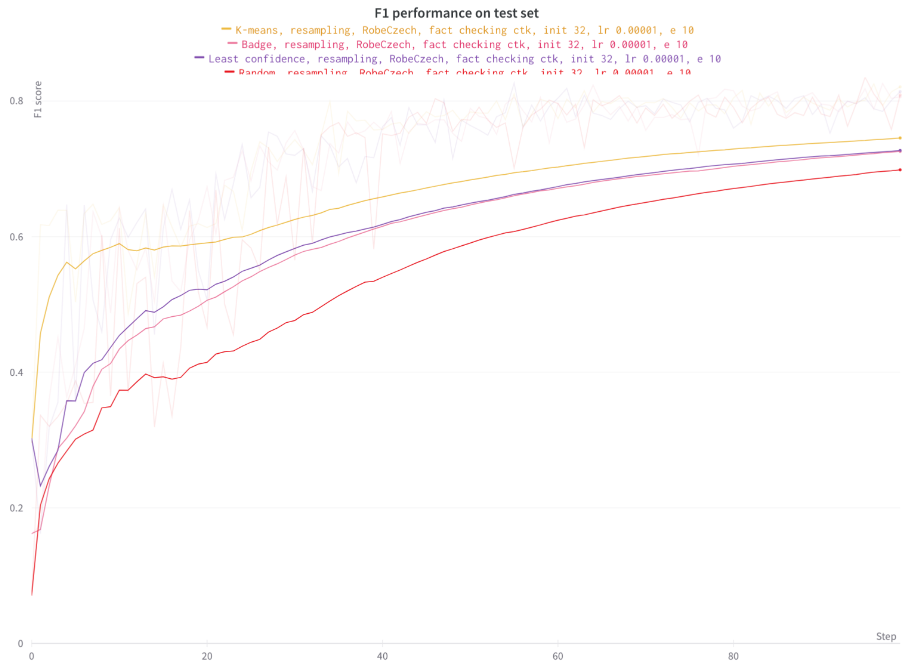
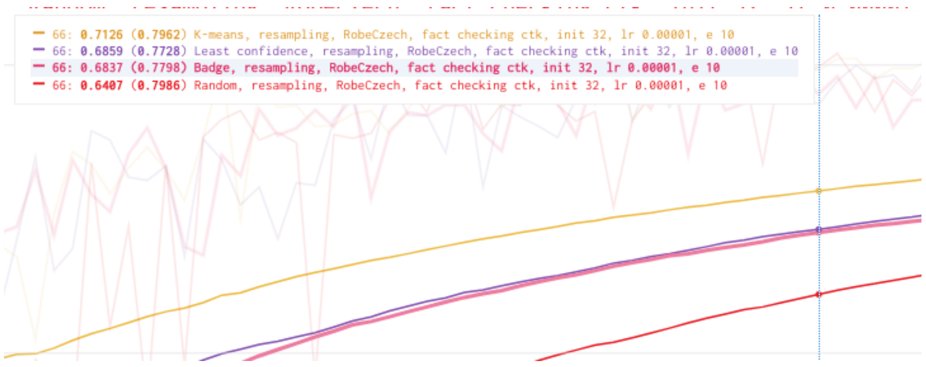


Figure C.1: Illustration of Weights & Biases metrics charts smoothing. Smoothing helps to see the trend and eliminate oscillations between steps - in active learning experiments this feature plays ultimate role. The picture also shows several metric score for several experiments, which are distinguished by differently coloured lines.

Charts plotted for one experiment in Weights & Biases are comparable with other experiments that share the same tracked variables, shown in C.1. The library draws lines of different colors and highlights particular experiments on the same grid so that a researcher can simply compare several lines, i.e. several experiments, at the same time, see C.2. This feature gave me much room for experimentation and comparison.

The last feature of high importance is system monitoring: CPU/GPU temperature, power usage, GPU utilisation or ratio of time spent ac-



```
— 66: 0.7126 (0.7962) K-means, resampling, RobeCzech, fact checking ctk, init 32, lr 0.00001, e 10
— 66: 0.6859 (0.7728) Least confidence, resampling, RobeCzech, fact checking ctk, init 32, lr 0.00001, e 10
— 66: 0.6837 (0.7798) Badge, resampling, RobeCzech, fact checking ctk, init 32, lr 0.00001, e 10
— 66: 0.6407 (0.7986) Random, resampling, RobeCzech, fact checking ctk, init 32, lr 0.00001, e 10
```

Figure C.2: A close-up of interactive part of the experiments charts - the library gives an opportunity to see the exact variable value at every step and to compare it with other experiments.

cessing memory instead of computing. All these metrics help leverage GPU under the hood and see how large language models fit into GPU memory, how GPU is utilised and whether it is possible to use it better - increase a batch size, run experiments in parallel, etc. An example of such monitoring for two experiments which utilise GPU differently is shown in figure C.3.

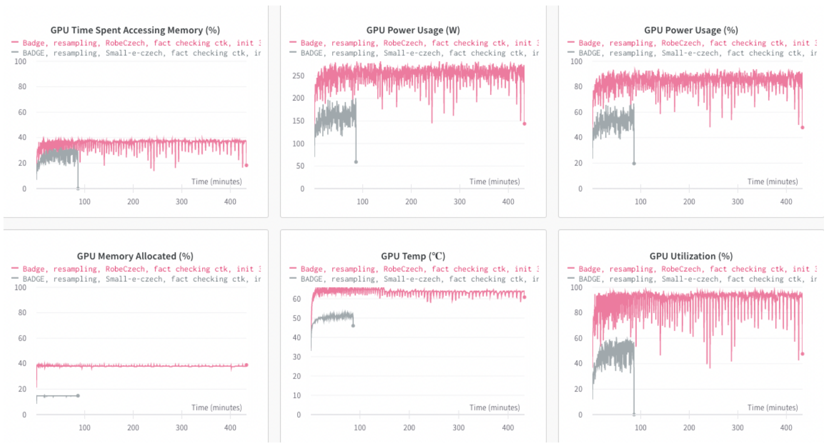


Figure C.3: Illustration of several system parameters (GPU state) that are natively supported by Weights & Biases and that are automatically monitored during each experiment. GPU utilisation charts help to figure out how the hardware is capable of processing large batches and store large models in GPU RAM.

Appendix D

Other figures

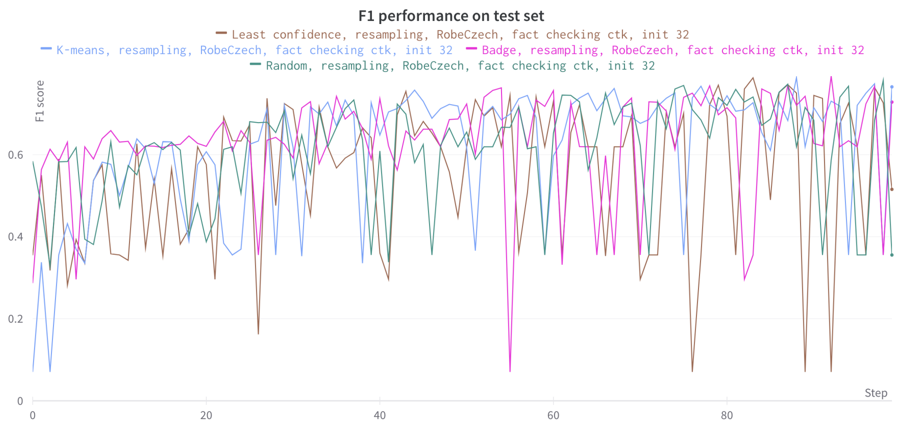


Figure D.1: F1 test set score development of four strategies: random, *least confidence*, *K-means*, *BADGE*. Non-smoothed version with real F1 scores reported. Czech NLI on CTKFactsNLI with RobeCzech model (in initial hyperparameters setup) results are reported.

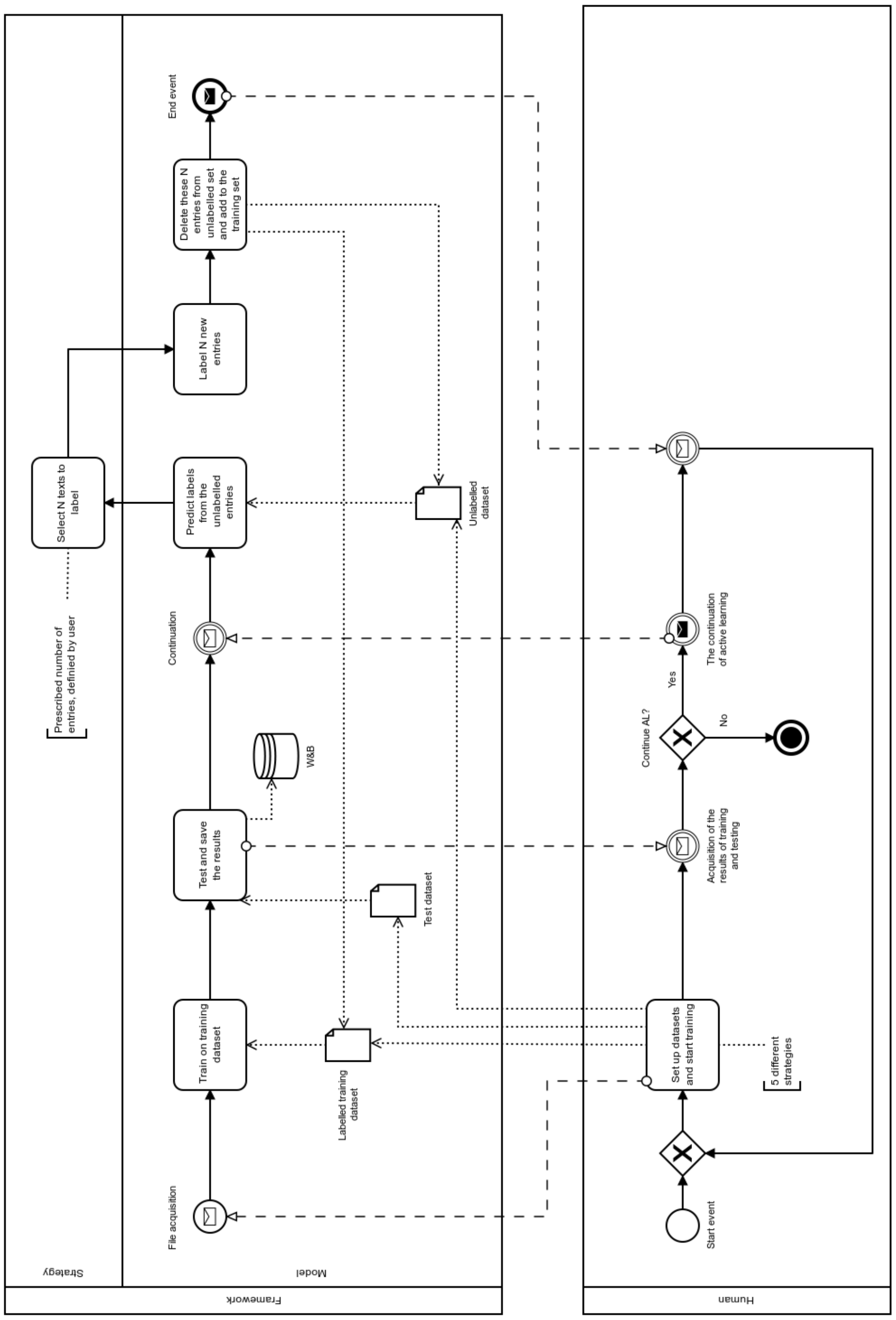

Framework
Strategy
Model
Human
File acquisition
Train on training dataset
Test and save the results
Continuation
Predict labels from the unlabelled entries
Label N new entries
Delete these N entries from unlabelled set and add to the training set
End event
Select N texts to label
Prescribed number of entries, defined by user
W&B
Labelled training dataset
Test dataset
Unlabelled dataset
Start event
Set up datasets and start training
5 different strategies
Acquisition of the results of training and testing
Continue AL?
No
Yes
The continuation of active learning

Figure D.2: BPMN process diagram of the implemented library.