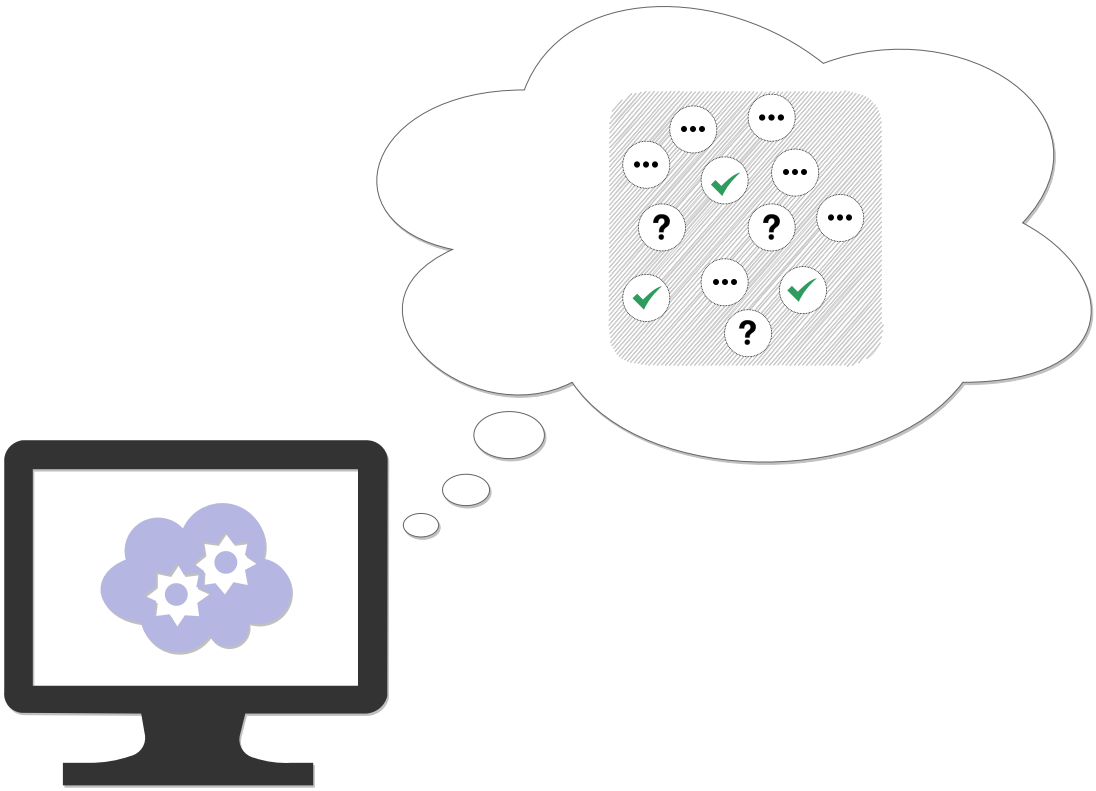

Deep Active Learning for Swedish Named Entity Recognition

An empiric evaluation of active learning algorithms for Named Entity Recognition

Master's thesis in Computer Science Algorithms, Language and Logic

Nadim Hagatulah
Kalle Arvidsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Deep Active Learning for Swedish Named Entity Recognition

An empiric evaluation of active learning algorithms for Named Entity Recognition

Nadim Hagatulah
Kalle Arvidsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology
Gothenburg, Sweden 2021

Deep Active Learning for Swedish Named Entity Recognition
An empiric evaluation of active learning algorithms for Named Entity Recognition

Nadim Hagatulah
Kalle Arvidsson

© Nadim Hagatulah 2021.
© Kalle Arvidsson 2021.

Academic Supervisor: Richard Johansson, Department of Computer Science and Engineering
Company Supervisor: Petter Wolff, Sahlgrenska Science Park
Examiner: Krasimir Angelov, professor, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The box illustrates a set with circles as data points. The checkmarks represent data points in the labeled set, question marks represent data that is queried to an oracle, and the triple dots represent the unannotated set.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Deep Active Learning for Swedish Named Entity Recognition
An empiric evaluation of active learning algorithms for Named Entity Recognition

Nadim Hagatulah
Kalle Arvidsson

Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Named entity recognition holds promise for numerous practical applications involving text data, such as keyword extraction and automated anonymization. However, successfully train a machine learning model for Named Entity Recognition is challenging due to the amount of annotated data required, especially for cases where language that is not globally common such as Swedish is involved. In such cases, using a Deep pre-trained model such as BERT in conjunction with the practice of active learning may be preferred. To obtain some insight into the implementation of such an approach, this thesis serves as an empirical study of various active learning strategies when used in conjunction with BERT-based name entity recognition. The performance of different active learning algorithms and the effect of acquisition size on the performance of active learning is the main focus of this study. In conclusion, after comparing and evaluating 17 different active learning methods, the study's empirical results demonstrate entropy sampling to be the best performing active learning algorithm for Named Entity Recognition of Swedish texts, and the choice of acquisition sizes is practically negligible to performance.

Keywords: Active Learning, Deep Learning, Transformer, BERT, NLP, Named Entity Recognition, Diversity-Based Sampling, Uncertainty-Based Sampling, Pool-Based Sampling, Cumulative Training.

Acknowledgements

The authors would like to thank our supervisors Richard Johansson and Petter Wolff, that throughout the whole project, have given invaluable help and feedback. The thesis would not have been possible without both of their involvement and helpful support.

Nadim Hagatulah and Kalle Arvidsson, Gothenburg, June 2021

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Purpose . . . 1
1.2 Related Works/Previous works . . . 2

2 Theory **3**
2.1 Learning Algorithms . . . 3
2.1.1 Supervised Learning . . . 3
2.1.2 Unsupervised Learning . . . 4
2.2 Active Learning . . . 4
2.2.1 Active Learning in Deep Learning . . . 6
2.2.1.1 Batch Awareness . . . 6
2.2.1.2 Model Uncertainty . . . 7
2.2.2 Uncertainty Sampling . . . 7
2.2.3 Diversity Sampling . . . 9
2.2.4 Clustering . . . 10
2.2.5 Diverse mini-batch Active Learning . . . 10
2.2.6 BatchBALD . . . 10
2.2.7 Coreset . . . 11
2.2.8 Discriminative Active Learning . . . 12
2.2.9 Expected Gradient Length . . . 13
2.2.10 Query By Committee . . . 13
2.3 Natural Language Processing . . . 14
2.3.1 Text Representation . . . 14
2.3.2 Named Entity Recognition . . . 16
2.3.2.1 Tagging Schemes . . . 16
2.4 Transfer Learning . . . 17
2.5 Transformer . . . 18
2.5.1 BERT . . . 20

3 Method **23**
3.1 Hardware . . . 23
3.2 Software Framework . . . 23
3.2.1 Hyperparameters . . . 24

3.2.2 Sentence Embedding . . . 26
3.3 Data Set . . . 27
3.3.1 SUC 3.0 . . . 27
3.4 Metrics . . . 29
3.4.1 Accuracy . . . 30
3.4.2 Precision . . . 30
3.4.3 Recall . . . 30
3.4.4 F1-Score . . . 31
3.4.5 Time . . . 31
3.5 Learning Model . . . 31
3.5.1 BERT for NER . . . 32

4 Results **35**
4.1 Acquisition size 16 . . . 35
4.1.1 Uncertainty Sampling . . . 35
4.1.2 Diversity Sampling . . . 36
4.1.3 Time . . . 38
4.2 Acquisition size 50 . . . 38
4.2.1 Uncertainty Sampling . . . 38
4.2.2 Diversity Sampling . . . 39
4.2.3 Time . . . 41
4.3 Comparison based upon acquisition size . . . 41
4.4 Consistency of performance . . . 42

5 Discussion **43**
5.1 Performances of Query Methods . . . 43
5.2 Impact of Acquisition Size . . . 44
5.3 Future Works . . . 46

6 Conclusion **49**

A Appendix 1 **I**

B Appendix 2 **III**

C Appendix 3 **V**

D Appendix 4 **IX**

List of Figures

2.1 The active learning loop, which begins at the first stage which is predicting sample from the unlabeled data set. In the second stage, a query method ranks the informative score of the predictions and picks the N best scores for the oracle to manually annotate. After the oracle has annotated the samples, the labeled data set is extended and the model is retrained. 5

2.2 The active learning loop with DAL. Steps 1, 2, and 3 are repeated N times for each mini-query before step 4 is performed, which introduces batch awareness and allows the DAL loop to scale to large batch sizes. 12

2.3 Simplified view of the Skip-Gram architecture with the example sentence "Kalle ska äta pizza deg" where "äta" is the main word. 15

2.4 Simplified view of the CBOW architecture with the example sentence "Kalle ska äta pizza deg" where "äta" is the main word. 15

2.5 The transformer architecture presented in Vaswani et at [1]. Note that positional encoding is applied to the input, as the attention-based model is otherwise incapable of distinguishing sequential relationship. 20

2.6 The intended training procedure of BERT according to Devlin et at [2]. Fine-tuning for specific task can be done through adapting the output layer and the input formulation according to the intended task. 21

3.1 The frequency of sentence lengths in the SUC 3.0 data set. Starting at sentence length one which represents a one word sentence. 29

4.1 Average F1-score across three different seeds versus number of iterations for uncertainty based query methods, using acquisition size 16. 36

4.2 Average F1-score across three different seeds versus number of iterations for diversity based query methods and combined methods, using acquisition size 16. 37

4.3 Accumulated query time in seconds for tested query methods, using acquisition size 16. 38

4.4 Average F1-score across three different seeds versus number of iterations for uncertainty based query methods, using acquisition size 50. 39

4.5 Average F1-score across three different seeds versus number of iterations for diversity based query methods and combined methods, using acquisition size 50. 40

4.6 Accumulated query time in seconds for tested query methods, using acquisition size 50. 41

A.1 16 samples per iteration, uncertainty-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300. I
A.2 16 samples per iteration, diversity-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300. II

B.1 50 samples per iteration, uncertainty-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300. III
B.2 50 samples per iteration, diversity-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300. IV

C.1 Comparison of the naive clustering algorithm combined with different uncertainty-based sampling algorithms. The first graph is with acquisition size 16 and second is acquisition size 50. V
C.2 Comparison of the DBAL algorithm combined different uncertainty-based sampling algorithms. The first graph is with acquisition size 16 and second is acquisition size 50. VI
C.3 Comparison of different distance metrics for the coreset algorithm. The first graph is for acquisition size 16 and second is for acquisition size 50. The coreset legend uses the cosine distance metric. VII

D.1 Mean and deviations of BatchBALD, Core-Set and DAL. IX
D.2 Mean and deviations of C-Entropy, DBAL-Entropy and Entropy. . . . X
D.3 Mean and deviations Marginal, bald, least and random sampling. . . XI

List of Tables

2.1 Example of different schemes are applied to the sentence "Kalle Arvidsson wants to watch the movie Justice League or Naruto ." 17

3.1 Two shared computer resource servers offered by Chalmers, Titan and Bayes, where Bayes includes Controller, Shannon and Markov and Titan is the singular computer in Titan server. 23
3.2 The algorithms that are supported by the framework. MHS denotes Mahalanobis, CRL denotes Correlation and CB denotes Cityblock (Manhattan). 24
3.3 The mapping of a sentence to numeric sentence vector embedding. . . 26
3.4 Example of how two sentences in the data frame looks like. 27
3.5 The entities provided by the SUC 3.0 data set. Second column corresponds to mapping to more common and easier tags to work with. . 27
3.6 Example of how a named tag in SUC 3.0 looks like. 28
3.7 Frequency of tokens in the SUC 3.0 data set after the preproccesing steps. 28
3.8 An example of predicted tokens on the second column, with the BIO-scheme on the sentence "Kalle ska äta på Condeco" on the first column and, correct tokens on the third column. 30
3.9 NER experiments performed with the different pretrained BERT models that support the Swedish language. Time is in format (minutes:seconds). 32
3.10 An example of how informativeness score is calculated for NER. The sentence "Kalle Arvidsson reser till Kina" is fed into the pretrained KB-BERT model (before fine-tuning) with the predefined labels PRS,LOC,ORG,MISC and O (no tagging scheme). Each word in the sentence gains probabilities for each label. Applying the Least Confidence query method for each word yields the informativeness score shown in the bottom row. The far right column shows the resulting informativeness score for whole sentence. 33

4.1 This table captures the regions of interests, mainly from iteration 9-29, from Graph 4.1. Iteration index is zero based and acquisition size is 16. The F1-scores are rounded to two decimals precision points. . 36
4.2 This table captures the regions of interests, mainly from iteration 9-29, from Graph 4.2. Iteration index is zero based and acquisition size is 16. The F1-scores are rounded to two decimals precision points. . 37

4.3 This table captures the regions of interests, mainly from iteration 3-11, from Graph 4.4. Iteration index is zero based and acquisition size is 50. The F1-scores are rounded to two decimals precision points. . 39
4.4 This table captures the regions of interests, mainly from iteration 3-10, from Graph 4.5. Iteration index is zero based and acquisition size is 50. The F1-scores are rounded to two decimals precision points. . 40
4.5 This table provides for each uncertainty based method a comparison of the F1-scores between acquisitions sizes at certain points where the amount of sampled data are similar. The left hand side value of the bar in a cell is from acquisition size 16 and right hand side is acquisition size 50. 41
4.6 This table provides for each diversity related method a comparison of the F1-scores between acquisitions sizes at certain points where the amount of sampled data are similar. The left hand side value of the bar in a cell is from acquisition size 16 and right hand side is acquisition size 50. 42

1
Introduction

Unstructured text data, such as patient stories [3], holds promising potential for various applications. However, to make proper use of such unstructured text data, information extraction methods would be required. Extracting information from unstructured text data has been a commonly researched theme within the field of Natural Language Processing (NLP), with Named Entity Recognition(NER) being a prime example of such a subject. Using NER, one can highlight the valuable information or omit sensitive privacy information within a body of text.

At the time of this thesis, a common approach to handle NLP tasks in general is through using supervised machine learning(ML). However, a common challenge within the NLP field is the lack of annotated data. To train an accurate supervised machine learning model, large amounts of accurate annotated data are required. For an information extraction task involving unstructured text data, accurate annotations would require manual work by domain experts, which is costly and inefficient [4]. There is therefore an interest in reducing the amount of annotated data required to successfully train a supervised machine learning model.

1.1 Purpose

There has been numerous research about information extraction from unstructured texts but relatively few attempts of applying such techniques to Swedish texts [5]. As such, information extraction tasks in Swedish is still a subject that requires further development. For that end, sizeable accurately annotated data sets are required. The high-level goal of the project is then to find a way to efficiently acquire accurate annotations for unstructured text data. To this end, Named Entity Recognition for unstructured Swedish texts has been selected as a representative, but the general idea should be extensible to other languages and NLP tasks as well.

To handle the task of extracting information from unstructured Swedish texts despite the lack of annotated data, this thesis proposes using active learning in combination with a transformer-based transfer learning model. The purpose of this study is then to gain an insight on the practical usage of active learning with the consideration of the performance and scalability, specifically for the case of information extraction of unstructured texts using supervised machine learning. In order to fulfill this purpose, the objective of this study is to investigate the performance, in particular the convergence rate of model performance but also the time consumption, of var-

ious Active Learning algorithms on the training of transformer-based NLP models on Swedish texts. For this end, the idea is to develop an annotation framework that can be used to evaluate active learning methods for reducing the amount of annotated data required for the model to reach a reasonable performance [6].

1.2 Related Works/Previous works

Using active learning for machine learning concerning NLP tasks has been done in numerous researches in conjunction with a variety of different machine learning models such as conditional random fields (CRF), convolutional neural networks (CNN), recurrent neural networks (RNN), and variations of BERT model. In all of those studies, various types of active learning algorithms were shown to have an advantage over randomly sampling data for training set in NLP tasks.

In studies related to CRF-based sequence labeling tasks, uncertainty based methods were shown to to have an advantage over other methods such as query-by-committee [7], information density [8], and diversity based sampling [9]. Additionally, Kholghi et al (2015) showed in their study concerning CRF-based entity recognition that not retrain the entire model from scratch for each active learning iteration reduces the amount of labeled data required to reach the performance comparable to that of supervised training [8].

As for the studies concerning active learning in conjunction with neural network based NLP, Shen et al (2018) evaluated three different uncertainty based active learning algorithms on named entity recognition using a CNN-LSTM model [6]. All three algorithms performed similarly, with the least confidence method underperforming slightly compared to mean normalized log-probability and Bayesian active learning by disagreement. Said study also noted the computational advantage of mean normalized log-probability over Bayesian active learning by disagreement, as the latter required multiple forward passes.

The study which is most similar to this is Ein-Dor et al (2020), which evaluated and compared a variety of active learning algorithms on binary text classification tasks using BERT [10]. Experiments were performed across a number of different data sets with different categorization to classify, with the target class prior of the data sets ranging from 10% to 50%. The research has shown that none of the tested active learning algorithms consistently outperformed its counterparts in the case of BERT-based binary text classification. The run-time difference of different active learning algorithms is also noted in the study.

2
Theory

This chapter discusses the theory behind the key concepts of the study. The background and general idea of active learning are thoroughly discussed. Popular acquisition methods from literature that will be used for experiments are also introduced and motivated. An overview of the NLP task which will be solved with active learning is discussed then finally the transformer model is thoroughly explained.

2.1 Learning Algorithms

A machine learning model generally requires a learning algorithm and training data. The ML model artifact is created by the training process, and the goal of the training process is for the ML model to capture the patterns which lie within the training data. It is therefore important for the training data to be sufficiently representative of the domain in order to produce a well-functioning model which correctly generalizes to predicting new unseen data. There are multiple different training algorithms that utilize the training data in different methods in order to best capture the patterns for the ML model.

2.1.1 Supervised Learning

In supervised learning, a model is typically trained by feeding it with training data that have labels that are the desired mapping to the data points. The weight parameters of the model are then adjusted until a loss function is reduced to a satisfactory level, which indicates that the model has learned the patterns of the training data well enough.

Supervised learning has some challenges which include requiring domain experts to label a sufficient amount of labeled data for a model to be able to converge to a satisfactory level [11]. However, factors including experiences, emotions, and circumstances would often affect a person's judgment during annotation. As such, annotations from a single person would regularly contain bias and other mistakes, which means they are not reliable. It is therefore generally required to have multiple annotators to guarantee the integrity of the data annotations through redundancy, which in turn makes the process expensive and time-consuming. A popular method to measure agreement between annotators is with Cohen's kappa [12] which assigns a score based on the agreement between annotators.

2.1.2 Unsupervised Learning

In contrast to supervised learning, unsupervised learning does not require any annotations of the training data. This is because the goal of an unsupervised learning algorithm is to find structure in the training data on its own by finding patterns based on data points features, often in the form of clusters or density estimation. The challenges that unsupervised learning presents are that it is a more complex method than supervised learning and therefore generally requires more data and time for convergence in training. The results can also be hard to interpret, and, because no labels are given, the correct answers can not be accurately determined which allows the risk of erroneous results [13]. This would require external evaluation from a human domain expert or an internal evaluation in the form of an objective function.

2.2 Active Learning

Annotating data for supervised learning is usually a task with high demands for time and manpower, especially if the task involves expertise in a certain field. In order to reduce the amount of annotated data required to train a model, one may apply the practice of active learning which is to use an algorithm to find a sample set within an unannotated data set that is most likely to yield the most significant performance improvement for a given ML model [14] [7], should the sample set be manually annotated and used to train the given model.

There are generally three active learning scenarios [15]. **Membership Query Synthesis** where the learner generates fictional data point instances based on the underlying distribution of unlabeled pool. The generated data points are queried to an oracle for a label and are accepted as a successfully generated data point if the oracle can recognize and assign a label, otherwise if unrecognizable, the data point will be discarded. **Stream-Based Selective Sampling** where unlabeled data points are examined one by one against some informativeness criteria. If the examined data point is accepted according to the informativeness criteria, it is then queried to the oracle for a label assignment but otherwise discarded. **Pool-Based Sampling** considers the whole unlabeled data set by assigning all data points an informativeness score and then selects the top highest scoring data points.

Membership query synthesis is only useful for cases where the unlabeled data pool is small [15] and the focus is therefore on expanding the data set. The Stream-based Selective Sampling and Pool-Based Sampling scenarios both rely on an informativeness score. The Pool-Based Sampling method has to perform an expensive informativeness assignment over the whole unlabeled data set before deciding which data points to include, while on the contrary Stream-Based Selective Sampling immediately decides for each data point. Therefore if resources and hardware are limited then Stream-Based Selective Sampling is the appropriate method, otherwise, Pool-Based Sampling generally produces better results since it has an underlying knowledge of whole unlabeled data set [16].

The algorithm which determines the informativeness of the sample set drawn from the unlabeled data set is called a query method. There exists a variety of query methods [7] with the most common approaches being uncertainty-based sampling and diversity-based sampling. In uncertainty-based sampling, the sample set is constructed by using the samples which the model is most uncertain of, whereas in diversity-based sampling the algorithm tries to select a diverse set of queried samples.

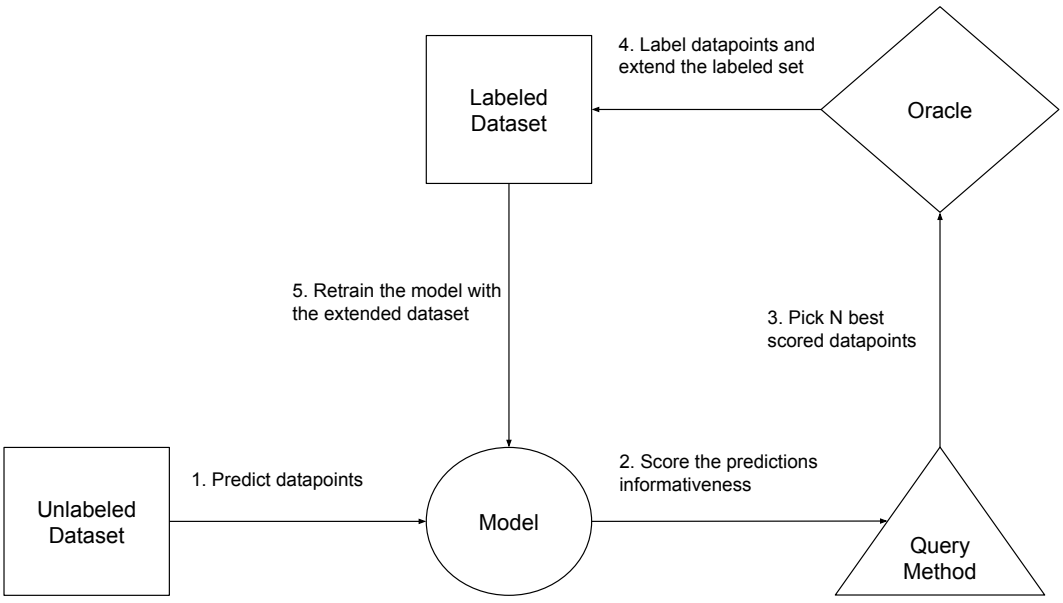

Figure 2.1: The active learning loop, which begins at the first stage which is predicting sample from the unlabeled data set. In the second stage, a query method ranks the informative score of the predictions and picks the N best scores for the oracle to manually annotate. After the oracle has annotated the samples, the labeled data set is extended and the model is retrained.

The samples are assigned an informativeness score and then the most informative data is queried to an oracle, which is an agent such as a human annotator. The oracle annotates the sample with a ground truth label so it can be used for the supervised training of the model. The training in active learning can be done in two methods. The first being Incremental learning, which is to use the already trained model and update the weights with only the newly acquired data from the iteration in the loop. The second method is Cumulative training where the model weights reset every iteration and is then retrained from scratch with all the cumulative labeled data acquired from the active learning process [17]. The whole general

active learning loop is described in Figure 2.1.

2.2.1 Active Learning in Deep Learning

Combining deep learning with active learning presents some challenges. The main issue is that the Softmax output from deep learning models tends not to correctly capture the uncertainty of the model for the classes [18] and the fact that most active learning algorithms are intended for one-by-one query and are not batch aware.

2.2.1.1 Batch Awareness

In theory, uncertainty-based active learning should query one sample per iteration. This is impractical in cases involving deep learning models for two reasons. First, deep learning models are usually trained with batches of multiple samples for the purpose of parallelization and stabilization of the training. Second, to retrain the model from scratch for each sample added to the training set is inefficient as Deep Learning models are usually resource-intensive to train. Thus, it is more practical to query samples in batches of size K where K is an acquisition size greater than one. However, the intuitive approach of naively choosing top-K highest informative scoring samples will usually result in K near-identical samples [19].

A method to alleviate the problem is to choose K to be small enough in between training rounds. If at some point model M is uncertain in predicting some class c, then if K is chosen to be large, a large number of similar samples consisting of class c would be chosen. The training would be inefficient because the model would always be skewed towards some dominating class c which affects the generalization of the model negatively. Instead, if K was chosen to be small, then sufficient identical K points would be chosen for the model's generalization to not be affected but instead positively improved on the weak class c predictions.

Another method is solving the issue through different methods of diversity-based sampling, which sought to prevent the querying of near-identical samples. Such an approach would require the diversity-based method to perform well on the unannotated data. For instance, in order for diversity sampling with clusters to succeed the cluster sizes and centroid positions needs to be well placed, which requires a good representation of the data points. A cluster size that is too small would create some large dominating cluster and other small clusters, if the smaller clusters are depleted, then querying will only occur from the dominating cluster which results in only identical points being chosen.

Algorithms such as DAL, DBAL, Coreset, and BatchBALD are developed with batch awareness in mind and are more thoroughly discussed in the subsequent sections.

2.2.1.2 Model Uncertainty

Usually, the uncertainty-based query methods rely on the predictions of the learning model. However, the Softmaxed raw output from a deep learning model tends to be overconfident [20], e.g high confidence can be assigned to unseen data and therefore the Softmax output of the final layer is unreliable as a measure of confidence and shown in the literature to be worse than random sampling [21].

A method to estimate uncertainty is to cast the deep learning models as Bayesian models and thus leveraging Bayesian probability theory which offers tools to reason about the model confidence in predictions. By enabling dropouts during the predictions of the model, outputs are no longer deterministic. Each run will produce different results which are interpreted as samples from a probabilistic distribution. Thus, averaging over multiple Softmax outputs allows the model to be Bayesian approximated as the Gaussian process [20].

Using dropout during evaluation is called the **Monte Carlo dropout (MC)** method, where every neuron in some layer has a probability p to be dropped, (the value of the neuron multiplied with 0). Applying Monte Carlo dropout multiple times and summing the results is the equivalent to Monte Carlo integration, thus for some $p(y = c|x)$ where x denotes a sample and given a sample, the probability for class c can be represented through the following formula:

$$p(y = c|x) = \int p(y = c|x, \omega)p(\omega)d\omega$$

Here, $p(\omega)$ denotes the probability for a weight to be active or not, that is if the neuron is dropped, thus the integration over ω considered all possible weights by its probability, therefore:

$$p(y = c|x) \approx \int p(y = c|x, \omega) * q^*(\omega)d\omega$$

Where $p(\omega) \approx q^*(\omega)$ and $q^*(\omega)$ denotes the dropout distribution. The Monte Carlo integration can further be approximated by running multiple instances T and then averaging over some probability class c, which finally gives:

$$p(y = c|x) \approx \frac{1}{T}\sum_t p(y|x, \omega_t) = \frac{1}{T}\sum_t p_c^t$$

This yields the output probability for every class c. A higher amount of runs T samples more from the assumed probability distribution, which therefore gives a better approximation of the probability distribution.

2.2.2 Uncertainty Sampling

As uncertainty sampling methods are based upon the idea of selecting the samples which the model is most uncertain of, the question is then how the uncertainty is measured. As mentioned, the common approach is to evaluate the predictions made by the model in form of the probabilistic distribution of the classes. Based upon

the evaluation function used to measure uncertainty, there are multiple uncertainty-based query methods including the following examples:

Least confidence (LC) is the most basic query method which measures uncertainty by taking the difference between the most confident prediction and absolute certainty.

$$\mathrm{LC} = 1 - \max_{c \in C} p(y = c \mid x)$$

This method only considers the highest valued model output and high informativeness scores are given to output with low confidence. The intuition being that for those samples the model is most uncertain of, training will be most rewarding. It only considers the topmost confident class and discards the other information.

Margin confidence (MC) is a method that looks at the margin difference between the two topmost confident predictions and then the difference from 100% confidence.

$$\mathrm{MC} = 1 - (\max_{c \in C} p(y = c \mid x) - \max_{d \in C/\{c\}} p(y = d \mid x))$$

In MC, the informativeness scoring both most confident and second most is considered, therefore amending some of the weaknesses of LC [22] by using more information. LC queries samples that have a small margin between classes and thus should allow the model to easier discriminate between those classes during training.

Entropy score (ES) is a query method based on the average information inherent in variables' possible outcomes as defined by Shannon [22].

$$\mathrm{ES} = \mathrm{H}[y \mid x] = -\sum_{c \in C} p(y = c \mid x) \log_2(p(y = c \mid x))$$

This method is most general and uses all information, every label class probabilities are used to measure the informativeness of the sample. Thus it benefits models with a large amount of classes [22]. If class probabilities are evenly distributed then a high informative score is given because it indicates the model being confused between classes.

Bayesian Active Learning by Disagreement (BALD) [23] is a method that evaluates uncertainty from a Bayesian statistical perspective. It is supposed to be used in conjunction with a Bayesian framework.

$$\mathrm{BALD} = \mathrm{H}[y \mid x] - \mathbf{E}[\mathrm{H}[y \mid x, \omega]]$$

Here $\mathrm{H}[y \mid x]$ is the general entropy of the output which captures the overall uncertainty of the model, and $\mathrm{H}[y \mid x, \omega]$ is the entropy of the output given model parameter ω which captures the uncertainty with a specific model setting. As in a Bayesian framework the model parameters ω can be considered as a random variable, $\mathbf{E}[\mathrm{H}[y \mid x, \omega]]$ is then the expected entropy of the output for any specific model settings. Intuitively, the objective is to find the samples which the model is most uncertain of due to there being settings of the parameters which produce predictions that are confident yet contradicting. This approach is technically equivalent

to finding the samples which cause a maximal decrease in the expected uncertainty of the parameters of a Bayesian model [23].

2.2.3 Diversity Sampling

Uncertainty sampling is sensitive to selecting outliers and data which does not represent the data set [24]. In contrast, diversity sampling alleviates such issues by selecting the subset of samples that are considered to cover the entire data set as thoroughly as possible. Depending on the approach used to generate the subset, there are numerous types of diversity sampling methods including the following [25]:

Model-based Outliers, that explores the logits and gradients of a model, grouping and assigning informativeness score to samples based on how the neurons of the model react to the unlabeled data points, which are methods such as EGL (section 2.2.9).

Representative Sampling, where each sample is scored with an informativeness score through evaluating how representative it is of the whole data set relative to the labeled set. This includes methods such as DAL (section 2.2.8), Coreset (section 2.2.7), and BatchBALD (section 2.2.6).

Cluster-based Sampling, where an unsupervised method is used to find structure and trends in the unlabeled data pool by clustering similar samples together into clusters, methods such as DBAL (section 2.2.5) and Clustering (section 2.2.4). Clustering samples can be done using various different algorithms but the most commonly used algorithm is K-Means, which is defined as; Given a predefined integer K for the number of clusters and a set N of samples, then the samples will be clustered into K clusters with each cluster having a centroid point μ [26] which minimized the inertia defined as:

$$\operatorname{argmin} \sum_{j=0}^{K} \sum_{i=0}^{N} (||x_i^j - \mu_j||^2)$$

Where x_i^j defies the data point x_i that belongs to cluster j and μ_j is the centroid of that cluster. The sum of squared error uses the Euclidean metric which is not efficient in capturing in the high dimensionality of document classification [27].

Because of the curse of dimensionality, cosine distance is the preferred metric for document similarity classification, but to alleviate the problem of Euclidean distance, L_2-normalisation can be applied to the data set before application of Euclidean distance which acts as a proxy for cosine distance. L_2-normalization unit normalizes each vector by setting the sum of squares of vector elements to 1. Thus for two vectors X and Y the Euclidean distance is denoted as $||X - Y||$, thus for the minimized inertia in cluster minimization,

$$||X - Y||^2 = (X - Y)^T (X - Y) = ||X||^2 + ||Y||^2 - 2X^T Y$$

Thus because X and Y is normalized then $||X||^2 = ||Y||^2 = 1$ equation can be simplified as

$$2(1 - cos(X, Y))$$

which shows Euclidean distance and cosine similarity relates and the equivalence of order between the metrics.

2.2.4 Clustering

A naive approach to the clustering-based diversity sampling method is to split the whole unlabeled set into clusters and then pick samples from each cluster and add them to the labeled set so that the labeled set will be balanced with regard to the defined clusters. It is expected that this method would ensure more diversity than uncertainty sampling and the quality of diverse data is dependent on a predefined number of K clusters. Choosing samples from clusters can be done randomly but also combined with an uncertainty sampling method, thus choosing the most informative sample in each cluster.

2.2.5 Diverse mini-batch Active Learning

Another diversity sampling method that utilizes clusters is Diverse mini-batch Active Learning (DBAL). The method also combines methods of diversity with uncertainty sampling, but instead of first clustering and then ranking by an informativeness score assigned by the uncertainty acquisition function as in naive clustering, DBAL first ranks whole unlabeled data set by informativeness score and in a second step then clusters [28]. The samples closest to the centroids are selected and added to the labeled set. For instance, first, the whole set is scored on informativeness by an uncertainty sampling method, then top $\beta * K$-samples are prefiltered, where β is some prefilter constant and K is the number of samples to query per iteration. The set is prefiltered for efficiency purposes since running K-means can be computationally expensive. The prefiltered set is then clustered with weighted K-means clustering into K clusters where the weights are the informativeness score assigned by the uncertainty method for each sample. The K-means clustering objective function is then modified to;

$$\text{argmin} \sum_{j=0}^{K} \sum_{i=0}^{N} s_i^j (||x_i^j - \mu_j||^2)$$

Here s_i^j represents the uncertainty informativeness of sample x_i^j and acts as a weight. Finally, a batch of K samples is chosen by picking the samples closest to each K centroids.

2.2.6 BatchBALD

BatchBALD is an extension of BALD that instead of scoring informativeness for a single sample, calculates the informativeness of the whole batch of K samples

at once, where K denotes the acquisition size. BatchBALD is then a batch-aware method that both scores a whole batch but also finds the best batch which represents the data set. It is thus a representative diversity sampling method combined with uncertainty sampling. Naively calculating informativeness of all possible batches is not feasible because there exists an exponential amount of possible subsets [19]. To at least find good enough subsets with a greedy algorithm, the submodularity property is exploited, which yields a $1 - 1/e$ approximate solution [19] and the subsets are scored as;

$$\text{batchBALD} = \text{H}[y_0, .., y_n \mid x_0, .., x_n] - \mathbf{E}[\text{H}[y_0, .., y_n \mid x_0, .., x_n, \omega]]$$

Just as the standard BALD algorithm, the first term captures the general uncertainty of the selected batch, and the second term computes the expected uncertainty for the selected batch. High informativeness is then assigned to batches that have low confidence in predictions which is represented by the disagreement of what label to assign which yields a large first term and small second term [19]. Notably, BatchBALD increases the amount of computation required to select a batch compared to the naive BALD algorithm [19].

2.2.7 Coreset

Coreset is a batch-aware active learning algorithm that tries to query a batch K set of samples, where K is a predefined acquisition size [29]. The algorithm tries to query samples by selecting a batch K subset of unlabeled set that represents the larger population of unlabeled data set by using the Coreset selection algorithm [30], thus falling into the representative diversity algorithm category and which is also a purely diversity-based method. Bounds between average Coreset loss of a subset of data and rest of unlabeled data set via a geometric representation are defined, then minimizing the bound would then find the subset of samples that represents the whole unlabeled data set and is therefore valuable as an active learning algorithm. Minimizing the bound is equivalent to solving the k-center problem [29] [31]. Then the algorithm selects K points such that the largest distance between a sample and nearest center is minimized. The problem is defined as:

$$\min_{s^1:|s^1|<K} \max_i \min_{j \in s^1 \cup s^0} D(x_i, x_j)$$

Where D is a distance function between two data points. The problem is NP-hard, but a greedy 2-OPT approximated solution is to first pick an arbitrary center, then set the next center at the farthest data point and repeat until K centers have been selected. The 2-OPT solution is formulated as:

$$\max_i \min_{j \in s^1 \cup s^0} D(x_i, x_j) \leq 2 \times \mathbf{OPT}$$

The solution can further be improved by defining a mixed-integer program and query on upper-bounds. The greedy approach can also be improved by defining an upper limit on distances, which excludes unwanted outliers [29].

Various distance metrics can be used for computing the distance between data points with the distance function D. In this project, the Cosine, Correlation, Mahalanobis and Cityblock (Manhattan) distance metrics will be considered.

2.2.8 Discriminative Active Learning

Discriminative Active Learning (DAL) is a representative diversity sampling method that tries to create a diverse labeled data set by choosing examples that are different from what samples are already in the labeled data set. To achieve different labels, a binary classifier is used to discriminate if a sample is similar to samples in labeled set or different [32].

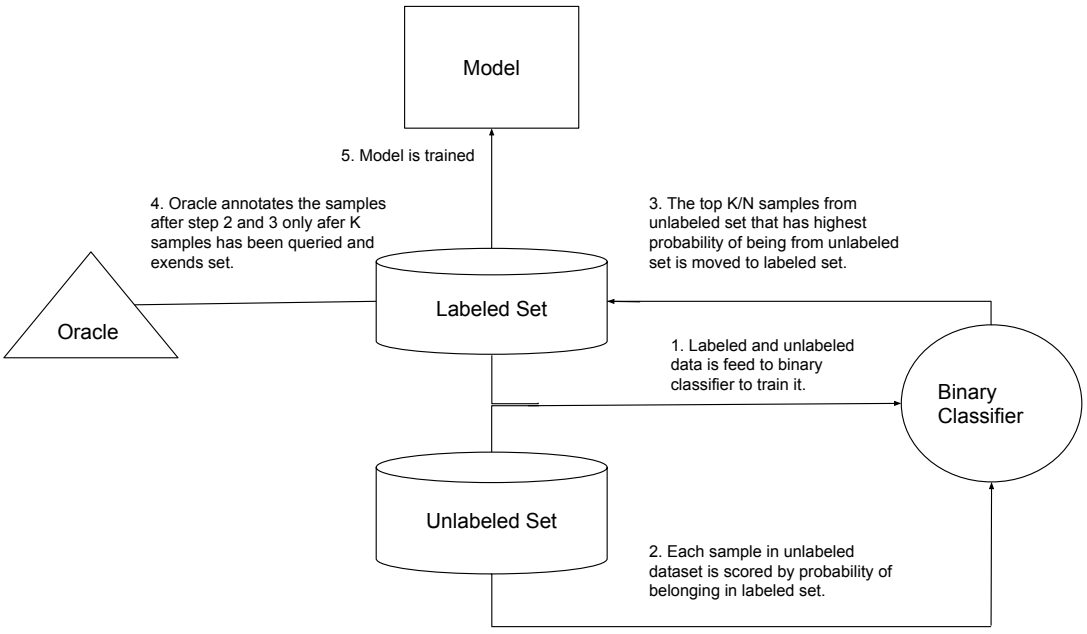

Figure 2.2: The active learning loop with DAL. Steps 1, 2, and 3 are repeated N times for each mini-query before step 4 is performed, which introduces batch awareness and allows the DAL loop to scale to large batch sizes.

The binary classifier outputs a probability of sample either belonging to the labeled data set or unlabeled set, then scoring each sample in unlabeled set by an informativeness score that corresponds to the probability of the sample either belonging to the labeled or unlabeled set. Then for a batch acquisition size K, naively selecting top K samples would only select K similar samples. To introduce batch awareness, instead of selecting the top K samples, the top sub-batches K/N samples are selected for some mini query amount N. The binary classifier is then retrained with new sub batch and the process is iterated until K samples have been selected. Be-

cause the binary classifier has to be retrained each iteration, the trade-off for batch awareness is more computational time for a higher number of sub-batches N. The active learning loop with DAL is presented at Figure 2.2, which has some differences from the more general loop in Figure 2.1. The main difference is that no expensive predictions are performed by the model which allows for a much faster active learning process.

2.2.9 Expected Gradient Length

A model-based outlier diversity sampling active learning method is gradient-based algorithms. Unseen or rare data is generally the most valuable data for a deep learning model to train because such data causes a large impact on the gradients of the model during back-propagation and the measurement is called true gradient length (TGL) [33].

Since TGL would require already labeled data to know in advance, Expected Gradient Length (EGL) is a method that calculates the highest expected gradient length by calculating the gradient length norm $||\nabla L(x_i, y_i; \theta)||$, where L represents the gradients of the loss function with respect to the parameters θ, for each possible label and then taking the sum [34]. Then if D^L represents the labeled data set, EGL is defined as;

$$\text{EGL} = \underset{x}{\textbf{argmax}} \sum_{y \in C} P(\hat{y}|x; \theta)||\nabla L(D^L + (x, \hat{y}); \theta)||$$

For efficiency, since training instances are assumed to be independent the calculation $||\nabla L(D^L + (x, \hat{y}); \theta)||$ can be simplified to only$||\nabla L((x, \hat{y}); \theta)||$. Since the algorithm has to consider every possible label for a sample, if feature space or label space is large then it will become computationally infeasible [7]. For instance, if the task is NER (discussed in section 2.3.1) then for a sample x, the number of label combinations will be exponential in size of sample x. EGL is otherwise well suited for tasks such as classification etc.

2.2.10 Query By Committee

Query By Committee (QBC) is a classical active learning method that instead of measuring uncertainty or samples by diversity, instead constructs an ensemble of models, where each model has a vote and then depending on the output of the model, which represents the vote of said model for a label, assigns informativeness score to samples. The models vary by having different hyperparameters, such as learning rates, epochs, seeds, etc. The ensemble is then trained on the same data set but the outputs of each model are potentially different which therefore generates competing hypotheses. Samples that the ensemble disagrees on the most are assigned the highest informativeness score, which is typically done by taking the difference of the max label output from the ensemble and 1 (Least Confidence) [35]. Other methods include calculating the entropy of the votes or taking the difference between topmost voted labels.

QBC is a suitable active learning method if the ensemble is large but does therefore require an inexpensive underlying model. Deep learning models generally consist of large neural networks and are therefore not suitable to be used in QBC but instead, cheap models such as Trees, SVM, etc are more feasible for a QBC solution.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a subfield and discipline within the domain of Artificial Intelligence (AI). The goal is to process and manipulate human natural language to solve various tasks [36], such as information retrieval, information extraction, text translation, text summarization, and speech recognition.

2.3.1 Text Representation

Generally, before the document texts can be processed and used for computations in NLP tasks, it has to be numerically represented for a machine to interpret it. There are multiple methods to achieve text to numeric representation. One method is to generate one-hot-encoded vectors for each word, which is a fast and computationally inexpensive operation. This can be done by using Bag-Of-Words (BoW) or TF-IDF algorithms. A flaw of assigning each word in a corpus a one-hot-encoded vector is that the representation will be large and therefore memory inefficient and a case of the curse of dimensionality [27]. Another more important flaw is that context and semantic meaning of words is lost with one-hot encoding, as each word is simply represented by a numeric one, and the rest of the vector is filled with numeric zeroes which hold no semantic value. Ergo, clustering a collection of words such as documents or sentences with one-hot-encoding would generally only result in clusters that are generated based upon similarities in word frequencies.

One way to address the flaws of one-hot-encoding is to use more complex word embedding methods. The meaning of words is numerically encoded in vectors such that semantically similar words are closer in the vector space [37]. Documents in the corpus can be mapped to numeric vectors by various methods such as neural networks, dimensionality reduction on word co-occurrence matrices, probabilistic models, etc but the most popular and highest performing methods to date are neural networks. Therefore, typically for a neural network, the drawbacks are requiring a lot of data to be trained on and being time exhaustive.

Two popular word vector generating methods with underlying neural networks are the Skip-Gram model and Continuous Bag of Words Model (CBOW) [38]. Training is generally performed in an unsupervised method, the models training are done through predicting word with regards to the neighboring words of a sentence. The number of neighbors to consider in prediction is determined by a window size parameter, e.g for a window size of 2 then both words that occurred before and two words that occurred after some word is being considered by the algorithm.

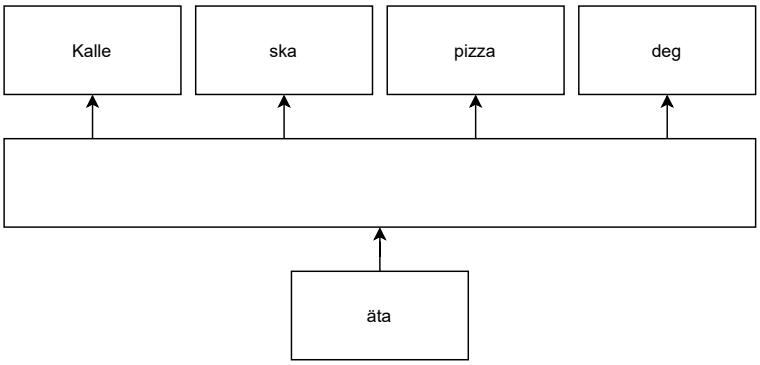

Figure 2.3: Simplified view of the Skip-Gram architecture with the example sentence "Kalle ska äta pizza deg" where "äta" is the main word.

Skip-Gram models try to predict the neighbors of some main word which is the main word context. The architecture has an input size of 1xV where V is the total number of words in the corpus, e.g a one-hot-encoded word is the input. Then one hidden layer with size VxE where E is the size of the word embedding, i.e the dimension. Finally, the output of the hidden layer is a 1xE which is sent to the final output layer, that through a Softmax activation function generates probabilities of size 1xV, which represents a predicted one-hot-encoded word and one prediction for each neighboring context word is feed to the network. Figure 2.3 shows an example of Skip-Gram where the main word "äta" is fed into the model. The window size is 2, then the output is two words occurring before the main word, "Kalle" and "ska", then two words after the main word which are "pizz" and "deg".

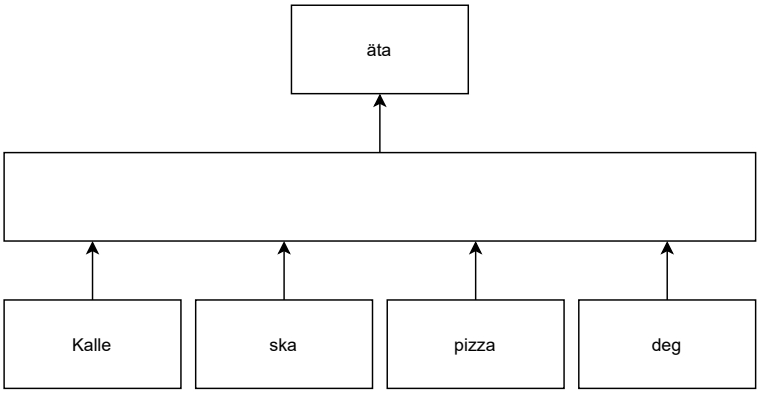

Figure 2.4: Simplified view of the CBOW architecture with the example sentence "Kalle ska äta pizza deg" where "äta" is the main word.

The CBOW model has the same dimension of hidden and output layer but the goal is different. On the contrary of predicting neighboring words to get the context of a word, CBOW tries to predict the main word based on neighbors and context, and therefore the difference in architecture lies in the input layer. The input is instead B 1xV, where B is each neighbor context word and the resulting output is the singular main word, which makes CBOW a faster model to train. Figure 2.4 shows an example of CBOW where the window size is set to 2. The two words occurring before the main word "Kalle" and "ska", then the two words occurring

after the main word, "pizza" and "deg" is fed into the CBOW model to produce the main word output "äta".

2.3.2 Named Entity Recognition

Named entity recognition (NER) is an information extraction task within the field of NLP problems. The goal is to detect and classify entities related to a certain subject within a text, most often unstructured or semi-structured, into predefined categories [39] such as organizations and locations. Extracting information from unstructured data is mostly known for its applications in the field of biochemistry and medicine.

The four main common methods of NER solutions are:

- **Rule-based approach:** Works using hand-crafted rules, therefore limited to rules but does not require labeled data, usually a gazetteer method [40].
- **Unsupervised learning approach:** Unsupervised methods which do not require labeled data, such as clustering.
- **Feature-based approach:** Uses traditional machine learning methods to find the feature patterns and structures behind the entities, such as decision trees and support vector machines. Requires careful feature engineering [41].
- **Deep-learning approach:** Requires a lot of labeled data, usually RNN, LSTM models, or transformer models.

In recent researches, this is usually done by using a transformer-based ML model, such as BERT which has shown considerably better performance than the older more common methods of LSTM or gazetteer methods [42].

2.3.2.1 Tagging Schemes

There are multiple different labeling schemes for NER token annotations. The different labeling schemes affect the result and which schema is most appropriate depends mostly on corpus text language [43] [44] and domain. Some common schemes [45] are:

- **IO:** Two tags exist in this scheme. Either a token has the inside tag (I-), or the outside tag (O).
- **IOB:** Also referred to as BIO scheme, it consists of three tags, inside tag (I-), outside tag (O), but also extends the more simple IO scheme with a beginning tag (B-).
- **IOE:** Similar to IOB scheme, it also consist of three tags, an inside tag (I-), outside tag (O), and an end tag (E-) instead of the beginning tag.
- **IOBES:** Has five tags, inside tag (I-), outside tag (O), beginning tag (B-), and the accompanying ending tag (E-) but also the single tag (S-).

Table 2.1 presents the usage of the different tagging schemes on an example sentence. The different schemes can affect the result of the NER model a lot [43] because the

structure of languages varies. Having more tags also increase the complexity of the NER model which also affects the result.

Tokens	IO	IOB	IOE	IOBES
Kalle	I-PRS	B-PRS	I-PRS	B-PRS
Arvidsson	I-PRS	I-PRS	E-PRS	E-PRS
wants	O	O	O	O
to	O	O	O	O
watch	O	O	O	O
the	O	O	O	O
movie	O	O	O	O
Justice	I-WORK	B-WORK	I-WORK	B-WORK
League	I-WORK	I-WORK	E-WORK	E-WORK
or	O	O	O	O
Naruto	I-WORK	B-WORK	E-WORK	S-WORK
.	O	O	O	O

Table 2.1: Example of different schemes are applied to the sentence "Kalle Arvidsson wants to watch the movie Justice League or Naruto ."

Using the IO schema allows for a low number of $n + 1$ tags where n denotes the number of entities. A limitation is that it can not differentiate between consecutive entities. IOB schemes have $2n + 1$ tags, allowing for the identification of consecutive entities but double the number of tag classes a NER model has to learn. IOBES increases to $3n+1$. The more complex schemes as IOBES captures more information but if the data set is not balanced in the $3n + 1$ number tags then the NER models performance will be negatively influenced.

2.4 Transfer Learning

In machine learning, transfer learning is the practice of using an already trained model and repurposing it for another problem. This approach has three main advantages. The first is that the learned parameters in a trained model are adjusted from some task to fit for another related task, which may immensely reduce the amount of training data required for a new related task. This is because the model is not trained from scratch but the patterns learned from solving a related task are reused. The second advantage is that the performance of the model tends to increase. The third is that since the model is not trained from scratch, training time to achieve a certain performance may also be immensely reduced [46]. The limiting disadvantage of transfer is negative transfer, which happens when the model is trained with samples that may affect the performance negatively. Such samples are called negative samples [47]. Examples of negative samples are samples that do not relate to the original domain of the model.

Transfer learning is a useful method for NLP tasks since training a well-functioning model requires a huge amount of labeled data and computational power which both are hard to come by since domain experts are required and computational power requires expensive resources.

Depending on the source domain and source task of a pre-trained model and target of domain with another target task, different transfer learning scenarios can arise. The marginal probability distribution of source and targets domain are different which is the general standard idea of transfer learning, also known as domain adaption [48], e.g data sets contain different topics. A special case of domain adaption is cross-lingual adaption [49] in NLP, which corresponds in transfer learning to when the feature space between target and source is different, e.g different languages in data set. Another scenario is that conditional probability could be different between source and target tasks which arise when the distribution of classes between data sets are different. Another instance is the difference in label space between the source and target task since the output is the difference then this usually requires that either output layer of the pre-trained model be adjusted or replaced with a new layer suited for the target task.

There are different ways to fine-tune a pre-trained model, and which method is most appropriate depends on the scenarios that can arise and the severity of differences between both domain and the task of source and target but also available time and resources. If the difference between source and target is small, then training time can be vastly reduced by freezing layers in the pre-trained model. Freezing layers sets the weights of the neurons in a frozen layer to constant, which means that backpropagation calculations are skipped in frozen layers that allows for a huge decrease in computational time. However, since neurons in the frozen layers will not be updated, freezing layers make the model less adaptable to new data. Thus, freezing layers with a sufficiently large difference between source and target may result in reduced accuracy.

2.5 Transformer

A transformer is a deep learning model that is designed to handle sequential data and is frequently used in the field of NLP. Since the introduction of the model in 2017, the transformer has become the model of choice for NLP and replaced Recurrent neural networks (RNN) and long short-term memory (LSTM) methods. The major difference between transformers and the older RNN and LSTM methods is how the sequential data is processed: a transformer processes the entire sequence at once as a set of inputs, whereas in RNN and LSTM each input is processed sequentially as presented. This allows the transformer to learn dependencies between inputs with high positional differences in the sequence more effectively [1]. This improvement combined with training with larger data sets leads to the success of complex pre-trained systems such as Bidirectional Encoder Representations from Transformers (BERT) [2], which has been trained on a large language data set and

can be fine-tuned to serve more specific tasks.

The essential mechanism transformer relies upon is called attention. The goal of attention is to obtain a representation that determines the importance of each item in the set of input. This is done by using three different vectors called query, keys, and values. Usually, these vectors are acquired through multiplying an input with corresponding learnable weight matrix W_Q, W_K, and W_V. The query vector and keys vector are used to compute a set of weights which represent the importance of each value, and the output is generated by computing the weighted sum using said set of weights and the value vector. The process can be described in the following formula:

$$Attention(Q, K, V) = softmax(f(Q, K) * V)$$

where Q represents the query, K represents the keys, V represents the values, and f is the function used to compute the weights using Q and K. The case where all vectors used by attention are obtained from the same input is called self-attention.

Transformer uses the scaled dot-product attention:

$$f(Q, K) = \frac{QK^T}{\sqrt{d_k}}$$

as it provides a computational advantage over additive attention by using optimized matrix multiplication algorithms. The scaling factor $\frac{1}{\sqrt{d_k}}$ which is based upon the dimension of query and keys d_k is used to counteract vanishing gradients caused by dot products of high magnitude. Additionally, the transformer uses a technique called multi-headed attention which employs multiple weighted scale-dot attention in conjunction to obtain and make use of information from multiple representation subspaces. This is done through concatenating the output of h parallel scale-dot attention layers and multiplying the result with a weight matrix W_O.

The transformer model follows an encoder-decoder architecture. The encoder blocks consist of a self-attention layer followed by a position-wise fully connected layer. These layers are connected through a residual connection with layer normalization. The decoder blocks consist of a masked self-attention layer followed by an attention layer with query and keys obtained from the encoder, and a final position-wise fully connected layer. All three layers are connected through residual connection with layer normalization, same as the encoder blocks [1]. The architecture presented in the original papers is shown in Figure 2.5.

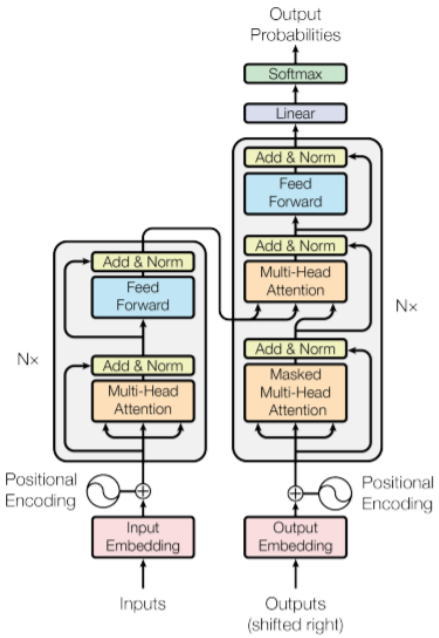

Figure 2.5: The transformer architecture presented in Vaswani et at [1]. Note that positional encoding is applied to the input, as the attention-based model is otherwise incapable of distinguishing sequential relationship.

2.5.1 BERT

BERT is a pre-trained transformer-based machine learning model designed with the purpose to be easily fine-tuned to a variety of NLP tasks, presented by Devlin et al [2]. The idea is to obtain a word embedding through pre-training the model with unsupervised tasks on a large corpus, and use the part of the model which represents the word embedding for transfer learning. It was mentioned by the authors in the original paper that this approach was proven in previous researches to provide significant advantages over training an embedding from scratch [2].

As can be observed in Figure 2.6, the structure of BERT consists of a stack of the aforementioned encoder blocks. While the approach of using transformer-based models to acquire linguistic embedding is not unheard of, what differentiates BERT from similar approaches is the tasks used for its unsupervised pre-training, which are masked language modeling (MLM) and next sentence prediction. In MLM, a certain percentage of the input is replaced with a mask token, and the task is to predict the original tokens which were replaced. This fill-in-blank approach allows for the model to obtain a bi-directional representation, as both the context before and after the masked tokens will be used for the prediction of said tokens. As for next sentence prediction, a number of pairs of sentences (A, B) are picked from the corpus. The sentence pairs are then labeled binarily for their sequential relation, that is whether if B directly follows A. The task is to correctly predict the binary label for each pair of sentences. The goal of pre-training the model with next sentence prediction is to train the model for the relationship between sentences, which

is crucial for question answering and natural language inference tasks [2].

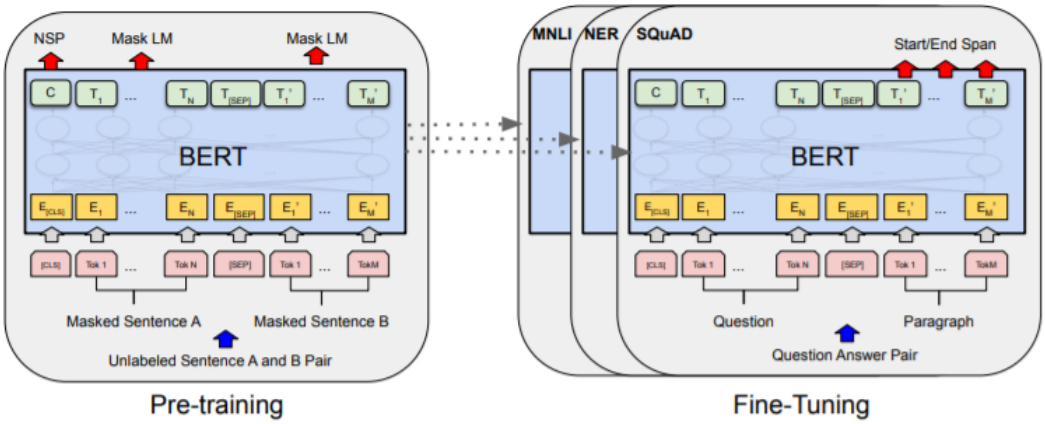

Figure 2.6: The intended training procedure of BERT according to Devlin et at [2]. Fine-tuning for specific task can be done through adapting the output layer and the input formulation according to the intended task.

In order to make raw text processable by BERT, the text needs to be tokenized into a sequence of numerical representations. The tokenization method used for BERT is Wordpiece, introduced by Wu et al [50]. The Wordpiece tokenization is based upon a word-level vocabulary, yet if a word is not present from the vocabulary then the word is recursively split into sub-words until the missing word consists of words present in the vocabulary. In the worst case, the word will be split into individual characters, which are guaranteed to exist in the vocabulary. The individual sub-words are then tokenized accordingly. This approach allows for word-level representations with a limited vocabulary size while also ensuring that no out-of-vocabulary words exist.

3
Method

This chapter explains the basis of how the experiments were conducted during the thesis. The experiment setup, which includes the hardware used, the hyperparameters of the model, and the implementation details of the active learning algorithms are hereby described. A description of the data set and the preprocessing steps are also provided. The metrics and explanation of how they are used to evaluate the performance of the active learning algorithms are presented, then finally the model's specific configurations for the NER task are explained.

3.1 Hardware

The experiments were conducted on the Bayes and Titan servers. Bayes was mainly used, which is a shared resource for master students supervised by faculty in the DS&AI department and classes in the ADS GU program. There are three computers in total available on the Bayes and one on the Titan server. The computers hardware are described on Table 3.1.

	Processor	GPU	RAM (GB)
Controller	Intel Xeon Gold 6126	Tesla V100-PCIE-16GB	768
Shannon	AMD EPYC 7451	Tesla V100-PCIE-32GB	512
Markov	AMD EPYC 7451	Tesla V100-PCIE-32GB	512
Titan	Intel i7-5930K	GeForce GTX TITAN X-12GB	128

Table 3.1: Two shared computer resource servers offered by Chalmers, Titan and Bayes, where Bayes includes Controller, Shannon and Markov and Titan is the singular computer in Titan server.

There is multiple available hardware on the server, but since it is a shared resource, the available hardware at the moment was used to conduct the experiment because of the shared resource policy. Experiments are only performed on one CPU core and are limited to not use any multiprocessing or multi-threading to allow for the most consistent and fair comparison between active learning algorithms.

3.2 Software Framework

A Pool-Based Sampling active learning framework for easy and quick experiment testing between query algorithms was developed. The framework is built as a tool

that supports manual annotation of general NLP tasks but also automatic annotation if labels are already provided for the data points (the experiment cases).

Algorithm 1: Software Framework General Loop

initialization;
$M \leftarrow$ pre-trained transformer model, initial state is saved in a reset function
$Q \leftarrow$ Chosen active learning query function
$L \leftarrow$ labeled data set
$U \leftarrow$ unlabeled data set
$T \leftarrow$ test data set
$K \leftarrow$ Acquisition size
$N \leftarrow$ Number of iterations to run
for $0...N$ **do**
  $L, U = Q(L, U, K, M)$ // K data points is moved to labeled set.
  M.reset() // Model is reset to initial state.
  $M = M$.train(L) // Retrain data with extended labeled set.
  M.eval(T) // Evaluate on test set and log the results.
end

The experiment's main loop is described in Algorithm 1, and it can be observed that the training is done cumulatively. The model is retrained from scratch at each iteration. For a neural network, incremental training is suitable if only one epoch is run during training [17] but this is generally not practical in neural networks and therefore cumulative training was chosen for the framework. The available supported algorithms in the framework, which are also thoroughly described in the theory section, are presented at Table 3.2.

Baseline	Uncertainty	Diversity	Combined (DBAL)	Combined (Clustering)
Random	BALD	Coreset-Cosine	D-BALD	C-Random
	Least	Coreset-MHS	D-Least	C-BALD
	Margin	Coreset-CRL	D-Margin	C-Least
	Entropy	Coreset-CB	D-Entropy	C-Margin
		DAL		C-Entropy

Table 3.2: The algorithms that are supported by the framework. MHS denotes Mahalanobis, CRL denotes Correlation and CB denotes Cityblock (Manhattan).

The uncertainty sampling methods measure the output uncertainty of the model with Monte Carlo dropouts in all cases. The framework then supports a total of 17 active learning algorithms that will be evaluated and compared to each other.

3.2.1 Hyperparameters

The general settings and model hyperparameters for the experiments were:

- **Limit = 1500**, Since the server is shared among multiple users, there is restrictions in CPU time and memory usage. Therefore a random amount of data is selected from data set to be used in test.
- **Seed = 100/200/300**, Different seeds for different test cases, allows for a unique data set for each test case, but same when algorithms are compared to each other.
- **Iterations = 30**, Number of query iterations. It is limited because of limited hardware access.
- **Batch size = 16**, Batch size, was determined by BERT author recommendation.
- **Acquisition size = 50 or 16** How many samples chosen to be queried each iteration.
- **Learn Rate** = $2e^{-5}$ The models learning rate, chosen by recommendation of BERT author.
- **Epochs = 5** Number of training epochs, chosen from BERT author recommendation.
- **Number of Clusters = 5** Number of clusters for diversity sampling.
- **Monte Carlo Drop Iterations = 10** Authors recommend the Monte Carlo Dropouts to be in the range of 30-100 [20] but it is deliberately chosen to be vastly lower because of hardware limitations.

Identical parameters and settings were used across all experiments because optimizing hyperparameter for some experiment setup may introduce bias for that experiment setup. Therefore no effort in parameter optimization has been made and only recommended hyperparameters from authors [2] was chosen. Other parameters were chosen with CPU time and memory in mind so that test cases could be finished in a reasonable time window without clogging shared server resources.

The DAL-specific parameters are the number of mini-batch queries and the architecture of the binary classifier. The number of mini-batches, that introduce the degree of batch awareness, is chosen to be 4 or 10 depending on the acquisition size being either 16 or 50. The neural network architecture setup for binary classification is chosen to be a neural network with 3 layers, a width of 256 neurons with ReLU activation functions. The binary classifier used here is intended to be identical to the one used in the original paper [32].

Since the dimensions of word representations are large (Section 3.3.2), the distance metric for Coreset is set to cosine. As Coreset authors specified, the distance metric is vital for the performance but the defaulted Euclidean distance [29] would be affected by the curse of dimensionality [27].

Finally, the prefilter coefficient constant β, which is a DBAL parameter, is chosen to be 10 by recommendation of authors [28].

3.2.2 Sentence Embedding

Texts have to be numerically represented, and since semantic information is important in most NLP tasks, a sentence vector embedding method is most appropriate. Since training a model from scratch is resource and time-intensive, a pre-trained model is chosen to be used.

For the diversity algorithms, naive clustering, DBAL, Coreset, and DAL, a pre-trained CBOW model is used to map the sentences to numeric sentence vector embedding.
The pre-trained word vector[1] of the CBOW model was trained on Common Crawl[2] and Swedish Wikipedia word entries [51]. The hyperparameters of the model are;

- **Position-Weights = True**,
- **Dimensions = 300**,
- **n-grams size = 5**,
- **window size = 5**,
- **negatives = 10**

The pretrained CBOW Fasttext model is then trained to generate word embeddings, but can also generate sentence embeddings by averaging the word embedding in a sentence [52].

	Word Vector	Normalized
Kalle	[-0.068, 0.230, 0.044]	[-0.280, 0.943, 0.178]
ska	[-0.612, 0.133, 0.018]	[-0.977, 0.212, 0.029]
äta	[-0.376, 0.150, -0.367]	[-0.687, 0.275, -0.672]
Sentence Vector		[-0.648, 0.477, -0.155]

Table 3.3: The mapping of a sentence to numeric sentence vector embedding.

To get the resulting sentence embedding $s(\text{"}x_1..x_n\text{"})$ where $\text{"}x_1..x_n\text{"}$ is a sentence (sequence of words) e.g "Kalle ska äta". Word embeddings are first generated for each word, which is shown in Table 3.3. The word embeddings normalization step is performed by dividing the word embedding by its ℓ_2-normalized value.

$$s(x_1...x_n) = \frac{1}{N}\sum_{i=1}^{N}\frac{w(x_i)}{\ell_2(w(x_i))}$$

Where $w(x)$ denotes the learned sentence's word x to word vector mapping, $\ell_2(w(x) = \sqrt{(\sum x^2)}$ denotes the normalization of the word vector and the sentence length is denoted by N.

[1] https://fasttext.cc/docs/en/crawl-vectors.html
[2] https://commoncrawl.org/

3.3 Data Set

The testing framework pipeline described in Section 3.1 requires the data set input to be a Pandas data frame with three mandatory columns as shown in table 3.4. A column header should be sentence_id that encodes every sentence with a unique integer. Another column header needs to be named 'words' that is a string with whitespace-separated tokens of a sentence. Lastly, there needs to be a column named 'labels' that is a python list with labels, each label in list mapping to the same rows words based on list index.

sentence_id	words	labels
68	Av MICHAEL WINIARSKI .	[O,B-PRS,I-PRS,O]
410	Av DN:s speciella medarbetare .	[O,B-ORG,O,O,O]

Table 3.4: Example of how two sentences in the data frame looks like.

3.3.1 SUC 3.0

SUC 3.0 was released in 2012 and is a data set that consists of a collection of Swedish texts from 1990. An effort has been made to keep the corpus balanced with different types of styles and texts. There are a total of 1,166,593 tokens in 74,245 sentences. The corpus contains 9 different entity category tags for the tokens, which are person, place, institution, work, animal, product, myth, and other. Explanations for the entity tags can be found in Table 3.5.

Entity	Mapping	Frequency	Explanation
person	PRS	22062	Person name
myth	PRS	301	Name of mythological entities
place	LOC	9517	Locations
inst	ORG	8464	Organizations
work	MISC	4402	Creations (eg books, paintings)
product	MISC	945	Products
event	MISC	368	Event
other	O	1119285	Not an entity

Table 3.5: The entities provided by the SUC 3.0 data set. Second column corresponds to mapping to more common and easier tags to work with.

The data set is contained in an XML file, a sentence begins with a <sentence> tag and each word is wrapped in a <w> tag. Words that are entities are wrapped around a <name> tag, but also sometimes a <ne> tag. Both tags denote named entities but <ne> tags are automatically tagged with the annotation tool Sparv [53] while <name> tags are partially manually tagged and also automatically derived, but

most importantly, <named> tags are always checked manually to be correct [54]. Table 3.6 shows an example of how a <name> tag entry can look like. The other tags, such as <ana> that stands for analysis, <ps> for part-of-speech, <m> for morphosyntactic information and <b> for base form [54], are not used in this study and therefore ignored.

<name type=person>
<w n=494>drottning<ana><ps>NN<m>UTR SIN IND<b>drottning</w>
<w n=495>Sofia<ana><ps>PM<m>NOM<b>Sofia</w>
</name>

Table 3.6: Example of how a named tag in SUC 3.0 looks like.

Some preprocessing is applied to the SUC 3.0 data set. Firstly, since the experiment framework required formatting as Table 3.4 shows, it is preprocessed to the appropriate format and the entities need to be assigned a tagging scheme. To follow the practice of previous and generally most NER tasks [55] [56] [57], the BIO scheme is chosen.

Also with the motivation of following previous and generally most NER tasks practice, the predefined named categories in SUC were mapped to [PRS,LOC,ORG,MISC,O], as can be seen in table 3.5. This also allows for better overall performance because fewer labels give less complexity and some categories such as events with only 368 instances would be hard to gain any representation because of the imbalance.

Entity	Frequency
B-PRS	15373
I-PRS	6855
B-LOC	8764
I-LOC	665
B-ORG	6445
I-ORG	1920
B-MISC	3892
I-MISC	3034
O	1111747

Table 3.7: Frequency of tokens in the SUC 3.0 data set after the preproccesing steps.

Figure 3.1 shows the SUC 3.0 data sets distribution of sentence lengths up to length 50 where majority of the sentence lengths were concentrated. Furthermore, 7898 sentences in the data set contained less than 10 characters and were removed to ensure the data sets integrity. The removed sentences such as "§ 6" were deemed to contain no valuable information and bloats the data set. Thus the table 3.7 represents the final preprocessed data set token frequency used for the NER experiments.

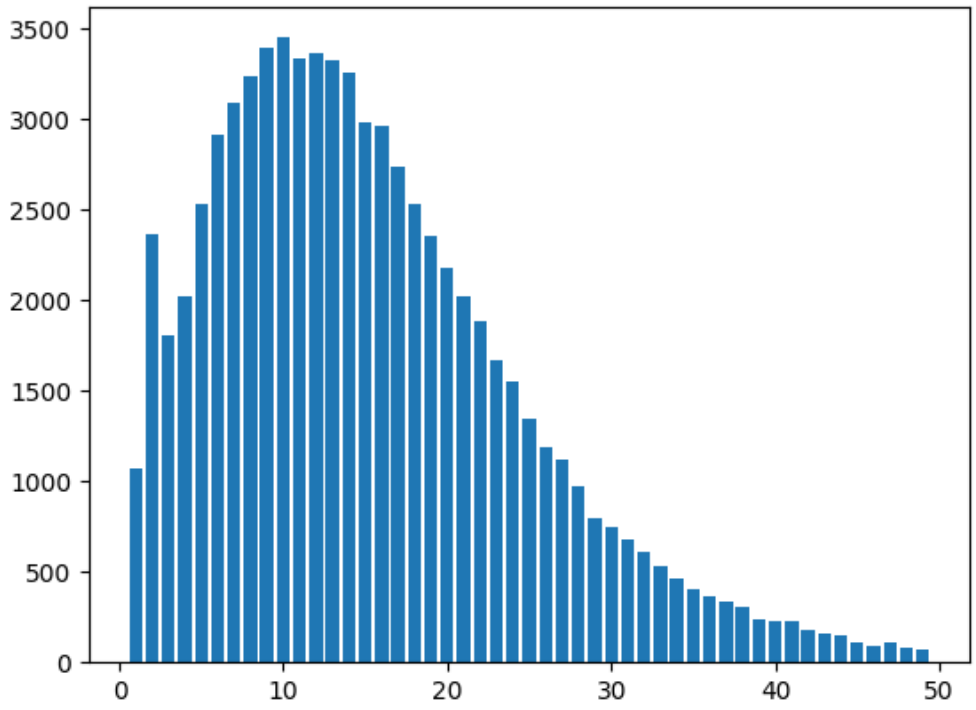

Figure 3.1: The frequency of sentence lengths in the SUC 3.0 data set. Starting at sentence length one which represents a one word sentence.

3.4 Metrics

The generally most common evaluation and scoring metrics for NER are precision, recall, and F1-score [55] [56] [57], where F1-score is regarded as the final benchmark performance metric of models. The same scoring scheme is chosen during this project to follow the same scoring practice as previous NLP classification tasks have done, for both classification and NER tasks but our project will also consider and give weight to the processing time of algorithms with regards to hardware.

Precision, recall and F1-score are based on the concepts of:

- **True Positive (TP)** Correctly predict x to belong to class c.
- **True Negative (TN)** Correctly predict that x does not belong to any class c.
- **False Positive (FP)** Predict that x should be assigned to a class but assign to wrong class.
- **False Negative (FN)** Incorrectly predict that x does not belong to class c

Thus there are three ways to incorrectly label an entity. First to notice that there is an entity but assign it the wrong label, secondly to notice that there is an entity

but incorrectly assign the boundary, and thirdly to make both mistakes, give the wrong label and incorrectly assign the boundary.

Sentence:	Predicted	Actual
Kalle	B-MISC	B-PRS
ska	I-MISC	O
äta	O	O
på	B-LOC	O
Condeco	B-ORG	B-ORG

Table 3.8: An example of predicted tokens on the second column, with the BIO-scheme on the sentence "Kalle ska äta på Condeco" on the first column and, correct tokens on the third column.

Computing TP,TN,FP and FN on table 3.8 would yield $TP = 1$ (Condeco), $TN = 1$ (äta), $FP = 2$ (ska,på), $FN = 1$, (Kalle).

3.4.1 Accuracy

The most intuitive method of model evaluation would be accuracy which is a simple but naive metric because it only considers the correctly predicted data samples over the whole data set.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Calculating accuracy for example in table 3.8 would yield $\frac{1+1}{1+1+2+1} \approx 0.40$. The drawback and reason for accuracy metric not being used more often in previous works are because of the accuracy paradox [58], which means that in imbalanced data sets, which often is the case in NLP tasks, then naively only predicting the dominant class would misleadingly give a high accuracy score.

3.4.2 Precision

Precision only considers samples predicted as an entity and gives a measure of how correct the positives are.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

This metric is important for cases where false positives has a high cost and are unwanted. Precision applied to example in Table 3.8 yields $\frac{1}{1+2} \approx 0.33$.

3.4.3 Recall

Recall measures the real positive cases that are correctly predicted positive.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Applying it to Table 3.8 yields $\frac{1}{1+1} = 0.5$. This metric is more important for cases where false negatives is the main focus of task at hand.

3.4.4 F1-Score

F1-score is a commonly used metric of performance for various classification tasks, including token-wise classification such as NER. By default, the F1-score is used to evaluate binary classification tasks. It is calculated by taking the harmonic mean of the precision and recall, where precision is the percentage of true positives within the samples flagged as positive, and recall is the percentage of true positives within all actual positive samples. The formula is as follows:

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

In order to use F1-score for evaluating classification tasks with multiple labels or classes, the classification of each label/class is considered as a binary classification task and the F1-score is calculated for each label/class independently. The resulting individual F1-scores are then averaged to provide an overall representation of the performance. Applying it for the example in Table 3.8 yields $2 * \frac{0.33*0.5}{0.33+0.5} \approx 0.40$.

3.4.5 Time

Time is also measured and an important factor in active learning because the goal for active learning is to reduce the time and resources required for training a model. Therefore if the time required for active learning to process is high then it defeats the whole purpose of time and resource-saving. The time is measured both for training time and query processing time of the active learning algorithms. To keep the time measurement consistent and fair across experiments, time will only be considered for the Bayes controller computer.

3.5 Learning Model

Since training and creating a well-functioning transformer model from scratch requires a combination of lengthy training time, high-end hardware, and an enormous amount of training data, it was decided that a pre-trained model shall be used. A criterion for choosing a pre-trained model suited for this project is that it needs to perform well with Swedish texts, thus naturally a model pre-trained with Swedish texts is preferred.

There exist multiple pre-trained models that support and have been trained with Swedish texts. Some of the considered pre-trained models are Kungliga Biblitoeket (KB-Bert) which has a standard 12-layer, 768-hidden, 12-heads, 110M parameters model. Arbetsförmedligen (AF-AI) has a 12-layer, 768-hidden, 12-heads, 110M parameters model but also a large 24-layer, 1024-hidden, 16-heads, 340M parameters model. A more complex model should have better performance with computational

time trade-off according to BERT authors [2]. Also, Google has published a multilingual model(M-Bert) that supports 104 languages including Swedish and has a standard 12-layer, 768-hidden, 12-heads, 110M parameters model architecture.

Models	F1-score	Time
AF-AI-BERT	0.566	10:06
AF-AI-Large-BERT	0.571	17:58
M-BERT	***0.640***	10:37
KB-BERT	***0.640***	***09:30***

Table 3.9: NER experiments performed with the different pretrained BERT models that support the Swedish language. Time is in format (minutes:seconds).

The highest priority is for computational time to be low and performance to be adequate. Speed is a crucial aspect as multiple experiments need to be conducted in a limited time window. As such, the choice of the model is decided through experimentation concerning Swedish NER, where all models were trained and evaluated using the same training set and evaluation set which were randomly sampled from a subset of 1,500 sentences sampled from SUC 3.0 using seed 100. The size of the training set and evaluation set were 480 and 2,000, respectively. Training parameters used were the batch size of 16, and the learning rate of $2e^{-5}$. With consideration of computational processing time and performance from test results shown in table 3.9, the KB-Bert model is chosen to be used for this research. The base BERT models, AF-AI-BERT, M-BERT and KB-BERT have the same architecture and parameters. The reasons for a difference in computation time between the base models is because the experiments in this study are performed on a shared resource. Therefore, the time is approximated and mainly considered to highlight the difference between the base model and the BERT-large model.

KB-BERT, which is developed by the National Library of Sweden, used a corpus of various types of modern Swedish texts from 1940 to 2019, including newspaper articles, official government reports, legal e-deposits, social media comments, and Wikipedia articles. The data set used for its pre-training consists of 3,497 million words and 260 million sentences. The tokenization method used is SentencePiece with a vocabulary size of circa 50,000 [59].

3.5.1 BERT for NER

When BERT is purposed for a NER task, the transformer acts as an encoder for a linear classifier that is added on top of the BERT model [60]. The linear classifier gets the input from the transformer and the number of output corresponds to the number of available labels.

In our case with KB-Bert, the complete architecture is the standard base 12 layer BERT model [2] [59] with a classifier on top. The classifier is a simple fully connected

neural network with 768 inputs, no hidden layer, and 9 outputs followed by a Softmax layer. The input size is chosen to accommodate the size of the word-wise output of BERT, while the output size is the amount of assignable labels. The assignable labels are: ['O', 'B-ORG', 'B-PRS', 'I-PRS', 'B-LOC', 'B-MISC', 'I-MISC', 'I-ORG', 'I-LOC'].

A sentence is first processed by the BERT model which returns the word-wise feature vectors. Each individual feature vector in the sequence is then processed by the aforementioned classifier, yielding the predicted probabilities of the labels for each word in the sentence, which is the final output of the model. The label with the highest predicted probability is taken as the final prediction for the corresponding word.

Sentence:	Kalle	Arvidsson	reser	till	Kina	Informativeness
PRS	0.12	0.10	0.07	0.08	0.09	
LOC	0.22	0.21	0.25	0.24	0.28	
ORG	0.15	0.28	0.14	0.17	0.13	
MISC	0.22	0,11	0.32	0.27	0.19	
O	0.30	0.30	0.22	0.24	0.31	
Least	0.7	0.7	0.68	0.73	0.69	0.7

Table 3.10: An example of how informativeness score is calculated for NER. The sentence "Kalle Arvidsson reser till Kina" is fed into the pretrained KB-BERT model (before fine-tuning) with the predefined labels PRS,LOC,ORG,MISC and O (no tagging scheme). Each word in the sentence gains probabilities for each label. Applying the Least Confidence query method for each word yields the informativeness score shown in the bottom row. The far right column shows the resulting informativeness score for whole sentence.

The query methods assign and calculate informativeness score only based on the generated probabilities of the model, which in the case of NER means the informativeness score obtained is for individual words. In order to gain the informativeness score for an entire sentence, which is necessary for the case of NER [39], the individual informativeness score for each word is calculated and then averaged over the number of words in the sentence. This is the same approach used for BALD by Shen et al in their experiments [10]. An example of the process can be seen in table 3.10 where the final total informativeness score of the sentence is then calculated as $(0.7 + 0.7 + 0.68 + 0.73 + 0.69)/5 = 0.7$.

4
Results

In this chapter, an overview of the performance of each query method is provided through plots and tables that describe the averaged F1-score across three different seeds against the number of iterations. The different seeds, which are 100, 200, and 300, decide which data points the subsets unlabeled pool and validation set contain, but also allow for the same subsets for each query method experiment. Furthermore, the region of interest of the plots is highlighted with tables to more clearly distinguish the differences in the average F1-score of each query method.

For the sake of simplicity, only the result of the best performing variant of Coreset, Clustered uncertainty, and DBAL will be considered. As seen from Appendix C, the Entropy variant of both Clustered uncertainty and DBAL are the best performing amongst their peers, and will be used to represent the respective algorithm in the following results. As there is no clear best performing Coreset variant, the simplest variant which uses cosine distance will be used to represent said algorithm.

The experimental results of the individual query methods for each of the three different seeds are presented in Appendix A and B, to present the consistency of their performance across different subsets of the data.

To emphasize the impact of different acquisition sizes, the results for uncertainty and diversity sampling algorithms are presented for acquisition sizes 16 and 50 respectively in sections 4.1 and 4.2. The comparison of performance depending on acquisition size is presented in section 4.3.

4.1 Acquisition size 16

The following results are obtained from experiments performed in 30 iterations with an acquisition size of 16. Each experiment results in a total of 480 data points being acquired from an unlabeled pool of size 1500.

4.1.1 Uncertainty Sampling

Figure 4.1 and Table 4.1 describe the performance of the uncertainty based methods for this experiment. Per observation, all tested uncertainty-based methods converged at a noticeably faster rate than random sampling, with entropy-based uncertainty converging fastest amongst the tested methods. By the final iteration,

most of the uncertainty-based methods have achieved an F1-score of at least 0.75, with the Least Confidence method being the exception with an F1-score of 0.71.

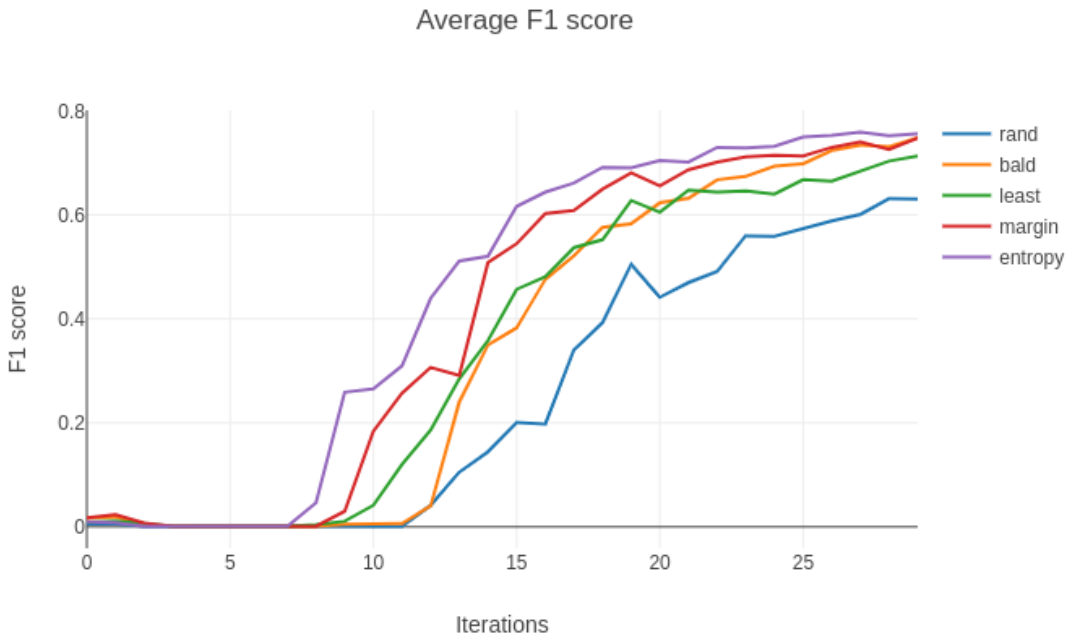

Figure 4.1: Average F1-score across three different seeds versus number of iterations for uncertainty based query methods, using acquisition size 16.

Iteration	Data	Entropy	BALD	Margin	Least	Rand
9	160	**0.26**	0.01	0.03	0.01	0.01
11	192	**0.31**	0.01	0.26	0.12	0.00
13	224	**0.51**	0.24	0.29	0.28	0.10
15	256	**0.62**	0.38	0.54	0.46	0.20
17	288	**0.66**	0.52	0.61	0.54	0.34
19	320	**0.69**	0.58	0.68	0.63	0.50
21	352	**0.70**	0.63	0.69	0.65	0.47
23	384	**0.73**	0.68	0.71	0.65	0.56
25	416	**0.75**	0.70	0.71	0.67	0.57
27	448	**0.76**	0.73	0.74	0.68	0.60
29	480	**0.76**	0.75	0.75	0.71	0.63

Table 4.1: This table captures the regions of interests, mainly from iteration 9-29, from Graph 4.1. Iteration index is zero based and acquisition size is 16. The F1-scores are rounded to two decimals precision points.

4.1.2 Diversity Sampling

Figure 4.2 and Table 4.2 describe the performance of the diversity-based methods and the combined methods for this experiment. Most of the tested methods

performed better than random sampling with the exception of coreset. Combined methods performed in general better than any given purely diversity-based methods, with the worst of the combined methods performing similarly to the best of the purely diversity-based methods. While clustered entropy sampling had the highest convergence rate overall, DBAL was the only method that achieved an F1-score of at least 0.75 by the final iteration. In general, none of the tested methods provided any noticeable advantage over purely uncertainty-based methods in this case.

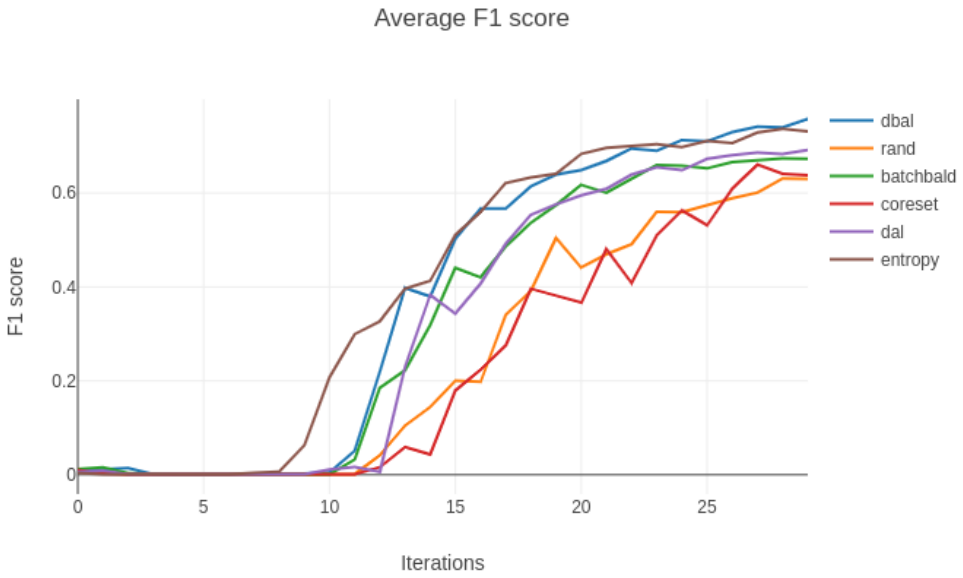

Figure 4.2: Average F1-score across three different seeds versus number of iterations for diversity based query methods and combined methods, using acquisition size 16.

Iteration	Data	C-Entropy	B-BALD	CoreSet	DAL	DBAL	Rand
9	160	**0.06**	0.00	0.00	0.00	0.00	0.00
11	192	**0.30**	0.03	0.00	0.02	0.05	0.00
13	224	**0.40**	0.22	0.06	0.23	**0.40**	0.10
15	256	**0.51**	0.44	0.18	0.34	0.50	0.20
17	288	**0.62**	0.49	0.28	0.49	0.57	0.34
19	320	**0.64**	0.57	0.38	0.58	**0.64**	0.50
21	352	**0.70**	0.60	0.48	0.60	0.67	0.47
23	384	**0.70**	0.65	0.51	0.65	0.69	0.56
25	416	**0.71**	0.65	0.53	0.67	**0.71**	0.57
27	448	0.73	0.67	0.66	0.69	**0.74**	0.60
29	480	0.73	0.67	0.64	0.69	**0.76**	0.63

Table 4.2: This table captures the regions of interests, mainly from iteration 9-29, from Graph 4.2. Iteration index is zero based and acquisition size is 16. The F1-scores are rounded to two decimals precision points.

4.1.3 Time

For an acquisition size of 16, the result of the experiment indicates that the purely diversity-based query methods require significantly less query time than uncertainty-based methods. As seen in Figure 4.3, most methods involving uncertainty required similar query times of about 1200 seconds, while BatchBALD required a query time about double of any other uncertainty-related method. Coreset required mere seconds of query time in total, making it the fastest amongst the methods tested.

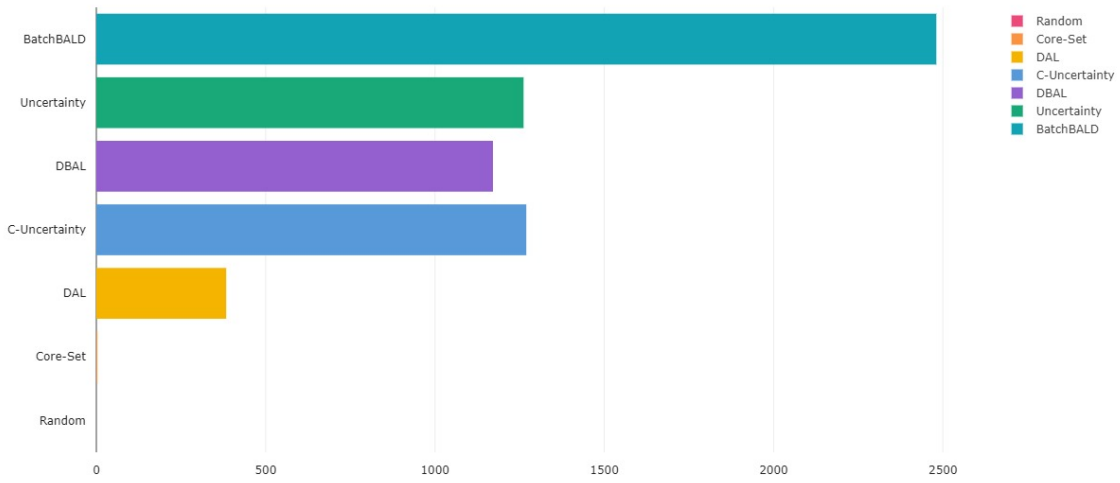

Figure 4.3: Accumulated query time in seconds for tested query methods, using acquisition size 16.

4.2 Acquisition size 50

The following results are obtained from experiments performed in 30 iterations with an acquisition size of 50, which results in all 1500 samples being acquired from the unlabeled pool.

4.2.1 Uncertainty Sampling

Figure 4.4 and Table 4.3 describe the performance of the uncertainty based methods for this experiment. The experiment has shown the tested uncertainty-based methods to have an improved convergence rate than random sampling. Entropy-based sampling is shown to perform noticeably better than other methods, reaching an F1-score of 0.77 already by the eighth iteration. BALD and margin confidence performed similarly, reaching F1-scores of 0.75 by the ninth iteration. The difference between query methods diminished as the iterations increased that by the nineteenth iteration, the maximum difference between each method is no higher than 0.03.

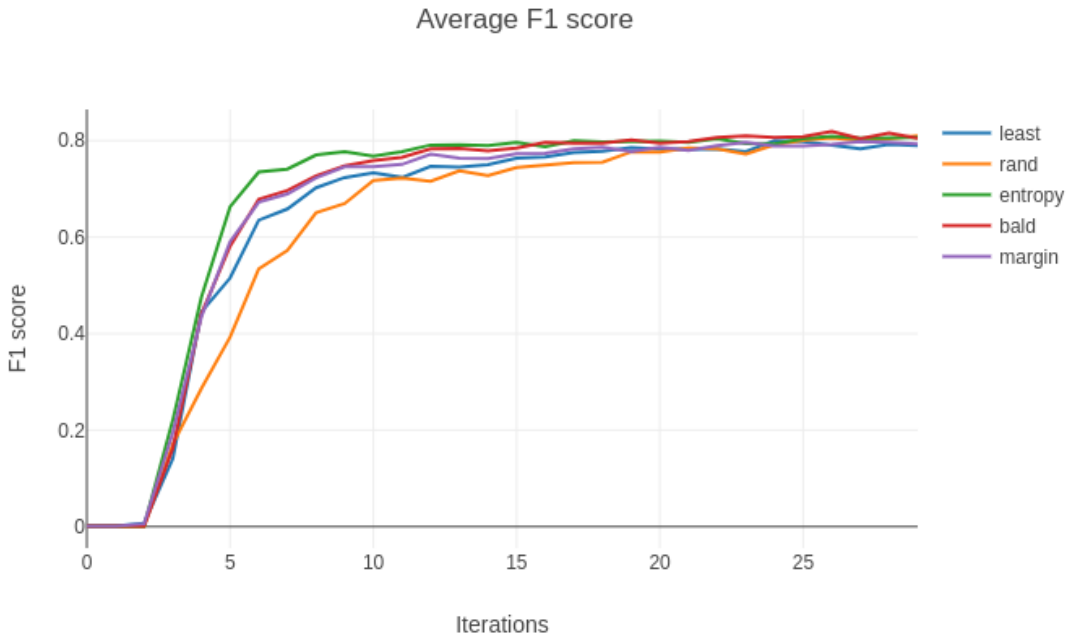

Figure 4.4: Average F1-score across three different seeds versus number of iterations for uncertainty based query methods, using acquisition size 50.

Iteration	Data	Entropy	BALD	Margin	Least	Rand
3	200	**0.22**	0.16	0.19	0.14	0.17
4	250	**0.48**	0.44	0.44	0.44	0.28
5	300	**0.66**	0.58	0.59	0.52	0.39
6	350	**0.74**	0.68	0.67	0.64	0.53
7	400	**0.74**	0.70	0.69	0.66	0.57
8	450	**0.77**	0.73	0.72	0.70	0.65
9	500	**0.78**	0.75	0.75	0.72	0.67
10	550	**0.77**	0.76	0.75	0.73	0.72
11	600	**0.78**	0.77	0.75	0.72	0.72
19	1000	**0.80**	**0.80**	0.78	0.79	0.78
29	1500	**0.81**	**0.81**	0.80	0.79	**0.81**

Table 4.3: This table captures the regions of interests, mainly from iteration 3-11, from Graph 4.4. Iteration index is zero based and acquisition size is 50. The F1-scores are rounded to two decimals precision points.

4.2.2 Diversity Sampling

Figure 4.5 and Table 4.4 describe the performance of the diversity-based methods and the combined methods for this experiment. Amongst the diversity-based methods and combined methods, clustered entropy showed a drastic advantage over other methods by reaching an F1-score of 0.75 at the eighth iteration. Starting from the

fourth iteration, DBAL had a consistent yet less noticeable advantage over random sampling, while BatchBALD on the other hand only had a noticeable advantage over random sampling from the fourth to tenth iteration. Coreset performed nigh-identical to random sampling except in the third and fourth iteration where it had a disadvantage. DAL was shown to be outclassed by every other tested method, performing consistently worse than random sampling. By the nineteenth iteration, the maximum difference between F1-score between each tested query method has diminished to no more than 0.04.

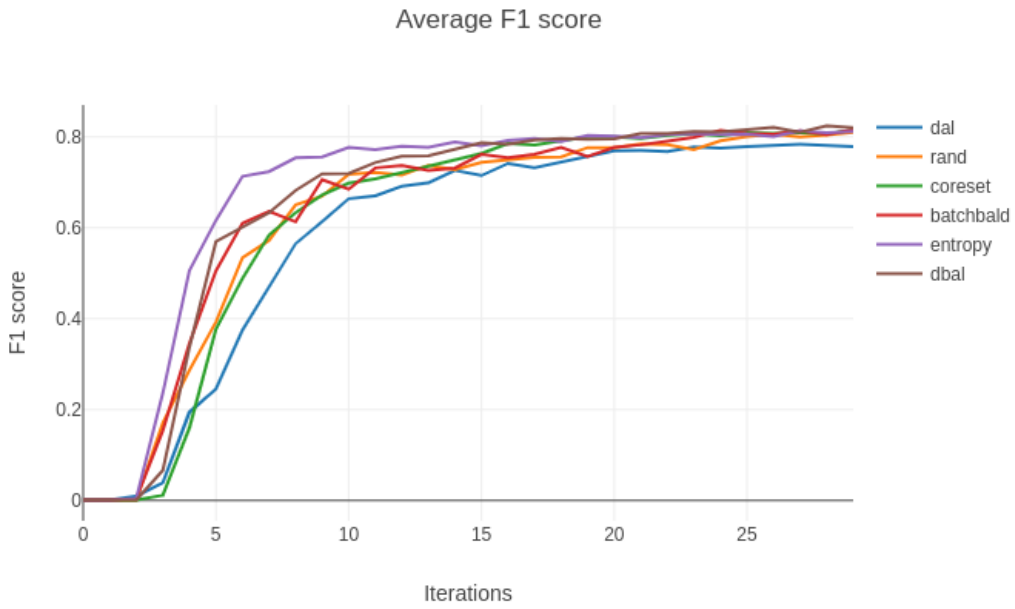

Figure 4.5: Average F1-score across three different seeds versus number of iterations for diversity based query methods and combined methods, using acquisition size 50.

Iteration	Data	C-Entropy	B-BALD	Coreset	DAL	DBAL	Rand
3	200	**0.24**	0.15	0.01	0.04	0.07	0.17
4	250	**0.51**	0.35	0.16	0.19	0.34	0.29
5	300	**0.62**	0.51	0.38	0.25	0.57	0.39
6	350	**0.71**	0.61	0.49	0.37	0.60	0.53
7	400	**0.72**	0.64	0.58	0.47	0.63	0.57
8	450	**0.75**	0.61	0.63	0.57	0.68	0.65
9	500	**0.76**	0.71	0.67	0.61	0.72	0.67
10	550	**0.78**	0.68	0.70	0.66	0.72	0.72
19	1000	**0.80**	0.76	**0.80**	0.76	**0.80**	0.78
29	1500	0.81	0.82	0.82	0.78	**0.82**	0.81

Table 4.4: This table captures the regions of interests, mainly from iteration 3-10, from Graph 4.5. Iteration index is zero based and acquisition size is 50. The F1-scores are rounded to two decimals precision points.

4.2.3 Time

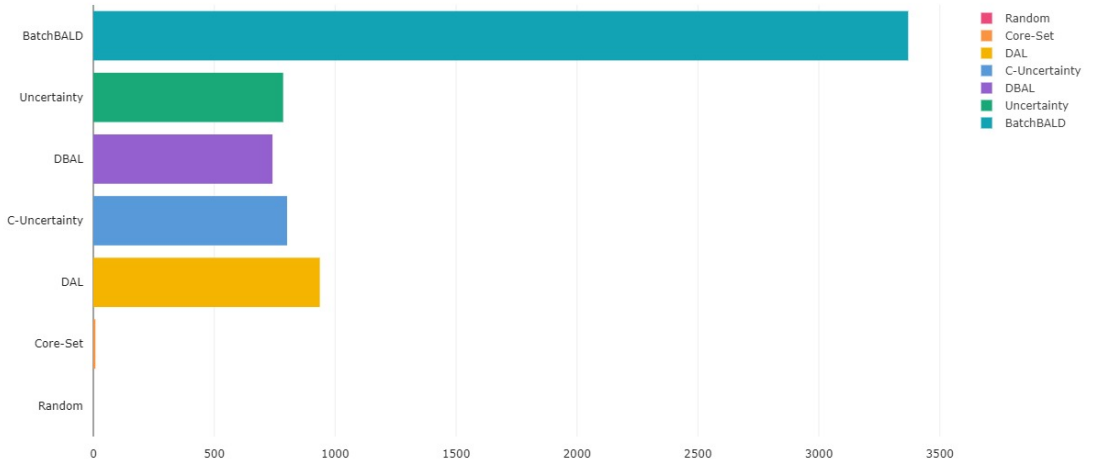

Figure 4.6: Accumulated query time in seconds for tested query methods, using acquisition size 50.

For an acquisition size of 50, all uncertainty-related methods except BatchBALD required a similar total query time of circa 800 seconds. BatchBALD required a query time about four times that of the other uncertainty-based methods, making it the slowest amongst the methods tested. DAL required a query time of 900 seconds, exceeding that of the average uncertainty-related method. Coreset once again required the least amount of query time, which is mere seconds.

4.3 Comparison based upon acquisition size

A generic overview of the effects of acquisition size on performance is presented in the following table, which displays the F1-score with certain specific amounts of samples queried on respective acquisition size for each of the tested query methods.

Iteration Data	11 | 3 192 |200	15 |4 256 | 250	18|5 304 | 300	21 | 6 352 | 350	24 | 7 400|400	27 | 8 448 | 450
Entropy	0.31 | 0.22	0.62 | 0.48	0.69 | 0.66	0.70 | 0.74	0.73 | 0.74	0.76 | 0.77
Least	0.12 | 0.14	0.46 | 0.44	0.55 | 0.52	0.65 | 0.64	0.64 | 0.66	0.68 | 0.70
Margin	0.26 | 0.19	0.54 | 0.43	0.65 | 0.59	0.69 | 0.67	0.71 | 0.69	0.74 | 0.72
BALD	0.01 | 0.16	0.38 | 0.44	0.58 | 0.58	0.63 | 0.68	0.69 | 0.70	0.73 | 0.73
Random	0.00 | 0.17	0.20 | 0.27	0.39 | 0.39	0.47 | 0.53	0.56 | 0.57	0.60 | 0.65

Table 4.5: This table provides for each uncertainty based method a comparison of the F1-scores between acquisitions sizes at certain points where the amount of sampled data are similar. The left hand side value of the bar in a cell is from acquisition size 16 and right hand side is acquisition size 50.

For uncertainty-based methods, using a smaller acquisition size provided a noticeable advantage to entropy sampling and margin sampling until circa 250 samples are queried, while BALD gains an advantage by having a higher acquisition size until the same point. The acquisition size had no significant impact on the F1-score gain per sample otherwise.

Iteration	11 | 3	15 | 4	18 | 5	21 | 6	24 | 7	27 | 8
Data	192 | 200	256 | 250	304 | 300	352 | 350	400 | 400	448 | 450
C-Entropy	0.30 | 0.24	0.51 | 0.51	0.63 | 0.62	0.70 | 0.71	0.70 | 0.72	0.73 | 0.75
B-Bald	0.03 | 0.15	0.44 | 0.35	0.54 | 0.51	0.60 | 0.61	0.66 | 0.64	0.67 | 0.61
CoreSet	0.00 | 0.01	0.18 | 0.16	0.40 | 0.38	0.48 | 0.49	0.56 | 0.58	0.66 | 0.63
DAL	0.02 | 0.04	0.34 | 0.19	0.55 | 0.25	0.61 | 0.37	0.65 | 0.47	0.69 | 0.57
DBAL	0.05 | 0.07	0.50 | 0.34	0.61 | 0.57	0.67 | 0.60	0.71 | 0.63	0.74 | 0.68
Rand	0.00 | 0.17	0.20 | 0.27	0.39 | 0.39	0.47 | 0.53	0.56 | 0.57	0.60 | 0.65

Table 4.6: This table provides for each diversity related method a comparison of the F1-scores between acquisitions sizes at certain points where the amount of sampled data are similar. The left hand side value of the bar in a cell is from acquisition size 16 and right hand side is acquisition size 50.

As for diversity-based methods and combine methods, DAL performed drastically better when using a smaller acquisition size across almost any amount of samples queried. BatchBALD and DBAL also performed better with smaller acquisition sizes, but the difference is less pronounced after circa 250 samples has been queried. No other significant effects caused by the difference in acquisition size have been observed otherwise.

4.4 Consistency of performance

Plots showing the mean and extremes of F1-score with different seeds throughout the experiments are shown in Appendix D. Per observation, for any of the tested query methods the maximum difference between F1-score with any specific seed and the mean F1-score does not exceed 0.1. Most of the query methods appear to perform more consistently when using a larger acquisition size with the exception of DAL where the opposite effect has been observed. Despite there being a difference in performance across different seeds for all tested query methods, the performance ranking of the tested query methods does not seem to be noticeably affected as seen from Appendix A and B. No clear pattern concerning inconsistency between different seeds has been noticed otherwise.

5
Discussion

In this section, the results from the previous section will be analyzed and discussed in detail. An attempt shall be made to explain the various causes which lead to those results, to highlight the limitations, and then also to provide suggestions for future research.

5.1 Performances of Query Methods

As seen from the results, the uncertainty-based query methods usually provide some apparent improvement on the rate of convergence than random sampling, which is consistent with previous works involving active learning with NLP. However, combining uncertainty-based methods with clustering did not provide any significant impact on their performances. Also, the purely diversity-based methods, DAL and Coreset, were shown to have performed consistently worse than uncertainty-based methods, and for certain cases, even similar or worse than that of the random sampling method. These results indicate the measures meant to increase diversity for the queried samples were not effective for improving the convergence rate of the F1-score. That is curious since the performance of those methods was at least competitive to that of uncertainty-based methods according to Ein-Dor et al [10].

The most likely cause of this discrepancy is the difference in the text representation. Unlike the uncertainty-based methods which depend purely on the output of the learning model, diversity-based methods depend heavily on the representation of the input. The inadequacy of the method used to acquire the representation would most likely cause such query methods to query samples in a way that does not contribute or even detrimental to the training of the learning model. The theory of inadequate representation being the cause of the lack of performance of the diversity-based methods is sound with the results from Chen et al[9] in consideration, as in said research the choice of the method to acquire representation from the samples was shown to have an observable effect on the performance of diversity based query methods. However, considering the results of Ein-Dor et al [10] which, in the context of binary text classification, showed varying performance of DAL and core-set on numerous data sets, the difference between data sets may also be a factor that affects the performance of diversity based query methods. In such a case, it is possible that the data set used is incompatible with diversity-related methods.

Amongst the uncertainty-based methods, it is unexpected for entropy sampling,

which is computed similarly to BALD, to perform marginally better against other tested uncertainty-based methods. This is especially surprising considering the results from Shen et al which showed similar performance between three uncertainty-based query methods, including least confidence and BALD [6]. The difference between entropy sampling and BALD lies within the second term of BALD which captures the estimated uncertainty of the model with individual model settings. It is possible that this term is actually detrimental to measuring uncertainty, as uncertainty regardless of the cause may be equally important to be resolved for the successful training of the model.

A fact that should be noted is that the implementation of BatchBALD in this study did not precisely follow the description in Kirsch et al [19]. The algorithm described in Kirsch et al required the dropout mask used for MC dropout to be kept consistent when computing the joint entropies for each set of samples, which is not done in this study. This may result in the informativeness scores being obtained from samples with different model parameters, which losses the correlation between sets. The resulting informative scores for the sampled sets will then be effectively no different from random sampling which does not allow the algorithm to reach its full potential [19].

5.2 Impact of Acquisition Size

Analyzing the impact of acquisition sizes based upon a similar annotation budget by observing Table 4.5 and Table 4.6 reveals that while the performance of different query methods may become affected by the choice of acquisition size, for most query methods the effect on the convergence rate is only pronounced for the initial 250 samples queried. By the time the model has achieved a sufficiently meaningful performance, the effect of acquisition size on the convergence rate has already diminished to a negligible level. Thus, in practical cases, the choice of acquisition size up to 50 will not cause any impediment for any query method. DAL is the only exception to this statement, as its performance using a higher acquisition size is consistently worse than using a lower one according to the results.

The hard difference for DAL between acquisition sizes 16 and 50 was likely caused by the dissimilar sub-batch parameter. The number of sub-batches for acquisition size 16 was 4, which results in 4 samples being selected each time the binary classifier retrains. Acquisition size 50 did instead have the sub-batch amount of 10, which resulted in 5 samples for each time the binary classifier retrained. The authors show a notable increase in performance from a difference in sub-batch size, of 1 to 4, but a low increase from 4 to 10 and 25 which shows a diminishing return. The authors also emphasize that a higher number of possible label classes increases the impact of sub-batch amount. Therefore the performance of acquisition 50 suffers in relation to acquisition 16 that has a lower amount of samples per binary classifier retraining.

For both acquisition size of 16 and 50, it can be seen that all purely uncertainty-based active learning algorithms convincingly outperformed the random sampling

baseline, despite comments in numerous previous researches such as Kirsch et al [19] and Sener et al [29] claimed that these algorithms should not be viable in a batched setting. The presented results show on the contrary that both batch size of 16 and 50 did not cause any noticeable liability, and it is still a viable and a moderately large enough acquisition size for a deep learning method to be impacted.

These results, which contradict the theory of batch awareness, are in fact consistent with the result of previous researches such as Shen et al [6] and Ein-Dor et al [10] where active learning methods have been applied in conjunction with Deep learning models for NLP problems. In these studies, uncertainty-based active learning methods were used in a batched setting and consistently performed better than random sampling. Another notable fact is that the studies concerning batch awareness such as Kirsch et al [19] and Sener et al [29] mostly involve image classification. Considering this fact in conjunction with the results of this study and other studies concerning active learning for NLP problems, a plausible theory is then the issue concerning batch awareness is not as significant for NLP as it is for image processing.

Another factor that is impacted by varying acquisition size is the total query time for all iterations, which can be seen in graph 4.3 for acquisition size 16 and graph 4.6 for acquisition size 50. Core-Set and Clustering (div) only negligibly increased the query time, but are otherwise almost zero. This is expected because those algorithms do not interact with any neural networks. DAL is the only amongst purely diversity based methods that saw a noticeable increase in query time with the increase in acquisition size, and the reason being that for acquisition size 50, the discriminator model has to train more frequently. At acquisition size 16, the amount of mini queries are set to 4 which means that the discriminator retrains after 4 data points and with an annotation budget of 480, there is a total of $480/4 = 120$ discriminator training steps. For acquisition size 50, the amount of mini quires are instead set to 10 and discriminator has to retrain after 5 data points. Then with an annotation budget of 1500, the amount of retrain steps are in total $1500/5 = 300$. DAL has a 2.5 times more training steps in acquisition size 50 than in 16 which is reflected by the algorithms computational time.

For the query methods that involve uncertainty, the choice of acquisition size had a significant effect on the query time. The query time of DBAL, uncertainty and combined clustering were all considerably decreased through using the larger query size of 50. The reason behind this is that the uncertainty measurement is obtained in each iteration by running predictions over every sample the in unlabeled data set. Having a higher acquisition size, which depletes more of the unlabeled data set pool per iteration, reduces the total amount of predictions required. This can be proven by calculating the total amount of predictions required through using the formula $i \cdot s_D - \sum_{n=0}^{i-1} n \cdot q$, where s_D is the initial size of the unannotated set, q is the query size, and i is the number of iterations. Less prediction leads to less computation, which is the cause behind the reduced query time. Note also that in this case, the annotation budget differs when the different acquisition sizes were used. If the annotation budget were the same, more querying iterations would be required for a

smaller query size. That will further increase the total query time and also the total training time of the active learning process.

A distinction to make with regards to query time shown in figures 4.3 and 4.6 is that it only reflects the summation of query algorithms computation for all iterations. There is no labeling or training time included. In this case, labels are provided automatically when the query algorithms have selected suitable samples, but it would otherwise also be a constant time factor depending on the oracle annotator. The model training time is also a constant time factor for all algorithms depending on the acquisition size.

5.3 Future Works

Algorithms that are dependant on text representation such as DAL, Core-Set, and DBAL may have more potential than the presented results have demonstrated. The low query time for DAL and Core-Set, and the inherited batch awareness of these algorithms are benefits that can be practically utilized should their performance be comparable to the uncertainty based algorithms. Naively representing sentences with a pre-trained word vector is not an optimal representation, but a properly trained sentence embedding would give a more fair representation. However, no such embedding that supports Swedish sentences does to date exist. Should a such embedding exist, it is possible that the aforementioned query algorithms would be competitive to the uncertainty based ones in performance, as seen in Ein-Dor et al [10].

Some algorithms could also benefit from a more careful hyperparameter and setup selection. For DAL, instead of the discriminator authors used for image classification with a CNN[32], a more suitable discriminator could be chosen. Since NER is a more difficult task than image classification in theory, a more complex discriminator would potentially perform better. Furthermore, DAL would also benefit from a more thorough mini-batch parameter selection, and DBAL's prefilter constant β could also be better selected by testing various values to find and choose the optimal one.

Furthermore, the implementation of coreset in this project is not the optimal version. The authors presented two versions, a greedy and robust version. The robust version is much more complex and requires more time for implementation but as authors have shown, gives better results[29]. Coreset would also benefit from exploring which distance metric that would be most suitable. The implementation of BatchBALD is not optimal either because the authors state that batchBALD, differently from standard Monte Carlo Dropout, requires consistent Monte Carlo Dropouts for the joint entropy's to be correctly computed[19]. Such algorithm-specific implementations were not done in this project because of the time limitations. Of the same reason, no proper computational optimization was done for the implementations of the algorithms in this study. Allowing the algorithms to properly utilize the computational resources may potentially reduce the time consumption drastically.

Finally, extending the empirical study of active learning to other NLP tasks such as classification, question-answering, etc would be beneficial in understanding how well the active learning algorithms would work across multiple different tasks. Due to the inaccessibility of complete Swedish annotated NLP data sets, this could not be done in this study. The study was also limited to only using KB's Swedish BERT model, but other models such as KB's ALBERT, which during projects duration is in unstable alpha stage would also be of interest to evaluate the active learning models to gain insight on how well the algorithms can extend to different models.

6
Conclusion

The purpose of the thesis was to investigate active learning algorithms on a Swedish NLP task with a transformer-based model, with regards to the performance and computational time of the algorithms. Using KB's pre-trained BERT model, which is the current state-of-the-art pre-trained model that supports Swedish text [59] to date, 20 different active learning algorithms were compared on different acquisition sizes for their performance on a NER task. **The study suggests that the Entropy sampling algorithm is the most optimal active learning algorithm to use based on empirical results which show it to have the highest performance**.

The Entropy sampling algorithm has a stable performance over experiments and also has the fastest converge rate with a large margin over any other compared algorithms, even though it is an exhaustive greedy uncertainty-based algorithm which requires a resource and time demanding prediction to be performed on every unlabeled data point. **The thesis results furthermore suggest that for NER tasks, query methods that incorporate uncertainty-based measurements are more viable than those which do not**. The computationally faster and purely diversity-based algorithms, such as DAL and Coreset, are showing underwhelming and unstable performances in comparison with uncertainty related algorithms. In some instances, they performed similarly or worse than the random sampling baseline, which therefore excludes them as practically useful candidates.

With the exception of DAL, the difference in performance for the tested query algorithms on the different acquisition sizes were presented to be mostly negligible. For the algorithms that incorporated uncertainty measurements, the notion in certain works concerning uncertainty sampling being non-viable for deep learning is also challenged by the thesis, as the results show that uncertainty sampling methods are viable and competitive for the moderate acquisition sizes such as 16 and 50. **The thesis suggests that for NER tasks, a larger acquisition size up to 50 will not cause any noticeable negative effect on most active learning algorithms, and therefore suggests the acquisition size to always be at least 50**. As shown in the computational time results, opting for higher acquisition allows for fewer retraining steps between iterations which allow for faster computation time.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Rachel Grob, Mark Schlesinger, Lacey Barre, Naomi Bardach, Tara Lagu, Dale Shaller, Andrew Parker, Steven Martino, Melissa Finucane, Jennifer Cerully, and Alina Palimaru. What words convey: The potential for patient narratives to inform quality improvement. *The Milbank Quarterly*, 97:176–227, 03 2019.

[4] 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*, 58:11 – 18, 2015.

[5] Olof Mogren Simon Almgren, Sean Pavlov. Named entity recognition in swedish medical journals with deep bidirectional character-based lstms. In *Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining (BioTxtM 2016)*, page 1, 2016.

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

[7] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52, 07 2010.

[8] Mahnoosh Kholghi, Laurianne Sitbon, Guido Zuccon, and Anthony Nguyen. Active learning: a step towards automating medical concept extraction. *Journal of the American Medical Informatics Association*, 23(2):289–296, 08 2015.

[9] 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*, 58:11 – 18, 2015.

[10] 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. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7949–7962, 2020.

[11] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[12] Victoria Bobicev and Marina Sokolova. Inter-annotator agreement in sentiment analysis: Machine learning perspective, 11 2017.
[13] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.
[14] A. Carlson, S. Hong, Kevin S. Killourhy, and Sophie H. Wang. Active learning for information extraction via bootstrapping, 2008.
[15] Liantao Wang, Xuelei Hu, Bo Yuan, and Jianfeng Lu. Active learning via query synthesis and nearest neighbour search. *Neurocomputing*, 147:426–434, 2015. Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012).
[16] Chen Change Loy, Timothy M. Hospedales, Tao Xiang, and S. Gong. Stream-based joint exploration-exploitation active learning. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1560–1567, 2012.
[17] John Daniel Bossér and Erik Sörstadius. Active learning for artificial neural network models. Master's thesis, Chalmers Institue of Technology, 2020.
[18] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A survey of deep active learning, 2020.
[19] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning, 2019.
[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.
[21] D. Wang and Y. Shang. A new active labeling method for deep learning. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 112–119, 2014.
[22] Burr Settles. Active learning literature survey, 07 2010.
[23] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.
[24] Tianxu He, Zhang Kui, Jie Xin, Pengpeng Zhao, Jian Wu, Xuefeng Xian, Chunhua Li, and Zhiming Cui. An active learning approach with uncertainty, representativeness, and diversity. *TheScientificWorldJournal*, 2014:827586, 08 2014.
[25] Robert (Munro) Monarch. *Human-in-the-Loop Machine Learning*. Manning, 2021.
[26] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.
[27] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery.
[28] Fedor Zhdanov. Diverse mini-batch active learning, 2019.
[29] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018.

[30] Pankaj Agarwal, Sariel Har, Peled Kasturi, and R Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52, 11 2004.

[31] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

[32] Daniel Gissin and Shai Shalev-Shwartz. Discriminative active learning, 2019.

[33] Yang Yuan, Soo-Whan Chung, and Hong-Goo Kang. Gradient-based active learning query strategy for end-to-end speech recognition. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2832–2836, 2019.

[34] Jiaji Huang, Rewon Child, Vinay Rao, Hairong Liu, Sanjeev Satheesh, and Adam Coates. Active Learning for Speech Recognition: the Power of Gradients. *arXiv e-prints*, page arXiv:1612.03226, December 2016.

[35] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery.

[36] Elizabeth D Liddy. Natural language processing, 2001.

[37] Mohammad Norouzi, Tomas Mikolov, Samy Bengio, Yoram Singer, Jonathon Shlens, Andrea Frome, Greg S Corrado, and Jeffrey Dean. Zero-shot learning by convex combination of semantic embeddings. *arXiv preprint arXiv:1312.5650*, 2013.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[39] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30, 01 2007.

[40] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[41] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, 01 2004.

[42] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition, 2020.

[43] Nasser Alshammari and Saad Alanazi. The impact of using different annotation schemes on named entity recognition. *Egyptian Informatics Journal*, 2020.

[44] Kahaerjiang Abiderexiti. Annotation schemes for constructing uyghur named entity relation corpus, 11 2016.

[45] Vijay Krishnan and V. Ganapathy. Named entity recognition, 2005.

[46] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[47] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer, 2019.

[48] Sebastian Ruder. Transfer Learning - Machine Learning's Next Frontier. `http://ruder.io/transfer-learning/`, 2017.

[49] Peter Prettenhofer and Benno Stein. Cross-lingual adaptation using structural correspondence learning. *ACM Trans. Intell. Syst. Technol.*, 3(1), October 2011.

[50] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc Le, and Mohammed Norouzi. Google's neural machine translation system: Bridging the gap between human and machine translation, 10 2016.

[51] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[52] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

[53] Lars Borin, Markus Forsberg, Martin Hammarstedt, Dan Rosén, Roland Schäfer, and Anne Schumacher. Sparv: Språkbanken's corpus annotation pipeline infrastructure. In *SLTC 2016. The Sixth Swedish Language Technology Conference, Umeå University, 17-18 November, 2016*, 2016.

[54] Sofia Gustafson-Capková and Britt Hartmann. *Manual of the Stockholm Umeå Corpus version 2.0*. SpråkBanken.

[55] Sherzod Hakimov, Salih Oto, and Erdogan Dogdu. Named entity recognition and disambiguation using linked data and graph-based centrality scoring, 05 2012.

[56] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition, 2016.

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

[58] Francisco José Valverde-Albacete, Jorge Carrillo-de Albornoz, and Carmen Peláez-Moreno. A proposal for new evaluation metrics and result visualization technique for sentiment analysis tasks. In Pamela Forner, Henning Müller, Roberto Paredes, Paolo Rosso, and Benno Stein, editors, *Information Access Evaluation. Multilinguality, Multimodality, and Visualization*, pages 41–52, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[59] Martin Malmsten, Love Börjeson, and Chris Haffenden. Playing with words at the national library of sweden – making a swedish bert, 2020.

[60] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Portuguese named entity recognition using bert-crf. *arXiv preprint arXiv:1909.10649*, 2019.

A

Appendix 1

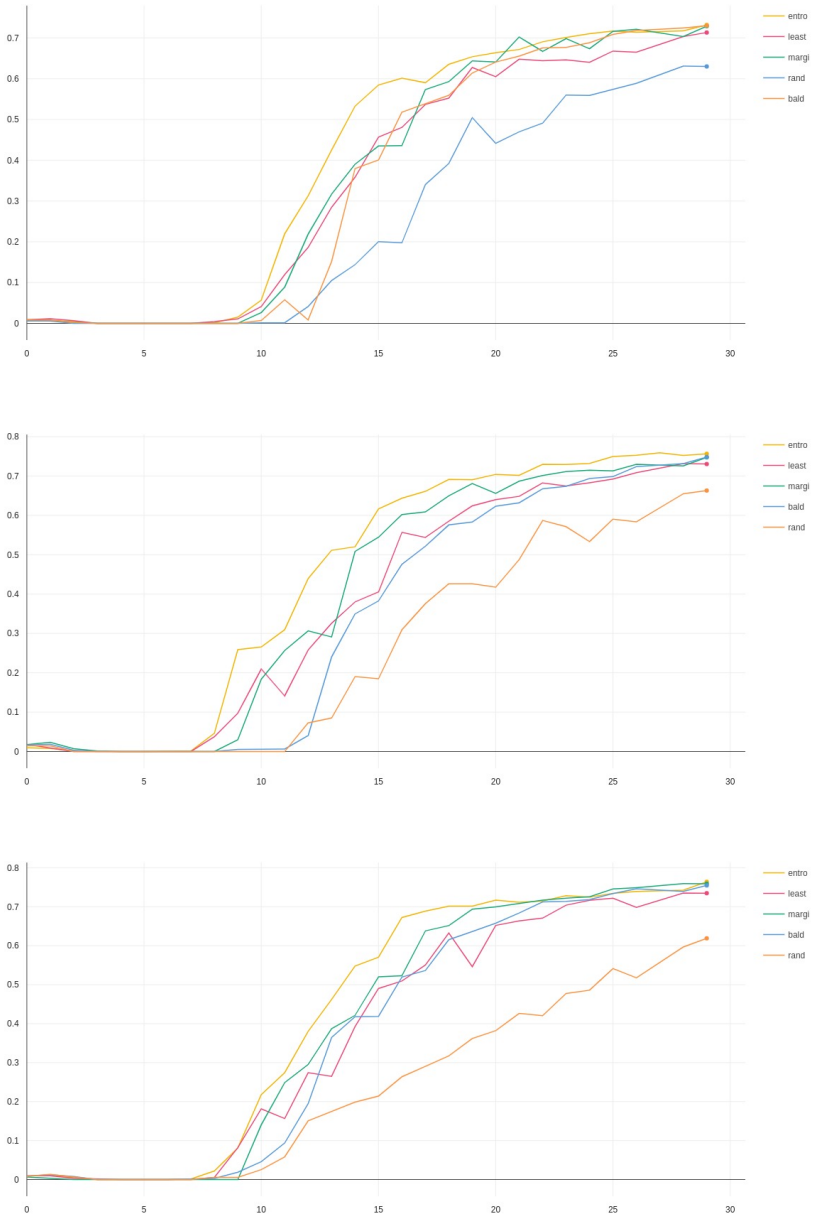

Figure A.1: 16 samples per iteration, uncertainty-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300.

Figure A.2: 16 samples per iteration, diversity-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300.

B

Appendix 2

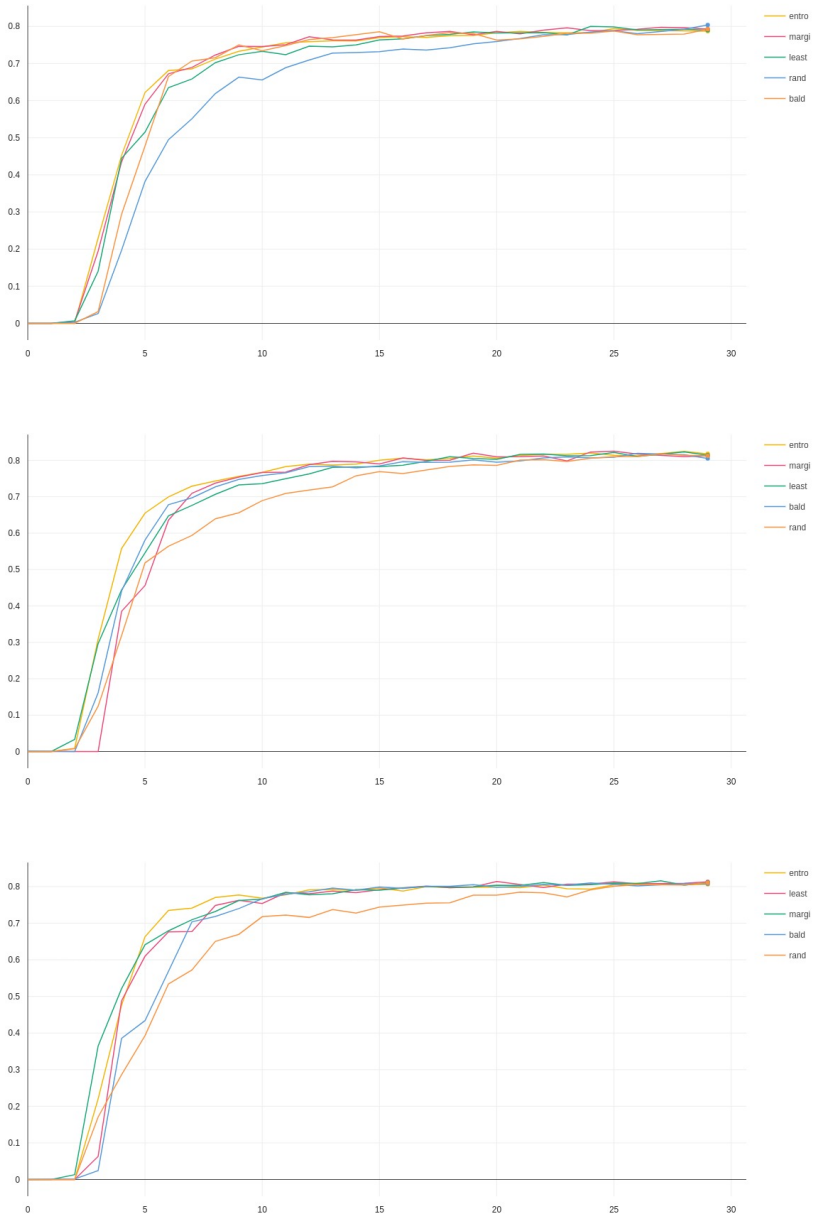

Figure B.1: 50 samples per iteration, uncertainty-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300.

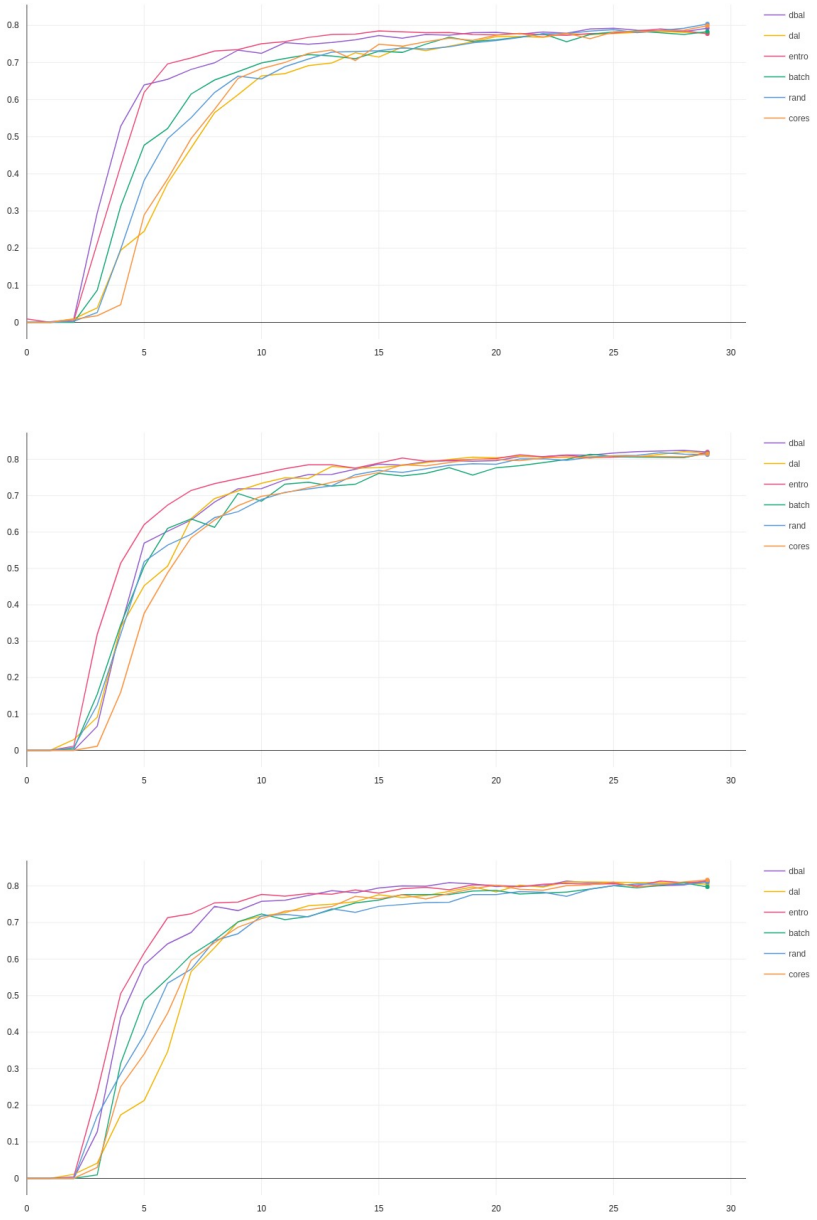

Figure B.2: 50 samples per iteration, diversity-based sampling algorithms, first graph has seed 100, second seed 200 and third seed 300.

C

Appendix 3

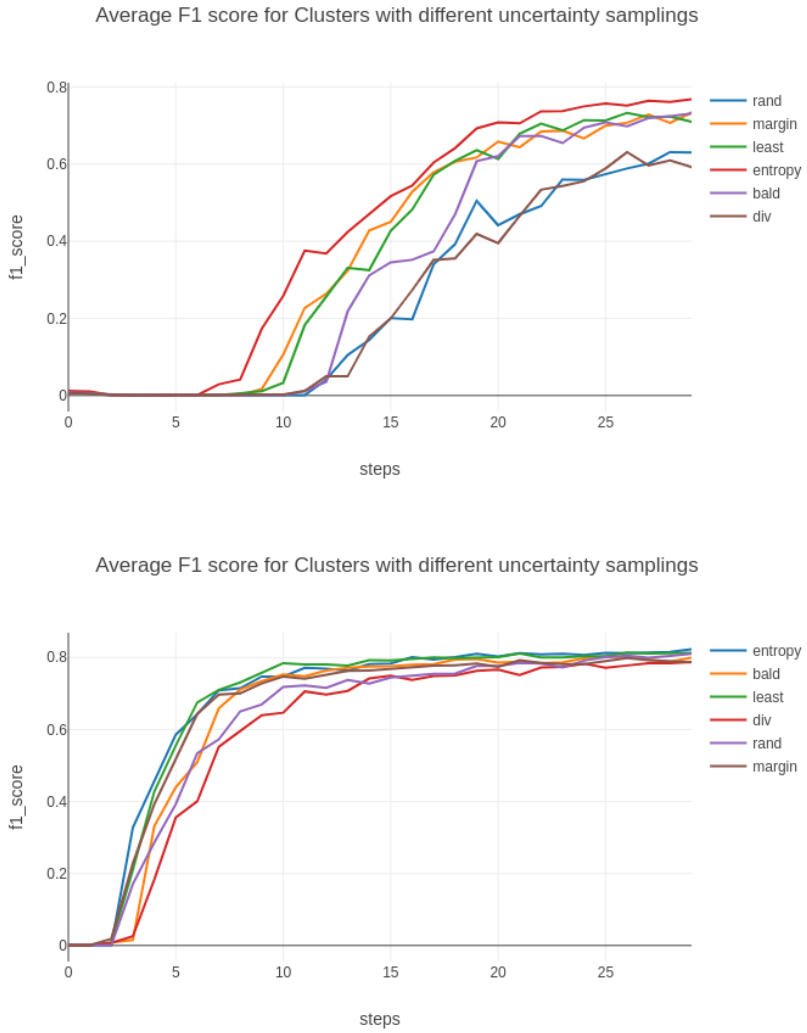

Figure C.1: Comparison of the naive clustering algorithm combined with different uncertainty-based sampling algorithms. The first graph is with acquisition size 16 and second is acquisition size 50.

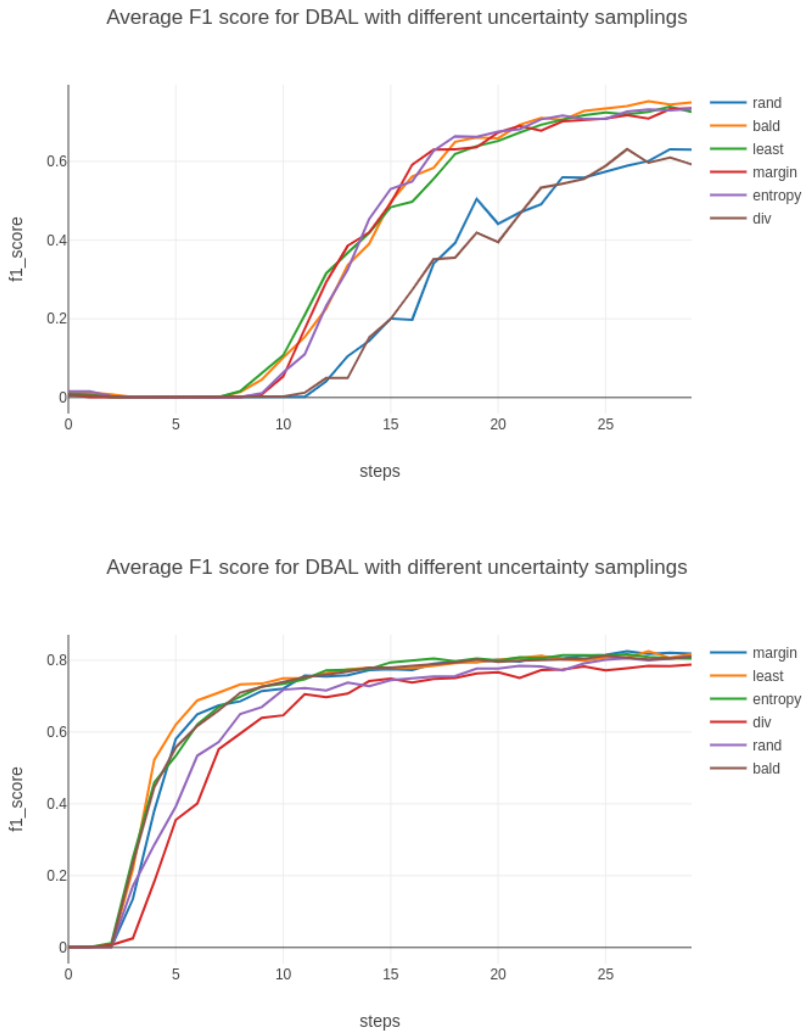

Figure C.2: Comparison of the DBAL algorithm combined different uncertainty-based sampling algorithms. The first graph is with acquisition size 16 and second is acquisition size 50.

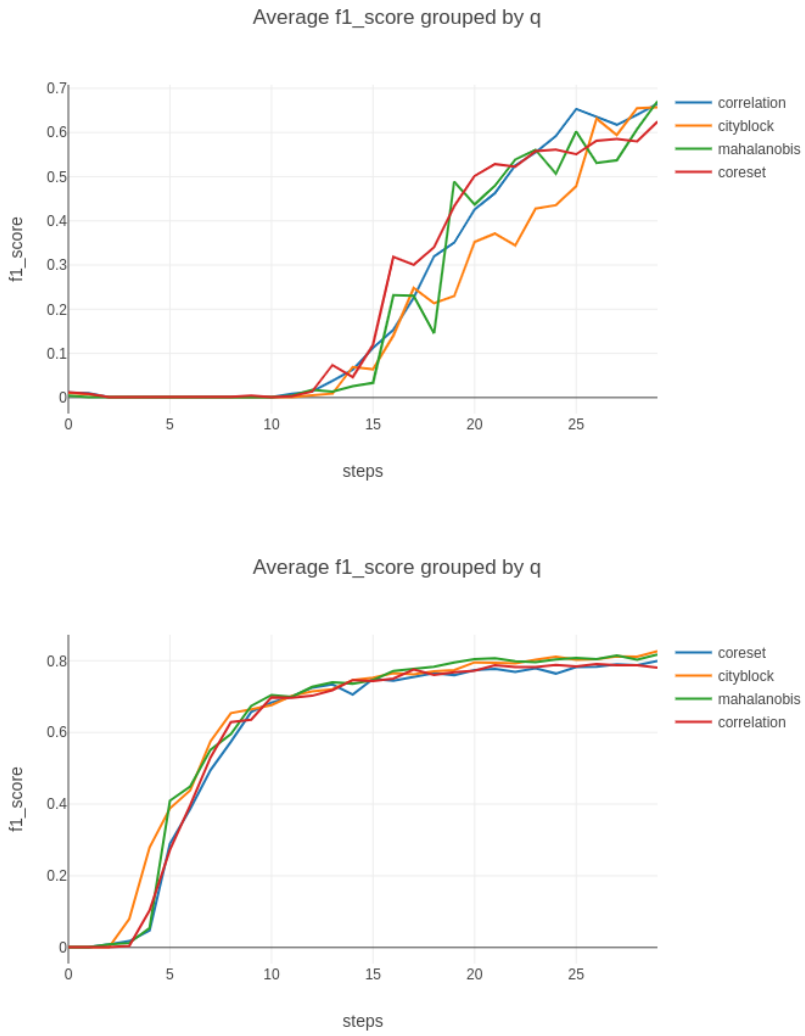

Figure C.3: Comparison of different distance metrics for the coreset algorithm. The first graph is for acquisition size 16 and second is for acquisition size 50. The coreset legend uses the cosine distance metric.

D

Appendix 4

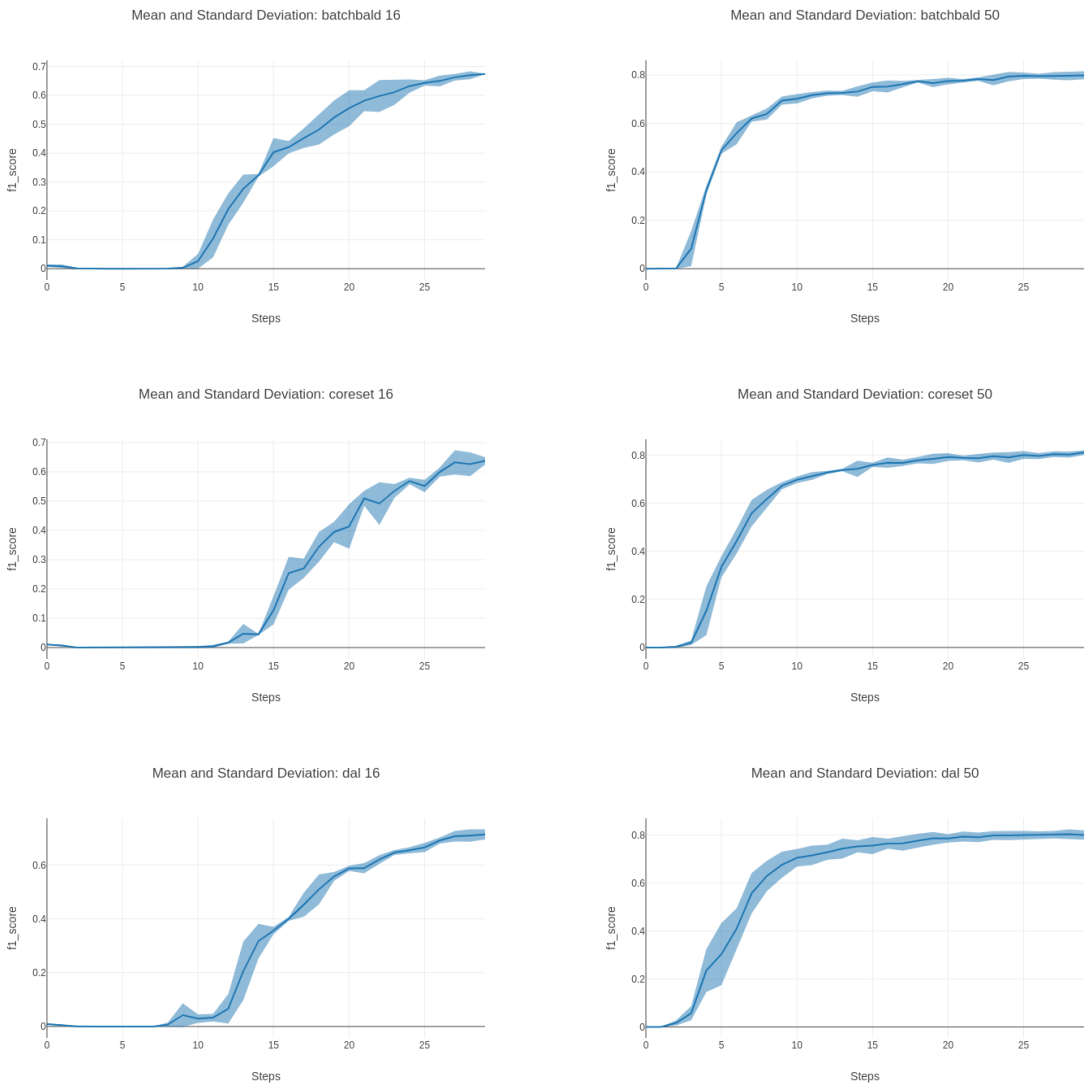

Figure D.1: Mean and deviations of BatchBALD, Core-Set and DAL.

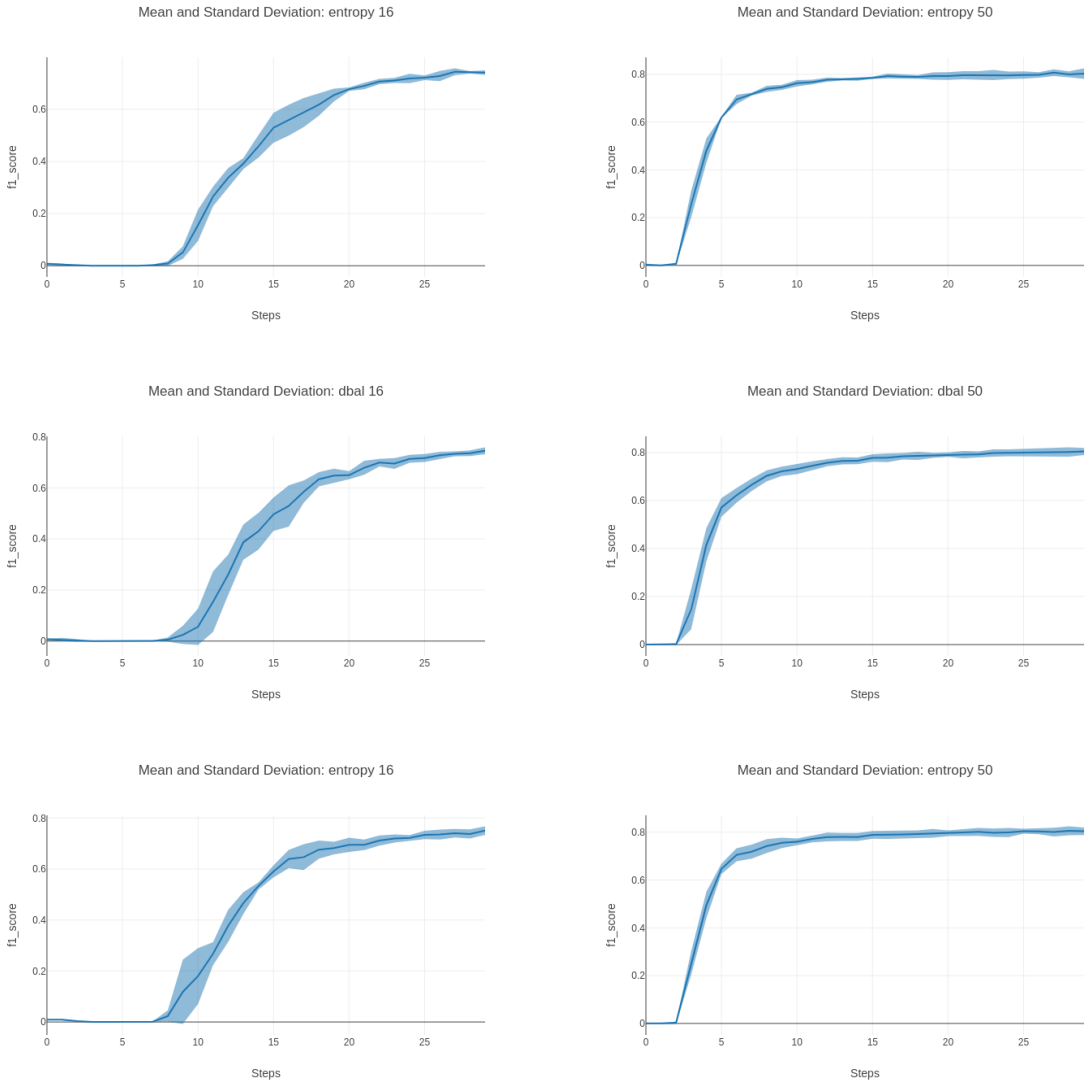

Figure D.2: Mean and deviations of C-Entropy, DBAL-Entropy and Entropy.

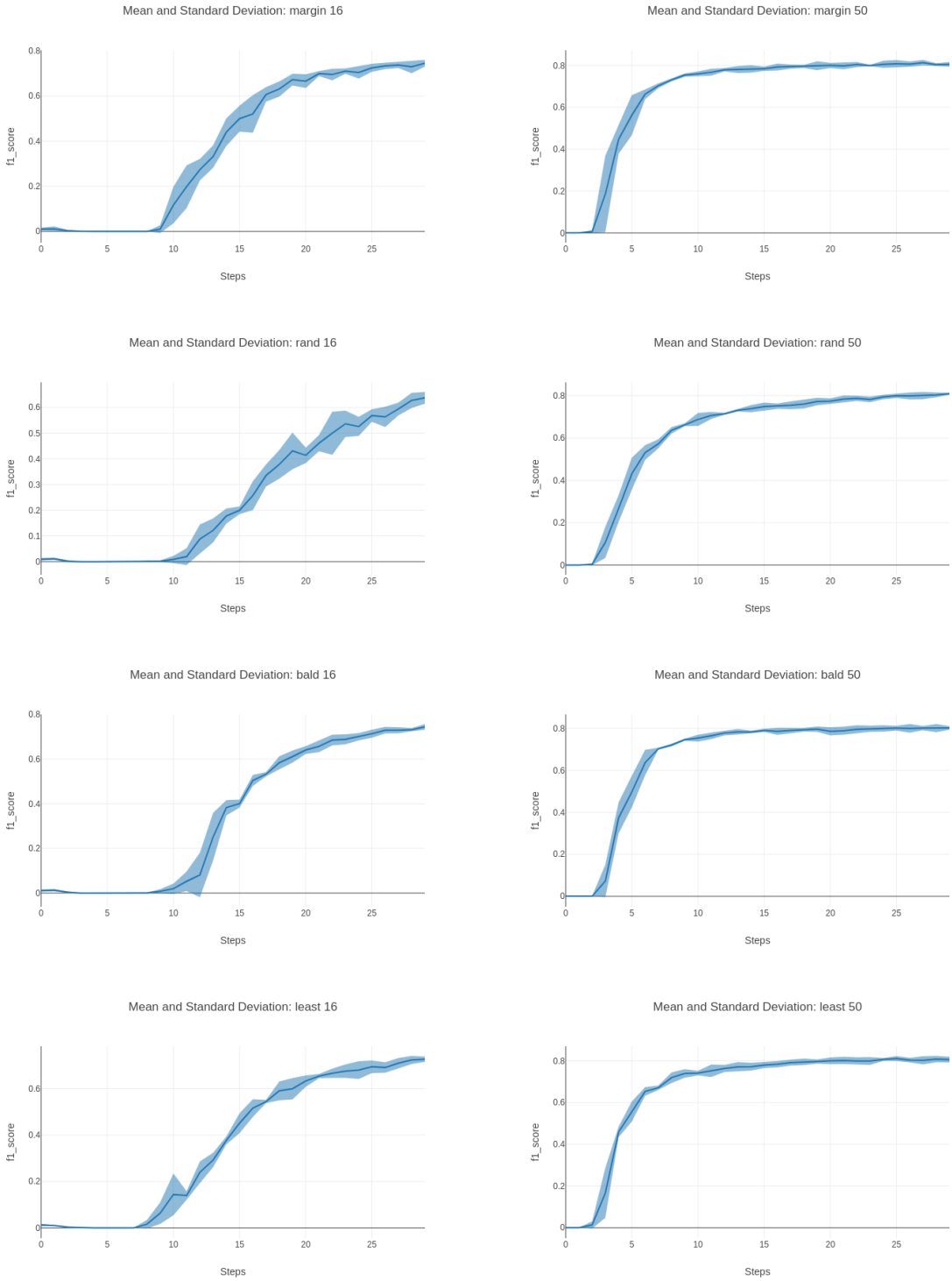

Figure D.3: Mean and deviations Marginal, bald, least and random sampling.

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY